UNIVERSITY OF CALIFORNIA
SANTA CRUZ

ARTIFICIAL INTELLIGENCE AS A TOOL FOR UNDERSTANDING NARRATIVE CHOICES
A CHOICE-POINT GENERATOR AND A THEORY OF CHOICE POETICS

A dissertation submitted in partial satisfaction
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Peter A. Mawhorter

March 2016

The dissertation of Peter A. Mawhorter is approved:

_____
Professor Michael Mateas, chair

_____
Professor Noah Wardrip-Fruin

_____
Professor R. Michael Young

_____
Tyrus Miller
Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

Abstract

Artificial Intelligence as a Tool for Understanding Narrative Choices

Peter A. Mawhorter

This document describes a research approach that uses artificial intelligence (by way of a generative system) as a tool to explore the poetics of narrative choices (e.g., those found in Choose-Your-Own-Adventure books). Building on lessons learned from a rational reconstruction of Scott Turner's *Minstrel*, *Dunyazad* is a novel system which generates narrative choices. Developed in tandem is a framework for analyzing choices based on player goals, as part of a broader theory of *choice poetics*. Two experiments involving human subjects have confirmed that *Dunyazad* is able to successfully generate a variety of choices, and both its successes and failures have informed the theory that drives it.

The research questions addressed involve both computer science and poetics, namely: "How can a computer automatically generate choices that achieve specific poetic effects?" and "What poetic effects can choices accomplish within a narrative and how can these be recognized by examining said choices?" Regarding the second question, chapter 5 outlines a broad theory of choice poetics which emphasizes the importance of player motivations. This theory asserts that in interactive narratives, choices are an essential part of poetic effects like transportation, agency, autonomy, responsibility, and regret. A method for analyzing choices relative to player goals is also developed, which includes discrete steps for understanding choices. By separating notions like player goals, outcome likelihoods, and option expectations, this goal-based choice analysis method aims to help dissect a player's perception of a choice and give authors and critics a tool for talking about how choices work within a narrative.

Regarding the first research question above, *Dunyazad* is a technical demonstration that a direct operationalization of goal-based choice analysis can effectively generate narrative choices with specific low-level poetic effects, such as obviousness. *Dunyazad* uses answer-set programming to encode the inferences used in goal-based choice analysis as logical rules, and it can then solve for choices which meet criteria specified in terms of analytical labels like "this choice is a dilemma." Because answer-set programming offers a direct path from theory

to code, the choices that *Dunyazad* constructs can be understood in terms of the rules of goal-based choice analysis which enabled them.

This last aspect is what allows *Dunyazad* to productively contribute back to the theory of choice poetics: when it generates a choice which doesn't live up to the desired label, the answer set behind that choice serves as a proof that goal-based choice analysis would mis-characterize similar choices. Any unaccounted for aspects of that choice can then be noted as important considerations in the theory, and corresponding rules can be added to *Dunyazad* to alter its generative space. This process is amplified when the choices *Dunyazad* generates are evaluated not just by its author, but in an experimental setting. Accordingly, this document presents the results of two experiments involving 90 and 270 participants which looked at how choices generated by *Dunyazad* were perceived in terms of their options and outcomes. The results of these experiments confirmed *Dunyazad*'s ability to generate relaxed, obvious, and dilemma choices, as well as choices with both expected and surprising outcomes. However, *Dunyazad* was not exclusively successful, and its failures were informative, emphasizing the importance of moral reasoning in the case of conflicting goals and of relative goal analysis when judging whether options seem balanced or not. These results point to directions for improving the system, and these improvements correspond with further developments of the theory of choice poetics.

In the broader context of research into generative systems, *Dunyazad* represents both a system with a novel domain and a new application for a generative system. In following the rules laid out for it, *Dunyazad* makes no assumptions whatsoever, in contrast to the 'common sense' and myriad biases that a human would employ given the same instructions. Because of this, it acts as a check against these assumptions, regularly coming up with examples which violate them, and which thereby help elaborate the definitions of the underlying concepts that it is using. If *Dunyazad*'s goal was to persuade or entertain, these edge cases would be failures, but employed as a tool for reflection, they are successes, because they lead to a deeper understanding of choice poetics. Hopefully, *Dunyazad* is a convincing argument in favor of a new form of critical technical practice, where technical progress can be seen as not merely in need of critical contextualization but also as a tool that enables deeper critical insight.

To Zhi Li (李智) for her tremendous love and support during the preparation of this document, and to my parents Richard and Jennifer Mawhorter, my mother-in-law Ying Su (苏颖), and my siblings Sarah, John, and Ross for all of the ways that they have influenced me to become the person I am today.

ACKNOWLEDGEMENTS

I would like to first acknowledge the outstanding contributions of Michael Mateas to my personal growth as a scholar. His mentorship has contributed incalculably to my entire graduate career, and without his patient guidance and encouragement, this document would not exist in any form. I also want to thank the other members of my advancement and dissertation committees, Noah Wardrip-Fruin, Arnav Jhala, and R. Michael Young, who through their feedback have developed the ideas expressed here into their current form.

Additionally, I want to thank fellow students who have proven indispensible through their feedback, frank discussion, and moral and technical support. Brandon Tearse, Ben Weber, Adam Smith, Chris Lewis, Gillian Smith, Josh McCoy, Serdar Sali, Anne Sullivan, Mike Treanor, Sherol Chen, Teale Fristoe, Ken Hullett, and Ron Liu all served as my role models during graduate school and have each contributed in some way to this document. Joe Osborn, Adam Summerville, Kate Compton, John Murray, Bryan Blackford, John Grey, April Grow, Eric Kaltman, Zhongpeng Lin, Johnathan Pagnutti, Ben Spalding, Deirdra Kiai, Heather Logas, Paulo Gomes, and Sarah Harmon have been contemporaries who I have at one time or another depended upon for support, feedback, and intellectual growth. I would also like to thank Aaron Reed, Jacob Garbe, Ben Samuel, James Ryan, and Stacey Mason in particular for their friendship and support during my graduate career; their viewpoints have helped shape my own and they have in many ways made this work possible.

Finally, I want to thank Michael Mateas and Noah Wardrip-Fruin one more time for their dedication to the creation of an unmatched research environment in the form of the Expressive Intelligence Studio at UCSC. They have both worked ridiculously hard to foster a unique collaborative environment, and their efforts have given me the opportunity to be influenced by so many during my graduate career. They are both outstanding advisors and I consider myself lucky to have been their student.

Chapter 1

Introduction

When I began the work described here, I was a computer scientist by training, and having recently discovered the existing work on computer-generated narrative, I had two initial impressions. First, I was intrigued at the idea of applying artificial intelligence techniques to such an artistic domain (I had previously done research in AI for robots and game-playing). Second, I was firmly convinced that there was great potential for improvement in this domain. In particular, it seemed to me then (as it apparently has to a variety of computer scientists since at least the 1980s) that by capturing the underlying rules of what makes a story interesting, a computer ought to be able to produce interesting, albeit formulaic, stories. Certain pieces of writing advice and narrative theory reinforced this second impression, and I saw one of the main goals of my dissertation as the creation of a story generation system that would outperform existing work.

As it turned out, a new and improved story generator is not one of the main contributions of my work (I have built new narrative generation technology, but it isn't a game-changer in terms of the quality of automatically-generated stories). Working first to reconstruct Scott Turner's *Minstrel* (Turner, 1993) and then on my own *Dunyazad*, I have begun to understand a little about what makes

automatic narrative generation such a difficult problem (although some of my lessons are perhaps those that every student in the area will learn through experience). In the end, rather than simply a generative system, the main contribution of this document is actually an example of a hybrid research method which advances both artificial intelligence and narrative theory at once by combining the two. Said research method is applied to choice-based narratives,[1] and in fact I have built the first narrative generation system that creates choices via explicit reasoning about how players perceive options and outcomes.

This research method is an extension of Phil Agre's idea of critical technical practice (Agre, 1997): engaging in technical AI research while addressing and respecting the critical implications of that work. Rather than proposing that *Dunyazad* as a generative system helps us understand human perceptions of choices by accurately modelling how humans think (as someone like Herbert Simon might advocate (Herbert A Simon, 2006)), I am using *Dunyazad* as a tool to generate inhuman perceptions of choices, and by reflecting on how these differ from human perceptions, finding a new perspective that illuminates the assumptions and biases of the human perception of choices. In Agre's words, I am "[approaching] technical work in the spirit of reductio ad absurdum: faced with a technical difficulty, [I am] diagnosing it as deeply as possible," (Agre, 1997) and in fact I am intentionally pursuing technical difficulties precisely because interrogating them can help develop the details of an underlying theory.

In my case, I am using a formal model of narrative choices to test and explore what I call "choice poetics:" a narrative theory of audience response to choice

[1]The output of *Dunyazad* and the focus of my narrative theory roughly corresponds to a Choose-Your-Own-Adventure book. These books are young-adult fiction which contain choices at the end of some pages, where directions state that the reader should turn to one of several pages next according to what they wish to 'do.' They thus give the reader limited control over the protagonist through explicit, discrete choices (usually on the order of dozens to perhaps a hundred per book). This format thus combines narrative with explicit choices to function as what I term a "choice-based narrative."

structures. Both the process of constructing such a formal model and the experiments that can be performed with it can help develop the theory of choice poetics. At the same time, the AI system relies on the theory, and the result of the research is thus both an improved theory of choice poetics and a novel AI architecture for generating narrative choices.

1.1 Contributions

As implied earlier, one of the contributions of my work is some general knowledge about the strengths and limitations of certain approaches to story generation. Working with Brandon Tearse on the *Skald* system, which is a rational reconstruction of Turner's *Minstrel*, I learned not only the limitations of that system, but also some broad limitations of case-based story generation. In particular, the ability of any case-based story generation system to recombine content is effectively limited by the depth at which it can reason about its story domain.

Based on this limitation, I built *Dunyazad*, a choice-based-narrative generator which focuses on deep reasoning about a narrow story domain using answer-set programming. *Dunyazad* is also designed to generate narrative choices, rather than linear narratives, as this problem has not received sufficient attention in the literature until now. While developing *Dunyazad*, I also began to develop a theory of choice poetics—a theory of how audiences respond to different choice structures. The main contribution of my work is thus both a theory (choice poetics) and a system (*Dunyazad*) which both explore new territory around choices in games.

Finally, as *Dunyazad* has reached a stage where it can generate individual choices targeting some specific poetic effects, I have run two experiments to see how people actually react to the choices it generates. The results of these experiments have helped me improve the system, but they also have some implications for the underlying theory. The lessons of these experiments, both for AI systems and for choice poetics theory, are another contribution of this dissertation.

1.2 *Minstrel* and *Skald*

At the time that I first developed an interest in story generation, Brandon Tearse was working in my lab on a rational reconstruction of Scott Turner's 1993 *Minstrel* system (Turner, 1993). The project was motivated by the impressive quality of the example stories given in Turner's thesis (Turner, 1993): Tearse thought that *Minstrel* would be a good foundation for developing an even-more-sophisticated story generator. *Minstrel* used case-based reasoning to assemble new stories by remixing pieces from a story library according to special transform-recall-adapt methods ("TRAMs"). In particular, Tearse was interested in finding out whether *Minstrel* could be used as a general-purpose story generation system, with different kinds of stories being produced simply by switching out the story library (and perhaps altering some of the TRAMs). I joined the project, and over the next few years we built *Skald* by carefully reading Turner's thesis and reconstructing his system.

As a rational reconstruction project, one of our goals was to better understand how Turner's architecture worked not by re-coding every line of the original source, but by building a new system using the principles of the original. This technique can expose new information about the reconstructed system, for example when a relationship which is invariant given the original architecture requires explicit reasoning in the reconstruction. Building *Skald*, we found ourselves unable to reproduce the quality of *Minstrel*'s example stories with any regularity. After performing some experiments, we came to the conclusion (supported, in retrospect, by the language of Turner's dissertation) that *Minstrel* was more a proof-of-concept for case-based story generation than a robust and extensible system (see section 4.7).

Minstrel's example stories were high-quality because *Minstrel* was carefully tuned to produce examples of what was possible for case-based story generation.

However, adding new stories to *Minstrel*'s case library or trying to get it to generate different kinds of stories turned out to be a laborious and error-prone process. A large part of this process was writing code to check for and correct mistakes introduced when story fragments were combined in inappropriate ways. We hoped that *Skald* would be a useful architecture for creating custom story generators, and we started building a system called *Problem Planets* that would use *Skald* to generate science fiction stories (which were also going to include choices). Unfortunately, the difficulties we encountered led us to conclude that *Skald* was not suitable for our purposes. Chapter 4 contains the details of our rational reconstruction as well as the lessons we learned from building *Skald* and from working on *Problem Planets*.

Ultimately, these experiences led me to focus on the kinds of consistency constraints that we introduced to clean up *Skald*'s assembled stories. I reasoned that if much of the work necessary to produce a consistent story was being done by these constraints, then a generator that focused on them might be more effective. My work on *Skald* was thus the impetus for building *Dunyazad*.

1.3 *Dunyazad*

Dunyazad is a choice-point generator that relies on answer-set programming to build tailored choices. It generates individual choices aimed at provoking specific reactions from the audience—that is, aimed at achieving specific (choice) poetic effects. Each choice generated by *Dunyazad* takes place in an individual scene during which a few actions takes place; these actions can be compared to planning operators in that they have pre- and post-conditions. *Dunyazad* is able to assemble individual choices into a tree with actions as edges between states, but it does not currently reason about things like character development or plot arcs that would be necessary to construct interesting stories consisting of multiple scenes.

Dunyazad builds its action trees by taking unfinished states and generating a collection of options to create a choice, setting up new scenes as necessary. While this high-level process is a simple iterative one, the construction of each choice (and the setup for each new scene) is accomplished using answer-set programming. Effectively, *Dunyazad* searches the space of all possible choice configurations for those that are allowed by its constraints and picks one arbitrarily.

This means that for *Dunyazad* to work well, it needs to have a set of constraints that captures information about story consistency and choice structures. These constraints are effectively a theory of choice poetics, and because I am using answer-set programming, this theory is encoded as a set of first-order logical predicates and inference rules (as opposed to say, being implicit in the weights of a neural network). In building *Dunyazad*, I thus *necessarily* created an implicit theory of choice poetics. By making this implicit theory explicit and using craft advice and exemplary choice-based narratives to inform it, *Dunyazad*'s generation was improved. At the same time, the work on *Dunyazad* informed the theory: the answer set solver inevitably found problems with my initial attempts at defining various choice structures, resulting in malformed choices.

Dunyazad is thus a useful tool for developing the theory of choice poetics. Alongside traditional techniques such as analysis of existing choice-based narratives and the critical dialogues surrounding them, the operationalization of choice poetics as a component of a generative system has unique insights to offer. Furthermore, by performing experiments with the system, the lessons from building and debugging the system can be supplemented with empirical results. Chapter 5 describes my theory of choice poetics and in particular a technique for analyzing choices with respect to player goals that *Dunyazad* has helped shape. While chapter 6 describes how *Dunyazad* works in detail, chapters 7 and 8 present the results of two empirical investigations.

Two key results from these experiments are the importance of using relative rather than absolute value judgements when comparing options, and the idea that the perception of an outcome colors later assessments of the choice that led to it. These results are consistent with existing research on real-world decision making (see e.g., (Mellers, A. Schwartz, and Ritov, 1999; Shepperd and McNulty, 2002)), but this work verifies them to some degree in a very different context. Another important finding was that a simple high/low priority model of player goals was insufficient to model player expectations even when faced with relatively simple choices. Based on these findings, both future work for *Dunyazad* as a generative system and revisions to the theory of choice poetics are discussed in chapters 7 and 8.

1.4 Outline

Before diving into the main body of work, chapter 2 lays out the research methodology that has guided my work on *Dunyazad*, and chapter 3 introduces related work to put what follows into context. Chapter 4 describes how *Skald* works and its relationship to *Minstrel*, and section 4.7 goes into detail about the problems we encountered building *Problem Planets* and the inspiration for *Dunyazad*.

Chapter 5 describes the foundations of a theory of choice poetics, and chapter 6 goes on to describe how *Dunyazad* works and which parts of the theory it operationalizes. Finally, chapters 7 and 8 present and analyze the results of two experiments conducted using *Dunyazad*, and chapter 9 summarizes the contributions of the entire document.

Chapter 2

Methodology

My work on *Dunyazad* has used an approach that's a little bit different from many other computer science research projects. The problem I set out to solve, namely "How can a computer automatically generate choices that achieve specific poetic effects?" cannot be approached without first having an understanding of what poetic effects choices can create and how they do so. However, when I began my research, there was no existing literature focused on the specific poetics of choices (several authors have extended narratology to deal with various forms of interactive stories, but those projects are quite broad). Thus in order to build my system, I would need to create my own theory describing how elements of a choice come together to produce effects like regret or hesitation, which led to a second research question: "What poetic effects can choices accomplish within a narrative and how can those be recognized by examining said choices?" Rather than simply hack away at my system until I got results that I liked and leave a theory implicit in the code that I wrote, I decided to come up with an explicit theory of choice poetics that I would develop alongside my system, thereby giving both research questions equal ground. In the end, the system has become as much a tool for the development of the theory as the theory is a tool for the

development of the system, and my research methodology has become a hybrid of AI systems development and narrative theoretics. As evidenced by the rest of this document, and in particular chapters 7 and 8, this hybrid research methodology has a lot to recommend it, and the *Dunyazad* project as a concrete example of such a methodology in action is one of the key contributions of my work.

2.1 Two Approaches

Faced with the problem of coming up with a theory about how choices produce poetic effects (a theory of *choice poetics*), a student of classical poetics might begin by identifying key works of choice-based narrative. Given such a corpus of important works, a theorist could study them carefully and attempt to observe patterns in their choice structures to build a general theory of choice poetics. Such a theory would be anchored by strong examples from particular well-known choice-based narratives, and might also use comparisons between contrasting choice structures to show the relationships between particular choice structures and specific poetic effects. Individual contributors to such a theory might analyze specific important works or look at particular affects across a range of works.

All of the techniques just mentioned are established approaches in the humanities, and have been used to understand the poetics of linear narratives. Together, they might be called the "classic approach" to the problem. In contrast, the method described here might be called a "mechanical approach." Instead of relying on general principles extracted carefully from many examples via human reasoning, the mechanical approach uses a mechanism for automatically constructing examples based on principles in order to subject principles to broad experimental testing. In other words, the mechanical approach takes intuitions or nascent theories about how choice poetics work and validates or refines them via experimentation based on the output of an artificial system.

This artificial system constructs choices using initial principles, and then experiments can use those choices in order to understand whether the principles embedded in the system are working as desired. When mismatches are found between what the experimenter expects and the choices that are generated, these are used to refine the system's underlying principles into a more robust theory. Key to this process is a system that can generate choices from principles *in a manner that avoids human biases* (hence the term "mechanical approach"). After all, if the procedure for generating examples based on principles were simply "Ask a human to apply the principles and construct choices," the human creator would employ not only the principles being tested but also their own common-sense reasoning and biases to the problem, potentially hiding certain inadequacies of the theory. Productive experimentation is thus enabled by *inhuman* reasoning during example generation (such as that of an answer-set solver).

Note that this mechanical approach is similar to the approach a psychologist might use to understand human responses to choice structures. However, while a psychologist interested in studying human thought and perception might use a carefully controlled experiment to show a link between one specific choice structure and a particular poetic effect, it would take many such experiments to build a broad theory of choice poetics. The mechanical approach proposed here instead tests potentially dozens of interrelated assertions about the psychology of choice structures at once by using a system of rules to construct choices and then seeing whether those choices produce the predicted effects. The advantage of this approach is that it can quickly validate a broad and complicated theory and also provide rich feedback when things go wrong. The disadvantage is that positive results are not as rigorous: a psychologist might hope to prove that a certain mechanism underlies the connection between one kind of choice structure and a particular emotional response, for example, and results of experiments following the mechanical approach won't be able to establish that kind of connection.

Instead, the mechanical approach hopes to show merely that reasoning using a certain theory is *sufficient* to produce choices which impact players in a certain way, and proving why such reasoning works or establishing a strong causal link between choice structures and poetic effects is left as a subject for further, more detailed experiments. Ultimately, the mechanical approach is suitable for initial investigations aimed at efficiently establishing a broad theoretical framework that encompasses many different poetic effects, while investigating the particulars of those effects calls for more refined tools.

The mechanical approach also has strengths and weaknesses compared to the classic approach. The classic approach depends heavily on the insight of the theorists who contribute to it, and often develops idiosyncratically according to their particular interests. However, it has more potential to provide insight about complex phenomena which may be difficult to capture in the kind of generative system that the mechanical approach requires. On the other hand, the mechanical approach requires effort to develop and maintain a generative system, and sometimes ends up producing results that seem trivial in retrospect. The mechanical approach results in theories which are more specific, however, and tends to discover caveats that the classic approach omits because they seem simple or obvious. Ultimately, the classic, psychological, and mechanical approaches are not exclusive: they can be pursued simultaneously and each benefit from this arrangement. In the work describe here, although the mechanical approach is foregrounded, aspects of the classic and psychological approaches are also present, and many of the theories that I draw on were developed entirely using classic or psychological approaches.

2.2 Hybrid Theory/System Development

Part of my research is straightforward from a computer science perspective: I want to build a computer system that does something new (in this case generate

choices). To do this I look for existing theories and craft wisdom about how to put together choices, and start encoding rules into my system. However, as this progresses, the hybrid research approach dictates that I also take what I find and construct a formal theory of choice poetics, suitable for application by humans to existing interactive narratives with the goal of understanding and analyzing how their choices function. By applying my partially-formed theory to existing interactive narratives I can see where it breaks down and start addressing those problems in both the theory and the system that I'm building. This refinement is a natural part of the nascent development of any theory, and corresponds to debugging a computer program during development. In fact, one of the strengths of the hybrid approach is that information flows both ways: refining the theory helps me foresee and avoid problems in the development of my system, but debugging my system also leads to improvements to my theory. By developing both the theory and the system simultaneously, each can inform the other.

2.3 Exploratory Experimentation

Once a system reaches a certain stage of development it's usual to begin experimenting with it formally, proposing hypotheses and collecting results in order to prove the capabilities of the system. In the hard sciences, hypotheses are usually directly derived from existing theories, and experimental evidence either confirms or disconfirms these hypotheses, leading to changes in the theory where appropriate. However, in my case, as the theory is still under development, many of my hypotheses are *exploratory*: I have an idea about how, say, certain outcome configurations will be more satisfying than others, and I can immediately try to use my system to test it.

Because I'm developing both the theory and the system, the goals of my experiments are not to confirm existing theories, but to help suggest whether proposed

theories are viable. For example, although grossly incomplete or misguided theories will fail to produce the expected results when used to generate choices, some subtle confusions or ambiguity may work in the generative case but fall flat when used in an attempt to analyze existing choices. While the exploratory experiments described here establish my theory's capacity to be used in *generating* choices, they are not designed to measure its *predictive* or *explanatory* capacity. Once the theory of choice poetics reaches a satisfying stability as demonstrated by experimental successes in generating choices, further experiments designed to measure its predictive and explanatory power would be in order. Of course, the experiments described here also help test *Dunyazad*, so they serve a dual purpose: they validate my system's capabilities, and also provide high-level feedback for theoretical development. Chapters 7 and 8 present the results of two exploratory experiments, and the analysis in those chapters illustrates how results can shed light on both the system and the theory that it implements.

2.4 System Development Drives Theory Refinement

One complaint that some computer scientists have about theories from the humanities is that they are too vague to be implemented as a program. Creating a computer system that generates predictions or content based on a theory of narrative or the like is a difficult task, because such theories are largely developed to serve humans interested in understanding and analyzing a work, and computers don't have the same common-sense reasoning capabilities as humans. Unsurprisingly then, the system development half of my hybrid research approach tends to contribute details and refinements to the theory half.

A concrete example is illustrative. One of the things that *Dunyazad* reasons about is the impact of outcomes on player goals, and these in turn are defined with reference to states of the world. For example, the "avoid threats" goal of a

victim is negatively impacted when a "threatening" relationship targeting that victim is created or persists. The same goal is advanced when such a relationship disappears. This seems straightforward, but the system quickly figured out that having the player simply exit a scene (thereby disposing of the contents of that scene in preparation for creating a new scene) was a great way to cause "threatening" relationships to go away. Thus when faced with some merchants threatened by bandits, the system thought that players would perceive a "travel onwards" option as unequivocally positive, when in fact it provokes serious moral questions about the responsibility of bystanders, especially if the player's character is well-equipped to fight off the bandits. The solution at a system level is of course to declare that states which change as a result of changing scenes don't have the usual effect on goals. Although this was clearly a bug in my system, finding it also helped me add nuance to my theory: it reminded me that perceptions of goal progress are not just dependent on states which directly affect those goals, but also on other conditions that might make it more difficult for a goal to succeed. A theory developed in the abstract might just talk about "conditions that either positively or negatively impact a player goal," but as a result of having to define those conditions in a way that a computer can understand, I can refine my theory by giving a list of some such conditions and caveats to take into account during analysis.

Figure 2.1 illustrates the basic flow of joint system and theory development: intuitions and existing theories seed a nascent theory of choice poetics, which in turn informs the development of a generative system. That system produces example choices based on the theory, which provide the material for experiments that test the theory. These experiments produce concrete counterexamples—cases where reasoning based on the theory produces an unexpected result—and these counterexamples are used to both debug the choice-point generator and refine the theory. Note that the production of concrete counterexamples is

only possible because the choice-point generator's operation is transparent: the reasoning used to produce examples can be traced back to theoretical statements when those examples lead to unexpected results.

2.5 Related Methodologies

The approach described here can be contrasted with existing methodologies in artificial intelligence. One popular approach in the early days of the field was to set computer programs up as models of human cognition and then test both how well they corresponded to human thought processes and how they performed overall as a means of discovering how intelligence works in general (see e.g., (Newell and Herbert A. Simon, 1976; Herbert A Simon, 2006)). My approach maintains the program as explicitly separate from the theory, however, and

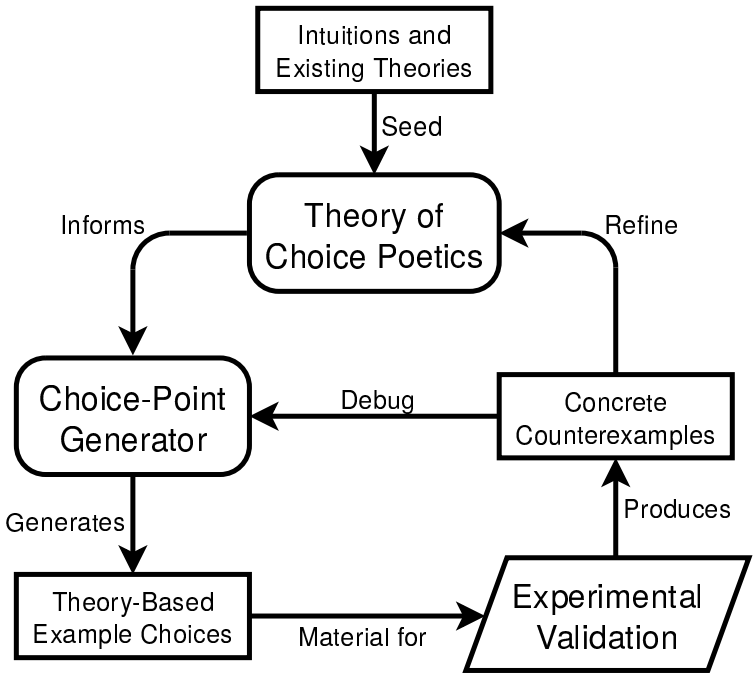

Figure 2.1: The general flow of hybrid theory/system development.

views the program not as a simple incarnation of the theory but as a potentially distorted version which is nevertheless a useful tool for refining the theory. While someone like Herbert Simon might claim that a program *is* a theory (of, for example, cognition), I want to claim that my program *operationalizes* a theory (of choice poetics) and can therefore be a useful tool for revising that theory despite its differences. The distinction here has much to do with the efficiency of prose as a means of capturing ideas too complex for formal code: If one compares the theory (an idea) described in chapter 5 with the ideas latent in the code listed in appendix B, it becomes obvious that the ideas invoked by the prose are much more sophisticated and nuanced than those that could be captured by the code (although of course they are less formal and specific). Especially when a theory deals with something so inherently subjective as human emotional responses to art, tying the theory to a purely formal and procedural structure such as a program seems to impoverish it, and the many caveats to the program's analysis of human perceptions discussed in chapters 7 and 8 bear witness to this.

One major critic of the theory-as-program idea in artificial intelligence was Phil Agre, who argued that the AI systems of his day were limited by the false equivalences the set up between specific computer procedures and general aspects of human thought (Agre, 1997). For example, the word "planning" denoted a very specific kind of computational procedure in the AI literature of the 1970's, but this specific approach was often discussed as if it were equivalent to the common-language meaning of the word in everyday life. Agre argues persuasively that equating the two concepts limits the imagination of computer scientists, who might otherwise be inspired to invent a variety of algorithms by the wide range of activities to which humans apply the word "planning." Agre advocated the application of critical theory to AI and AI discourse: computer scientists should defamiliarize their subjects of study, identify the centers and margins of the

metaphors that they use, and seek to come up with new ideas and approaches that can recenter the margins of existing theories.

Ultimately, I view my methodology as one inspired by Agre's vision. Rather than equate my system with a theory, I view the system and theory as joint efforts which can inform each other, and the system's role is not simply to model what the theory describes, but also to inform the theory when its descriptions are insufficient. In this respect the computer system has explicitly become a tool for *interrogating* the theory, as opposed to a representation designed only to validate it. Accordingly, when the system fails, the response is not simply "The system has been found to be an imperfect implementation of the theory," but a process of blame assignment between the system and the theory which often leads to changes in both.

Thus the results of working with a concrete system ultimately lead to a more detailed theory. As evidence of this process, chapters 7 and 8 present results from two experiments and include information on how the data gathered was used to improve both *Dunyazad* (described in chapter 6) and the theory of choice poetics (chapter 5). As you read the rest of this document, keep in mind that many of my design decisions and experimental procedures wind up being influenced by the goals of both research and system development, and that neither *Dunyazad* nor choice poetics exist merely to enable the other.

Chapter 3

Related Work

The work described in this document derives mainly from the tradition in computer science of building programs that generate narratives. While one aim of this project is to build a program which can intentionally generate narrative choices (a computer science concern: "How can we build a system that does X?") another is to use such a program to learn more about the nature of such choices (a media theory concern: "How does the construction of narrative choices shape player reactions?"). Because of this, it also owes much to narratological theories which have nothing to do with computing, and to ideas related to the psychology of reading, decision-making, and persuasion. The remainder of this chapter discusses these ideas that *Dunyazad* builds on in depth, starting with theories of narrative and moving on to computer systems that generate narrative.

3.1 Narrative Theory

3.1.1 The Foundations of Narrative Theory

Aristotle is one of the earliest scholars to discuss narrative theory, and his *Poetics* is a treatise on the inner workings of Greek drama (Aristotle, 1917). Aristotle

discussed which plot constructions were more effective than others, and identified basic types of dramatic plot, including the comedy and the tragedy, attempting to explain using examples what properties were important in these forms. Although the idea that audience response is a simple function of the content of a narrative work has been recognized as naïve, the ideas of Aristotle sowed the seeds of modern narrative theory. Along with other ancient critiques of drama, such as Horace's *Ars Poetica* (Horace, 1783), *Poetics* helped establish drama, and by extension narrative, as cites of critical study.

An important next step in for narrative theory happened in 1894, when Gustav Freytag published *Technique of the Drama* (Freytag, 1894). Freytag's ideas about the emotional arc of traditional dramas have become standard fare in today's classrooms when students first begin learning how to unpack meaning in narrative. His work on the dramatic structure of the traditional 5-act drama represents a further step from critique or craft advice towards more formal theories of drama (and more generally, narrative).

3.1.2 Formalism and Structuralism

In the early 20th century, this formalization of narrative theory was taken even farther by a group of Russian literary critics known as the Russian formalists. These scholars were exemplified by Vladimir Propp, whose 1928 *Morphology of the Folk Tale* (Propp, 1971) dissected common characters and scenes within a group of Russian folk-tales to come up with an abstracted representation which revealed fundamental similarities between the stories. Propp observed that a simple grammar of a set of thirty-one basic 'functions' could account for the plots of all of the stories that he was analysing, thus creating an abstract and formal description of their common structure. Although this formal structure has proved an attractive theory for computer scientists interested in creating stories, some contemporary literary theorists disagreed with Propp, asserting

that Propp's analysis neglected many important aspects of a story, such as tone and mood. In Prop's (and the formalists') attempts to understand similarities common to diverse narratives, he (and they) are forced to ignore many of the details that make individual stories stand out. Although these details may be safely abstracted while still understanding the *sequence of events* embedded in a particular story, they are often crucial to the *audience's experience* of a story, and thus to poetics.

The Russian formalists began a movement within literary criticism that gave rise to the discipline of narratology. Rather than explicating the details of an individual work or the works of an author or movement, scholars such as Greimas and Barthes sought structural theories that could explain broad phenomenon in many narrative contexts (Barthes and Duisit, 1975; Greimas, 1988). Central to narratology is the idea that to some degree, narratives can be separated into an abstract series of events (the *fabula*, or story) and a particular telling of those events (the *syuzhet*, or discourse). While work like Propp's focuses almost myopically on story at the expense of discourse, some later narratologists focused on discourse-related effects such as focalization. Narratologists created theories and methods of analysis that could be applied to a broad range of narrative contexts. Particularly because of the abstract nature of these theories and methods, computer scientists interested in creating artificial story generators have drawn on them as inspiration for their systems.

3.1.3 Cognitive and Psychological Narrative Theories

Although narratology grew out of literary criticism, in the late 20th century researchers in fields such as psychology and cognitive science began to approach narrative from new perspectives. Psychologists were interested in questions such as "Why do humans enjoy stories?" or "How do humans comprehend stories?" They put forth models of narrative comprehension (Kintsch, 1980; Brewer and

Lichtenstein, 1982), and asked questions about why people liked stories (Iran-Nejad, 1987). These researchers backed up their claims not with careful citation of literary works, but instead with experimental data gathered by observing and surveying human readers.

In the early 1990's, this work accelerated, with work on suspense (Gerrig and Bernardo, 1994), narrative inferences (Graesser, M. Singer, and Trabasso, 1994), and emotion (Oatley, 1995). Models of comprehension continued to develop, including the event-indexing model of Zwaan, Langston, and Graesser (Zwaan, Langston, and Graesser, 1995), which has influenced several recent works in the field of narrative intelligence (see below). This interest in narrative consumption as a psychological phenomenon was largely separate from literary criticism and theory, in large part because it dealt with much lower-level questions. However, in the early 2000's, works like Palmer's *Fictional Minds* (Palmer, 2004) and Zunshine's *Why We Read Fiction* (Zunshine, 2006) began to connect literary theory with psychology. Zunshine in particular draws on the work of psychologists like Oatley (e.g., (Oatley, 1999)) to talk about how novels exercise our theory of mind when we read them. Her work is at heart a work of literary criticism, however, and she uses critical analyses of several well-regarded novels as the basis of her argument. *Why We Read Fiction* is an example of cross-disciplinary analysis: new theories from psychology inspired her to understand well-theorized works from a new perspective. Zunshine has begun the work of combining psychological and critical perspectives on narrative, but of course there is still much unexplored overlapping territory between these two disciplines.

3.1.4 Craft and Literary Criticism

Although narratologists and psychologists have attempted to find a more 'rigorous' and 'objective' understanding of narrative, more traditional literary criticism

and practical advice provides another source of knowledge about narrative. Novelists, writing teachers, and critics (who often wear more than one of these hats) have written down their own views on narrative for both pedagogical and critical purposes. Works like E. M. Forster's *Aspects of the Novel* (Forster, 1974) offer insight into how producers and critics view narrative. As someone who is acknowledged as skilled at the craft of narrative composition, Forster is in a unique place to observe and reveal underlying aspects common across different narrative contexts.

Aspects of the Novel and other works like it critically provide practical advice aimed at prospective authors attempting to *create* effects such as foreshadowing or suspense. It should not be surprising therefore that these resources are sometimes more useful than psychological analyses of the experience of these effects for researchers interested in automatically generating narrative. Of course, craft and critical advice also exists regarding interactive works, some of which is discussed in section 3.2.3.

3.2 Theories of Interaction

Johan Huizinga's *Homo Ludens* is a seminal work on the history and culture of play which laid the foundation for the study of games as a medium (Huizinga, 1949). Huizinga focused broadly on *play* as a social phenomenon, but his major philosophical points apply equally to games as a cultural medium. Ideas like the transformation of norms afforded by the magic circle are just as applicable to the culture of modern videogames as they were to the historical play contexts that Huizinga studied.

Although interactive narratives are by no means a modern invention, the rise of available leisure time in combination with the advent of the computer drastically transformed them over the course of the 20th century. The rise of

media like tabletop role-playing games and wargames (documented in (Peterson, 2012)) coincided with the development of gamebooks (predicted by the work of Borges (Borges, 1956) and experimented with by writers like Queneau at the Oulipo, see e.g., (Queneau, 1967)). Including both purely choice-based narratives like the Choose-Your-Own-Adventure series and more complex books involving dice and combat systems, gamebooks were the first interactive stories to enjoy widespread popularity. Both gamebooks and multi-person tabletop role-playing games gained popularity in the 1960's and '70's which was also when the first electronic games began to reach mainstream popularity (the original arcade *Pong* appeared in 1972). By the late '70's and into the early '80's, a commodity market for computer-enabled interactive fiction began to appear with titles like *Zork* ultimately selling hundreds of thousands of copies (Infocom, 1980).

A fourth interrelated development during this period was the advent of hypertext, which also drew inspiration from Borges' *The Garden of Forking Paths*. Originally developed with DARPA funding to explore computer-based collaboration, rather than for artistic purposes, hypertext nonetheless eventually attracted the interest of writers. Compared to the authors of gamebooks, these writers were interested in the medium more as a new experimental form than as an extension of existing forms like tabletop roleplaying and adventure novels. Because of this, authors like Michael Joyce and Shelley Jackson, who were sometimes also developers of hypertext systems, experimented extensively with their work (Joyce, 1990; Jackson, 1995).

3.2.1 Theories of Interactive Fiction

Early Work

Brenda Laurel's 1986 thesis titled *Toward the Design of a Computer-Based Interactive Fantasy System* is one of the earliest theoretical works that discusses

computer-enabled interactive fiction (Laurel, 1986). Laurel's thesis was a visionary description of a future medium in which computers would enable a drastically more interactive form of theater. Along with other early work like Sean Smith and Joseph Bates' 1989 *Towards a Theory of Narrative for Interactive Fiction* (Smith and Bates, 1989) Laurel's thesis was inspired by interactive fiction software and looked forward to potential future forms of interactive narrative.

Marie-Laure Ryan's *Possible Worlds, Artificial Intelligence, and Narrative Theory*, published in 1991, makes a slightly different connection between narrative and computers: it brings ideas from artificial intelligence (and philosophy) to bear on narratology (M.-L. Ryan, 1991). Ryan's work represents a link between traditional narratology and the study of interactive narratives. Notably, she cites early computer story-generation systems such as *Tale-Spin* and *Universe* (discussed in section 3.3.1).

Foundations

As computer games became more main-stream, work that theorized about interactive narrative as a medium proliferated. In 1997, Espen Aarseth published *Cybertext: Perspectives on Ergodic Literature* and Janet Murray published *Hamlet on the Holodeck: The Future of Narrative in Cyberspace* (Aarseth, 1997; Murray, 1997). These two books are foundational to the study of electronic interactive fiction, and they both include significant theoretical components based on analysis of the interactive fiction of the preceding decades.

Aarseth's Cybertext connects the dots between gamebooks, hypertext, and interactive fiction. In all of these forms, he sees "ergodic literature:" narratives which are not merely present to be observed, but which require work to traverse. The theory of choice poetics presented here is concerned with one manifestation of this phenomenon: choices as a site of player work within the context of interactive stories. By focusing on choices as a specific form of interaction, choice poetics

hopes to understand how authors set up interactive fiction so that the work that players do contributes to their overall experience. In this sense, choice poetics fits into the broader theory of ergodic literature as an in-depth study of a particular form of player-work in interactive narratives.

Murray's *Hamlet on the Holodeck* explores the concept of computer-driven interactive theater envisioned a decade prior by Laurel, but in the context of a vastly different ecosystem of computer games. Between 1986, when Laurel published her thesis, and 1997, when Murray published her book, electronic entertainment had undergone radical changes, including the advent of three-dimensional graphics in games and mass-market personal gaming (in America, the Nintendo Entertainment System was released in 1985; the Nintendo 64 was released in 1995). Murray devotes an entire section of her book to aesthetics, with chapters on immersion, agency, and transformation. These concepts, especially agency, have become central in the study of electronic game aesthetics, and choice poetics fits within this framework as well as a study of one microcosm of interactive experience. While Murray talks broadly about the pleasures and annoyances of interactive media, choice poetics is interested in investigating the detailed mechanisms by which choices as a unit of interaction give rise to specific feelings. In a sense, choice poetics as a study of detailed mechanisms is only possible in a world in which authors like Aarseth and Murray have first explored the larger shape of the space of interactive narrative.

Agency

Drawing on the work of Murray and Laurel, Michael Mateas published *A Preliminary Poetics for Interactive Drama and Games* in 2001 (Mateas, 2001), uniting ideas from Aristotle's *Poetics* with Murray's discussion of agency to propose the idea of agency as a balance between formal and material affordances. By this, Mateas meant that the player's sense of agency in an interactive experience

depends on a balance between actions that the system allows them to do and agendas that the system encourages them to pursue. According to this analysis, a game like *Quake* (id Software, 1996) is a high-agency game because the actions available to the player (killing enemies and progressing through levels) are aligned with the agendas that the fictional world suggests (killing the otherworldly invaders and finding a way to end the invasion of Earth). Although *Quake* may do a good job of providing agency, it only does so for players interested in pursuing a violent agenda, and few games in 2001 provided agency for players who were interested in social interactions. Mateas suggests an architecture for an interactive drama which would provide agency in social interaction, and this project was eventually published as *Façade* (Mateas and Stern, 2002).

This concept of agency was further developed in (Wardrip-Fruin et al., 2009) and later extended by Stacey Mason in (Mason, 2013) to distinguish between diegetic and extra-diegetic agency. Seen as one of the qualities of interactive media that distinguishes them from books and movies, agency in various conceptions is an important topic in the theory of interactive narrative. In a context where discrete choices are the only game mechanic, as in a Choose-Your-Own-Adventure book, any agency present can be ascribed to the choice structures in a work, so in some cases, agency is purely a product of choice poetics. However, because agency as a phenomenon has been so thoroughly considered by other authors, the theory presented in chapter 5 does not focus on agency as the primary goal of choice construction, but instead sees it as one of many poetic effects.

Operational Logics

Another important concept in the theory of interactive media is that of operational logics, first proposed by Noah Wardrip-Fruin in 2005 (Wardrip-Fruin, 2005) and described in detail in (Mateas and Wardrip-Fruin, 2009). An operational logic is a system of processes which work together to present a coherent domain of action

or perception to a player. For example, a resource logic might include mechanisms for buying, selling, trading, mining, and consuming resources within a game. Each individual operation is backed by a distinct set of software routines which each enforce their own logic, but taken together, the player perceives resources as a coherent game element that can be manipulated in various ways. Another example would be the two-dimensional graphical logics of platform games: on-screen representations move in certain ways, affected by gravity but stopping when in contact with a surface below. A host of internal logic supports these behaviors, and together, they are *perceived* as expressing a virtual environment which a character can traverse by running, jumping and falling.

As a theory focused on how computational commitments give rise to player perceptions and conceptions, the idea of operational logics is similar to the idea of choice poetics, which focuses on how choice structures encourage player emotions and reactions. Although choice poetics is focused on the emotional rather than conceptual implications of choices, choice structures within a work can together form an operational logic, and the language of choice poetics can help understand this. For example, if the player is consistently given options to travel north, south, east, or west in certain circumstances, that can be understood from the perspective of choice poetics as a recurring choice structure which presents neutral (and presumably usually mysterious) options, encouraging an exploratory mode of engagement (see chapter 5). At the same time, such regular choices together form an operational logic which presents to the player the concept of a navigable space, something that the same choices would not do if they only appeared once or twice throughout a work. An analysis of choice structures from the perspective of operational logics can thus be helpful for an analysis of their poetics, and vice versa.

Procedural Rhetoric

Related to the concept of operational logics (how low-level system logics combine to form coherent representations) is the concept of procedural rhetoric: how interactive representations can further rhetorical goals. Ian Bogost writes about procedural rhetoric in his 2008 article *The Rhetoric of Video Games* (Bogost, 2008), and Mike Treanor expands on the concept in his thesis (Treanor, 2013). Procedural rhetoric is the study of how interactive artifacts can advance rhetorical goals through patterns of interaction, and choice poetics can be thought of as a microcosm of procedural *poetics*. Of course, rhetoric and poetics inevitably intermingle with each other: a rhetorical strategy such as an appeal to a shared ethical principle falls flat without poetics which reinforce the sense of community between the author and the reader. If procedural rhetoric is concerned with how interactive experiences persuade, procedural poetics would be concerned with how interactive experiences give rise to aesthetics, and choice poetics represents a special case of that.

Motives in Play

In 1996, Richard Bartle wrote about different player types in multi-user dungeons (MUDs) (Bartle, 1996). Bartle's work represents a vein of inquiry that focuses on players as active agents within interactive fiction, and has this in common with theories of reader response in literary criticism. By attempting to categorize players according to their approach to a game, critics can better understand why different players might respond differently to a particular design element or in-game situation. Building on this work in the context of massively multiplayer online role-playing games, Nick Yee published *Motivations for Play in Online Games* in 2006, analyzing play motives in a more recent multiplayer context (Yee, 2006). Although these studies focused on games as social contexts, players

of single-player games also exhibit a range of motivations, as shown in e.g., (Kallio, Mäyrä, and Kaipainen, 2011). Juho Hamari and Janne Tuunanen have published a broad meta-analysis of work on player types (Hamari and Tuunanen, 2014), which identified five key dimensions of player motivation: achievement, exploration, sociability, domination, and immersion. There has even been some work that explores alternative modes of engagement, such as Mary Flanagan's book *Critical Play*, which explores the notion of play as critique of both games and larger social structures (Flanagan, 2009).

Studies of player motivations are important for understanding choice poetics because players with different goals will respond to choice structures differently. In order to understand which outcomes of a choice a player will evaluate as positive or negative, choice poetics is interested in which particular aspects of play each player finds rewarding. Any attempt to discuss choice poetics must acknowledge that player experience is subjective, and must therefore account for differences in option and outcome evaluations between players.

The Player Experience

Like work on motives for play, theories about the player experience center player behaviors rather than games. These theories often develop out of media psychology or communication research instead of literary or film criticism. Studies on specific effects like immersion (Douglas and Hargadon, 2001; Ermi and Mäyrä, 2005) and identification (Klimmt, Hefner, and Vorderer, 2009) are relevant because players of choice-based narratives experience these effects, and because they may arise from both the gameplay and the narrative. Additionally, studies of reasons for engaging with games (as distinct from reasons for specific behaviors *within* gameplay) such as (R. M. Ryan, Rigby, and Przybylski, 2006; Olson, Kutner, and Warner, 2008) provide insight into some of the unique pleasures

that games have to offer, such as autonomy. These studies often focus on differences between games and traditional media such as literature and film, and so choice-based narratives are an edge case in their analyses. However, they also draw parallels between effects experienced by readers and players, and in a choice-based narrative, an effect like immersion could be supported by both story elements and choice structures.

One work that stands out in this category is Alex Mitchell's dissertation on re-reading in interactive fiction (Mitchell, 2012). Focused on player experiences of re-reading in interactive narratives, Mitchell's work draws in part on player experiences with choice-based narratives, and helps untangle the complicated topic of re-reading when choice is present. Although *Dunyazad* does not yet enable or anticipate re-reading, it is an important part of the average player's experience of modern interactive fiction (in the obvious case through saving and loading in digital games), and thus the topic is an important one to choice poetics.

3.2.2 Hypertext Theory

Parallel to the development of early theories about interactive narrative focused on electronic games, researchers and critics interested in hypertext developed their own theories about narratives with links. This particular subset of interactive narrative developed its own strong genre conventions, but among digital narratives of the '80's and '90's, hypertext stories most resemble gamebooks in the sense that they consist of connected blocks of text. If one views each virtual page of a hypertext as presenting the reader with a single choice, where each link on the page is an option, hypertexts can be analyzed in terms of choice poetics (although such an analysis is distorted, as this view of hypertext is not how readers usually approach it). By the same token, hypertext can be used to implement a purely choice-based narrative such as a gamebook, and in fact most of *Dunyazad*'s output formats work within a hypertext engine.

At the end of the '90's, several scholars published on the aesthetics and poetics of hypertext, including Mark Bernstein, Wendy Morgan, and Susana Tosca (Bernstein, 1998; Morgan, 1999; Tosca, 1999, 2000). Of particular interest to choice poetics are discussions of the suggestive nature of links, and of course, of the links-as-choices pattern. In the context of hypertext literature, links are often ambiguous, connecting a word or phrase with an obtusely-related scene. In fact, figuring out the significance of the relationship between linked nodes is often integral to reaching a full understanding of a work. Because of this, hypertext scholars have talked at length about the potentials suggested by a link, and the process of following a link and resolving what one finds on the other side with one's original expectations (see e.g., (Tosca, 2000)). The same process happens at a discrete choice, although some aspects are magnified by the context: the player generally considers the implications of each option, compares them against one another, and then upon making a decision and seeing an outcome, has to reconcile that outcome with the option chosen and the other options available.

3.2.3 Craft and Criticism of Interactive Narratives

Absent a vibrant body of theoretical work, much current knowledge about choice poetics is contained in craft advice and criticism. Although there may be no existing theory of how, for example, regret is created and functions in interactive narratives, authors of interactive fiction certainly know how to evoke regret and use it within plot structures. Through practice and experience, a trial-and-error authoring process is honed into authorial instinct, and so when authors talk about their craft, theorists should listen. Likewise, critics develop a sense for why certain stories are better than others at achieving poetic goals, and can describe in detail which elements of an interactive narrative contribute to or detract from specific effects.

Regarding interactive narrative, resources for the game masters of tabletop roleplaying games provide a wealth of information about how players respond to choices as well as how to manipulate them; see e.g., (Laws, 2001). Even more relevant to *Dunyazad* as a project are blog posts on the Choice of Games website that talk about game design (Choice of Games LLC, 2010). Choice of Games is a company that publishes online gamebooks, and their authors and editors understandably have good advice to give about constructing choices. Articles like "5 Rules for Writing Interesting Choices in Multiple Choice Games" give concrete advice about how to construct choices and in particular, highlight some constructions that are likely to sustain players' interest (Fabulich, 2010). Of course sources like these are tailored to specific writing styles and reflect authorial biases, but in some sense they represent known facts about choice poetics in their particular contexts, and are thus invaluable to a project attempting to lay the groundwork for a theory of choice poetics.

3.2.4 The Psychology of Decisions

Perhaps because digital games offer so many new interaction paradigms and are in fact constantly inventing new ways to tell stories, there are few scholarly resources dedicated to the study of choice-based narratives exclusively. However, the subject of decisions in real-life behavior is of great interest to economists and psychologists, and in these contexts decision-making has received a great deal of attention. A full review of the literature on the psychology of choices is beyond the scope of this work, but several studies that are key to choice poetics are worth mentioning.

First, Amos Tversky has studied framing and context with several co-authors including Daniel Kahneman (Tversky and Kahneman, 1981) and Itamar Simonson (Tversky and Simonson, 1993), famously showing that even when two options are identical from an economic perspective (the opportunity to prevent

a loss of $20 vs. the opportunity to gain $20, for example) they can elicit very different reactions. Because framing has such important effects on preference, it is necessarily important in the construction of choices designed to achieve poetic effects. Tversky's research also demonstrated that the idea of independence from irrelevant alternatives doesn't hold up in many situations, meaning that the presence or absence of a non-chosen option can affect which option a person will choose in various ways. This result is important for choice poetics because it implies that the poetics of a choice depend on the entire set of options presented, and suggests that analysis of a single option or a subset of options is insufficient to understand the full cognitive impact of a decision. Because of this, systems which generate choices passively by independently generating several possible continuations of a plot and then letting a player choose between them will not be able to reason about the poetics of the choices they create: alternatives and the option text that introduces them must be considered simultaneously.

Context is not only important for options, but it can also change how outcomes are perceived. Barry Schwartz, Andrew Ward, John Monterosso, Sonja Lyubomirsky, Katherine White, and Darrin Lehman have proposed a theory of 'satisficing' and 'maximizing' personalities, showing that some people feel regret unless they are confident that they selected the best option at a choice (maximizers), while others are happy as long as the outcome they got was good (satisficers) (B. Schwartz et al., 2002). This again implies that considering the full set of options at a choice is necessary for assessing it's poetic impact, a finding that is reinforced by the data presented in chapters 7 and 8 (see for example section 8.5.4). It also stresses that individual players will react differently to each choice, and authors must be aware of this.

Another psychological theory relevant to the results presented here is decision affect theory (Mellers, A. Schwartz, Ho, et al., 1997; Mellers, A. Schwartz, and Ritov, 1999). Decision affect theory broadly states that when faced with risky

choices, humans make decisions that maximize their expected emotional reaction to outcomes (accounting for various cognitive biases) rather than decisions that maximize their expected overall payoff. One of decision affect theory's specific predictions is that unexpected results will be perceived as more positive than expected results when they're positive overall, but will be perceived as more negative when they're negative (unexpectedness effectively amplifying the perception of goodness or badness) (Shepperd and McNulty, 2002). The results presented in this dissertation largely confirm this hypothesis, and the broad implication for choice poetics is again that the perception of outcomes will be colored by the structure of the choices that lead to them.

Although these various theories of decision making help inform choice poetics, they should not simply be taken at face value in the context of choice-based narratives. Because player motivations are often different from people's motives in real-life situations, and because norms can be substantially altered in the context of playing a game, players may make in-game decisions differently than real-life decisions and may have different responses to their outcomes. Studies of the psychology of everyday choices are thus only a starting point for the study of choices in interactive narratives.

3.3 Computational Narrative Systems

3.3.1 The Foundations of Computational Narrative

While the theories of both interactive and traditional narrative just discussed mainly relate to the development of choice poetics, *Dunyazad* also includes a system-building component. As a system designed to generate poetic choices, *Dunyazad* is part of a long history of computer systems designed to create (or help create) stories. The earliest example of such a system appeared in 1971, in a technical report by Sheldon Klein, John Oakley, David Suurballe, and Robert

Ziesemer titled *A Program for Generating Reports on the Status and History of Stochastically Modifiable Semantic Models of Arbitrary Universes* (Klein, Oakley, et al., 1971). Two years later another publication about the project was simply titled *Automatic Novel Writing: A Status Report*, which made it a bit more clear what the aim was (Klein, Aeschiliman, et al., 1973). Klein and his collaborators created a computer representation of a mystery world that included some random elements, and thus generated different output each time it was run. Although presented as an "automatic novel writer" most of the plot structure in their program's output was hard-coded in the form of events or actions scheduled to take place at particular times. Still, as an initial foray into narrative generation, their system was impressive, especially given the limitations of the computing systems available to them.

It did not take long for the idea of an artificial author to receive more attention. In 1976, James Meehan published his thesis titled *The Metanovel: Writing Stories by Computer* (Meehan, 1976). Influenced by Roger Schank (his advisor at Yale), Meehan built a program that allowed autonomous characters to build and carry out plans. Based on the idea that fables were stories about problem-solving, his program would place characters in problematic situations and then narrate their execution of a plan to resolve their problems. Much more dynamic than the system built by Klein, *Tale-Spin* (as Meehan dubbed it) became an inspiration for future work on narrative generation. Although *Dunyazad* is neither plan-based nor character-driven, the echoes of these ideas are still present in the character motivation system, and in its reliance on canned dramatic situations which drive dynamic character reactions.

Following Meehan at Yale, Natalie Dehn proposed a different model of story generation in 1981 (Dehn, 1981). Her proposed system *Author* would simulate not the actions of characters, but instead the goals and plans of an author. Dehn realized that for many plots, an explanation purely at the level of character

motivations fails to account for many serendipitous events that make the work interesting. However, if we appeal to an author who plans for things like suspense or foreshadowing, it is much easier to explain why events occur as they do. Dehn accordingly planned a system that would simulate an author, thus inspiring an author-simulation-based model of narrative intelligence systems that remains popular today.

Following Dehn, Michael Lebowitz published *Creating a Story-Telling Universe* in 1983, proposing a story-generation system for generating soap-opera episodes called *Universe* (Lebowitz, 1983). Like *Author*, *Universe* worked at the author level, but it focused more on character creation, and it used hierarchical planning to assemble plots (Lebowitz, 1984, 1985). Most of *Universe*'s generation activity consisted of picking plot fragments that elaborated on higher-level plot fragments. For example, *Universe* might start with a high-level plot fragment such as "Mary breaks up with John," and then decide to instantiate this using "Mary and John have a fight and Mary leaves John." Both "Mary and John have a fight," and "Mary leaves John," might themselves be abstract plot fragments with multiple possible instantiations, and the choice of instantiations at any level might depend on the character attributes of Mary and John. Lebowitz' *Universe* approach is thus radically different from the forward-planning approach of Meehan and Dehn, and in some ways can be compared to a grammar-based or template-based system.

As mentioned above, Brenda Laurel's dissertation on interactive theater was published in 1986 (Laurel, 1986), so around this time the idea of computer-generated narratives was supplemented by the idea of computer-mediate narratives. *Dunyazad* ultimately sits between these two traditions, as it includes interactivity in the form of choices for the player to make, but focuses more on offline generation than on-line mediation. Researchers have continued to develop non-interactive story generators, and two more landmark systems debuted in

the 1990's: Scott Turner's *Minstrel* in 1993 and Rafael Pérez y Pérez' *Mexica* in 1999 (Turner, 1993; Pérez y Pérez, 1999).

Minstrel was in fact the impetus for the creation of *Dunyazad* , as explained in chapter 4. It used the author-centric approach of Dehn and Lebowitz but relied heavily on a mechanism called imaginative recall which worked to re-use bits of remembered stories from a story library as parts of a new story. Although Turner does not use the term, it was in large part a case-based reasoning system for story generation.

Mexica took the author-centric approach and pushed it further, modelling not just authorial planning but also reflection (Pérez y Pérez and Sharples, 2001). After *Mexica* creates an episode, it reflects on it, revising in much the same way an author revises their work to create a finished product. Although *Dunyazad* does not have a second-pass refinement step, nor does it explicitly simulate an author, it does devote much of its processing to reasoning about player perceptions. In this regard it is more similar to a system like *Mexica* than it is to systems like *Tale-Spin* or *Minstrel* which do not explicitly reason about audience reception.

3.3.2 Modern Computational Narrative

Story Generators

Following early efforts in computer-enabled story generation, a wide range of methods have been applied to this challenge. Methods including case-based reasoning (beyond *Minstrel*) (Gervás et al., 2005), computational analogy (Zhu and Ontañón, 2010), evolutionary algorithms (Bui, Abbbass, and Bender, 2010), and crowd-sourcing (Li et al., 2013) have been explored. One particularly prolific vein of research has been planning-based story generators, exemplified by the work of R. Michael Young and Mark Riedl, among others (R. Michael Young,

1999; M. Riedl, 2004). These systems work at the author level, but nonetheless use planning to decide which characters will carry out what actions in service of author goals. This line of research has been able to accommodate formal models of various narrative effects as additional constraints at the planning level, including things like suspense (Cheong and R Michael Young, 2006), foreshadowing (Bae and R Michael Young, 2008), and character conflict (Ware et al., 2014). These planning-based story generators share a predicate representation of the world with *Dunyazad*, and to some degree exhibit a similar reasoning process, which is able to integrate relatively arbitrary declarative constraints into the creation of stories. As is evident, this approach has provided fertile ground for experimentation with particular psychological and critical theories of narrative because these models can be integrated as declarative constraints on the output without worrying about designing a process that can reliably produce specific effects. Of course, *Dunyazad* at the moment is focused on constructing individual choices, and so its capabilities for reasoning across multiple actions are much more limited than those of a planner. Because of this, one promising direction for future work would be to integrate *Dunyazad* with a narrative planner, letting the planner specify choice structures at each choice point and later relying on *Dunyazad* to construct the specifics of each choice. Such a setup would involve solving some hard problems, however, such as how *Dunyazad* and the planner would communicate about the actions and outcomes that correspond to each option at a choice.

Interactive Narrative

One of the earliest and most successful projects in interactive narrative was the Oz Project at Carnegie Mellon University. Spearheaded by Joseph Bates (already mentioned in regards to theories of interactive narrative above, which also came out of this project), the Oz Project was a decade-long investigation into

interactive fiction focused on stories supported by believable characters that could exhibit emotional reactions to a player's actions (Bates, 1992). Using reactive planning architectures, the Oz project created virtual characters and then used these characters to tell stories. The most successful system that resulted was Michael Mateas and Andrew Stern's *Façade* (Mateas and Stern, 2002). *Façade* was a true interactive drama that incorporated free-form text recognition input, procedural character animation, and a beat-based drama management system, putting players in the shoes of a visitor in the house of a pair of old friends as their marriage falls apart (or doesn't, depending on the player's actions).

Façade combines multiple approaches including character simulation and drama management into a single experience, but these ideas have been explored by other projects as well. The idea of drama management as an AI technique for improving player experience in interactive fiction was first proposed by Peter Weyhrauch in 1997 (Weyhrauch, 1997). *Façade* incorporated some of the ideas of drama management, and then in 2006 two projects proposed new approaches. Mark Nelson, David Roberts, and Charles Isbell worked with Michael Mateas on declarative optimization-based drama management, a perspective that saw the drama-manager's intervention in an interactive experience as an optimization problem (Nelson et al., 2006). At the same time, Bradford Mott and James Lester saw drama management as a decision problem, which they addressed using dynamic decision networks in *Crystal Island*, an educational interactive narrative (Mott and Lester, 2006). Mott and Lester's work is notable because they used a dynamic model of the player to inform their system's decisions. Later approaches to drama management have included case-based drama management (Sharma et al., 2010), and collaborative filtering for personalized drama management (Yu and M. O. Riedl, 2013). Even when not the focus of research, many interactive narrative projects include some system for managing and directing player actions.

Drama management usually takes the form of subtle manipulations of the world state that make certain actions more- or less-attractive (or simply unavailable) to the player in order to guide them towards a system-preferred path without taking away their agency entirely. In this context *Dunyazad* could be a useful tool for drama management because it knows how to construct choices that not only evoke particular feelings but which also encourage the player to choose particular options. The work of Hong Yu and Mark Riedl (Yu and M. O. Riedl, 2013) deserves further mention here because Yu and Riedl used Choose-Your-Own Adventure stories as the basis for their drama manager. In order to combat player frustration, Yu and Riedl gathered data from many playthroughs of a digital version of a classic CYOA book and used player ratings to determine which decision paths led to more satisfying experiences. At the same time, they developed alternate text for the options at each choice, and learned a model for which alternate option text was most enticing. Then, when a new players interacted with their system, they could determine, based on the sequence of choices a player had made so far, which option similar players had found most rewarding in the path. Using this desirability metric, they were further able to figure out which of their variant option texts would be most enticing for the desirable option (and least-enticing for the non-desirable options) and substitute those option texts, thus subtly influencing player choices. In this manner, without ever changing the choices available to players, they were able to show that players influenced by their system reported more positive experiences overall. Results like those of Yu and Riedl strongly motivate my work on *Dunyazad*: If this kind of choice manipulation can be used to increase player satisfaction, a deeper understanding of choice poetics should be able to explain how they successfully influenced players, and a choice-generating system has an obvious use-case within interactive narrative.

Another important influence on *Dunyazad* comes from Nicolas Szilas' *IDTension* system (Szilas, 2003, 2007). Like Mott and Lester's *U-Director*, *IDTension* incorporates a formal model of the player in making judgements about story construction. *IDTension* also uses predicates to represent its story states, and reasons over them using logical rules.

Its player model is used to score possible actions, and includes components for ethical consistency, motivation, conflict, and relevance. Note that *IDTension*'s player model, like *Dunyazad*'s, is static: it is a projection by the author about generic users, rather than an on-line measurement of an actual user.

The idea of player modelling has been taken further by other systems, including David Thue, Vadim Bulitko, and Marcia Spetch's *PaSSAGE* (Thue, Bulitko, and Spetch, 2008), which tries to identify player preferences and select appropriate content in response to these. Other work has built on this approach, but although *Dunyazad* could potentially make good use of on-line player modelling, it's current player model is static. The problem of recognizing and categorizing player behavior is a difficult one, but *Dunyazad* could also enable a new approach: because *Dunyazad* deliberately constructs choices, one can imagine a setup where it creates some choices designed explicitly to discriminate between player categorizations, and uses these strategically to gain information about a player unobtrusively. This potential synergy between online player-modelling techniques and deliberate choice construction is a promising direction of future work enabled by *Dunyazad*.

One final interactive system with significant similarities to *Dunyazad* is Heather Barber and Daniel Kudenko's dilemma-based interactive fiction system (Barber and Kudenko, 2007a,b). Unlike most interactive narrative systems, Barber and Kudenko's project reasons explicitly about the choices it presents to the user: its main process for event generation is a planning system that continually tries to force the player's character into dilemmas. Although Barber

and Kudenko focus on only a single choice structure, they identify several specific variants of the dilemma (such as "betrayal," which pits the character's needs against those of a friend, and "favor," which positions the player as king-maker between two other characters). The five subspecies of dilemma that Barber and Kudenko identify are used in their system as templates for story situations, but their analysis is also a micro-theory within choice poetics. *Dunyazad* reasons at a slightly higher level, using 'dilemma' as a unified label for one kind of choice that it can produce, but extending it to distinguish Barber and Kudenko's subcategories would be possible. Ultimately, the strategy of using dilemmas to drive narrative engagement is one that *Dunyazad* should enable, along with strategies involving other choice structures.

Studies of Narrative Choices

In the past several years there has been a movement within computational narrative circles towards attempts to reproduce specific affects within generated and/or interactive stories. Several of these have focused on topics that bear on choice poetics, including the topic of agency. In 2012, Matthew Fendt, Brent Harrison, Stephen Ware, Rogelio Cardona-Rivera, and David Roberts conducted a study where participants played through a simple Choose-Your-Own-Adventure-style game (albeit with only a few sentences per 'page') and answered questions about their perceptions of agency afterwards (Fendt et al., 2012). They found that techniques like explicitly acknowledging a player's choice were quite effective at increasing the perception of agency within their story, which reinforces the notion that the poetics of an interactive narrative can depend on textual details as much as abstract events. Although the applicability of their results to full-scale interactive fiction is not clear, their research bears some resemblance to the experiments presented in chapters 7 and 8.

Another very similar study by Rogelio Cardona-Rivera, Justus Robertson, Stephen Ware, Brent Harrison, David Roberts, and R. Michael Young explored player perceptions of the divergence of outcomes (Cardona-Rivera et al., 2014). Again using a simplified CYOA format, they found that when choices had clearly diverging outcomes, they were perceived as more influential than choices whose options indicated similar outcomes. Of course, this might be the case because those choices were in fact more meaningful, but regardless, a link between divergence of outcomes and agency is established. From the perspective of choice poetics, these studies can be seen as investigations into individual facets of specific poetic effects. While agency is a fascinating and important effect, however, there are other effects, even some unique to interactive contexts (like regret) that are deserving of study, and results like this also need to be placed within a broader theoretical context. Accordingly, one of the goals of *Dunyazad* as a project is to establish a broad context for choice poetics and lay down theoretical foundations within which specific effects like agency or regret can be talked about as parts of a larger space.

3.4 *Dunyazad*'s Position

As both a theoretical and practical project, *Dunyazad* inherits much from existing systems and theories. It can been seen as a specialization or extension of several existing theories, including procedural rhetoric, operational logics, formalist narratology, link poetics, and theories of agency. It also owes much to several related fields, such as the psychology of real-world decisions, the cognitive psychology of reading, and of course theories of interactive narrative. As a technical effort, it grows out of rich traditions in both static narrative generation and interactive narrative, and it further draws inspiration from craft advice and critical resources directed at authors of both traditional and interactive narratives.

What distinguishes *Dunyazad* from related projects is first a focus on choices as a poetic form, and second, a commitment to simultaneous and mutually-informative theory and system development. By privileging the explicit discrete choice as the object of study, a variety of interesting effects and interactions are brought into focus that do not necessarily seem important in broader contexts such as 'interactive narrative' or 'hypertext.' This focus also allows an investigation of specific mechanisms, taking a question like "How can interactive narratives bias players' decisions?" and reformulating it as "What specific arrangements of framings, options, and outcomes can cause a player to feel that a choice is obvious?" Furthermore, the development of a generative system which operationalizes the theoretical answer to produce obvious choices allows the theory to be "field-tested," and experimental results can be used to directly inform the theory. Although there are many systems and theories that resemble individual aspects of *Dunyazad*, as the following chapters demonstrate, *Dunyazad* as a project is much more than the sum of its parts.

Chapter 4

Minstrel and *Skald*

Skald is an open-source[1] narrative generator that was created via a process of rational reconstruction, using Scott Turner's 1993 *Minstrel* as the object of study. Like *Minstrel*, it has an underlying graph-based representation of stories, and constructs new graphs via a case-based reasoning process that draws on a fixed library of pre-authored story graphs. I worked on *Skald* with Brandon Tearse, and together we re-created *Minstrel*'s original functionality and then used *Skald* to study *Minstrel*'s strengths and weaknesses (Tearse et al., 2011, 2012, 2014). This section explains how *Skald* works, and then discusses what we learned from building it, and ultimately, why I was unsatisfied with Turner's imaginative recall process for constructing stories.

4.1 Rational Reconstruction

Before getting into the details of *Skald*, first a brief note about rational reconstruction, which was our methodology for constructing *Skald* based on *Minstrel*. The "Rational Reconstruction as an AI Methodology" section of Patrdige and Wilks' *The Foundations of Artificial Intelligence: A Sourcebook* (Patridge and

[1] *Skald*'s source code is available at `https://sites.google.com/a/soe.ucsc.edu/eis-skald/`

Wilks, 1990) provides an introduction to this approach, and examples of it in action include work by Ritchie, Haase, and Peinado & Gervás (Ritchie, 1984; Haase, 1986; Peinado and Gervás, 2006). In fact, Peinado and Gervás' work is closely related to *Skald* as it also used *Minstrel* as source material. As a technical approach in computer science, rational reconstruction begins with an in-depth study of an existing system, so that it can be understood at an algorithmic level. If the system is available for direct study, this includes inspecting the source code and running the original system to see how it behaves. If not, descriptions of the original system's behavior are used to understand how it functions.

Once a system's behavior is well-understood, rational reconstruction proceeds by developing a new codebase to reproduce the functionality of the original. The reason not to use the original code is that developing new code is a means of exposing quirks in the original. Large software projects often contain implicit architectural decisions that are the result of idiosyncrasies in the original code. The original programmer(s) may have been unaware of these decisions, as in their implementation the programming language, or some other feature of their code design, precluded some alternatives. By developing a separate codebase, often in a different programming language from the original, rational reconstruction projects can expose these implicit properties of the original software, and thus learn more about the algorithm being investigated. Rational reconstruction as a technical approach thus echoes rational reconstruction as a philosophical or historical approach in that it works "from the outside" and seeks formal systems of representation that differ from those of its subject as a means of creating a productive alternative perspective (see e.g. (Lakatos, 1971) for several examples of how rational reconstruction has been applied to the history of science).

For our work on *Skald*, we got in touch with Turner, who graciously offered to supply us with magnetic tapes containing *Minstrel*'s source code. Given that we had neither a magnetic tape reader nor a machine that could run *Minstrel*'s

LISP variant on hand, we decided to proceed without the source code, using Turner's dissertation as the reference for *Minstrel*'s design. Turner's dissertation includes detailed descriptions of all of *Minstrel*'s modules, in addition to several appendices, one of which contains an annotated trace of a run of *Minstrel*.

4.2 *Skald*

Turner called *Minstrel*'s core operating principle "imaginative recall." Humans often make up new stories using pieces of stories they've heard in the past, and Turner reasoned that a computer could operate using the same principle: supplied with a story library, it could recall fragments from that library and modify them to fit together into a new story. Turner was interested in computational creativity, and set out to demonstrate that a computer program could exhibit some of the same kinds of creativity as humans do when making up stories.

Besides imaginative recall, Turner's *Minstrel* used a system of what he called "author-level plans" (ALPs) to guide the story generation process. Each ALP took a partially-finished story and helped move it towards completion in some way, usually making use of imaginative recall to fill in some part of the story. Turner's ALPs were responsible for some of the higher-level story structures that *Minstrel* could generate, but *Minstrel* also started each story from a template which dictated a general moral or lesson that the story would convey.

4.3 Story Templates and the Story Library

As a case-based reasoning system, *Minstrel* relies heavily on its story library. Furthermore, *Minstrel* starts each new story by importing a story template from its template library, which is another source of human-authored content. In *Minstrel* and *Skald*, both of these resources are represented in a unique graph-based format which *Minstrel* uses to represent all story content.

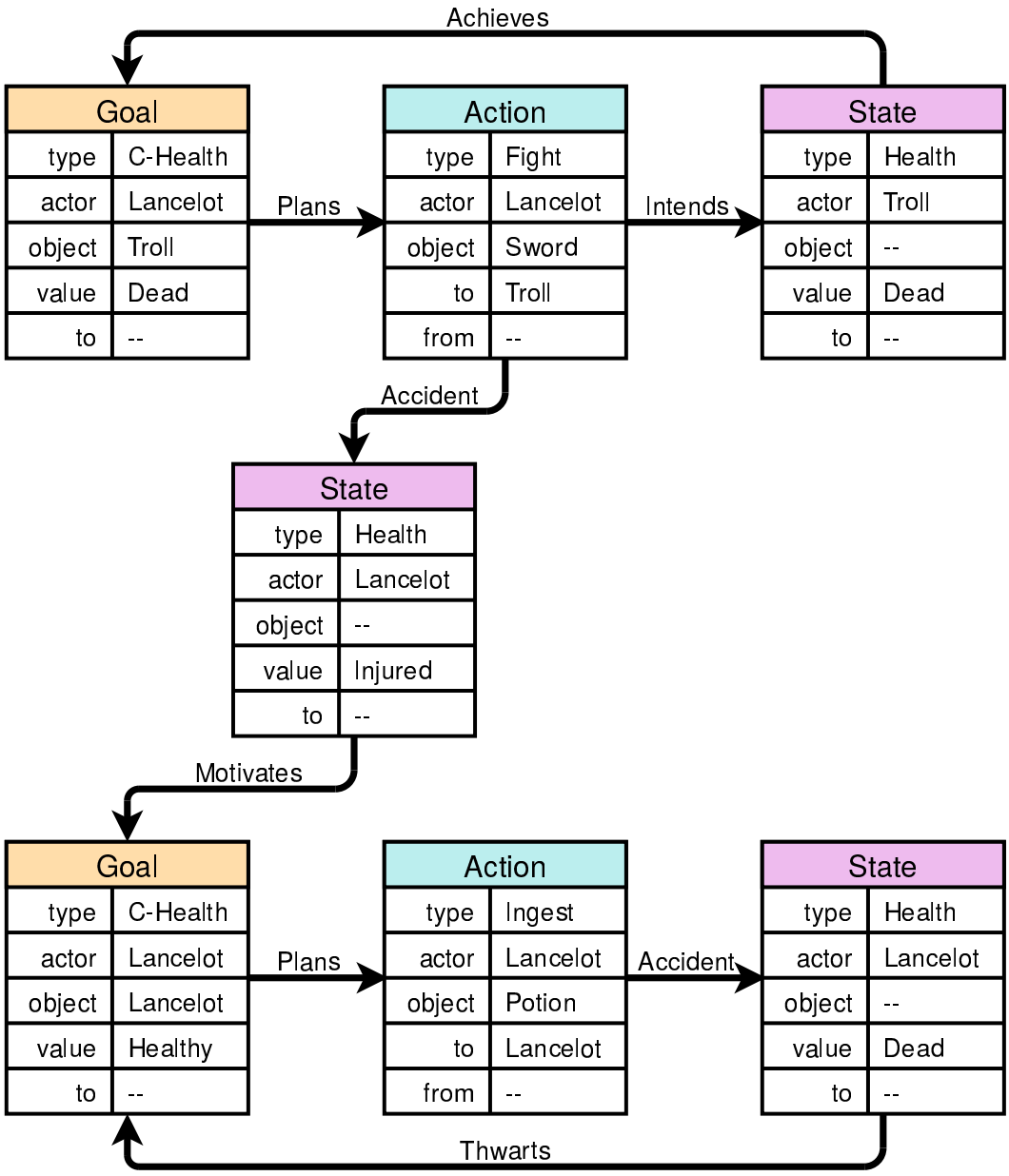

Figure 4.1: An example of a *Skald* story graph. It represents a story where Lancelot plans to kill a troll, attacks it with his sword and kills it, but becomes injured. After that, Lancelot wants to heal himself so he drinks a potion, but the potion kills him instead.

Minstrel's story graphs are directed graphs where each node is a conceptual dependency schema and edges are relations between these. The four core node types are `goal`, `act`, `state`, and `belief` nodes, and they are usually found in `goal-act-state` triangles linked by `plans`, `intends`, and `achieves` links. Essentially, these `goal-act-state` triples each represent a single event (along with its motivation and outcome) and they are linked to each other when the `state` of one triple has a `motivates` link to the `goal` of another. Within each node, the details of a particular `goal`, `act`, `state`, or `belief` are represented using conceptual dependency schemas, as shown in fig. 4.1.

A few of the stories from *Minstrel*'s original library are described in Turner's dissertation, but Turner's complete library was not available to us. Because of this, we created our own ad-hoc story library by hand-authoring several stories that we thought would provide interesting source material. One of the things we learned from this was that *Minstrel*'s story library must be carefully managed to avoid generating malformed stories (this point is discussed in more detail in both (Tearse et al., 2012) and (Tearse et al., 2014)). In particular, whenever there are multiple ways to represent the same concept in terms of story graph nodes, if different stories in the library use different encodings, the results of the imaginative recall process may be poor. The same was true of *Minstrel*'s starting templates: the templates had to match the story library closely in order to generate sensible stories.

4.4 Author-Level Plans

Minstrel's author-level plans are essentially black-box code fragments that modify a story during construction. Each plan is invoked in response to an author-level goal (ALG) and can itself add new ALGs to the current generation process. There can be multiple plans which can satisfy a single goal, in which case they are

tried in a fixed order until one is found whose preconditions are met. ALGs are stored in a priority queue, and if a selected plan fails to achieve the current goal, that goal is re-enqueued at a lower priority level (until it falls below a priority threshold at which it is marked as permanently failed).

This system allows the results of one plan to inform the operations of another when they both consider overlapping regions of the growing story graph. Once all goals have been achieved, the story is considered finished. The first goal of the system (called simply "tell story") is invoked after a story template has been imported, and it assigns both an "instantiate" and a "check consistency" goal to each node in the imported template. The plans invoked to achieve these goals are responsible for most of the generated story, although there are a few plans which run under certain conditions that add additional nodes to the imported template as opposed to simply filling in empty schema fields.

One category of these template-expanding plans is the "make-consistent" plans. These are triggered by consistency goals that arise when consistency-checking plans (triggered by the "check consistency" goals added by "tell story") find an inconsistent node. For example, the `CheckGoalConsistency` plan (which satisfied the "check consistency" goal for `goal` nodes) might find a goal which is lacking a motivating state. In this case, it could trigger the `MakeConsistent MotivatingState` plan, which adds a new `state` node linked to the `goal` node by a `motivates` edge and adds a new "instantiate" ALG targeting the added node. In this manner, the original story graph can be expanded during generation; *Minstrel* is not simply a complicated mad-libs system.

Although *Minstrel* and *Skald* both contain more than 20 ALPs, the single ALP that is responsible for most of the story content generated is the `General Instantiate` plan. This plan is quite simple at the code level: it looks at the node in consideration along with its neighbors in the story graph, and asks the imaginative recall system to fill in the node based on the story library. Of course,

the operation of the imaginative recall system (*Minstrel*'s case-based reasoning engine) is quite complex, and it is mostly responsible for the creative aspects of *Minstrel*'s stories.

4.5 Imaginative Recall

Turner's goal in developing *Minstrel* was to demonstrate the creative potential of imaginative recall, and in particular, how it could recreate elements of of human narrative creativity. Working with stories represented in terms of graphs of story schemas as described above, imaginative recall makes new stories using a story library by taking an incomplete story fragment and filling it in using a similar scene from the story library. For example, a scene where a knight slays a dragon might be recalled when building a story in which a troll is killed, filling in a knight as the protagonist in the new story. Intuitively, imaginative recall recognizes that trolls and dragons are similar because they are both monsters, and so it considers the scene where the knight slays the dragon appropriate source material for the new story.

Imaginative recall actually works via a three-step recursive process: Transform, recall, and adapt. First, an input query is transformed so that it can more easily match story fragments from the story library. Second, a matching fragment is picked out from the library, possibly by recursively applying another transform-recall-adapt sequence. Finally, the matching fragment is adapted so that it matches the context of the current story. In fig. 4.2, the query is a scene in which a troll is killed. This is represented as an `act` node liked to a `state` node with an `intends` link. The `act` node specifies the type of action ("fight") and that a troll is the target, but the actor and tool in this case are still unknown (otherwise the story fragment would already be complete). The `state` node specifies that the troll has a health value of "dead," and it doesn't have any unknowns.

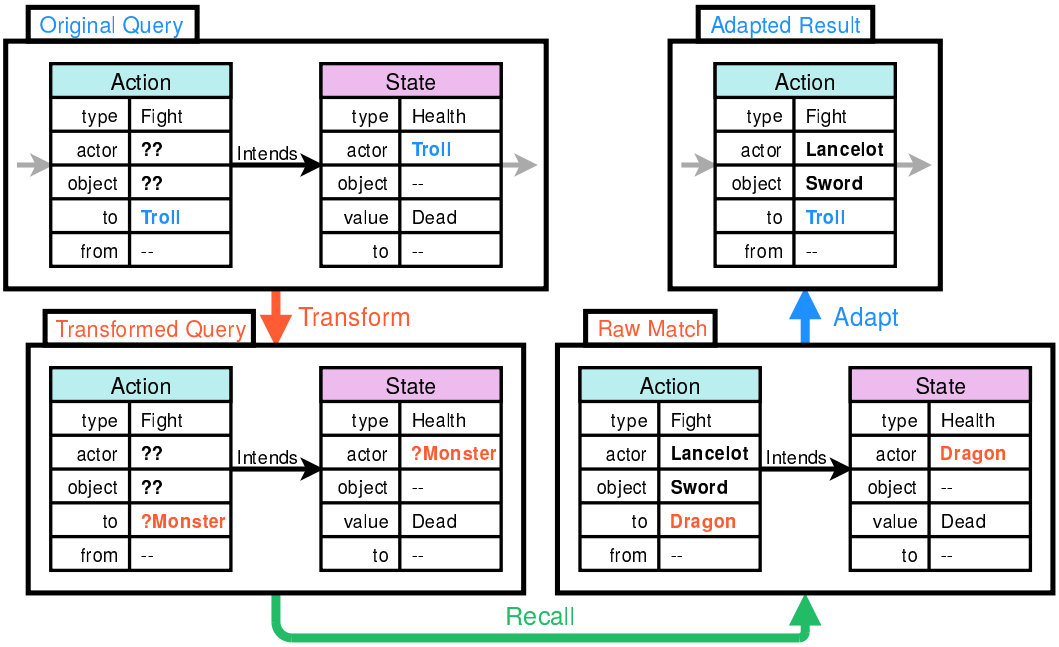

Figure 4.2: An example of imaginative recall using a Transform Recall Adapt method (TRAM). The original query (top left) must be Transformed (bottom left) before a match can be Recalled from the story library (bottom right). The match is then Adapted to fill in the query's target node (context nodes are discarded). Note that each empty slot is potentially connected with others in the graph, so that when "Lancelot" is chosen as the actor here other slots corresponding to that same actor will also be changed to "Lancelot."

Using this query, the imaginative recall system begins by searching for a direct match in the story library: a graph fragment in which all of the fields match with the query (for unknown fields of the query, any value can match). Usually, there is no exact match, as would be the case here if the story library didn't contain any stories about trolls. At this point, the system would begin to consult its library of Transform Recall Adapt Methods (or TRAMs). Each TRAM specifies a specific transformation of a query, along with the steps needed to undo that transformation during the adapt step. Various TRAM selection strategies are possible, including anything from random selection to exhaustive search through the space of possible TRAM applications. Because multiple TRAMs

can be applied to a single query, there's effectively a search space of TRAM applications branching out from each starting query, and somewhere within that space (hopefully) is a query that has an exact match with a fragment in the story library. Ultimately, the odds of a match increase as more TRAMs are applied, because each TRAM tends to generalize the query.

Coming back to our example, in the case of the troll being slain by an unknown actor, one available transformation is the "GeneralizeActor" transformation. This TRAM says that a specific actor, like "a troll," can be replaced with a generic actor of the specific actor's type (using a simple type ontology of actors). In this case, the TRAM replaces all occurrences of the troll in the query fragment with "a generic monster." After transformation, the new query is checked against the story library, and this time, there's a result: the scene where Lancelot the knight kills a dragon now matches our fragment, because the dragon can match "a generic monster." Having Recalled a scene from the story library, the Adapt step is triggered. In this case, the adapt part of the "GeneralizeActor" TRAM indicates that the actor which matched the generalized actor from the story library scene should be replaced by the actor that was generalized. So the fragment from the library in which a knight slays a dragon is adapted by replacing the dragon with our specific troll. If multiple TRAMs had been necessary to find a match, their adapt steps would all have been applied in reverse order. The adapted story fragment can be incorporated into the story, in this case our result is that a knight kills the troll with his sword (see fig. 4.2). The TRAM process has eliminated all of our initial unknowns, and this story fragment is now completed; the ALP system takes over again to select a new target for imaginative recall.

4.6 *Minstrel*'s Potential

The promise of imaginative recall is that TRAMs can capture a notion of story scene similarity, enabling any corpus can be turned into a story library and

become a source for generating new material. Turner acknowledged that some content-specific TRAMs might be needed, and some specific to his Arthurian domain were included in *Minstrel*, but these were a minority. Turner also demonstrated that the TRAM system could create stories that the system had never seen before: it did not just regurgitate what was in the story library. For example, *Minstrel* was able to "invent" suicide by taking a scene where someone was killed and ultimately filling in the killer and the victim with the same person. Because of the kinds of information embedded in *Minstrel*'s story graphs, this process isn't just chaos either: `goal` and motivating `state` nodes constrain how their neighboring `act` nodes are filled in and enforce a measure of believability.

However, despite the potential of *Minstrel*'s imaginative recall process, it is fundamentally not robust. When building *Skald* we imagined that after providing a new story library containing roughly a dozen stories and doing some minor tuning, we'd be able to have *Skald* generate dozens to hundreds of new stories that included significant variations. Unfortunately, what we found was that the story library, the story templates, and the particular TRAMs and ALPs all had to be carefully balanced against each other to avoid nonsensical results.

In our early evaluations of the TRAM system (see (Tearse et al., 2011)), we found that generating 1000 results for a single TRAM query using our new library resulted in an average of only 35.2% sensible results: the remaining 64.8% of results were somehow nonsensical (note that these numbers included duplicate results). We tested several alternate configurations, and found that by removing some TRAMs which generalized too strongly, we could increase the sense-to-nonsense ratio and get an average of 92.4% sensible results (again including duplicates). Unfortunately, the removal of these TRAMs reduced the number of unique results per 1000 queries from an average of 69.4 to an average of 12, and it accordingly reduced the average number of differences expected between any two results from 6.9 to 3.1 (where a difference is something like the substitution

of one character for another; see (Tearse et al., 2011) for methodological details). This result was early evidence of the shape of the tradeoff between sense and variety in results, but it only involved the TRAM system.

Once the full reconstruction of *Minstrel* was complete, including the ALP system, we took some more measurements of TRAM query results in the context of generating full stories (see (Tearse et al., 2012)). We compared a vanilla version of *Skald* (which mimicked *Minstrel* as closely as possible) with a version that used modified boredom and TRAM selection systems in an attempt to trade off some variety for more consistency. The modified version was able to resolve TRAM queries using much less effort (72.2% vs. 59% direct matches; 7.7 vs. 13.8 average TRAMs tried per successful query) largely due to the relaxed boredom restrictions. However, the average number of TRAMs used in each non-direct match fell from 2.38 in the vanilla configuration to 1.39 in the modified version: in other words, the modified version's stories were in general more similar to story content in the library than the vanilla version's results. Based on the earlier experiment, this reduced usage of TRAMs translates to more narratively consistent results at the expense of variety. We had thus successfully found a mechanism for trading variety for consistency, but not a method for increasing one without sacrificing the other.

In fact, in Turner's dissertation, he mentions a trial run where *Minstrel* is asked to generate stories repeatedly based on a single template, in which the coherence of the stories decreases drastically after only 6 reasonable results (see page 672 of (Turner, 1993)). Rather than a robust generation engine with the capacity for creative feats, therefore, *Minstrel* is more like a carefully balanced system designed to demonstrate the possibility of such feats. The process of generating interesting stories while avoiding nonsensical ones using *Minstrel* inevitably involves looking at its output and altering the ALPs, TRAMs, and/or story library to correct errors.

4.7 Problems with *Problem Planets*

In fact, this is exactly the issue that we ran into when we tried to build *Problem Planets*, a Choose-Your-Own-Adventure-style game about climate change that was going to use *Skald* for narrative generation. Brandon Tearse and I teamed up with Aaron Reed to work on the project, which was the first major project where we attempted to put *Skald* to use. Our plan was to set up some domain-specific ALPs and TRAMs for *Problem Planets*' domain (a science fiction domain in which the protagonist travels from planet to planet solving climate-related problems on each) and then have Aaron (a noted author of interactive fiction) create a story library that we'd use to generate new stories for each player. Given this design, *Skald* would have to be capable of generating stories unsupervised: we'd be able to tune the system by changing the library or adding ALPs and TRAMs up to release, but at that point it would ideally be able to produce stories we'd never seen before which were nevertheless coherent and interesting.

What we discovered during this process was that *Skald* alone was incapable of consistent unsupervised generation. Despite our ability to trade off coherence and consistency to some degree, even when boredom and the TRAM system were set up to maximize consistency, *Skald* would produce internally inconsistent or otherwise incoherent stories within the first dozen or so it generated in a batch. This was consistent with Turner's result of 6 stories in a run before incoherence and with the results of our earlier experiments of course, but our efforts to control *Skald* led us to the conclusion that we needed to go beyond the imaginative recall framework to achieve better consistency. In particular, we explored the idea that expanding the story library might give *Skald* more reach, but the opposite turned out to be true: expanding the story library only gave the TRAM system more room to make mistakes.

Ultimately, we turned to the ALPs to fix things. When asking *Skald* to generate a batch of stories in the *Problem Planets* domain, we would observe

broken outputs and try to identify patterns. Sometimes, a single story or group of stories in the story library could be re-written to eliminate a problem (this was often the case when different stories in the library used the same pattern of nodes in slightly different ways, and *Skald* crossed these senses during recall). But this meant that Aaron had to be extremely careful in authoring stories for the library, and in particular, the effort of maintaining the library's internal consistency grew quadratically with the size of the library, as each additional story could potentially conflict with any of the existing stories. Furthermore, problems that couldn't be pinned on specific stories were patched using ALPs that checked for and attempted to fix them.

Although Turner did include a few consistency-checking ALPs in *Minstrel*, as we began to add more and more of them to *Skald*, we realized that imaginative recall alone was no longer a driving force for story consistency. Our architecture was shifting towards one in which imaginative recall supplied creative but often incoherent results, and a complex system of rules implemented by a tangle of ALPs tried to force these results to be consistent. Effectively, we were trying to write a programmatic definition of what made a story consistent within the ALP system. This problem is fundamental to case-based approaches to story generation: either the case library must be rigorously maintained and developed to account for the intricacies of how retrieved cases are joined into a story, or some other system must fix up mis-matched cases after recall. In either case, a limiting factor emerges: the effort required to maintain the story library in the first case, or the system's ability to understand and thus repair mis-matched story content in the second.

Additionally, we wanted the stories to be interactive, and so we decided to generate choices by simply having the story graphs branch and allowing the user to select a path when this happened. We quickly found that this approach required careful hand-selection of branching points to make the choices sensible.

Only at very specific points in a story is it appropriate to present choices to the player, and the range of options that are reasonable is tightly constrained. For choices to be dynamically generated, a strong model of character motivation is necessary, so that options can include a set of motivated actions without including unmotivated or irrelevant ones. Furthermore, interesting options at a choice point can't simply be generated sequentially by a system designed for generating linear stories. Instead, the system must generate all of the options are presented to the player at once, so that the option set avoids repetition and redundancy.

As we struggled with these difficulties in building *Problem Planets*, I came to the conclusion that focusing just on rules of story consistency would be more productive. Seeking to build a story generator that could provide robust variation and also deal with embedded choices, I turned to answer-set programming (ASP), which would allow me to write rules of story consistency as logical statements and let a solver use them to generate concrete stories. In particular, encoding consistency rules as logical formalisms to be obeyed rather than a complex structure of imperative consistency-checking procedures was much more natural. In addition, building a new system would allow me to address from the start the issues with choice generation that I had observed in *Problem Planets*. Thus the idea for *Dunyazad* was inspired by the difficulties of *Problem Planets*.

Chapter 5

Choice Poetics

The aim of this chapter is twofold—first, to establish a framework for talking clearly and precisely about choices and their outcomes, and second, to introduce a method for analyzing choices based on how they relate to player goals. Choice poetics should be able to address questions like: "What is it about a choice that makes some players feel regret after picking a particular option?" This chapter attempts to establish terminology and formal perspectives that can be used to talk about choices, but it's far from a "complete" theory of choice poetics: it only deals with explicit, discrete choices, for example. Some parts of the theory are more developed (those that supported and were influenced by the development of *Dunyazad*), but the ideas here are merely a foundation for an in-depth study of the poetics of narrative choices.

Note that here I define "poetics" broadly as the compliment of hermeneutics: if hermeneutics is the study of the meaning of a text, poetics can be understood as the study of its non-meaning-related qualities, such as the emotions or themes it evokes. Poetics deals with the feelings that a player has before, during, and after making a choice, including simple feelings like the impression that a choice

Note: a preliminary discussion of some of the material in this chapter can be found in (Mawhorter, Mateas, Wardrip-Fruin, and Jhala, 2014).

is difficult to make, but also complex feelings like regret. It is also concerned with how these feelings help a work express themes or otherwise enhance the player's experience of a narrative, although the work presented here focuses mainly on the specifics of how choice structures engender feelings in the players who experience them.

5.1 Inspirations

Chapter 3 contains a full review of literature related to choice poetics, but some of the most relevant sources are worth reiterating here. While choice poetics clearly has roots in the work of Aristotle (Aristotle, 1917) (and more recent narrative formalists like Freytag (Freytag, 1894) and Barthes (Barthes and Duisit, 1975), to name only a few) choice poetics is also inspired by investigations into the psychology of both reading and decisions. In the first camp, psychologists have studied specific narrative effects such as transportation (Melanie C Green and Timothy C Brock, 2000), surprise (Iran-Nejad, 1987), and identification (Oatley, 1995). In the second camp, researchers have studied how things like framing (Tversky and Kahneman, 1981; Tversky and Simonson, 1993), and personality (B. Schwartz et al., 2002) affect decisions and preferences.

Including craft wisdom from places like the Choice of Games development blog (Choice of Games LLC, 2010), existing sources already contain a lot of information about how narrative choices function. However, much of it is focused on either linear narratives or real-world choices, and the resources that do explicitly deal with narrative choices lack a solid theoretical foundation. Some of the work of this chapter is therefore synthesizing this existing information into a theory concerned primarily with discrete, explicit choices in narrative contexts.

5.2 Modes of Engagement

In developing a theory of choice poetics, one must respect the lessons of the development of traditional poetics. In particular, modern scholars have largely rejected "objective" narrative analysis and concluded that the experience of the individual reader must be taken into account. Unlike Aristotle, who stated authoritatively what was good and bad about this drama or that, modern narrative scholars will analyze a work from a particular perspective, without claiming that this perspective is universal, perhaps performing a feminist or Marxist reading of a novel to see what insights a particular lens has to offer about the work.

Choice poetics must also respect the perspective of the reader, but it is in a slightly different position, as its readers are also players. Insofar as the narrative that contains choices is experienced through the fabric of a game, the reader/players (henceforth players) are stepping into the magic circle of the game (Huizinga, 1949) and intentionally taking on certain attitudes. A narrative experienced within the framework of a game thus implicitly biases the attitudes of its readers, for example by setting up a score counter which players can try to maximize. Counter-play is of course possible and important, but even counter-play is influenced by the rules of the game by virtue of being set against them. The formal structures of the game rules that accompany a narrative-with-choices thus allow the theoretician to make an educated guess about the attitudes of players interacting with a piece.

There are still a range of attitudes that players can take on, however, and some understanding of this range is important before any analysis of choices in a narrative. This range of attitudes has to some degree been studied by others interested in player types, although such studies are mostly focused on games without regard to narrative. From simple binary "casual/hardcore" distinctions to more complex typologies such as Bartle's "achievers," "explorers," "socializers,"

and "killers," player type classifications to some degree incorporate notions of player intent or attitude (Bartle, 1996). However, player type classifications also focus on player actions or approaches within the game, for example a distinction between those who enjoy stealth or direct combat more. These player content preferences are as important as any other player preferences when it comes to how players approach choices, but they're less important than player motivations, which establish entire contexts within which preferences can be expressed. For example, if one's motivation to play is a desire to act out a role, one's preferences might be expressed in terms of the kind of role one is trying to act out. If one's motivation to play is to score points, on the other hand, preferences might be expressed in terms of different strategies for doing so.

The idea of "modes of engagement" captures these different player motivations. Modes of engagement are different ways in which a player can approach a game.[1] At this point it is important to note that modes of engagement are neither exclusive nor permanent: players often engage in several modes at once (to varying degrees) and may change their modes of engagement during the course of play. The modes of engagement presented here are also not intended to be comprehensive: these are some of the most common modes, but others may be possible and even the norm for certain games. All of that notwithstanding, consideration of the mode(s) of engagement that players bring to a work is the first step in analyzing the choice poetics of a work.

This can take the form of assuming a particular mode: much as one can perform a feminist reading of a novel, one could perform a power-playing playthrough of a game. This could also take the form of analyzing the game itself to determine which mode(s) of engagement it encourages and rewards. It could even take the form of a qualitative study of actual players, to determine which

[1]Note that the idea of modes of engagement is not unique to games: different audience approaches have been acknowledged as important in areas such as education and music composition (Langer, 1995; Brown, 2001).

mode(s) of engagement they are employing. In any case, an analysis of choice poetics that does not consider modes of engagement is incomplete.

Table 5.1 lists some of the most common modes of engagement, and gives examples of decisions that employ them. The last column references Nick Yee's work on motivations for play in massively-multiplayer online role-playing games as a comparison (Yee, 2006). The grouping of Yee's motivation components into these modes of engagement has to do with a difference in focus: Yee is focused primarily on aspects of play, including a strong focus on online social interaction, whereas choice poetics is interested primarily in single-player offline engagement with an emphasis on narrative. For example, Yee makes fine distinctions between different motivations grouped into the "power play" category here, but from a choice poetics perspective, whether someone is trying to maximize score or compete with an opponent doesn't matter, because both motivations are equally orthogonal to the narrative, and thus they have a similar effect on the player's experience of the narrative.

Although modes of engagement are important to choice poetics, in my work with *Dunyazad* I have not given them great attention: *Dunyazad* encourages avatar play and all of its evaluations assume that players will engage primarily in this mode. Extending *Dunyazad* to support power play and role play better, and to account for these possible approaches when evaluating choices, would be a very interesting line of future work. Developing *Dunyazad* in that direction would allow it to be used as a tool to study modes of engagement in more detail.

5.3 Dimensions of Player Experience

When using choice poetics to analyze a narrative or even just a single choice, one must consider the range of expressiveness of narrative choices. Aristotle devoted much of his analysis of emotion in *Poetics* to tragedy and the heroic

Mode	Decision Process	Example	(Yee, 2006)
Avatar Play	Decide as if you were in the character's situation.	When picking a pet, pick the cat because you like cats.	Role-Playing, Customization, Escapism
Role Play	Decide in order to act out a persona.	Choose the wizard character class because you want to play a shy, bookish person.	Role-Playing, Customization, Escapism
Power Play	Choose options that advance game metrics like score, beating other players, or quick completion.	Sacrifice an ally to obtain a powerful item because it helps you beat the game more quickly.	Advancement, Mechanics, Competition, Teamwork
Exploratory Play	Choose options to see what will happen.	Turn away from the path of your quest to explore the world.	Discovery
Social Play	In a multiplayer situation, choose options because of social considerations.	Turn down a high-level quest in order to accompany your friend on a lower-level quest.	Socializing, Relationship
Analytical Play	Make decisions in order to analyze the work.	Repeatedly load a saved game to explore every option at a choice.	*none*
Critical Play	Make decisions as performance to deconstruct or criticize a work.	Drive your character into poverty in order to demonstrate a game's biased depiction of the poor.	*none*

Table 5.1: Some modes of engagement relevant to choice-based narratives.

Dimension	Description	Potentially Supported By
Immersion	The degree to which the player's attention is focused exclusively on the game.	Outcomes that are believable given options; option coverage of desired actions.
Identification	How comfortable the player feels in the role they play as their avatar.	Options that support player self-expression.
Transportation	The degree to which a player thinks in terms of the game world rather than the real world.	Choices which force the player to reason from a character's point of view.
Agency	Alignment of player goals with game affordances.	Transparent choices where outcomes align with player goals.
Influence	Degree to which the player feels they influence in-game events (even if they do not control them).	Choices where outcomes are divergent and impactful.
Autonomy	The player's ability to choose and pursue a variety of goals at their own discretion.	Choices between goals rather than between methods of pursuing a fixed goal; non-exclusive outcomes.
Responsibility	Player feelings of responsibility for the actions of their avatar.	A mix of intentional outcomes that are good for and bad for other game characters.
Regret	Player feelings of regret for an in-game decision.	Negative outcomes that result from tempting options but which are still believable.

Table 5.2: Some aspects of player experience that are influenced by choice poetics, along with hypotheses about what kinds of choices might support them.

epic as these were traditional forms in his day, but the full range of poetic effects is very broad, and includes both momentary impressions such as a single phrase that provokes disgust, and overall feelings, such as sympathy for a tragic figure. To some degree, more complex effects depend on simpler ones, for example when an overall experience of sympathy depends on many individual positive evaluations of a character, along with negative reactions to bad things that happen to that character. Of course, once choices are involved, the repertoire of effects is expanded, and it's important to have some understanding of what poetic effects a choice can possibly have. Although no summary of possible poetic effects could hope to be complete, I have tried to list a collection of important high-level aspects of narrative experience that are strongly influenced by choice structures, including some which I believe are unique to interactive narratives.

Table 5.2 provides a brief overview of these important dimensions of player experience. Except for identification, transportation, and immersion, each of these possible poetic qualities are unique to interactive narratives: without choices, they simply aren't possible. Because of this, figuring out how choice structures contribute to these effects is an important task for choice poetics. Of course, some of these effects, like agency, have already been extensively studied, often in broader contexts than just choice-based interactive narrative. But it will still be important to understand how choice structures specifically contribute to these poetic qualities. What follows is a preliminary analysis of each quality, based on craft wisdom and existing studies of games and real-world choices.

- **Immersion**—There are several different phenomena that the word immersion can refer to in games, including sensory immersion, mechanical immersion, and narrative immersion (see (Ermi and Mäyrä, 2005)). Narrative choices of the kind studied here bear most on narrative immersion, although they could also be a source of mechanical immersion in some

cases. Immersion goes hand-in-hand with effects like identification and transportation. Although counting immersion as a poetic quality discounts the role of the reader to some degree, immersion isn't independent of the structure of a narrative either. In particular, it is possible for a narrative (choice-based or otherwise) to disrupt player engagement when it contains contradictions or otherwise complicates the reader's ability to understand it (see e.g., (Douglas and Hargadon, 2001)). This is particularly relevant when choices become involved, as they can easily become a source of frustration, either because the player wants to take an action which isn't provided as an option, or because the player views an outcome as inconsistent with the option that led to it.

- **Identification**—Identification in traditional narrative is a key property and is heavily influenced by the actions and attitudes of characters (see e.g., (Feilitzen and Linné, 1975; Oatley, 1995)). In narratives that include choices, the player usually has some control over the actions of one or more character(s), although this control is often incomplete, especially when it comes to attitudes rather than actions. This degree of control can enable a different type of identification in which the player actually assumes the psychological perspective of a character, as opposed to viewing them as a role model (see (Klimmt, Hefner, and Vorderer, 2009)). Choices could have a big impact on identification if they fail to include options which the player believes are reasonable and which correspond to the choice that the player would be most comfortable with in a given situation, simply because such a situation is an explicit manifestation of a difference between the player and the character (see e.g., (Busselle and Bilandzic, 2009)'s discussion of identification in linear narratives). For example, if the player is a pacifist, but there are never options for negotiating with enemies or

otherwise resolving conflicts peacefully, one would expect identification to be hindered. This is complicated by role-playing, however: there is no reason that a player who is a pacifist in real life cannot enjoy the opportunity to play a bloodthirsty character in a game.

- **Transportation**—Transportation is the re-centering of a player's perspective from the real world into a fictional world, and is closely linked with narrative immersion (see (Melanie C Green and Timothy C Brock, 2000; Melanie C. Green, Timothy C. Brock, and Kaufman, 2004)). Although the original research on transportation focused on linear narratives (both textual and visual), a similar phenomenon called presence has been studied in interactive contexts (Witmer and M. J. Singer, 1998). Unlike immersion, transportation is exclusive to contexts that involve fictional worlds. Transportation is also to some degree linked to identification, as when one identifies with a character it becomes easier to project oneself into that character's world (see (Busselle and Bilandzic, 2009) on this point). Again, the believability and completeness of options and outcomes is important when choices come into the picture. In particular, feelings of control and an ability to successfully predict developments have been implicated as contributing to presence (see (Witmer and M. J. Singer, 1998)).

- **Agency**—Agency has been an important subject for games studies as a feeling uniquely enabled by interactive media (see e.g., (Murray, 1997; Mateas, 2001; Wardrip-Fruin et al., 2009; Mason, 2013)). The feeling of empowerment that results from being able to achieve one's goals contributes to agency, and an alignment of player goals with player-influenced outcomes is a powerful formula for agency (Mateas, 2001). Obviously, such an alignment relates intimately to choice structures, as choice structures establish which outcomes can be influenced by the player and also help set

player expectations as to their goals. Given this, opaque choices—where the player is not able to foresee consequences given the setup and options—may decrease agency, because even if an outcome advances player goals, the player may not feel responsible for their success. Additionally, choices where outcomes are tangential to a player's goals can frustrate agency: if you're told to save the princess but then given only choices about which commodities to buy or sell, your ability to proactively pursue your goal has been stifled.

- **Influence**—If diegetic agency is an ability to proactively exert control over the story world (see (Mason, 2013)) then influence is half of that: players' feelings of being influential in the narrative world. Influence is important because fantasies of being powerful are one of the pleasures of both traditional and interactive narratives (see e.g. (Olson, Kutner, and Warner, 2008)). While the experience of agency also requires that players are able to use their influence *proactively* (they must have enough information to make informed decisions, for example), there are plenty of games where the player characters are *influential* but players cannot freely exercise this power to do whatever they choose. For example, a game like *Final Fantasy* (Square, 1987) fully supports player fantasies of being heroic and influential, and if the player makes a mistake this role can be jeopardized, but the protagonists do not have any capacity to control the outcome of their story beyond fulfilling (or failing to fulfill) their fixed destiny. Such a game does not offer players the pleasures of diegetic agency that are uniquely afforded by an interactive format, but it may still offer the pleasure of a power fantasy (a pleasure that is common to both interactive and traditional narratives). Interactivity takes such power fantasies one step further by making them participatory: even when denied agency and

thus unable to decide how their character's influence is applied, players are responsible for their character's achievements by virtue of "pulling the trigger:" without their input the story cannot move forward, even if that input is a simple "yes" or "no." Of course, the choices that a player has can serve to advance or undermine this feeling of influence, especially when, for example, a player re-plays to explore multiple paths in a narrative and finds that nothing changes. Divergent outcomes which matter within the story world are critical for maintaining a sense of influence across multiple playthroughs, although a mere suggestion of such can be sufficient when players do not re-play. Important blind choices (where the player must pick an option knowing that the choice will be momentous but uncertain about the exact effects) are a common source of narrative tension which makes players feel influential without giving them a real sense of agency.

- **Autonomy**—Autonomy is another relative of agency and influence, and can be summarized as the player's ability to choose and pursue their own goals within a game (see e.g., (R. M. Ryan, Rigby, and Przybylski, 2006)). Some amount of active influence is required to have autonomy, but whereas influence is concerned with outcomes, autonomy is concerned with goals. Autonomy is also distinct from agency: when a game sets up nicely-balanced formal and material affordances, agency can be experienced even if autonomy is absent, as in a game like *Quake*. The converse is also possible: an overabundance of autonomy can leave players uncertain about what to do, undermining feelings of agency. A game that supports player autonomy provides a play-space within which players can choose their own goals. To the extent that a narrative-focused game can provide this, it will appear more as a space for player-created stories than a fixed story that affords the player some choice. Games like *Sim City* (Maxis, 1989) and *The Sims*

(Maxis, 2000) provide play spaces within which players can create their own narratives, and this is achieved by offering non-explicit non-discrete choices which give the player a huge amount of options. A lesser degree of autonomy can be present in more linear narratives when the player has choices which affect the goals and priorities of characters within the story.

- **Responsibility**—A feeling of responsibility for one's own actions (as distinct from empathizing with a character who feels the burden of responsibility) is something uniquely afforded by interactive narratives (see e.g. (Murphy and Zagal, 2011)). Choice structures are key to creating this, and there are several conditions for enabling responsibility. First, the player must be given moral agency within the story world, which means that different outcomes at a choice must have consequences which have morally distinct outcomes. Second, the player must be willing to take responsibility for their actions, and this can be impeded when they feel that the outcomes of their actions were unforeseeable or otherwise out of their control (of course, the feeling of a player who mentally justifies their lack of culpability for an in-game consequence is also an interesting poetic effect). Note that this direct player-responsibility is different from role-played responsibility: it's possible for a player to role-play responsibility even when the actions they are assuming responsibility for were completely outside their control. An example of responsibility (or the shirking thereof) as a poetic effect happens in the game Portal (Valve Corporation, 2007), when the player is forced to incinerate their companion cube in order to progress within the game. The complex emotions that arise at that moment are the result of a forced choice between two bad outcomes (being unable to progress and incinerating the companion cube[2]), and the moment has emotional

[2]Note that this analysis makes assumptions about the player's goals and feelings: some player communities see the companion cube itself as an obstacle worthy of hatred rather than a helper who deserves pity, and so view this choice in a very different light.

resonance later in the game when the player has the opportunity to destroy GLaDOS—the agent which forced the decision upon the player—using similar means.

- **Regret**—Like responsibility, the capacity to make players feel regret is mostly unique to interactive narrative (see (Frome, 2006; Zagal, 2009)). In linear narrative it is possible to sympathize and empathize with a character who is feeling regret within the story, but that's a different feeling than personal regret for one's own actions.[3] As with responsibility, making a player feel regret is quite delicate: the player not only has to achieve some level of transportation and identification, but the choices involved must be structured to resist the player's natural tendency to reassign blame. With a negative emotion like regret, the basic psychological mechanisms of denial are the brain's first line of defence, and so naturally the narrative and choices leading up to a moment of regret will be scrutinized for interpretations that leave the player blameless. In particular, if the player can convince themselves that the game forced them to make a choice that led to a negative outcome, or that the outcome was unforeseeable given the option that led to it, they can often avoid regret. Of course, this process of denial is an interesting poetic effect in its own right, and in some cases is exactly what an author wants (perhaps so they can later force the player to re-examine their denial and admit blame, thus experiencing even more poignant regret). A game like The Walking Dead (Telltale Games, 2012) forces the player to make many difficult decisions between options with divergent but uniformly negative consequences, and this pattern engenders feelings of desperation and regret.

[3] A book or film might also cause one to revisit a personal regret, but that is still different from being a source of regret.

The dimensions of player experience listed above could each be (and some have already been) the topic of dedicated research. However, they are presented here merely to demonstrate how much there is to learn about choice poetics—*Dunyazad* as a system does not focus on any of these, but rather attempts to get at some simpler phenomena, such as what makes an individual choice obvious, or what is required for a choice to be seen as a dilemma. Patterns of choices with specific properties such as obviousness or unexpectedness of outcomes come together to produce higher-level poetic effects like those described here, and a better understanding of these low-level effects is required in order to aim at higher-level effects. The poetic details that *Dunyazad* actually operationalizes are based on player goals, and the perceived relevance of options and outcomes to them. As *Dunyazad* only reasons about one choice at a time, its logic effectively comprises an analysis technique for examining the basic poetic properties of an individual choice. The next section describes this technique and how a human would apply it to an individual choice within an interactive narrative, while chapter 6 describes how *Dunyazad* operationalizes this theory, and chapters 7 and 8 describe how results from two experiments using *Dunyazad* have informed both the theory and the system.

5.4 Goal-Based Choice Analysis

When authoring a choice or attempting to understand an existing choice, the method of analysis presented here can help tease out how a choice might be perceived by players. This method is grounded in player goals: it views each option and outcome through the lens of what the player wants, and then tries to synthesize this information to understand how an entire choice is perceived. Figure 5.1 gives an overview of the technique described in this section, which consists of seven steps. The basic idea is to carefully break down how the options

and outcomes relate to player goals, attempting to consider as many details as possible, and then synthesize those detailed assessments back into high-level assessments of the choice as a whole. The next section describes the methodology used to come up with this analysis method, while the following section describes in detail how a discrete, explicit choice is represented for this analysis, and the seven analysis steps are described in the remaining sections. Finally, an example of the application of the analysis technique is presented in section 5.4.10.

5.4.1 Methodology

It is worth reiterating the process used for the creation of this analysis technique. A traditional approach might start with a critic observing regular patterns in choices, and invent a rudimentary framework for recognizing these. Such a critic might then attempt to apply this nascent analysis technique to choices in a few well-known games, and then refine it based on these results, continuing this cycle until the method was deemed satisfactory.

Instead, this technique has been developed with reference to the generative system described in chapter 6. Using the desire to generate poetic choices as motivation, the question "What aspects of a choice need to be understood by the system?" was posed, and a combination of existing systems, craft advice, and theories of both psychology and narrative were consulted to come up with a preliminary answer (see chapter 3; e.g., (Shepperd and McNulty, 2002; Mott and Lester, 2006; Wardrip-Fruin et al., 2009; Fabulich, 2010)). This initial approach pointed to several important factors: player goals, players' evaluations of the relationship between outcomes and these goals, and things like expectations and likelihoods. These are factors that have been used by existing systems, are the subject of craft advice, appear as variables in psychological studies of decision-making, and are components of theories of interactive narrative.

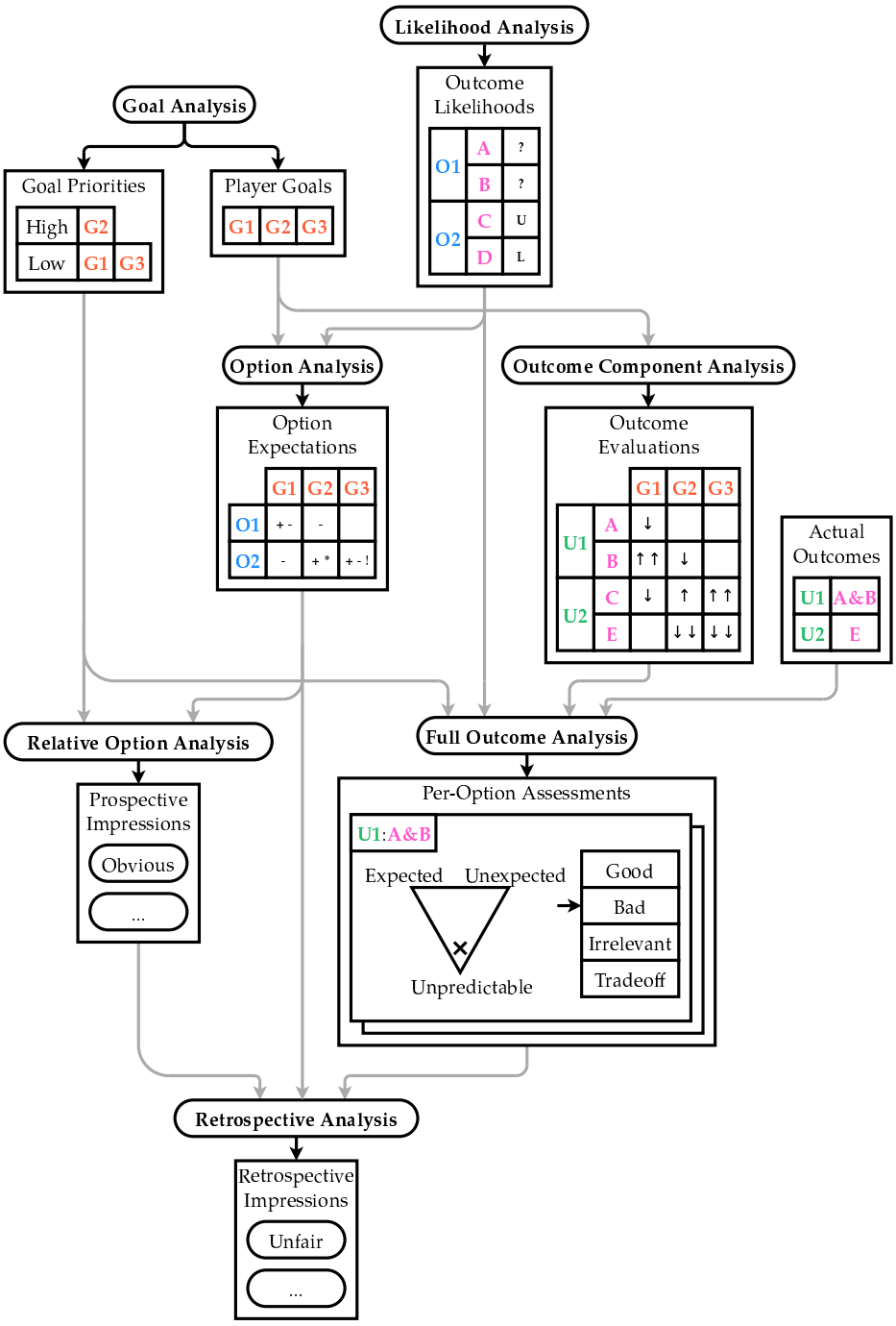

Figure 5.1: Goal-based choice analysis for a single choice—an overview of the different steps and the information produced.

After identifying these factors as important, the next step was a technical one: using these factors as underlying variables, an answer-set programming framework for representing choices was developed (see chapter 6). Technical development proceeded until the system could manipulate these variables and generate rudimentary choices. Of course, the choices it generated at first were flawed, and development of both the theory and the system proceeded through several rounds of revision, using flawed results from the system to motivate elaborations of the theory.

The end product of this methodology for theory development looks a bit different than the results of the traditional approach. For one thing, it is more detailed: because the theory was operationalized early in its development, it naturally contains enough specifics to be evaluated mechanically without necessarily depending on human 'common sense' to apply complex criteria.[4] Another result of the system's unique development path is modularity: each step of the analysis is designed to move from one set of evaluations to another without depending directly on the details of any other step. This modularity is not necessarily beneficial for the resulting analysis technique as a technique, but it does help when attempting to diagnose problems with the system (and thus also with the theory). The separation of modules also allows the theory to be subject to incremental change: an improved method for a specific step could be proposed without needing to overhaul the entire approach.

5.4.2 Choice Representation

This method is developed primarily for the analysis of explicit discrete choices, and for these choices a detailed breakdown of their structure is possible. Figure 5.2

[4]Of course, humans are still better at applying even the specific criteria of this analysis method, for example when comparing the impact of outcomes they can discern many differences that the generative system cannot. As seen in the following chapters, there is no substitute for humans when it comes to understanding how humans feel; the technique developed here only allows *Dunyazad* to guess correctly in *some* situations, although refinements suggested by experimental results may been able to improve things.

shows how a choice (in this case one generated by *Dunyazad*) can be broken down into framing, options, and outcomes. Note that it can be useful to further break down outcomes into individual *outcome components*: individual changes that result from choosing an option which are significant on their own. By doing so, the impact of each outcome component on individual player goals can be analyzed separately, which helps understand outcomes that involve complex trade-offs between multiple goals. Not shown in fig. 5.2 are unrealized outcome components which nevertheless factor into the choice analysis. For example, if the player were unlucky, the third option might not have resulted in the bandits calming down, and one could even imagine a situation where the bandits might turn on the player. Separate analysis of potential outcome components, such as "the bandits calm down" or "the bandits become angry with the player" can be used to examine in detail how each option differs from alternatives, even when some outcome components are exclusive of or linked to each other. It's also the case that understanding what the player thinks *might* happen as a result of choosing a given option is just as important as recognizing what does happen as the story unfolds. Finally, this idea of potential outcome components can also help analyze cases where a system is nondeterministic, and the outcomes of a choice vary between playthroughs. Each part of a choice is subject to scrutiny using the methodology presented here, and having precise language to talk about these choice components is necessary for detailed analysis.

5.4.3 Goal Analysis

As already mentioned, poetic analysis of a choice depends on some understanding of the player's approach to the choice. For goal-based choice analysis, this is taken into account by coming up with a list of player goals. Depending on the objective of analysis, there are different ways to come up with this list. One method is to simply list the goals that the analyst themselves would have when approaching

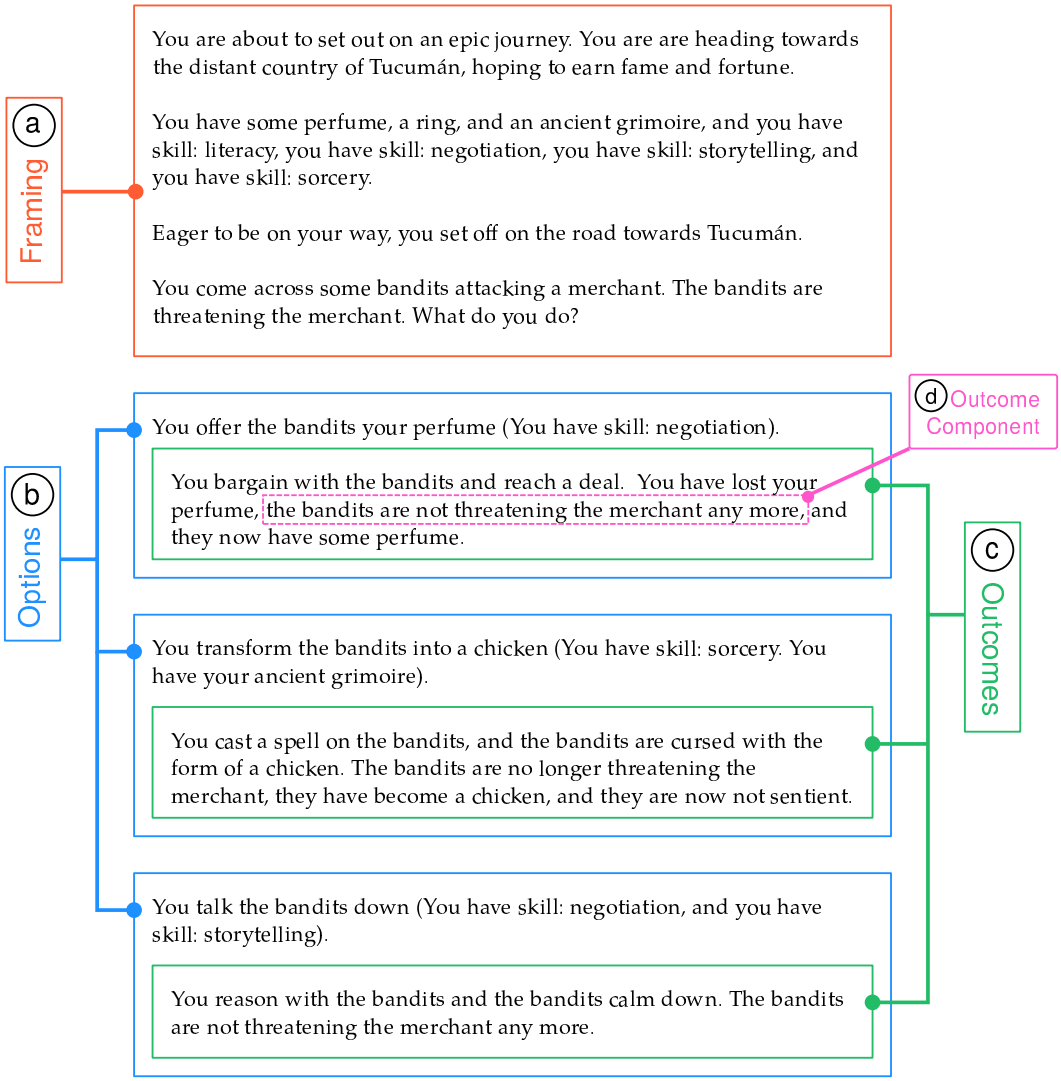

Figure 5.2: Anatomy of a choice (using one generated by *Dunyazad*). (a) Framing—sets up the narrative situation at the choice. (b) Options—discrete options available to the player. (c) Outcomes—Not initially visible to the player, but revealed after a decision is made. Each option has a single corresponding outcome. (d) Outcome components—Individually significant changes that result from picking a particular option.

the choice. Another is to try to put oneself in the shoes of a particular type of player, perhaps according to one of the modes of engagement listed above. This can be useful for an author trying to figure out how different types of players will perceive a choice: just come up with goal lists corresponding to each player type of interest and perform separate analyses using each list.

In any case, coming up with these lists should take into account not only the desires of the actual or imagined player, but also what aspects of the framing of the choice and previous content encourage which player goals. Based on the options available at previous choices and the narrative content, games can not only encourage certain general modes of engagement, but can establish and encourage the pursuit of specific goals. This encouragement itself often depends on a player's mode of engagement to function as intended, however. For example, game rules directly establish goals for power gamers, but players who have a different mode of engagement may not care about those goals.

If an important goal is missing from the goal list, the entire analysis may be skewed, so it's important to consider the full set of probable player goals. On the other hand, it's always possible to come back to the goal analysis if one realizes during a later step that there's an additional goal that might be relevant. This is not uncommon, as the particular set of options and outcomes at a choice determines which goals are relevant. However, part of the point of performing goal analysis before even considering the structure of the choice in question is to avoid bias. This also means that the results of goal analysis can be re-used when analyzing multiple choices. Of course, player goals may change over the course of a game, and this must be taken into account, but performing a full goal analysis from scratch for each individual choice is generally not necessary.

Besides just listing player goals, a goal analysis must give some notion of their relative priorities. This can be as simple as sorting the goals into "high" and "low" priority tiers, but a more detailed representation of priorities can

also be used. This priority information is used when combining information about multiple options and outcomes during the relative option analysis and full outcome analysis steps. Because goal priorities are even more volatile and difficult to estimate than goals themselves, sometimes it's simplest just to wait until the later analysis steps to consider goal priorities, as there will likely be only a few specific goals (those relevant to the choice in question) for which priorities actually affect the analysis.

5.4.4 Likelihood Analysis

The second step of analysis is to figure out which potential outcome components are made to seem likely by the framing and option text of the choice, and assign each component a label of **likely**, **neutral**, or **unlikely**. For each option, this step starts by identifying a set of outcome components that seem plausible, and then figuring out which of those are likely and unlikely. The goal of this step is to identify outcome components that the player can *presume* will result from each option, so if there are actually multiple conflicting likely scenarios (combinations of outcome components), only outcome components present in *all* probably scenarios count as **likely**. For example, if one option involves rolling a six-sided die, but the player has been given a prophecy that the result will be either a six or a one, then the outcome components associated with rolling a six and those associated with rolling a one are all **neutral** in this analysis, while outcome components associated with other rolls are **unlikely**. Any outcome component that's associated with both rolling a six and rolling a one would still be marked as **likely**: given the information the player has, they can assume that such an outcome component will happen if they choose to roll the die.

The outcome likelihoods produced by this step are used during option analysis in combination with the player goals from goal analysis to figure out how each option portrays itself as affecting each goal. Figure 5.1 displays these assignments

in the 'Outcome Likelihoods' box: the choice has two options, O1 and O2, and O1 suggests outcome components A and B, while O2 suggests outcome components C and D. Components A and B are both **neutral** (indicated by a '?'), while component C is **unlikely** (a 'U') and component D is **likely** (an 'L'). Note that it's possible and even common for the same outcome component to be a plausible result of multiple options, although such a situation is not shown in fig. 5.1.

During this step, the actual outcomes of each option are irrelevant; what's important are the cues present in the framing and options that hint at what might happen if an option is elected. This often includes a large body of implicit player knowledge that depends on player experience with a game: players who have played a game already (or even just games from the same genre) may build strong expectations from minimal cues. Much like player goals, potential and likely outcomes from a player's perspective can be difficult to predict perfectly. However, from a designer's perspective, each choice is usually explicitly crafted to present certain outcomes as most salient, and in that case, this step of analysis is mostly focused on double-checking the framing and options to ensure that they highlight the intended outcomes properly.

5.4.5 Option Analysis

Given the player goals and estimates of likely-seeming outcome components, option analysis proceeds by analyzing how each option seems likely to impact each goal, aggregating information across outcome components at each option. First, all *possible* outcome components are considered, to come up with an estimate of the goals that could possibly be impacted by each option. These evaluations are **enables** and **threatens**—which goals each option might possibly advance, and which goals it might possibly impede. In fig. 5.1 the '+' and '-' symbols in the 'Option Expectations' box are these **enables** and **threatens** assignments.

Essentially, these **enables** and **threatens** evaluations are produced by considering the impact of each outcome component on each goal. If any possible outcome component of an option could negatively impact a particular player goal, it is said that that option **threatens** that goal. Likewise, if an outcome component can positively impact a goal, it is said that that option **enables** that goal. It is possible and even common that an option both **enables** and **threatens** the same goal, and of course it's also possible that it neither **enables** nor **threatens** a particular goal (or even any goal at all). For example, an option to "attack the dragon" presumably both **enables** and **threatens** the goal of preserving one's life, as both being killed and surviving are *possible* outcome components (even if one seems more likely than the other).

Besides *possible* impacts of each choice, option analysis is also concerned with *likely* impacts. Using the same procedure, but this time only considering **likely** outcome components, **advances** and **hinders** option expectations are assigned to each option/goal pairing (these are the '*' and '!' symbols in the 'Option Expectations' box of fig. 5.1). In other words, when an option's **likely** outcome component positively impacts a goal, that option is said to **advance** that goal, and a similar negative impact results in a **hinders** label. If it's not clear whether an option **advances** or **hinders** a goal (perhaps because of multiple conflicting outcome components), neither label should be assigned: the **enables** and **threatens** labels should already be present and serve to indicate the possible but uncertain outcome relative to that goal. On the other hand, if contrary indicators are present but one is clearly stronger or much more likely than another, the label more strongly indicated can be assigned, but this should only be done if the player can safely assume that the goal will be affected as indicated, as that is the purpose of these labels. Note that an **advances** label can only be present if there is an **enables** label, and similarly a **hinders** label requires the presence of a **threatens** label. The combination of **enables**, **threatens**, and one of either **advances** or

hinders is common, and indicates an option that will probably advance or hinder a goal but with some risk involved. For example, the hypothetical "attack the dragon" option discussed above might not only **enable** and **threaten** the goal of self-preservation but also **hinder** it, if the player believes that being killed is a **likely** outcome component and that staying alive is not.

The separate analysis of all vs. **likely** outcome components and the four labels that result allow a wide range of choice structures to be accurately described in simple terms, especially when comparing labels assigned between one option and multiple goals. The analysis of all possible outcome components for the **enables**/**threatens** labels captures a broad view of option/goal relationships while the **advances** and **hinders** labels based on only **likely** outcome components capture information about the perceived "most likely scenario." Patterns across multiple goals (for example a "tradeoff" when one goal has an **advances** label while another has a **hinders** label at a particular option) are easily recognizable, and this is the basis for the relative option analysis step.

Remember that this step still makes no use of the actual outcomes of an option: it is just concerned with how the player views each option before making a decision. It can also ignore the goal priorities because each goal is considered separately. The option expectations produced by this step are used in the relative option analysis and retrospective analysis steps as a formal representation of how the player views each option. During those analyses more fine-grained evaluations of player perceptions may be required on a case-by-case basis, but the **enables**/**threatens**/**advances**/**hinders** labels allow such complicated cases to quickly be separated from the simple cases.

5.4.6 Relative Option Analysis

After producing option expectations for each goal/option pairing, a full-blown analysis of the choice from a pre-decision standpoint can be produced by comparing the options available. A set of prospective impression labels can be used

to identify common option structures, and choices that don't easily fit into a known category can be examined more closely. This step uses not only the option expectations from the option analysis step, but also the goal priorities that were produced during goal analysis.

Certain common patterns of option expectations create well-understood prospective impressions. For example, a dilemma has a specific feel, and several types of dilemma can be easily identified by comparing option expectations. If there are exactly two options at a choice, each has overall negative impacts on high-priority player goals (and these impacts are roughly balanced), and each option negatively impacts a different set of goals, then the choice is a classic dilemma. All of this information (besides the balance of impacts) is directly encoded in the goal analysis (priority information) and the option analysis (**hinders** labels) results, so classic dilemmas are easy to identify using this analysis method. From a design standpoint, it's also easy to identify when an option isn't a classical dilemma and which aspects of that option would need to change to make it one. Given the detailed language of choice analysis it's further possible to describe other kinds of dilemmas: a two-option choice with balanced positive outcomes, or a three-option choice that still has balanced outcomes, or even a tradeoff choice where each option has positive impacts on one of two goals and negative impacts on the other. This is one of the benefits of a detailed analysis technique, in fact: developing the precise language necessary for describing choices formally establishes a framework within which variations on a common pattern can be identified and enumerated.

Table 5.3 gives an overview of some prospective impression labels and the criteria for applying them based on goal priorities and option expectations. Note that the criteria are designed to be sufficient for applying the label, but not necessary—there are certainly some obvious choices which do not meet the strict criteria set out here, but from a design standpoint, if the criteria are met, a

Label	Description	Criteria
Depressing	A choice where no matter which option you choose, there's a goal that's hindered .	Each option **hinders** at least one top-priority goal. No option should **enable** or **advance** any top-priority goals.
Dilemma	A difficult decision between two strictly negative outcomes. (A particular kind of **depressing** choice.)	Exactly two options, each of which **hinders** one of two different top-priority player goals. The priorities of the goals and the severity of the consequences should be balanced and neither option should **enable** or **advance** any goals (even low-priority ones).
Empowering	A choice where every option has a positive impact.	Every option **advances** a player goal, and may **threaten** one or more goals but does not **hinder** any.
Obvious	A choice that has one option which is clearly better than the rest.	One option that **advances** a top-priority player goal without **hindering** any (although it may **threaten** some), while none of the rest of the options **enable** any top-priority goals, and each of them **threatens** some goal.
Relaxed	A choice that has no impact on any high-priority goals and where no goals are threatened.	There are no option expectations involving high-priority goals (positive or negative), and there are no **threatens** expectations (and thus no **hinders** expectations).
Mysterious	A choice where there is no indication of which option is best or how the outcomes might affect things.	There are zero option expectations, including **enables** and **threatens** expectations.

Table 5.3: Several different prospective impression labels, with formal criteria based on option expectations and goal priorities.See section 8.11 for a detailed discussion of some of these labels informed by experimental results.

choice is almost certainly obvious (assuming also that the goal and outcome analyses are accurate of course). The labels in table 5.3 and their criteria were developed as part of *Dunyazad*'s choice analysis component, and some of them (the **depressing**, **empowering**, **obvious**, and **relaxed** labels) were refined based on experimental results (see chapters 7 and 8).

As a preview of those results, one caveat that applies to the **relaxed** label is that some people will always seek a best possible outcome when given a choice, and thus even at a choice among supposedly positive outcomes, options which appear to lead to *relatively* worse outcomes may be seen as bad options (B. Schwartz et al., 2002). See section 8.11 for further discussion of some extra considerations for applying these labels.

The labels presented here give an idea of how players will feel when faced with this choice before they make a decision. Classic dilemmas, for example, are stressful decisions which are difficult to make: they represent significant negative events and are a source of narrative tension (see (Barber and Kudenko, 2007b)). In linear narrative, dilemmas and their consequences are commonly used as low points in a story, and are themselves often a result of a character's previous poor decision, in effect serving as extended punishment: the character must not only suffer a negative consequence, but must also choose which consequence to accept, thereby being forced to reflect on their initial decision while agonizing about their present options. Despite being negative moments, dilemmas can also serve as moments of clarity: when a character finally confronts the bad options available to them and makes a decision, this can mark the end of denial about their situation and ultimately the beginning of redemption.

In an interactive narrative, presenting the player with a dilemma can serve the same purposes: to accentuate an earlier mistake and punish the player, while perhaps also establishing a new goal of moving towards atonement. A classic dilemma is a moment of heightened tension, and when a player is forced to make

the decision it also represents a pause in the action, although a dilemma where the player has limited time to choose can also be used to accentuate feelings of urgency and haste (*The Walking Dead* makes effective use of both kinds of dilemmas (Telltale Games, 2012)). Additionally, when a player is confronted with a diegetic dilemma, any reasoning about which option is best is done from a point of view within the narrative, which may serve to heighten transportation. At the same time, since diegetic dilemmas can force the player to make an important decision for their avatar and to "think as" their avatar, they may promote identification.

Dilemmas can be very fragile, however, because if the goals hindered by the two options are not balanced, the choice can become obvious (this impacted the experiments described in chapters 7 and 8). A dilemma may also force a player to re-evaluate their priorities, especially when the player goals involved relate to different modes of engagement. For example, if one option hinders a player's ludic goal of becoming more powerful (without necessarily affecting a similar diegetic goal) while another hinders the diegetic goal of preserving the safety of their avatar's ally, many players may simply make a decision to prioritize one mode of engagement (avatar play or power play, in this example) over the other, and thus avoid the dilemma. Of course, a choice like this that forces the player to decide between orthogonal goals can be used intentionally by an author to build divergent experiences for players who presumably prioritize different things. Such a choice not only measures a player's priorities but also probably affects them: human cognitive biases related to self-consistency and rationalization mean (see e.g., (Hall, Johansson, and Strandberg, 2012)) that once a player commits to a certain set of priorities they are likely to stick with that decision.

A label like **dilemma** can thus be unpacked into a rich understanding of the poetics of a choice. Beyond assigning such well-understood labels, however, relative option analysis can also find choices which don't strictly conform to any

labels, and analyze them on a case-by-case basis. In these cases it is useful to compare a specific choice structure to a known structure and focus on the differences. For example, a choice that meets all the criteria for being **obvious**, but where the best option also **hinders** a top-priority goal, could be compared to an **obvious** choice. In this case, the relative merits of the goals **advanced** and **hindered** by the best option might mean that the decision is still straightforward, but even so the player may feel some reluctance that a normal **obvious** choice would not create. Alternatively, because of the **hindered** goal and despite the **advanced** goal, an option which **threatens** some unimportant goal and has no other expectations might be more desirable than the 'best' option if the player is risk-averse, creating a very different impression. The relatively narrow labels presented here thus serve as anchor points for the analysis of more complex choices. Of course, it's possible to establish many more clearly-defined labels than those presented in table 5.3, and doing so is a straightforward way to increase our understanding of choice poetics.

5.4.7 Outcome Component Analysis

While the results of relative option analysis give a sense of how a choice is perceived just before the player makes a decision, more work is necessary to understand how a player feels after experiencing an outcome. This begins with outcome component analysis: Using the list of player goals and the actual outcome components at each option (as opposed to the potential components identified during likelihood analysis), a breakdown of the actual impact of each outcome component on each player goal is created (unlike the option analysis step, this step does not aggregate information across outcome components at each option, instead evaluating each outcome component/goal pairing individually). Recall that the option analysis step already summarized how the framing and options of a choice indicated goals would be affected, but during this step, actual effects

based on actual outcome components are considered instead of hypothetical effects based on potential outcome components.

An example of this distinction can be helpful: let's say the player has been flung off of a cliff and faces a choice between grabbing a nearby branch to save themselves or grabbing the outstretched hand of their avatar's only child, who is also falling. As presented, the outcomes look grim: sacrifice any hope of saving one's child to save oneself, or catch one's child and at least suffer the same fate. Assuming that the player can't reasonably expect some sort of miracle to save them (even given the full implications of genre conventions, etc.) both options are portrayed to have likely negative impacts on several player goals (assuming an avatar play mode of engagement). But perhaps if the player chooses to grab their child, the actual outcome is that a freak gust of wind blows them both to safety, while if they choose to grab the branch, their child falls and dies. This outcome component of being blown to safety would not appear during likelihood analysis and it would thus not be considered during option analysis, but as it is an actual outcome component, outcome component analysis would evaluate it (see outcome component 'E' in fig. 5.1). The goal of outcome components is to analyze the valence of actual outcome components with respect to each player goal, including any components that weren't suggested by the framing and options of a choice.

The method is simple: for each outcome component that actually happens at each option (or when outcomes are non-deterministic, for each outcome component that could happen), estimate its impact on each player goal. The 'Outcome Evaluations' table in fig. 5.1 that results from outcome component analysis illustrates this, using '↑', and '↓' for minor positive and negative consequences, and '↑↑' and '↓↓' for major consequences respectively. Of course, a finer gradation of consequences could be used if desired, and certainly should be employed on a case-by-case basis where specific consequences must be compared, but gross distinctions are often sufficient to analyze straightforward choice structures.

Note that even though the outcome component 'C' of outcome 'U2' does not actually occur according to the 'Actual Outcomes' table, it is still analyzed in this example, perhaps because it might occur depending on the circumstances. In contrast, the suggested outcome component 'D' of option 'O2' does not appear in the analysis of 'O2's outcome 'U2,' presumably because although implied as a possibility, it will never actually occur. Outcome component 'E' is the opposite: it was not implied, but does occur, and so must be analyzed. There are thus several categories of outcome components: *suggested* components that are hinted at by the framing and option text of a choice, but which may or may not even be possible, *potential* components which are possibilities given a specific decision, *actual* components which actually happen in a given play-through, and *unrealized* components which could have happened given a particular decision but did not. In the example analysis from fig. 5.1, outcome components 'A,' 'B,' and 'C' are both suggested and potential, while component 'D' is only suggested. Meanwhile, components 'A,' 'B,' and 'E' are actual, while 'C' is unrealized; note that component 'E' is not a suggested component.

Depending on the interactive narrative, complex rules may exist that establish what configurations of actual outcome components out of the set of potential outcome components are valid, and the player may or may not be aware of these rules. If analysis is being conducted without a specific playthrough in mind, then in latter stages the different possible combinations of potential outcome components will have to be considered separately, as each may give rise to a different player experience. Of course, player experiences must already be separated based on which option is chosen, but this does add some complexity to the analysis. In the rest of this chapter outcomes are assumed to be deterministic, as that is the case for choices generated by *Dunyazad*, and it generally makes analysis much more straightforward. Once an understanding of how each potential outcome component affects each goal is reached, this information can be used to analyze the possible outcomes.

5.4.8 Full Outcome Analysis

By combining information from the goal analysis, likelihood analysis, and outcome component analysis steps, as well as information on which outcome components are actual for each option, a full picture of the effects of choosing each option can be assembled. This stage of analysis summarizes some details of earlier stages to make common choice configurations easy to recognize, although for complicated structures these summaries may be insufficient and the results of earlier analyses will need to be used directly by the final retrospective analysis. The goal of this stage of analysis is to establish two main evaluations for each option (and the corresponding outcome components): first, to what degree was the result **expected**, **unexpected**, or **unpredictable**, and second, was the outcome overall a **good** one, a **bad** one, a **tradeoff**, or **irrelevant**.

The evaluation of expectedness relies on comparing option likelihoods from the likelihood analysis step with actual outcomes. First, however, each outcome component is established as either **important** or **unimportant** based on whether or not it affects any top-priority player goals (using actual outcome components, not merely suggested components). Any outcome which is established as an alternative to an important outcome is also important, even if that outcome does not itself interact with a top-priority player goal. When judging expectedness and valence of outcomes, **unimportant** outcome components are ignored (for a more detailed analysis using graded goal priorities, just factor goal priority into the expectedness and valence judgements as a weighting factor).

Given **important** outcome components, the results of an option can be said to be completely **predictable** if every **important** actual outcome component at that option was identified as a **likely** component by the likelihood analysis. Options where the **important** outcome components have a mix of **likely** and **neutral** likelihood fall somewhere between **predictable** and **unpredictable**, with options where all **important** components have **neutral** likelihood are fully **unpredictable**.

If *any* **important** outcome component was marked as **unlikely**, however, the results are **unexpected**, and the same is true if there is an **important** outcome component that was not a suggested component at all (and thus does not appear in the likelihood analysis). A simple analysis can use these three evaluations as exclusive labels, while a more nuanced analysis might represent the space between **predictable**, **unpredictable**, and **unexpected** outcomes as a continuous triangle as shown in the 'Per-Option Assessments' block of fig. 5.1 (in this case only the assessment for the first option is shown).

Besides predictability, outcomes have valences, which represent whether they are overall **good**, **bad**, or somewhere in-between. Shown in fig. 5.1 is a simple **good**/**bad**/**irrelevant**/**tradeoff** scheme, but finer distinctions can be made within each category (for example whether a tradeoff is generally worth it or not). Essentially, these evaluations summarize all goal impacts of the actual outcome components of an option, taking into account the relative strengths of each impact and the relative priorities of the associated goals. Although this summarization step may muddle things in more complicated cases, it can be helpful for quickly identifying the simple cases: often the results of each option are simply purely **good** or **bad** in terms of important player goals, and their impact on the player can be judged accordingly.

By assigning a single predictability value and valence to each option at a choice, it becomes much easier to get an overall picture of how the choice might look in retrospect given a particular decision. Along with the prospective impressions established by the relative option analysis step, these per-outcome assessments are used by the final retrospective analysis step to understand how the player perceives an option after they've made a decision.

5.4.9 Retrospective Analysis

Retrospective analysis is the final step in the goal-driven option analysis framework presented here, although the results of the relative option analysis step can

sometimes be used in isolation. Retrospective analysis complements relative option analysis by providing an understanding of how the player feels *after* making a decision and experiencing the outcome of a particular option at a choice. Note that further comparative analysis of outcomes would be necessary to understand how a player might view a choice after experiencing multiple outcomes, but this is unnecessary in the common case where a player only experiences a single outcome at each choice.

As with the relative option analysis, this step is focused on matching against a set of characteristic option structures. Note that each option is treated independently, although prospective impressions of the entire option set apply no matter which option is under scrutiny. Table 5.4 lists some retrospective impressions identified during the creation of *Dunyazad* and the formal criteria for each.

As with the prospective impression labels, retrospective impression labels are fairly narrow, but "near misses" can often be analyzed as a variant of a known case. For example, an option that almost fits the "bad gamble" profile but for which there *is* another option that seemed distinctly better has a slightly different feel from a normal bad gamble, but the difference isn't drastic (mainly, the player may place more blame for the failure on their own decision-making than on simple bad luck). A full picture of how player feels after a choice combines prospective and retrospective impressions, for example: An **obvious** choice at which the player chose the 'best' option which resulted in an **expected success**. Options other than the chosen one can still influence the overall impression, especially by accentuating bad results (when multiple options seemed promising) or good results (when all options seemed bad). Regret in particular is an example of a poetic effect that depends heavily on the nature of options-not-chosen (this is backed up by experimental data presented in chapter 8; see for example the discussion of regret in section 8.6.5).

Label	Description	Criteria
Expected Success	A predictable outcome that advances player goals.	The selected option was expected to **advance** a player goal without **hindering** any, and its outcome is both **predictable** and **good**.
Expected Failure	A predictable outcome that hinders player goals.	The selected option was expected to **hinder** a player goal and not **advance** any, and its outcome is **predictable** and **bad**.
Nice Gamble	An option that seemed like a gamble but that turned out well.	The selected option was not expected to **advance** or **hinder** any player goals, or it was expected to both **advance** and **hinder** goals that were approximately balanced. The outcome is **unpredictable**, but also **good**. The selected option was not distinctly worse-seeming than any other options.
Bad Gamble	An option that seemed like a gamble and did not pay off.	As above, but with a **bad** outcome.
Unfair	An option that seemed good but had unexpected negative consequences.	The selected option was expected to **advance** at least one top-priority player goal, while not **hindering** any. It has an **unexpected** and **bad** outcome.
Miracle	An option that seemed like a lost cause but that turned out well against expectations.	The selected option was expected to **hinder** at least one top-priority goal, while not **advancing** any. It has an **unexpected** and **good** outcome.

Table 5.4: Several retrospective impression labels, with formal criteria based on option expectations, prospective impressions, and option assessments. See section 8.11 for some caveats based on experimental results.

5.4.10 Analysis Example

To help understand the analysis technique presented here a bit better, it's useful to look at an example of its application. Consider the choice shown in fig. 5.3, from the game *Papers Please* (Pope, 2013). In *Papers Please*, the player takes on the role of a border security worker in charge of approving or rejecting entry applicants in a fictional dictatorship. Complicating the player's role is the fact that their job is their only means of earning money to support their family, whose food, shelter, and medical needs become more pressing as the game progresses. By processing applicants, the player earns credits (regardless of whether an application is approved or denied), so there is an incentive to make decisions quickly (and to avoid accepting travelers who don't meet the entry requirements, as this *will* result in penalties). The game focuses heavily on ethical gray areas, and the construction of its choices is central to this. Although the game has various elements that go beyond explicit choices, for each applicant the player

Figure 5.3: A screenshot from *Papers Please* showing the interface as the player makes a decision about whether to approve or deny a visa application.

must ultimately choose to either approve or reject their visa request, and these explicit choices can be analyzed using the framework developed here.

Figure 5.3 shows the screen as the player is considering an entry application from a potential refugee. The conversation transcript in the middle of the screen shows that this applicant claims that they will be killed if they return to their country of origin, and because of this, they seek entry despite lacking the required documents (an entry permit in this case). However, the tenor of the game and the presence of applicants who will try to trick the player in various ways creates an atmosphere of suspicion, and so the player might suspect that this applicant is lying in order to gain entry. As a closer analysis will show, this suspicion has interesting implications for the player's perception of this choice.

Goal Analysis

The first step of goal-based choice analysis is goal analysis, and in the absence of a real player to observe, a model player will have to suffice. For the purposes of this analysis, we will assume that our model player is engaged mainly in avatar play and has the following goals (with the listed priorities):

- `[high]` Provide for their family—the player wants to earn credits and avoid penalties in order to pay for food and shelter at the end of the round.
- `[high]` Act ethically—as much as possible, the player wants to treat applicants ethically and avoid acting in ways that would intentionally harm them without reason.
- `[medium]` Avoid being scammed—separate from their desire to earn credits, the player actively wants to identify applicants who are trying to trick them and deny their applications.

- `[low]` Admit approved travellers—in line with ethical treatment in life-or-death scenarios, the player generally wants to treat travellers fairly by letting them in if they meet the requirements for entry. This goal has a lower priority, however.

Of course not all players will have these exact goals with these exact priorities, but we can assume that many players will incorporate similar goals, and use these to develop an initial impression which can be refined further if deemed necessary. In particular, an initial analysis can be modified to account for differing goal priorities or the addition or subtraction of goals, and contrasting the resulting analysis can provide further information about a choice. As analysis moves forward using this list of goals, there will be opportunities to judge the results that it produces at each step and refine it if that seems necessary.

Likelihood Analysis

After analyzing goals, the next step is to analyze outcome components. To simplify the analysis, we will assume that there are only two options available to the player: "approve" and "deny" (in the game, other actions are possible, but but they all either simply defer the approve/deny choice or implicitly deny the application). Table 5.5 shows outcome components and their likelihoods for this choice; this assumes that the player is already familiar with basic gameplay in *Papers Please*. In particular, the player knows that if they let this traveller in, their 'mistake' will not go unnoticed, and they will both fail to earn a credit and receive a punishment for their transgression (for the first mistakes of a round the punishment is merely a warning, but the player still doesn't earn credit for the mistaken approval or denial). The player's doubts about the sincerity of the applicant are represented here as neutral likelihood estimates of four opposing outcomes. If the applicant is telling the truth, then they are a refugee, and they

Approve	Deny
[unlikely] Earn a credit	[likely] Earn a credit
[likely] Don't earn a credit	[unlikely] Don't earn a credit
[likely] Get punished	[unlikely] Get punished
[unlikely] No punishment	[likely] No punishment
[neutral] Refugee is saved	[neutral] Refugee is saved
[neutral] Refugee is condemned	[neutral] Refugee is condemned
[neutral] Scam is rewarded	[neutral] Scam is rewarded
[neutral] Scam is thwarted	[neutral] Scam is thwarted

Table 5.5: Likelihood analysis for the example choice shown in fig. 5.3.

are either saved by approval or condemned by denial. At the same time, if they are lying, they are trying to gain entry unlawfully, and thus approving their petition has different connotations.

This is an example of how uncertain outcomes can't always be mapped directly as outcome components. If "applicant is lying" and "applicant is telling the truth" were used as outcome components, these wouldn't have straightforward relationships to the player's goals, as they would be mediated by "applicant gains entry" and "applicant is denied entry" outcomes. Binding the truth status of the applicant's claims together with the end result for the applicant will simplify the goal analysis step as seen in the next section.

Option Analysis

Given the provide-for-family, act-ethically, avoid-scams, and admit-approved goals identified during goal analysis and the outcome components from likelihood analysis, the next step of goal-based choice analysis is to combine this information in an assessment of each option's perceived relationship to each player goal. Table 5.6 shows the results, using the **enables**, **threatens**, **advances**, and **hinders** labels described in section 5.4.5. Note that the provide-for-family goal exhibits a strong distinction between the two cases, based on the **likely** and **unlikely**

Goal	Approve	Deny
provide-for-family	**threatens** **hinders**	**enables** **advances**
act-ethically	**enables**	**enables** **threatens**
avoid-scams	**enables** **threatens**	**enables** **threatens**
admit-approved	<none>	<none>

Table 5.6: Option analysis for the example choice shown in fig. 5.3.

credit and punishment outcome components. In contrast, the act-ethically and avoid-scams goals have only weak expectations, because the associated outcome components are uncertain. The admit-approved goal is irrelevant here, because the applicant in question is not an approved traveller.

The assignments in table 5.6 are based on individual outcome-component/goal relationships: the credit- and credit-related outcome components are relevant to the provide-for-family goal, while the other four outcome components relate to the act-ethically and avoid-scams goals. As stated above, the breakdown of outcome components was chosen so that their relationships to player goals would be straightforward. This allows the likelihood labels to capture any uncertainty in the situation, as opposed to having unclear or conditional relationships between outcome components and goals.

Relative Option Analysis

The relative option analysis step looks for patterns in the results of option analysis, but in this case, the option analysis doesn't fit any established patterns (see table 5.3). In this case, known patterns can still be used as guideposts if we can figure out a known pattern that almost fits and analyze why it doesn't. For

this particular choice, it turns out that the **dilemma** pattern is a near match if the option analysis is a bit different.

In fact, the player's doubt of the applicant's story has a pivotal effect on their perception of this choice. Table 5.7 shows an alternative option analysis if the player assumes that the applicant is telling the truth. Such an alternate assumption changes the likelihood analysis (not shown), making the refugee-related outcomes **likely**/**unlikely**, and the scam-related outcomes universally **unlikely**. The results for the option analysis in table 5.7 show that the **advances**/**hinders** dichotomy of the provide-for-family goal is now reversed at the act-ethically goal. The resulting pairing of opposed **hinders** labels almost meets the requirements for the **dilemma** pattern, except that it also includes **enables**/**advances** assignments. In fact, the **dilemma** pattern criteria in table 5.3 is specific to a *classic* dilemma where neither option has redeeming qualities, but in this case we have a dilemma where the choice is between two conflicting goals and in either case the one that isn't **hindered** will be **advanced**.

Despite the caveats, we can see that if the player trusts the applicant they face a kind of dilemma. This means that the original choice is like a dilemma, except for the issue of doubt: if the player trusts the applicant, they face an ethical quandary—a decision between feeding their family and sheltering a refugee. However, if the player begins to doubt the applicant, the results of option analysis indicate that denying the application is clearly superior to approving it: denying the application **advances** a high-priority goal that approving it would **hinder**, and no other high-priority goals include such a clear distinction in terms of expected outcomes.

Looking at this result, we can see a broader message that *Papers Please* is trying to convey. In this choice and others through the game, would-be ethical dilemmas are tilted by doubt, and the overall formula becomes "dilemma + doubt $\rightarrow$ complicity"—by introducing doubt about the people who want to cross the

Goal	Approve	Deny
provide-for-family	**threatens** **hinders**	**enables** **advances**
act-ethically	**enables** **advances**	**threatens** **hinders**
avoid-scams	<none>	<none>
admit-approved	<none>	<none>

Table 5.7: An alternate option analysis for the example choice shown in fig. 5.3. This analysis assumes that the applicant is telling the truth about their situation (compare this table to table 5.6).

border, while making the risk/reward situation clear with respect to the player's prospects for feeding their family, the player's choices are tilted towards complicity with the regime's calculating but ultimately inhumane priorities. The callous response—leaving the potential refugee to their fate—becomes the rational one, and every would-be smuggler further justifies these decisions. Through its choice structures, *Papers Please* is showing the player what it feels like to be in this kind of situation and thus demonstrating why someone in that situation might make a particular set of decisions. The poetics of its choices, particularly when they give rise to feelings of moral uncertainty or doubt of the applicants, are an important part of this.

Outcome Component Analysis

Although the prospective analysis of our example choice has revealed something interesting about its structure, analyzing how the actual results are perceived is also important. For outcome component analysis, each *actual* outcome component should be analyzed with respect to each player goal. If desired, this can focus only on the outcome components of a single option to understand what players who choose that option will feel.

Focusing then on the option to *approve* the applicant's petition for entry, what are the actual outcome components? In *Papers Please*, whenever someone is let through who does not meet the technical requirements for entry the player accrues a citation, the first two of which in a given round result in warnings with the third and subsequent citations resulting in a fine equal to the amount of credits earned for processing one applicant. At the same time, letting in someone who should not be admitted does not earn credit towards one's salary (although interestingly, denying someone who deserves entry does, even when a citation is issued for an erroneous denial). In this specific case, even assuming that the player has not made any mistakes in the current round, a warning means that future mistakes are more risky, so accruing one has a minor negative effect on the player's provide-for-family goal. At the same time, the no-credit-earned outcome of an approval has a major negative impact on the player's provide-for-family goal: a denial would have earned credit for the time spent considering the application, and time is a very limited resource in the game.

Although these two outcome components (the citation and lack of credit) are straightforward, what about the outcome components related to either saving a refugee or catching a scammer? In *Papers Please*, after approving entry, one does not magically get to know whether the applicant was honest or not. In other words, even once a decision is made in this case, the player still doesn't find out whether they were justified or not. These outcome components remain uncertain: the applicant was let in, but was that a good thing or not? The player can only rely on their judgement of the sincerity of the applicant's claims in this case.

Although goal-based choice analysis has no explicit suggestions as to how to handle this case of post-decision lingering uncertainty (perhaps this could be a direction for future work), we can think about what impact it might have in this specific case. To handle the outcome component analysis step, it suffices to simply analyze both components separately and keep in mind that the player

doesn't know which of them occurs. Doing so results in the outcome component judgements shown in table 5.8.

Recall that our choice of outcome components made the option analysis step simple; in this case that has in turn made the outcome component analysis step complex. Because the example choice is complicated by doubts about the applicant's honesty, its structure is not easy to represent in terms of outcome components with direct relationships to player goals, but breaking it down into outcome components is still useful in order to identify where the complexity resides. An alternate analysis would simply use "applicant admitted" as an outcome component, and label its goal-impacts on both the act-ethically and avoid-scams goals as uncertain (labels which have not been considered in this chapter). The breakdown used here instead provides more detail by showing the evaluations corresponding to each potential player belief (in other words, by showing what possibilities the player is uncertain about).

	no-credit-earned	**citation-issued**	**refugee-sheltered***	**scam-rewarded***
provide-for-family	major negative	minor negative		
act-ethically			major positive	
avoid-scams				major negative
admit-approved				

Table 5.8: Outcome component analysis for the *approve* option at the example choice shown in fig. 5.3. Each entry lists the impact of an outcome component on a goal. The outcome components marked with asterisks are opposing potential components that the player is uncertain of even after making a decision.

Full Outcome Analysis

Based on the results of outcome component analysis, full outcome analysis summarizes each option as overall good/bad and expected/unexpected/unpredictable with respect to player goals. Focusing again on the approve option, we can see that its outcome is mixed but mostly negative. It has definite negative impacts on the high-priority goal of providing for the player's family, but it also has an uncertain positive impact on the high-priority goal of treating applicants ethically. Even if the uncertain impact on the avoid-scams goal is ignored because that goal has a lower priority than the two just mentioned, the positive impact on the act-ethically goal is less certain than the negative impacts on the provide-for-family goal, and so the overall impact can be summarized as a **tradeoff** with both good and bad components which leans towards the negative.

In terms of predictability, there's nothing surprising about this outcome (all of the *actual* outcome components were *expected*) so it falls into the **expected** corner, although we have to keep in mind the persistent uncertainty about whether the applicant was being truthful or not, which adds an element similar to unpredictability. As a whole then, the "approve" option is an expected tradeoff which leans negative, and which comes with a lingering uncertainty about its actual outcomes. The "deny" option sustains a mirrored analysis (omitted here for brevity), and appears as an expected tradeoff which leans positive, again with some uncertainty over actual outcomes.

Note that the psychological implications of these outcomes are interesting given the human tendency to rationalize one's decisions (see e.g., (Hall, Johansson, and Strandberg, 2012)). One might expect players who chose to approve this applicant's entry to believe more strongly after-the-fact that the applicant was telling the truth, and vice versa for those who chose to deny entry. This is one mechanism by which *making a decision* can actually influence players'

perceptions of the structure of a choice. This is also a mechanism which reinforces complicit behavior as mentioned above, and *Papers Please* is intentionally creating these choice structures in order to force the player to experience exactly this kind of fraught decision-making.

Retrospective Analysis

Having analyzed each outcome as a unit, the final step in goal-based choice analysis is to try to understand a particular choice in terms of existing choice-structure patterns. In this case, none of the retrospective impression labels in table 5.4 apply to either option at this choice, because they are all aimed at clearly-good or -bad outcomes, and the outcomes of both options here are tradeoffs which involve additional uncertainty. Even without matching against a known structure however, the breakdown of outcome evaluations has already been informative.

As mentioned above, *Papers Please*'s illustrates how uncertainty can lead to complicity when ethically-questionable decisions are seen as having uncertain results on one side and guaranteed results on another. In order for this to be clear, the player must experience both moral doubt and uncertainty, and these feelings arise from the structure of choices such as the one analyzed here. The uncertainty that the player experiences can be seen as having two parts which have been identified by this analysis: before making a decision, the player is uncertain about the applicant's honesty, and afterwards, this translates to uncertainty about the true outcome of their decision.

The player's initial uncertainty is due to a pattern of options and outcomes established across many choices by the game. As the player plays, they face many applicants who attempt to lie or bluff their way past the checkpoint despite having suspicious documents. Other events, such as a terrorist attack on the checkpoint, set a grim mood, and this pattern of events serves to make the player

generally suspicious (the interface and basic interaction loop of identifying and questioning discrepancies also contribute). Altogether, this suspicion leads the player to question the moral character of applicants in general, and in the specific choice discussed here this results in an uncertain likelihood analysis which views the applicant's refugee status as an unknown variable.

This suspicion-of-the-applicant then translates to a questioning of the player's decision once it has been made. Whether the player approves or denies the applicant's entry petition, concrete results (in terms of earning or not earning credits to help support their family) are contrasted with lingering doubts about whether the applicant deserved entry or not. *Papers Please* intentional leaves the true outcomes of the player's actions ambiguous with respect to their goals, creating lingering doubt which builds as the player continues to make questionable decisions. This creates a situation where the player can come to doubt their own judgements and second-guess themselves, which adds a second layer of uncertainty. Especially when confronted with similar choice structures repeatedly throughout the game, this second-guessing can help push the player to confront their decisions and thereby think about what message the game is trying to convey in this regard.

In this case, even though the elements of choice structure identified did not fit the pattern of any already-identified retrospective impression, identifying elements such as uncertainty and moral ambiguity was enough to gain insight into how this choice functions within *Papers Please*. Breaking things down also allows us to understand why the choice produces the effects that it does, and which specific elements of the choice are necessary for it to function as intended. For example, the analysis presented here makes it clear that this choice functions as it does because it brings two ethical imperatives into conflict (supporting those who depend on you and avoiding harm to innocents) but then has uncertain outcomes in terms of one of them (avoiding harm). Altering one of these elements

(for example, making it so that earning credits is random, and thus the reward in terms of supporting one's family is also ambiguous) would change the character of this choice.

5.5 Overview & Future Work

As just demonstrated, goal-based choice analysis breaks down a choice into individual options and components, analyzes them with respect to a specific set of player goals, and by doing so helps present a clear picture of how the structure of a choice contributes to its poetic function[5] within a work. The analysis method presented here is heavily dependent on identifying well-understood prospective and retrospective impressions and good criteria for their application. However, this chapter only presents a few of these impressions, and it does not contain a comprehensive analysis of the poetic effects that they can create. In other words, the theory of choice poetics described here is still preliminary. However, this goal-based choice analysis is a strong framework for understanding nuanced choices, because decomposing options and outcome components can help untangle complex choice structures. Additionally, by separating player goal analysis from the main choice analysis and taking modes of engagement into account, this method makes the dependencies between desired impressions and particular player goals visible to designers. Without such visibility, the confusion of players faced by a choice that pits two modes of engagement against each other might be mysterious to a designer who favors one of the modes and doesn't realize that the other is relevant. Goal-based choice analysis could thus be helpful for debugging key choice structures that aren't producing a desired impression, and for figuring out whether a game supports specific modes of engagement.

[5] As demonstrated such analysis may also be useful for understanding a choice's rhetorical and/or hermeneutic functions.

The few impressions that have been presented here have also been verified to some degree by the experiments presented in chapters 7 and 8. In fact, the results of those experiments did not unanimously confirm my hypotheses, which has led to refinements in the analysis technique presented here. In particular, tables 5.3 and 5.4 are revisited in tables 8.20 and 8.21 in section 8.11. These revisions and others that were made as I built *Dunyazad* are the primary product of my hybrid research method: a more nuanced understanding of choice poetics obtained by attempting to operationalize it.

One might wonder what advantage this analysis technique has over similar theories that could be applied to narrative choices, such as decision affect theory[6](Mellers, A. Schwartz, Ho, et al., 1997). One difference lies in specificity and directness: while decision affect theory could be consulted to come up with a set of hypotheses about audience response to a choice, it does not provide a list of specific steps to follow in doing so. Another difference is the aims of the theories: decision affect theory has been shown to have explanatory and predictive power: it can be used to explain observed reactions to choices, and it can be used to predict reactions given a choice. In contrast, goal-based choice analysis as presented here was designed to have *generative* power: it can be used to *construct* a choice that produces a specific reaction. Although further studies could hopefully demonstrate that goal-based choice analysis has explanatory and predictive power as well, the results presented in chapters 7 and 8 are a validation of generative power. Of course, if one is interested in generating rather than evaluating choices, this evidence is encouraging.

The next step in developing goal-based choice analysis would be to greatly expand the number of prospective and retrospective impressions scrutinized. An efficient approach would be to take a well-known interactive experience built

[6]Complicating such a comparison is the fact that the analysis technique presented here is in part based on decision affect theory.

around explicit discrete narrative choices and analyze it from the beginning, breaking down each choice within this framework and looking for patterns in their usage. *Dunyazad* could then be used to double-check and refine the insights generated by this process: take each impression identified during the initial analysis, encode it in *Dunyazad*, and ask *Dunyazad* to generate choices that involve that impression. Some of the choices generated will likely not appear to give that impression upon initial inspection, and *Dunyazad*'s full output can be used to figure out exactly how the criteria that were developed technically fit the generated choice. With this information, refinements to *Dunyazad* and/or the underlying theory can be made, eventually producing both an analytical and a generative model of the impression in question, and completing the analysis $\rightarrow$ theory $\rightarrow$ generative model $\rightarrow$ experimentation cycle.

The next chapter describes in detail how *Dunyazad* operates, and part of it mirrors this one: an automated version of the analysis technique described here forms the core of *Dunyazad*'s reasoning capabilities. After that, chapters 7 and 8 discuss the results of two experiments that showed outputs from *Dunyazad* to humans and gathered feedback about their impressions. Chapter 8 includes some updates to the material presented here (mainly tables 5.3 and 5.4; see section 8.11), as these refinements make the most sense when presented alongside the experimental results that led to them.

Chapter 6

Dunyazad

As stated in the introduction, *Dunyazad* has two purposes as a project: first, to produce a *system*, and second, to produce a *theory*. The system is a novel story generator that is focused on generating interesting choices, and on providing fine-grained control over the kinds of choices it uses. The theory is a theory of choice poetics: a theory about how specific choice structures give rise to certain feelings, such as regret. In order to satisfy these two goals, *Dunyazad* uses an answer set solver to turn a logical criteria for a certain kind of choice into an instance of such a choice—this is its core operating principle.

Dunyazad's code therefore largely consists of a set of logical statements describing what a choice structure is and when choices fit into one of several categories. For example, there are statements that define a "relaxed" choice, and these rely on statements about what it means for an outcome to seem good or bad, and important or unimportant, and so on. Answer sets that satisfy these constraints are found using Potassco Labs' `clingo` solver (Gebser et al., 2011). Besides these rules, there is an external control structure which invokes the

Note: a much less detailed description of how *Dunyazad* works has been published as (Mawhorter, Mateas, and Wardrip-Fruin, 2015b).

solver repeatedly, generating multiple choices and connecting them into a story.[1] The focus of development so far has been on individual choices, but the capacity exists for *Dunyazad* to generate a full branching story. Of course, there is also some imperative code for turning a logical choice structure into English text, in the form of a template-based generation system. The English-generation code and the control code are both written in Python.

The goal of this chapter is to describe in detail how *Dunyazad* functions. First, section 6.1 describes why *Dunyazad* uses answer set programming in a bit more detail, and how answer set programming (ASP) helps with the theory development goal. Section 6.2 then describes how *Dunyazad* generates individual choices, while section 6.3 describes the higher-level control mechanism. Finally, section 6.4 summarizes the entire system and works through an example of how *Dunyazad* generates a choice from beginning to end. Note that the template-based English generation system is described separately in appendix A.

6.1 ASP and Critical Technical Practice

The idea of critical technical practice started as a call for technical practitioners (and especially AI researchers) to be more aware of the limits placed on their technical approaches by the central metaphors of their fields. In Phil Agre's 1997 *Computation and Human Experience* he describes a process of critical examination to identify core metaphors and what those core metaphors marginalize, followed by an inversion that makes those marginalized concepts central (Agre,

[1]The model for *Dunyazad*'s output is the Choose-Your-Own-Adventure book. These books are young-adult novels which contain explicit choices: at the end of certain pages, the reader is instructed to choose an action for the protagonist, and continue reading at one of several different pages depending on their choice. This combination of traditional linear narrative with intermittent discrete choices is one of the most straightforward examples of what I call "choice-based narrative." Note that in contrast to Choose-Your-Own-Adventure books, choices in *Dunyazad* are separated by only a few sentences, rather than pages of text.

1997). This process can be used to drive innovation and find hidden limitations in a technical field by scrutinizing it using the tools of critical theory.

One common AI practice identifies a motivating theory from another discipline and then attempts to build a system that faithfully operationalizes that theory, taking the theory for granted as a known truth. In contrast, critical technical practice seeks to question common assumptions (in particular through the application of critical theory) and then embark upon technical projects that use non-standard assumptions, thereby broadening the technical field. *Dunyazad* as a project pursues a different relationship between theory and practice by seeing its driving theory (choice poetics) as both a *product* of technical practice and a precursor to it, integrating theory development tightly with system development so that both the theory and the system are developed in dialogue with each other. *Dunyazad* thus questions its theoretical assumptions as a matter of course and in fact uses difficulties encountered during technical development to prompt changes in the theory of choice poetics. In fact, *Dunyazad* can be seen as using artificial intelligence as a tool of criticism in the refinement of a theory, just as much as it uses theory to guide the development of an artificial intelligence.

In order to jointly develop *Dunyazad* as a system and choice poetics as a theory, a special technical approach is required—this is where answer set programming comes in. One could imagine choice-point generators constructed using many different technologies, from case-based reasoning to something as complicated as neural networks. However, most of these would fail to achieve the level of *transparency* required for the system to provide useful feedback to the underlying theory. In a neural-network based approach, for example, it would be very difficult to understand which part of the underlying theory is responsible for a particular generated choice structure, and even more difficult to use experimental results to inform changes in the theory. Answer-set programming as a technical approach

is thus motivated by the goal of developing choice poetics in tandem with a technical system for generating choices.

The reason that answer-set programming is useful for critical technical practice is that answer set programs consist of a series of logical statements which dictate which combinations of predicates are allowed in the output. This means that every predicate in the output set can be traced back to a set of rules in the program which allowed it to be present. Sometimes this process is laborious, as many rules may chain together to allow a predicate, but ultimately, every aspect of an answer-set program's output can be analyzed to figure out which specific statements were responsible for its presence. Furthermore, because those logical statements correspond directly to elements of the underlying theory, the relationship between that theory and the program's output is visible.

As an example, *Dunyazad* has rules that define one way in which a choice can be obvious—by having exactly one option which suggests a positive outcome and one or more options all of which suggest negative outcomes. Originally, the criterion that the other options must suggest negative outcomes was not present, but upon looking at generated choices, some clearly didn't fit an intuitive test for obviousness because of their inclusion of neutral options. Identifying the inclusion of these neutral options as the problem was easy, and it was also clear from inspection of the code that without a clause in the definition of "obvious" prohibiting neutral options, they would be allowed. The fix at the system level was the extra clause requiring that all options at an obvious choice except the "correct" option should suggest negative outcomes. Of course, this was not only a change in the code, it was also a change in the theory, because *Dunyazad*'s code is a direct operationalization of choice poetics. Obvious choices don't just have "only one good option," instead they have "one option which stands out from the rest as significantly better."

In contrast with many techniques that could be applied to the problem of generating choice points, answer-set programming uniquely enables a tight integration of system development with theory development. The use of answer-set programming as a design choice is thus driven not only by considerations at the system level, but also at the project level: in order for *Dunyazad* to produce both a system and a theory, the use of answer-set programming is critical. In *Dunyazad*, answer-set programming helps support the aims of critical technical practice by helping make the system's underlying assumptions clearly visible, and by making every change to the system suggest a parallel change to the theory that it implements.

6.2 Choice Generation

Dunyazad generates choices by solving complicated logic problems. These problems are set up so that any solution will take the form of a choice, and further constraints can dictate certain properties of that choice (for example, that it must be an obvious choice). Broadly, *Dunyazad*'s constraints can be split into four categories:

- **Representational constraints** establish the basic elements that *Dunyazad* arranges to form choices and stories.
- **Constituent constraints** define the most basic rules for configuring representative elements to construct choices and stories.
- **Aesthetic constraints** carve out a region within the space allowed by the constituent constraints, excluding gibberish and other categorically undesirable configurations.
- **Poetic constraints** make distinctions between different desirable configurations and are selectively activated to produce different kinds of choices.

To understand how *Dunyazad* functions, it is first necessary to understand *Dunyazad*'s (rather limited) view of what a choice is, and how choices fit together to form stories. These things are defined by *Dunyazad*'s representational and constituent constraints.

6.2.1 Story Representation

In *Dunyazad*, a story is represented as a sequence of actions, each drawn from a pre-determined set. Besides actions, each timepoint in a story has an initial state, which can include characters, items, and relations between them. *Dunyazad* also has "setups," which are used to establish interesting situations that motivate action. This kind of representation is amenable to logical reasoning and similar to representations used in planning-based story generation systems. It is sufficient to represent a wide variety of simple stories focused on action (as opposed to say, character growth or the development of relationships). Luckily, many of the original Choose-Your-Own-Adventure books are focused on actions and consequences, and the format of choice-based narrative lends itself to such stories, because they give rise to interesting choices.

States

Dunyazad's representational constraints define predicates for "instances," which may be either "actors" or "items," as well as "states" (binary properties of an instance), "properties" (instance properties which can be multi-valued), and "relations," (named directional instance-to-instance links). Besides these things, which can change from one state to the next as a result of the outcomes of actions, instances can have timeless "surface properties" like a name, which is stored but cannot be changed by events in the story. Figure 6.1 gives examples of each of these and fig. 6.2 shows how they come together to describe a situation.

Core states in *Dunyazad* describe characters and any items or skills that they possess. Beyond these are states describing transient relationships that set the stage for action; these are called "potentials." Potentials include things like "`injured`," and "`threatening`," and each is classified as either a "problem" or an "opportunity." Although these special potential states are represented using normal "`state`," "`property`," and "`relation`" predicates, actions which get rid of them must specify whether each potential is "`resolved`," "`manifested`," or "`nullified`." Additionally, extra predicates specify whether a potential is "`problematic_for`" one or more of the characters involved (for example, the "`injured`" state is "`problematic_for`" any character to which it applies).

Figure 6.1: Predicates used to describe states in *Dunyazad*.

Dunyazad understands based on this information whether the action which eliminated a potential was good or bad for different characters. For example, an action that "`resolves`" an "`injured`" state is good for the actor who was injured, and thus taking that action makes sense for that character. States themselves thus directly encode some of the information used to decide which actions are viable. This not only helps the system reason about motivation, but it also makes the design of actions and setups easier by providing a standard set of states that trigger certain actions.

Scene:

A merchant carrying some perfume is being threatened by bandits.

Representation:

```
inst(actor,businessperson_4).
inst(actor,tough_3).
inst(item,treasure_5).
property(type,inst(actor,businessperson_4),merchant).
property(type,inst(actor,tough_3),bandits).
property(type,inst(item,treasure_5),perfume).
relation(
  has_item,
  inst(actor,businessperson_4),
  inst(item,treasure_5)
).
relation(
  threatening,
  inst(actor,tough_3),
  inst(actor,businessperson_4)
).
surface_property(name,inst(item,treasure_5),"perfume").
```

Figure 6.2: An example scene description using *Dunyazad*'s internal representation, showing the most-relevant predicates. In real output, each of these predicates except the "`surface_property`" would be tied to a specific timepoint, and there would be many more "`surface_property`" predicates describing things like the name of each instance and whether it is plural or singular. Note that each instance is assigned a unique identifier that ends with a number.

Figure 6.2 is not quite accurate, because *Dunyazad*'s non-surface state predicates are never encountered alone. Instead, they are always wrapped in "st(<T>, <state>)" predicates, to define different states of the world for each timepoint T in the story. The timepoints in a story form a directed acyclic graph that branches out from a single root node. This graph is defined using "successor(<previous>, <option>, <next>)" predicates which specify that the result of choosing option "<opt>" at timepoint "<previous>" is the state designated "<next>." The root timepoint is named "root," and each successor is named for its parent plus the number of the option that leads to it, so for example, if there were three options at the "root_2" node, they would lead to timepoints labelled "root_2_1," "root_2_2," and "root_2_3." The timepoints do not necessarily form a tree, however: if two outcomes lead to identical states, as long as it would not form a cycle in the graph, *Dunyazad* may connect them to the same timepoint.

Actions

As mentioned above, timepoints in *Dunyazad* are connected by options. Each option is associated with a single action, and arguments describe the details of that action (for example, who initiated it or who the target is). If a timepoint has more than one option, it represents a choice to be made by the player, otherwise it is simply an event that happens. Actions specify lists of outcome variables with values for each variable. For each outcome variable that an action has, a single outcome value is assigned to each option that uses that action, thereby defining the entire impact of that option on the world state. These assignments have a shorthand notation "o(<variable>, <value>)" which indicates that the outcome variable "<variable>" takes on the value "<value>." Each such assignment is an outcome component, and together, all such assignments for an action are the outcome of that action. Figure 6.3 shows an example where at timepoint "root," the action associated with option 1 is "talk_down," with two outcomes:

"o(attitude, convinced)" and "o(is_enraged, not_enraged)". Note that each outcome variable is responsible for deciding whether or not a particular state change occurs (or between several possible state changes one of which must occur). The consequences of an action in terms of the world state are thus fully determined by the values of its outcome variables.

Each action thus has multiple possible outcomes. For the "talk_down" action, there are theoretically four: the "attitude" outcome variable has two exclusive values "convinced" and "unconvinced," while the "is_enraged" outcome variable has two more values: "enraged" and "not_enraged." However, the definition of "talk_down" stipulates that the outcome "o(is_enraged, enraged)" is only possible if the outcome "o(attitude, unconvinced)" is also present, so there are only three possibilities:

1. The target is persuaded to calm down ("o(attitude, convinced)" and "o(is_enraged, not_enraged)"), in which case they stop threatening whoever they were threatening.

2. The target cannot be persuaded to calm down, so they continue threatening their target ("o(attitude, unconvinced)" along with "o(is_enraged, not_enraged)").

3. The target is unconvinced, and furthermore gets mad at the person who tried to convince them ("o(attitude, unconvinced)" along with "o(is_enraged, enraged)").

Note that each action could be broken into several sub-actions with fixed pre- and post-conditions, with the current actions each becoming a mix of several primitives depending on their outcome variable values. The advantage of grouping primitives together conceptually is that this represents the player's view of things: the text introducing an option doesn't indicate the values of its

Option:

You try to talk the bandits down.

Outcome:

You talk to the bandits and convince them to back off.

Representation:

```
at(root, action(option(1), talk_down)).
at(root, arg(option(1), asking, inst(actor, you))).
at(root, arg(option(1), listening, inst(actor, tough_3))).
at(root, arg(option(1), victim,
  inst(actor, businessperson_4))).

at(root, outcome(option(1), o(attitude, convinced))).
at(root, outcome(option(1), o(is_enraged, not_enraged))).

at(root, consequence_of(option(1), o(attitude, convinced),
  _not, relation(threatening,
    inst(actor, tough_3), inst(actor, businessperson_4)))).
at(root, consequence_of(option(1), o(is_enraged, enraged),
  relation(threatening,
    inst(actor, tough_3), inst(actor, you)))).

at(root, consequence(option(1),
  _not, relation(threatening,
    inst(actor, tough_3), inst(actor, businessperson_4)))).
```

Figure 6.3: An example of action representation in *Dunyazad*. Note that even outcomes which do not occur ("`o(is_enraged, enraged)`" in this case) have their potential consequences noted via "`consequence_of`" predicates (so that the system can reason about counterfactuals), but only outcomes that do occur (as specified by the "`at(<T>, outcome(option(<opt>), o(<outvar>, <outval>)))`" predicates) have actual consequences. The "`o(attitude, unconvinced)`" and "`o(is_enraged, not_enraged)`" possible outcome components are not listed because they make no changes to the world state when they occur (they each represent the omission of a change).

outcome variables, and so the player can never be completely sure what outcomes an action will have. This way of representing actions thus helps the system reason about how the player might perceive actions; this is the same reason that possible but unrealized state changes are explicitly represented by the system using "consequence_of" predicates (see fig. 6.3).

accuse	explain_innocence	play_song	talk_down
arrive	flee	polymorph	tell_story
attack	gossip	pursue	trade
buy_healing	leave	reach_destination	travel_onwards
deny_blame	pacify	shift_blame	treat_injury
dispel	pay_off	steal	

Table 6.1: The 23 actions currently defined in *Dunyazad*. The "arrive," "leave," and "pursue" actions aren't currently used because the system has no setups that motivate them. Appendix B.4 contains listings of the source files that define each action.

Besides defining outcomes, actions also define how skills affect their outcomes, by asserting that certain outcome values are linked to the presence or absence of particular skills on the part of one or more participants of the action. These links are denoted by "skill_link" predicates, and tools may also be involved. For example the "attack" action has an outcome variable "success" with three values: "victory," "defeat," and "tie." It specifies that the outcomes "o(success, victory)" and "o(success, defeat)" are both linked to the "fighting" skill as alternate outcomes of a skill contest between the "aggressor" and "target" of the action, where tools are advantageous. This means that if the aggressor of an attack action has both the "fighting" skill and a tool for that skill, while the target of the attack has neither (or even if they're just missing a tool) the "o(success, victory)" outcome is more likely, and the "o(success, defeat)"

outcome is less likely (the exact consequences of these assignments will be discussed in section 6.2.4). Effectively, the definition of an action thus includes enough information for *Dunyazad* to consider both possible consequences of an action and what initial states might best foreshadow each different outcome.

Dunyazad currently has a total of 23 different actions, listed in table 6.1. Given the setups that exist, however, the "`arrive`," "`leave`," and "`pursue`" actions are never motivated and thus impossible to use, so there are effectively 20 possible actions. The actual definitions for each action are listed in appendix B.4. Most actions have at least two possible outcome configurations, and a few (such as "`attack`") have four or more. However, the number of actions that are possible in a given state are limited by the constituent and aesthetic constraints. This is where the setups come in: most actions require some sort of motivating state to make sense (the "`talk_down`" action, for example, requires that either a "`threatening`" or an "`accusing`" relationship be present). Setups represent existing situations that the player might encounter while travelling, and each sets up conditions for a particular subset of actions.

Setups

Besides the initial state of the world having to do with the protagonist(s) and their starting skills and equipment, most of the state in *Dunyazad* comes from setups. The situations *Dunyazad* creates are designed to fit into an overarching travel narrative: the protagonist(s) are on a journey, and encounter various obstacles. This provides an excuse to repeatedly clean up the world state by having the main character(s) "travel onwards," getting rid of any states except those pertaining to the main character(s) and introducing a new setup. When putting multiple timepoints together into a complete branching story, *Dunyazad* starts by introducing a setup and then adding a few timepoints which resolve any outstanding potentials in that setup. It then adds a "`travel_onwards`" option to

each branch where potentials have been resolved, and adds a new setup to the timepoint that follows this option.

Setups are thus expected to set the stage for action: they provide a situation that contains the potential for something interesting to occur. Some setups are quite flexible while others are specific. For example, the "`healer`" setup simply adds an actor with the "`healing`" skill and a tool for healing who is offering to treat injuries for a price; it may only be used when a protagonist is injured. In contrast, the "`market`" setup potentially includes a lowlife, a healer, a laborer, an aristocrat, and up to two merchants. These characters can have several potentials between them: the lowlife can be threatening one of the merchants, the noble might be accusing the peasant or one of the merchants, or perhaps the merchants are simply selling things and the noble or laborer knows some gossip. The wide range of possibilities allowed by the "`market`" setup lets *Dunyazad* create a variety of situations in service of creating particular choice structures, whereas the "`healer`" setup is designed to be deployed in a very particular circumstance. Although the "`market`" setup eclipses the "`healer`" setup in terms of actions enabled, the "`healer`" setup's lack of extraneous characters provides a very different feeling to the player, and its surface text is more specific.

As mentioned above, not every timepoint includes state changes that are induced by a setup: they only occur for the "`root`" node and for timepoints that follow "`travel_onwards`" actions. Internally, *Dunyazad* refers to everything that occurs between one setup and the "`travel_onwards`" events which follow that setup on each branch from it as a vignette. The states associated with a setup are introduced after the "`travel_onwards`" action first removes all states associated with the old location—only states directly associated with a protagonist (such as an injury sustained or an item acquired) are retained. Because *Dunyazad* effectively solves entire timepoints at once, however, the particular configuration of a setup's flexible elements can be changed at the same time that the actions,

arguments, and outcomes of options are being decided. This means that *Dunyazad* has the freedom to consider all possible setups as well as all configurations of actions given those setups when trying to build a particular choice structure, unless a certain setup is mandated by outside restrictions.

6.2.2 Constituent Constraints

Dunyazad's representation of stories as a graph of timepoints leaves lots of room for structures that don't make any sense. This is where the constituent constraints come in. While the basic predicates that define representational elements ensure things like continuity of states between timepoints, constituent constraints help ensure that actions get assembled into a story, instead of just a random sequence of unrelated events. For example, rules that prohibit the same action from happening twice in a row, or that require that each action be motivated, are constituent constraints.

While there is sometimes a gray area between constituent and aesthetic constraints, constituent constraints can often be identified as being concerned with getting *Dunyazad* to produce stories at all, while aesthetic constraints help defined *Dunyazad*'s particular flavor of story. If constituent constraints are removed, the results are often nonsensical; if aesthetic constraints are removed results still make sense, but they no longer fit with other stories that *Dunyazad* constructs. Table 6.2 lists all of the source code files from *Dunyazad*'s main answer set problem definition code and briefly indicates what rules each file contains from each constraint type.

The major topics covered by the constituent constraints are as follows:

- Incapacitation—constraints that prevent injured actors from taking vigorous actions like fighting, and prevent dead actors from doing anything.

`actions.lp`
[representational] Action representation; consequences.
[constituent] Incapacitation.
[poetic] 'Surprising' outcomes based on likely/unlikely outcomes.

`actors.lp`
[representatonal] Unpacking for actors; top level of actors ontology.

`choice_structure.lp`
[constituent] Motivation; relevance; redundancy; repetition.
[aesthetic] Setup variety; bans boredom and trick options; narrative perspective (second-person).
[poetic] Story length and pacing.

`core.lp`
[representational] Basic structure (options, events, choices, etc.); exclusivity for states; reflexivity for actions; ontology basics (inheritance).

`eval.lp`
[poetic] Expectations; stakes; option feels; option structures; outcome perceptions; outcome predictabilities; outcome feels.

`goals.lp`
[poetic] Player goals; guilt.

`grow.lp`
[representational] Timepoint ordering and links; timepoint creation and state transfer; state matching.

`items.lp`
[representational] Unpacking code for items; tool possession for skills; communal ownership for trading.

`potential.lp`
[representational] Resolution methods, initiators, urgency/immediacy, and importance of potentials; unresolved and hidden potentials.

`settings.lp`
[content] The possible settings.
[representational] Setting assignment; setting continuity.

`setup.lp`
[representational] Unpacking for setups; setup state creation.

`skills.lp`
[content] The list of skills.
[representational] Relevance definition for skills and tools.
[constituent] Outcome likelihoods.

`surface.lp`
[representational] Surface properties like names and genders.

`the_party.lp`
[content] Rules that define the starting state of the protagonist(s).

`utils.lp`
[other] Helpers; name assignment.

`vignettes.lp`
[constituent] Scoping of vignettes (actions in a single location).

`content/*`
[content] These files define the actions, goals, potentials, and setups that *Dunyazad* can use, along with its ontology of actors and items.

Table 6.2: An inventory of *Dunyazad*'s constraints organized by file and by constraint type. Code listings can be found in appendix B.

- Motivation—these constraints require that the actor that initiates each action have a motivation for taking that action.
- Relevance—constraints which require that actions address an existing important potential. These get rid of situations where someone fighting for their life might stop to buy fruit, for example.
- Redundancy—these constraints prevent two options at the same choice from being redundant.
- Repetition—constraints preventing direct repetition: repeated actions at consecutive timepoints and repetition of a setup in consecutive vignettes.
- Outcome likelihoods—these constraints describe how skills and tools make certain outcomes more or less likely. Without them, outcomes would have no relation to the skills and tools of the participating actors.
- Vignette scoping—constraints that determine when "`travel_onwards`" actions are appropriate and thus where new setups should be applied.

Along with the representational constraints discussed above, these constituent constraints force answer sets to represent something that resembles a story, as opposed to a random mish-mash of incomprehensible actions. The aesthetic constraints take things one step further and isolate a particular kind of story that *Dunyazad* tries to create.

6.2.3 Aesthetic Constraints

Dunyazad's aesthetic constraints go beyond the basics required to produce actions that make sense and can be read as a story. Effectively, aesthetic constraints define what kinds of stories *Dunyazad* will produce. As shown in table 6.2 there are several different categories:

- Narrative perspective—these constraints require that all actions at choice points be initiated by the protagonist(s) and that all other actions are initiated by non-protagonists, effectively establishing a loose second-person narrative perspective.

- Setup variety—These constraints force *Dunyazad* to use a variety of setups over the course of a story, rather than just revisiting a few. This goes beyond prohibiting direct repetition and establishes a "breadth" requirement: before you can repeat a setup for the nth time, you must have used a total of at least $n + 1$ unique setups (including the setup in question). Given the limited number of setups this equation does currently place an upper bound on the length of stories *Dunyazad* can generate, but until more work is done on long-term plot structures, this limit is largely theoretical.

- Boredom—Beyond the repetition constituent constraints, these constraints target things like repeated failed attempts to solve the same problem, or the recurrence of a problem within a single scene. Although not something that would seem obviously broken to readers, these possibilities were deemed less interesting and prohibited.

- Trick options—If there are multiple protagonists and one has the skill necessary to perform a certain action while another lacks it, giving the player the option of having the unskilled protagonist perform that action is a kind of trick and this can feel very bad for the player as they're essentially not given the most rational option (which would be to have the skilled party member perform the action) but at the same time reminded of its presence. These constraints simply rule out this possibility, even though one could imagine using it for dramatic effect.

These aesthetic constraints are admittedly a bit underdeveloped. In particular, there are not enough related to ensuring that long-term actions form a

coherent plot, and this is one reason[2] that the experiments discussed in chapters 7 and 8 focus on single choices rather than entire stories. This is an open area of future work for *Dunyazad*: developing stronger plot structures across multiple actions would allow it to produce interesting short stories and thus to support more complicated experiments.

6.2.4 Poetic Constraints

Dunyazad's poetic constraints are mechanisms for authorial control. Together, they establish a set of predicates which can be required or forbidden in order to drive the creation of different kinds of choices. These constraints are thus not about declaring what is universally acceptable or unacceptable, but rather about distinguishing between multiple interesting possibilities and providing the author the ability to make choices between these at a high level. The bulk of these constraints form *Dunyazad*'s operationalization of choice poetics, but a few are concerned with other effects:

- Surprising outcomes—Using the notion of likely and unlikely outcomes based on skills and tools, *Dunyazad* labels outcomes which are deemed unlikely as 'surprising.' In an earlier version, 'surprising' outcomes were prohibited; now it's up to the author whether or not to allow them.
- Story length and pacing—*Dunyazad* has rules for constraining how many actions must occur before a story ends, and how many of these must be choices as opposed to simple events.

Besides these two categories of constraint, *Dunyazad*'s poetic constraints are focused on choice poetics, and in particular, implementing the goal-based choice analysis method described in section 5.4 and illustrated in fig. 5.1. These constraints are the focus of the experiments described in chapters 7 and 8, and

[2] A more important reason was the elimination of confounding factors.

they have also been the focus of development on *Dunyazad* so far: these are the constraints that allow *Dunyazad* to attempt to create specific poetic effects when constructing choices. Although the details of goal-based choice analysis were the main subject of chapter 5, the following section provides a walk-through of *Dunyazad*'s technical implementation of the technique.

Choice Poetics

Dunyazad's choice poetics constraints rely on a few underlying properties of its stories that were just discussed, most notably the concept of likely and unlikely outcomes based on skills and tools. In *Dunyazad*'s code it is assumed that players will be roughly aware of these things, and the surface text code does its best to ensure this. If this assumption is violated, *Dunyazad*'s choices may have unintended poetic effects, just as when an author fails to clearly communicate and leaves some of their audience confused. Results so far seem to indicate that this assumption is mostly correct, however, so *Dunyazad*'s conflation of its internal likelihoods with players' perceptions of such seems to be acceptable.

It is no accident that the breakdown of choice structures described in section 5.4.2 corresponds exactly to *Dunyazad*'s internal representation of a choice as having an initial state, several options, and several outcomes at each option. When a choice occurs directly after a setup (which always happens when *Dunyazad* generates a single choice) the setup becomes the framing. The options are each a single action, described to the player before a choice is made without mention of any outcomes. Once the player chooses, they see the outcome for the option they selected, which is a description of each outcome component (an "`o(<variable>, <value>)`" pair) of that action. The idea of potential outcomes also has a direct analogue: these are the other possible values of outcome variables for an action.

Each of the seven steps of choice analysis described in section 5.4 has a corresponding set of rules in *Dunyazad* which are defined using the rules for earlier steps:

1. **Goal Analysis**

   This step is quite simple in *Dunyazad*: the file `goals.lp` includes rules that directly state the player's goals in every situation (a listing can be found in appendix B.2). Of course, forcing the author to estimate the player's goals like this is not perfect, but results so far indicate that it works tolerably well. Individual files in the `content` directory specify how each goal works and provide rules for determining the priority of a goal in a given situation (this is binary, either 'high' or 'low'). These files also define at least one state that is "`good_for`," "`bad_for`," "`great_for`," or "`awful_for`" the goal. For example, the "`preserve_health`" goal always has high priority and specifies that the states "`injured`" and "`killed`" both "`fail`" the goal when true of the goal's target. `goals.lp` then specifies that at every timepoint, the player has a "`preserve_health`" goal for each protagonist, allowing *Dunyazad* to understand that an action which might have an outcome that kills a protagonist will be viewed by the player as carrying some risk. The full list of player goals assumed by the system is as follows:

   - The player always has a "`preserve_health`" goal for each protagonist. As mentioned above, these goals are always high-priority and are failed by the "`injured`" and "`killed`" states.
   - The player always has an "`avoid_threats_to`" goal for each protagonist. Actions which cause "`threatening`" relationships with a protagonist as the object "`hinder`" these goals, as do actions which allow such a state to persist (i.e., actions which don't have a consequence that resolves such a state when one exists). These goals are always high-priority.

- The player always has an "avoid_accusations" goal for each protagonist. These goals work just like "avoid_threats_to" goals but for the "accusing" relationship, which indicates that someone is being accused of a crime. As with "avoid_threats_to", these goals are high-priority.
- The player always has a "preserve_original_form" goal for each protagonist. The "polymorph" action represents a magic spell that can turn someone into a chicken; these goals assume that this is a state to be avoided and/or reversed whenever possible. These goals are high-priority.
- Whenever an item has been stolen from a protagonist, the player has a "reclaim_property" goal for that item/protagonist. This goal is achieved when the "has_item" relation holds between the original owner and the item, and it it always high-priority.
- The player always has an "as_intended" goal for each protagonist. These goals make use of the "default_intent" predicates specified in the definition of each action. Essentially, each action declares one or more outcome components to be its default intent: the outcome that the initiator of the action is hoping for. In the case of the "attack" action, for example, the outcome component "o(success, victory)" is a "default_intent". When a "default_intent" outcome component is present, the "as_intended" goal of the action's initiator is achieved, when absent, the goal is failed. These goals are always low-priority, because they don't represent any particular bad or good states: there are many situations where an unintended outcome can be good for a character, and actual good or bad results are more important than things going as intended.
- The player always has a "have_tool_for" goal for each skill of each protagonist. These goals are achieved by possessing an item which counts as a tool for a particular skill, and they are always low-priority.

The assumption that the player will have these goals is probably the weakest such, as players may not always remember at each moment what skills they have and which ones they have tools for. However, the presence of these goals means that trading items for goods or services can have value, and they can differentiate good trades from bad ones. In situations where players had a chance to play multiple stories, players would be more likely to consider these goals when making choices.

- The final goals assigned to the player presume that players are generally sympathetic: unless a character is actively threatening or accusing another (deemed 'guilty' by the system), the player is assumed to have both "`avoid_threats_to`" and "`avoid_accusations`" goals for that character. These goals mean that when the player encounters non-protagonists threatening or accusing each other, the system assumes that the player will want to help the victim. As above, these goals are high-priority.

The predicates created by `goals.lp` include "`at(<T>, player_goal(<goal>))`" predicates, which specify the goals of the player for each timepoint, and "`at(<T>, goal_stakes(<goal>, <priority>))`" predicates, which assign priorities to each goal.

Although this static goal analysis works reasonably well, an interesting opportunity for future work presents itself here. Not only could *Dunyazad* benefit from dynamic goal analysis (perhaps along the lines of the user model used for dilemma generation in (Barber and Kudenko, 2007a)), as a system that produces choice-based narratives, it could incorporate explicit choices-of-goals as part of the stories it generates, and use the results of these choices as direct statements of player intent rather than relying on authorial guesswork. This would require a major overhaul of the goals system, however, and goals might have to be represented as mutable states of the world, which would place an additional burden on the solver.

One thing that should be mentioned here is that *Dunyazad*'s goal assumptions are targeted at players whose mode of engagement is a mix of avatar and power play. There are no goals related to character believability of the protagonist, nor are there goals related to things like curiosity. These could in principle be declared just like the current goals, but *Dunyazad* focuses on avatar and power play because these are the modes that its genre traditionally encourages (and in fact, these were the most popular modes of engagement among experiment participants; see section 8.8).

2. **Likelihood Analysis**

   This step is already performed by *Dunyazad*'s constituent constraints, using the "`skill_link`" predicates supplied by action definitions along with information on the skills and tools available to the actors involved in each action. Recall that individual outcome components are labelled as likely or unlikely, rather than entire outcomes (section 5.4.4). This step produces the "`at(<T>, likely_outcome(<option>, <outcome>))`" and "`at(<T>, unlikely_outcome(<option>, <outcome>))`" predicates. Figure 6.4 shows an example of output from this step based on just the single option shown in fig. 6.3 (normally each option at a timepoint would have a similar set of likelihood predicates).

3. **Option Analysis**

   The option analysis step uses the results of goal and likelihood analysis to attach expectations to each option. Goal predicates specify which state(s) affect them, and the likelihood of all outcomes is known, so option analysis is straightforward. First, all possible outcome components of an option are considered and if any of them negatively impacts a goal, that option **threatens** that goal. Similarly, any positive impact entails an **enables** relationship. Since each option has multiple outcome components, a single goal may be both threatened and enabled by the same option, of course. For the **advances** and

Scene:

A merchant carrying some perfume is being threatened by bandits.

Option:

You try to talk the bandits down (You have skill: negotiation and you have skill: storytelling).

Likelihood Predicates:

```
at(root, action(option(1), talk_down)).
...
at(root,
  likely_outcome(option(1),
  o(attitude, convinced))
).
at(root,
  likely_outcome(option(1),
  o(is_enraged, not_enraged))
).
at(root,
  unlikely_outcome(option(1),
  o(attitude, unconvinced))
).
at(root,
  unlikely_outcome(option(1),
  o(is_enraged, enraged))
).
```

Figure 6.4: An example of likelihood analysis predicates for the action shown in fig. 6.3 at the state shown in fig. 6.2, assuming that the player has the negotiation and storytelling skills. The "`action`" predicate is included for clarity. Note that the full definition of the "`talk_down`" action including the associated "`skill_link`" predicates can be found in appendix B.4.

hinders evaluations the same logic applies, but considering only outcome components marked as "`likely`". The result of the option analysis constraints is a set of "`at(<T>, expectation(<option>, <expectation>, <goal>))`" predicates that list the expectations for each option/goal pair. Figure 6.5 shows an example of these "`expectation`" predicates for the same option whose likelihood predicates are shown in fig. 6.4.

Scene:

A merchant carrying some perfume is being threatened by bandits.

Option:

You try to talk the bandits down (You have skill: negotiation and you have skill: storytelling).

Expectation Predicates:

```
at(root,
  expectation(
    option(1),
    advances,
    as_intended(inst(actor,you))
  )
).
at(root,
  expectation(
    option(1),
    advances,
    avoid_threats_to(inst(actor,businessperson_4))
  )
).
at(root,
  expectation(
    option(1),
    enables,
    avoid_threats_to(inst(actor,businessperson_4))
  )
).
at(root,
  expectation(
    option(1),
    threatens,
    avoid_threats_to(inst(actor,you))
  )
).
```

Figure 6.5: An example of option analysis predicates for the action shown in fig. 6.3 at the state shown in fig. 6.2, assuming that the player has the negotiation and storytelling skills. Note that the full definition of the "`talk_down`" action can be found in appendix B.4, while the definitions of each goal can be found in appendix B.3. The "`irrelevant`" expectations have been omitted for brevity.

4. **Relative Option Analysis**

   Integrating goal priorities and option expectations, relative option analysis produces "at(<T>, option_feel(<option>, <feel>))" predicates, which collapse the various expectations across goals at each option to produce a general evaluation of that option. For example, if an option has at least one "advances" expectation and it has no "threatens" or "hinders" expectations across all goals (only counting goals tied for highest priority at that option) an option is labeled as a "sure_thing." These rules establish one or more "option_feel"s for each option, and these are further aggregated into "option_structure" assignments, which are the prospective impressions show in table 5.3 (that table defines "option_structure" labels directly in terms of "expectation" labels for simplicity). Although "option_structure" labels could be assigned directly based on "expectation" predicates, the "option_feel" labels act as an intermediate level of representation, summarizing the "expectation" predicates across all goals at a particular option. There are nine (sometimes overlapping) "option_feel" values:

   (a) "safe"—An option where there are no "threatens" or "hinders" expectations, and there is at least one "enables" or "advances" expectation.

   (b) "sure_thing"—Any "safe" option that includes an "advances" expectation (instead of merely having "enables" expectation(s).

   (c) "hopeful"—An option which "advances" some goal, but may "threaten" a goal as well, although it doesn't "hinder" any goals.

   (d) "risky"—An option which at least has both "enables" and "threatens" expectations, but no "advances" or "hinders" expectations.

   (e) "tradeoff"—An option which "advances" one goal but "hinders" another.

   (f) "irrelevant"—An option which has no expectations associated with it.

(g) "longshot"—The converse of "hopeful": an option which "hinders" some goal but also "enables" a goal (although it does not "advance" any).

(h) "bad"—An option which "threatens" and/or "hinders" at least one goal and neither "enables" or "advances" any goals.

(i) "doomed"—Any "bad" option which "hinders" a goal (as opposed to merely having a "threatens" expectation).

The "option_structure" assignments derived from these "option_feel" labels are the subject of the experiment described in chapter 7 and provide one primary means of controlling *Dunyazad*'s output. Although they are discussed here as being 'assigned' based on existing choice structures, because they're just predicates in an answer set program, they can be required and the solver will solve for answer sets that include them. Requiring these "option_structure" predicates thus allows an author (or experimenter) to generate choices that (hopefully) make a desired prospective impression on the player.

Figure 6.6 shows an example of relative option analysis results. Based on having one "hopeful" option and one option which is both "bad" and "doomed," the choice in question is both "obvious" and "pressured" from a prospective standpoint. While formal definition of the "obvious" label was described in table 5.3, "pressured" is not explained there. The "pressured" label applies to choices where there are neither "safe" nor "irrelevant" options but there is at least one "hopeful" or "tradeoff" option, and it indicates a choice where risk is inevitable but success still seems possible. Note that exact definitions for these labels can be found in the file `eval.lp` listed in appendix B.2.

5. **Outcome Component Analysis**

Using information about player goals and the outcomes of each action, outcome component analysis maps the relationships between every possible outcome

component and each goal. These are represented using "`at(<T>, outcome_perception(<option>, <perception>, <goal>))`" predicates and have four basic "`<perception>`" values, similar to the "`expectation`" predicates from option analysis: "`good_for`," "`bad_for`," "`great_for`," and "`awful_for`." Outcome component analysis directly deduces these labels from the definitions of each goal and the states that a particular outcome component adds or removes. So for example, the "`o(success, dispelled)`" outcome component of the "`dispel`" action has a consequence which removes the "`polymorphed`" property from the action's target. Accordingly, any option that consists of a "`dispel`" action with a polymorphed target will be "`good_for`" the "`preserve_original_form`" goal of its target, (because the "`polymorphed`" state which is being removed is "`awful_for`" the "`preserve_original_form`" goal[3]).

[3]Although the counterpart of "`awful_for`" is "`great_for`" rather than "`good_for`," the inference here softens the expectation. The stronger "`great_for`" and "`awful_for`" perception labels are reserved for situations where they explicitly apply (i.e., a state which a goal labels as "`great_for`" or "`awful_for`" itself is the consequence of an option).

Scene:

A merchant carrying some perfume is being threatened by bandits.

Options:

1. *You try to talk the bandits down (You have skill: negotiation and you have skill: storytelling).*
2. *You travel onwards (No relevant skills).*

Prospective Predicates:

```
at(root, option_feel(option(1), hopeful)).
at(root, option_feel(option(2), bad)).
at(root, option_feel(option(2), doomed)).
at(root, option_structure(obvious)).
at(root, option_structure(pressured)).
```

Figure 6.6: An example of relative option analysis predicates for a choice at the state shown in fig. 6.2 between the action shown in fig. 6.3 and an alternative of simply travelling onwards. Definitions of these "`option_feel`" and "`option_structure`" values can be found in the listing for `eval.lp` in appendix B.2.

Scene:

A merchant carrying some perfume is being threatened by bandits.

Options/Outcomes:

1. *You try to talk the bandits down (You have skill: negotiation and you have skill: storytelling).*
   → *You talk to the bandits and convince them to back off.*
   The bandits are no longer threatening the merchant.
2. *You travel onwards (No relevant skills).*
   → *You continue your journey.*
   You have traveled to a new location.

Outcome Perception Predicates:

```
at(root,
  outcome_perception(
    option(1),
    good_for,
    avoid_threats_to(inst(actor,businessperson_4))
  )
).
at(root,
  outcome_perception(
    option(1),
    great_for,
    as_intended(inst(actor,you))
  )
).
at(root,
  outcome_perception(
    option(2),
    awful_for,
    avoid_threats_to(inst(actor,businessperson_4))
  )
).
```

Figure 6.7: An example of outcome component analysis of the choice shown in fig. 6.6. Definitions for these "`outcome_perception`" values can be found in the listing for `eval.lp` in appendix B.2.

Because of the way that goals are defined in *Dunyazad* this step of analysis is fairly trivial. Of course, not all goals are simple to define in terms of a globally applicable set of states that triggers them. For example, the "as_intended" goal deals directly with outcomes as opposed to world states, which may be different depending on the context of an action. Because of this, the "as_intended" goal definition has custom code which directly creates "expectation" and "outcome_perception" predicates by considering which outcomes are likely (and which is the "default_intent," as mentioned above). The "avoid_accusations" and "avoid_threats_to" goals also have a bit of custom code to make them more urgent: when an accusation or threat state is present, actions which have no effect on that state are given "hinders" expectations and "bad_for" outcome perceptions. All other goals simply specify one or more states that are "good_for," "bad_for," "great_for," or "awful_for" them, as described above.

Figure 6.7 shows the "outcome_perception" predicates that arise from analysis of the same choice presented in fig. 6.6 (this time with specific outcomes). Note that the "avoid_threats_to(inst(actor, you))" goal which generated a "threatens" expectation in fig. 6.5 was not affected by any actual outcomes: the imagined threat did not manifest itself.

6. **Full Outcome Analysis**

Full outcome analysis concerns the expectedness and overall valence of the entire outcome of each option. In *Dunyazad*, valence is represented by "at(<T>, outcome_overall(<option>, <evaluation>))" predicates, while "at(<T>, overall_predictability(<option>, <predictability>))" predicates represent predictability. The predictability of individual outcome components is represented by "at(<T>, outcome_predictability(<option>, <outcome>, <predictability>))" predicates. The "outcome_overall" predicates summarize the individual "outcome_perception" predicates across all player goals

and across different outcome components at an action. There are eight possible "`outcome_overall`" values:

(a) "`great`"—Reserved for options which are "`great_for`" at least one player goal and neither "`bad_for`" nor "`awful_for`" any.

(b) "`good`"—Options which would be "`great`", except that they are merely "`good_for`" at least one player goal as opposed to being "`great_for`" any.

(c) "`tradeoff`"—Options which are both "`great_for`" one goal and "`awful_for`" another, or which are both "`good_for`" one goal and "`bad_for`" another (without being either "`great_for`" or "`awful_for`" any).

(d) "`worth_it`"—Options where the best outcome is "`great_for`" a goal, but the worst outcome is "`bad_for`" a (presumably different) goal.

(e) "`not_worth_it`"—Options where the worst outcome is "`awful_for`" a goal, while the best outcome is merely "`good_for`" a goal.

(f) "`bad`"—Options which are neither "`good_for`" nor "`great_for`" any goals, and where the worst outcome is "`bad_for`" a goal.

(g) "`awful`"—as "`bad`" options but "`awful_for`" some goal rather than merely "`bad_for`" it.

(h) Neutral—Options where no outcome is relevant to any player goal.

Note that for all of these evaluations, only top-priority goals are considered (i.e., if there is at least one high-priority goal, only high-priority goals count; if not then low-priority goals can come into play). Summarizing the potentially complex per-outcome-component evaluations from the outcome component analysis step like this helps make the retrospective analysis more tractable (both for humans and for the solver). Most common cases avoid the ambiguous "tradeoff" label anyways, so not much information is lost.

Scene:

A merchant carrying some perfume is being threatened by bandits.

Options/Outcomes:

1. *You try to talk the bandits down (You have skill: negotiation and you have skill: storytelling).*
   → *You talk to the bandits and convince them to back off.*
   The bandits are no longer threatening the merchant.
2. *You travel onwards (No relevant skills).*
   → *You continue your journey.*
   You have traveled to a new location.

Overall Outcome Predicates:

```
at(root,
  outcome_predictability(
    option(1),
    o(attitude, convinced),
    predictable
  )
).
at(root,
  outcome_predictability(
    option(1),
    o(is_enraged, not_enraged),
    predictable
  )
).
at(root,
  outcome_predictability(
    option(2),
    o(onwards, onwards),
    predictable
  )
).
at(root, overall_predictability(option(1), predictable)).
at(root, overall_predictability(option(2), predictable)).
at(root, outcome_overall(option(1), good)).
at(root, outcome_overall(option(2), awful)).
```

Figure 6.8: An example of full outcome analysis of the choice shown in fig. 6.7. Definitions for these "`outcome_predictability`," "`overall_predictability`," and "`outcome_overall`" values can be found in the listing for `eval.lp` in appendix B.2.

Along with "outcome_overall" labels, this step assigns one of the following six "outcome_predictability" values to each outcome component and also (as "overall_predictability" predicates) to each option:

(a) "predictable"—An outcome component is "predictable" when its value is the sole likely value for its variable. An entire option is "predictable" when every single important outcome component (an outcome component that affects one or more goals tied for the highest priority) is "predictable", and there is at least one such outcome component.

(b) "expected"—An outcome component is "expected" when it is one of several likely values for its variable, at least one which is important. An entire option is "expected" when every outcome at that option is either "predictable," "expected," or "average," and they're not *all* "average."

(c) "average"—These outcome components aren't likely, but neither are they unlikely, and no other values for their variable are likely, while at least one is unlikely. In other words, an outcome component is "average" when a variable avoids an unlikely value but the value that it takes on isn't deemed likely. An entire option is "average" if every one of its important outcome components is average, and it has at least one important outcome.

(d) "unpredictable"—An outcome component is "unpredictable" if for its variable, there are no likely or unlikely values, or if all values for its variable are unlikely. An option as a whole is "unpredictable" if it has at least one "unpredictable" component and no "unexpected" or "unfair" components (counting only important outcome components).

(e) "unexpected"—An "unexpected" outcome component is one where an "unlikely" value is selected despite the presence of at least one "neutral" value, but no actually "likely" values are possible. For an option to be

labeled "unexpected" as a whole, it must have at least one important "unexpected" component and no "unfair" components.

(f) "unfair"—An "unfair" option component is one where an "unlikely" value has been selected for a variable over a possible "likely" value. Every option that has an important "unfair" outcome component is considered "unfair" as a whole.

There are actually two other predictability values that are used only for entire options: "irrelevant" (used for options that are entirely lacking in "outcome_perception" predicates) and "unrecognized" (used as a catch-all for situations that don't fit any other label). As with the "outcome_overall" summarization, building "overall_predictability" predicates reduces the complexity of possible "outcome_predictability" configurations to a few cases which cover common situations. In the same way that this is useful for efficient analysis (and more detailed analysis is always possible when necessary), it is helpful when authoring constraints (and likewise, constraints directly targeting complex "outcome_predictability" situations can be written if necessary). The "outcome_overall" and "overall_predictability" predicates from this step are used alongside the "outcome_feel" predicates from relative option analysis as inputs for retrospective analysis.

Figure 6.8 shows an example of these constraints using the same choice shown in 6.7. In this case, the outcomes were uniformly "predictable," and their aggregate positive/negative evaluations were simple as each had either all-good or all-bad outcome perceptions[4]. As above, the full conditions under which the predicates shown here apply are available in the source listing of the file eval.lp in appendix B.2.

[4]If this choice strikes you as a bit more complicated than the labels assigned by *Dunyazad* suggest, see the discussion of similar choices in sections 8.5.3 and 8.5.5.

Outcome Feel	**Option Feels**	**Overall Predictabilities**	**Overall Outcomes**
"expected_success"	"sure_thing" "safe" "hopeful"	"predictable" "expected"	"great" "good"
"unfair"	"sure_thing" "safe" "hopeful"	"unexpected" "unfair"	"bad" "awful"
"nice_gamble"	"risky" "tradeoff" "irrelevant"	"average" "unpredictable"	"great" "good" "worth_it"
"bad_gamble"	"risky" "tradeoff" "irrelevant"	"average" "unpredictable"	"not_worth_it" "bad" "awful"
"expected_failure"	"long_shot" "bad" "doomed"	"predictable" "expected" "average"	"bad" "awful"
"miracle"	"long_shot" "bad" "doomed"	"unexpected" "unfair"	"great" "good" "worth_it"

Table 6.3: *Dunyazad*'s six outcome feel structures. If at least one of each of the given "option_feel," "overall_predictability," and "outcome_overall" values applies to an option, the listed "outcome_feel" applies. For example, if a choice has a "safe" "option_feel," a "predictable" "overall_predictability," and a "good" "outcome_overall", then it meets the criteria for having the "expected_success" outcome feel.

7. **Retrospective Analysis**

   By comparing outcomes with option feels, an overall impression of a choice can be obtained. These are represented using "`at(<T>, outcome_feel(<option>, <feel>))`" predicates. *Dunyazad* has six different "`outcome_feel`" structures that it recognizes, which are shown in table 6.3. Figure 6.9 provides an example, showing the "`outcome_feel`" predicates that result from analysis of the choice shown in fig. 6.7.

   It's worth noting that contrary to fig. 5.1, *Dunyazad* doesn't directly incorporate information from its option analysis step into its retrospective analysis. This is because it doesn't use this more nuanced information, just using the higher-level "`outcome_overall`" predicates. This connects to a deeper point: as a tool for automated *analysis* of choices, *Dunyazad* would not perform very well, because its definitions are intentionally narrow. The reason for this is twofold: first, broad definitions are harmful during generation, because they make it easier to produce things which don't actually fit the categories

Scene:

A merchant carrying some perfume is being threatened by bandits.

Options/Outcomes:

1. *You try to talk the bandits down (You have skill: negotiation and you have skill: storytelling).*
   → *You talk to the bandits and convince them to back off.*
   The bandits are no longer threatening the merchant.
2. *You travel onwards (No relevant skills).*
   → *You continue your journey.*
   You have traveled to a new location.

Outcome Feel Predicates:

```
at(root, outcome_feel(option(1), expected_success)).
at(root, outcome_feel(option(2), expected_failure)).
```

Figure 6.9: An example of retrospective analysis predicates for of the choice shown in fig. 6.7. Definitions for these "`outcome_feel`" values can be found in the listing for `eval.lp` in appendix B.2.

they are supposed to represent. Second, scope and to some degree breadth of definitions was limited by effort—with more work, these could be improved. Because of these limitations, *Dunyazad*'s formal definitions of various properties are designed to be sufficient, but never necessary conditions for the effects they describe. In other words, they are idiosyncratic: *Dunyazad*'s definition of an 'obvious' choice might be one way to make choices obvious, but there could be other rules for constructing choices that also resulted in obviousness without necessarily having any overlap with the kinds of obvious choices that *Dunyazad* generates. Their idiosyncrasy is not a problem, of course: we can still learn about obviousness from studying a subset of obvious choices. However, we need to be careful not to generalize our results too far.

Coming back to the results of retrospective analysis, these "`outcome_feel`" predicates give authors another high-level means of control over *Dunyazad*. As with the "`option_feel`" predicates, constraints requiring the presence of a particular "`outcome_feel`" force *Dunyazad* to produce choices that include such options (if it can), and this is how the experiment described in chapter 8 was set up.

As is hopefully clear from this description, *Dunyazad*'s poetic constraints are a direct mirror of the analysis method presented in section 5.4. Although they operate simultaneously rather than sequentially, each step of analysis shown in fig. 5.1 corresponds to a set of constraints in *Dunyazad*'s ASP code. Of course, this is not simply a case of the code mimicking the theory: both the human-usable analysis method and the machine-usable code were designed together, with changes in each informing the development of the other. Furthermore, the close parallels provided, which extend to the structural level, make this two-way information flow smoother, allowing both bug fixes to the code and caveats added to the theory to inform each other.

6.3 High-Level Control

Dunyazad's ASP rules allow it to represent an entire story without any trouble, but the complexity of the problem of generating an answer set when all elements of a story are left unconstrained is prohibitive. In theory, *Dunyazad*'s rules allow an entire story's worth of content to be solved for, but in practice, given the exponential runtime complexity of the solving algorithm and the size of the problems that *Dunyazad* can create, this is impractical. Some simple statistics illustrate this: when asked to generate a single "relaxed" choice, *Dunyazad* produced a problem with 1,066,956 atoms and 24,974,521 grounded rules, according to the output of `clasp --stats`. The grounded output for this problem was 994,932 lines of code[5]; a total of 3.5 gigabytes of data (albeit in a relatively memory-inefficient format for human readability). Grounding and finding the first answer set for this problem took almost 24 seconds of CPU time on a modern Intel i7 laptop with 12 gigabytes of RAM. Even given *Dunyazad*'s simple target genre, a full story would contain dozens to hundreds of nodes because of branching, and asking the solver to tackle that all at once is clearly out of the question (unless perhaps you are reading this several decades from now and things have changed?).

The solution to this problem was to add a higher-level control layer that would ask the answer set solver to solve just one node at a time. This layer keeps track of all of the answer sets received and integrates the parts of them that represent story content into a running predicate representation of the entire story, which is fed into every solver query. This means that even though each ASP task can only make changes to one timepoint at once, it can see the entire story so far, and can take that information into account. Adding many nodes as fixed constraints like this doesn't cost extra time; in fact, it usually saves time

[5]The number of lines of grounded output can be less than the number of grounded rules because a single line can express many rules, particularly when it instantiates a cardinality constraint.

because the constraints added can make a solution easier to find for the solver (in particular, solving the first timepoint of a new vignette which necessarily involves a setup is much slower than solving intermediate timepoints which just play out action that has already been initiated). The high-level control even goes a step further: the process of filling in information for a timepoint is actually split into four independent steps that can be processed separately (although at a slight cost, since choices in separately-processed steps can no longer affect each other). The four steps for filling in a timepoint are defined in the file `grow.lp`:

1. "`initialize_node`"—This phase involves choosing a setup for the target timepoint if necessary and locking down the starting state of the timepoint, including any changes introduced by the chosen setup. Player goals for a timepoint can also be discovered during this phase. Values for any variable elements of a setup must be chosen during this phase.

2. "`build_options`"—The bulk of *Dunyazad*'s constraints are tied to this phase, which solves for the options and outcomes at a timepoint.

3. "`add_branch_nodes`"—This phase adds successor links to the target timepoint and creates new successor timepoints if necessary. It's also responsible for transferring state between timepoints. The only decisions made during this phase are whether or not two branches which have exactly equivalent world states should link to the same timepoint or to two different timepoints, so running it separately loses little.

4. "`add_surface`"—This phase doesn't change any of the essential facts of the story, but merely adds "`surface_property`" predicates and the like for the convenience of the English generation code. There are currently no constraints that link surface properties to story world state, so there is no cost associated with running this phase separately.

Although four phases are defined, the control code actually makes a compromise for the sake of greater flexibility and only runs two separate problems per timepoint. First, the "`initialize_node`" and "`build_options`" phases are run together, so that choices about setup variables can be influenced by choices about actions and their outcomes and vice versa. Next, the "`add_branch_nodes`" and "`add_surface`" phases are run together, which usually takes a fraction of the time taken by the other two phases.

Using two steps per node, the high-level code first fills in the root node of a story and then proceeds to randomly select an empty leaf timepoint to fill in, followed by random selection of an unbranched node to add branches to, and so on, until either there are no more branches left (because each has hit an ending) or a time- or story-size-limit is reached. By proceeding in this manner, the high-level control code can solve for a story with dozens of nodes in a matter of minutes. Of course, there is a sacrifice associated with this: it separates decisions about different timepoints, meaning that later timepoints must always do their best with the constraints placed on them by the past. In practice, *Dunyazad* may even become stuck, when an authorial constraint (such as "make all choices relaxed") combines with accumulated state to create an unsatisfiable problem. With *Dunyazad*'s current design this happens quite rarely, so a backup procedure (perhaps deleting the unsatisfiable node and several of its parents before retrying) has not been necessary. It would of course be possible to implement full backtracking search at the higher level if conflicts were frequent, although the cost per timepoint instantiated (currently in the tens of seconds) would seem to suggest a search for alternative solutions.

As mentioned previously, *Dunyazad*'s high-level control code is written in Python (version 3.x). Besides the simple iterative process just described, there is code for managing constraint sets to be fed into the solver during each run (including extracting just the story predicates from previous runs), running

the solver via the command line and capturing and parsing its output, and feeding the results into the English generation code. Along with classes for representing predicates and answer sets, a full parser for `clasp`'s output is included, although it is a bit slow and could do with more optimization. For situations like the experiments described in chapters 7 and 8, there are also mechanisms for inserting extra constraints that will be used during every call to the solver. Ultimately, the high-level Python code is relatively straightforward, with the exception of the English generation code. Although *Dunyazad* generates English text using a somewhat sophisticated template-based system, the inner workings of that system (beyond its basic function of expressing predicate story representations as English text) are not critical to *Dunyazad*'s function as a choice point generator. Those interested can find a description of the template-based text generator that *Dunyazad* uses in appendix A.

6.4 Summary

No project of *Dunyazad*'s scope (modest though it may be) can be perfectly described in a technical write-up. There are details that were left out or glossed over in the sections above, and to thoroughly understand how *Dunyazad* works down to the smallest details reading its source code is unavoidable. *Dunyazad* is an open-source project, and full source code can be found online at `https://github.com/solsword/dunyazad`. Additionally, an archived snapshot current as of this writing can be found at `www.escholarship.org/uc/item/32d6p0kg`. Luckily, a deep understanding of the source code should be unnecessary for most scholars who want to take something away from this work. For example, the analysis method presented in the chapter 5 does not depend on the details of *Dunyazad* for its validity (although it is not impossible that it has undiscovered caveats which could be revealed by inspection of *Dunyazad*'s code).

Another reason that the description of *Dunyazad* given here can be appreciated on its own is that it describes all of the driving principles behind *Dunyazad*'s inner workings, in enough detail for someone else to construct their own working system that generates narrative choices. Technical considerations (such as iterative vs. all-at-once story construction) aside, the principles driving *Dunyazad* are not terribly complicated: a story representation that includes the concept of choices and actions with variable consequences along with systems for estimating the player's interpretation of options and outcomes in terms of player goals. One could re-implement it using an entirely different theory of choice poetics, for example, or without using logic programming at all.

From the perspective of generative systems, *Dunyazad* makes the statement: "Generating narrative choices intentionally is computationally tractable, and here's one way to do it." Ideally this chapter is a convincing demonstration of that, but *Dunyazad* has ambitions beyond being a working generative system. In chapters 7 and 8, I will discuss the results of two experiments that both demonstrate *Dunyazad*'s capabilities as a generative system and provide useful insight into choice poetics by leveraging its capacity as a tool for theory development.

Chapter 7

Experiment I: Prospective Impressions

The goal of this chapter and the one that follows is to demonstrate *Dunyazad*'s ability to manage player reactions by generating distinctive choice structures. Because of the way it uses answer set programming, *Dunyazad*'s choice generation system not only generates choices, but itself constitutes a theory of choice poetics. The survey data presented here thus not only validate that players' perceptions match *Dunyazad*'s intent, but also inform the theory of choice poetics. Statistical analysis of the data indicates that *Dunyazad* is largely successful in its goals, but also reveals some places where either the code, the theory, or both can be improved. These unexpected results are in fact one of the larger goals of *Dunyazad* as a project: by operationalizing choice poetics, *Dunyazad* enables experiments which can reveal details that wouldn't be obvious from simply observing human-authored choices, because as a computer program, *Dunyazad* makes *inhuman* mistakes.

Note: parts of this chapter have appeared in abridged form in (Mawhorter, Mateas, and Wardrip-Fruin, 2015a).

The results presented here are of course limited by *Dunyazad*'s specificity: *Dunyazad* has a particular approach for creating e.g., **obvious** choices and information about properties of its **obvious** choices doesn't necessarily generalize to all obvious choices. Largely, however, the extra considerations that this data suggests should be taken into account *do* generalize, because they apply to any analysis or generation scheme which uses a particular subset of *Dunyazad*'s constraints. For example, one result that will be discussed suggests that when trying to figure out how players will evaluate different options, analysis in terms of absolute values is insufficient. This result (which is unsurprising, as it echoes psychological research on real-life choices; see (B. Schwartz et al., 2002)) clearly isn't something that's likely to be limited to just the particular choices that *Dunyazad* generates (in terms of genre or any other factor). There might be some cases where humans do use only absolute value judgements, but until such cases are identified, it's safer to assume that both absolute and relative value judgements between options should be accounted for. In any case, the idea that absolute value judgements are insufficient for understanding choice poetics generalizes beyond the specifics of *Dunyazad*'s choices.

The experiment presented here (and the experiment discussed in chapter 8) was set up in order to test *Dunyazad*'s functionality and thereby also inform the theory of choice poetics that it is based on. This first experiment is mainly concerned with prospective impressions, which are the result of the "relative option analysis" step in the goal-based choice analysis method described in section 5.4. In *Dunyazad*, these are represented using "`option_feel`" predicates, as described in section 6.2.4. After analyzing the results of this experiment, I conducted a second experiment (discussed in chapter 8) focused on retrospective impressions, which result from retrospective analysis and are represented by "`outcome_feel`" predicates. This chapter describes the details of the first experiment, including details of the experimental setup that are common to both experiments.

Both experiments broadly confirmed *Dunyazad*'s abilities to generate the kinds of choices it was asked to, but also uncovered areas where it struggled. In particular, *Dunyazad* has trouble balancing options to create dilemmas because it does not have a sufficiently detailed model of goal priorities, and it has trouble analyzing situations where multiple goals conflict, especially when such situations constitute moral dilemmas. Additionally, it has trouble with actions which expend resources in service of a goal, and its indirect management of expectations sometimes leaves something to be desired. All of these shortcomings suggest future work, of course, but also prompt reflection on the goal-based choice analysis method from section 5.4. Accordingly, section 8.11 revisits some of the concepts from goal-based-choice-analysis in light of the results from this experiment and the one presented in chapter 8. The main takeaways for goal-based choice analysis are that humans should be careful to apply their faculties for comparing options in terms of things like moral imperatives and opportunity costs when analyzing options and outcomes. Some other caveats are that relative option analysis is important in most situations, rather than only being needed when options are very similar, and that the outcome of an option can influence post-decision impressions of its desirability as an option pre-decision.

7.1 Overview

To exercise *Dunyazad*'s prospective impressions system, I set up *Dunyazad* to construct three different kinds of choices:

- **Relaxed** choices, where the stakes were low and there were no bad options.
- **Obvious** choices, where there was a single option that stood out as more advantageous than the rest.
- **Dilemmas**, where every option was about equally undesirable.

These three prospective impressions cover positive and negative option impressions, low- and high-stakes choices, and contrasting and similar option sets, so they exercise every dimension of player expectations that *Dunyazad* attempts to model. Of course, the full set of *Dunyazad*'s prospective impressions (shown in table 5.3) is broader, and there are plenty of prospective impressions that *Dunyazad* does not include definitions for, but **relaxed**, **obvious**, and **dilemma** choices were chosen as representative for this experiment. Note that factors which give rise to "obviousness" or "being a dilemma" are quite a bit simpler than, say, factors that make a player feel regret. These experiments, and indeed work on *Dunyazad* so far, has focused on these simple effects because if they can't be produced reliably, more complex effects are unlikely to work either. The results presented here are thus only the first step in *Dunyazad*'s use as a tool for exploring choice poetics.

After constructing choices, I ran a survey that asked participants to read a single choice generated by the system and answer some questions about their perception of the choice. I analyzed the responses across choice categories and compared them against a uniform distribution to determine if players' perceptions match what the system intended. The data show that *Dunyazad* was mostly able to produce the desired prospective impressions, but in a few specific cases there were surprising results. Because *Dunyazad* is a transparent operationalization of the goal-based choice analysis technique described in section 5.4, both the expected and surprising results can usefully inform not only the system's development but also the theory of choice poetics.

7.2 Method

The primary goal of this experiment was to assess *Dunyazad*'s ability to manage and predict player's prospective impressions, i.e., perceptions of a choice before

making a decision. To that end, the experiment focuses on players' perceptions of options at a choice, and does not even present outcomes to the participants at all. The three choice types that were generated were chosen because they are each distinct in terms of the player expectations they engender, and because, as stated earlier, they together exercise *Dunyazad*'s capacity to reason about stakes, positive and negative indicators, and both contrasting and similar options.

To gather data on player expectations, I generated choices using *Dunyazad*, showed them to study participants, and asked participants a series of questions about specific qualities of the choice they just read. Because I used Amazon Mechanical Turk to gather data, my participants were each paid a small amount and presumably approached the survey as a means to earn money rather than as a voluntary undertaking. Because of this, questions were asked in a hypothetical manner (e.g., "If you were reading this story, which option would you choose?") rather than directly (e.g., by having participants pick an option) to imply that the task at hand was asking them to judge the choice as someone reading it for entertainment might. Of course, this framing (and being asked specific questions in general) might encourage an analytical mode of engagement, which is not what *Dunyazad* is designed to support, but that limitation is to some degree inevitable when survey responses are solicited.

To control for participants paying little attention, being unfamiliar with English, or simply filling in random responses, two check questions were asked. Responses from participants who failed to answer these questions satisfactorily were excluded from the analysis, as were responses where one or more questions were left blank (about 15% of all participants).

7.2.1 Treatments

For this experiment, there were three experimental treatments, each corresponding to a different set of rules used by *Dunyazad* to generate the choice experienced

by a subject. These are the "`obvious`," "`relaxed`," and "`dilemma`," choice types described above (see also table 5.3 for formal definitions of these as *Dunyazad* perceives them). The system definitions given in that section represent extra constraints placed on the choices generated by *Dunyazad* beyond its common core rules. Each participant thus saw a choice shaped by one of three different constraint sets.

Of course, each constraint set can generate a potentially large range of specific choices, but this study was interested in perceptions common across choices generated using the same constraints. One possibility would be to show each participant a unique choice from the space of choices possible given one of the treatment conditions. However, this setup would mean that no single choice would be seen by more than one participant, and so there would be no way to analyze the contribution of individual choices to the perception of the different treatments. Instead, I generated three different choices per treatment, and showed each choice to ten participants, for a total of 30 participants per treatment pre-attrition.

You come to a tavern and decide to rest for a while. A merchant is bored and a noble is bored and an innkeeper seems knowledgeable. What do you do?

1. *You play a song for the noble*
   (You have skill: musician. You have no tool for music).
2. *You gossip with the innkeeper*
   (You are missing skill: negotiation).
3. *You play a song for the merchant*
   (You have skill: musician. You have no tool for music).

Figure 7.1: An example choice.

7.2.2 Setup

To set up the experiment, I used *Dunyazad* in its "experiment" mode (which causes it to generate only a single choice and to use a special framing) to generate three choices for each of the experimental conditions. Additionally, each choice was required to have exactly three options, so that the number of options wasn't a confounding factor in the data. These nine choices were generated sequentially by the system, so there was no opportunity to cherry-pick "good" examples of each treatment category. The choice shown in fig. 7.1 is the first choice that was generated; it is in the "`dilemma`" treatment. The framing for each choice presented the skills that the system had assigned the player character for that choice, and established a basic context for the choice (see fig. 7.2). The framing for each choice differed only in the skills presented and the fictional destination name, which *Dunyazad* chooses randomly from a fixed list of made-up names.

> *You are about to set out on an epic journey. You are are heading towards towards the distant country of Jyväsky, hoping to earn fame and fortune. You have some perfume and a book of legends, and you have skill: literacy, you have skill: musician, and you have skill: healing. Eager to be on your way, you set off on the road towards Jyväsky.*

Figure 7.2: Example framing. The repetition of "towards" is a typo that was present in the text shown to participants.

Once the choices were generated, their text was broken into parts and put into a comma-separated values file for upload to Amazon Mechanical Turk where the parts would be substituted into a template. Given a survey template, Mechanical Turk generated an individual survey page for each choice, and ten tasks were posted per choice (a total of 90 tasks), which workers on Mechanical Turk were able to preview, accept, and fill out for payment. Each worker was

paid 50 cents[1] upon completing their survey. Myle Ott's "uniqueturker" script (`https://uniqueturker.myleott.com/`) was used to ensure that no individual worker filled out the survey more than once. To avoid being targeted by bots, the tasks required workers with a 97% acceptance rate across at least 1000 accepted tasks (this is lower than the default settings).

7.2.3 Survey Content

Each survey was divided into three sections. The first section "Preliminaries" began with a prompt that read:

> *To provide useful data for this survey, you must be at least 18 years old and able to read English. To confirm this (and to confirm that you aren't a bot), please answer the following question.*

This section just contained the following question designed to ensure that subjects were at least 18 years old and had basic English proficiency:

> *If you're at least 18 years old, please don't write "Age of years eighteen least at am I that confirm I," as the answer here, instead write that sentence backwards, ended with an exclamation point. If not, please do a different HIT, as I cannot use your data in my results, and thus I will not accept your response.*

The second section of the survey was titled "The Choice" and it included the choice that *Dunyazad* generated. It began with a prompt:

[1]This was chosen based on online advice to pay about minimum wage for tasks on Mechanical Turk. Given the median response time, the hourly pay rate was $7.50, but in retrospect, the inconvenience of survey tasks (where a worker cannot complete the same task many times and thus work more efficiently) suggests that a higher pay rate would be appropriate. For this reason (and because the second survey included more questions) I used a higher pay rate for the second experiment. The total cost (about $50 in this case, counting Amazon's 10% fee) was quite cheap.

> *Please read the following short story which presents you with a choice, then answer the questions about that choice below.*

In this section, each survey displayed one of the nine generated choices, followed by a single multiple-choice question:

> *If you were reading this story, which option would you pick?*

This question had three options: "Option 1," "Option 2," and "Option 3."

The final section of the survey was titled "Opinion Questions" and began with the following prompt:

> *Please rate your agreement with the following statements, from 1 (strongly disagree) to 5 (strongly agree).*

This section contained 8 Likert items[2] in the fixed order shown here (the quotes were part of the survey):

1. *"There are no bad options at this choice."*
2. *"There is a clear best option at this choice."*
3. *"The stakes for this choice are low."*
4. *"There are no good options at this choice."*
5. *"All of the options at this choice are about equally promising."*
6. *"There are options at this choice." (This is a trick question to test whether you're paying attention. Please simply indicate that you are in complete disagreement.)*
7. *"This is a difficult choice to make."*
8. *"This choice feels like it will have important consequences."*

[2]These were individual Likert items which did not compose a Likert scale, as the goal of the survey was to directly measure opinions, and there were no underlying psychological variables presumed to be giving rise to behavior.

Each question in this section was followed by the same five numbered options presented vertically:

1. *strongly disagree*
2. *somewhat disagree*
3. *neutral*
4. *somewhat agree*
5. *strongly agree*

For these questions (and the multiple-choice question in the previous section) participants selected an option by clicking a radio button next to that option. There were no default responses, so someone who didn't click any of the radio buttons would submit a blank response for that question. Responses were treated as ordinal data, and labeled with the numbers 1 through 5 in the same order they appeared here (i.e. $1 \rightarrow$ strongly disagree; $5 \rightarrow$ strongly agree).

7.3 Hypotheses

Before conducting the survey, I came up with a set of initial hypotheses about how participants would answer these questions based on the treatment conditions. There were three types of hypothesis: single-treatment hypotheses, between-treatment hypotheses, and stakes hypotheses. Each singe-treatment hypothesis posited that under a particular treatment, respondents would generally agree with or disagree with a particular question. The single-treatment hypotheses used are listed in table 7.1. Agreement with a question was determined by a one-sided Mann-Whitney-Wilcoxon U test (Wilcoxon, 1945; Mann and Whitney, 1947)

against a uniform distribution[3] of responses with the alternate hypothesis being "The median of the survey responses is significantly higher than the median of the uniform distribution." Disagreement likewise used a Mann-Whitney-Wilcoxon U test with the alternate hypothesis that the median of the survey responses was smaller than that of a uniform distribution. In both cases, a confirmation of a hypothesis was taken to be significant for $p < 0.05$.

For the between-treatment hypotheses, survey data from two different treatments were compared using a Mann-Whitney-Wilcoxon U test to test whether one was statistically more-agreed-with than another (with the threshold for significance again being set at $p < 0.05$). The statistics are the same for the converse cases, so only one test was performed per hypothesis (i.e., if responses to question 3 showed significantly more agreement for the "`obvious`" treatment than for the

[3]Another possible standard for comparison would be an all-neutral distribution. Comparison against a uniform distribution helps demonstrate that the answers aren't random, however, and the unsupervised nature of the study meant that participants might have answered some questions at random in order to complete it quickly (although there were some other guards against this). To verify that a uniform distribution was an acceptable standard I ran the same hypothesis checks against an all-neutral distribution and found that support for each was equal to or greater than support when comparing to the uniform distribution used.

Question	Obvious	Relaxed	Dilemma
1. There are no bad options at this choice.	-	agree	disagree
2. There is a clear best option at this choice.	agree	-	disagree
3. The stakes for this choice are low.	-	agree	-
4. There are no good options at this choice.	disagree	disagree	agree
5. All [options] are about equally promising.	disagree	-	agree
6. There are options at this choice. [...]	-	-	-
7. This is a difficult choice to make.	disagree	-	agree
8. This choice [has] important consequences.	-	-	agree

Table 7.1: Prospective single-treatment hypotheses by treatment

Question	Hypotheses
1. There are no bad options at this choice.	relaxed > dilemma
2. There is a clear best option at this choice.	obvious > dilemma
3. The stakes for this choice are low.	-
4. There are no good options at this choice.	dilemma > obvious & relaxed
5. All [options] are about equally promising.	dilemma > obvious
6. There are options at this choice. [...]	-
7. This is a difficult choice to make.	dilemma > obvious & relaxed
8. This choice [has] important consequences.	dilemma > obvious & relaxed

Table 7.2: Prospective between-treatment hypotheses by treatment. The ">" signs indicate more agreement relative to an alternate treatment. "> obvious & relaxed" indicates that that treatment is hypothesized to be more-agreed-with than both of those treatments individually (two separate hypotheses).

"`relaxed`" treatment, it follows automatically that they show significantly less agreement for the "`relaxed`" treatment than for the "`obvious`" treatment—the test is symmetric). Table 7.2 lists the between-treatment hypotheses. Combined across all three treatments, there were a total of 13 single-treatment and 9 between-treatment hypotheses.

The stakes hypotheses were simpler: across all treatments, participants who were shown a choice identified by the system as low-stakes should agree with the statement "The stakes for this choice are low." Additionally, participants shown high-stakes choices were expected to disagree with that statement. Both of these hypotheses were validated using Mann-Whitney-Wilcoxon U tests against uniform distributions as above. A fall-back hypothesis for stakes was that participants who saw low-stakes choices would agree more strongly with that statement than participants who saw high-stakes choices (a between-cases hypothesis).

7.4 Results

Before processing the data from Amazon Mechanical Turk, responses that showed signs of inattentiveness, non-proficiency with English, or excess haste were filtered out. In fact, as responses were being submitted, responses where either the age question was left blank or where the trick question (question 6) had an answer other than "1 - strongly disagree" or "5 - strongly agree" were rejected, meaning that Mechanical Turk would not pay the responder and the task would be reposted. Including 6 rejected responses, a total of 96 responses were gathered, with 30 non-rejected responses to each treatment (10 per question). Of the responses which remained, further filtering was performed:

- Responses where the answer to question 6 wasn't "1 - strongly disagree" were dropped. These indicate a responder who may not be paying close attention to the survey.
- Responses where the total time spent on the task was less than or equal to 90 seconds were dropped. Answering all 8 questions in just 90 seconds is probably not possible if each question is given reasonable consideration. The median response time was 240 seconds (4:00) before filtering and 280 seconds (4:40) after filtering, although these times are simply the time between a participant accepting and submitting a task; they don't necessarily work on the task exclusively during that time, and in general accepting several tasks before doing them is a common pattern of behavior.
- Responses where any question was left blank were dropped. Given the "neutral" response option and the content of the survey, leaving a question blank is more likely a sign of lack of attention than of hesitation to answer.

After these filtering steps, a total of 79 valid responses remained, with 25 responses to the "`obvious`" treatment and 27 responses each to the "`relaxed`" and

"`dilemma`" treatments. Given these final response counts, I used a uniform distribution of 25 data points to test my single-treatment hypotheses, as that was the closest multiple of 5 (the number of response options) to the sizes of my data sets. The low- and high-stakes conditions had 44 and 35 responses respectively; I used the same 25-point uniform distribution to test my stakes hypotheses.

A summary of the data is shown in fig. 7.3. The percentages shown are total percent of participants who disagreed, were neutral, or agreed with the given question (from left to right) and the percentages for each separate response are plotted as colored bars. For each question, data from each of the three treatments is plotted separately, and in some cases divergence is immediately apparent. Of course, differences that seem apparent on this summary graph may or may not be statistically significant.

I tested my hypotheses as described above, and the results of those tests are show in tables 7.3 and 7.4. Table 7.3 shows the single-treatment hypothesis results: 9 of my 13 hypotheses were confirmed by my data, while 4 were not. Each entry in this table indicates the hypothesis (agree $\rightarrow$ "A" and disagree $\rightarrow$ "D"), the p-value for that hypothesis (the hypothesis is confirmed if the p-value is below 0.05), and if confirmed, the common-language effect size for that test. Table 7.4 shows the between-treatment hypothesis results: 8 of my 9 hypotheses were confirmed by my data and 1 was not.

Note that the common-language effect size is just the percentage of comparisons between the cases being tested that support the alternate hypothesis. This means that, for example, if all responses are neutral, the common-language effect size when asserting that the responses are greater than a uniform distribution will be exactly 50% (the theoretical minimum common-language effect size). This is because 50% of comparisons between an all-neutral data set and a uniform data set will support the alternate hypothesis (that the neutral data's median is higher) and 50% will refute it (counting tied comparisons as half-supporting and

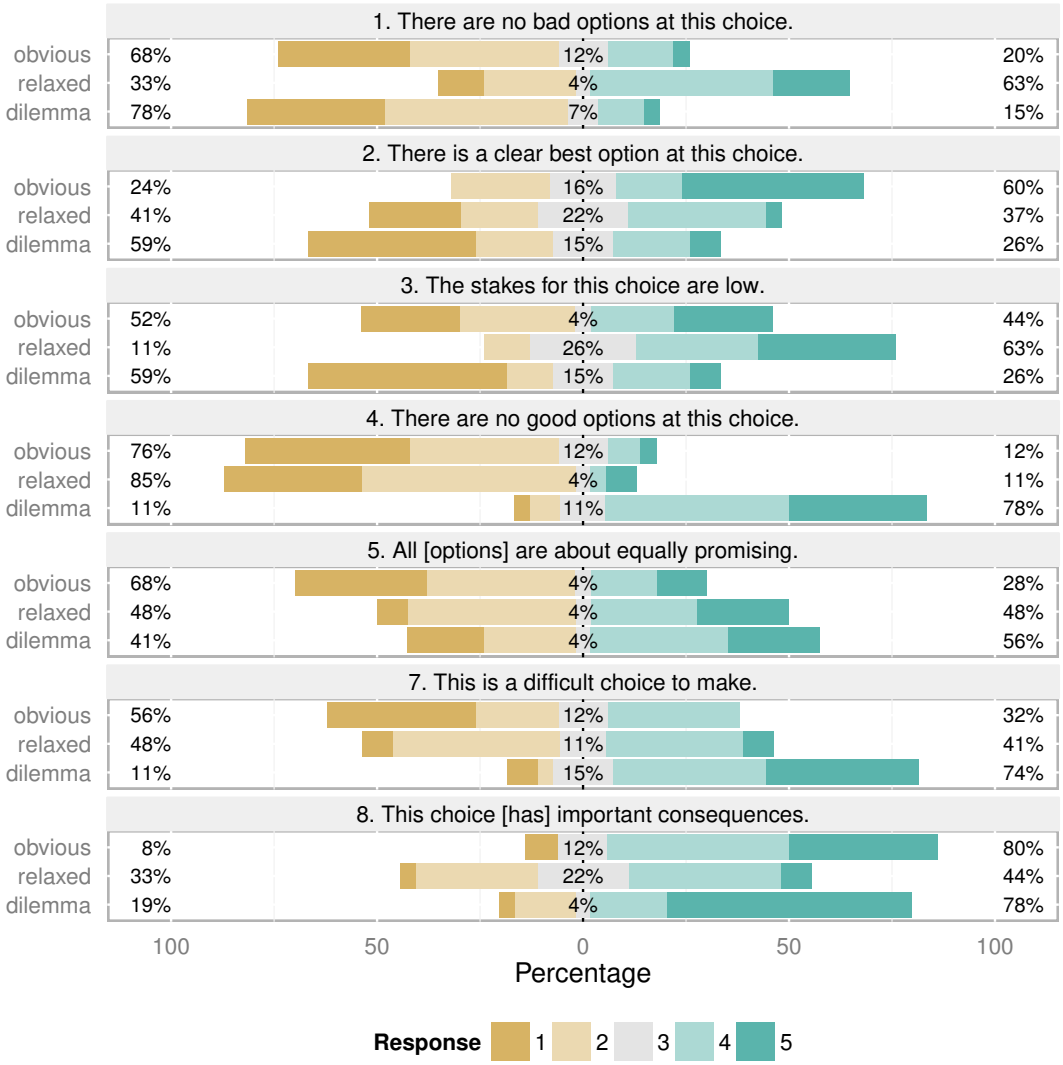

Figure 7.3: A summary of the data by question plotted as percentages of respondents per treatment who gave each possible answer following (Robbins and Heiberger, 2011). Responses range from 1 (strongly disagree) to 5 (strongly agree) with 3 being "neutral." Disagreeing responses are plotted to the left, and agreeing responses are plotted to the right.

Question	Obvious			Relaxed			Dilemma		
1. No bad options.	-			A	0.176	×	**D**	**0.009**	**69%**
2. Clear best option.	**A**	**0.023**	**66%**	-			**D**	**0.047**	**63%**
3. Low stakes.	-			**A**	**0.015**	**67%**	-		
4. No good options.	**D**	**0.006**	**70%**	**D**	**0.005**	**70%**	**A**	**0.007**	**69%**
5. Option balance.	D	0.069	×	-			A	0.322	×
7. Difficult choice.	D	0.073	×	-			**A**	**0.009**	**69%**
8. Consequences.	-			-			**A**	**0.001**	**73%**

Table 7.3: Single-treatment results. Each entry has a letter indicating the hypothesis ('A' for agree or 'D' for disagree) followed by the p-value for that test. Significant entries ($p < 0.05$) are listed in bold, and indicate a common-language effect size (the percentage of comparisons supporting the hypothesis). Non-significant entries are highlighted in blue. Note that question 6 (the trick question) is omitted here.

Question	Hypothesis	p-value	Effect
1. No bad options.	**relaxed>dilemma**	$\mathbf{3.1\times10^{-4}}$	**76%**
2. Clear best option.	**obvious>dilemma**	$\mathbf{1.1\times10^{-4}}$	**78%**
4. No good options.	**dilemma>obvious**	$\mathbf{1.9\times10^{-7}}$	**88%**
	dilemma>relaxed	$\mathbf{1.2\times10^{-7}}$	**87%**
5. Option balance.	**dilemma>obvious**	**0.036**	**64%**
7. Difficult choice.	**dilemma>obvious**	$\mathbf{2.7\times10^{-5}}$	**80%**
	dilemma>relaxed	**0.001**	**73%**
8. Consequences.	dilemma>obvious	0.140	×
	dilemma>relaxed	$\mathbf{3.8\times10^{-4}}$	**75%**

Table 7.4: Between-treatments results. Each row indicates a hypothesis, the corresponding p-value, and the effect size if the result is significant ($p < 0.05$). Significant results are shown in bold; non-significant results are in blue.

half-refuting). By the same logic, if responses were all "somewhat agree," the effect size would be 70%, and if responses were all "strongly agree," the effect size would be 90% (the theoretical maximum effect size in the studies presented here). The effect sizes listed in table 7.3 range from 62% to 71%, which are moderate to strong effects. The effect sizes in table 7.4 range from 63% to 83%, with most being strong effects at $>$70% effect size.

Besides the single-treatment and between-treatment hypotheses, all three of the stakes hypotheses were confirmed. For the first (low-stakes choices would elicit agreement that their stakes were low) the p-value was 0.0047 and the effect size was 65%. The second stakes hypothesis (that high-stakes choices would elicit disagreement with the same statement) had a p-value of 0.0011 and an effect size of 70%. Finally, the backup hypothesis (that agreement would be higher in the low-stakes case than the high-stakes case) was a given as the first two were confirmed; it had $p = 9.7 \times 10^{-11}$ and an effect size of 83%.

7.5 Discussion

Out of the 25 specific hypotheses, 20 were supported by the data. This indicates that most of the perceptual qualities I expected given the constraints used to generate choices were in fact identified by most of the survey participants. In particular, the fact that all of the stakes hypotheses were confirmed indicates that *Dunyazad*'s author-based estimation of which player goals are more and less important is working. On a treatment-by-treatment basis, the observed properties were:

- "`obvious`" choices—Participants tended to agree that "`obvious`" choices had a clear best option (question 2) and they tended to disagree with the statement that they had no good options (question 4). I expected that participants would disagree that all of the options were equally promising

(question 5) and that these choices were difficult (question 7) but in both cases the data did not confirm these expectations.

- "`relaxed`" choices—Participants tended to agree that the stakes for these choices were low (question 3) and they tended to disagree with the statement that these choices had no good options (question 4). I expected participants to agree that these choices had no bad options (question 1) but the data did not support this hypothesis.

- "`dilemma`" choices—Participants tended to disagree that there were "no bad options" at these choices (question 1) and agree that there were "no good options" at these choices (question 4). Furthermore, they disagreed with the statement that these choices had a clear best option (question 2), and agreed that these choices were difficult and had important consequences (questions 7 and 8). However, I expected participants to agree that all options at these choices were about equally promising, but the data did not support this hypothesis.

Overall, the data support *Dunyazad*'s ability to control stakes and outcome valences, but also show that it has a bit of trouble making outcomes seem similar. The areas where its choices were able to produce the desired poetic effects are important successes for automatic reasoning about choice poetics, and they imply that the goals survey participants considered when judging options align at least somewhat with those the system assumed players would have.

Places were the data did not support my hypotheses are opportunities for further scrutiny. To start with, I wanted to investigate whether the failed hypotheses were the result of general trends across all choices generated under a treatment condition or whether any single choice contributed disproportionately to an unexpected result. To do this I broke down the results by individual questions within a treatment and plotted them to see if there was any indication of per-question differences.

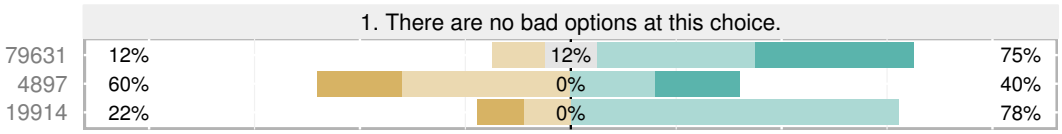

Figure 7.4: A graph of responses to question 1 under the "`relaxed`" treatment. The three numbers are the seeds used to generate the three different choices for this treatment. The graph setup is the same as in fig. 7.3.

7.5.1 Option Relativity

The first of my failed hypotheses involved the "`relaxed`" treatment. I expected the "`relaxed`" treatment to elicit agreement with the statement "There are no bad options at this choice," but it didn't do so definitively (the statistical test failed to reject the null hypothesis that the answers to this question were consistent with a uniform distribution of underlying responses). A per-choice breakdown of the data for the "`relaxed`" condition shown in fig. 7.4 gives a strong indication that the question with seed 4897 elicited qualitatively different responses than the two other questions in this treatment. That particular question is shown in fig. 7.5, and reveals one possible reason for what I observed: unlike the other two questions in the "`relaxed`" case, option three of this question lists "no relevant skills" rather than giving a relevant skill possessed by the player.

The fact that the player doesn't have any skills relevant to that action does not mean that the action will fail, but it might make that option seem less desirable than the others at that choice. None of the options at the other two choices in the "`relaxed`" treatment listed "no relevant skills," they all listed some skill that the player had as relevant, which explains why there might be a difference in responses. If that wording caused the shift, it would be consistent with Schwartz et al.'s theory of satisficing versus maximizing personalities (B. Schwartz et al., 2002) for real-world choices. Schwartz et al. have found that while some people are happy as long as their choices lead to satisfactory results, others are unhappy if their choices lead to good but nevertheless suboptimal results. The strong split

You come to a tavern and decide to rest for a while. A noble is bored and a peasant is bored and a merchant is selling a book of herbal lore. What do you do?

1. *You tell the peasant a story*
   (You have skill: storytelling).
2. *You tell the noble a story*
   (You have skill: storytelling).
3. *You offer to trade the merchant your dragon scale for the merchant's book of herbal lore*
   (no relevant skills).

Figure 7.5: The "`relaxed`" choice with seed 4897 (minus the framing, which is largely the same as that shown in fig. 7.2).

in responses for this specific case (including both significant "strongly disagree" and significant "strongly agree" contingents) indicates that some people may be interpreting the phrase "bad option" as meaning options that are absolutely bad, while others may be comparing the options against each other. It would take more data to discern whether this distinction is what is at work here, but it is clear that it is an important distinction for choice poetics, and it is not yet something that *Dunyazad* reasons about.

Although *Dunyazad* does not reason about this, it is to some degree aware of the distinction between the question with seed 4897 and the other two questions in that treatment. The constraints for the "`relaxed`" condition were that each option either "`enables`" or "`advances`" a goal (in the technical senses; see page 136 in item 3), and in this case, the system generated two options that "`advanced`" a goal and one that merely "`enabled`" a goal, thus creating a distinction even on its own terms. The other two questions in the "`relaxed`" category each included three options which "`advanced`" a goal. In light of the survey results, it is clear that to construct choices that unambiguously have "no bad options" the system should not only require that each option works towards a player goal, but that each option is balanced against all others.

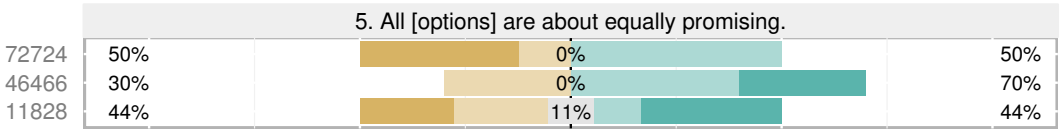

Figure 7.6: A graph of responses to question 5 under the "`dilemma`" treatment. The three numbers are the seeds used to generate the three different choices for this treatment. The graph setup is the same as in fig. 7.3.

7.5.2 Balancing Failures

Another unsupported hypothesis was that in the "`dilemma`" treatment participants would agree with the statement "All of the options at this choice are about equally promising." I expected this because one of the constraints of the "`dilemma`" treatment was that all of the threatened goals should have the same priority. However, as shown in fig. 7.6, even for the individual choice in the "`dilemma`" treatment where participants reported the most agreement, 30% of participants answered "somewhat disagree."

Figure 7.7 shows the options that participants said they would choose for the three dilemma choices. For the first two choices, option two is a clear loser, and looking at the choices, it's easy to see why. Both of those choices (which

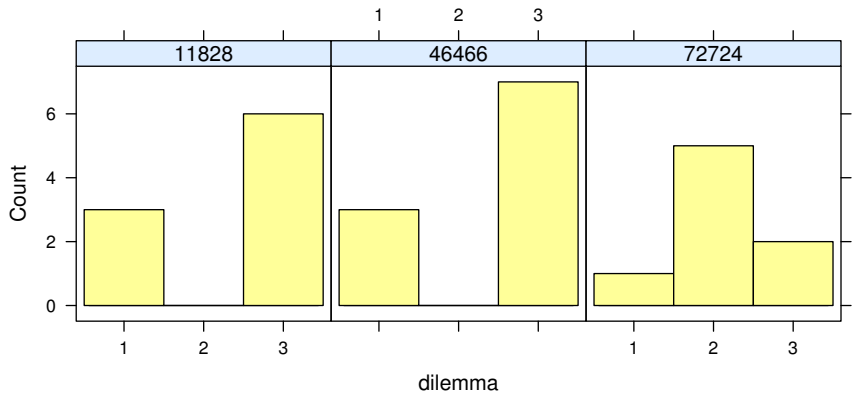

Figure 7.7: A histogram of options selected by participants at the three different "`dilemma`" choices (each labeled by seed).

are nearly identical) involve being attacked by a dragon (fig. 7.8 shows the first choice). In both choices, option two is an option to attack the dragon yourself, but of course you have neither the "fighting" skill nor a weapon, and the dragon has both. Although you also lack relevant skills for the other options, making a desperate attempt to flee from or pacify a dragon seems like a better choice than fighting it (at the very least, it did to all of my participants).

There are at least three factors that might contribute to the system's divergence from players' analysis of these choices. The first has to do with the granularity of expectations, the second with the granularity of goal priorities, and the third with stacking goals.

First, there are some clear arguments available to players as to why fighting might be a worse option than fleeing (for example) in an unfavorable situation. One is that fighting presents a *greater risk* of a bad result, and another is that fighting is directly related to a *categorically worse* result. The first argument has to do with the granularity of expectations: *Dunyazad* just recognizes events as "`unlikely`," "`neutral`," or "`likely`." In this case, it reasons that since both fleeing and fighting are "`likely`" to "`fail`" the goal of avoiding the present threat, both options fail a high-priority goal. However, given that all options are marked negatively, the prospect of fleeing evokes more hope for unwarranted success than

As you continue your journey, a dragon swoops down from the skies, scales glinting in the sun. It is threatening you. What do you do?

1. *You try to flee from the dragon*
   (You are missing skill: wilderness lore. It has skill: wilderness lore).
2. *You attack the dragon*
   (You are missing skill: fighting. It has skill: fighting. It has some claws).
3. *You attempt to pacify the dragon with music*
   (You are missing skill: musician).

Figure 7.8: The "`dilemma`" choice with seed 11828 (minus the framing, which is largely the same as that shown in fig. 7.2).

the prospect of fighting. This could be expressed in *Dunyazad* by introducing evaluations like "somewhat_likely" and "very_likely" that would distinguish these cases. Of course, the "skill_link" mechanism would have to be updated to give estimates of these levels of likelihood in different circumstances.

Another argument a player could make as to why fighting is worse than fleeing in this situation has to do with the relative magnitude of the results. Internally, *Dunyazad* recognizes that the "attack" option is likely to "fail" both the "avoid_threats_to" and "preserve_health" goals, while the "flee" option is only expected to "fail" the "avoid_threats_to" goal. However, as *Dunyazad* is written now, it treats an option that indicates one high-priority goal failure no differently form one that indicates multiple failures. Particularly for choice structures which are supposed to have balanced options, some kind of counting logic would be useful to determine when one option is better or worse than another, even when they're in the same general category.

In the same vein, one could argue that even without counting how many goals fail, the "preserve_health" goal should be higher-priority than the "avoid_threats_to" goal. In part because of complexity concerns (although I have not tested this extensively) I made a decision to limit *Dunyazad* to two priority levels. One could imagine instead a partially directed "more-important-than" graph between all player goals at each timepoint, which would be another way for *Dunyazad* to realize that fighting is worse than fleeing in this case.

The problem here is that the system's representation of player expectations is not fine-grained enough. To the system, all of the options at these choices are expected to "hinder" the player's goal of keeping their character alive and uninjured, but the system makes no distinction beyond that. How certain does the failure of these goals seem to the player? Exactly how badly does the player expect to fare when their goal is not met? In this case, even when told that the situation is hopeless (or perhaps especially then), fleeing seemed a better option

You come to a tavern and decide to rest for a while. A merchant is bored and a noble is bored and an innkeeper seems knowledgeable. What do you do?

1. *You play a song for the noble*
   (You have skill: musician. You have no tool for music).
2. *You gossip with the innkeeper*
   (You are missing skill: negotiation).
3. *You play a song for the merchant*
   (You have skill: musician. You have no tool for music).

Figure 7.9: The "`dilemma`" choice with seed 72724 (minus the framing, which is largely the same as that shown in fig. 7.2).

than attacking the dragon head-on, but the system doesn't distinguish those cases. Based on this data, the system should be improved by adding more detail to its assessments of goal failure and success.

Any one of these corrections would help this particular case, but especially given that *Dunyazad* also has difficulty producing balanced positive options as mentioned in the previous sub-section, I suspect a more focused approach would be better. After all, the likelihood reasoning *is* doing a good job of predicting which options players will view positively or negatively; it just has a hard time figuring out whether options are equally positive or equally negative. As fig. 7.7 shows, even the last dilemma choice (shown in fig. 7.9 produced a somewhat skewed pattern of player decisions. Even though the reason for players to have a preference may be different for this choice (perhaps the difference in tool availability?) the choices here were also not perceived as balanced (see fig. 7.6). Rather than further overload the current reasoning, an additional subsystem dedicated to direct relative analysis of options should be added to address problems like these. This system could separately maintain more-nuanced representations of goal priorities and outcome likelihoods, and then build arguments as to why each option is more- or less-desirable than each other option. In many cases, this nuanced relative analysis could be short-circuited when the old low-fidelity analysis detects a clear difference.

In terms of the choice analysis method presented in chapter 5, this exact problem was already mentioned in section 5.4.6 (see the second paragraph where it talks about the balance of impacts). I hypothesized here that *Dunyazad* would be able to fake a more detailed analysis by just insuring that the threatened goals had equal priorities, but that turned out not to be true. In this case, what was learned from *Dunyazad*'s failure backs up the theory's point about balancing impacts: dilemmas are a case which needs closer inspection. Getting this experimental result is still useful, because it provides a concrete example of why this is the case.

7.5.3 Outcome Clarity

Not only did I expect participants to agree that options were balanced for "`dilemma`" choices, but I also expected them to disagree for "`obvious`" choices. Here again my hypothesis was not supported by the data. A per-choice analysis of question 5 in the "`obvious`" condition (the top half of fig. 7.10) shows that again, one choice is divergent from the other two. Figure 7.11 shows that middle choice, which compared to the other two "`obvious`" choices is low-stakes (one of the other choices starts with a dragon attack, the other with bandits attacking some merchants).

Not only are the stakes for this choice low, but they are unclear. What exactly does the player hope to gain by gossiping or by telling a story? Perhaps friendship or some useful information, but those potential rewards seem both uncertain (even given a "successful" action) and questionably useful. In contrast (albeit a contrast that participants did not see directly) the utility of fleeing from an attacking monster is clear, even if it is uncertain whether you will succeed. Furthermore, there aren't any obvious risks associated with options 1 and 3, so even if the player is missing a relevant skill, they might still be worth trying.

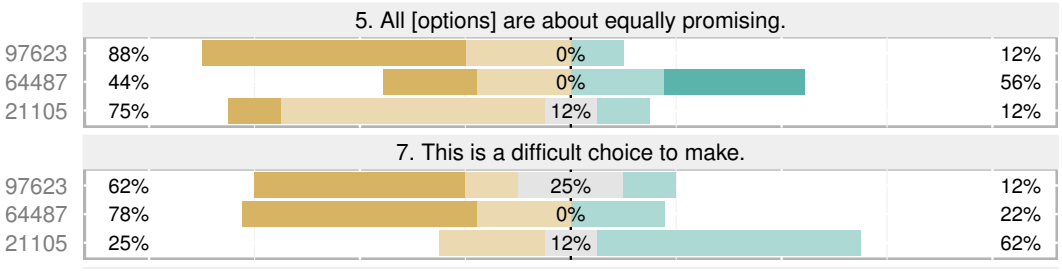

Figure 7.10: A graph of responses to questions 5 and 7 under the "`obvious`" treatment by seed. For question 5, the choice with seed 64487 is clearly different from the other two, and for question 7, the choice with seed 21105 is the one that stands out. Figures 7.11 and 7.12 show the text of the divergent choices.

You come to a tavern and decide to rest for a while. A merchant is bored and a peasant seems knowledgeable and an innkeeper seems knowledgeable. What do you do?

1. *You gossip with the innkeeper*
   (You are missing skill: negotiation).
2. *You tell the merchant a story*
   (You have skill: storytelling).
3. *You gossip with the peasant*
   (You are missing skill: negotiation)

Figure 7.11: The "`obvious`" choice with seed 64487.

You come across some bandits attacking a merchant. The bandits are threatening the merchant. What do you do?

1. *You attack the bandits*
   (They have skill: fighting. You are missing skill: fighting. They have no tool for fighting).
2. *You travel onwards*
   (no relevant skills).
3. *You talk the bandits down*
   (no relevant skills).

Figure 7.12: The "`obvious`" choice with seed 21105 (discussed in section 7.5.4).

Given this combination of low stakes, a dubious reward for the most-successful-seeming option, and seemingly consequence-free options all around, it is not hard to see how some might find these options "about equally promising."

On the other hand, the system's attempt to construct an obvious choice in this case was still somewhat successful. 7 of 9 participants who saw this choice "strongly agreed" with the statement "There is a clear best option at this choice" and 8 of those 9 picked option #2 as the option they would choose. While it might seem like a contradiction to agree (as 3 participants did) with both the statement that a choice has equally promising options and the statement that it has a clear best option, this highlights the difference between outcomes-focused evaluation of individual options and choice-oriented option comparison. The phrasing of "All of the options at this choice are about equally promising," suggests evaluating the utility of each option independently and comparing those utilities. In contrast, "There is a clear best option at this choice," suggests comparing options against each other directly to find one that is better than the others. The fact that people often make decisions inconsistent with simple utility calculation is well-known (see e.g., (Tversky and Simonson, 1993)), so it should not be surprising that a context in which someone is asked to make a decision might elicit a different response than a context in which someone is asked to rate responses.

The implications for choice poetics are interesting, because choice poetics is concerned with both mindsets. At least in terms of impact on the player, there's clearly a difference between a choice where all of the options seem "about equally promising" and where that's definitely not the case, even if both choices have "a clear best option." Choices where options seem equally promising but most players choose a particular route regardless might even be good candidates for the focus of regret. For example, if it turns out that the "clear best option" leads to failure and players are forced to revisit the original decision, having seen some potential in the alternatives they're more likely to feel that the choice

was fair, and thus presumably blame their own decision for the consequences (section 8.6.5 discusses some data relevant to this point that resulted from the experiment on outcome evaluations). At the same time, if the designer can be confident that most players will go down the "obvious" route first even without making the contrast with the other options huge, they can assume that most players will experience the regretful path as opposed to choosing the ultimately correct option the first time.

Despite the interesting revelations prompted by this case, *Dunyazad* does actually have a problem here: it is analyzing outcomes according to the game mechanics that it knows about, but the players in this case aren't playing a game: they're experiencing a single choice with very little context. If someone had played a few of *Dunyazad*'s output stories already, they'd have a view of the possibility space much more similar to *Dunyazad*'s, especially in terms of the outcomes of actions like "`gossip`" and "`tell_story`." If the players knew what to expect from both success and failure at these actions, and had a stronger sense of the role skills play in determining outcomes, they wouldn't get their hopes up, and the choice *would* be "obvious."

That is the deeper issue here as well: the simple notions of "success" and "failure" when performing an action often don't correspond to overall positive and negative results when an action isn't clearly oriented towards some pressing player goal. For example, an attempt to "gossip" despite lacking the relevant skill seems like it might still yield interesting results, and it isn't likely to be disastrous. The same is not true of attacking an enemy: when they're more skilled or better-armed, there's a clear sense of danger (and of what is at stake). In other words, *Dunyazad* is doing a good job of manipulating player expectations when it is heavy-handed, threatening high-priority player goals to make options appear hopeful or doomed. However, when *Dunyazad* attempts to use low-priority

goals in the same fashion, players don't follow its lead: "My gossiping probably won't go 'as intended?' Well, it might still be awesome!"

The simplest response to this is to just tread carefully when using low-stakes choices and assume that some of them aren't going to get the result you want, while making sure to use high-stakes choices if a particular property like obviousness is critical. In the longer term, *Dunyazad* should recognize things like the desire to explore rather than always travel the beaten path (especially when the stakes are low). At the same time, one area for future work is goal-probing questions, which would allow *Dunyazad* to ask the player (either directly or indirectly) to indicate their goals. These questions would serve two purposes: first, they would allow *Dunyazad* to dynamically determine which goals (including low-priority goals) the player thinks are important. Second, these questions, when successful, may subtly influence players by committing them to a goal: one a player affirms a goal, they may be more likely to behave in a manner consistent with that goal in the future (see e.g., (Hall, Johansson, and Strandberg, 2012) on the powerful instinct for post-hoc justification when an opinion is perceived—even incorrectly—to be one's own).

7.5.4 Conflicting Goals

My hypothesis that respondents would not find "`obvious`" choices difficult highlights a different choice in the "`obvious`" category. The data did not support this hypothesis, and as shown in fig. 7.10, the choice with seed 21105 accounts for the majority of all responses that contradict it. That choice is shown in fig. 7.12, and from the system's perspective, it satisfies the definition of an obvious choice (see table 5.3) because the second option "`advances`" a player goal, while neither the first nor the third do, and both the first and third "`threaten`" a player goal.

The perceived difficulty of this decision probably stems from the fact that it pits two player goals against one another: the goal of self-preservation is

best served by option two, but the goal of helping others in need is best served by one of the other options. This is similar to the lack of distinction between threatening one and two goals encountered above. Even when one option at a choice clearly has the most-positive outcome for the player considering all goals to be equal, when that choice pits multiple goals against one another, it may be very difficult indeed. In fact, choices with these structures may give rise to an entirely different set of poetic concerns that involves moral judgements (this is exactly how morality thought experiments like the "trolley problem" are constructed, for example—see (Thomson, 1976)). A difficult decision between two desirable or undesirable outcomes feels completely different from a decision between two outcomes each justified by competing moral principles, and it can resonate powerfully to the broader narrative structures of a story. In its current state, *Dunyazad* doesn't reason about morality at all: it makes no distinction between goals in terms of their root motivations, and it has no compunction about how a result is achieved as long as no player goals are threatened in the process. While *Dunyazad* gets perhaps surprisingly far without these capabilities, they're clearly important and can come up by accident even when *Dunyazad* isn't trying to make use of them.

That said, a detailed inspection of the answer set that resulted in this choice reveals another problem with the system: ignoring the lack of moral reasoning, it's simply not working as intended. In this case, *Dunyazad* actually thinks that travelling onwards serves the goal of preventing the threat to the merchant (because if the player travels onwards, that entire situation is left behind and therefore the threat no longer exists) while it has no conception that this serves a goal of self-preservation (although it understands that interfering by either means threatens the player's safety). There are thus two more changes to the system suggested by this data: first a bug-fix related to the consequences of travelling onwards, and second, the addition of relative goal relevance across

options: the idea that if all but one option threatens an important goal, then the remaining option can be seen as indirectly supporting that goal even if none of its outcomes directly further that goal. Without running this experiment, I would eventually have found and fixed the "travel onwards" bug, but I may not have thought to make the second change. In this case, the data served to help find and diagnose an anomaly in my system, which turned out to involve an error in the code, but at the same time also pointed to two future goals for the choice structure model: adding explicit moral reasoning, and a notion of relative goal relevance when all but one option relates to a goal in the same way.

7.5.5 Stakes and Consequences

The final unsupported hypothesis was that participants would feel more strongly that "`dilemma`" choices had important consequences than that "`obvious`" choices did. This hypothesis was based on the idea that consequences might seem more relevant (and thus important) when a decision was more difficult. Given that one of my "`obvious`" choices seemed difficult to many participants as discussed above, it is unsurprising that this hypothesis was not supported.

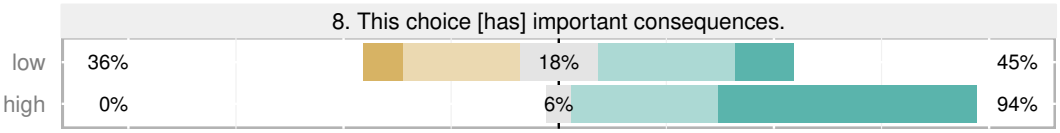

Figure 7.13: A graph of responses to question 8 across all treatments by stakes.

What is interesting is the effect of the choice stakes on this question. The similar hypothesis that participants would agree more in the "`dilemma`" case than the "`relaxed`" case *was* supported by the data, and one big difference between those two treatments is the stakes associated with them.[4] In fact, when analyzing

[4] A statistical analysis of the assertion that participants agree more with the statement that a choice's stakes are low in the "`relaxed`" case than the "`dilemma`" case (not one of my original hypotheses) reveals a strong significant effect ($p = 2.88 \times 10^{-5}$; effect size 76%).

the data for question 8 according to stakes rather than the three treatments, high-stakes choices overwhelmingly seem as if they will have important consequences, while low-stakes choices are mixed (see fig. 7.13).

Statistics confirm the obvious: not only did the high-stakes condition elicit significantly more agreement on question 8 than the low-stakes condition ($p = 2.14 \times 10^{-8}$; effect size 79%), it was also significantly above a uniform distribution ($p = 4.9 \times 10^{-6}$; effect size 78%). Because all of the "`relaxed`" choices were low-stakes by design, there is of course correlation between the stakes and the treatments, but the 35 high-stakes cases were about evenly distributed between the "`obvious`" and "`dilemma`" treatments (16 in "`obvious`" and 19 in "`dilemma`"). Such an overwhelming effect (none of the 35 respondents who saw a high-stakes choice thought it would *not* have important consequences) further indicates that the system is successful in predicting high-stakes player goals: for 94% of choices where the system thought that an important player goal was affected, players agreed at least somewhat that the consequences seemed important.

7.6 Conclusions

Overall, this study confirmed *Dunyazad*'s ability to construct choices based on player expectations when certain criteria are met. Notably, many of the failed hypotheses involved situations where several options together affected how a choice was perceived:

- For question 1 ("There are no bad options at this choice.") relative rather than absolute judgements of what is a "bad" option seem to have been employed by some participants.

- For question 5 ("All of the options at this choice are about equally promising.") *Dunyazad* may need to make finer distinctions between different

modes of failure, as several options expected to lead to failure may still seem to offer a range of possibilities when no better options are present.

- Question 7 ("This is a difficult choice to make.") showed that a choice can be difficult when multiple goals conflict, even when expectations for one goal are much better than for another.

These results point to several possible improvements for *Dunyazad*, and collectively reinforce the importance of considering choices holistically when evaluating choice poetics.

As already discussed, there are a number of changes that could be made to *Dunyazad* based on these results:

- Implement separate "satisfaction" and "maximization" player decision modes so that the system can reason about how difficult a decision might seem, whichever decision modality a player is using.

- Upgrade *Dunyazad*'s system for estimating how individual options affect goals by adding more detail so that it can further distinguish different magnitudes of risk and reward.

- Have *Dunyazad* represent players' uncertainty about the possible outcomes of actions like "gossip" and "tell story" so that it can have a clearer picture of which options seem promising in low-stakes situations.

- Implement goal-probing questions (either direct or indirect) that allow the system to gain direct knowledge about player goals, especially for low-stakes options. This would also allow *Dunyazad* to further support role-playing by allowing different players to choose options which imply different goals or goal priorities.

- Implement a model of goal conflicts and more detailed relative goal priorities including moral reasoning.

- Implement the idea of relative goal relevance: if all but one option at a choice either threatens or enables a goal, then the remaining option implicitly does the opposite.

These changes could make *Dunyazad* even more successful at constructing choices which produce particular player expectations (and thus which can produce specific poetic effects). They are also in line with the goal of getting *Dunyazad* to generate choices with more complex poetics, such as choices that elicit regret or choices that prompt deliberation.

One important functionality of *Dunyazad* not tested in this first study was its ability to judge outcomes. This study did largely verify that *Dunyazad*'s player expectations about options are working, and it can generate outcomes that both support and betray those expectations. The experiment described in chapter 8 relies on this functionality to test how accurate *Dunyazad*'s estimates of retrospective perceptions are.

Chapter 8

Experiment II: Retrospective Impressions

This chapter describes results from a second study involving *Dunyazad*, this time scrutinizing player's impressions of a choice *after* making a decision. The setup is largely the same as the first experiment, so reading chapter 7 will help to understand the results presented here. In particular, section 7.2 describes the experimental procedures used in the first experiment, which were largely the same in this experiment, and section 7.3 describes how hypotheses were tested in the first experiment, which was also largely replicated in this experiment.

Key results echo those from chapter 7 in that *Dunyazad* needs to be able to reason better about conflicting goals and use more detailed goal priorities. Additionally, results relevant to outcome perspectives show that *Dunyazad* might do well to adopt a strategy that reasons explicitly about (and/or explicitly constructs) option implications. The current system which automatically assumes option implications based on *Dunyazad*'s internal representation of actions and their potential consequences in some cases does not align with the expectations that players seem to develop based on the actual text generated for the options. These

results and others, along with the results from the experiment described in chapter 7, have implications for the goal-based choice analysis technique presented in section 5.4 as well, and these are discussed in section 8.11.

8.1 Overview

Because the results from testing *Dunyazad*'s prospective impression subsystem were largely successful, I ran another experiment to test the retrospective impression system. In this experiment both prospective and retrospective impression constraints were used to create six kinds of choices defined in terms of their prospective and retrospective impressions (see tables 5.3, 5.4 and 6.3):

- **Expected Success**—These choices have three options which present themselves as leading to success, and each option has a successful outcome. In terms of "`outcome_feel`" values (see table 6.3) each option at these choices is an "`expected_success`" option. No matter which option a participant selects, they will be picking an option that suggests success, and they will get a successful outcome (as far as *Dunyazad* understands things).

- **Expected Failure**—The opposite of expected success: every option suggests and results in failure. Internally, each option is an "`expected_failure`."

- **Unexpected Failure**—These choices consist of three "`unfair`" options. In other words, each option seems as though it will be successful, but each ultimately results in failure. These choices should appear largely the same as "`expected_success`" choices before an option is chosen.

- **Unexpected Success**—The opposite of unexpected failures, these choices consist of three "`miracle`" options—each indicates failure but results in success. These choices mimic "expected failure" options until the player selects an option and sees an outcome.

- **Obvious Success**—These choices have a single "`expected_success`" option and two "`expected_failure`" options, forming an "`obvious`" "`option_structure`." The outcomes of each choice align with what the option text suggests: the "good" option leads to success and the others lead to failure.
- **Obvious Failure**—These choices have a single "`unfair`" option and two options which are either "`miracle`"s or "`nice_gamble`"s. Just like "obvious success" choices, their "`option_structure`" is "`obvious`." Of course, instead of leading to the expected results, each option results in the opposite of what it suggests.

These six choice structures were designed to manipulate three variables: valence of outcomes, expectedness of outcomes, and the presence of viable alternatives. Note that due to the study design, each participant only ever saw a single choice, they could not form judgements based on comparing choices across categories. The bulk of the hypotheses were designed simply to verify that the participants agreed with *Dunyazad*'s internal evaluation of these variables, but there were a few hypotheses designed to probe for possible relationships between these six conditions. Overall the results show that *Dunyazad* did a good job producing positive outcomes, but struggled with negative outcomes, and (partially as a result of this) had some trouble producing unexpected results. Despite these problems, the data reveal some interesting interactions between the three underlying variables, such as a difference in satisfaction based on the presence or absence of viable alternatives and a relationship between the valence of unexpected outcomes and their perceived fairness.

8.2 Method

The experimental setup for this experiment was largely the same as the setup for the prospective impressions experiment described in chapter 7. This section lists

the major differences between the two setups but does not reiterate the details of the original setup (see section 7.2) that were unchanged. The first obvious difference was the sample size. As already mentioned, there were six types of choice generated, rather than 3, for a total of 18 choices (three per condition, just as in the original experiment). Instead of showing each choice to 10 subjects, each choice was shown to 15 subjects, which meant that there were nominally 270 participants instead of 90. Luckily the use of Amazon Mechanical Turk means that there is very little effort per participant, so the logistics of the second study weren't drastically different from those of the first.

8.2.1 Extra Constraints

In an attempt to avoid some of the problems with low-stakes choices in the original experiment, every choice in this experiment was required to have high stakes. Furthermore, rather than allowing *Dunyazad* to pick any setup it wanted, the three choices in each condition were generated using fixed setups ("`market`," "`monster_attack`," and "`threatened_innocents`"). This helped insure that the setup text wasn't a significant variable between the cases, and that each case would include a variety of setups. Setup parameters were still allowed to vary freely; the "`market`" setup in particular has multiple possible configurations.

8.2.2 Framing

Unlike the first survey, where participants were asked to pick the option that they "would have chosen," participants in this study were instructed: "Please read the following short story which presents you with a choice, make a decision, and then answer the questions about that choice below." The choices were presented with a radio button next to each option, and hovering over the text of each option highlighted it. When the participant clicked anywhere over the text of an option,

the outcome text for that option would be displayed in the space directly below it (moving any following options down). At the same time, the selected option text and the outcome text were highlighted in bold and the radio button for the selected option was filled in, while the non-selected options were faded to gray and their radio buttons were disabled. Once an option was selected the highlight-on-hover and click-to-choose functionality were disabled. Immediately before the option texts the phrase "To choose, click on an option below. Once you make a decision you will not be able to change it," appeared to make sure that participants understood the format. The option selected by the player was recorded as part of the study data; every single participant selected an option and thus saw an outcome.

Of course, although this decision mechanism mimicked the decision-making environment of modern browser-based narrative choice games, it meant that participants would see the outcome of their decision before answering questions about the options that they had. This approach was chosen intentionally to maintain the flow of the setup into the choice; asking a set of survey questions after seeing the choice but before making a decision would have an impact on the poetics of the choice. Of course, having seen an outcome inevitably changes one's perception of an option (especially if that outcome was unexpected). The data in the first survey therefore is more informative about option-related poetics. To counteract this somewhat, the following text appeared before the option questions: "For these questions, consider just the options presented, and ignore the outcome of the option that you chose." Complying with such a request is not entirely possible, of course, but the option-related questions in this survey were only present to establish a point of comparison with the first study, and were not the focus of the results.

8.2.3 Compensation

Participants were still Amazon Mechanical Turk workers, but for this study, each participant was paid $2.00 instead of $0.50. This was partially because this study involved more questions, and partially because of the desire to pay more fairly for participants' time. Based on the quality of the responses that I received and on reviews of the tasks posted on `https://turkopticon.ucsd.edu/`, I believe that this level of pay was satisfactory both from my own perspective and from the perspective of the participants. The total cost of the experiment including fees paid to Amazon and the costs of pilot tasks (4 pilot tasks were posted to debug the survey before posting the 270 main tasks) was $765.20.

8.2.4 Questions

Of course, one major difference between the studies was the set of statements included. This study had three sections: statements about the options, statements about the outcome, and statements about motivation. The option statements were actually a subset of the statements from the original survey, with extra text to encourage participants to think only about the options and ignore the outcome as mentioned above. The following five option statements (listed by their internal labels) were included:

Clear best 1. *"Considering just the options, there seems to be a clear best option at this choice."*

Balanced 2. *"Ignoring outcomes, the options at this choice all seem about equally good (or bad)."*

No bad 3. *"Ignoring outcomes, there are no options that seem bad at this choice."*

No good 4. *"Ignoring outcomes, none of the options at this choice seem good."*

Low stakes *5. "Considering just the options, the stakes for this choice seem low."*

In addition to option statements, this survey included statements about the outcome that participants saw:

Fair *1. "Given the options available, the outcome I got is fair."*

Sense *2. "The outcome that I got makes sense given the option that I selected."*

Bad *3. "I got a bad outcome."*

Satisfied *4. "I'm happy with the option that I chose."*

Unfair *5. "The outcome that I got is unfair, given the options available."*

Unexpected *6. "The outcome that I got is completely unexpected."*

[Trick] *7. "There is an outcome." (This is a trick question to test whether you're paying attention. Please simply indicate that you are in complete* disagreement*.)*

Broken *8. "There might be a problem with this choice—the outcome I got does not make sense."*

Good *9. "The outcome that I got is a good outcome."*

Expected *10. "I pretty much expected the outcome that I got."*

Dissatisfied *11. "I wish I had chosen a different option."*

Underlying these statements were 5 concepts which were each addressed by a pair of statements with counterbalanced answers. These underlying concepts were: **fairness** (questions 1 and 5), **sense** (questions 2 and 8), **valence** (questions 3 and 9), **satisfaction** (questions 4 and 11), and **expectedness** (questions 6

and 10). The order of the questions was scrambled and chosen such that no pair of complimentary statements would appear adjacent to each other in order to encourage subjects to approach each question individually.

Although the outcome questions were the main focus of this study, some questions asking participants to self-report their motivations were added at the end of the study to get direct feedback about modes of engagement. These questions were prefaced with the prompt:

> *Please answer the following questions about your motivations for making the decision you made, as well as about your motivations and goals in general when playing/reading interactive stories.*

The first question asked directly about motivation:

> *1. Which of the following motive(s) contributed to your decision? (pick one or more)*
> *(Note: no submissions will be rejected based on this information; wanting to get this survey done quickly is a perfectly valid reason for making a decision.)*
>
> - ❑ *I'm taking an online survey. I just chose an option quickly so that I could complete the survey.*
> - ❑ *I was just curious to find out what would happen if I chose the option I did.*
> - ❑ *I imagined a character in the story situation and chose what that character would do.*
> - ❑ *I chose what I would have chosen were I in the situation described in the story.*
> - ❑ *I chose the option that I thought would lead to the most interesting/satisfying result.*

- ❏ *I looked at the skill information and chose the option that I thought would be most successful.*
- ❏ *Other (please explain):*

A free text area was provided below the "Other" option. As suggested in the prompt, participants were free to select more than one option or leave them all blank. Note that the options here mostly correspond to the modes of engagement proposed in chapter 5. Of course, by providing exactly these options players may be subconsciously coerced into aligning their motivations with the archetypes that I have defined, and so the results of this study shouldn't be taken to be a confirmation that player motivations *do* fit into the theoretical framework discussed above. However, the relative frequency of different responses still provides some information about whether players' motives are diverse or generally similar, for example. The responses to this question were labelled respectively as `speed`, `curious`, `role`, `avatar`, `interest`, `power`, and `other`.

In a similar vein, the next question asked how players evaluate outcomes:

2. Which of the following judgement(s) contributes to how you generally define a "good" outcome in interactive experiences like the one you just played? (pick one or more)

- ❏ *I feel an outcome is good when something good happens to my character in the story world.*
- ❏ *I feel an outcome is good if it is an interesting development in the story.*
- ❏ *I feel an outcome is good when it fits the role that I am building for my character.*
- ❏ *I feel an outcome is good when it makes my character more powerful.*

- ❑ *I feel an outcome is good when it makes progress towards beating a game.*
- ❑ *Other (please explain):*

These responses were labelled respectively as `avatar`, `interest`, `role`, `power`, `progress`, and `other`. Directly following was effectively the same question but for "bad" outcomes:

> *3. Which of the following judgement(s) contributes to how you generally define a "bad" outcome in interactive experiences like the one you just played? (pick one or more)*
>
> - ❑ *I feel an outcome is bad when something bad happens to my character in the story world.*
> - ❑ *I feel an outcome is bad if it doesn't develop the plot.*
> - ❑ *I feel an outcome is bad when my character doesn't do what I expected them to do.*
> - ❑ *I feel an outcome is bad when my character expresses a value or opinion that is different from what I wanted them to express.*
> - ❑ *I feel an outcome is bad when it makes my character less powerful.*
> - ❑ *I feel an outcome is bad when it prevents me from making progress towards beating a game.*
> - ❑ *Other (please explain):*

These responses were labelled as `avatar`, `interest`, `no.control`, `value.clash`, `power`, `progress`, and `other`.

The final question in the motives section asked about consistency of motives. Unlike the other questions in this section, it forced participants to select a single response (although it still had an "other" response):

4. Do you feel you approach all interactive experiences (e.g., Choose-Your-Own-Adventure novels, video games, tabletop role-playing games, etc.) with a consistent set of motivations and judgements, or do your motivations and judgements change from story to story?

- ❍ *I feel that my motivations and judgements are the same no matter what interactive story I'm playing.*
- ❍ *I feel that my motivations and judgements change from story to story.*
- ❍ *I don't play interactive stories often enough to have a definite answer.*
- ❍ *Other (please explain):*

The responses to this question were labelled `consistent`, `variable`, `no.exp`, and `other`. Although the questions in the motivation section weren't subject to any statistical tests, they provide some informal information about how players approach choices. A further experiment designed to seriously investigate player motivations would require a much tighter design, and would likely derive no benefit from using *Dunyazad* to generate its choices, as *Dunyazad* isn't designed to be aware of, or work with, modes of engagement.

Unlike the first survey, this survey included an optional text area at the end for any additional feedback participants wanted to share. This additional feedback was not analyzed, but responses were used to help with approval decisions (several participants indicated honest confusion about some aspect of the age/fluency question; their responses were accepted).

Besides these differences, the details of this survey were the same as the first survey, including the format of response options and the framing of the various sections. Of course, this survey included a different set of hypotheses.

8.3 Hypotheses

The hypotheses in this study were tested in largely the same way as those in the prospective study: Each "agree" or "disagree" hypothesis was tested by comparing the results under a given condition with a uniform distribution of responses using a Mann-Whitney-Wilcoxon U test with an appropriate alternate hypothesis. A uniform distribution with 35 elements was used for all comparisons, as this was the multiple of five (the number of response categories) closest to the number of samples in most of the relevant cases (32-36 for each condition, although as low as 8 in some sub-conditions; these low-n comparisons are discussed where they appear). For between-condition hypotheses, the two sample populations in question were compared directly using Mann-Whitney-Wilcoxon U tests. In all cases, results are taken to be significant when their p-values are <0.05, and common-language effect sizes (see section 7.4 for an explanation) are indicated wherever the results are significant.

Question	Expected Success Unexp. Failure	Obvious Success Obvious Failure	Expected Failure Unexp. Success
Clear best	disagree*	agree	disagree*
Balanced	agree*	disagree	agree*
No bad	agree*	disagree	disagree
No good	disagree	disagree	agree
Low stakes	disagree	disagree	disagree

Table 8.1: Hypotheses about option statements in the retrospective study. Each entry corresponds to two hypotheses: one for each condition listed in the header of that column. Because the two conditions in each column share option structures, they are predicted to elicit identical responses to the option-related statements. Low-confidence hypotheses are marked with a '*'.

8.3.1 Option Hypotheses

The three option structures used in this study were "positive_alternatives," "negative_alternatives," and "obvious," as described above. Each option structure corresponded to two conditions with different outcomes. These paired conditions are listed in the header of table 8.1; each option structure was predicted to yield a different set of answers to the five option statements, as shown in that table. These 30 hypotheses (5 statements $\times$ 3 option structures $\times$ 2 conditions per structure) were meant to check whether *Dunyazad* was producing the desired option structures successfully.

Given the results of the first experiment, several of these hypotheses were not expected to be supported by the data. These tentative hypotheses are marked with '*'s in table 8.1. In particular, hypotheses having to do with options being balanced or there being no clear best option were not expected to be validated.

8.3.2 Basic Outcome Hypotheses

Assuming that *Dunyazad* is able to construct options which indicate either success or failure as intended, and that players agree with *Dunyazad*'s evaluations of outcomes, there are four different types of option in this study. Each option indicates either failure or success, and each outcome is either largely successful or largely unsuccessful. The "expected_success" condition, for example, consists of three options which both indicate and lead to a good outcome (or at least, *Dunyazad* has constructed them with that intent). The "obvious_success" and "obvious_failure" conditions are more complicated than the other four, which each consist of three options of the same type. "obvious_success" has one option which suggests and results in success, but two that suggest and lead to failure. Likewise, "obvious_failure" has one option which suggests success but leads to failure, and two that suggest failure but lead to success. This leads to the

Question	Expected Success Obv. Success [main]	Unexp. Success Obv. Failure [alt]
Fair	agree	agree
Unfair	disagree	disagree
Sense	agree	agree
Broken	disagree	disagree
Good	agree	agree
Bad	disagree	disagree
Satisfied	agree	agree
Dissatisfied	disagree	disagree
Expected	agree	disagree
Unexpected	disagree	agree
Question	**Expected Failure Obv. Success [alt]**	**Unexp. Failure Obv. Failure [main]**
Fair	agree	disagree
Unfair	disagree	agree
Sense	agree	disagree
Broken	disagree	agree
Good	disagree	disagree
Bad	agree	agree
Satisfied	-	disagree
Dissatisfied	-	agree
Expected	agree	disagree
Unexpected	disagree	agree

Table 8.2: Outcome-related hypotheses for the retrospective study. Each column lists two conditions in each half of the table; these conditions have the same expected and actual outcome valences, and are thus predicted to elicit the same responses. Eight conditions are listed here because the "`obvious_`" conditions have sub-cases: [main] for participants who chose the "best" option and [alt] for participants who chose otherwise. These sub-cases arise because participants experience outcomes with different valences depending on the option they choose.

[main] and [alt] sub-cases, [main] indicating participants who were shown an "`obvious_`" choice and chose the "best" option, and [alt] indicating participants who chose a different option at these choices.

Table 8.2 lists all of the outcome-related hypotheses. As with table 8.1, each cell corresponds to two hypotheses: one for each condition/case listed at the top of its column. The top half lists hypotheses about ultimately positive outcomes, while the bottom half lists hypotheses about negative outcomes. At the same time, the left column covers expected results (where the option indicates a result of the same valence as the outcome) and the right column covers unexpected results. The third underlying variable in this study, namely, whether the options at a choice seemed to indicate similar or divergent outcomes, was not expected to cause any "disagree" predictions to become "agree" predictions or vice versa, which is why each cell in this table can list two hypotheses. Table 8.2 represents a total of 76 individual hypotheses, which makes a total of 106 flat "agree" or "disagree" hypotheses when combined with the 30 hypotheses in table 8.1. Taken together, these hypotheses are designed to ensure that *Dunyazad*'s choices have the properties *Dunyazad* assumes they have in terms of both what is suggested by their options and in terms of how their outcomes are interpreted.

8.3.3 Relative Outcome Hypotheses

As with the first study, there are some hypotheses about the relative agreement between two different statements as opposed to just agreement with or disagreement with a single statement. These hypotheses are probing for a variety of interaction effects which may or may not be present, as opposed to trying to verify that *Dunyazad* is working as intended.

The first such proposed effect is roughly stated as: "Unexpected failure is more acceptable when a choice seems free vs. forced." In other words, I suspected that when a promising outcome leads to failure, players will be more unhappy if that

Question	Hypothesis
Fair	unexp. failure>obv. failure [main]
Unfair	unexp. failure<obv. failure [main]
Sense	unexp. failure>obv. failure [main]
Broken	unexp. failure<obv. failure [main]
Good	unexp. failure>obv. failure [main]*
Bad	unexp. failure<obv. failure [main]*
Satisfied	unexp. failure<obv. failure [main]*
Dissatisfied	unexp. failure>obv. failure [main]*
Expected	unexp. failure>obv. failure [main]*
Unexpected	unexp. failure<obv. failure [main]*

Table 8.3: Relative hypotheses regarding free vs. forced failures. Hypotheses marked with a '*' are low-confidence hypotheses.

seemed to be the only promising outcome, rather than one of several promising outcomes. The reasoning behind this is that when a choice seems to have both good and bad options, players may expect to be rewarded for successfully picking a good option. On the other hand, if only good options are present, players may not see their choice as meritorious, and thus may have lower expectations of the result. Additionally, after seeing an unexpected negative outcome, if there were other seemingly good options present players may reason that one of those options would have led to a better results, while no such reasoning is available if the other options seemed bad. This hypothesis also parallels the idea in decision affect theory that outcomes are judged relative to salient counterfactual alternatives (Mellers, A. Schwartz, and Ritov, 1999) (although in this case the alternate outcomes are merely suggested instead of clearly stated).

Table 8.3 lists the individual hypotheses associated with this proposed effect, but they can be stated more simply in terms of the underlying variables. This proposed effect predicts that compared to surprising failures at "`obvious`" choices,

Question	Hypothesis
Good	exp. success<obv. success [main]
Bad	exp. success>obv. success [main]
Satisfied	exp. success<obv. success [main]
Dissatisfied	exp. success>obv. success [main]

Table 8.4: Relative hypotheses regarding chosen vs. inevitable successes.

surprising failures at "positive_alternatives" choices will be perceived as more fair, more sensible, better, more regretted (because changing options seems more promising), and more expected. When I formulated these hypotheses I was most confident in the effects on perceptions of fairness and sense, so the other hypotheses are marked as low-confidence with a '*'.

Another proposed difference was between the "expected_success" case and the "obvious_success[main]" case. Both of these involve expected success outcomes, the only difference being whether other outcomes at the choice were promising or not. One hypothesis was that when alternatives seemed dangerous, a positive outcome would seem slightly better (both cases were predicted to evaluate their outcomes as good overall) because of a greater perceived difference in outcomes (although participants didn't get to see the outcomes of non-chosen options). This accounts for the first two rows of table 8.4, while the other two indicate a prediction that participants will feel more satisfied with the option that they chose when the alternatives are negative. The reasoning here is that curiosity about the results of non-chosen but still positive-seeming options would slightly change the degree of agreement with the statements about being satisfied and regret (both cases were predicted to elicit overall agreement with the satisfied statement and disagreement with the regret statement). This line of reasoning also echoes decision affect theory's ideas about comparison between actual results and salient alternatives. As with the free-vs-forced hypotheses

Question	Hypothesis
Fair	unexp. failure<unexp. success
Unfair	unexp. failure>unexp. success
Sense	unexp. failure<unexp. success
Broken	unexp. failure>unexp. success

Table 8.5: Relative hypotheses regarding good vs. bad surprises.

above, these hypotheses were predicting a change in the degree of agreement/disagreement on some items, rather than an outright contrast between one category which is expected to agree with a statement while another disagrees.

Yet another predicted effect was a difference in perceived fairness between good and bad unexpected results, with hypotheses listed in table 8.5. In both the unexpected failure and unexpected success case, *Dunyazad* effectively contradicts the normal effects of skill and item possession on the outcomes of actions to produce an effect that it labels as either "`unfair`" or a "`miracle`." In both cases, it could be argued that these outcomes are "unfair," although perhaps in two possible senses of the word ("unjust" vs. "cheating"). The prediction in this case is that participants will respond more strongly to negative surprises than to positive surprises (in line with (Shepperd and McNulty, 2002)), being more likely to label negative surprises as "unfair" or "broken" than positive surprises.

Besides the effect of valence given surprise, I also made predictions regarding the effect of surprise given valence, for both positive and negative outcomes. Tables 8.6 and 8.7 list the relevant hypotheses, which are separate for failures and successes. For failures, the expectation is that unexpected bad things will be perceived as worse all-around than expected bad things. A secondary prediction is that participants will be more likely to want to change their choices when a bad result is unexpected than when it is hinted at beforehand. These predictions continue to follow (Shepperd and McNulty, 2002) on judgements about expected

Question	Hypothesis
Good	unexp. failure<exp. failure
Bad	unexp. failure>exp. failure
Satisfied	unexp. failure<exp. failure
Dissatisfied	unexp. failure>exp. failure
Good	obv. failure [main]<exp. failure
Bad	obv. failure [main]>exp. failure
Satisfied	obv. failure [main]<exp. failure
Dissatisfied	obv. failure [main]>exp. failure

Table 8.6: Relative hypotheses regarding expected vs. unexpected failures.

versus unexpected outcomes. Because there are two conditions that give rise to unexpected failures, there are a total of eight individual hypotheses that result instead of just four.

For successes, somewhat similar logic applies: a good surprise may seem sweeter than something which is expected beforehand. This translates to hypotheses that unexpected successes will be perceived as better and will leave participants more satisfied with their choices than expected successes.

Note that for both of these proposed surprise-based effects, a fourth condition is technically relevant: the "`obvious_success[alt]`" case consists of outcomes that are expected failures, and the "`obvious_failure[alt]`" case consists of unexpected successes. However, neither of these cases were expected to attract many participants, and expectations when choosing an option which presumably seems worse than the "correct" option are probably not the same as expectations when forced to choose one of three negative options, so these two cases were not included in these hypotheses.

In total, the proposed effects are backed by 34 relative hypotheses. These hypotheses are exploratory: they aren't designed to verify that *Dunyazad* works

Question	Hypothesis
Good	exp. success<unexp. success
Bad	exp. success>unexp. success
Satisfied	exp. success<unexp. success
Dissatisfied	exp. success>unexp. success
Good	obv. success [main]<unexp. success
Bad	obv. success [main]>unexp. success
Satisfied	obv. success [main]<unexp. success
Dissatisfied	obv. success [main]>unexp. success

Table 8.7: Relative hypotheses regarding expected vs. unexpected successes.

as intended, but instead they represent guesses about how participants might react. If they are confirmed, they have interesting implications for the choice poetics, and each proposed effect could be investigated further.

8.3.4 Motivation Hypotheses

The final category of hypotheses for the retrospective study was a set of hypotheses about the motivation questions. These questions were not really tied to *Dunyazad*'s performance, but rather designed to gather preliminary data on player motivations that could help guide the design of a study focused on modes of engagement. The hypotheses are thus simple thresholds, and they don't have associated statistical tests. Each of the motivation questions had at least one associated hypothesis:

- The first motivation question was about what motives contributed to participants' decisions. The hypotheses for this question were that the first (`speed`), fourth (`avatar`), and sixth (`power`) options would each be selected by at least 50% of participants.

- The second motivation question concerned how participants decide that an option is "good." The hypotheses were that the first (`avatar`), fourth (`power`), and fifth (`progress`) options would each be selected by at least 50% of participants.

- The third motivation question was like the second but it focused on "bad" outcomes. The hypotheses were that the first (`avatar`), third (`no.control`), fifth (`power`), and sixth (`progress`) options would each be selected by at least 50% of participants.

- The final motivation question forced participants to choose a single answer and asked whether they believed their motives were consistent or malleable across different interactive experiences. The hypothesis for this question was that among participants who did not select the third option (`no.exp`, indicating that they did not have much experience with interactive narratives), at least 70% would select the second option (`variable`) over the first (`consistent`) and fourth (`other`) options.

All of they hypotheses in the motivation section were motivated mostly by my own sense of how people play games, which is of course idiosyncratic. The real value of the motivation questions is not in whether their related hypotheses are supported or not, but in the aggregate data that they supply. The hypotheses are still an important point of reference, however, as they help avoid a completely post-hoc analysis of the data.

8.4 Results

As with the previous experiment, some submissions were rejected because they failed to answer screening questions correctly (the same two screening questions as the first survey: one asking for age confirmation in a manner that required

English proficiency and the other an attention check within the body of the survey). There were 10 rejected responses, and these were resubmitted to Amazon so that new participants were found; there a total of 280 responses including rejections. Rejected responses were not used in the analysis.

Unfortunately, the system for ensuring that the same participant did not take the survey more than once was configured improperly at the beginning of data collection, and some participants submitted multiple responses (across different conditions). When the issue was discovered it was corrected, and all responses from individuals who saw more than one choice were not used during data analysis (to preserve the condition that judgements were not based on cross-category comparisons). A total of 65 responses from 14 individuals who submitted more than one survey were removed, reducing the total number of valid responses from 270 to 205. Luckily, because workers who accepted multiple tasks generally did not do more than one from the same batch, the impact of this error was distributed fairly evenly. After filtering out these responses and accounting for places where valid responses were missing an individual item, each condition still had between 32 and 36 valid responses, and each choice had between 8 and 14. Thus while the statistical power of the study was reduced, the problem wasn't severe enough to call into question the validity of the results.

Unlike the first study, data was only filtered based on submission approval and this multiple-submission issue. This meant that data from respondents who left some items blank was still included in the analysis. Before doing the first study, I was worried that leaving a question blank might be a sign that someone was completing the survey haphazardly, and I wanted to filter out such responses. After seeing the data from the first survey, including the distribution of response times, I was much less worried about this issue, and although leaving a question blank might be a sign that a participant was distracted, they might still be answering the other items honestly, especially if they answered the

trick question correctly (as every single response that was accepted and wasn't a duplicate did). The second study also included a mechanism that changed the color of each response to green to encourage participants to fill out every question. In the end there were a total of only 13 missing responses out of the 3075 hypothesis-related Likert items seen by non-multiple-participants whose submissions were accepted. The most-impacted single item was the statement "[...] there are no options that seem bad [...]," which was missing 5 of the 205 responses that it should have had.

8.4.1 Response Times

The median response time was 672 seconds (11:12) before filtering and 592 seconds (9:52) after filtering, which is in line with this survey being a little more than twice as long as the first survey. The minimum accepted response time was 155 seconds (2:35) which did not seem short enough to filter out categorically, especially as a quick look at the answers given did not suggest random responses. There were a total of eight accepted responses that were completed in less than 240 seconds (4:00). The fact that the median decreased significantly after filtering likely has to do with the filtering of multiple responses from individual participants. Normally if a worker on Mechanical Turk wants to do more than one task from a single category, they will accept a batch of tasks and then proceed to work through them, to ensure that other workers don't use up the remaining tasks while they work on one. This means that their response times will be linearly increasing, because the response time is just the time between accepting and submitting a task. This also means that multiple-submitters will account for more than their fair share of long-response-time submissions, and so filtering them out will tend to reduce the median response time. Even workers who only submit a single task are likely doing other tasks on Mechanical Turk and may engage in the same accept-to-reserve behavior, which is why I did not consider

filtering out responses that took "too long." Without clever instrumentation of the survey page, the data only provides an upper bound on the time that it actually takes a worker to complete a task.

8.4.2 Summary

A summary of the results appears in figs. 8.1 to 8.4, which have the same format as fig. 7.3. Figure 8.1 includes all of the option-related questions, and can be compared directly to results in fig. 7.3, especially for the last four conditions, which correspond in pairs to the "`obvious`" and "`dilemma`" conditions of the first experiment, respectively. In figs. 8.2 to 8.4, the questions are ordered such that complimentary question pairs are adjacent, and comparing the data visually there seems to be a good deal of symmetry between these pairs, as expected. Overall, the data indicate that *Dunyazad* is better at creating positive expectations and outcomes than negative ones: compare the incredibly one-sided results for the "`expected_success`" condition to the much more mixed results for the "`expected_failure`" condition, for example.

As mentioned above, each of the rows in these figures represents between 32 and 36 responses, depending on the condition. The sub-cases (e.g., "`obvious_failure[main]`" vs. "`obvious_failure[alt]`") are not separated here, although they are analyzed separately for some of the hypotheses. For the "`obvious_success`" condition, 25 of the 33 responses fell into the [main] sub-case, while the remaining 8 participants chose an option other than the one that *Dunyazad* expected would be most promising. For the "`obvious_failure`" condition, the "`[main]`" sub-case contained 20 of the 35 responses, while the [alt] sub-case contained the remaining 15. Note that each condition can be broken down into three individual choices, and there is a general expectation that these three choices will elicit generally similar responses, as they are constructed using the same set of constraints (although they are required to use three different

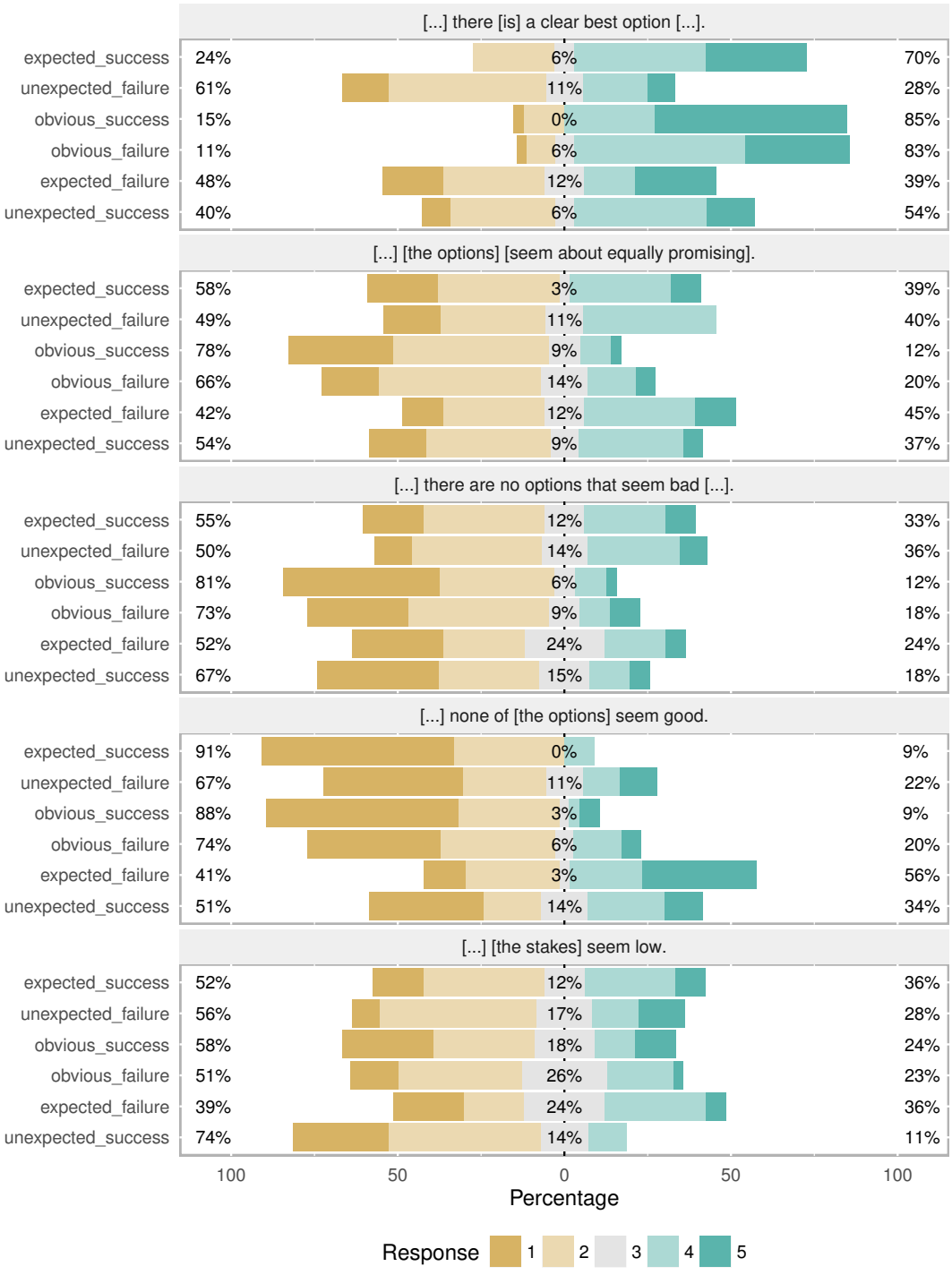

Figure 8.1: Option-related results from the retrospective study. Cf. fig. 7.3.

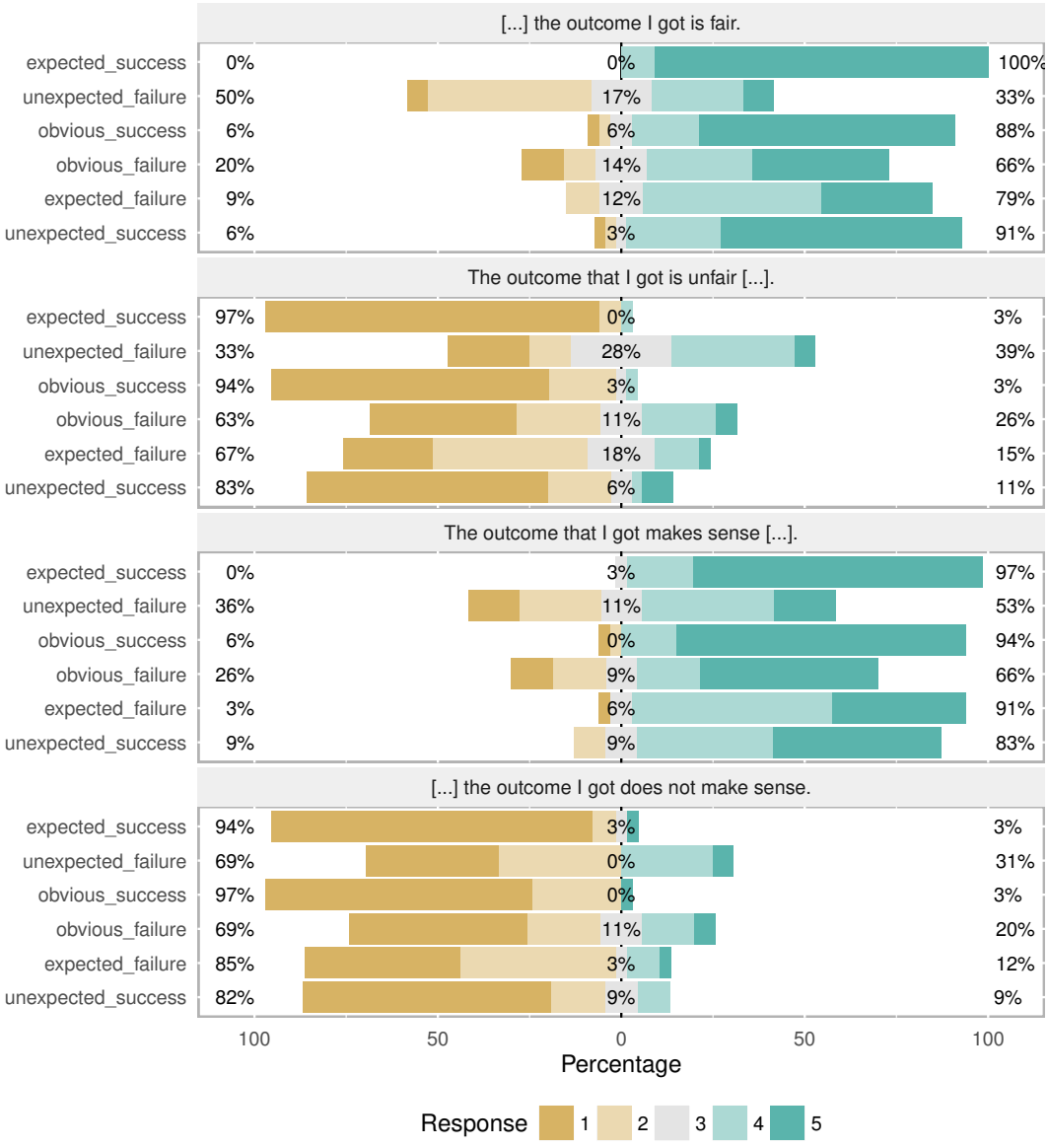

Figure 8.2: Fair/unfair and sense/nonsense results from the retrospective study. Setup is the same as fig. 7.3.

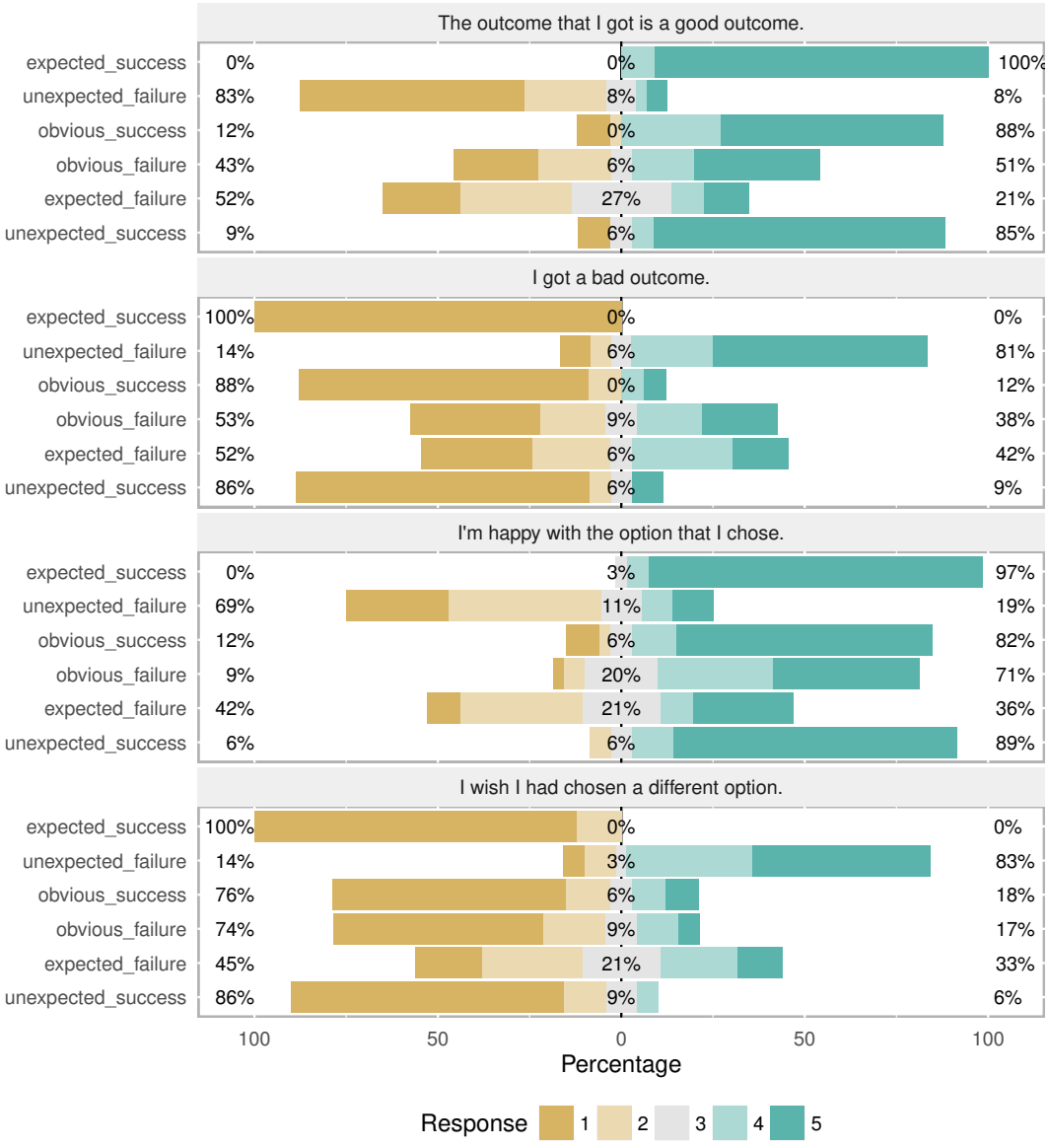

Figure 8.3: Good/bad and satisfied/dissatisfied results from the retrospective study. Setup is the same as fig. 7.3.

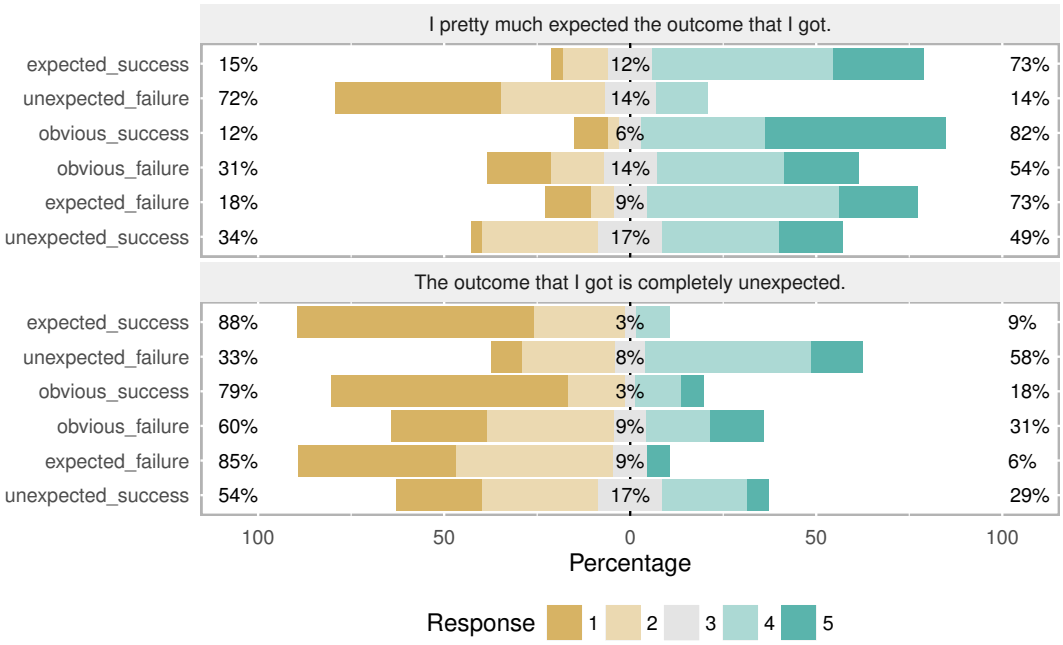

Figure 8.4: Expected/unexpected retrospective results. Setup as in fig. 7.3.

setups). Slight variation between choices within a condition should not disrupt the overall results, of course, but if there is significant divergence, it may be a sign that *Dunyazad*'s internal reasoning about a choice doesn't agree with how participants experience that choice.

8.5 Option Results

Table 8.8 shows the results for the option-related hypotheses, with significant results listed in bold. Note that in line with the results of the first study, none of the low-confidence hypotheses were supported by the data. These hypotheses (marked with '*') involved predictions which don't hold up if participants judge options relative to each other instead of in absolute terms, and the prospective study indicated strongly that this was the case for many participants.

Question	Expected Success / Unexp. Failure			Obvious Success / Obvious Failure			Expected Failure / Unexp. Success		
Clear best	D*	0.987	×	**A**	**8.3×10^{-5}**	**75%**	D*	0.467	×
	D*	0.128	×	**A**	**0.001**	**70%**	D*	0.717	×
Balanced	A*	0.809	×	**D**	**0.003**	**69%**	A*	0.485	×
	A*	0.767	×	**D**	**0.045**	**61%**	A*	0.794	×
No bad	A*	0.807	×	**D**	**4.5×10^{-4}**	**72%**	D	0.082	×
	A*	0.683	×	**D**	**0.013**	**65%**	**D**	**0.011**	**66%**
No good	**D**	**1.1×10^{-5}**	**78%**	**D**	**4.1×10^{-5}**	**76%**	A	0.140	×
	D	**0.013**	**65%**	**D**	**0.004**	**68%**	A	0.883	×
Low stakes	D	0.279	×	D	0.081	×	D	0.310	×
	D	0.257	×	D	0.121	×	**D**	**0.003**	**68%**

Table 8.8: Option-related results in the retrospective experiment. Each row lists results for a single question; each column stacks results for two conditions (listed at the top) with identical "`option_feel`" constraints. Each entry indicates the hypothesis ('D' for 'disagree' and 'A' for 'agree'), the p-value, and the common-language effect size for confirmed hypotheses (which are marked in bold instead of blue where $p < 0.05$).

8.5.1 Stakes

One immediate pattern in these results is the lack of support for the "low stakes" hypotheses. In the initial study, the hypothesis that choices which *Dunyazad* labelled as high-stakes would appear that way to participants was clearly confirmed by the data, with a p-value of 0.0011 and an effect size of 70% (a strong effect). In this study, every single choice was required to be high-stakes, but only in a single case (the "`unexpected_success`" condition) was the relevant hypothesis confirmed. One reason why this might be the case is that in this study, *Dunyazad* was not allowed to control either the setup or the stakes of the choices, whereas in the initial study *Dunyazad* was free to control both. The high-stakes

choices in this study therefore necessarily included choices with "`market`" setups, where it is impossible for *Dunyazad* to construct a scenario that directly threatens the player's character. A breakdown of responses to the stakes question across all conditions by setup (fig. 8.5) seems to confirm that the "`market`" setup is a problem: it elicits an even split between agree and disagree responses while the other two setups lean towards disagreeing.

To test this theory, I did a followup analysis with three hypotheses—each individual setup as a condition was hypothesized to elicit disagreement with the low stakes statement. The results of this analysis confirmed my suspicions. The test for the "`market`" hypothesis does not reject the null hypothesis ($p = 0.502$), while the tests for the "`threatened_innocents`" and "`monster_attack`" hypotheses are confirmed ($p = 0.041$; effect size 60% and $p = 0.0093$; effect size 64% respectively). Essentially, when *Dunyazad* creates choices without constraints regarding their stakes, the high-stakes choices that it does create are in fact perceived to be high-stakes by players. However, when *Dunyazad* is specifically asked to create high-stakes choices using certain fixed setups, such as the "`market`" setup, it doesn't manage to create choices that are convincingly high-stakes. Whether this means that the perception of stakes has more to do with the setup text than the actual content of the options and/or outcomes is an open question that could be tested.

8.5.2 Expected Failure—No Bad Options?

Besides the unconfirmed stakes hypotheses, there were two other places where high-confidence option-related hypothesis were not confirmed by the data. The first is in the "`expected_failure`" case for the "no bad" statement. Although the "`unexpected_success`" condition, which in theory generates options which set identical expectations, was confirmed to disagree with this statement, the data did not support this hypothesis for the "`expected_failure`" condition. Of

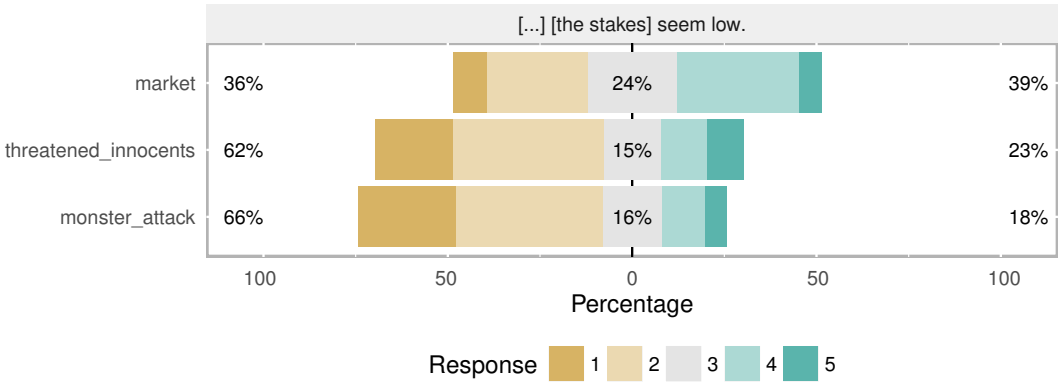

Figure 8.5: Stakes results across all conditions grouped by the setup used.

course, this condition is one susceptible to being undermined by relative value judgements: if all of the options seem bad, one might reason that it doesn't matter which is chosen, and therefore none of them are "bad" in the sense that choosing them would be a mistake. It is telling that the "`obvious_`" cases confirmed corresponding hypotheses: when a single positive option was present alongside negative options, participants strongly rejected the notion that there were "no bad options" present.

But what about the difference between the "`expected_failure`" case and the "`unexpected_success`" case? Figure 8.6 shows the results for each individual choice in both cases, and there isn't a clear culprit. The choice with seed 58403 is the single choice among the six in that figure that elicited the most agreement with the statement, but it still tilts towards disagreement. Given that both conditions include a substantial minority of "agree" and even "strongly agree" responses, there doesn't seem to be a clear underlying reason for the division in the results. What is clear is that giving *Dunyazad* the ability to make relative rather than purely absolute value judgements should be a high priority. In future studies, it might also be more productive to just ask participants to label each option as good/bad (and also suggests-success/suggests-failure), although as

mentioned in the discussion of the setup for the first study, this kind of language promotes an analytical approach and that isn't necessarily desirable.

8.5.3 Dilemma Cases—Competing Goals

Another point of interest among the option-related results was the data's failure to support the hypotheses that the "`expected_failure`" and "`unexpected_success`" cases would be viewed as having "no good options." Again, the "`dilemma`" case in the first experiment produced reactions that supported an equivalent hypothesis, so this result is somewhat surprising. Here however the reasons for this discrepancy are clear.

Figure 8.7 shows the responses for these cases, and it is clear that the choices with seeds 87991, 8015, 47794, and 33152 are not yielding the expected results: participants are indicating that there is at least one good option at these choices. Figure 8.8 shows the choice with seed 87991, and the problem is immediately

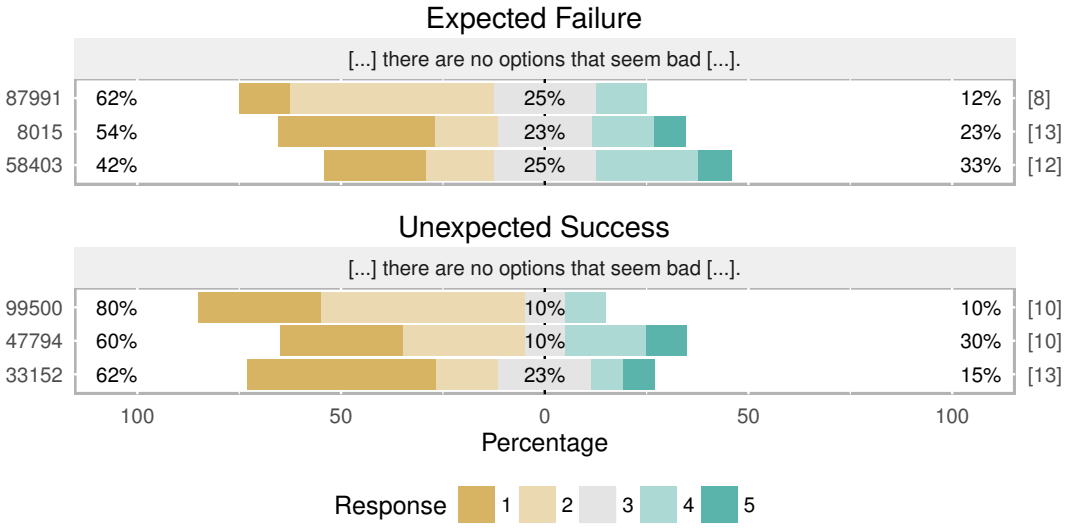

Figure 8.6: Results for the "no bad options" statement in the "`expected_failure`" (top) and "`unexpected_success`" (bottom) conditions, grouped by choice. Each choice is identified by the numeric seed that was used to create it. The bracketed numbers on the right indicate the sample size for each choice.

apparent: *Dunyazad* has used a "`travel_onwards`" action as a "clearly bad result." This is a result of an overcorrection from the problems with "`travel_onwards`" that were discovered during the first study. In that study, "`travel_onwards`" was a cure-all option which could get rid of any problem due to the way that it handled switching scenes. For this second study, that bug was fixed, and instead, "`travel_onwards`" is marked as failing any goals relating to unresolved situations that it leaves behind. In this case, *Dunyazad* thinks that the player will have an "`avoid_accusations`" goal with respect to the accused merchant, and simply leaving the merchant behind clearly fails this goal.

Of course, the underlying problem here is goal prioritization. If given an good option to save the merchant, many players are eager to do so, but when such options are risky, they prefer to preserve their own safety. With *Dunyazad*'s two-level goal priority system, which was unchanged from the first experiment, it's not possible to represent this complexity. Under the current system then, instead of "`travel_onwards`" being seen as an option with too few consequences, it

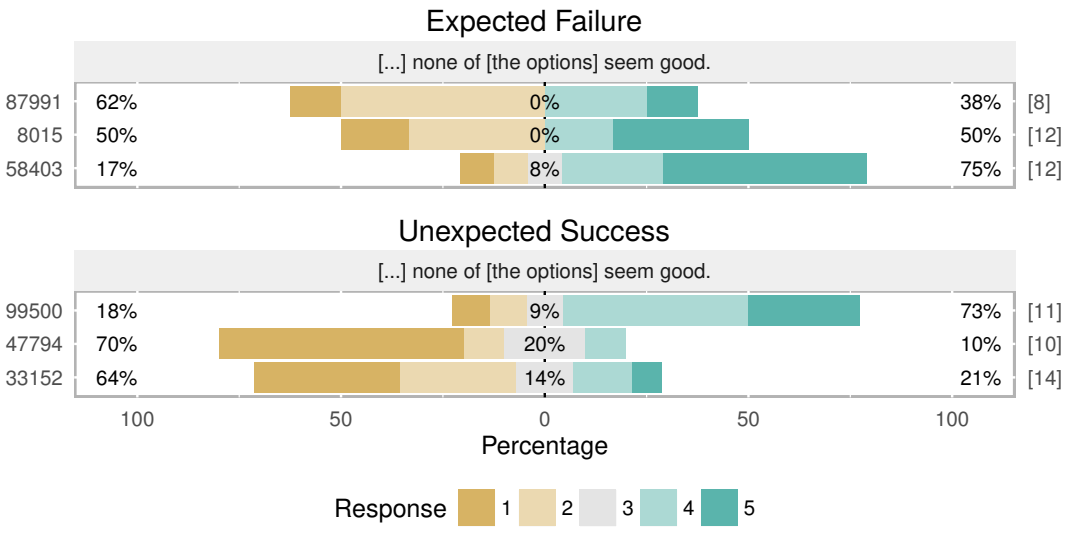

Figure 8.7: Results for the "no good options" statement in the "`expected_failure`" (top) and "`unexpected_success`" (bottom) conditions, grouped by choice. The bracketed numbers in on the right indicate the sample size for each choice.

You come to a chaotic market and decide to browse for a bit.
A merchant seems knowledgeable and a noble is baselessly accusing the merchant of insolence and the merchant is selling a spear.
What do you do?

1. *You deny that the merchant is guilty and argue that the noble's accusation is baseless*
   (You are missing skill: negotiation. The noble has skill: negotiation).
   [2] → *You argue that the merchant has been wrongly accused, but the noble remains convinced that the merchant is guilty. The situation remains the same.*
2. *You transform the noble into a chicken*
   (You have skill: sorcery. You have no tool for sorcery).
   [1] → *You try to cast a spell on the noble, but nothing happens. The situation remains the same.*
3. *You travel onwards*
   (no relevant skills).
   [5] → *You continue your journey. You have traveled to a new location.*

Figure 8.8: The choice with seed 87991 minus the framing (which is mostly the same between choices). Boxed numbers before each outcome indicate the number of participants that chose that outcome. Note the popularity of the third option.

is seen as an option with too many consequences. The choice with seed 8015 also includes a "`travel_onwards`" option in a similar situation, and the inclusion of these options was likely the reason that players felt that the "`expected_failure`" cases actually *did* have some good options.

8.5.4 Dilemma Cases—Outcomes Affect Option Perception

Recall that the "`unexpected_success`" case exhibited the same problem. The reason for this was different, however, because neither the choice with seed 47794 nor the choice with seed 33152 had "`travel_onwards`" options. Instead, the reason that these choices were rated as having at least one good option is probably their outcomes. Because participants saw an outcome before answering

You come across some bandits attacking a merchant.
The bandits are threatening the merchant.
What do you do?

1. *You talk the bandits down*
   (You are missing skill: negotiation).

   [6] → *You reason with the bandits and the bandits agree to back off. The bandits are no longer threatening the merchant.*

2. *You transform the bandits into a chicken*
   (You are missing skill: sorcery).

   [1] → *You cast a spell on the bandits, and the bandits are cursed with the form of a chicken. The bandits are no longer threatening the merchant, they have been cursed and transformed into a chicken, and they are now not sentient.*

3. *You attack the bandits*
   (You are missing skill: fighting. They have skill: fighting. They have no tool for fighting).

   [3] → *You attack them. They fight back but then they are hit by your attack and then you dodge the bandits' attack. They are defeated. You have acquired a book of herbal lore, the bandits are not threatening the merchant any more, and they have been killed.*

Figure 8.9: The "`unexpected_success`" choice with seed 47794 minus its framing. Boxed numbers indicate the number of participants that selected each outcome.

any questions, the responses to the option questions could have been affected by the outcomes seen. Figure 8.9 shows the choice with seed 47794, and looking at it reveals a likely explanation for the "no good options" response.

As shown in that figure, six of ten participants who saw that choice chose the first option, and were "unexpectedly" successful. But remember that *Dunyazad*'s notion of what is expected is written in terms of system-wide rules: "normally" someone missing the negotiation skill should *always* fail to talk someone else down, so if such an action succeeds the result is unexpected. This kind of expectation is something that players would learn if they interacted with the system over a period of time, but participants in this study only saw a single

choice. Faced with a positive outcome, it would be natural to conclude that perhaps the negotiation skill isn't important after all, and therefore that the option that was chosen was a good one. This line of reasoning would not be available to players before seeing an outcome, but as stated before, the risk of such influences was intentionally accepted as the price of a more natural decision-making experience.

Because the choice with seed 33152 included a very similar "`talk_down`" option which was also quite popular, this outcome-dependent effect can explain the failed hypothesis in the "`unexpected_success`" condition. Unfortunately, the impact of this effect and the "`travel_onwards`" issue affecting the choices with seeds 87991 and 8015 is not negligible. When it comes to evaluating the outcome-related hypotheses, these choices didn't convey the intended prospective poetics, and this could impact their retrospective poetics as well.

8.5.5 Problems with "`travel_onwards`"

Besides choices in the two dilemma conditions, two other choices—the "`obvious_success`" choices with seeds 47371 and 20739—also included "`travel_onwards`" options. These are shown in fig. 8.10. Luckily, the choice with seed 47371 happens to give participants an option that's more attractive than the "`travel_onwards`" option, so the same problem doesn't occur. In fact, this is a good example of correct use of "`travel_onwards`" as an option that seems bad, which is reinforced by the fact that zero participants chose it. However, based on the option popularities in choice 20739, that choice has the same problems as choices 87991 and 8015: the "correct" option involves some perceived risk or cost (in this case giving away an item to resolve the situation) and so simply ignoring the merchant's plight is seen as acceptable.

Note that these options are in the "`obvious_success`" condition, and as such participants who saw them are divided into the "`[main]`" and "`[alt]`" sub-cases

You come across some bandits attacking a merchant.
The bandits are threatening the merchant.
What do you do?

1. *You talk the bandits down*
   (You are missing skill: negotiation).

   2 → *You talk with the bandits, but the bandits refuse to back down. The bandits have started to threaten you.*

2. *You transform the bandits into a chicken*
   (You have skill: sorcery. You have your ancient grimoire).

   11 → *You cast a spell on the bandits, and the bandits are cursed with the form of a chicken. The bandits are no longer threatening the merchant, they have been cursed and transformed into a chicken, and they are now unintelligent.*

3. *You travel onwards*
   (no relevant skills).

   0 → *You continue your journey. You have traveled to a new place.*

You come to a chaotic market and decide to browse for a bit.
A merchant seems knowledgeable and a peasant is bored and a noble is baselessly accusing the merchant of disrespect and the merchant is selling a spear.
What do you do?

1. *You offer the noble your sack of gold*
   (You have skill: negotiation).

   2 → *You bargain with the noble and reach a deal. You have lost your sack of gold, the noble is not accusing the merchant any more, and the noble has a sack of gold now.*

2. *You travel onwards*
   (no relevant skills).

   5 → *You continue your journey. You have traveled to a new place.*

3. *You transform the noble into a chicken*
   (You are missing skill: sorcery).

   1 → *You try to cast a spell on the noble, but nothing happens. The situation remains the same.*

Figure 8.10: The "`obvious_success`" choices 47371 and 20739 minus framing. Boxed numbers indicate the number of participants who saw each outcome.

according to which option they chose. For choice 47371, all 11 participants who chose option two are in the "[main]" sub-case, while the other two participants are in the "[alt]" sub-case. For choice 20739, there are only two participants that the system thinks chose the "correct" option, and the majority of participants are put into the "[alt]" sub-case. This means that the meaning of the sub-cases is somewhat changed from its intent. The "[alt]" cases are supposed to consist of participants who intentionally chose a non-positive option despite the presence of a positive one, and as such they are assumed to be quite small, to the point that not many hypotheses were proposed which included "[alt]" cases. Because of *Dunyazad*'s failure to predict the complexities of player goals, the "[alt]" case consists largely of participants who thought they were choosing the best option available, but who disagreed with *Dunyazad*'s evaluation of what that was. Although these effects diminish the number of participants in the "[main]" cases, at least their meaning is not diluted very much: only a few participants from poorly-formed choices like 20739 wound up choosing the options that *Dunyazad* considered best, and so only a few participants in the "[main]" cases weren't choosing what they thought was an obviously superior option.

8.5.6 Costly Actions

One failure in *Dunyazad*'s reasoning here, besides the inability to properly track goal priorities, is its focus on goals to the detriment of costs. In other words, *Dunyazad* thinks about actions only in terms of the goals that they might achieve or threaten, and not in terms of the costs of those actions. If the player must use an item to get what they want, *Dunyazad* neglects this cost and just sees that a goal is achieved. Of course, this could be handled by adding player goals concerned with preserving resources, but the two-priority goal system doesn't leave room for such goals to affect high-stakes decisions even a little bit.

The choice with seed 20739 in fig. 8.10 is an example of this: In making a decision, even players who recognize that the first option will help the merchant have to somehow justify giving up their gold (and to a noble who doesn't deserve it, no less). As the numbers indicate, most decided to simply travel onwards: the goal of helping the merchant was not worth the cost associated with it. Of course, it doesn't help that the option text doesn't clearly state your intent in performing the action (to exonerate the merchant).[1] In fact, one of the comments in the "other" free text response to the motivation question reveals exactly this kind of cost analysis in action:

[1]This issue has already been addressed by making the text more explicit for "`pay_off`" actions.

You come to a busy market and decide to browse for a bit.
A thief seems knowledgeable and a noble is baselessly accusing a merchant of insolence.
What do you do?

1. *You talk the noble down*
   (You have skill: negotiation).

   12 → *You reason with the noble and the noble calms down. The noble is no longer accusing the merchant.*

2. *You offer the noble your sack of gold*
   (You have skill: negotiation).

   0 → *You bargain with the noble, but can not seem to reach a deal. The situation remains the same.*

3. *You transform the noble into a chicken*
   (You are missing skill: sorcery).

   0 → *You cast a spell on the noble, and the noble is cursed with the form of a chicken. The noble is not accusing the merchant any more, the noble has been transformed into a chicken, the noble is now not sentient, the noble no longer has some perfume, and the noble no longer has an ancient grimoire.*

Figure 8.11: The "`obvious_failure`" choice with seed 95923 minus framing. Boxed numbers indicate the number of participants that selected each outcome.

> *You only really had 2 options. I wouldn't want to give up the gold if I didn't have to. So just negotiation made the most sense.*

This response was given by a participant who saw choice 95923, an "`obvious_failure`" choice shown in fig. 8.11. The fact that players are factoring costs into their decisions but *Dunyazad* is not tracking them is a problem. Of course, the quote above reveals another problem: the choice with seed 95923 was supposed to be an "`obvious`" choice with a single best option. How did *Dunyazad* justify having two options that include positive skill indications? In this case, *Dunyazad* has found what amounts to a loophole in its rules: the "bad" options at an obvious choice are allowed to be merely "risky"—the reasoning is that compared to a really good option, a risky option will still be clearly inferior. Because the "`talk_down`" action includes a risk that the target will begin accusing the initiator instead of their original victim, and because, internally, having the negotiation skill doesn't guarantee success, *Dunyazad* views the first option as a "risky" one: one that has an unsubstantiated benefit but also a potential drawback. Three factors combined to throw off *Dunyazad*'s analysis. First, the unaccounted-for cost of the "good" option meant that it really wasn't as attractive as it should have been. Second, the fact that talking the noble down carried a risk wasn't something that was made clear from the surface text. Third, the fact that in the first option, *Dunyazad* does not count the negotiation skill as decisive while in the second option it does creates a false equivalence at the surface text level: both options appear equally supported by the "negotiation" skill.

This problem of costliness was specific to the "`pay_off`" action, which was present in the five choices with seeds 20739, 95923, 40550, 67832, and 82994. Choice 20739 was just discussed in the context of the "`travel_onwards`" action and choice 95923 is the choice that prompted this analysis, but the other three choices have not yet been considered. Luckily, choice 40550 was a "`unexpected_failure`"

choice and choices 67832 and 82994 were both "`expected_success`" choices: all three were in conditions that were supposed to have three good options. As a result, these costly options were presented along with two actually-good options, and in fact the costly options at these choices were chosen by one, zero, and zero participants respectively. In other words, these options probably had little effect on the outcome-related results, as they were almost never chosen. On the other hand, they do compromise the *option* structures of the choices in question, and so they have a potentially important effect on the between-cases comparisons.

One good thing in all of this is that none of the factors that *Dunyazad* is currently neglecting are fundamentally difficult to account for. For example, the fact that all 12 participants at choice 95923 chose the first option, which can be deduced to be the best using some simple extensions to *Dunyazad*'s current logic, means that *Dunyazad*'s general strategy of using *character* motivations to predict *player* actions is probably on the right track. Extending *Dunyazad*'s option evaluation code to reason about action costs, handle complex goal priorities, and reason about directly-presented implications rather than implications implied by hidden action logic may be technically challenging, but it's a clear set of features to implement. Despite *Dunyazad*'s failures, the results of this study strongly imply that with these features, *Dunyazad* will be able to correctly predict player's option evaluations most of the time.

8.6 Outcome Results

While the results of the option-related hypotheses were somewhat lackluster, this was to some degree expected based on the results of the first study. Had more improvements been made to *Dunyazad* before the second study, this might have been avoided, but the primary purpose of this study was to evaluate *Dunyazad*'s outcome predictions, which had not been tested by the first study. Tables 8.9

and 8.10 show the results for nominally-positive- and nominally-negative-outcome conditions respectively (the top and bottom halves of table 8.2). Overall, *Dunyazad* seems to be generally successful at producing positive outcomes, although it has some trouble making them surprising. On the other hand, its track-record for negative outcomes is not as good. In particular, surprising negative outcomes were more consistently rated as actually being "bad," but participants didn't necessarily accept the idea that they were "unfair" or "nonsense."

8.6.1 Outcomes and Option Perception (Again)

The only hypotheses regarding nominally successful outcomes that weren't confirmed were those regarding the expectedness of the nominally unexpected cases. These two conditions, the "`unexpected_success`" and "`obvious_failure[alt]`" conditions, were supposed to be cases where a participant chose an option which carried some indicator that it would fail, but got a successful result. The choices involved had seeds 99500, 47794, and 33152 ("`unexpected_success`"); and 8638, 28306, and 95923 ("`obvious_failure`"). Of course, only participants who didn't choose the nominally best option at choices 8638, 28306, and 95923 fell into the "`obvious_failure[alt]`" case.

Unfortunately, the failure to confirm these hypotheses is not terribly surprising: out of all of the outcome-related statements, the statements about expectedness and unexpectedness are the most heavily dependent on participant's perception of the option text as opposed to only the outcome text. Because several of the choices listed above had problems with option perceptions, they didn't really fit the outcome profile that the hypotheses were expecting. Choices 47794 (see fig. 8.9) and 33152 (not shown) together account or 24 of the 35 "`unexpected_success`" responses, and as already discussed, they each included a popular option that *in light of its outcome* did not seem like a bad option. Because the option text alone didn't give players a strong negative expectation for these

Question	Expected Success Obv. Success [main]			Unexp. Success Obv. Failure [alt]		
Fair	**A**	6.8×10^{-11}	**88%**	**A**	1.6×10^{-6}	**80%**
	A	1.7×10^{-8}	**87%**	**A**	1.1×10^{-5}	**85%**
Unfair	**D**	2.7×10^{-10}	**87%**	**D**	3.5×10^{-5}	**76%**
	D	1.7×10^{-8}	**87%**	**D**	4.2×10^{-6}	**86%**
Sense	**A**	8.2×10^{-9}	**85%**	**A**	1.3×10^{-4}	**74%**
	A	4.7×10^{-8}	**85%**	**A**	4.2×10^{-6}	**86%**
Broken	**D**	6.9×10^{-9}	**85%**	**D**	7.1×10^{-6}	**78%**
	D	1.7×10^{-6}	**82%**	**D**	1.1×10^{-5}	**85%**
Good	**A**	6.8×10^{-11}	**88%**	**A**	1.5×10^{-6}	**79%**
	A	3.4×10^{-7}	**84%**	**A**	4.2×10^{-6}	**86%**
Bad	**D**	6.7×10^{-13}	**90%**	**D**	9.2×10^{-7}	**80%**
	D	1.3×10^{-7}	**84%**	**D**	4.2×10^{-6}	**86%**
Satisfied	**A**	1.7×10^{-10}	**88%**	**A**	1.4×10^{-7}	**82%**
	A	4.7×10^{-8}	**85%**	**A**	1.5×10^{-6}	**87%**
Dissatisfied	**D**	2.2×10^{-10}	**88%**	**D**	4.8×10^{-7}	**81%**
	D	1.1×10^{-6}	**82%**	**D**	1.5×10^{-6}	**87%**
Expected	**A**	**0.011**	**66%**	D	0.795	×
	A	3.0×10^{-4}	**75%**	D	0.996	×
Unexpected	**D**	6.6×10^{-6}	**78%**	A	0.896	×
	D	3.0×10^{-4}	**75%**	A	0.997	×

Table 8.9: The results from the retrospective study for conditions that have nominally positive outcomes. Each row stacks results from the two (sub-)conditions listed at the top. Each result lists the hypothesis ('A' for agree or 'D' for disagree), the p-value, and if significant ($p < 0.05$) the common-language effect size.

Question	Expected Failure Obv. Success [alt]			Unexp. Failure Obv. Failure [main]		
Fair	**A**	**0.001**	**70%**	D	0.343	×
	A	0.274	×	D	0.460	×
Unfair	**D**	**0.017**	**65%**	A	0.634	×
	D	**0.024**	**72%**	A	0.419	×
Sense	**A**	**1.2×10^{-4}**	**74%**	D	0.716	×
	A	**0.012**	**75%**	D	0.508	×
Broken	**D**	**4.2×10^{-4}**	**72%**	A	0.981	×
	D	**6.2×10^{-4}**	**85%**	A	0.783	×
Good	D	0.129	×	**D**	**2.6×10^{-5}**	**76%**
	D	0.519	×	**D**	**0.006**	**70%**
Bad	A	0.762	×	**A**	**1.7×10^{-4}**	**73%**
	A	0.919	×	**A**	**0.014**	**68%**
Satisfied		-		**D**	**0.024**	**63%**
		-		D	0.809	×
Dissatisfied		-		**A**	**3.7×10^{-4}**	**72%**
		-		A	0.897	×
Expected	**A**	**0.031**	**63%**	**D**	**9.5×10^{-4}**	**71%**
	A	0.129	×	D	0.168	×
Unexpected	**D**	**2.9×10^{-4}**	**73%**	A	0.184	×
	D	**0.036**	**70%**	A	0.313	×

Table 8.10: The results from the retrospective study for conditions that have nominally negative outcomes. The format is the same as that of table 8.9.

options, positive results were not only not surprising; they made the option seem like it had been a good one from the start.

The fact that participants marked the outcomes of these options as expected might also point to a different effect: player trust in game-like systems. In most modern games, unless the player has already received substantial warnings that they are on the wrong path, there won't be choices that have no "correct" option: even the worst situations will have some means of escape if the player is thrust into them without control over prior situations. Because participants in this study only saw one choice, they may have assumed that there would be a "correct" option that would lead to a successful result. With this expectation in mind, it's not surprising at all that choosing what one thinks is the best result will lead to a good outcome, even if the option text includes some negative indicator.

From the perspective of choice poetics, both of these possible effects are important to keep in mind. First is the idea that options and outcomes both influence the perception of the other. In this case, because *Dunyazad* did not make clearly bad-seeming options, a positive outcome was able to make those options seem like good options (at the very least, relative to other options at those choices). Of course, given that those options seemed good, the notion that a good outcome would be surprising was no longer valid. In other words, players reason neither strictly from options to outcomes nor from outcomes to options, but their perceptions of both affect their perceptions of the other. If fact, there is ample evidence of this kind of reasoning even in real-world decisions (for an extreme example see studies on choice blindness such as (Hall, Johansson, and Strandberg, 2012)). If an author is interested in constructing an option that feels a specific way in retrospect, then, they must be mindful of how the consequences of that option make it appear in hindsight.

If the fact that options meant to seem indicative of failure were not explains the expectedness result in the "`unexpected_success`" case, what about

the "`obvious_failure[alt]`" case? Choice 95923 from that case has already appeared here as well (see fig. 8.11) and in combination with a graph of all responses to the "expected" question in "`obvious_failure[alt]`" cases (fig. 8.12) explains what is going on. Essentially, *Dunyazad*'s failure to make choice 95923 obvious flooded the "`obvious_failure[alt]`" sub-case with 12 responses that dominated the three from the other two choices (which were clearly much better at getting participants to choose the intended choice). The "`obvious_failure[alt]`" sub-case therefore does not mainly consist of players who chose a bad option but got a good result. Instead, it mainly consists of players who chose option one at choice 95923, which as already discussed, seemed to be a clearly positive option. Because of this, getting a good result was not surprising, and the corresponding hypotheses were not confirmed.

8.6.2 Outcome Text vs. States

Before moving on to discuss the nominally-negative outcome hypotheses, there's one more choice that didn't quite give expected results from the "`unexpected_success`" condition. This is choice 99500 (shown in fig. 8.13), which stood out because a sizeable minority felt that the outcome they got was bad. Inspecting

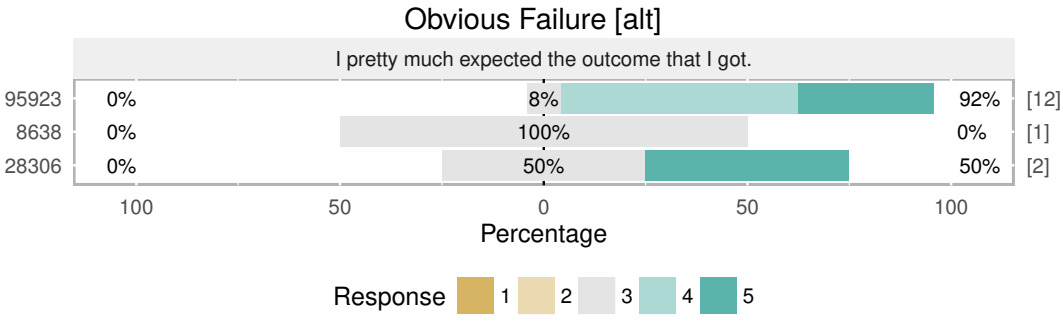

Figure 8.12: Results for the "expected" statement in the "`obvious_failure[alt]`" sub-case, grouped by seed. Note the bracketed numbers on the right which indicate the sample size for each choice: seed 95923 dominates this category.

As you journey onwards, an ogre approaches you slowly, gnashing its teeth. She is threatening you.
What do you do?

1. *You attack the ogre*
   (You are missing skill: fighting. She has skill: fighting. She has no tool for fighting).

   [4] → *You attack her. You strike at her but she thrusts at you and then you are hit by the ogre's attack. Suddenly she rallies and fiercely strikes back at you, and you are defeated. The ogre is no longer threatening you, and she is now injured.*

2. *You attempt to pacify the ogre with music*
   (You are missing skill: musician).

   [7] → *You weave a soothing melody, and the ogre becomes calm. The ogre is not threatening you any more.*

3. *You transform the ogre into a chicken*
   (You are missing skill: sorcery).

   [0] → *You cast a spell on the ogre, and the ogre is cursed with the form of a chicken. The ogre is no longer threatening you, and she has been transformed into a chicken.*

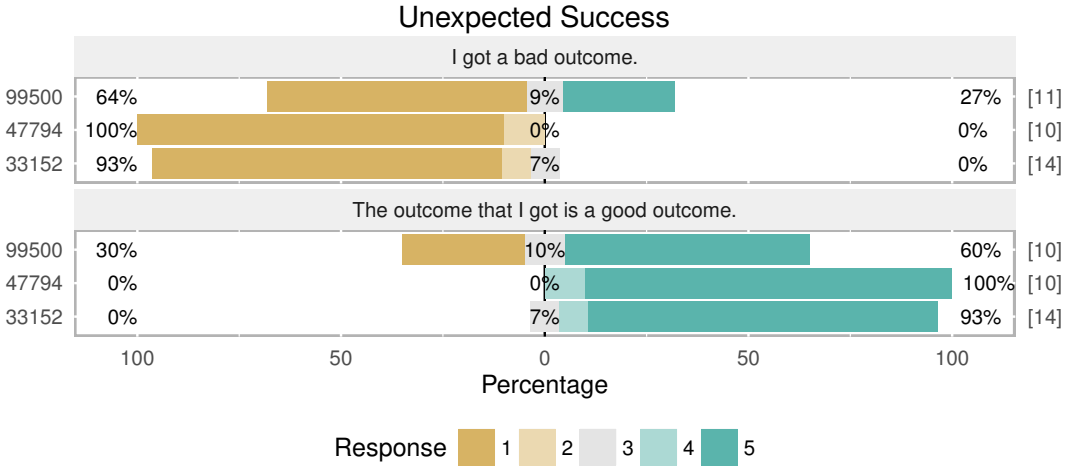

Figure 8.13: The choice with seed 99500 along with valence results from the "`unexpected_success`" condition. Boxed values indicate the number of participants that selected each outcome at choice 99500, and bracketed numbers on the right of the graph below indicate sample sizes for each row. Note the diverging responses for choice 99500 and the text of the outcome for option one.

the choice reveals the problem: the result of option one includes outcome text that seems at odds with itself: "you are defeated," but the ogre is "no longer threatening you" and "she is now injured." Within the system of outcome constraints placed on the "`attack`" action, this is allowed: defeat of the target precludes the death of the attacker, but they may still be injured. The real problem here is that the "defeated" outcome does not have any direct effects on the state of the world, unless the defeated party is threatening or accusing someone, in which case those states are removed.

Because *Dunyazad* generated a "defeat" with no negative consequences attached in terms of world state, it viewed that "defeat" as it would a victory: state changes effected by the action are unambiguously positive for the player, so the result is a "success." Of course, participants felt otherwise: of the four participants that selected that option, three gave the anomalous responses to the valence questions shown in fig. 8.13 and the fourth supplied both neutral responses. The disconnect here between the text and *Dunyazad*'s internal representation of the world causes a rift between its predictions and player's actual impressions. Furthermore, one can question whether *Dunyazad* should be able to generate such a situation in the first place: shouldn't "defeat" come bundled with a negative consequence such as being at the mercy of the victor? In this case, the effect was isolated enough that it didn't upset any hypotheses, but forcing a clear link between every outcome reflected in the text and a definite internal state would help avoid situations like this.

8.6.3 The Outcome of "`travel_onwards`"

As already mentioned, in its present state *Dunyazad* views "`travel_onwards`" actions as much more negative than participants, who often found simply ignoring a problem to be an acceptable choice. This was not only true of option perceptions, but of outcome perceptions as well. In fact, the failure of "`expected_failure`"

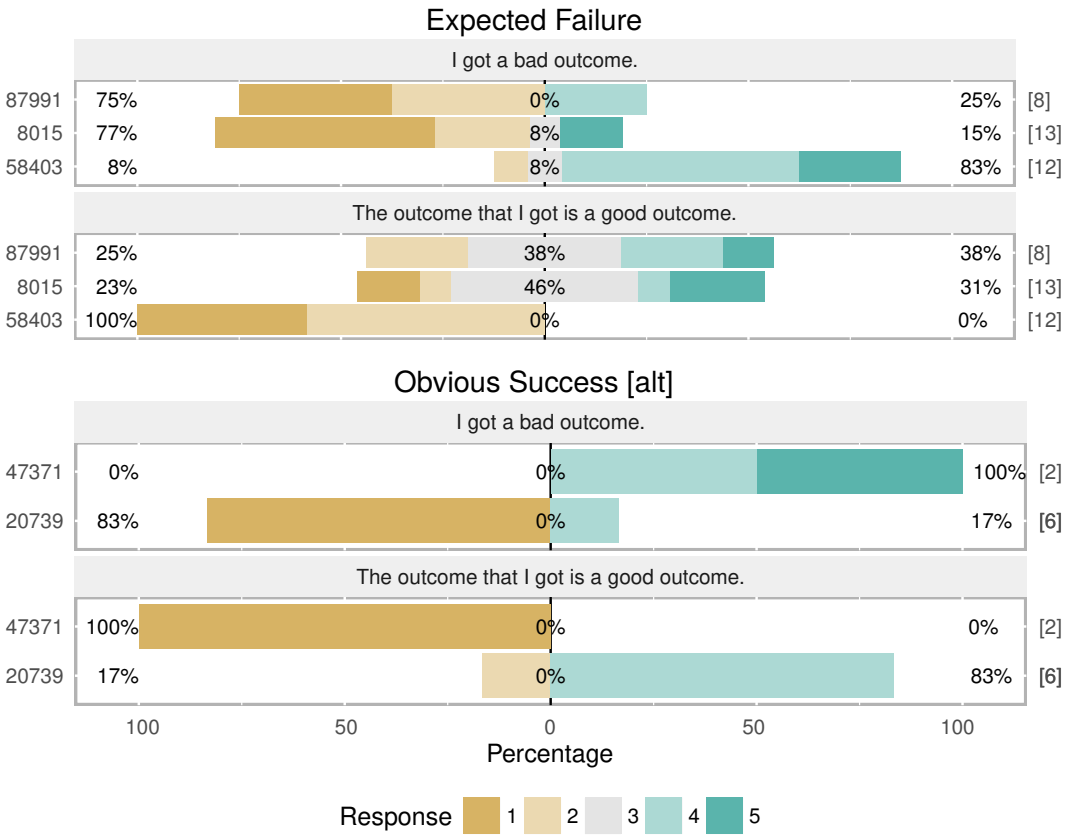

Figure 8.14: Results for the "good" and "bad" items from the "`expected_failure`" and "`obvious_success[alt]`" conditions. Bracketed numbers on the right indicate sample sizes for each row; note that the "`obvious_success[alt]`" condition is dominated by choice 20739, and even then has only 8 samples in total.

and "`obvious_success[alt]`" choices to give reliably negative results (shown in table 8.10) can be attributed to this problem. As shown in fig. 8.14, which graphs responses to the "good" and "bad" statements for the "`expected_failure`" and "`obvious_success[alt]`" conditions, the choices where a majority of participants felt that the outcome was positive are all choices containing "`travel_onwards`" options (choices 87991, 8015, and 20739, discussed above). These "`travel_onwards`" options, which the system considers to be unambiguously bad (because the player's character fails to help someone in need), are seen by players as a good

or at least mixed result: they have successfully avoided an otherwise troublesome situation. The presence of these options neatly explains the left-hand column of table 8.10: they aren't viewed as bad, but they are seen as fair, sensible, and expected. The only remaining unconfirmed hypotheses are those for the "`obvious_success[alt]`" sub-case, but those can also be explained with reference to fig. 8.14: the simple lack of data for the "`[alt]`" sub-case of the "`obvious_success`" condition (eight responses total) makes such hypotheses difficult to confirm statistically. The fact that the bulk of the "`obvious_success[alt]`" case responses (5) were from participants who selected to "`travel_onwards`" at choice 20739 (cf. fig. 8.10) didn't help matters.

8.6.4 Fairness and The Strength of Expectations

Examining the *unexpected* failure conditions (the right half of table 8.10) it is immediately apparent that participants found them to be more fair and less broken than expected. The relevant rows of fig. 8.2 indicate that these two conditions did stand out as the *least* fair and *most* broken cases, but they were not found to be more unfair than fair or more broken than sensible, as expected. Figure 8.15 breaks down the responses by individual choice, and gives a sense as to which choices were viewed as more fair (and/or less broken) than others. In the "`unexpected_failure`" condition, the choice with seed 46585 stands out as seeming more fair than its companions, and in the "`obvious_failure[main]`" condition, the choice with seed 28306 seems to be the culprit.

Figures 8.16 and 8.17 display the choices in question, and they share a common property that's likely related to why they were seen as fair: despite their differing option structures, they include options which generate only weak expectations of success, and which have somewhat mixed results. In choice 46585, both the first and third options involve some kind of debate, and while the "negotiation" skill probably helps debate successfully, it's not the kind of action where the result

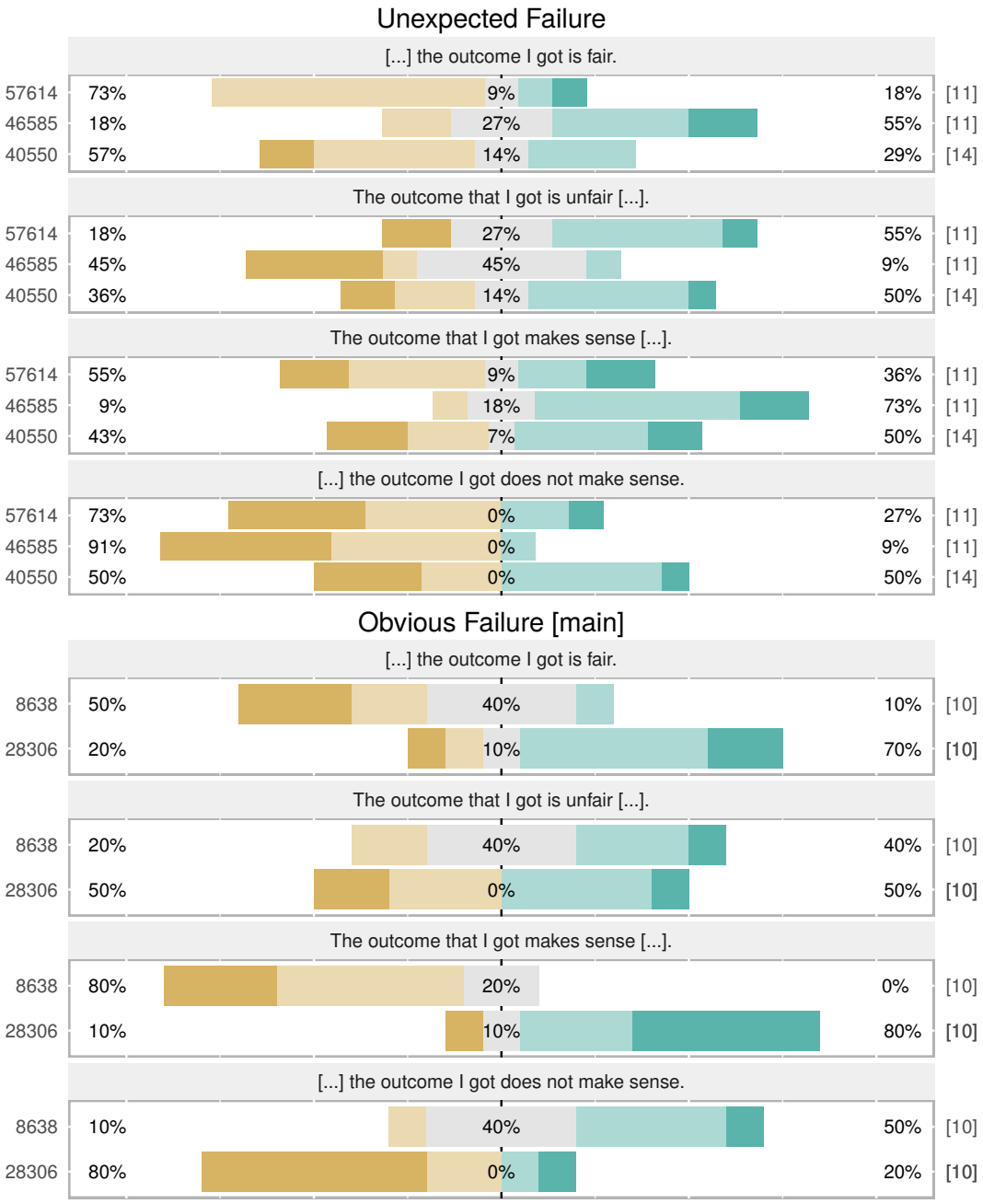

Figure 8.15: Results for the fairness and sense items in the "`unexpected_failure`" and "`obvious_failure[main]`" conditions. Bracketed numbers on the right indicate sample sizes for each row. Scale as per previous figures.

would ever be a forgone conclusion. The same is true in choice 28306: the attack option involves fighting, which is an action often prone to unexpected results, and furthermore, the advantage ascribed to the player in the option text is only marginal: possession of a weapon when both parties have the relevant skill.

Besides the fact that these choices don't set up strong expectations with their option text, they also don't have unreservedly negative results. For example, in the third outcome of choice 46585, while it's true that failing to get rid of an accusation as intended is a bad thing, it's not necessarily awful: maybe the player will get another chance to address the situation before something really bad happens. Similarly in outcome three of choice 28306, despite sustaining an injury the overall battle ends in a draw (which the player might further perceive

You come to a busy market and decide to browse for a bit.
A merchant seems knowledgeable and a peasant is bored and a noble is baselessly accusing the merchant of disrespect.
What do you do?

1. *You talk the noble down*
   (You have skill: negotiation, and you have skill: storytelling).
   6 → *You talk with the noble, but the noble refuses to back down. The noble has started to yell at you.*
2. *You transform the noble into a chicken*
   (You have skill: sorcery. You have your ancient grimoire).
   0 → *You try to cast a spell on the noble, but nothing happens. The situation remains the same.*
3. *You deny that the merchant is guilty and argue that the noble's accusation is baseless*
   (The noble is missing skill: negotiation. You have skill: negotiation).
   5 → *You argue that the merchant has been wrongly accused, but the noble remains convinced that the merchant is guilty. The situation remains the same.*

Figure 8.16: The "`unexpected_failure`" choice with seed 46585. Boxed numbers indicate the number of participants who selected each outcome.

As you continue your journey, an ogre approaches you, stomping its feet. She is threatening you.
What do you do?

1. *You attempt to pacify the ogre with music*
   (You have skill: musician. You have no tool for music).
   1 → *You weave a soothing melody, and the ogre becomes calm. The ogre is no longer threatening you.*
2. *You try to flee from the ogre*
   (The ogre has skill: wilderness lore. You are missing skill: wilderness lore).
   1 → *You flee from the ogre and escape. The ogre is no longer threatening you.*
3. *You attack the ogre*
   (The ogre has skill: fighting. You have skill: fighting. She has no tool for fighting. You have your spear).
   10 → *You attack the ogre. You strike at the ogre and then the ogre fights back. Finally you and the ogre withdraw from combat, exhausted. You are now injured.*

Figure 8.17: The "`obvious_failure`" choice with seed 28306. Boxed numbers indicate the number of participants who selected each outcome.

as ending the ogre's threat, although in the system's representation of the world the threat persists). While these *are* negative outcomes (and the results for the statement "I got a bad outcome" show that most participants agreed on this), are they negative enough to be considered "unfair" with respect to the expectations set up by the option text? The subjects in the study didn't think so, and thus the hypotheses that these conditions would be perceived as unfair and possibly broken were not supported.

In *Dunyazad*'s internal calculus, these options all simply "suggest success," and "result in failure" but evaluating them in human terms shows that the expectations that they set up are limited. The contrast between these options and ones which players overwhelmingly found to be unfair (such as option two of

You come across some bandits attacking a merchant.
The bandits are threatening the merchant.
What do you do?

1. *You talk the bandits down*
   (You are missing skill: negotiation).

   [1] → *You reason with the bandits and the bandits calm down. The bandits are not threatening the merchant any more.*

2. *You transform the bandits into a chicken*
   (You have skill: sorcery. You have your ancient grimoire).

   [10] → *You try to cast a spell on the bandits, but nothing happens. The situation remains the same.*

3. *You attack the bandits*
   (The bandits have skill: fighting. You are missing skill: fighting. They have no tool for fighting).

   [0] → *You attack them. You are hit by their attack but then you dodge their attack, then you dodge their attack. They are defeated. You have acquired an ancient grimoire, the bandits are no longer threatening the merchant, and they become injured.*

Figure 8.18: The "`obvious_failure`" choice with seed 8638. Boxed numbers indicate the number of participants who selected each outcome. This is an example of *Dunyazad* successfully creating a choice that participants viewed as unfair.

choice 8638, shown in fig. 8.18) is illustrative. Choice 8638 combines an obvious option structure (so players feel they are making the correct decision by picking option two) with a disappointing failure. The desire to see the results of the polymorph spell might have something to do with the strong reactions to choice 8638, but the fact that the option text of option two mentions two positive factors and no negative ones is likely important as well. Based on these results, if an author is intentionally trying to create a choice that is perceived as unfair, it seems that being subtle when suggesting a positive outcome is not the most productive approach.

8.6.5 Regret and Alternatives

Another unexpected result from an unexpected failure case was the fact that participants weren't dissatisfied with their decisions at "`obvious_failure`" choices. In retrospect, these hypotheses were poorly thought out: the "`obvious_failure [main]`" case should be one in which a participant selects what appears to be the only available good option, but is given a negative outcome. Although such outcomes should be perceived as bad (which they were), the questions about satisfaction were asking whether participants *would rather have picked a different option.* If a participant truly thinks that the option they picked is the only good option, then despite a bad result, they would not believe that a different option might lead to a better outcome. However, it's not the case that they *clearly* wouldn't want to try a different option—the outcome they got *was* bad—so neither enthusiastic agreement nor firm denial seem like appropriate responses.

The answers that I got reflected these competing impulses: the "satisfied" and "dissatisfied" statements both received a substantial number of neutral responses in the "`obvious_failure[main]`" condition, and the non-neutral responses were about evenly split overall, as shown in fig. 8.19. Figures 8.17 and 8.18 (just shown) show the two choices that had any "`[main]`" responses in the "`obvious_failure`" condition; the data in fig. 8.19 all comes from the 20 participants who chose the most popular options at one of those choices. While both choices are split, choice 28306 leans a bit more towards satisfaction, which may reflect being perceived as more fair, as just discussed. Ultimately, the hypotheses that the "`obvious_failure[main]`" case would leave players feeling dissatisfied with their choice was shortsighted, and the mixed results that I got should have been expected given the tension between a bad result and a lack of viable alternatives produced by these choices.

8.6.6 Expectations and... Expectedness

The differences between choices 8638 and 28306 in terms of perceived fairness, brokenness, and satisfaction have been discussed, but there is one more pair of hypotheses relating to these choices that was not confirmed: the hypotheses stating that their results would be unexpected. However, the issue of their differing expectations has already been discussed in relation to perceptions of fairness. If choice 28306 is perceived as more fair because it gives rise to weaker expectations of success, then its negative result should be less unexpected than that of choice 8638. Looking at the bottom two graphs in fig. 8.19, this is exactly what was observed.

The weak positive expectations in choice 28306 are thus largely responsible for the failure of the "`obvious_failure[main]`" sub-case's results to be viewed as unexpected. Of course, another factor was the makeup of choice 95923, the third "`obvious_failure`" choice, which did not contribute any samples to the "`[main]`" case. If just choice 8638 is considered, the hypotheses that its outcome was unexpected are confirmed ($p = 0.01337$, effect size 72% for the "not expected" hypothesis and $p = 0.014$, effect size 72% for the "was unexpected" hypothesis) even with only 11 samples (including the non-"`[main]`" sample). Had all three choices in the "`obvious`" condition produced results like choice 8638 (both in terms of participants choosing the expected option and the reactions to survey statements) the hypothesis that these types of outcomes were unexpected would have been confirmed.

The final unconfirmed hypothesis from the unexpected negative outcome column was that the "`unexpected_failure`" condition would produce unexpected results. The full hypothesis is actually half-confirmed, because participants did significantly disagree that the results were *expected*, but they didn't agree that they were *unexpected*. Figure 8.20 shows the results for both items in this condition, and it implicates choice 46585 as the culprit. This choice was

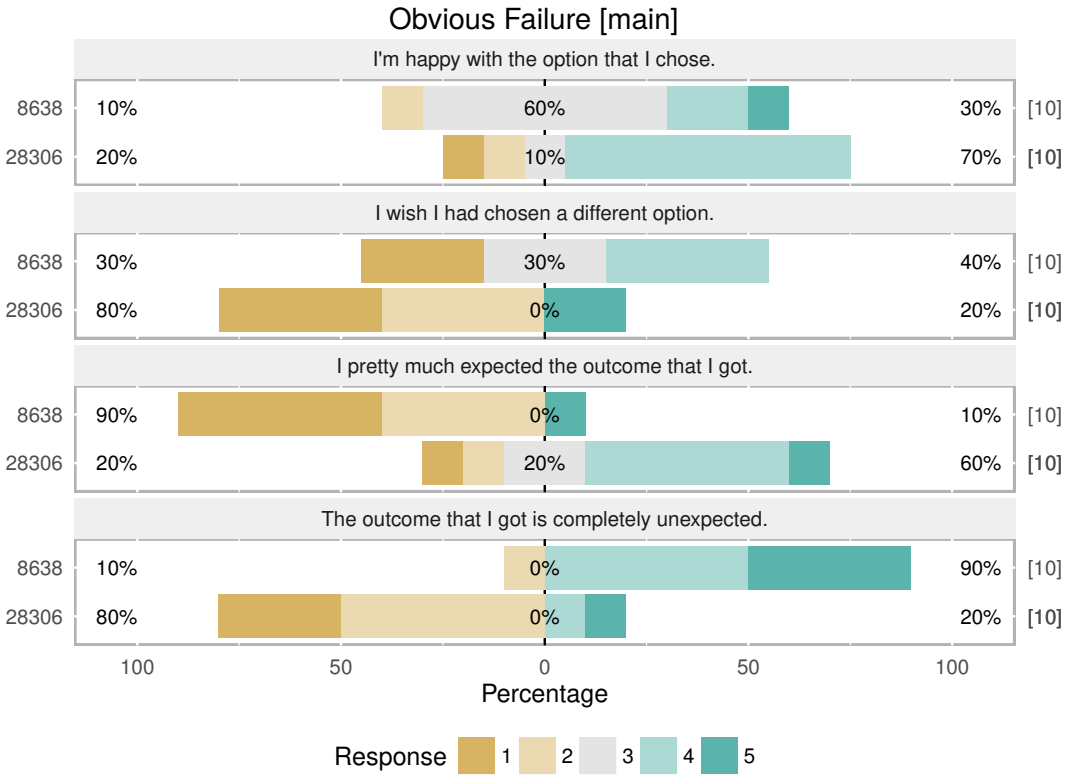

Figure 8.19: Satisfaction and expectedness in the "`obvious_failure[main]`" condition. Bracketed numbers on the right indicate sample sizes for each row.

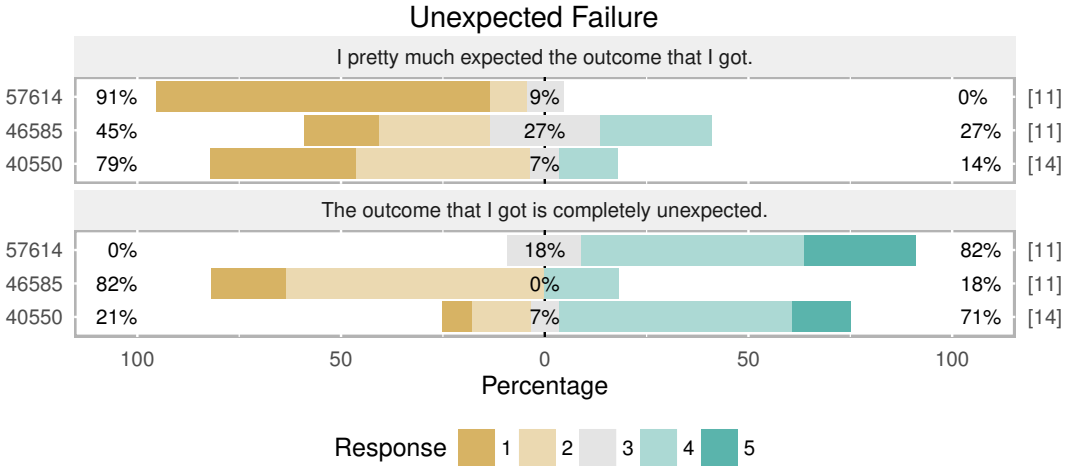

Figure 8.20: Results for the expectedness items in the "`unexpected_failure`" condition. Bracketed numbers on the right indicate sample sizes for each row.

already discussed in the context of weak expectations, and is shown in fig. 8.16. That context is not a coincidence: the fact that the actions that it involves are inherently open-ended means that participants choose them with some doubt in their minds. The difference between the positive and double-negative cases here is likely due to the outcomes at choice 46585 being neither expected nor unexpected. Falling somewhere in between, they elicited disagreement with both the statement that they were "pretty much expected" and the statement that they were "completely unexpected," thus helping confirm the positive hypothesis while working against the double negative hypothesis.

8.7 Comparative Results

There is much more to be said about the results of the outcome hypotheses and *Dunyazad*'s overall performance (see section 8.9), but first, it is useful to examine the results for the between-condition hypotheses. These hypotheses were grouped into five topics which each probed for evidence of a different proposed effect, as discussed in section 8.3.3.

8.7.1 Free vs. Forced Unexpected Failure

The first group predicted broadly that unexpected failure would be perceived as more acceptable when it seemed to be the result of a free choice rather than a forced one. Stated another way, participants who experienced failure at a choice where all options seemed good might be more likely to blame themselves for picking a "wrong" choice, while participants who picked an option that seemed to be clearly better than the alternatives might blame the system, accusing it of being unfair or broken and generally viewing their lot as worse.

The results for the 10 individual hypotheses underlying this prediction are shown in the top half of table 8.11. Unfortunately, all but two of them were

Question	Hypothesis	p-value	Effect
Fair	unexp. failure>obv. failure [main]	0.640	×
Unfair	unexp. failure<obv. failure [main]	0.292	×
Sense	unexp. failure>obv. failure [main]	0.339	×
Broken	unexp. failure<obv. failure [main]	0.164	×
Good	unexp. failure>obv. failure [main]*	0.916	×
Bad	unexp. failure<obv. failure [main]*	0.902	×
Satisfied	**unexp. failure<obv. failure [main]***	**7.7×10^{-4}**	**75%**
Dissatisfied	**unexp. failure>obv. failure [main]***	**3.7×10^{-5}**	**80%**
Expected	unexp. failure>obv. failure [main]*	0.945	×
Unexpected	unexp. failure<obv. failure [main]*	0.539	×

Question	Hypothesis	p-value	Effect
Fair	unexp. failure>8638 [main]	0.116	×
Unfair	unexp. failure<8638 [main]	0.233	×
Sense	**unexp. failure>8638 [main]**	**0.004**	**77%**
Broken	**unexp. failure<8638 [main]**	**0.005**	**76%**
Good	unexp. failure>8638 [main]	0.305	×
Bad	unexp. failure<8638 [main]	0.606	×
Satisfied	**unexp. failure<8638 [main]**	**0.004**	**76%**
Dissatisfied	**unexp. failure>8638 [main]**	**7.9×10^{-4}**	**81%**
Expected	unexp. failure>8638 [main]	0.296	×
Unexpected	**unexp. failure<8638 [main]**	**0.018**	**71%**

Table 8.11: Retrospective hypotheses for the claim that "Unexpected failure is more acceptable when it happens at a freely-chosen option than when the player feels there are no viable alternative options." The bottom half shows the results when choice 8638 is allowed to stand in for the entire "`obvious_failure [main]`" case. Each line lists the hypothesis, the p-value, and if significant ($p < 0.05$), the common-language effect size. Low-confidence hypotheses are marked with a '*'. Note that each pair of rows contains opposing predictions, because complementary questions are arranged together.

not confirmed (and ironically, the two that were confirmed were "low-confidence" hypotheses). Despite these results, the two confirmed hypotheses show extremely strong effects, and can be taken to indicate that a subset of the originally proposed effect is at work here. Forced-choice unexpected failures may not be seen as more fair, less broken, better, or more expected than free-choice unexpected failures, but participants are definitely more-satisfied with their decisions in the forced-choice case.

Part of the reason that these hypotheses were mostly unsupported is that the choices in question didn't confirm the basic hypotheses about whether participants would agree or disagree with most of these questions (the two cases being compared here make up the right-hand column in table 8.10). Only the valence and satisfaction statement hypotheses were confirmed as a pair for either of these cases, and the hypotheses for the satisfaction statements were only confirmed for the "`unexpected_failure`" case. As discussed above, the reasons for this have a lot to do with the general failure of the "`obvious_failure[main]`" case to live up to its name. Because of the not-so-bad outcomes in the "`obvious_failure[main]`" case, not to mention the complications that the "`unexpected_failure`" case had, the comparison between the "`unexpected_failure`" and "`obvious_failure[main]`" cases isn't really testing what these hypotheses assumed it would be.

Comparing the "`unexpected_failure`" case against just the most emblematic choice from the "`obvious_failure[main]`" case (choice 8638) gives a slightly different results, shown in the bottom half of table 8.11. We can see that the prediction about fairness was not supported (although it wasn't firmly rejected either), but the prediction about sense/nonsense holds up. The valence prediction is still unsupported, and the expectedness hypotheses are split, but overall it seems that the original idea wasn't wrong, but just overly broad. Although more data would be needed to pin down this effect, it seems clear that the perception of alternatives to a chosen option as viable or not has an effect on how surprising

negative outcomes are perceived. If options perceived as viable alternatives are available, players will be both less satisfied with their decision (unsurprisingly) but they will also be more likely to label the result as one that makes sense, and less likely to label it as surprising (though not more likely to label it as expected).

8.7.2 Chosen vs. Inevitable Success

A second proposed between-conditions effect was that cases where success seemed to be the result of choosing a correct option would seem to have *better* outcomes than cases where every option seemed likely to be successful, although both conditions were expected to be rated as generally good. This proposed effect extended to the satisfied/dissatisfied items as well, but the results of testing these hypotheses, shown in table 8.12, fail to support this claim. A look at the distribution of responses to these questions, shown in fig. 8.21, reveals one possible reason for this: the results are simply oversaturated.

With the data I collected, the proposed subtle differences could have shown up as a shift from "somewhat agree" to "strongly agree" between the "`expected_success`" and "`obvious_success[main]`" conditions, but it turned out that even in the "`expected_success`" condition, which I predicted to score lower, the responses were overwhelmingly[2] "strongly agree." To get data that might show such an effect, one approach would be to run a study using a 7- or 9-point Likert scale and hope that variation showed up in the upper regions of the scale. Another approach would be to ask participants to directly rate *how* good or satisfied they were, and provide response scales that included gradations of "good" such as "good—great—excellent."

[2]Note that although the responses to choice 20739 stand out, they represent only 2 of 25 samples in the "`obvious_success[main]`" case, and thus don't have a meaningful effect on the statistics of these hypotheses. The problems with choice 20739 (a costly choice paired off against a "`travel_onwards`" choice) have already been discussed.

Question	Hypothesis	p-value	Effect
Good	exp. success<obv. success [main]	0.942	×
Bad	exp. success>obv. success [main]	1.000	×
Satisfied	exp. success<obv. success [main]	0.725	×
Dissatisfied	exp. success>obv. success [main]	0.791	×

Table 8.12: Retrospective between-conditions hypotheses concerning differences between expected successful outcomes in cases where the alternative options are either both positive or both negative.

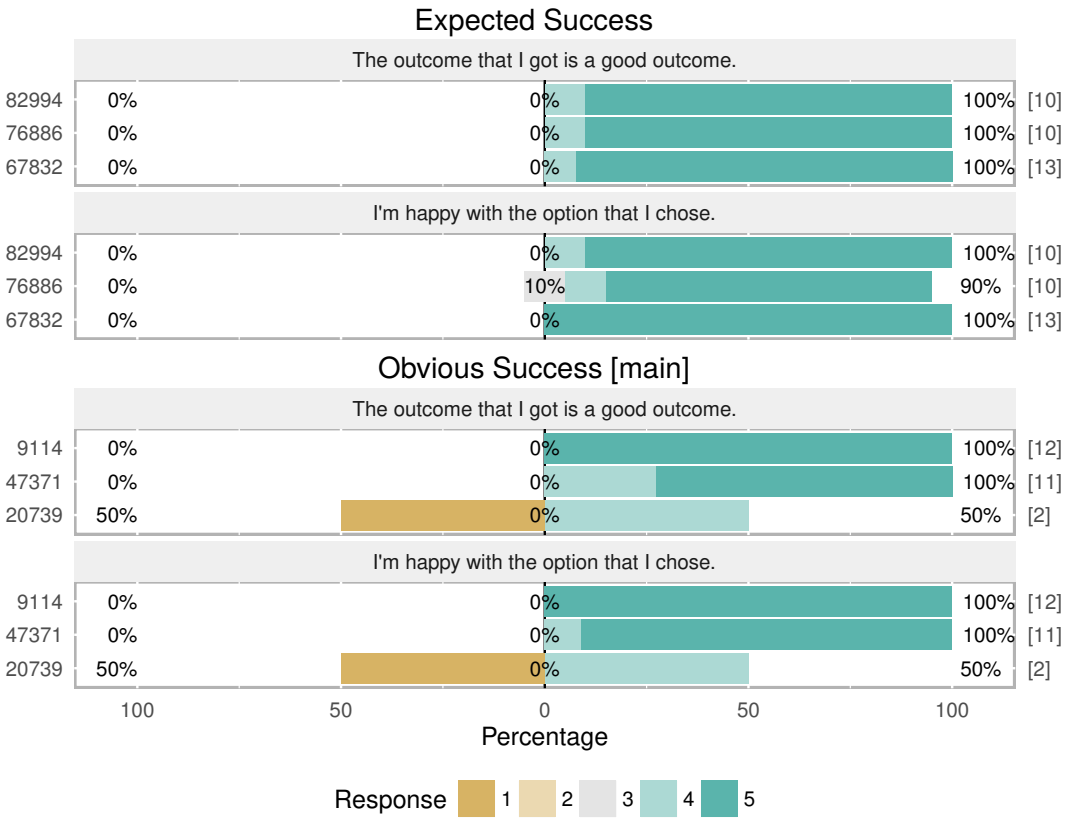

Figure 8.21: Summary of the data in the "`expected_success`" and "`obvious_success[main]`" conditions for the "good" and "satisfied" items. The "bad" and "dissatisfied" items are not show but are similarly one-sided in the opposite direction. The last rows in the "`obvious_success[main]`" case are negligible as they contain only 2 of the 25 samples in each plot (the problems with choice 20739 were discussed earlier).

Question	Hypothesis	p-value	Effect
Fair	**unexp. failure<unexp. success**	$\mathbf{4.5{\times}10^{-9}}$	**86%**
Unfair	**unexp. failure>unexp. success**	$\mathbf{5.7{\times}10^{-5}}$	**75%**
Sense	**unexp. failure<unexp. success**	$\mathbf{3.7{\times}10^{-4}}$	**72%**
Broken	**unexp. failure>unexp. success**	**0.004**	**67%**

Table 8.13: Results for hypotheses asserting that unexpected positive outcomes will be perceived as more fair and less broken than unexpected negative outcomes.

8.7.3 Good vs. Bad Unexpected Results

Table 8.13 shows the results for another proposed effect: that unexpected good results would be seen as more fair and less broken than unexpected bad results. In other words, when a result is the opposite of what is suggested by option text, if it's a good result players will tend to accept it, but if it's a bad result, they will tend to protest. This is not a controversial hypothesis, and it was confirmed on all counts by the data. An interesting followup would be to attempt to assess whether gradations of better and worse results have a linear effect on perceptions of fairness and sense or whether there is some kind of inflection point between the extremes used in this study.

8.7.4 Expected vs. Unexpected Failures

Another uncontroversial prediction was that unexpected failures would feel worse than expected failures. This proposed effect is in line with theories of outcome evaluation including both decision affect theory and consistency theory, and has been confirmed in non-game scenarios (see e.g., (Shepperd and McNulty, 2002)). I predicted that not only the valence items but also the satisfaction items would be affected, and compared both expected success conditions ("`unexpected_failure`" and "`obvious_failure[main]`") against the "`expected_failure`" condition.

The results, shown in table 8.14, confirm most of the hypotheses, except for the satisfaction hypotheses relating to the "`obvious_failure[main]`" vs. "`expected_failure`" comparison. In light of the "`obvious_failure[main]`" condition's mixed satisfaction results, the lack of confirmation for the last two hypotheses is not surprising, in fact it would be surprising if they had been confirmed, because the "`obvious_failure`" cases include competing influences for the satisfaction items, as discussed above. The data thus agree with existing non-game results suggesting that people perceive unexpected negative outcomes as being worse than expected negative outcomes, even when the outcomes are identical. Furthermore, when all available alternatives seem positive, failure provokes a stronger desire to have chosen a different option than when all alternatives seem negative. Of course, this last effect is more likely due to the valence of the alternatives than the expectedness of the outcome, which would explain why it did not carry over into the "`obvious_failure[main]`" case.

Unfortunately, although the hypotheses here are supported by the data and agree with existing literature, there may be another cause for my findings: several of the choices involved in these conditions have already been called out for not

Question	Hypothesis	p-value	Effect
Good	**unexp. failure<exp. failure**	$\mathbf{3.0{\times}10^{-4}}$	**72%**
Bad	**unexp. failure>exp. failure**	$\mathbf{3.3{\times}10^{-5}}$	**76%**
Satisfied	**unexp. failure<exp. failure**	**0.006**	**67%**
Dissatisfied	**unexp. failure>exp. failure**	$\mathbf{2.6{\times}10^{-5}}$	**77%**
Good	**obv. failure [main]<exp. failure**	**0.040**	**64%**
Bad	**obv. failure [main]>exp. failure**	**0.004**	**72%**
Satisfied	obv. failure [main]<exp. failure	0.783	×
Dissatisfied	obv. failure [main]>exp. failure	0.815	×

Table 8.14: Results for hypotheses predicting that unexpected failures will be viewed as more negative than expected failures.

producing their intended poetic effects. These hypotheses were supposed to look for gradations of badness between choices whose results were uniformly viewed as bad, but the choices that *Dunyazad* constructed for these cases didn't turn out that way. The data for the "good" item in all three relevant conditions, shown in fig. 8.22, reveal that several choices are acting up (similar anomalies appear for the "bad" item). As already discussed, choices 87991 and 8015 included "`travel_onwards`" options which resulted in mixed evaluations, and choices 46585 and 28306 both had weak expectations and somewhat mixed outcomes. In this case these problems happened to push things in favor of confirming the hypotheses, but including the aberrant choices changes the meaning of the hypotheses, as they are no longer clearly comparing failures to failures.

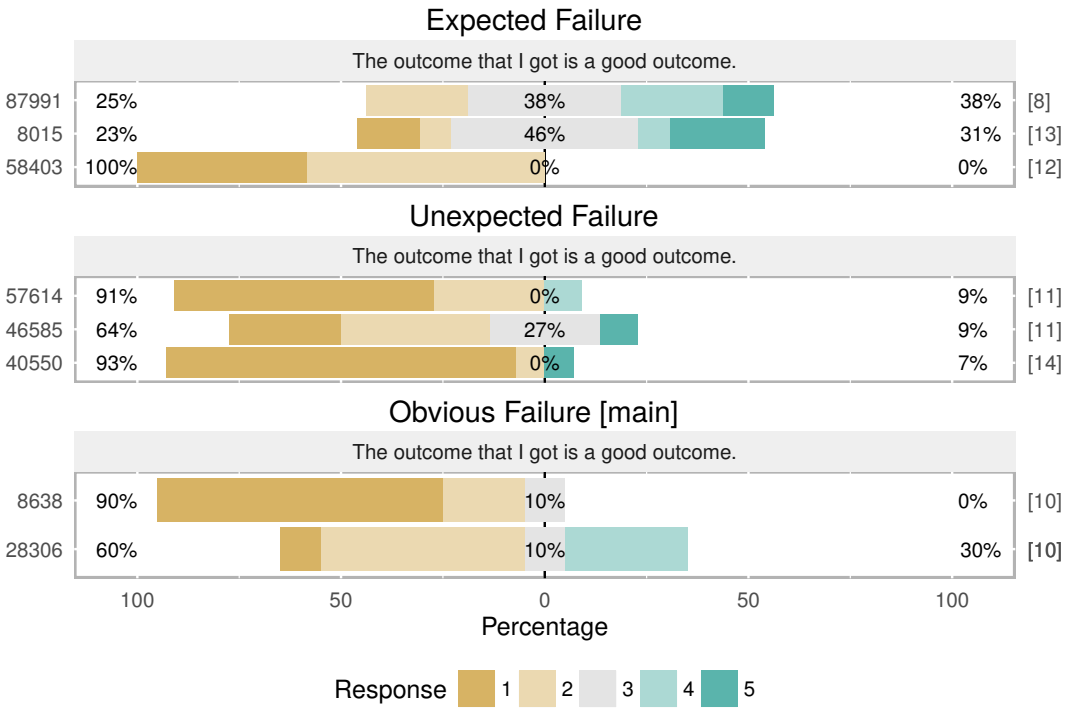

Figure 8.22: Summary of the data in the "`expected_failure`," "`unexpected_failure`," and "`obvious_failure[main]`" conditions for the "good" item. All of these should have elicited general disagreement, but choices 87991, 8015, and to some degree 28306 did not live up to this expectation.

The results after filtering out the suspect choices are shown in table 8.15. Although the effect is still (narrowly) confirmed for the "`expected_failure`" vs. "`unexpected_failure`" case, the "`obvious_failure[main]`" case no longer shows a strong effect. Interestingly, the effect seems to be significantly stronger for negatively-worded questions than for positively-worded ones. Whether or not the option structure of the obvious case interferes with this effect for the valence items as it almost certainly does for the satisfaction items is open for debate, the data collected here don't support either argument.

8.7.5 Expected vs. Unexpected Success

A corollary of the previous proposed effect in decision affect theory is that unexpected positive results will be viewed as more-positive than expected positive results (consistency theory in this case would disagree). I also tested this prediction against the data, but ran into the same problem that I did when trying to compare chosen vs. inevitable success: the data was oversaturated.

Question	Hypothesis	p-value	Effect
Good	**57614+40550<58403**	**0.048**	**65%**
Bad	**57614+40550>58403**	**0.007**	**73%**
Satisfied	**57614+40550<58403**	**0.049**	**66%**
Dissatisfied	**57614+40550>58403**	$\mathbf{1.2\times10^{-4}}$	**85%**
Good	8638(main)<58403	0.185	×
Bad	8638(main)>58403	0.196	×
Satisfied	8638(main)<58403	0.971	×
Dissatisfied	8638(main)>58403	0.722	×

Table 8.15: Results for hypotheses predicting that unexpected failures will be viewed as more negative than expected failures, using only choices which fit the expected structure of each condition. Compare with table 8.14.

Figure 8.21 demonstrated this for the "`obvious_success[main]`" and "`expected_success`" cases, fig. 8.23 shows that this carries over to the "`unexpected_success`" case as well (data for the three other items involved in these hypotheses are not shown, but were similarly saturated). Although there were a few responses that were affected by *Dunyazad*'s quirks, even e.g., removing choice 99500 from consideration does not change the statistical results. There was simply not enough room on the scale for a subtle shift in valence to register. As with the chosen vs. inevitable success case, running a study with broader scales or asking for a direct evaluation of the degree of goodness/satisfaction would be viable techniques for pursing this effect further. In any case, the data that I gathered do not provide enough information to argue either for or against the presence of this effect.

8.8 Motivation Results

The last few hypotheses in the retrospective study concerned the answers to the motive questions. The results for these are shown in table 8.17, and the actual data is graphed in fig. 8.24. The hypotheses aren't statistical in nature, and just

Question	Hypothesis	p-value	Effect
Good	exp. success<unexp. success	0.954	×
Bad	exp. success>unexp. success	1.000	×
Satisfied	exp. success<unexp. success	0.961	×
Dissatisfied	exp. success>unexp. success	0.960	×
Good	obv. success [main]<unexp. success	0.633	×
Bad	obv. success [main]>unexp. success	0.747	×
Satisfied	obv. success [main]<unexp. success	0.847	×
Dissatisfied	obv. success [main]>unexp. success	0.750	×

Table 8.16: Results for hypotheses predicting that unexpected success will be seen as more positive than expected success.

serve to establish some expectations to compare against. Each (except the last) essentially names an aspect of motivation that is predicted to be widespread and asks if a simple majority of participants indicated that they used/experienced that motivation. For example, the `avatar` motive—which corresponds to viewing situations from the point of view of the player's character, i.e., from within the story—was predicted to be commonly employed in making decisions as well as in evaluating individual options as both good and bad.

As a reminder, tables 8.18 and 8.19 show the answers that correspond to the labels used in fig. 8.24. Another view of the motive responses is presented in figs. 8.25 and 8.26. These each graph unique response combinations that individually accounted for at least 5% of all responses (i.e., at least 11 responses).

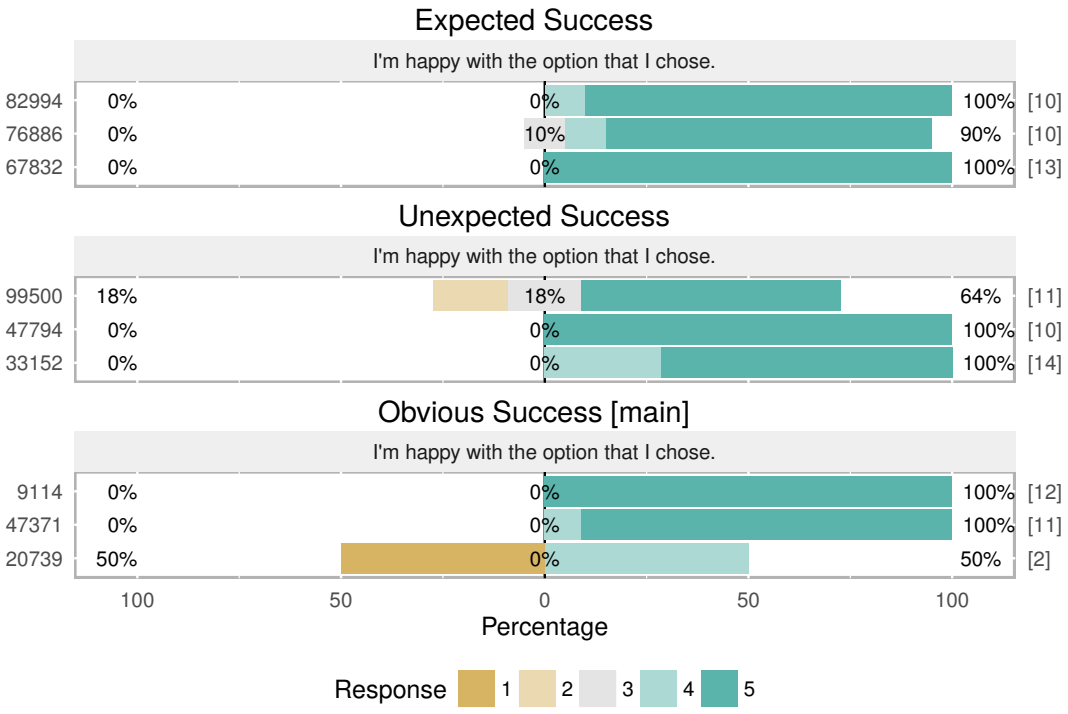

Figure 8.23: Summary of the data in the "`expected_failure`," "`unexpected_failure`," and "`obvious_failure[main]`" conditions for the "good" item. All of these should have elicited general disagreement, but choices 87991, 8015, and to some degree 28306 did not live up to this expectation.

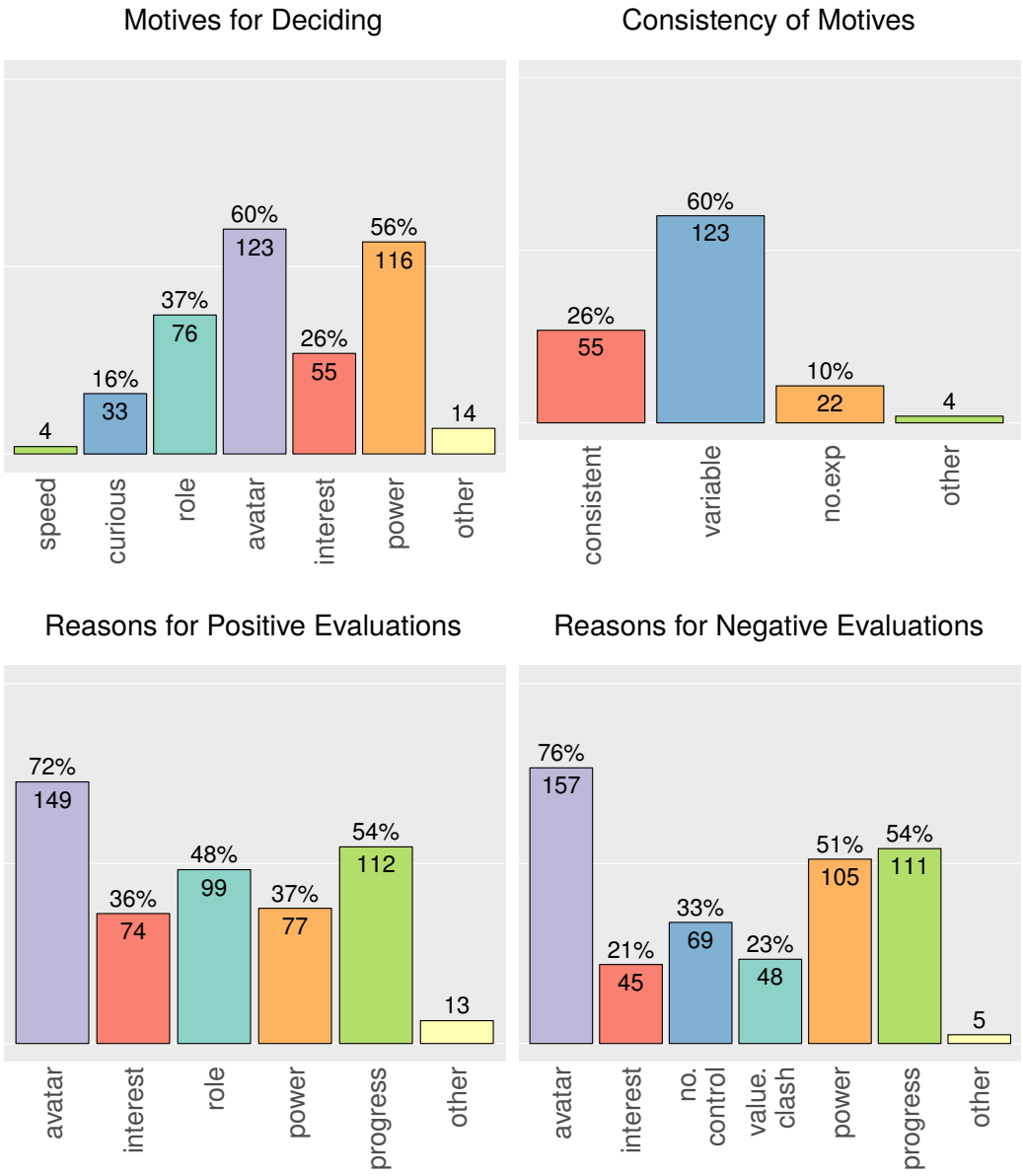

Figure 8.24: Histograms of responses to the four motive questions. The total number of responses is 205 in all cases, except for the consistency question where there was a single missing response. However, all of the other questions allowed multiple responses, so their counts do not sum up to 205. The middle background line in all cases is placed at 102.5, or 50% of the maximum possible count.

Note that each bar in these graphs represents exact matches only, so for example an "avatar + power" bar would not include an "avatar + power + role" response. These graphs thus show the most-common unique player-types, rather than the most common motivations across player-types.

This data should be taken with a grain of salt: it's self-reported and not backed up by other measures which could reveal potential biases. The top two categories among reasons for positive evaluations, for example, are the first option alone and "all of the above." However, the strong grouping of results is an indication that responses aren't simply random, and in the motivation question the `avatar` category alone is most popular despite being the third option.

Question	Hypothesis	Count	Total	Percentage	Result
[Which] motive(s) contributed to your decision? [...]	`speed` $> 50\%$	4	205	2%	×
	`avatar` $> 50\%$	123	205	**60%**	✓
	`power` $> 50\%$	116	205	**57%**	✓
[How do you define a "good" outcome]?	`avatar` $> 50\%$	149	205	**73%**	✓
	`power` $> 50\%$	77	205	38%	×
	`progress` $> 50\%$	112	205	**55%**	✓
[How you define a "bad" outcome]?	`avatar` $> 50\%$	157	205	**77%**	✓
	`no.control` $> 50\%$	69	205	34%	×
	`power` $> 50\%$	105	205	**51%**	✓
	`progress` $> 50\%$	111	205	**54%**	✓
Do [you] approach [all experiences] [the same way]?	`variable` $> 70\%$ of non-`no.exp`	123	183	67%	×

Table 8.17: A table of the motive-related hypotheses and their results. Note that except for the last question, all questions allowed multiple responses to be selected, so the numbers don't add up to the total (also, not all possible answers had a corresponding hypothesis). These are simple true/false outcomes, as the hypotheses just predict that a certain fraction of participants will select a particular outcome.

In terms of the hypotheses, the first result is the most divergent from the prediction, but in this case it's potentially a good thing. Only 4 out of 205 participants indicated that they made a decision just to get the survey over with. Of course, there's some motivation to lie here, even despite the explicit declaration that this answer would not affect acceptance, because the decision to accept or reject a submission is completely up to the requester and can have a large impact on a worker's qualifications for tasks (as in this study, a common qualification criteria is % acceptance rate). The other two predictions about motives were that `avatar` and `power` would be popular, and these turned out to be true. As noted in item 1 when talking about goal analysis, *Dunyazad*'s fixed goals are targeted at avatar play and power play as modes of engagement, and so confirming these hypotheses shows that *Dunyazad* is on the right track.

The predictions about positive evaluations were that the `avatar`, `power`, and `progress` considerations would be popular. As it turned out, `avatar` and `progress`

Motive	`speed`	*I'm taking an online survey. I just chose an option quickly so that I could complete the survey.*
	`curious`	*I was just curious to find out what would happen if I chose the option I did.*
	`role`	*I imagined a character in the story situation and chose what that character would do.*
	`avatar`	*I chose what I would have chosen were I in the situation described in the story.*
	`interest`	*I chose the option that I thought would lead to the most interesting/satisfying result.*
	`power`	*I looked at the skill information and chose the option that I thought would be most successful.*
	`other`	*Other (please explain):*

Table 8.18: Answers to the motive question by label.

Good When	`avatar`	*I feel an outcome is good when something good happens to my character in the story world.*
	`interest`	*I feel an outcome is good if it is an interesting development in the story.*
	`role`	*I feel an outcome is good when it fits the role that I am building for my character.*
	`power`	*I feel an outcome is good when it makes my character more powerful.*
	`progress`	*I feel an outcome is good when it makes progress towards beating a game.*
	`other`	*Other (please explain):*
Bad When	`avatar`	*I feel an outcome is bad when something bad happens to my character in the story world.*
	`interest`	*I feel an outcome is bad if it doesn't develop the plot.*
	`no.control`	*I feel an outcome is bad when my character doesn't do what I expected them to do.*
	`value.clash`	*I feel an outcome is bad when my character expresses a value or opinion that is different from what I wanted them to express.*
	`power`	*I feel an outcome is bad when it makes my character less powerful.*
	`progress`	*I feel an outcome is bad when it prevents me from making progress towards beating a game.*
	`other`	*Other (please explain):*

Table 8.19: Answers to the evaluation questions by label.

were both highly popular considerations, but `power` was not, being less popular than as `role` and effectively tied with `interest`. The predictions about negative evaluations were also all correct but one. `Avatar`, `power`, and `progress` were all popular negative evaluation criteria, but `no.control` was unexpectedly not: only 33% of participants indicated that it was an important consideration for them.

The results for the consistency question were interesting. The prediction was that a strong majority (in this case 70%) of participants would select the `variable` response over the `consistent` response, not counting participants who selected `no.exp`. The result was that of the participants who felt qualified to respond, 67% selected `variable`, while 30% selected `consistent`. This was a larger proportion of `consistent` players than anticipated; overall most of the results showed more diversity than I expected. Note that 90% of participants felt

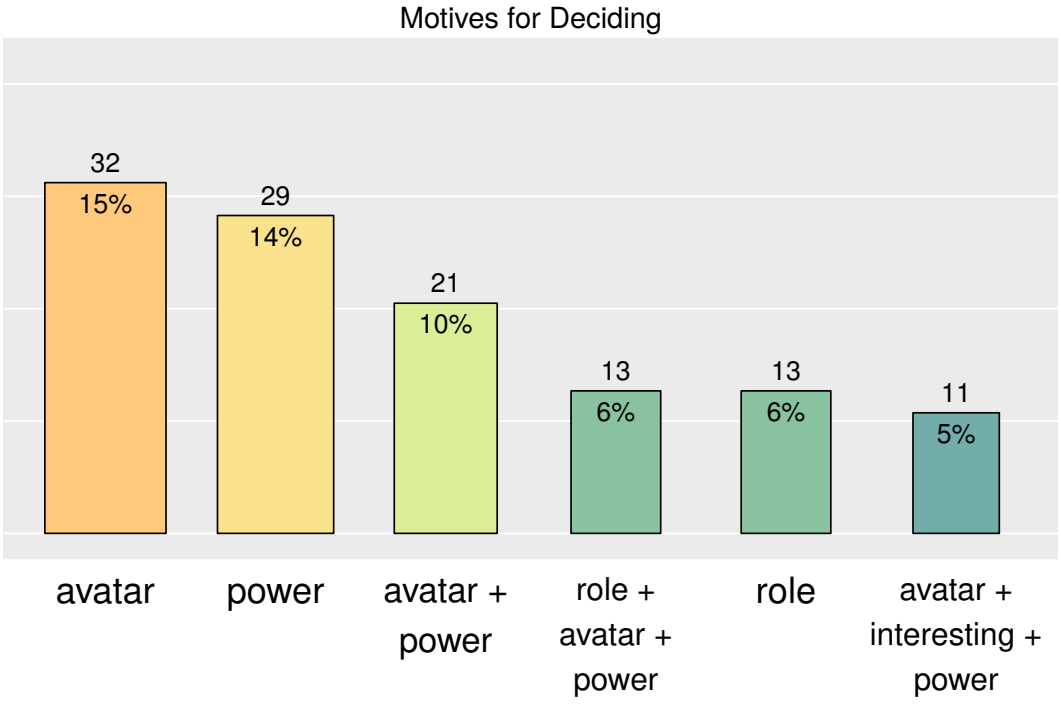

Figure 8.25: Unique motive combinations accounting for at least 5% of the total responses. There were a total of 45 unique response combinations.

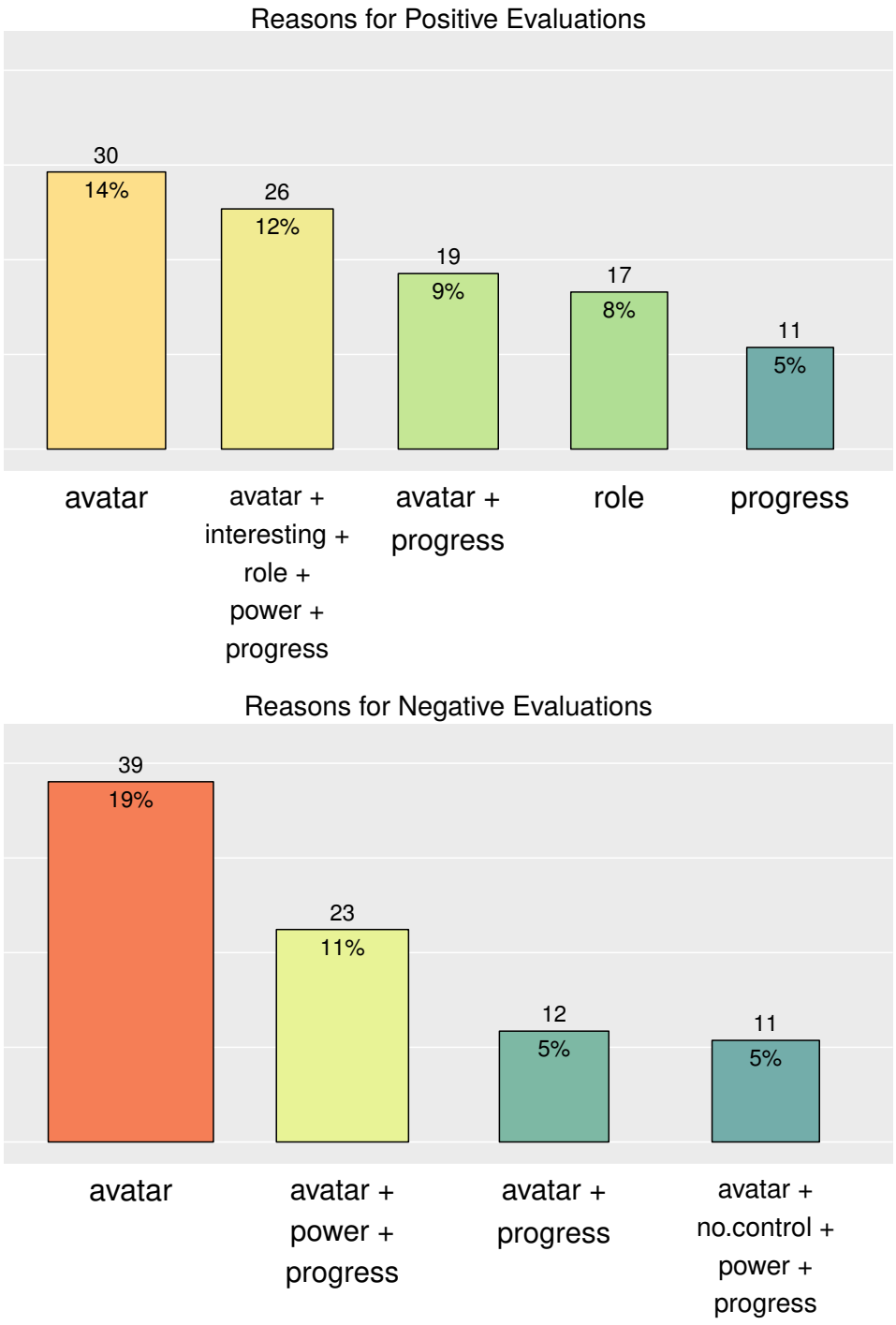

Figure 8.26: As fig. 8.25. There were 35 and 50 unique combinations respectively.

qualified to answer this question, indicating at least some experience with a form of interaction fiction, although there may be some self-selection involved in that, and Amazon Mechanical Turk workers are certainly not a very representative sample. This does seem encouraging for the use of Mechanical Turk in studies of game-related behavior and opinions, however.

8.9 Discussion

Looking at all of the results of the retrospective experiment, the main trend seems to be that while *Dunyazad* was largely successful at creating positive impressions, it has trouble creating negative and contrasting impressions, sometimes making a dilemma when it wants to create an obvious choice or vice versa. This mainly stems from trouble with goal estimation: it's two-level priority system is too simple, and it simply doesn't model some common player concerns like resource costs. Luckily, these shortfalls have clear avenues of redress in terms of technical implementation. It's also the case that these shortfalls don't fundamentally invalidate that goal-based choice analysis method proposed in section 5.4. After all, when *Dunyazad doesn't* make a mistake, it can be eerily good at creating choices that a variety of people respond to in the same way: an obvious choice like choice 47371 can convince 11/13 participants to choose the option that the system thinks is best. Although *Dunyazad* is somewhat ham-handed in its construction of options, remember that its only guidance in selecting actions and outcomes is the operationalization of goal-based choice analysis described in section 6.2. The success of *Dunyazad*'s better choices is convincing proof that the goal-based choice analysis method offers at least some insight into how players make decisions.

As for the high level effects which the between-conditions hypotheses probed for in this study, although several of them were rendered inscrutable by *Dunyazad*'s limitations, there were some interesting confirmed results. First, it

seems that when confronted with unexpected failure, player's perceptions of brokenness, and to some degree expectedness, are sensitive to the presence or absence of viable-seeming alternatives. This is not terribly surprising, but confirmation of the effect opens the doors for new questions that could be the subject of a more specific study, such as how gradations in the desirability of alternatives might affect perceptions of brokenness.

A second high-level result was a confirmation of decision affect theory's assertion that unexpected failure should feel worse than expected failure (Mellers, A. Schwartz, and Ritov, 1999) in the context of a narrative game. While again, confirming a well-studied psychological phenomenon like this is not surprising, context matters, and uncritically assuming that in-game decision making will exactly mimic everyday real-life decision-making is flawed. The results also go both ways: confirming this effect is another sign that *Dunyazad* is actually creating both expected and unexpected results. Unfortunately, the same hypothesis applied to successful results was not supported by the data, but as mentioned, this was an issue of scale saturation rather than actually conflicting results.

The final component of the study was designed to give a rough sense of player's motivations, and the results indicated a substantial diversity of motives. Perspective-taking was a very common motive in all categories as expected, and it was also the most popular choice for participants who had a single motive. Consideration of in-game power and progress was a close second to character-centric judgements, and role-playing (in the true sense of the words) was surprisingly common, with more than one in three participants indicating that they actively role-play when making decisions, and almost half indicating that they consider role-building as a reason for deciding when an outcome is desirable. Although these are only rough figures, they serve to highlight the importance of considering modes of engagement when designing and evaluating choices. In terms of motives for decision making, no single motive accounted for more than 60%

of participants, meaning that if a choice is designed while taking only one perspective into account, as many as four out of ten players might approach that choice with motivations that its designer never considered. In fact, because there is considerable overlap between cases, 16% of participants in this study listed neither of the two most-popular responses (`avatar` and `power`) as motives; only by including the third-most-popular motive, `role`, does coverage exceed 90%. Of course this exact mix of motives is influenced by both the sample population (Mechanical Turk workers) and the genre of the choice they just experienced, but I suspect that a similarly broad variety of motives is to be expected under most conditions where one might want to study choice poetics.

Just like the first study, the retrospective study achieved its goals. It highlights some of *Dunyazad*'s strengths and weaknesses, as well as some areas, like goal priorities and morality, that choice poetics theory could say more about. The motivation data further make a strong case for the importance of considering modes of engagement when evaluating choices, while at the same time showing that *Dunyazad*'s targeted modes of engagement are broadly present as was hoped. The results overall offer a plethora of interesting details that could be the basis for future studies, either with or without an automatic choice-point generator. In the end, this data can be used to improve *Dunyazad*, refine the theory of choice poetics, and provide clues as to productive avenues for future research.

8.10 Conclusions

Having combed through the data, there seem to be two big takeaways from these experiments. The first is that *Dunyazad* has weaknesses. Not only does it have weaknesses, but the results also show what those are, and how to fix them. In this sense, these studies are a successful product test: the system isn't perfect yet, but it does a lot, and if a few things are improved, it can do more.

The second takeaway from these experiments is that choice poetics are more complicated than I hoped but also simpler than I feared. Certainly, adding moral reasoning and relative goal analysis capabilities to *Dunyazad* are technically intimidating tasks, which may require some theoretical research of their own. But at the same time, a lot of the basics seemed to just work: when *Dunyazad* made a broken choice the reasons were clear and addressable, and when it made good choices, the responses of participants were startlingly predictable. The draw of *Dunyazad*'s well-crafted obvious answers was almost irresistible, and its dilemmas made participants feel trapped. Although it accomplished this often through drastic contrasts instead of the more subtle differences between options that a human author might use, the point is that it was able to recognize and deploy those contrasts successfully using the goal-based evaluation framework presented here. For this reason these studies are not only a confirmation of *Dunyazad*'s capabilities (and an inventory of its shortfalls) as a system, but also an endorsement of goal-based choice analysis as a useful tool for understanding how players will view choices.

8.11 Impressions Revisited

Section 5.4.6 promised that this chapter would revisit the prospective impressions that it put forward in light of experimental results. Between the first and second experiment, the dilemma, depressing, empowering, obvious, and relaxed prospective structures were used to create choices, and the feedback on those choices has exposed several ways that they can break down. Table 8.20 lists the original definitions of those choices, along with some extra details that should be taken into account during analysis.

Likewise, table 8.21 lists caveats for the retrospective impressions from table 5.4. In neither case do the results completely invalidate a definition. Instead,

results that show flaws in *Dunyazad*'s mechanical reasoning serve to highlight the human faculties that must be brought to bear on the problem. For example, *Dunyazad* currently lacks the ability to properly reason about moral injunctions, such as "You must help others," which are limited by exceptions like "Unless you must put yourself at risk to do so." Luckily, humans have this capacity, and the fact that its lack can derail the construction of an obvious choice in *Dunyazad*'s case is evidence that humans should be careful to exercise it when evaluating such choices (or when constructing them themselves). Thus each flaw in *Dunyazad*, as long as it's cause can be identified, becomes a useful warning for humans interested in choice analysis. Of course, this is exactly the property that makes answer set programming an attractive approach for *Dunyazad*'s technical implementation: using ASP makes it possible for *Dunyazad*'s implementation of choice poetics to be as transparent as possible, and thereby facilitates this process of turning problems into advice.

Label	Criteria	Caveats
Depressing	Each option **hinders** at least one top-priority goal. No option should **enable** or **advance** any top-priority goals.	Options should create clear negative expectations, especially if they have positive outcomes.
Dilemma	Exactly two options, each of which **hinders** one of two different top-priority player goals. The priorities of the goals and the severity of the consequences should be balanced and neither option should **enable** or **advance** any goals (even low-priority ones).	Proper goal balance is difficult to achieve. Moral injunctions often have exceptions that need to be considered.
Empowering	Every option **advances** a player goal, and may **threaten** one or more goals but does not **hinder** any.	If option costs are not balanced, it may collapse to "obvious."
Obvious	One option that **advances** a top-priority player goal without **hindering** any (although it may **threaten** some), while none of the rest of the options **enable** any top-priority goals, and each of them **threatens** some goal.	Resource costs must be taken into account. "Bad" options should clearly indicate which goal(s) they threaten.
Relaxed	There are no option expectations involving high-priority goals (positive or negative), and there are no **threatens** expectations (and thus no **hinders** expectations).	Slight differences between options can worry satisficers. Try to contrast stakes with those of previous choices.

Table 8.20: The prospective impression labels from table 5.3 that were used in the experiments, showing caveats exposed by the results.

Label	Criteria	Caveats
Expected Success	The selected option was expected to **advance** a player goal without **hindering** any, and its outcome is both **predictable** and **good**.	Resource costs should be taken into account. Positive outcome states can be described negatively.
Expected Failure	The selected option was expected to **hinder** a player goal and not **advance** any, and its outcome is both **predictable** and **bad**.	Badness of outcomes may be evaluated relative to expected outcomes of other choices.
Unfair	The selected option was expected to **advance** at least one top-priority player goal, while not **hindering** any. It has an **unexpected** and **bad** outcome.	May be perceived as broken if no attractive alternative options were present. Inherently chaotic actions dampen expectations of success.
Miracle	The selected option was expected to **hinder** at least one top-priority goal, while not **advancing** any. It has an **unexpected** and **good** outcome.	As for expected successes, descriptive text is important, not just "story-world result." Positive results are seen as less surprising than negative ones.

Table 8.21: Several retrospective impression labels from table 5.4, along with caveats exposed by the results.

Chapter 9

Conclusion

Throughout the previous three chapters which describe the theory of choice poetics, the design of *Dunyazad*, and results from experiments that test its capabilities, there have been mentions of "future work;" of unexplored paths. The theory developed in chapter 5 is incomplete and shallow; it doesn't directly address some of the most interesting poetic effects of choices, such a regret. The system description on chapter 6 admits that *Dunyazad* is focused on single choices for now, and doesn't have strong mechanisms for guiding plot arcs. The experimental results detailed in chapters 7 and 8 are mixed, and highlight several areas that need work. Taken together, this perhaps gives the impression that *Dunyazad* as a project is woefully incomplete.

However, recall the purpose of *Dunyazad* as a project: learn new things about choice poetics using a hybrid research method that incorporates artificial intelligence into a humanistic agenda, while producing a novel generative system centered on choices. Measured against these goals, the theoretical and technical results that have been demonstrated so far represent an important first step in a new direction. The ideas about choice poetics described in chapter 5, for example, are not ideas that would have resulted from a close reading of multiple choice-based narratives, and this is also the case for the caveats explored in chapters 7

and 8. Furthermore, *Dunyazad* as a generative system does things that no other existing system attempts, giving users a uniquely detailed level of control over the choices that it generates: it is a system that attempts to understand and provide direct control over the impressions that its choices make, and it is largely successful at this. Finally, the experimental results from chapters 7 and 8 are something that would not have been possible without *Dunyazad*: based on *Dunyazad*'s transparent operationalization of choice poetics, they are able to establish a direct link between assessments that result from particular ways of reasoning about choices and audience reactions to those choices. Knowledge of where *Dunyazad*'s reasoning breaks down is in fact just as interesting as knowledge of where it succeeds, and is the main benefit of the use of AI in this project: without a technical implementation of the theory of choice poetics, these hidden assumptions would have gone unnoticed.

Viewed in this light, I'm happy with the results of the project so far. It has broken new theoretical and technical ground, backed up by rigorous experimental results. The following sections summarize the key results of the *Dunyazad* project and suggest important directions for future work.

9.1 A Theory of Choice Poetics

Chapter 5 describes both a broad foundation for understanding the poetics of choices and a detailed procedure for analyzing choices as they related to player goals. It provides language for talking about explicit, discrete choices in terms of *framing*, *options*, and *outcomes*, and it introduces the notion of modes of engagement, which help conceptualize players' sometimes complex motivations. When discussing poetic effects, it goes beyond well-established aspects of player experience like agency and touches on effects like autonomy, responsibility, and regret. Finally, chapter 5 describes in detail a technique for analyzing choices based on player goals; this is the main theoretical contribution of this chapter.

By mechanically breaking down a player's experience of a choice into formal components such as 'player goals' and 'outcome evaluations,' goal-based choice analysis seeks to expose hidden connections and account for a range of possible player motives. While a naïve evaluation of a choice might be able to come up with the most common reaction and predict which option most players would choose, the detailed dissection involved in goal-based choice analysis helps understand how alternative motives might color some player's perceptions and exposes the perceived trade-offs between options. For authors of branching narratives, goal-based choice analysis should be a useful tool for viewing a choice from multiple player perspectives, and it may be helpful when "debugging" a choice that playtesting has identified as provoking unexpected reactions. For critics, goal-based choice analysis ought to help go from a gut reaction to an explicit description of why a particular choice is effective or ineffective at eliciting a particular response, as demonstrated in section 5.4.10.

The theory developed in chapter 5 is based on existing theories of traditional and interactive poetics along with craft advice, as described in chapter 3. But what sets it apart from similar theories is that it was developed in tandem with a generative system. It is intentionally more mechanical to allow for easy operationalization, and this also makes it easy to incorporate feedback from experimental results.

9.2 A Choice-Point Generator

The main technical result of the *Dunyazad* project is a program that generates narrative choices. While any interactive narrative project necessarily gives the player some kind of choices, until now there have only been a few projects that focused on those choices as a first-class design problem: most projects generate (or manage) some other structure which implicitly contains choices without ever

focusing on the set of options available to the player at any one moment. As the results in chapters 7 and 8 demonstrate, however, the set of options available can profoundly impact player reactions, and even options which are never chosen by the player color their perception of the outcomes that they do see. While a few other projects such as (Barber and Kudenko, 2007b; Yu and M. O. Riedl, 2013) have made explicit choices and their structures the focus of their attention, these projects have not attempted to generate a variety of choices in a principled manner. Relative to existing work, then, *Dunyazad* stands out as a project that aims to be able to generate a range of different choice structures by using a general theory of choice poetics.

Dunyazad's reliance on an operationalization of choice poetics is key here. The same results in terms of measurable audience impact could likely have been achieved using a variety of approaches. Certainly human authors could have been employed to write choices given similar criteria (e.g., "make it have an obvious best option"). What is the advantage of using a computer for this task (or of using answer-set programming as opposed to some other technique)? The answer lies in the other half of the hybrid research approach: all of these alternative techniques, including paying human authors, could have achieved equivalent audience impacts, but none of them offer the same level of insight into the mechanisms behind that impact.

Because *Dunyazad* uses answer set programming, both its successes and failures can be reliably understood in terms of the logical rules of implication that it operates over, and these rules directly correspond to tenets of the choice poetic theory that it was developed in dialogue with. In other words, *Dunyazad* can establish a sufficiency relationship between the statements of choice poetic theory and the effects that they purport to describe. By generating choices which audiences perceive as, say, "obvious," *using only the rules of choice poetics to guide it*, *Dunyazad* is a procedural proof that those rules are *sufficient* to give

rise to "obviousness," at least in the story contexts that *Dunyazad* operates in. Likewise, when a choice is labelled "obvious" but humans don't perceive it as such, *Dunyazad* has demonstrated that its rules are insufficient as an explanation of what "obviousness" is, and often, the human perceptions that do result can explain *why* this is the case in terms, again, of the theoretical rules of choice poetics. Few other approaches to generating choices, computational or otherwise, have the same potential to generate experimental results which can directly inform the underlying theory. *Dunyazad* is thus not only an example of a successful system that explores a novel generative space, it is also a unique tool for the study of choice poetics, and thus a demonstration of a new research approach that employs artificial intelligence as a tool for the development of a humanistic theory.

9.3 Experimental Results

The experimental results presented in chapters 7 and 8 are the linchpin of the hybrid research approach that drives *Dunyazad*. These results establish, via *Dunyazad*'s rules, a direct link between actual human reactions to choices and the theory of choice poetics which attempts to predict those reactions. Tables 8.20 and 8.21 at the very end of chapter 8 show the results of this feedback: caveats and amendments to the analysis method described in chapter 5 that resulted from an analysis of experimental data.

With the participation of hundreds of paid subjects contacted via Amazon Mechanical Turk, the options and outcomes that *Dunyazad* generates were evaluated in two experiments, focused on immediate reactions such as obviousness or the valence of outcomes. *Dunyazad* generated choices for these experiments by following the implications of simple constraints, such as "make this choice obvious" within the body of choice poetics theory that it is programmed with.

Because of this, unexpected results could be traced back to the rules that allowed them. Critically, *Dunyazad*'s inhuman exploration of the possibilities allowed by its rules discovered edge cases that are hidden by human assumptions, such as the inability of straightforward goal analysis to account for certain kinds of moral reasoning. These edge cases would probably not have been discovered by a human attempt to work within the same constraints, because a human instructed to "create an obvious choice" would unconsciously rule out structures inconsistent with moral reasoning without being told to do so.

While the experiments did reveal that *Dunyazad* needs more work to consistently produce the desired low-level poetics, they also confirmed that it was largely successful at generating specific types of choices. The flaws that the results exposed in *Dunyazad*'s reasoning have also served to directly inform the theory of choice poetics from chapter 5. For example, *Dunyazad*'s difficulty in reasoning about moral conflicts of interest (between e.g., self-preservation and helping an innocent victim in need) shows that when humans apply the goal-based choice analysis method presented in section 5.4 they should be careful to exercise their capacity for moral reasoning. This result, and the others like it presented in chapters 7 and 8, are the real fruits of the hybrid research method employed in *Dunyazad*. By using *Dunyazad*'s inhuman reasoning to double-check the human reasoning behind the theory of choice poetics, that theory can be improved and developed more thoroughly than by using human reasoning alone.

9.4 A Hybrid Research Process

The hybrid research approach described here is not just applicable to choice poetics. I hope that the success of *Dunyazad* as a project can inspire other researchers to find value in the application of artificial intelligence as a tool for research in the humanities and social sciences. Beyond looking for patterns

in large amounts of data or running intricate simulations designed to model real-world phenomenon, computers can use symbolic reasoning to explore logical theories along inhuman lines of inquiry. As frustrating as this can be at times, the laborious process of explicitly encoding human assumptions about how things work tends to reveal hidden biases and assumptions that were taken-for-granted before a computer without them attempted to put a theory to use. By putting a computer's output in front of other humans for judgement, a feedback loop can be formed, allowing the computer's "mistakes" to expose the underlying assumptions of a theory's author, thus informing the theory and the next batch of generated results.

9.5 Future Work

As a generative system, despite not being able to produce entire gamebooks at a command, *Dunyazad* can produce interesting choices, and it has a big enough generative space to occasionally surprise authors without disappointing them (although some of its surprises, like those of most generative systems, are disappointing). Perhaps more importantly, the choices that it generates are complex and consistent enough to reveal edge-cases of the theory that drives it, and not just question its implementation of that theory. Given these successes, there are multiple directions that the project could go in.

First, the flaws revealed by the experimental data should be addressed. This involves implementing relative value judgements as well as moral reasoning (perhaps along the lines of the defeasible reasoning approach presented by Joseph Blass and Ian Horswill in (Blass and Horswill, 2015)). *Dunyazad* will also need to implement a more complicated approach to goal priorities, and this will have to be backed up by an understanding of how players approach conflicting goals: the theory of choice poetics will need to become more specific on this point. As

with the functionality that *Dunyazad* already has, once the theory and system have evolved, they should be tested in an experimental study, and further revised based on the results.

Besides developing *Dunyazad*'s core reasoning capabilities, it should also be extended to reason about larger plot structures. Currently, *Dunyazad* can chain together choices to play out a single scene and transition from scene to scene by moving the action to a new location, but it lacks high-level reasoning about plot and elements like recurring characters between scenes that would enable it to generate longer stories. Implementing these things could enable the study of more complex poetic effects like regret, although of course these effects also provide challenges in terms of experimental design. Having *Dunyazad* generate full stories would also open up new applications: at that point, it could be studied as a full interactive narrative experience in its own right.

Finally, the theory of choice poetics should be put to use to gauge its effectiveness in analyzing human-constructed narrative choices. Not only should it enable more detailed and explicit analysis of works like Choose-Your-Own-Adventure books and visual novels, but the act of applying it will result in a different kind of feedback that can drive the development of the theory. This in turn could expose new avenues for the growth of *Dunyazad* as a generative system. Ultimately, both the system and the theory should benefit from the theory's application.

All of this work fits into a broader research agenda of developing generative AI systems that can both push the boundaries of interactive media and simultaneously become sites of study for understanding it. As digital media continue to impact our society, from Facebook's filtering algorithms to the online virtual environments that have become important spaces of socialization and growth for teens and young adults, I hope to see more projects like *Dunyazad* that use computers not just to create these environments but to help us make sense of them and understand how they impact the humans that use them.

Bibliography

Aarseth, Espen J (1997). *Cybertext: Perspectives on Ergodic Literature*. JHU Press.

Agre, Philip (1997). "Toward a Critical Technical Practice: Lessons Learned in Trying to Reform AI". In: *Bridging the Great Divide: Social Science, Technical Systems, and Cooperative Work, Mahwah, NJ: Erlbaum*, pp. 131–157.

Aristotle (1917). *The Poetics of Aristotle*. Trans. by S. H. Butcher. London: Macmillan.

Bae, Byung-Chull and R Michael Young (2008). "A Use of Flashback and Foreshadowing for Surprise Arousal in Narrative Using a Plan-Based Approach". In: *Interactive Storytelling*. Springer, pp. 156–167.

Barber, Heather and Daniel Kudenko (2007a). "A User Model for the Generation of Dilemma-Based Interactive Narratives". In: *Workshop on Optimizing Player Satisfaction at AIIDE*.

— (2007b). "Dynamic Generation of Dilemma-based Interactive Narratives". In: *3rd Artificial Intelligence and Interactive Digital Entertainment Conference*.

Barthes, Roland and Lionel Duisit (1975). "An Introduction to the Structural Analysis of Narrative". In: *New Literary History* 6.2, pp. 237–272.

Bartle, Richard (1996). "Hearts, Clubs, Diamonds, Spades: Players Who Suit MUDs". In: *Journal of MUD Research* 1.1, p. 19.

Bates, Joseph (1992). *The Nature of Characters in Interactive Worlds and The Oz Project*. Tech. rep. CMU-CS-92-200. School of Computer Science, Carnegie Mellon University.

Bernstein, Mark (1998). "Patterns of Hypertext". In: *9th ACM Conference on Hypertext and Hypermedia*. ACM, pp. 21–29.

Blass, Joseph and Ian Horswill (2015). "Implementing Injunctive Social Norms Using Defeasible Reasoning". In: *Papers from the AIIDE 2015 Joint Workshop on Intelligent Narrative Technologies and Social Believability in Games*. URL: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE15/paper/view/11559.

Bogost, Ian (2008). "The Rhetoric of Video Games". In: *The Ecology of Games: Connecting Youth, Games, and Learning*, pp. 117–39.

Borges, Jorge Luis (1956). "The Garden of Forking Paths". In: *Ficciones*. Buenos Aires: Emecé Editores.

Brewer, William F and Edward H Lichtenstein (1982). "Stories are to Entertain: A Structural-Affect Theory of Stories". In: *Journal of Pragmatics* 6.5, pp. 473–486.

Brown, Andrew R (2001). "Modes of Compositional Engagement". In: *Mikropolyphony* 6. URL: http://eprints.qut.edu.au/168/.

Bui, Vinh, Hussein Abbbass, and Axel Bender (2010). "Evolving Stories: Grammar Evolution for Automatic Plot Generation". In: *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, pp. 1–8.

Busselle, Rick and Helena Bilandzic (2009). "Measuring Narrative Engagement". In: *Media Psychology* 12.4, pp. 321–347.

Cardona-Rivera, Rogelio E et al. (2014). "Foreseeing Meaningful Choices". In: *10th Artificial Intelligence and Interactive Digital Entertainment Conference*.

Cheong, Yun-Gyung and R Michael Young (2006). "A Computational Model of Narrative Generation for Suspense". In: *AAAI*, pp. 1906–1907.

Choice of Games LLC (2010). “Game Design” blog category. URL: http://www.choiceofgames.com/category/blog/game-design/ (visited on 02/04/2016).

Dehn, Natalie (1981). “Memory in Story Invention”. In: *3rd Annual Conference of the Cognitive Science Society*, pp. 213–215.

Douglas, J. Yellowlees and Andrew Hargadon (2001). “The Pleasures of Immersion and Engagement: Schemas, Scripts and the Fifth Business”. In: *Digital Creativity* 12.3, pp. 153–166. DOI: 10.1076/digc.12.3.153.3231. eprint: http://www.tandfonline.com/doi/pdf/10.1076/digc.12.3.153.3231. URL: http://www.tandfonline.com/doi/abs/10.1076/digc.12.3.153.3231.

Ermi, Laura and Frans Mäyrä (2005). “Fundamental Components of the Gameplay Experience: Analysing Immersion”. In: *DiGRA &#3905 - Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play*. URL: http://www.digra.org/wp-content/uploads/digital-library/06276.41516.pdf.

Fabulich, Dan (2010). *5 Rules for Writing Interesting Choices in Multiple Choice Games*. URL: http://www.choiceofgames.com/2010/03/5-rules-for-writing-interesting-choices-in-multiple-choice-games/ (visited on 02/04/2016).

Feilitzen, Cecilia von and Olga Linné (1975). “Identifying With Television Characters”. In: *Journal of Communication* 25.4, pp. 51–55. ISSN: 1460-2466. DOI: 10.1111/j.1460-2466.1975.tb00638.x. URL: http://dx.doi.org/10.1111/j.1460-2466.1975.tb00638.x.

Fendt, Matthew William et al. (2012). “Achieving the Illusion of Agency”. In: *Interactive Storytelling*. Springer, pp. 114–125.

Flanagan, Mary (2009). *Critical Play: Radical Game Design*. Cambridge, MA: The MIT Press.

Forster, Edward M (1974). *Aspects of the Novel*. Vol. 12. Stallybrass, Oliver, Ed. 25 Hill Street, London W1X 8LL: Edward Arnold Ltd. ISBN: 0713157690.

Freytag, Gustav (1894). *Technique of the Drama*. Trans. by Elias J. MacEwan. Chicago, IL: S.C. Griggs and Company.

Frome, Jonathan (2006). "Representation, Reality, and Emotions Across Media". In: *Film Studies* 8.1, pp. 12–25.

Gebser, Martin et al. (2011). "Potassco: The Potsdam Answer Set Solving Collection". In: *AI Comm.* 24.2, pp. 107–124.

Gerrig, Richard J. and Allan B.I. Bernardo (1994). "Readers as Problem-Solvers in the Experience of Suspense". In: *Poetics* 22.6, pp. 459–472. ISSN: 0304-422X. DOI: http://dx.doi.org/10.1016/0304-422X(94)90021-3. URL: http://www.sciencedirect.com/science/article/pii/0304422X94900213.

Gervás, Pablo et al. (2005). "Story Plot Generation Based on CBR". In: *Knowledge-Based Systems* 18.4-5, pp. 235–242. ISSN: 09507051. DOI: 10.1016/j.knosys.2004.10.011. URL: http://linkinghub.elsevier.com/retrieve/pii/S0950705105000407.

Graesser, Arthur C, Murray Singer, and Tom Trabasso (1994). "Constructing Inferences During Narrative Text Comprehension". In: *Psychological Review* 101.3, p. 371.

Green, Melanie C and Timothy C Brock (2000). "The Role of Transportation in the Persuasiveness of Public Narratives". In: *Journal of Personality and Social Psychology* 79.5, p. 701.

Green, Melanie C., Timothy C. Brock, and Geoff F. Kaufman (2004). "Understanding Media Enjoyment: The Role of Transportation Into Narrative Worlds". In: *Communication Theory* 14.4, pp. 311–327. ISSN: 1468-2885. DOI: 10.1111/j.1468-2885.2004.tb00317.x. URL: http://dx.doi.org/10.1111/j.1468-2885.2004.tb00317.x.

Greimas, Algirdas Julien (1988). *Maupassant: The Semiotics of Text: Practical Exercises*. Vol. 1. John Benjamins Publishing.

Haase Jr., Kenneth W. (1986). *Discovery Systems*. Tech. rep. AIM-898. Massachusetts Institute of Technology. URL: http://18.7.29.232/handle/1721.1/6450.

Hall, L., P. Johansson, and T. Strandberg (2012). "Lifting the Veil of Morality: Choice Blindness and Attitude Reversals on a Self-Transforming Survey". In: *PLoS ONE* 7.9, e45457. DOI: 10.1371/journal.pone.0045457.

Hamari, Juho and Janne Tuunanen (2014). "Player Types: A Meta-Synthesis". In: *Transactions of the Digital Games Research Association* 1.2.

Horace (1783). *The Art of Poetry: an Epistle to the Pisos*. Trans. by G. Colman. The Strand, London: T. Cadell. URL: http://www.gutenberg.org/ebooks/9175 (visited on 01/29/2016). Project Gutenberg, online, January 29, 2016 (EBook 9175).

Huizinga, Johan (1949). *Homo Ludens*. Vol. 3. Taylor & Francis.

id Software (1996). *Quake*. GT Interactive. PC; various.

Infocom (1980). *Zork*. Personal Software; Infocom. Various platforms.

Iran-Nejad, Asghar (1987). "Cognitive and Affective Causes of Interest and Liking". In: *Journal of Educational Psychology* 79.2, p. 120.

Jackson, Shelley (1995). *Patchwork Girl*. Watertown, MA: Eastgate Systems.

Joyce, Michael (1990). *Afternoon, a Story*. Watertown, MA: Eastgate Systems.

Kallio, Kirsi Pauliina, Frans Mäyrä, and Kirsikka Kaipainen (2011). "At Least Nine Ways to Play: Approaching Gamer Mentalities". In: *Games and Culture* 6.4, pp. 327–353. DOI: 10.1177/1555412010391089. eprint: http://gac.sagepub.com/content/6/4/327.full.pdf+html. URL: http://gac.sagepub.com/content/6/4/327.abstract.

Kintsch, Walter (1980). "Learning From Text, Levels of Comprehension, or: Why Anyone Would Read a Story Anyway". In: *Poetics* 9.1–3. Special Issue on Story Comprehension, pp. 87–98. ISSN: 0304-422X. DOI: http://dx.doi.org/10.1016/0304-422X(80)90013-3. URL: http://www.sciencedirect.com/science/article/pii/0304422X80900133.

Klein, S., J. F. Aeschiliman, et al. (1973). *Automatic Novel Writing: A Status Report*. Tech. rep. TR186. Computer Sciences Department, The University of Wisconsin–Madison.

Klein, S., J. D. Oakley, et al. (1971). *A Program for Generating Reports on the Status and History of Stochastically Modifiable Semantic Models of Arbitrary Universes*. Tech. rep. TR142. Computer Sciences Department, The University of Wisconsin–Madison.

Klimmt, Christoph, Dorothée Hefner, and Peter Vorderer (2009). "The Video Game Experience as "True" Identification: A Theory of Enjoyable Alterations of Players' Self-Perception". In: *Communication Theory* 19.4, pp. 351–373. ISSN: 1468-2885. DOI: 10.1111/j.1468-2885.2009.01347.x. URL: http://dx.doi.org/10.1111/j.1468-2885.2009.01347.x.

Lakatos, Imre (1971). "Proceedings of the 1970 Biennial Meeting of the Philosophy of Science Association". In: ed. by Roger C. Buck and Robert S. Cohen. Dordrecht, Netherlands: Springer. Chap. History of Science and its Rational Reconstructions, pp. 91–136. ISBN: 9789401031424. DOI: 10.1007/978-94-010-3142-4_7. URL: http://dx.doi.org/10.1007/978-94-010-3142-4_7.

Langer, Judith A (1995). *Envisioning Literature: Literary Understanding and Literature Instruction*. ERIC.

Laurel, Brenda (1986). "Toward the Design of a Computer-Based Interactive Fantasy System". PhD thesis. Ohio State University.

Laws, Robin (2001). *Robin's Laws of Good Game Mastering*. Steve Jackson Games.

Lebowitz, Michael (1983). *Creating a Story-Telling Universe*. Tech. rep. CUCS-055-83. Columbia University.

— (1984). "Creating Characters in a Story-Telling Universe". In: *Poetics* 13.3, pp. 171–194. ISSN: 0304-422X. URL: http://linkinghub.elsevier.com/retrieve/pii/0304422X84900019.

— (1985). "Story-Telling as Planning and Learning". In: *Poetics* 14.6, pp. 483–502.

Li, Boyang et al. (2013). "Story Generation with Crowdsourced Plot Graphs". In: *AAAI Conference on Artificial Intelligence*. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6399.

Mann, H. B. and D. R. Whitney (1947). "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other". In: *Ann. Math. Statist.* 18.1, pp. 50–60. DOI: 10.1214/aoms/1177730491. URL: http://dx.doi.org/10.1214/aoms/1177730491.

Mason, Stacey (2013). "On Games and Links: Extending the Vocabulary of Agency and Immersion in Interactive Narratives". In: *Interactive Storytelling*. Springer, pp. 25–34.

Mateas, Michael (2001). "A Preliminary Poetics for Interactive Drama and Games". In: *Digital Creativity* 12.3, pp. 140–152.

Mateas, Michael and Andrew Stern (2002). *Architecture, Authorial Idioms and Early Observations of the Interactive Drama Façade*. School of Computer Science, Carnegie Mellon University.

Mateas, Michael and Noah Wardrip-Fruin (2009). "Defining Operational Logics". In: *Digital Games Research Association (DiGRA)*.

Mawhorter, Peter, Michael Mateas, and Noah Wardrip-Fruin (2015a). "Generating Relaxed, Obvious, and Dilemma Choices with Dunyazad". In: *Proceedings*

of the 11th Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AIIDE '15, pp. 58–64.

Mawhorter, Peter, Michael Mateas, and Noah Wardrip-Fruin (2015b). "Intentionally Generating Choices in Interactive Narratives". In: *Proceedings of the 6th International Conference on Computational Creativity*. ICCC '15, pp. 292–299.

Mawhorter, Peter, Michael Mateas, Noah Wardrip-Fruin, and Arnav Jhala (2014). "Towards a Theory of Choice Poetics". In: *International Conference on the Foundations of Digital Games*. FDG '14.

Maxis (1989). *Sim City*. Maxis. Macintosh.

— (2000). *The Sims*. Electronic Arts. PC.

Meehan, James R. (1976). "The Metanovel: Writing Stories by Computer". PhD thesis. Yale University.

Mellers, Barbara A., Alan Schwartz, Katty Ho, et al. (1997). "Decision Affect Theory: Emotional Reactions to the Outcomes of Risky Options". In: *Psychological Science* 8.6, pp. 423–429. ISSN: 09567976, 14679280. URL: http://www.jstor.org/stable/40063228.

Mellers, Barbara A., Alan Schwartz, and Ilana Ritov (1999). "Emotion-Based Choice". In: *Journal of Experimental Psychology* 128.3, pp. 332–345.

Mitchell, Alexander Ian (2012). "Reading Again for the First Time: Rereading for Closure in Interactive Stories". PhD thesis. Singapore: National University of Singapore.

Morgan, Wendy (1999). "Heterotopics, Towards a Grammar of Hyperlinks". In: *4th Hypertext Writers' Workshop: The Messenger Morphs the Media*. Vol. 99, pp. 21–25. URL: http://www.wordcircuits.com/htww/morgan1.htm.

Mott, Bradford W and James C Lester (2006). "U-Director: a Decision-Theoretic Narrative Planning Architecture for Storytelling Environments". In: *AAMAS*. Vol. 6. Citeseer, pp. 977–984.

Motte, Warren F (1986). *Oulipo: A Primer of Potential Literature*. University of Nebraska Press Lincoln & London.

Murphy, John and José P. Zagal (2011). "Videogames and the Ethics of Care". In: *International Journal of Gaming and Computer-Mediated Simulations* 3.3, pp. 69–81.

Murray, Janet H. (1997). *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*. New York: Free Press.

Nelson, Mark J et al. (2006). "Declarative Optimization-Based Drama Management in Interactive Fiction". In: *Computer Graphics and Applications, IEEE* 26.3, pp. 32–41.

Newell, Allen and Herbert A. Simon (1976). "Computer Science As Empirical Inquiry: Symbols and Search". In: *Communications of the ACM* 19.3, pp. 113–126. ISSN: 0001-0782. DOI: 10.1145/360018.360022. URL: http://doi.acm.org/10.1145/360018.360022.

Oatley, Keith (1995). "A Taxonomy of the Emotions of Literary Response and a Theory of Identification in Fictional Narrative". In: *Poetics* 23.1, pp. 53–74.

— (1999). "Why Fiction May be Twice as True as Fact: Fiction as Cognitive and Emotional Simulation". In: *Review of General Psychology* 3.2, p. 101.

Olson, Cheryl K., Lawrence A. Kutner, and Dorothy E. Warner (2008). "The Role of Violent Video Game Content in Adolescent Development: Boys' Perspectives". In: *Journal of Adolescent Research* 23.1, pp. 55–75. DOI: 10.1177/0743558407310713. eprint: http://jar.sagepub.com/content/23/1/55.full.pdf+html. URL: http://jar.sagepub.com/content/23/1/55.abstract.

Palmer, Alan (2004). *Fictional Minds*. University of Nebraska Press.

Patridge, Derek and Yorick Wilks, eds. (1990). *The Foundations of Artificial Intelligence: A Sourcebook*. Newcastle upon Tyne, UK: Cambridge University Press, pp. 237–246. ISBN: 9780521359443.

Peinado, Federico and Pablo Gervás (2006). "Proceedings of the Third International Conference on Technologies for Interactive Digital Storytelling and Entertainment". In: ed. by Stefan Göbel, Rainer Malkewitz, and Ido Iurgel. Berlin, Heidelberg: Springer. Chap. Minstrel Reloaded: From the Magic of Lisp to the Formal Semantics of OWL, pp. 93–97. ISBN: 9783540499350. DOI: 10.1007/11944577_9. URL: http://dx.doi.org/10.1007/11944577_9.

Pérez y Pérez, Rafael (1999). "MEXICA: A Computer Model of Creativity in Writing". PhD thesis. Brighton, UK: School of Cognitive and Computing Sciences, The University of Sussex.

Pérez y Pérez, Rafael and Mike Sharples (2001). "MEXICA: A Computer Model of a Cognitive Account of Creative Writing". In: *Journal of Experimental and Theoretical Artificial Intelligence* 13, pp. 119–139.

Peterson, Jon (2012). *Playing at the World*. San Diego, CA: Unreason Press. ISBN: 9780615642048.

Pope, Lucas (2013). *Papers Please*. 3909 LLC. Various platforms.

Propp, Vladimir (1971). *Morphology of the Folktale*. Ed. by Louis A. Wagner. Trans. by Laurence Scott. 2nd Revised edition. Austin, TX, USA: University of Texas Press.

Queneau, Raymond (1967). "A Story as You Like It". In: *(Motte, 1986) (translated)*.

Riedl, Mark (2004). "Narrative Planning: Balancing Plot and Character". PhD thesis. Raleigh, NC: Dept. of Computer Science, North Carolina State University.

Ritchie, Graeme (1984). "A Rational Reconstruction of the Proteus Sentence Planner". In: *Proceedings of the 10th International Conference on Computational Linguistics*. COLING '84. Stanford, California: Association for Computational Linguistics, pp. 327–329. DOI: 10.3115/980431.980557. URL: http://dx.doi.org/10.3115/980431.980557.

Robbins, Naomi B and Richard M Heiberger (2011). "Plotting Likert and Other Rating Scales". In: *Proceedings of the 2011 Joint Statistical Meeting*.

Ryan, Marie-Laure (1991). *Possible Worlds, Artificial Intelligence, and Narrative Theory*. Indiana University Press.

Ryan, Richard M., C. Scott Rigby, and Andrew Przybylski (2006). "The Motivational Pull of Video Games: A Self-Determination Theory Approach". In: *Motivation and Emotion* 30.4, pp. 344–360. ISSN: 1573-6644. DOI: 10.1007/s11031-006-9051-8. URL: http://dx.doi.org/10.1007/s11031-006-9051-8.

Schwartz, Barry et al. (2002). "Maximizing Versus Satisficing: Happiness is a Matter of Choice". In: *Journal of Personality and Social Psychology* 83.5, p. 1178.

Sharma, Manu et al. (2010). "Drama Management and Player Modeling for Interactive Fiction Games". In: *Computational Intelligence* 26.2, pp. 183–211.

Shepperd, James A and James K McNulty (2002). "The Affective Consequences of Expected and Unexpected Outcomes". In: *Psychological Science* 13.1, pp. 85–88.

Simon, Herbert A (2006). "Computational Models: Why Build Them?" In: *Encyclopedia of Cognitive Science*. John Wiley & Sons, Ltd. ISBN: 9780470018866. DOI: 10.1002/0470018860.s00017. URL: http://dx.doi.org/10.1002/0470018860.s00017.

Smith, Sean and Joseph Bates (1989). *Towards a Theory of Narrative for Interactive Fiction*. Tech. rep. CMU-CS-89-121. Carnegie Mellon University.

Square (1987). *Final Fantasy*. Square. Nintendo Entertainment System.

Szilas, Nicolas (2003). "IDtension: A Narrative Engine for Interactive Drama". In: *Technologies for Interactive Digital Storytelling and Entertainment Conference*. Vol. 3, pp. 187–203.

Szilas, Nicolas (2007). “A Computational Model of an Intelligent Narrator for Interactive Narratives”. In: *Applied Artificial Intelligence* 21.8, pp. 753–801.

Tearse, Brandon et al. (2011). “Experimental Results from a Rational Reconstruction of Minstrel”. In: *International Conference on Computational Creativity*, pp. 54–59.

— (2012). “Lessons Learned from a Rational Reconstruction of Minstrel”. In: *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

— (2014). “Skald: Minstrel Reconstructed”. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 6.2, pp. 1–10. DOI: 10.1109/TCIAIG.2013.2292313.

Telltale Games (2012). *The Walking Dead: Season One*. Telltale Games. Various platforms.

Thomson, Judith Jarvis (1976). “Ethical Theory”. In: ed. by Russ Shafer-Landau. Second Edition. Chichester, West Sussex, UK: Wiley-Blackwell. Chap. Killing, Letting Die, and the Trolley Problem, pp. 543–551.

Thue, David, Vadim Bulitko, and Marcia Spetch (2008). “PaSSAGE: A Demonstration of Player Modeling in Interactive Storytelling”. In: *4th Artificial Intelligence and Interactive Digital Entertainment Conference*.

Tosca, Susana Pajares (1999). “The Lyrical Quality of Links”. In: *10th ACM Conference on Hypertext and Hypermedia*. ACM, pp. 217–218.

— (2000). “A Pragmatics of Links”. In: *11th ACM Conference on Hypertext and Hypermedia*. ACM, pp. 77–84.

Treanor, Mike (2013). “Investigating Procedural Expression and Interpretation in Videogames”. PhD thesis. Santa Cruz, CA, USA: Dept. of Computer Science, The University of California Santa Cruz.

Turner, S.R. (1993). “MINSTREL: A Computer Model of Creativity and Storytelling”. PhD thesis. University of California, Los Angeles. URL: http://portal.acm.org/citation.cfm?id=166478.

Tversky, Amos and Daniel Kahneman (1981). “The Framing of Decisions and the Psychology of Choice”. In: *Science* 211.4481, pp. 453–458.

Tversky, Amos and Itamar Simonson (1993). “Context-Dependent Preferences”. English. In: *Management Science* 39.10, pp. 1179–1189. ISSN: 00251909. URL: http://www.jstor.org/stable/2632953.

Valve Corporation (2007). *Portal*. Valve Corporation. PC.

Wardrip-Fruin, Noah (2005). “Playable Media and Textual Instruments”. In: *Dichtung Digital* 1.

Wardrip-Fruin, Noah et al. (2009). “Agency Reconsidered”. In: *Breaking New Ground: Innovation in Games, Play, Practice and Theory. Proceedings of DiGRA 2009*.

Ware, Stephen G et al. (2014). “A Computational Model of Plan-Based Narrative Conflict at the Fabula Level”. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 6.3, pp. 271–288.

Weyhrauch, Peter (1997). “Guiding Interactive Fiction”. PhD thesis. Carnegie Mellon University.

Wilcoxon, Frank (1945). “Individual Comparisons by Ranking Methods”. English. In: *Biometrics Bulletin* 1.6, pp. 80–83. ISSN: 00994987. URL: http://www.jstor.org/stable/3001968.

Witmer, Bob G. and Michael J. Singer (1998). “Measuring Presence in Virtual Environments: A Presence Questionnaire”. In: *Presence: Teleoperators and Virtual Environments* 7.3, pp. 225–240. DOI: 10.1162/105474698565686. URL: http://dx.doi.org/10.1162/105474698565686.

Yee, Nick (2006). “Motivations for Play in Online Games”. In: *CyberPsychology & Behavior* 9.6, pp. 772–775.

Young, R. Michael (1999). “Notes on the Use of Plan Structures in the Creation of Interactive Plot”. In: *AAAI Fall Symposium on Narrative Intelligence*. AAAI Technical Report FS-99-01, pp. 164–167.

Yu, Hong and Mark O. Riedl (2013). “Data-Driven Personalized Drama Management”. In: *9th Artificial Intelligence and Interactive Digital Entertainment Conference*.

Zagal, José P. (2009). “Ethically Notable Videogames: Moral Dilemmas and Gameplay”. In: *DiGRA ’09 - Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory*. Brunel University. URL: http://www.digra.org/wp-content/uploads/digital-library/09287.13336.pdf.

Zhu, Jichen and Santiago Ontañón (2010). “Towards Analogy-Based Story Generation”. In: *1st International Conference on Computational Creativity (ICCC-X)*, pp. 75–84.

Zunshine, Lisa (2006). *Why We Read Fiction: Theory of Mind and the Novel*. Ohio State University Press.

Zwaan, Rolf A, Mark C Langston, and Arthur C Graesser (1995). “The Construction of Situation Models in Narrative Comprehension: An Event-Indexing Model”. In: *Psychological Science*, pp. 292–297.

Appendices

Appendix A

English Generation

Once *Dunyazad*'s iterative story generation has finished filling in the timepoints of a story with predicates that represent states, events, and choices, all of that information needs to be made playable. *Dunyazad* uses a template-based natural-language generation system to turn its predicate statements representing story actions into text (and choice interface elements). The obvious format is hypertext, but some practical problems and even some real generative work are entailed by the transformation from predicates to text. *Dunyazad* doesn't yet do anything groundbreaking in terms of discourse, but the experiments that it has been used in do depend on the text generation capabilities described here.

Dunyazad uses simple grammar-based text generation with small pre-defined grammars for each setup, action and potential (special world states that motivate actions). Beyond this it knows how to describe the initial state of the world, and has pre-defined introduction and epilogue grammars. In the expansion of these grammars to create text, it can substitute variables but also knows some simple conjugation rules to enforce subject-verb agreement and it does automatic pronominalization to help the text flow together as a whole.

A.1 Grammar Expansion

The basic unit of discourse content in *Dunyazad* is a grammar expansion, which looks like this:

```
:action/attack/option
N#?aggressor/they V#attack/prs/?aggressor N#?target/them
```

. . . and might be rendered into this:

"you attack the bandits"

. . . or this:

"the merchant attacks you"

The first line, which begins with a ':' character, indicates a substitution key (a non-terminal in the grammar), and another grammar rule could include an expansion using this key, which would look like this:

```
[[action/attack/option]]
```

Although not required, path-like key names like this example help organize the grammar and can be matched over using some wildcards (see below). After a key definition, one or more expansions for that key are given on the following lines, and when an expansion is made, one of those will be selected at random. Besides the key of the desired expansion, a grammar expansion can optionally contain flags and variable assignments, like this:

```
[[S|misc/you_ask_for@statement=[[action/?_action/option]]]]
```

Flags are capital letters given at the beginning of the expansion followed by a '|' character. The only valid flag is currently 'S' for 'sentence' which after expansion capitalizes the first letter of the result and adds a period to the end. Variable assignments are given by an '@' symbol followed by "<var>=<value>." Any number of variable assignments may be given and, as shown, both variables and grammar expansions can be nested in each other. In this example, the key "misc/you_ask_for" will be looked up and a random expansion for it will be chosen. Within text generated by that expansion and its children (if any), the variable "statement" will take on the literal value "[[action/?_action/option]]." If the expansion text includes a reference to that variable, of course, when the variable is substituted a further grammar expansion will be present, but the exact key to be looked up will depend on the value of the variable "_action" at the time of the substitution of the "statement" variable. (More precisely, when the string "?statementi"s processed as a variable within text that results from this example grammar expansion, the result will include the string "?_actionw"hich will be subsequently processed as a variable before any further grammar expansion.)

A.2 Variable Expansion

As already indicated, variable substitutions happen concurrently with grammar substitutions. A variable substitution is indicated by a question mark followed by a variable name, these must be valid Python identifiers (generally lowercase strings that may include underscores). Different variables are supplied by *Dunyazad* in different contexts, and grammar expansions may also manually assign variables as shown above.

The key in the first example above is "action/attack/option," which is used when *Dunyazad* needs to display option text (the text that the player will click on to trigger the action that follows the standard "What do you do?" prompt)

for an "attack" action. In this context, *Dunyazad* runs the grammar with all of the arguments of the current option's action bound as variables, so because the "attack" action has "aggressor" and "target" arguments, those variables names can be used. Other default variables like "_action" which identifies the current action are supplied, in the second example above this is used to switch between grammar keys in an expansion. Variables for actors and items expand to the unique identifier for the actor/item in question, so "?target" might expand to "bandits_17" during text generation.

Variables and grammar expansions may both contain each other because their expansion is interleaved. When expanding a grammar element, the first step is to scan for all variable substitutions and replace them with their values. This step uses a while loop, so variables whose values contain variable expansions will keep expanding until there are no expansions left (setting a variable's value to an expansion for itself would cause an infinite loop). Once there are (apparently) no more variables to substitute, the text is parsed into a list of static text elements separated by grammar expansions. In text order, the grammar expansions are then recursively expanded one-by-one (making for a depth-first expansion policy, although the grammar is context-free, so this matters little). The first step in each of these recursive calls, of course, is the exhaustive variable substitution mentioned above.

The net result is that whenever a variable is substituted which contains a grammar expansion, that expansion will be considered afterwards, and when a grammar expansion contains a variable, it will be substituted before any sub-expansions occur. The exhaustive variable substitution rule means that if a variable contains a grammar expansion with a variable key (as in the example above), the key variable will be substituted with its value before the grammar expansion is even parsed. Nested grammar expansions, on the other hand, are not allowed (in part simply to reduce parser complexity); this is one categorical

difference between grammar expansions and variables. Another is that all grammar key-expansion relationships are assigned at compile time (technically, when the English generation system starts up), while variable assignments can change at runtime.

A.3 Tags

The extra syntax that surrounds the variable expansions in the "`action/attack/option`" example above is *Dunyazad*'s approach to managing subject/verb agreement and pronominalization. These are called 'tags,' and they indicate a third, more dynamic type of substitution. Whereas variable substitution and grammar expansion happen concurrently, tag expansions happen only after the other two types are exhausted, and are scoped in terms of linear text rather than the tree of grammar expansions. Tags allow *Dunyazad* to look up all of the properties of an entity (an actor or item) and use this information to get the text right. Their two main purposes are dynamically substituting noun references which may be pronouns, and conjugating verbs while ensuring subject-verb agreement.

In the first example above, notice how "attack" changes to "attacks" to agree with the singular verb "the merchant" in the second example result. Because the verb tag ("`V#`") contains a reference to the subject of the sentence ("`?aggressor`"), it knows whether the noun it must agree with is singular or plural. Of course, forcing the author to manually specify which noun each verb agrees with is a significant burden, but integrating automatic analysis that could determine this correctly most of the time proved beyond the scope of the project (there are existing solutions to this problem). Besides noun agreement, verb tags are also used for conjugation, and thus specify a tense as well as an agreeing entity ("`prs`" in this case to indicate present-tense).

Noun tags ("`N#`") meanwhile include both an entity reference and a third person plural pronoun. These pronouns are used as a shorthand to identify the

case and position of noun use, because the third-person plural pronouns are unique for each case/position ('they,' 'them,' 'their,' 'theirs,' 'themselves'). The rest of the information necessary to figure out an appropriate pronoun (person, number, and gender) is gleaned from the entity reference. All of this information can be used to refer to an entity like a normal English speaker would instead of always using its full proper name.

Because *Dunyazad* often needs to assemble paragraphs from sentences that are written independently from one another, it's impossible for the author to specify which noun instances should be full references or pronouns at the discourse level. As an example of this, consider a paragraph that uses the same nouns multiple times with just a single form of address:

> As you travel onwards, you come across the bandits threatening the merchant. You defeat the bandits, and the bandits are injured. The bandits are no longer threatening the merchant.

The same paragraph using proper pronominalization and introduction flows much better, even if the sentences are still a bit choppy:

> As you travel onwards, you come across some bandits threatening a merchant. You defeat the bandits, and they are injured. They are no longer threatening the merchant.

Especially if each sentence is authored in a different file and the final sentence comes from a grammar key that is used in multiple different contexts, the author can't determine when writing the underlying grammar fragments which nouns should be indefinite or pronominalized. The solution to this problem is to let *Dunyazad* handle introduction and pronominalization and just have the author indicate what pronoun would be used in each situation if one is necessary. *Dunyazad* detects the introduction of new entities, inserting "a merchant" instead

of “the merchant” the first time a specific merchant is encountered, and it uses a table of recently-used nouns to insert pronouns when they are unambiguous.

The system for pronominalization is fairly simple: it keeps track of the last noun used which would be a referent for three different pronoun ‘slots:’ ‘he/she,’ ‘it,’ and ‘they.’ When a noun is used, if it matches the latest referent for its slot, a pronoun is used instead; otherwise, the information in that slot is marked as confounded. After an adjustable number of nouns pass without renewal, old slots are cleared so that new information can be stored. Only when a slot is empty is new information stored upon encountering a noun that fits it. These rules are an attempt to avoid confusing situations where, for example, the word ‘it’ can potentially refer to two different objects. To complement this system, the slot values are forcibly cleared at certain points (such as between setup text and action text) to avoid using too many pronouns (note the occurrence of ‘the bandits’ in the pronominalized text above instead of ‘them,’ which would sound a bit weird).

Besides verb and noun tags, the third and final type of tag is the directive tag (“`D#`”), which is used for the “`timeshift`” directive. Tense specifications included in verb tags make it possible in simple cases to automatically alter the tense of a statement, as seen in one expansion for the key “`misc/you_ask_for`”:

```
N#you/they V#ask/prs/you
  D#timeshift/infinitive@@?statement@@D#timeshift/pop@@
```

Timeshift is the only valid directive, and it takes one argument: a tense to shift into, or the special value ‘pop,’ which returns to the previous tense (tense shifts are maintained in a stack, with the top shift overriding all below it). In this example, it is used to take option text that would be given as the action of a secondary protagonist and turn it into a request from the main character for

the secondary character to perform that action (in order to maintain second-person focalization). Recall that the "`statement`" variable will have a value which looks like "`[[actions/attack/option]]`" because of the forced assignment in the "`misc/you_ask_for`" expansion shown above (assuming that "`_action`" has the value "`attack`"). In this case, which is chosen by the Python code when an option represents the action of a protagonist other than the main character, the "`aggressor`" variable will hold the identifier of the acting character. If the acting character is named Peter, the innermost grammar substitution would therefore normally produce text along the lines of:

Peter attacks the bandits.

However, that text has been placed between the two "`timeshift`" directives shown above, and so it will be shifted from the present tense (as specified in the original verb tag) into the infinitive tense, resulting in:

Peter to attack the bandits.

Of course, the outer noun and verb tags will accompany it, so the end result will be:

You ask Peter to attack the bandits.

The advantage of the "`timeshift`" directive is the ability to represent "asking-for" a task and individual task descriptions separately, as opposed to having to write every single task with both a "direct" and an "asked-for" version. Although this doesn't happen in *Dunyazad*, it could also be used to reframe narratives, for example, by having a character tell another character about events that happened to them in the past tense (the perspective change involved would also be automatic because of automatic pronominalization).

A.4 Conditional Expansions

There are three more mechanisms that *Dunyazad*'s English generator makes use of in special cases. The first are conditional expansions: by placing a line which starts with '{' and ends with '}' where an expansion would normally go, all following lines defining expansions for the current key are made conditional. Conditional expansions are only considered valid expansions for a key when the current story facts contain a set of predicates which taken together produces a match for each predicate listed on the condition line (separated by semicolons). So for example, if the following grammar rules are defined:

```
:color
{favorite_color(X);warm(X)}
a warm ?_X
{favorite_color(X);cool(X)}
a cool ?_X
```

...and the predicates representing the current story were these:

```
color(red).
color(blue).
color(yellow).
warm(red).
warm(yellow).
cool(blue).
favorite_color(blue).
```

...then expansion of the grammar "`[[color]]`" would result in the text "a cool blue." If there are multiple predicate sets that can match a condition, *Dunyazad* considers each possible match as a separate expansion possibility. Internally,

Dunyazad parses each precondition and looks for matches in the fact set, allowing uppercase names to take on any value (even a compound predicate). It proceeds to bind any uppercase names to the values required by the match(es) found and, for each match, recursively attempts to match the rest of the preconditions subject to that binding. The end result works a bit like variables in ASP constraints, and allows fairly complex preconditions, although negation has not been implemented, and only conjunction is allowed. An added bonus illustrated in this example is that any matched precondition variables appear as substitutable grammar variables during expansion of a conditional match (with an '_' prepended to their names).

To allow for meaningful preconditions, a handful of special variable names starting with '_' are provided which if used inside a precondition will only trigger a match when a specific value is found:

- "`_Now`" matches the current timepoint (whichever timepoint the text currently being expanded is associated with).
- "`_Opt`" matches the current option. In some contexts (like setups) there is no current option, and this won't match anything.
- "`_Act`" matches the name of the current action. Only valid when "`_Opt`" is.

These variables allow conditions to be expressed over things like the outcomes of the current action (which as predicates include information about the current timepoint and option). Note that preconditions may not contain normal variable substitutions, grammar expansions, or tags, because they are not evaluated in the same way that grammars are.

The main use of conditional expansions is to describe actions differently depending on their outcomes. Although *Dunyazad* does describe the consequences of actions independently, perfectly generic action descriptions leave a lot to be

desired, and being able to write different text for an action like “attack” when you know the end result will be victory or defeat is useful. On a more practical note, giving the grammar system the ability to interface directly with the ASP results cuts out the middleman so that you don’t have to route a variable from predicates through Python every time you want the text to respond more closely to the story situation.

A.5 Wildcard Keys

Another trick that *Dunyazad* has up its sleeve is wildcard keys. The grammar keys are organized into a directory-like structure with key elements separated by ‘/’ characters. When an entire path element consists of one of the ‘?’ and ‘*’ wildcard characters they take on a special meaning.

The ‘*’ wildcard is simpler: written in a grammar expansion, it means that keys must match up to that point, but afterwards may contain any path elements, which will be ignored for matching purposes. ‘*’ wildcards may only occur as the final element of a key, and may only occur in grammar expansions, not in key definitions. As an example of the ‘*’ wildcard, the expansion “`[[foo/bar/*]]`” matches key “`foo/bar/baz`,” and will thus consider expansions listed for “`foo/bar/baz`” along with any other expansions that match it when it is being expanded.

‘?’ matching is a bit more complicated, because ‘?’ is allowed to occur both in expansions and in key definitions. In expansions, the ‘?’ wildcard matches any single path element, just like the ‘*’ wildcard but a bit more limited. “`[[foo/bar/?]]`” will match “`foo/bar/baz`” but not “`foo/bar/baz/zzx`.” When ‘?’ is included in a key *definition*, however, it not only matches as a single wildcard but also binds the special variable “_” to the value that it matched. This allows “passing-in” a single argument via the “_” variable without having to make a variable assignment in

your expansion statement. In particular, this streamlines the case where one grammar expansion wants to modify a set of other expansions. For example, consider the following pair of definitions:

```
:misc/is_unskilled/?
N#?who/they V#be/prs/?who [[misc/unskilled/?_]]

:misc/unskilled/wilderness_lore
ignorant about plants and animals
[[misc/bad_at]] surviving in the wild
incapable of surviving in the wilderness
```

There are individual "misc/unskilled/<skill>" definitions for each skill, to help boost variety, although it would be easy enough to define a single generic expansion which worked for every skill. But now we want to go from "ignorant about plants and animals," to "the merchant is ignorant about plants and animals." By defining our expansion as "misc/is_unskilled/?" and then using the '_' variable, we can effectively provide a "misc/is_unskilled/<skill>" definition for each "misc/unskilled/<skill>" definition we had before. Of course, it could also be done using a variable (and in this case the variable "who" is required to specify the actor) but the wildcard key version is a bit less clunky and helps keep things mentally organized by folding the variable back into our directory tree structure where it's easier to remember.

A.6 Special Directives

Once all the grammar expansions and variable substitutions have been exhausted and the final pass has been made by substituting tags, it turns out that there are still a few loose ends to sweep up. One problem is that grammar expansions

don't know how they'll be used, and often start with a tag in any case, so proper capitalization is difficult. The 'S' flag for grammar substitutions helps a lot, but sometimes the author just needs to specify that a particular letter (which will vary because of some form of substitution) should be uppercase. This is handled by the special "`@CAP@`" directive, which removes itself and capitalizes the letter that immediately follows it.

Similarly, for formatting reasons normal text gets run together, but in some cases (like for the prologue), the author needs to be able to insert a paragraph break. The "`@PAR@`" directive does exactly this, and it even behaves appropriately depending on the output format (html vs. txt, for example). The final special directive is simply "`@@`," which actually appeared in an example above. This is used where a break is needed for parsing reasons, but no whitespace is desired in the output (likely because there's already enough whitespace around). Final processing simply removes the "`@@`" directive.

A.7 English Generation Results

Once special directives have been removed, the text is ready to be seen by humans. Along the way, Python code wraps grammar results up into larger structures depending on the desired output format. *Dunyazad* currently supports plain HTML (non-interactive output for debugging purposes), HTML-in-csv (used to define Amazon Mechanical Turk tasks for experiments), ChoiceScript (the first platform it worked with) and Twine output formats, with some additional options to control whether an entire story or just a single node is output. Figure A.1 shows an example of *Dunyazad*'s text output copy-pasted from the plain HTML output with some manual editing for formatting. Note that some of the pronominalization is imperfect: the odd option texts result when it tries to refer to items using their owner as context and the owner reference becomes a pronoun, even

though later in the same sentence it uses a definite reference. The text overall is easier to read than a version that used determinate noun references everywhere, however, and it has enough variation not to seem blatantly robotic.

Through automatic subject-verb agreement, noun introduction, and pronominalization, *Dunyazad* is able to render a wide variety of possible situations into text using only a few very generic text templates. Although there are plenty of discourse-focused systems that do much more complex things, *Dunyazad*'s simple grammar-based generator enables it to do what it needs to do without a prohibitive authoring burden. The size of its generative space relies on the fact that individual state and action predicates have descriptions which can be combined freely to describe complex situations. Of course, the quality of these descriptions is limited, but they are good enough to communicate clearly. The grammar-based system employed by *Dunyazad* also allows for the possibility of incremental refinement, as additional variety can be added by adding grammar expansions while conditional expansions allow an author to add nuanced phrases which only show up in specific situations.

You are about to set out on an epic journey. You are are heading towards the distant country of Jichish, hoping to earn fame and fortune.
You have some perfume and a book of herbal lore, and you have skill: literacy, you have skill: wilderness lore, you have skill: musician, and you have skill: sorcery.
Eager to be on your way, you set off on the road towards Jichish.
A merchant shows up and offers to trade an ancient grimoire.
The merchant is selling an oboe and he is selling his ancient grimoire.

1. *You steal his ancient grimoire from the merchant (You are missing skill: thievery. He is missing skill: thievery).*
   → You steal his ancient grimoire from the merchant, and he doesn't even notice.
2. *You offer to trade him your book of herbal lore for the merchant's oboe (no relevant skills).*
   → He agrees to trade his oboe for your book of herbal lore.
3. *You steal his oboe from the merchant (You are missing skill: thievery. He is missing skill: thievery).*
   → You steal his oboe from the merchant, and he doesn't even notice.

Figure A.1: An example of *Dunyazad*'s text output. In an interactive output format[2] the outcome text would be hidden until an option was chosen. Note the awkward pronominalization in the option text.

[2]At the time of this writing, an interactive example could be found online at `https://www.cs.hmc.edu/~pmawhorter/projects/dunyazad_example.html`

Appendix B

Dunyazad's ASP Code

This appendix contains listings for all of *Dunyazad*'s core answer set code, which run together can be used to produce a single choice. The imperative code that supervises story construction and turns answer sets into English text is not included here, but the entire project can be found at `https://github.com/solsword/dunyazad`, and an archived snapshot current as of this writing can be found at `www.escholarship.org/uc/item/32d6p0kg`. Each page in this appendix lists the source filename at the bottom of the page, and within each category, files are presented in alphabetical order. Additionally, every fifth line is numbered in the left margin for ease of reference.

B.1 Utilities

The `utils.lp` file contains a variety of utility definitions, along with some Python code for generating unique instance IDs and formatting strings (*Dunyazad* requires `clingo` to be compiled with Python support). This Python code is called internally by `clingo` during the solving process, in contrast to the external Python code that calls the solver for each individual choice in a story and translates

answer sets into English text (that code is not listed in this appendix).

— utils.lp —

```
% utils.lp
% Various utilities, including script functions.
% vim: syn=python

unique_key(State, @unique(M)) :-
  max_unique(M),
  get_unique_key(State).

unique_key_used(K) :-
  unique_key(State, K).

#script (python)
import gringo

def capitalize(arg):
  s = str(arg)
  return s[0].upper() + s[1:]

def the(arg):
  s = str(arg)
  return "the␣" + s

def an(arg):
  s = str(arg)
  if s[0] in "aeiouAEIOU":
    return "an␣" + s
  else:
    return "a␣" + s

def pred(str):
  return gringo.Fun(str)

def join(*args):
  return ''.join(str(a) for a in args)

def join_(*args):
  return '_'.join(str(a) for a in args)

def fmt(tmplt, *args):
  return str(tmplt).format(*(str(a) for a in args))
```

— utils.lp —

```
def join_lines(*args):
  return '\n'.join(str(a) for a in args)

def mkmem(*args):
  return '.'.join(str(a) for a in args)

def unique(lastroundmax, state=[0]):
  lastroundmax = int(lastroundmax)
  if state[0] <= lastroundmax:
    state[0] = lastroundmax + 1
  state[0] += 1
  return state[0]

def inst(base, number):
  return gringo.Fun(str(base) + "_" + str(number))

#end.
```

B.2 Core Files

The core files include all of *Dunyazad*'s main constraints, and are summarized in table 6.2. Not included in this category are the files in the `content/` directory, which each define individual instances of things like goals and actions. Those files do rely heavily on predicates defined by core files to concisely specify content, of course.

— actions.lp —

```
% actions.lp
% Rules about actions.

% Every option has an action:
1 = {
  at(N, action(option(X), Act)) : action(Act);
  error(m("Option without action.", N, option(X)))
} :-
  at(N, option(X)),
  story_op(N, build_options).
```

— actions.lp —

```
% And every action has a value for each outcome variable:
outcome_variable(Action, Var) :-
  outcome_val(Action, Var, Value).

1 = {
  at(N, outcome(X, o(Var, Val))) : outcome_val(Action, Var, Val);
  error(m("Unassigned␣outcome␣variable.", N, X, Action, Var))
} :-
  at(N, action(X, Action)),
  outcome_variable(Action, Var),
  story_op(N, build_options).

error(m("Exclusive␣outcome␣values.", N, X)) :-
  at(N, action(X, Action)),
  at(N, outcome(X, o(Var1, Val1))),
  at(N, outcome(X, o(Var2, Val2))),
  Var1 != Var2,
  outcome_excludes(Action, o(Var1, Val1), o(Var2, Val2)),
  story_op(N, build_options).

% And all action arguments are filled in:
1 = {
  at(N, arg(X, Arg, Inst)) : is_instance(N, Inst, Class);
  error(m("Unbound␣action␣argument.", N, Action, Arg, Class))
} :-
  at(N, action(X, Action)),
  argument(Action, Arg, Class),
  Class != action,
  story_op(N, build_options).

1 = {
  at(N, arg(X, Arg, ActType)) : action(ActType);
  error(m("Unbound␣action␣argument.", N, Action, Arg, action))
} :-
  at(N, action(X, Action)),
  argument(Action, Arg, action),
  story_op(N, build_options).

% The concept of an "initiator:"
at(N, initiator(X, Initiator)) :-
  at(N, action(X, Action)),
  at(N, arg(X, InitArg, Initiator)),
  initiator(Action, InitArg),
  story_op(N, build_options).

% An action without an initiator is an error:
error(m("Action␣without␣initiator.", N, X)) :-
  at(N, option(X)),
  0 = {
```

— actions.lp —

```
      at(N, initiator(option(X), Anyone))
    },
    story_op(N, build_options).

% Actual consequences depend on which outcome variables have which
% values:
at(N, consequence(X, Consequence)) :-
  at(N, consequence_of(X, Outcome, Consequence)),
  at(N, outcome(X, Outcome)),
  story_op(N, build_options).

at(N, consequence(X, Modifier, Consequence)) :-
  at(N, consequence_of(X, Outcome, Modifier, Consequence)),
  at(N, outcome(X, Outcome)),
  story_op(N, build_options).

% Being injured, polymorphed, dead, or off-stage limits what actions
% you can take and which actions can be taken on you:

error(m("Injured␣actor␣initiated␣an␣action.", N, X)) :-
  at(N, action(X, Action)),
  at(N, initiator(X, Initiator)),
  st(N, state(injured, Initiator)),
  not injured_can_initiate(Action),
  story_op(N, build_options).

error(m("Interaction␣with␣dead␣actor.", N, X)) :-
  at(N, action(X, Action)),
  at(N, arg(X, Arg, Inst)),
  st(N, state(dead, Inst)),
  not dead_okay(Action, Arg),
  story_op(N, build_options).

error(m("Interaction␣with␣off-stage␣instance.", N, X)) :-
  at(N, action(X, Action)),
  at(N, arg(X, Arg, Inst)),
  st(N, state(off_stage, Inst)),
  not off_stage_okay(Action, Arg),
  story_op(N, build_options).

% Outcomes are surprising if they're unlikely based on skills/tools:
at(N, surprising(X)) :-
  at(N, outcome(X, Outcome)),
  at(N, unlikely_outcome(X, Outcome)),
  story_op(N, build_options).

% Outcomes are also surprising if there's a likely outcome but
% something else happens:
at(N, surprising(X)) :-
```

— actions.lp —

```
    at(N, likely_outcome(X, o(OutVar, Likely))),
    at(N, outcome(X, o(OutVar, NotLikely))),
    NotLikely != Likely,
    story_op(N, build_options).

% Only chaotic actions may have surprising outcomes:
% Note: now any outcome may be surprising because outcome feel
% constraints can be used to constrain surprise if desired.

%error(m("Non-chaotic action had surprising outcome.", N, X)) :-
%  at(N, action(X, Action)),
%  at(N, surprising(X)),
%  not chaotic(Action).
```

— actors.lp —

```
% actors.lp
% Special rules for actors. See content/actors/*.lp for actor
% definitions.

% Actor definition unpacking:
subclass(Imm, Spec) :-
  actor_def(Spec, Imm, Name, Number, Gender).

concrete(Spec) :-
  actor_def(Spec, Imm, Name, Number, Gender).

default_name_for(Class, Name) :-
  actor_def(Class, Superclass, Name, Number, Gender).

default_number_for(Class, Number) :-
  actor_def(Class, Superclass, Name, Number, Gender).

default_gender_for(Class, Gender) :-
  gender(Gender),
  actor_def(Class, Superclass, Name, Number, Gender).

default_gender_for(Class, masculine) :-
  actor_def(Class, Superclass, Name, Number, either).

default_gender_for(Class, feminine) :-
  actor_def(Class, Superclass, Name, Number, either).

% Some  high-level actor ontology structure:
subclass(actor, person).
```

— actors.lp —

```
concrete(person).
default_name_for(person, "person").
default_number_for(person, singular).
default_gender_for(person, masculine).
default_gender_for(person, feminine).
subclass(actor, animal).
concrete(animal).
default_name_for(animal, "animal").
default_number_for(animal, singular).
default_gender_for(animal, masculine).
default_gender_for(animal, feminine).
class_skill(animal, wilderness_lore, always).
class_skill(animal, unintelligent, always).

% Power relations:

at(N, is_powerful(Inst)) :-
  is_instance(N, Inst, Category),
  powerful(Category).

at(N, is_powerless(Inst)) :-
  is_instance(N, Inst, Category),
  powerless(Category).
```

— choice_structure.lp —

```
% option constraints:
% -------------------

% relevance: any action which might cause a potential to dissappear
% (via any mechanism) is relevant
at(N, relevant_to(option(X), potential(PTyp, Something))) :-
  at(N, consequence_of(option(X), Outcome, _not, Something)),
  at(N, potential(PTyp, Something)),
  at(N, option(X)),
  not at(N, action(option(X), travel_onwards)),
  not at(N, action(option(X), reach_destination)),
  story_op(N, build_options).

error(m("Irrelevant␣option", N, option(X))) :-
  at(N, option(X)),
  0 = {
    at(N, relevant_to(option(X), Anything))
  },
  not at(N, action(option(X), travel_onwards)),
```

— choice_structure.lp —

```
    not at(N, action(option(X), reach_destination)),
    story_op(N, build_options).

error(
  m(
    "Option␣relevant␣but␣only␣to␣unimportant␣potential.",
    N,
    option(X)
  )
) :-
  at(N, option(X)),
  at(N, relevant_to(option(X), Potential)),
  0 = {
    at(N, relevant_to(option(X), ImportantPotential)) :
      at(N, relevant_to(option(X), ImportantPotential)),
      at(N, most_important(ImportantPotential))
  },
  not at(N, action(option(X), travel_onwards)),
  not at(N, action(option(X), reach_destination)),
  story_op(N, build_options).

error(m("Unaddressed␣important␣potential", N)) :-
  at(N, most_important(potential(PTyp, Something))),
  0 = {
    at(N, option(X))
      : at(N, relevant_to(option(X), potential(PTyp, Something)))
  },
  2 <= { at(N, option(X)) },
  node_type(N, choice),
  story_op(N, build_options).

% motivation

% Resolving or nullifying your own problems:
at(N, motivated(X)) :-
  at(
    N,
    consequence_of(X, Outcome, resolves, potential(PType, PState))
  ),
  at(N, potential(problem, PState)),
  at(N, initiator(X, Initiator)),
  at(N, problematic_for(potential(PType, PState), Initiator)),
  story_op(N, build_options).

at(N, motivated(X)) :-
  at(
    N,
    consequence_of(X, Outcome, nullifies, potential(PType, PState))
  ),
```

— choice_structure.lp —

```
    at(N, potential(problem, PState)),
    at(N, initiator(X, Initiator)),
    at(N, problematic_for(potential(PType, PState), Initiator)),
    story_op(N, build_options).

% Actions that enable a goal other than "succeed at your actions:"
at(N, motivated(X)) :-
    at(N, expectation(X, enables, Goal)),
    Goal != as_intended(Initiator),
    at(N, initiator(X, Initiator)),
    st(N, state(party_member, Initiator)),
    story_op(N, build_options).

% Taking advantage of opportunities (except those you're offering):
at(N, motivated(X)) :-
  at(
    N,
    consequence_of(
      X,
      Outcome,
      resolves,
      potential(opportunity, Opportunity)
    )
  ),
  at(N, potential(opportunity, Opportunity)),
  at(N, initiator(X, Initiator)),
  not at(
    N,
    initiated_by(potential(opportunity, Opportunity), Initiator)
  ),
  story_op(N, build_options).

% Manifesting opportunities that you initiated:
at(N, motivated(X)) :-
  at(
    N,
    consequence_of(
      X,
      Outcome,
      manifests,
      potential(PType, PState)
    )
  ),
  at(N, potential(PType, PState)),
  at(N, initiator(X, Initiator)),
  at(N, initiated_by(potential(PType, PState), Initiator)),
  story_op(N, build_options).
```

— choice_structure.lp —

```
% Let's just keep going, shall we?
at(N, motivated(X)) :-
  at(N, action(X, travel_onwards)),
  story_op(N, build_options).

% We're almost there!
at(N, motivated(X)) :-
  at(N, action(X, reach_destination)),
  story_op(N, build_options).

error(m("Unmotivated␣action", N, option(X))) :-
  at(N, option(X)),
  not at(N, motivated(option(X))).

% redundancy
error(m("Redundant␣option", N, X)) :-
  node_type(N, choice),
  at(N, action(X, Action)),
  at(N, action(Y, Action)),
  at(N, relevant_to(X, P1)),
  at(N, relevant_to(Y, P1)),
  X < Y,
  story_op(N, build_options).

error(m("Redundant␣option", N, X)) :-
  at(N, action(X, travel_onwards)),
  at(N, action(Y, travel_onwards)),
  X < Y,
  story_op(N, build_options).

error(m("Redundant␣option", N, X)) :-
  at(N, action(X, reach_destination)),
  at(N, action(Y, reach_destination)),
  X < Y,
  story_op(N, build_options).

% boredom (at a single node)

error(m("Boring␣options", N)) :-
  at(N, action(X, Same)),
  at(N, action(Y, Same)),
  at(N, action(Z, Same)),
  X < Y, Y < Z,
  story_op(N, build_options).

% boredom (over time)
error(m("Boring␣action", N, X)) :-
  at(Prev, action(X, Action)),
  successor(Prev, POpt, N),
```

— choice_structure.lp —

```
    at(N, action(X, Action)),
    0 = {
      at(N, outcome(X, Outcome))
        : at(N, outcome(X, Outcome)),
          not at(Prev, outcome(POpt, Outcome))
    }
    story_op(N, build_options).

% your party as the active agent:
error(m("Non-party-initiated␣option", N, option(X))) :-
  node_type(N, choice),
  at(N, initiator(option(X), Init)),
  not st(N, state(party_member, Init)),
  story_op(N, build_options).

% your party shouldn't act outside of choices:
error(m("Party-initiated␣event", N, X)) :-
  node_type(N, event),
  at(N, initiator(X, Init)),
  st(N, state(party_member, Init)),
  not at(N, action(X, travel_onwards)),
  not at(N, action(X, reach_destination)),
  story_op(N, build_options).

% No trick options:
error(m("Trick␣option", N, X)) :-
  at(N, action(X, Action)),
  at(N, arg(X, Arg, PartyMember)),
  st(N, state(party_member, PartyMember)),
  st(N, state(party_member, OtherMember)),
  default_intent(Action, Outcome),
  at(N, skill_link(Skill, required, NeedsTool, Action, Arg, Outcome)),
  not st(N, property(has_skill, PartyMember, Skill)),
  st(N, property(has_skill, OtherMember, Skill)),
  not st(N, state(injured, OtherMember)),
  not st(N, state(dead, OtherMember)).

error(m("Trick␣option", N, X)) :-
  at(N, action(X, Action)),
  at(N, arg(X, Arg, PartyMember)),
  st(N, state(party_member, PartyMember)),
  st(N, state(party_member, OtherMember)),
  default_intent(Action, Outcome),
  at(N, skill_link(Skill, promotes, NeedsTool, Action, Arg, Outcome)),
  not st(N, property(has_skill, PartyMember, Skill)),
  st(N, property(has_skill, OtherMember, Skill)),
  not st(N, state(injured, OtherMember)),
  not st(N, state(dead, OtherMember)).

```

— choice_structure.lp —

```
error(m("Trick␣option", N, X)) :-
  at(N, action(X, Action)),
  at(N, arg(X, Arg, PartyMember)),
  st(N, state(party_member, PartyMember)),
  st(N, state(party_member, OtherMember)),
  default_intent(Action, Outcome),
  at(
    N,
    skill_link(
      Skill, contest, NeedsTool,
      Action,
      between(Arg, Opponent),
      either(Outcome, Bad)
    )
  ),
  not st(N, property(has_skill, PartyMember, Skill)),
  st(N, property(has_skill, OtherMember, Skill)).

error(m("Trick␣option", N, X)) :-
  at(N, action(X, Action)),
  at(N, arg(X, Arg, PartyMember)),
  st(N, state(party_member, PartyMember)),
  st(N, state(party_member, OtherMember)),
  default_intent(Action, Outcome),
  at(
    N,
    skill_link(
      Skill, contest, NeedsTool,
      Action,
      between(Opponent, Arg),
      either(Bad, Outcome)
    )
  ),
  not st(N, property(has_skill, PartyMember, Skill)),
  st(N, property(has_skill, OtherMember, Skill)).

% overall structure constraints:
% ------------------------------

% the story shouldn't end too soon:
error(m("Story␣too␣short", N)) :-
  node_type(N, ending),
  shortest_path(N, L),
  L < min_story_length.

% choices should be frequent
error(m("Not␣enough␣choices", A, B, C)) :-
  successor(A, AOpt, B),
  successor(B, BOpt, C),
```

— choice_structure.lp —

```
    not node_type(A, choice),
    not node_type(B, choice),
    not node_type(C, choice),
    not resolves_vignette(B, BOpt),
    not at(B, action(BOpt, travel_onwards)),
    0 = {
      at(C, action(AnyOpt, travel_onwards))
    },
    story_op(C, build_options).

% events shouldn't re-hash the choice you just made or a prevoius
% event
arg_mismatch(N, option(NO), O, option(OO)) :-
  at(N, arg(option(NO), Arg, Val)),
  at(O, option(OO)),
  0 = {
    at(O, arg(option(OO), Arg, Val))
      : at(O, arg(option(OO), Arg, Val))
  },
  N < O,
  story_op(N, build_options).

arg_mismatch(O, OOpt, N, NOpt) :-
  arg_mismatch(N, NOpt, O, OOpt),
  story_op(N, build_options).

error(m("Event␣duplicates␣previous␣action", N, Prev, Action)) :-
  successor(Prev, Opt, N),
  node_type(N, event),
  at(Prev, action(Opt, Action)),
  at(N, action(AnyOpt, Action)),
  not arg_mismatch(Prev, Opt, N, AnyOpt),
  story_op(N, build_options).

% failing to deal with a potential twice is enough
error(
  m("Persistent␣potential␣is␣getting␣boring", Three, option(Z))
) :-
  story_node(One),
  successor(One, option(X), Two),
  successor(Two, option(Y), Three),
  at(One, potential(PTyp, Something)),
  at(Two, potential(PTyp, Something)),
  at(Three, potential(PTyp, Something)),
  at(One, consequence_of(option(X), DidntHappen, _not, Something)),
  not at(One, outcome(option(X), DidntHappen)),
  at(Two, consequence_of(option(Y), AlsoDidntHappen, _not,Something)),
  not at(Two, outcome(option(Y), AlsoDidntHappen)),
  at(Three, consequence_of(option(Z), MustHappen, _not, Something)),
```

— choice_structure.lp —

```
    not at(Three, outcome(option(Z), MustHappen)),
    story_op(Three, build_options).

% failing to deal with a problem is no excuse for ignoring it
% (opportunities don't care)
error(m("Ignored␣failure␣to␣solve␣problem", Two, option(Y))) :-
  story_node(One),
  successor(One, option(X), Two),
  successor(Two, option(Y), Three),
  at(One, potential(problem, Problem)),
  at(Two, potential(problem, Problem)),
  at(One, consequence_of(option(X), DidntHappen, _not, Problem)),
  0 = { at(Two, consequence_of(option(Y), Any, _not, Problem)) },
  story_op(Two, build_options).

% problems shouldn't re-occur within a vignette
error(m("Recurring␣potential", Second)) :-
  story_node(First),
  before(First, Second),
  vignette(First, V),
  vignette(Second, V),
  at(First, potential(PType, Pot)),
  not at(Second, potential(PType, Pot)),
  at(Second, consequence(option(X), Pot)),
  story_op(Second, build_options).

% vignettes shouldn't drag on too long
error(m("Vignette␣dragging␣on...", N, option(X))) :-
  vignette(N, R),
  before(R, I1),
  before(I1, I2),
  before(I2, I3),
  before(I3, N), % N is at least the fourth node since R on some path
  at(N, option(X)),
  not resolves_vignette(N, option(X)),
  not at(N, action(option(X), travel_onwards)),
  story_op(N, build_options).

% vignette structure constraints:
% -------------------------------

% vignette-level boredom
error(m("Repeated␣vignette␣setup", N, R)) :-
  story_op(N, build_options),
  setup(N, Boring),
  successor(Prev, Opt, N),
  vignette(Prev, R),
  setup(R, Boring).
```

— choice_structure.lp —

```
error(m("Repeated␣vignette␣setup", N, R)) :-
  story_op(Prev, add_branch_nodes),
  setup(N, Boring),
  successor(Prev, Opt, N),
  vignette(Prev, R),
  setup(R, Boring).

times_setup_used_before(N, Setup, Count) :-
  story_op(N, build_options),
  possible_setup(Setup),
  Count = {
    setup(Prev, Setup) : before(Prev, N)
  }.

unique_setups_used_before(N, Count) :-
  story_op(N, build_options),
  Count = {
    possible_setup(Setup) : setup(Prev, Setup), before(Prev, N)
  }.

error(m("Boring␣vignette␣setup", N)) :-
  story_op(N, build_options),
  setup(N, Boring),
  times_setup_used_before(N, Boring, Boredom),
  unique_setups_used_before(N, Unique),
  Boredom > 0,
  Boredom + 1 >= Unique.

% every vignette ends with one of the special actions "travel onwards"
% or "reach destination:"
% TODO: This!?!?

%1 = {
%  node_type(N, event);
%  node_type(N, ending);
%  error(m("Vignette resolution has wrong node type.", End, Opt, N))
%} :-
%  resolves_vignette(End, Opt),
%  successor(End, Opt, N),
%  story_op(N, build_options).
%
%at(N, action(option(1), travel_onwards)) :-
%  node_type(N, event),
%  resolves_vignette(End, Opt),
%  successor(End, Opt, N),
%  story_op(N, build_options).

at(N, action(option(1), reach_destination)) :-
  node_type(N, ending),
```

— choice_structure.lp —

```
    resolves_vignette(End, Opt),
    successor(End, Opt, N),
    story_op(N, build_options).

% choice poetics constraints:
% ---------------------------

choices_before(N, X) :-
    X = {
      node_type(B, choice) : before(B, N)
    },
    story_op(N, build_options).

% the story must contain at least a minimum number of choices:
error(m("Too few choices before ending", N)) :-
    node_type(N, ending),
    choices_before(N, L),
    L < min_story_choices.

% The first few choices have a set structure:
%option_structure_plan(0, relaxed).
%option_structure_plan(1, relaxed).
%option_structure_plan(2, straightforward).
%option_structure_plan(3, mysterious).

% TODO: Get rid of this?
%error(m("Deviated from initial choices plan", N)) :-
%  node_type(N, choice),
%  story_op(N, build_options),
%  choices_before(N, L),
%  option_structure_plan(L, CS),
%  not at(N, option_structure(CS)).
```

— core.lp —

```
% core rules:
% -----------

%#minimize { 1@100, error(Message) : error(Message) }.
:- error(Message).
%:- error(m("Choice node missing option 1.", N)).
%:- error(m("Choice node missing option 2.", N)).
%:- error(m("Error with instance number.", Inst)).

%:- error(m("Action without initiator.",N,X)).
%:- error(m("Non-party-initiated option",N,X)).
%:- error(m("Irrelevant option",N,X)).
%:- error(m("Unaddressed important potential",N)).
%:- error(m("Unmotivated action",N,X)).

%:- error(m("Story too short", N)).
%:- error(m("Skipped transition", N)).
%:- error(m("Transition is too early", N)).
%:- error(m("Reached destination before end of story", N, X)).
%:- error(m("Post-travel node without setup.", N, X, Next)).
%:- error(m("Unmotivated action", N, X)).
%:- error(m("Redundant option", N, X)).
%:- error(m("Left a non-hidden problem behind.", N, P)).
%:- error(m("Party-initiated event", N, X)).
%:- error(m("Vignette starts with an event!")).
%:- error(m("Irrelevant option", Node, Option)).
%:- error(m("Unbound action argument.", Node, Action, Arg, ArgType)).
%:- error(m("Price is not a treasure.", Node, Option)).

#const max_options=3.
#const max_outcome_variables=5.
#const min_story_length=5.
#const min_story_choices=3.
#const max_setup_argument_arity=4.
#const max_party_size=3.

opt_num(1..max_options).
setup_arg_id(1..max_setup_argument_arity).
party_size_value(1..max_party_size).

outcome_variable(Action, Variable) :-
  outcome_val(Action, Variable, AnyVal).

error(m("Action␣has␣too␣many␣outcome␣variables.", Action)) :-
  action(Action),
  Count = {
```

— core.lp —

```
      outcome_variable(Action, Var) :
        outcome_variable(Action, Var)
    },
    Count > max_outcome_variables.

% Core node construction:
% -----------------------

% Choice nodes have at least two options:
2 <= {
  at(N, option(X)) : opt_num(X);
  error(m("Choice node missing option 1.", N));
  error(m("Choice node missing option 2.", N))
} <= max_options :-
  node_type(N, choice),
  story_op(N, build_options).

% Ending and event nodes have exactly one "option:"
at(N, option(1)) :-
  node_type(N, ending),
  story_op(N, build_options).

at(N, option(1)) :-
  node_type(N, event),
  story_op(N, build_options).

% State and action rules:
% -----------------------

% Exclusive properties and relationships:
exclusive(has_item).
exclusive(type).

at(N, consequence_of(X, O, _not, property(Prop, Obj, Old))) :-
  at(N, consequence_of(X, O, property(Prop, Obj, New))),
  exclusive(Prop),
  st(N, property(Prop, Obj, Old)),
  Old != New,
  story_op(N, build_options).

at(N, consequence_of(X, O, _not, relation(Rel, Old, Obj))) :-
  at(N, consequence_of(X, O, relation(Rel, New, Obj))),
  exclusive(Rel),
  st(N, relation(Rel, Old, Obj)),
  Old != New,
  story_op(N, build_options).

at(N, consequence_of(X, O, _not, relation(Rel, Old, Obj))) :-
  at(N, consequence_of(X, O, relation(Rel, New, Obj))),
```

— core.lp —

```
    exclusive(Rel),
    st(N, relation(Rel, Old, Obj)),
    Old != New,
    story_op(N, build_options).

% Most actions can't be performed reflexively:
error(m("Improperly␣reflexive␣action.", N, Opt)) :-
  reflexive(N, Opt),
  at(N, action(Opt, Action)),
  not reflexive(Action).

reflexive(N, Opt) :-
  at(N, arg(Opt, Arg1, inst(Type, ID))),
  at(N, arg(Opt, Arg2, inst(Type, ID))),
  Arg1 < Arg2.


% Typing rules:
% -------------

general_category(actor).
general_category(item).
% TODO: locations?

category(N, Thing, Category) :-
  is_instance(N, Thing, Category),
  general_category(Category).

category_for(Category, Category) :-
  general_category(Category).

category_for(Class, Category) :-
  subclass(Category, Class),
  general_category(Category).

any_class(Class) :- category_for(Class, SomeCategory).

error(m("Instance␣doesn't␣belong␣to␣a␣general␣category.", N, I)) :-
  st(N, inst(Type, I)),
  0 = {
    category(N, inst(Type, I), Category)
  }.

is_instance(N, Thing, Class) :-
  st(N, property(type, Thing, Class)).

is_instance(N, Thing, Class) :-
  is_instance(N, Thing, Subclass),
  subclass(Class, Subclass).
```

— core.lp —

```
145
    subclass(General, Specific) :-
      subclass(General, Intermediate),
      subclass(Intermediate, Specific).

150 concrete_class_of(Abstract, Concrete) :-
      subclass(Abstract, Concrete),
      concrete(Concrete).

    concrete_class_of(Concrete, Concrete) :-
155   concrete(Concrete).
```

— eval.lp —

```
1  % eval.lp
   % Rules dealing with how choice structures are evaluated.

   % Expectations:
5  %--------------

   option_relevance(irrelevant).
   option_relevance(threatens).
   option_relevance(enables).
10 option_relevance(hinders).
   option_relevance(advances).

   stakes_level(none).
   stakes_level(low).
15 stakes_level(high).

   higher_stakes(low, none).
   higher_stakes(high, none).
   higher_stakes(high, low).
20
   at_least_as_high(SL1, SL2) :- higher_stakes(SL1, SL2).
   at_least_as_high(SL, SL) :- stakes_level(SL).

   % Options which have no other expectation are expected to be
25 % irrelevant:
   at(N, expectation(option(X), irrelevant, G)) :-
     at(N, option(X)),
     at(N, player_goal(G)),
     0 = {
30     at(N, expectation(option(X), threatens, G));
       at(N, expectation(option(X), enables, G));
```

— eval.lp —

```
      at(N, expectation(option(X), hinders, G));
      at(N, expectation(option(X), advances, G))
    }.

% Threatens and enables based on possible outcomes:

at(N, expectation(X, threatens, Goal)) :-
  at(N, consequence_of(X, Outcome, State)),
  state_hinders(Goal, State).

at(N, expectation(X, threatens, Goal)) :-
  at(N, consequence_of(X, Outcome, State)),
  state_fails(Goal, State).

at(N, expectation(X, threatens, Goal)) :-
  at(N, action(X, Action)),
  Action != travel_onwards, Action != reach_destination,
  at(N, consequence_of(X, Outcome, _not, State)),
  state_advances(Goal, State).

at(N, expectation(X, threatens, Goal)) :-
  at(N, action(X, Action)),
  Action != travel_onwards, Action != reach_destination,
  at(N, consequence_of(X, Outcome, _not, State)),
  state_achieves(Goal, State).

at(N, expectation(X, enables, Goal)) :-
  at(N, consequence_of(X, Outcome, State)),
  state_advances(Goal, State).

at(N, expectation(X, enables, Goal)) :-
  at(N, consequence_of(X, Outcome, State)),
  state_achieves(Goal, State).

at(N, expectation(X, enables, Goal)) :-
  at(N, action(X, Action)),
  Action != travel_onwards, Action != reach_destination,
  at(N, consequence_of(X, Outcome, _not, State)),
  state_hinders(Goal, State).

at(N, expectation(X, enables, Goal)) :-
  at(N, action(X, Action)),
  Action != travel_onwards, Action != reach_destination,
  at(N, consequence_of(X, Outcome, _not, State)),
  state_fails(Goal, State).

% Hinders and advances based on 'likely' outcomes:

at(N, expectation(X, hinders, Goal)) :-
```

— eval.lp —

```
    at(N, consequence_of(X, Outcome, State)),
    state_hinders(Goal, State),
    at(N, likely_outcome(X, Outcome)).

at(N, expectation(X, hinders, Goal)) :-
    at(N, consequence_of(X, Outcome, State)),
    state_fails(Goal, State),
    at(N, likely_outcome(X, Outcome)).

at(N, expectation(X, hinders, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence_of(X, Outcome, _not, State)),
    state_advances(Goal, State),
    at(N, likely_outcome(X, Outcome)).

at(N, expectation(X, hinders, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence_of(X, Outcome, _not, State)),
    state_achieves(Goal, State),
    at(N, likely_outcome(X, Outcome)).

at(N, expectation(X, advances, Goal)) :-
    at(N, consequence_of(X, Outcome, State)),
    state_advances(Goal, State),
    at(N, likely_outcome(X, Outcome)).

at(N, expectation(X, advances, Goal)) :-
    at(N, consequence_of(X, Outcome, State)),
    state_achieves(Goal, State),
    at(N, likely_outcome(X, Outcome)).

at(N, expectation(X, advances, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence_of(X, Outcome, _not, State)),
    state_hinders(Goal, State),
    at(N, likely_outcome(X, Outcome)).

at(N, expectation(X, advances, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence_of(X, Outcome, _not, State)),
    state_fails(Goal, State),
    at(N, likely_outcome(X, Outcome)).

% Special expectations for travel_onwards and reach_destination.
% Basically, if there's a state that would be removed by one of these
```

— eval.lp —

```
% actions, we know that we're actually leaving it behind, so if that
% statie is *either* bad_for/awful_for *or* good_for/great_for any
% goal, we expect to hinder that goal by leaving the state behind,
% because either we're leaving a problem behind which will now never
% get solved (in the case that the state is hindering/failing a goal)
% or we're moving out of a situaiton which is good/great for our goal.

at(N, expectation(option(X), hinders, Goal)) :-
  at(N, option(X)),
  1 = {
    at(N, action(option(X), travel_onwards));
    at(N, action(option(X), reach_destination))
  },
  at(N, consequence_of(option(X), Outcome, _not, State)),
  state_fails(Goal, State).

at(N, expectation(option(X), hinders, Goal)) :-
  at(N, option(X)),
  1 = {
    at(N, action(option(X), travel_onwards));
    at(N, action(option(X), reach_destination))
  },
  at(N, consequence_of(option(X), Outcome, _not, State)),
  state_hinders(Goal, State).

% This one is threatens instead of hinders.
at(N, expectation(option(X), threatens, Goal)) :-
  at(N, option(X)),
  1 = {
    at(N, action(option(X), travel_onwards));
    at(N, action(option(X), reach_destination))
  },
  at(N, consequence_of(option(X), Outcome, _not, State)),
  state_advances(Goal, State).

at(N, expectation(option(X), hinders, Goal)) :-
  at(N, option(X)),
  1 = {
    at(N, action(option(X), travel_onwards));
    at(N, action(option(X), reach_destination))
  },
  at(N, consequence_of(option(X), Outcome, _not, State)),
  state_achieves(Goal, State).


% Stakes are based on the goals that are threatened:

% Percieved stakes (per-option):
at(N, perceived_option_stakes(X, high)) :-
```

```
    story_op(N, build_options),
    at(N, expectation(X, Exp, Goal)),
    Exp != irrelevant,
    at(N, goal_stakes(Goal, high)),
    at(N, player_goal(Goal)).

at(N, perceived_option_stakes(X, low)) :-
    story_op(N, build_options),
    at(N, expectation(X, Exp, Goal)),
    Exp != irrelevant,
    at(N, goal_stakes(Goal, low)),
    at(N, player_goal(Goal)),
    not at(N, perceived_option_stakes(X, high)).

at(N, perceived_option_stakes(option(X), none)) :-
    story_op(N, build_options),
    at(N, option(X)),
    not at(N, perceived_option_stakes(option(X), high)),
    not at(N, perceived_option_stakes(option(X), low)).

% Percieved stakes (overall):
at(N, perceived_stakes(high)) :-
    at(N, perceived_option_stakes(X, high)).

at(N, perceived_stakes(low)) :-
    at(N, perceived_option_stakes(X, low)),
    0 = {
      at(N, perceived_option_stakes(Any, high))
    }.

at(N, perceived_stakes(none)) :-
    story_node(N),
    0 = {
      at(N, perceived_option_stakes(Any, high));
      at(N, perceived_option_stakes(Any, low))
    }.

% Actual stakes (per outcome):
at(N, outcome_stakes(X, high)) :-
    story_op(N, build_options),
    at(N, outcome_perception(X, Any, Goal)),
    at(N, goal_stakes(Goal, high)),
    at(N, player_goal(Goal)).

at(N, outcome_stakes(X, low)) :-
    story_op(N, build_options),
    at(N, outcome_perception(X, Any, Goal)),
    at(N, goal_stakes(Goal, low)),
    at(N, player_goal(Goal)),
```

— eval.lp —

```
    not at(N, outcome_stakes(X, high)).

at(N, outcome_stakes(option(X), none)) :-
    story_op(N, build_options),
    at(N, option(X)),
    not at(N, outcome_stakes(option(X), high)),
    not at(N, outcome_stakes(option(X), low)).

% Actual stakes (overall):
at(N, actual_stakes(high)) :-
    story_op(N, build_options),
    at(N, outcome_perception(X, Any, Goal)),
    Any != irrelevant,
    at(N, player_goal(Goal)),
    at(N, goal_stakes(Goal, high)).

at(N, actual_stakes(low)) :-
    story_op(N, build_options),
    at(N, outcome_perception(X, Any, Goal)),
    Any != irrelevant,
    at(N, player_goal(Goal)),
    at(N, goal_stakes(Goal, low)),
    not at(N, actual_stakes(high)).

at(N, actual_stakes(none)) :-
    story_op(N, build_options),
    not at(N, actual_stakes(high)),
    not at(N, actual_stakes(low)).

% different types of option feels:

at(N, option_feel(option(X), sure_thing)) :-
    story_op(N, build_options),
    at(N, option(X)),
    at(N, player_goal(Goal)),
    at(N, expectation(option(X), advances, Goal)),
    at(N, goal_stakes(Goal, Stakes)),
    Stakes != none,
    0 = {
      at(N, player_goal(G2))
        : at(N, expectation(option(X), threatens, G2));
      at(N, player_goal(G3))
        : at(N, expectation(option(X), hinders, G3))
    }.

at(N, option_feel(option(X), safe)) :-
    story_op(N, build_options),
    at(N, option(X)),
    1 <= {
```

— eval.lp —

```
      at(N, player_goal(G1)) :
        at(N, expectation(option(X), advances, G1)),
        at(N, goal_stakes(G1, Stakes1)),
        higher_stakes(Stakes1, none);
      at(N, player_goal(G2)) :
        at(N, expectation(option(X), enables, G2)),
        at(N, goal_stakes(G2, Stakes2)),
        higher_stakes(Stakes2, none)
  },
  0 = {
    at(N, player_goal(G3))
      : at(N, expectation(option(X), threatens, G3));
    at(N, player_goal(G4))
      : at(N, expectation(option(X), hinders, G4))
  }.

at(N, option_feel(option(X), hopeful)) :-
  story_op(N, build_options),
  at(N, option(X)),
  1 <= {
    at(N, perceived_stakes(low));
    at(N, perceived_stakes(high))
  },
  at(N, player_goal(HopeGoal)),
  at(N, expectation(option(X), advances, HopeGoal)),
  at(N, goal_stakes(HopeGoal, HopeStakes)),
  at(N, player_goal(ThreatGoal)),
  at(N, expectation(option(X), threatens, ThreatGoal)),
  at(N, goal_stakes(ThreatGoal, ThreatStakes)),
  at_least_as_high(HopeStakes, ThreatStakes),
  higher_stakes(ThreatStakes, none),
  0 = {
    at(N, player_goal(G3)) :
      at(N, expectation(option(X), threatens, G3)),
      at(N, goal_stakes(G3, S3)),
      higher_stakes(S3, ThreatStakes)
  },
  0 = {
    at(N, player_goal(G4))
      : at(N, expectation(option(X), hinders, G4))
  }.

at(N, option_feel(option(X), risky)) :-
  story_op(N, build_options),
  at(N, option(X)),
  at(N, player_goal(HopeGoal)),
  at(N, expectation(option(X), enables, HopeGoal)),
  at(N, goal_stakes(HopeGoal, HopeStakes)),
  at(N, player_goal(ThreatGoal)),
```

— eval.lp —

```
    at(N, expectation(option(X), threatens, ThreatGoal)),
    at(N, goal_stakes(ThreatGoal, ThreatStakes)),
    at_least_as_high(ThreatStakes, HopeStakes),
    higher_stakes(HopeStakes, none),
    0 = {
      at(N, player_goal(G3))
        : at(N, expectation(option(X), advances, G3));
      at(N, player_goal(G4))
        : at(N, expectation(option(X), hinders, G4))
    }.

at(N, option_feel(option(X), tradeoff)) :-
    story_op(N, build_options),
    at(N, option(X)),
    at(N, player_goal(AchievesGoal)),
    at(N, expectation(option(X), advances, AchievesGoal)),
    at(N, goal_stakes(AchievesGoal, Stakes)),
    at(N, player_goal(FailsGoal)),
    at(N, expectation(option(X), hinders, FailsGoal)),
    at(N, goal_stakes(FailsGoal, Stakes)),
    higher_stakes(Stakes, none).

at(N, option_feel(option(X), irrelevant)) :-
    story_op(N, build_options),
    at(N, option(X)),
    0 = {
      at(N, player_goal(G1)) :
        at(N, expectation(option(X), enables, G1)),
        at(N, goal_stakes(G1, Stakes1)),
        higher_stakes(Stakes1, none);
      at(N, player_goal(G2)) :
        at(N, expectation(option(X), advances, G2)),
        at(N, goal_stakes(G2, Stakes2)),
        higher_stakes(Stakes2, none);
      at(N, player_goal(G3)) :
        at(N, expectation(option(X), threatens, G3)),
        at(N, goal_stakes(G3, Stakes3)),
        higher_stakes(Stakes3, none);
      at(N, player_goal(G4)) :
        at(N, expectation(option(X), hinders, G4)),
        at(N, goal_stakes(G4, Stakes4)),
        higher_stakes(Stakes4, none)
    }.

at(N, option_feel(option(X), longshot)) :-
    story_op(N, build_options),
    at(N, option(X)),
    at(N, player_goal(FailGoal)),
    at(N, expectation(option(X), hinders, FailGoal)),
```

— eval.lp —

```
    at(N, goal_stakes(FailGoal, FailStakes)),
    at(N, player_goal(HopeGoal)),
    at(N, expectation(option(X), enables, HopeGoal)),
    at(N, goal_stakes(HopeGoal, HopeStakes)),
    at_least_as_high(FailStakes, HopeStakes),
    higher_stakes(HopeStakes, none),
    0 = {
      at(N, player_goal(G3))
        : at(N, expectation(option(X), advances, G3))
    }.

at(N, option_feel(option(X), bad)) :-
    story_op(N, build_options),
    at(N, option(X)),
    1 <= {
      at(N, player_goal(G1)) :
        at(N, expectation(option(X), threatens, G1)),
        at(N, goal_stakes(G1, Stakes1)),
        higher_stakes(Stakes1, none);
      at(N, player_goal(G2)) :
        at(N, expectation(option(X), hinders, G2)),
        at(N, goal_stakes(G2, Stakes2)),
        higher_stakes(Stakes2, none)
    },
    0 = {
      at(N, player_goal(G3))
        : at(N, expectation(option(X), enables, G3));
      at(N, player_goal(G4))
        : at(N, expectation(option(X), advances, G4))
    }.

at(N, option_feel(option(X), doomed)) :-
    story_op(N, build_options),
    at(N, option(X)),
    at(N, player_goal(DoomedGoal)),
    at(N, expectation(option(X), hinders, DoomedGoal)),
    at(N, goal_stakes(DoomedGoal, DoomedStakes)),
    higher_stakes(DoomedStakes, none),
    0 = {
      at(N, player_goal(G2))
        : at(N, expectation(option(X), enables, G2));
      at(N, player_goal(G3))
        : at(N, expectation(option(X), advances, G3))
    }.

% option-based choice structures:

at(N, option_structure(mysterious)) :-
    story_op(N, build_options),
```

— eval.lp —

```
    1 < { at(N, option(X)) },
    0 = #sum {
      1,at(N, option(X)) : at(N, option_feel(option(X), irrelevant));
      -1,at(N, option(X)) : at(N, option(X))
    }.

at(N, option_structure(uncertain)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    0 = #sum {
      1,at(N, option(X)) : at(N, option_feel(option(X), risky));
      -1,at(N, option(X)) : at(N, option(X))
    }.

at(N, option_structure(obvious)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    1 = {
      at(N, option(X)) : at(N, option_feel(option(X), hopeful));
      at(N, option(X)) : at(N, option_feel(option(X), safe))
    },
    0 = {
      at(N, option(X)) : at(N, option_feel(option(X), tradeoff));
      at(N, option(X)) : at(N, option_feel(option(X), irrelevant))
    }.

at(N, option_structure(relaxed)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    0 = #sum {
      1,at(N, option(X)) : at(N, option_feel(option(X), safe));
      -1,at(N, option(X)) : at(N, option(X))
    },
    at(N, perceived_stakes(Stakes)),
    higher_stakes(high, Stakes).

at(N, option_structure(powerful)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    1 <= {
      at(N, option(X)) : at(N, option_feel(option(X), hopeful));
      at(N, option(X)) : at(N, option_feel(option(X), safe))
    },
    0 = {
      at(N, player_goal(G1))
        : at(N, expectation(option(X), threatens, G1));
      at(N, player_goal(G2))
        : at(N, expectation(option(X), hinders, G2))
    },
```

— eval.lp —

```
    at(N, perceived_stakes(high)).

at(N, option_structure(uncomfortable)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    1 <= {
      at(N, option(X)) : at(N, option_feel(option(X), longshot));
      at(N, option(X)) : at(N, option_feel(option(X), risky))
    },
    0 = {
      at(N, option(X)) : at(N, option_feel(option(X), tradeoff));
      at(N, option(X)) : at(N, option_feel(option(X), irrelevant));
      at(N, option(X)) : at(N, option_feel(option(X), hopeful));
      at(N, option(X)) : at(N, option_feel(option(X), safe))
    },
    at(N, perceived_stakes(Stakes)),
    higher_stakes(Stakes, none).

at(N, option_structure(tradeoffs)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    0 = #sum {
      1,at(N, option(X)) : at(N, option_feel(option(X), tradeoff));
      -1,at(N, option(X)) : at(N, option(X))
    },
    at(N, player_goal(Goal1)),
    at(N, player_goal(Goal2)),
    Goal1 != Goal2,
    2 <= {
      at(N, option(X)) : at(N, expectation(option(X), advances, Goal1));
      at(N, option(X)) : at(N, expectation(option(X), advances, Goal2))
    },
    at(N, perceived_stakes(Stakes)),
    higher_stakes(Stakes, none).

at(N, option_structure(positive_alternatives)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    at(N, perceived_stakes(SomeStakes)),
    higher_stakes(SomeStakes, none),
    0 = #sum {
      1,at(N, option(X)) : at(N, option_feel(option(X), safe));
      1,at(N, option(X)) : at(N, option_feel(option(X), hopeful));
      -1,at(N, option(X)) : at(N, option(X))
    },
    0 = #sum {
      1,at(N, option(X)) :
        at(N, expectation(option(X), advances, SomeGoal)),
        at(N, goal_stakes(SomeGoal, SomeStakes)),
```

— eval.lp —

```
        at(N, player_goal(SomeGoal));
      -1,at(N, option(X)) : at(N, option(X))
    },
    at(N, player_goal(Goal1)),
    at(N, player_goal(Goal2)),
    Goal1 != Goal2,
    % TODO: This is a bit silly...
    2 <= {
      at(N, option(X)) : at(N, expectation(option(X), advances, Goal1));
      at(N, option(X)) : at(N, expectation(option(X), advances, Goal2))
    }.

at(N, option_structure(negative_alternatives)) :-
  story_op(N, build_options),
  1 < { at(N, option(X)) },
  at(N, perceived_stakes(SomeStakes)),
  higher_stakes(SomeStakes, none),
  0 = #sum {
    1,at(N, option(X)) : at(N, option_feel(option(X), longshot));
    1,at(N, option(X)) : at(N, option_feel(option(X), bad));
    -1,at(N, option(X)) : at(N, option(X))
  },
  0 = #sum {
    1,at(N, option(X)) :
      at(N, expectation(option(X), hinders, SomeGoal)),
      at(N, goal_stakes(SomeGoal, SomeStakes)),
      at(N, player_goal(Goal));
    -1,at(N, option(X)) : at(N, option(X))
  }.

at(N, option_structure(pressured)) :-
  story_op(N, build_options),
  1 < { at(N, option(X)) },
  0 = {
    at(N, option(X)) : at(N, option_feel(option(X), safe));
    at(N, option(X)) : at(N, option_feel(option(X), irrelevant))
  },
  1 <= {
    at(N, option(X)) : at(N, option_feel(option(X), hopeful));
    at(N, option(X)) : at(N, option_feel(option(X), tradeoff))
  },
  at(N, perceived_stakes(Stakes)),
  higher_stakes(Stakes, none).

at(N, option_structure(dangerous)) :-
  story_op(N, build_options),
  1 < { at(N, option(X)) },
  0 = {
    at(N, option(X)) : at(N, option_feel(option(X), safe));
```

— eval.lp —

```
      at(N, option(X)) : at(N, option_feel(option(X), hopeful));
      at(N, option(X)) : at(N, option_feel(option(X), tradeoff));
      at(N, option(X)) : at(N, option_feel(option(X), irrelevant))
    },
    1 <= {
      at(N, option(X)) : at(N, option_feel(option(X), risky))
    },
    at(N, perceived_stakes(high)).

at(N, option_structure(unfortunate)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    0 = {
      at(N, option(X)) : at(N, option_feel(option(X), safe));
      at(N, option(X)) : at(N, option_feel(option(X), hopeful));
      at(N, option(X)) : at(N, option_feel(option(X), tradeoff));
      at(N, option(X)) : at(N, option_feel(option(X), irrelevant))
    },
    1 <= {
      at(N, option(X)) : at(N, option_feel(option(X), risky))
    },
    at(N, perceived_stakes(Stakes)),
    higher_stakes(high, Stakes).

at(N, option_structure(bleak)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    0 = #sum {
      1,at(N, option(X)) : at(N, option_feel(option(X), longshot));
      1,at(N, option(X)) : at(N, option_feel(option(X), bad));
      -1,at(N, option(X)) : at(N, option(X))
    },
    at(N, perceived_stakes(Stakes)),
    higher_stakes(Stakes, none).

at(N, option_structure(depressing)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    0 = #sum {
      1,at(N, option(X)) :
        at(N, expectation(option(X), hinders, Goal)),
        at(N, player_goal(Goal));
      -1,at(N, option(X)) : at(N, option(X))
    },
    0 = {
      at(N, option(X)) :
        at(N, expectation(option(X), advances, Goal)),
        at(N, player_goal(Goal));
      at(N, option(X)) :
```

— eval.lp —

```
        at(N, expectation(option(X), enables, Goal)),
        at(N, player_goal(Goal))
    },
    at(N, perceived_stakes(Stakes)),
    higher_stakes(high, Stakes).

at(N, option_structure(doomed)) :-
    story_op(N, build_options),
    1 < { at(N, option(X)) },
    0 = #sum {
      1,at(N, option(X)) :
        at(N, expectation(option(X), hinders, Goal)),
        at(N, player_goal(Goal));
      -1,at(N, option(X)) : at(N, option(X))
    },
    0 = {
      at(N, option(X)) :
        at(N, expectation(option(X), advances, Goal)),
        at(N, player_goal(Goal));
      at(N, option(X)) :
        at(N, expectation(option(X), enables, Goal)),
        at(N, player_goal(Goal))
    },
    at(N, perceived_stakes(high)).

% TODO: Worry about options without any assigned feel?


% Outcomes:
%----------

outcome_perception(irrelevant).
outcome_perception(bad_for).
outcome_perception(good_for).
outcome_perception(awful_for).
outcome_perception(great_for).

% bad_for and good_for

at(N, outcome_perception(X, bad_for, Goal)) :-
  at(N, consequence(X, State)),
  state_hinders(Goal, State).

at(N, outcome_perception(X, good_for, Goal)) :-
  at(N, consequence(X, State)),
  state_advances(Goal, State).

at(N, outcome_perception(X, bad_for, Goal)) :-
  at(N, action(X, Action)),
```

```
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence(X, _not, State)),
    state_advances(Goal, State).

at(N, outcome_perception(X, good_for, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence(X, _not, State)),
    state_hinders(Goal, State).

at(N, outcome_perception(X, bad_for, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence(X, _not, State)),
    state_achieves(Goal, State).

at(N, outcome_perception(X, good_for, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence(X, _not, State)),
    state_fails(Goal, State).

% awful_for and great_for

at(N, outcome_perception(X, awful_for, Goal)) :-
    at(N, consequence(X, State)),
    state_fails(Goal, State).

at(N, outcome_perception(X, great_for, Goal)) :-
    at(N, consequence(X, State)),
    state_achieves(Goal, State).

% outcome perceptions for travel_onwards and reach_destination:

at(N, outcome_perception(option(X), awful_for, Goal)) :-
    at(N, option(X)),
    1 = {
      at(N, action(option(X), travel_onwards));
      at(N, action(option(X), reach_destination))
    },
    at(N, consequence(option(X), _not, State)),
    state_achieves(Goal, State).

at(N, outcome_perception(option(X), bad_for, Goal)) :-
    at(N, option(X)),
    1 = {
      at(N, action(option(X), travel_onwards));
      at(N, action(option(X), reach_destination))
    },
```

— eval.lp —

```
    at(N, consequence(option(X), _not, State)),
    state_advances(Goal, State).

at(N, outcome_perception(option(X), awful_for, Goal)) :-
    at(N, option(X)),
    1 = {
      at(N, action(option(X), travel_onwards));
      at(N, action(option(X), reach_destination))
    },
    at(N, consequence(option(X), _not, State)),
    state_hinders(Goal, State).

at(N, outcome_perception(option(X), awful_for, Goal)) :-
    at(N, option(X)),
    1 = {
      at(N, action(option(X), travel_onwards));
      at(N, action(option(X), reach_destination))
    },
    at(N, consequence(option(X), _not, State)),
    state_fails(Goal, State).

% Possible outcome perceptions (used to determine which outcomes are
% important):

% bad_for and good_for

at(N, possible_outcome_perception(X, O, bad_for, Goal)) :-
    at(N, consequence_of(X, O, State)),
    state_hinders(Goal, State).

at(N, possible_outcome_perception(X, O, good_for, Goal)) :-
    at(N, consequence_of(X, O, State)),
    state_advances(Goal, State).

at(N, possible_outcome_perception(X, O, bad_for, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence_of(X, O, _not, State)),
    state_advances(Goal, State).

at(N, possible_outcome_perception(X, O, good_for, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence_of(X, O, _not, State)),
    state_hinders(Goal, State).

at(N, possible_outcome_perception(X, O, bad_for, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
```

```
    at(N, consequence_of(X, O, _not, State)),
    state_achieves(Goal, State).

at(N, possible_outcome_perception(X, O, good_for, Goal)) :-
    at(N, action(X, Action)),
    Action != travel_onwards, Action != reach_destination,
    at(N, consequence_of(X, O, _not, State)),
    state_fails(Goal, State).

% awful_for and great_for

at(N, possible_outcome_perception(X, O, awful_for, Goal)) :-
    at(N, consequence_of(X, O, State)),
    state_fails(Goal, State).

at(N, possible_outcome_perception(X, O, great_for, Goal)) :-
    at(N, consequence_of(X, O, State)),
    state_achieves(Goal, State).

% possible outcome perceptions for travel_onwards and
% reach_destination:

at(
  N,
  possible_outcome_perception(option(X), Outcome, awful_for, Goal)
) :-
    at(N, option(X)),
    1 = {
      at(N, action(option(X), travel_onwards));
      at(N, action(option(X), reach_destination))
    },
    at(N, consequence_of(option(X), Outcome, _not, State)),
    state_achieves(Goal, State).

at(
  N,
  possible_outcome_perception(option(X), Outcome, bad_for, Goal)
) :-
    at(N, option(X)),
    1 = {
      at(N, action(option(X), travel_onwards));
      at(N, action(option(X), reach_destination))
    },
    at(N, consequence_of(option(X), Outcome, _not, State)),
    state_advances(Goal, State).

at(
  N,
  possible_outcome_perception(option(X), Outcome, awful_for, Goal)
```

— eval.lp —

```
  ) :-
    at(N, option(X)),
    1 = {
      at(N, action(option(X), travel_onwards));
      at(N, action(option(X), reach_destination))
    },
    at(N, consequence_of(option(X), Outcome, _not, State)),
    state_hinders(Goal, State).

at(
  N,
  possible_outcome_perception(option(X), Outcome, awful_for, Goal)
) :-
  at(N, option(X)),
  1 = {
    at(N, action(option(X), travel_onwards));
    at(N, action(option(X), reach_destination))
  },
  at(N, consequence_of(option(X), Outcome, _not, State)),
  state_fails(Goal, State).

% Mixed outcome perceptions:

% When we achieve a goal with no drawbacks:
at(N, outcome_overall(X, great)) :-
  at(N, outcome_perception(X, great_for, Goal)),
  at(N, player_goal(Goal)),
  at(N, outcome_stakes(X, Stakes)),
  at(N, goal_stakes(Goal, Stakes)),
  0 = {
    at(N, outcome_perception(X, awful_for, AnyGoal)) :
      at(N, outcome_perception(X, awful_for, AnyGoal)),
      at(N, player_goal(AnyGoal)),
      at(N, goal_stakes(AnyGoal, Stakes));
    at(N, outcome_perception(X, bad_for, AnyGoal)) :
      at(N, outcome_perception(X, bad_for, AnyGoal)),
      at(N, player_goal(AnyGoal)),
      at(N, goal_stakes(AnyGoal, Stakes))
  }.

% When we advance a goal with no drawbacks:
at(N, outcome_overall(X, good)) :-
  at(N, outcome_perception(X, good_for, Goal)),
  at(N, player_goal(Goal)),
  at(N, outcome_stakes(X, Stakes)),
  at(N, goal_stakes(Goal, Stakes)),
  0 = {
    at(N, outcome_perception(X, great_for, AnyGoal)) :
      at(N, outcome_perception(X, great_for, AnyGoal)),
```

```
865       at(N, player_goal(AnyGoal)),
          at(N, goal_stakes(AnyGoal, Stakes));
        at(N, outcome_perception(X, awful_for, AnyGoal)) :
          at(N, outcome_perception(X, awful_for, AnyGoal)),
          at(N, player_goal(AnyGoal)),
870       at(N, goal_stakes(AnyGoal, Stakes));
        at(N, outcome_perception(X, bad_for, AnyGoal)) :
          at(N, outcome_perception(X, bad_for, AnyGoal)),
          at(N, player_goal(AnyGoal)),
          at(N, goal_stakes(AnyGoal, Stakes))
875     }.

    % When we achieve at least one goal but fail at least one other goal,
    % or advance one goal while hindering another:
    at(N, outcome_overall(X, tradeoff)) :-
880   at(N, outcome_stakes(X, Stakes)),
      at(N, outcome_perception(X, great_for, Goal)),
      at(N, player_goal(Goal)),
      at(N, goal_stakes(Goal, Stakes)),
      at(N, outcome_perception(X, awful_for, FailedGoal)),
885   at(N, player_goal(FailedGoal)),
      at(N, goal_stakes(FailedGoal, Stakes)).

    at(N, outcome_overall(X, tradeoff)) :-
      at(N, outcome_stakes(X, Stakes)),
890   at(N, outcome_perception(X, good_for, Goal)),
      at(N, player_goal(Goal)),
      at(N, goal_stakes(Goal, Stakes)),
      at(N, outcome_perception(X, bad_for, HinderedGoal)),
      at(N, player_goal(HinderedGoal)),
895   at(N, goal_stakes(HinderedGoal, Stakes)),
      0 = {
        at(N, outcome_perception(X, great_for, AnyGoal)) :
          at(N, outcome_perception(X, great_for, AnyGoal)),
          at(N, goal_stakes(AnyGoal, Stakes)),
900       at(N, player_goal(AnyGoal));
        at(N, outcome_perception(X, awful_for, AnyGoal)) :
          at(N, outcome_perception(X, awful_for, AnyGoal)),
          at(N, goal_stakes(AnyGoal, Stakes)),
          at(N, player_goal(AnyGoal))
905     }.

    % When we achieve at least one goal but hinder at least one other:
    at(N, outcome_overall(X, worth_it)) :-
      at(N, outcome_stakes(X, Stakes)),
910   at(N, outcome_perception(X, great_for, Goal)),
      at(N, player_goal(Goal)),
      at(N, goal_stakes(Goal, Stakes)),
      at(N, outcome_perception(X, bad_for, HinderedGoal)),
```

— eval.lp —

```
    at(N, player_goal(HinderedGoal)),
    at(N, goal_stakes(HinderedGoal, Stakes)),
    0 = {
      at(N, outcome_perception(X, awful_for, AnyGoal)) :
        at(N, outcome_perception(X, awful_for, AnyGoal)),
        at(N, goal_stakes(AnyGoal, Stakes)),
        at(N, player_goal(AnyGoal))
    }.

  % When we advance at least one goal but fail at least one other:
  at(N, outcome_overall(X, not_worth_it)) :-
    at(N, outcome_stakes(X, Stakes)),
    at(N, outcome_perception(X, good_for, Goal)),
    at(N, player_goal(Goal)),
    at(N, goal_stakes(Goal, Stakes)),
    at(N, outcome_perception(X, awful_for, FailedGoal)),
    at(N, player_goal(FailedGoal)),
    at(N, goal_stakes(FailedGoal, Stakes)),
    0 = {
      at(N, outcome_perception(X, great_for, AnyGoal)) :
        at(N, outcome_perception(X, great_for, AnyGoal)),
        at(N, goal_stakes(AnyGoal, Stakes)),
        at(N, player_goal(AnyGoal))
    }.

  % When we hinder a goal and make no progress otherwise:
  at(N, outcome_overall(X, bad)) :-
    at(N, outcome_perception(X, bad_for, Goal)),
    at(N, player_goal(Goal)),
    at(N, outcome_stakes(X, Stakes)),
    at(N, goal_stakes(Goal, Stakes)),
    0 = {
      at(N, outcome_perception(X, great_for, AnyGoal)) :
        at(N, outcome_perception(X, great_for, AnyGoal)),
        at(N, player_goal(AnyGoal)),
        at(N, goal_stakes(AnyGoal, Stakes));
      at(N, outcome_perception(X, good_for, AnyGoal)) :
        at(N, outcome_perception(X, good_for, AnyGoal)),
        at(N, player_goal(AnyGoal)),
        at(N, goal_stakes(AnyGoal, Stakes));
      at(N, outcome_perception(X, awful_for, AnyGoal)) :
        at(N, outcome_perception(X, awful_for, AnyGoal)),
        at(N, player_goal(AnyGoal)),
        at(N, goal_stakes(AnyGoal, Stakes))
    }.

  % When we fail a goal and make no progress otherwise:
  at(N, outcome_overall(X, awful)) :-
    at(N, outcome_perception(X, awful_for, Goal)),
```

```
    at(N, player_goal(Goal)),
    at(N, outcome_stakes(X, Stakes)),
    at(N, goal_stakes(Goal, Stakes)),
    0 = {
      at(N, outcome_perception(X, great_for, AnyGoal)) :
        at(N, outcome_perception(X, great_for, AnyGoal)),
        at(N, player_goal(AnyGoal)),
        at(N, goal_stakes(AnyGoal, Stakes));
      at(N, outcome_perception(X, good_for, AnyGoal)) :
        at(N, outcome_perception(X, good_for, AnyGoal)),
        at(N, player_goal(AnyGoal)),
        at(N, goal_stakes(AnyGoal, Stakes))
    }.

  % When an outcome doesn't affect max-stakes goals in the end:
  at(N, outcome_overall(option(X), neutral)) :-
    at(N, option(X)),
    at(N, outcome_stakes(X, Stakes)),
    0 = {
      at(N, outcome_perception(option(X), NotIrrelevant, AnyGoal)) :
        at(N, outcome_perception(option(X), NotIrrelevant, AnyGoal)),
        at(N, player_goal(AnyGoal)),
        at(N, goal_stakes(AnyGoal, Stakes)),
        NotIrrelevant != irrelevant
    }.


  % Outcomes affecting non-max-stakes or non-player goals are not
  % considered important:
  at(N, important_outcome(X, Outcome)) :-
    at(N, outcome(X, Outcome)),
    at(N, outcome_stakes(X, Stakes)),
    at(N, goal_stakes(Goal, Stakes)),
    at(N, player_goal(Goal)),
    at(N, possible_outcome_perception(X, Outcome, Any, Goal)).

  % Non-effecacious outcomes that are the alternatives to outcomes that
  % would be important count as important.
  % TODO: Account for exclusion rules here as well to imply importance
  % across multiple outcome variables?
  at(N, important_outcome(X, o(SameVar, Happened))) :-
    at(N, outcome(X, o(SameVar, Happened))),
    at(N, outcome_stakes(X, Stakes)),
    at(N, goal_stakes(Goal, Stakes)),
    at(N, player_goal(Goal)),
    at(
      N,
      possible_outcome_perception(X, o(SameVar, CouldHave), Any, Goal)
    ).
```

— eval.lp —

```
    % Outcomes which are neither likely nor unlikely are labeled as
    % "neutral:"

    at(N, neutral_outcome(X, o(OutVar, OutVal))) :-
      at(N, action(X, Action)),
      outcome_val(Action, OutVar, OutVal),
      not at(N, likely_outcome(X, o(OutVar, OutVal))),
      not at(N, unlikely_outcome(X, o(OutVar, OutVal))).


    % Outcome predictabilities:

    at(N, outcome_predictability(X, o(OutVar, OutVal), predictable)) :-
      story_op(N, build_options),
      at(N, outcome(X, o(OutVar, OutVal))),
      at(N, likely_outcome(X, o(OutVar, OutVal))),
      1 = {
        at(N, likely_outcome(X, o(OutVar, AnyVal))) :
          at(N, likely_outcome(X, o(OutVar, AnyVal)))
      }.

    at(N, outcome_predictability(X, o(OutVar, OutVal), expected)) :-
      story_op(N, build_options),
      at(N, outcome(X, o(OutVar, OutVal))),
      at(N, likely_outcome(X, o(OutVar, OutVal))),
      2 <= {
        at(N, likely_outcome(X, o(OutVar, AnyVal))) :
          at(N, likely_outcome(X, o(OutVar, AnyVal)))
      }.

    at(N, outcome_predictability(X, o(OutVar, OutVal), average)) :-
      story_op(N, build_options),
      at(N, outcome(X, o(OutVar, OutVal))),
      not at(N, unlikely_outcome(X, o(OutVar, OutVal))),
      0 = {
        at(N, likely_outcome(X, o(OutVar, AnyVal))) :
          at(N, likely_outcome(X, o(OutVar, AnyVal)))
      },
      1 <= {
        at(N, unlikely_outcome(X, o(OutVar, AnyVal))) :
          at(N, unlikely_outcome(X, o(OutVar, AnyVal)))
      }.

    at(N, outcome_predictability(X, o(OutVar, OutVal), unpredictable)) :-
      story_op(N, build_options),
      at(N, outcome(X, o(OutVar, OutVal))),
      0 = {
        at(N, likely_outcome(X, o(OutVar, AnyVal))) :
```

— eval.lp —

```
        at(N, likely_outcome(X, o(OutVar, AnyVal)));
      at(N, unlikely_outcome(X, o(OutVar, AnyVal))) :
        at(N, unlikely_outcome(X, o(OutVar, AnyVal)))
    }.

at(N, outcome_predictability(X, o(OutVar, OutVal), unpredictable)) :-
  story_op(N, build_options),
  at(N, action(X, Action)),
  at(N, outcome(X, o(OutVar, OutVal))),
  0 = #sum { % All possible outcome values for this var are "unlikely"
    1, outcome_val(Action, OutVar, AnyVal) :
      outcome_val(Action, OutVar, AnyVal);
    -1, at(N, unlikely_outcome(X, o(OutVar, AnyVal))) :
      at(N, unlikely_outcome(X, o(OutVar, AnyVal)))
  }.

at(N, outcome_predictability(X, o(OutVar, OutVal), unexpected)) :-
  story_op(N, build_options),
  at(N, outcome(X, o(OutVar, OutVal))),
  at(N, neutral_outcome(X, o(OutVar, OutVal))),
  1 <= {
    at(N, likely_outcome(X, o(OutVar, AnyVal))) :
      at(N, likely_outcome(X, o(OutVar, AnyVal)))
  }.

at(N, outcome_predictability(X, o(OutVar, OutVal), unexpected)) :-
  story_op(N, build_options),
  at(N, outcome(X, o(OutVar, OutVal))),
  at(N, unlikely_outcome(X, o(OutVar, OutVal))),
  0 = {
    at(N, likely_outcome(X, o(OutVar, AnyVal))) :
      at(N, likely_outcome(X, o(OutVar, AnyVal)))
  },
  1 <= {
    at(N, neutral_outcome(X, o(OutVar, AnyVal))) :
      at(N, neutral_outcome(X, o(OutVar, AnyVal)))
  }.

at(N, outcome_predictability(X, o(OutVar, OutVal), unfair)) :-
  story_op(N, build_options),
  at(N, outcome(X, o(OutVar, OutVal))),
  at(N, unlikely_outcome(X, o(OutVar, OutVal))),
  at(N, likely_outcome(X, o(OutVar, OtherVal))),
  OutVal != OtherVal.

error(
  m(
    "Individual␣outcome␣has␣multiple␣predictabilities:",
    N, X, o(OutVar, OutVal), P1, P2
```

— eval.lp —

```
    )
) :-
  at(N, outcome_predictability(X, o(OutVar, OutVal), P1)),
  at(N, outcome_predictability(X, o(OutVar, OutVal), P2)),
  P1 < P2.

error(
  m(
    "Individual␣outcome␣has␣no␣predictability:",
    N,
    X,
    o(OutVar, OutVal)
  )
):-
  at(N, outcome(X, o(OutVar, OutVal))),
  0 = {
    at(N, outcome_predictability(X, o(OutVar, OutVal), Any)) :
      at(N, outcome_predictability(X, o(OutVar, OutVal), Any))
  }.

% Combined predictabilities:
at(N, overall_predictability(X, predictable)) :-
  story_op(N, build_options),
  at(N, outcome(X, SomeOutcome)),
  0 = #sum { % All important outcomes for this option are predictable
    1, at(N, outcome(X, Outcome)) :
      at(N, important_outcome(X, Outcome));
    -1, at(N, outcome(X, Outcome)) :
      at(N, important_outcome(X, Outcome)),
      at(N, outcome_predictability(X, Outcome, predictable))
  },
  1 <= {
    at(N, important_outcome(X, Outcome))
  }. % and there's at least one

at(N, overall_predictability(X, expected)) :-
  story_op(N, build_options),
  at(N, outcome(X, SomeOutcome)),
  not at(N, overall_predictability(X, predictable)),
  0 = #sum { % Outcomes for this opt are pred., expctd., or avg.
    1, at(N, outcome(X, Outcome)) :
      at(N, important_outcome(X, Outcome));
    -1, at(N, outcome(X, Outcome)) :
      at(N, important_outcome(X, Outcome)),
      at(N, outcome_predictability(X, Outcome, predictable));
    -1, at(N, outcome(X, Outcome)) :
      at(N, important_outcome(X, Outcome)),
      at(N, outcome_predictability(X, Outcome, expected));
    -1, at(N, outcome(X, Outcome)) :
```

— eval.lp —

```
        at(N, important_outcome(X, Outcome)),
        at(N, outcome_predictability(X, Outcome, average))
    },
    1 <= { % And they're not *all* average
      at(N, outcome(X, Outcome)) :
        at(N, outcome_predictability(X, Outcome, predictable)),
        at(N, important_outcome(X, Outcome));
      at(N, outcome(X, Outcome)) :
        at(N, outcome_predictability(X, Outcome, expected)),
        at(N, important_outcome(X, Outcome))
    }.

at(N, overall_predictability(X, average)) :-
    story_op(N, build_options),
    at(N, outcome(X, SomeOutcome)),
    0 = #sum { % All important outcomes for this option are average
      1, at(N, outcome(X, Outcome)) :
        at(N, important_outcome(X, Outcome));
      -1, at(N, outcome(X, Outcome)) :
        at(N, important_outcome(X, Outcome)),
        at(N, outcome_predictability(X, Outcome, average))
    },
    1 <= {
      at(N, important_outcome(X, Outcome))
    }. % and there's at least one

at(N, overall_predictability(X, unpredictable)) :-
    story_op(N, build_options),
    at(N, important_outcome(X, SomeOutcome)),
    at(N, outcome_predictability(X, SomeOutcome, unpredictable)),
    0 = {
      at(N, outcome(X, Outcome)) :
        at(N, outcome_predictability(X, Outcome, unexpected)),
        at(N, important_outcome(X, Outcome));
      at(N, outcome(X, Outcome)) :
        at(N, outcome_predictability(X, Outcome, unfair)),
        at(N, important_outcome(X, Outcome))
    }.

at(N, overall_predictability(X, unexpected)) :-
    story_op(N, build_options),
    at(N, important_outcome(X, SomeOutcome)),
    at(N, outcome_predictability(X, SomeOutcome, unexpected)),
    0 = {
      at(N, outcome(X, Outcome)) :
        at(N, outcome_predictability(X, Outcome, unfair)),
        at(N, important_outcome(X, Outcome))
    }.
```

— eval.lp —

```
at(N, overall_predictability(X, unfair)) :-
  story_op(N, build_options),
  at(N, important_outcome(X, SomeOutcome)),
  at(N, outcome_predictability(X, SomeOutcome, unfair)).

at(N, overall_predictability(option(X), irrelevant)) :-
  story_op(N, build_options),
  at(N, option(X)),
  0 = {
    at(N, outcome_perception(option(X), SomePerception, SomeGoal)):
     at(N, outcome_perception(option(X), SomePerception, SomeGoal)),
     at(N, player_goal(SomeGoal))
  }.

1 >= {
  at(N, overall_predictability(option(X), unrecognized))
} :-
  story_op(N, build_options),
  at(N, option(X)).

error(
  m("Outcome has multiple overall predictabilities:", N, X, P1, P2)
) :-
  at(N, overall_predictability(X, P1)),
  at(N, overall_predictability(X, P2)),
  P1 < P2.

error(m("Outcome has no overall predictability:", N, option(X))) :-
  at(N, option(X)),
  0 = {
    at(N, overall_predictability(option(X), Any)) :
      at(N, overall_predictability(option(X), Any))
  }.

% Outcome feels:

at(N, outcome_feel(option(X), expected_success)) :-
  story_op(N, build_options),
  at(N, option(X)),
  1 <= {
    at(N, option_feel(option(X), sure_thing));
    at(N, option_feel(option(X), safe));
    at(N, option_feel(option(X), hopeful))
  },
  1 = {
    at(N, overall_predictability(option(X), predictable));
    at(N, overall_predictability(option(X), expected))
  },
  1 <= {
```

```
      at(N, outcome_overall(option(X), great));
      at(N, outcome_overall(option(X), good))
    }.

at(N, outcome_feel(option(X), unfair)) :-
  story_op(N, build_options),
  at(N, option(X)),
  1 <= {
    at(N, option_feel(option(X), sure_thing));
    at(N, option_feel(option(X), safe));
    at(N, option_feel(option(X), hopeful))
  },
  1 = {
    at(N, overall_predictability(option(X), unexpected));
    at(N, overall_predictability(option(X), unfair))
  },
  1 = {
  % TODO: Re-enable this when we've got a better lock on what is
  % "worth it"
  %    at(N, outcome_overall(option(X), not_worth_it));
    at(N, outcome_overall(option(X), bad));
    at(N, outcome_overall(option(X), awful))
  }.

at(N, outcome_feel(option(X), nice_gamble)) :-
  story_op(N, build_options),
  at(N, option(X)),
  1 <= {
    at(N, option_feel(option(X), risky));
    at(N, option_feel(option(X), tradeoff));
    at(N, option_feel(option(X), irrelevant))
  },
  1 = {
    at(N, overall_predictability(option(X), average));
    at(N, overall_predictability(option(X), unpredictable))
  },
  1 = {
    at(N, outcome_overall(option(X), great));
    at(N, outcome_overall(option(X), good));
    at(N, outcome_overall(option(X), worth_it))
  }.

at(N, outcome_feel(option(X), bad_gamble)) :-
  story_op(N, build_options),
  at(N, option(X)),
  1 <= {
    at(N, option_feel(option(X), risky));
    at(N, option_feel(option(X), tradeoff));
    at(N, option_feel(option(X), irrelevant))
```

— eval.lp —

```
    },
    1 = {
      at(N, overall_predictability(option(X), average));
      at(N, overall_predictability(option(X), unpredictable))
    },
    1 = {
      at(N, outcome_overall(option(X), not_worth_it));
      at(N, outcome_overall(option(X), bad));
      at(N, outcome_overall(option(X), awful))
    }.

at(N, outcome_feel(option(X), expected_failure)) :-
    story_op(N, build_options),
    at(N, option(X)),
    1 <= {
      at(N, option_feel(option(X), longshot));
      at(N, option_feel(option(X), bad));
      at(N, option_feel(option(X), doomed))
    },
    1 = {
      at(N, overall_predictability(option(X), predictable));
      at(N, overall_predictability(option(X), expected));
      at(N, overall_predictability(option(X), average))
    },
    1 <= {
    % TODO: Re-enable this when we've got a better lock on what's "worth
    % it"
    %  at(N, outcome_overall(option(X), not_worth_it));
      at(N, outcome_overall(option(X), bad));
      at(N, outcome_overall(option(X), awful))
    }.

at(N, outcome_feel(option(X), miracle)) :-
    story_op(N, build_options),
    at(N, option(X)),
    1 <= {
      at(N, option_feel(option(X), longshot));
      at(N, option_feel(option(X), bad));
      at(N, option_feel(option(X), doomed))
    },
    1 = {
      at(N, overall_predictability(option(X), unexpected));
      at(N, overall_predictability(option(X), unfair))
    },
    1 = {
      at(N, outcome_overall(option(X), great));
      at(N, outcome_overall(option(X), good));
    % TODO: but we're leaving this in for now...
      at(N, outcome_overall(option(X), worth_it))
```

— eval.lp —

```
}.

% TODO: More outcome feels!
```

— goals.lp —

```
% goals.lp
% Rules dealing with player goals.

at(N, player_goal(preserve_health(Member))) :-
  st(N, state(party_member, Member)),
  story_op(N, initialize_node).

at(N, player_goal(avoid_threats_to(Member))) :-
  st(N, state(party_member, Member)),
  story_op(N, initialize_node).

at(N, player_goal(avoid_accusations(Member))) :-
  st(N, state(party_member, Member)),
  story_op(N, initialize_node).

at(N, player_goal(preserve_original_form(Member))) :-
  st(N, state(party_member, Member)),
  story_op(N, initialize_node).

at(N, player_goal(reclaim_property(Member, StolenItem))) :-
  st(N, state(party_member, Member)),
  st(N, relation(stolen_from, Member, StolenItem)),
  story_op(N, initialize_node).

at(N, player_goal(as_intended(Member))) :-
  st(N, state(party_member, Member)),
  story_op(N, initialize_node).

at(N, player_goal(have_tool_for(Member, Skill))) :-
  st(N, state(party_member, Member)),
  st(N, property(has_skill, Member, Skill)),
  story_op(N, initialize_node).

% the player doesn't want others to be threatened or accused as long
% as they're not aggressive:

at(N, guilty(inst(actor, Guilty))) :-
  st(N, relation(threatening, inst(actor, Guilty), Anyone)).
```

— goals.lp —

```
at(N, guilty(inst(actor, Guilty))) :-
  st(
    N,
    relation(accusing, inst(actor, Guilty), inst(actor, Innocent))
  ),
  not at(N, guilty(inst(actor, Innocent))).

at(N, guilty(inst(actor, Guilty))) :-
  st(N, relation(has_item, inst(actor, Guilty), inst(item, Stolen))),
  st(
    N,
    relation(stolen_from, inst(actor, Victim), inst(item, Stolen))
  ).

at(N, player_goal(avoid_threats_to(inst(actor, Innocent)))) :-
  st(N, inst(actor, Innocent)),
  st(N, relation(threatening, Someone, inst(actor, Innocent))),
  not at(N, guilty(inst(actor, Innocent))),
  story_op(N, initialize_node).

%at(N, player_goal(avoid_threats_to(inst(actor, Innocent)))) :-
%  st(N, inst(actor, Innocent)),
%  at(
%    N,
%    consequence(
%      0,
%      relation(threatening, Someone, inst(actor, Innocent))
%    )
%  ),
%  not at(N, guilty(inst(actor, Innocent))),
%  story_op(N, initialize_node).

at(N, player_goal(avoid_accusations(inst(actor, Innocent)))) :-
  st(N, inst(actor, Innocent)),
  st(N, relation(accusing, Someone, inst(actor, Innocent))),
  not at(N, guilty(inst(actor, Innocent))),
  story_op(N, initialize_node).

% TODO: Some way of making this not crash & overload things?
%at(N, player_goal(avoid_accusations(inst(actor, Innocent)))) :-
%  st(N, inst(actor, Innocent)),
%  at(
%    N,
%    consequence(
%      0,
%      relation(accusing, Someone, inst(actor, Innocent))
%    )
%  ),
%  not at(N, guilty(inst(actor, Innocent))),
```

— goals.lp —

```
%  story_op(N, initialize_node).

% TODO: more player goals!
```

— grow.lp —

```
% Nodes statuses
% --------------

% All nodes can be uninitialized, initialized, built, branched, or
% polished.

error(m("Node without status.", N)) :-
  story_node(N),
  0 = {
    node_status(N, uninitialized);
    node_status(N, initialized);
    node_status(N, built);
    node_status(N, branched);
    node_status(N, polished)
  }.

% Story operations allow nodes to reach new statuses:

node_status_reached(N, uninitialized) :- new_node(N).

node_status_reached(N, initialized) :- story_op(N, initialize_node).

node_status_reached(N, built) :- story_op(N, build_options).

node_status_reached(N, branched) :- story_op(N, add_branch_nodes).

node_status_reached(N, polished) :- story_op(N, add_surface).

% A node's actual status is the highest step reached:

node_status(N, uninitialized) :-
  node_status_reached(N, uninitialized),
  0 = {
    node_status_reached(N, initialized);
    node_status_reached(N, built);
    node_status_reached(N, branched);
    node_status_reached(N, polished)
  }.
```

— grow.lp —

```
node_status(N, initialized) :-
  node_status_reached(N, initialized),
  0 = {
    node_status_reached(N, built);
    node_status_reached(N, branched);
    node_status_reached(N, polished)
  }.

node_status(N, built) :-
  node_status_reached(N, built),
  0 = {
    node_status_reached(N, branched);
    node_status_reached(N, polished)
  }.

node_status(N, branched) :-
  node_status_reached(N, branched),
  0 = {
    node_status_reached(N, polished)
  }.

node_status(N, polished) :- node_status_reached(N, polished).

% You can't skip node status steps (but you can do several at once):

error(m("Node␣status␣skipped␣'uninitialized'.", N)) :-
  node_status_reached(N, initialized),
  0 = {
    node_status_reached(N, uninitialized)
  }.

error(m("Node␣status␣skipped␣'initialized'.", N)) :-
  node_status_reached(N, built),
  0 = {
    node_status_reached(N, initialized)
  }.

error(m("Node␣status␣skipped␣'built'.", N)) :-
  node_status_reached(N, branched),
  0 = {
    node_status_reached(N, built)
  }.

error(m("Node␣status␣skipped␣'branched'.", N)) :-
  node_status_reached(N, polished),
  0 = {
    node_status_reached(N, branched)
  }.
```

```
% Some sanity checks
% ------------------

% Choice nodes which have neither inherited potential nor a setup
% cannot reach "initialized":

inherited_potential(N) :-
  story_node(N),
  successor(Prev, Opt, N),
  0 < { unresolved_potential(Prev, Opt, Pt) }.

error(m("Built␣choice␣has␣no␣potential!", N)) :-
  node_status_reached(N, built),
  story_node(N),
  node_type(N, choice),
  0 = {
    setup(N, Setup) : possible_setup(Setup);
    inherited_potential(N)
  }.

% Nodes which have no options cannot reach "built":
optcount(N, X) :-
  story_node(N),
  X = {
    at(N, option(O)) : opt_num(O)
  }.

error(m("Built␣node␣has␣no␣options!", N)) :-
  node_status_reached(N, built),
  optcount(N, 0).

% Every option of a "branched" node must have a successor, unless it's
% an "ending" node:

error(
  m("Branched␣node's␣option␣is␣missing␣a␣successor.", N, option(O))
) :-
  node_status_reached(N, branched),
  at(N, option(O)),
  0 = {
    successor(N, option(O), X) : story_node(X);
    node_type(N, ending)
  }.

% TODO: sanity checking for "polished" nodes?


% Setup implementation
```

— grow.lp —

```
% --------------------

% A setup is needed at the beginning of the story and after every
% travel_onwards event:
1 = {
  setup(R, Setup) : possible_setup(Setup);
  error(m("Root without setup.", R))
} :-
  story_root(R),
  story_op(R, initialize_node).

1 = {
  setup(N, Setup) : possible_setup(Setup);
  error(m("Post-travel node without setup.", Prev, Opt, N))
} :-
  successor(Prev, Opt, N),
  at(Prev, action(Opt, travel_onwards)),
  story_op(N, initialize_node).


% Node ordering
% -------------

% Transitive beforeness of states given a direct "successor" relation:
before(Prev, New) :-
  story_node(Prev),
  story_node(New),
  successor(Prev, Opt, New).

before(Prev, New) :-
  story_node(Prev),
  story_node(Int),
  story_node(New),
  successor(Prev, Opt, Int),
  before(Int, New).

% Succession involves either an existing node that matches the outcome
% state of the branching node, or a new node.
matching_successor(Br, option(Opt), Succ) :-
  story_node(Succ),
  state_match(Br, option(Opt), Succ),
  not before(Succ, Br),
  story_op(Br, add_branch_nodes).

1 = {
  successor(Br, option(Opt), Succ) :
    matching_successor(Br, option(Opt), Succ);
  successor(Br, option(Opt), @pred(@join_(Br, Opt)));
  error(m("Node without successor.", Br, option(Opt)))
```

```
} :-
  story_op(Br, add_branch_nodes),
  at(Br, option(Opt)),
  not node_type(Br, ending).


% New node creation
% -----------------

% New nodes need to be marked as such and filled in:
new_node(@pred(@join_(Br, Opt))) :-
  successor(Br, option(Opt), @pred(@join_(Br, Opt))),
  story_op(Br, add_branch_nodes).

story_node(N) :- new_node(N).

1 = {
  node_type(N, choice);
  node_type(N, event);
  node_type(N, ending);
  error(m("Invalid/missing␣node␣type.", N))
} :-
  story_node(N),
  story_op(N, build_options).

path_length(Next, Shortest+1) :-
  successor(Prev, Opt, Next),
  shortest_path(Prev, Shortest).

shortest_path(N, Shortest) :-
  path_length(N, Shortest),
  0 = {
    path_length(N, Shorter)
      : path_length(N, Shorter), Shorter < Shortest
  }.


% State copying
% -------------

st(New, PSt) :-
  new_node(New),
  successor(Old, Opt, New),
  st(Old, PSt),
  not at(Old, consequence(Opt, _not, PSt)).

st(New, NSt) :-
  new_node(New),
  successor(Old, Opt, New),
```

```
    at(Old, consequence(Opt, NSt)).

error(m("New node picked up extra state", New, State)) :-
  new_node(New),
  successor(Old, Opt, New),
  st(New, State),
  not st(Old, State),
  not at(Old, consequence(Opt, State)),
  not spontaneous(st(New, State)).

error(m("New node lost a state", New, State)) :-
  new_node(New),
  successor(Old, Opt, New),
  st(Old, State),
  not st(New, State),
  not at(Old, consequence(Opt, _not, State)),
  not spontaneous(_not, st(New, State)).

error(m("New node violated negative consequence", New, State)) :-
  new_node(New),
  successor(Old, Opt, New),
  at(Old, consequence(Opt, _not, State)),
  st(New, State).


% State matching
% --------------

% The concept of a "state match":
state_match(Prev, option(Opt), New) :-
  story_op(Prev, add_branch_nodes),
  story_node(New),
  New != Prev,
  at(Prev, option(Opt)),
  not state_mismatch(Prev, option(Opt), New).

% mismatch due to ignored consequence:
state_mismatch(Prev, option(Opt), New) :-
  story_op(Prev, add_branch_nodes),
  story_node(New),
  New != Prev,
  at(Prev, consequence(option(Opt), State)),
  not st(New, State).

% mismatch due to invalidated _not consequence:
state_mismatch(Prev, option(Opt), New) :-
  story_op(Prev, add_branch_nodes),
  at(Prev, option(Opt)),
  story_node(New),
```

```
    New != Prev,
    at(Prev, consequence(option(Opt), _not, State)),
    st(New, State).

  % mismatch due to uncaused state addition:
state_mismatch(Prev, option(Opt), New) :-
    story_op(Prev, add_branch_nodes),
    at(Prev, option(Opt)),
    story_node(New),
    New != Prev,
    st(New, State),
    0 = {
      st(Prev, State);
      at(Prev, consequence(option(Opt), State));
      spontaneous(st(New, State))
    }.

  % mismatch due to uncaused state dissapearance:
  state_mismatch(Prev, option(Opt), New) :-
    story_op(Prev, add_branch_nodes),
    at(Prev, option(Opt)),
    story_node(New),
    New != Prev,
    st(Prev, State),
    0 = {
      st(New, State);
      at(Prev, consequence(option(Opt), _not, State));
      spontaneous(_not, st(New, State))
    }.

% insurance:
  error(m("Option␣has␣state␣mismatch␣with␣successor.", Old, Opt)) :-
    state_mismatch(Old, Opt, New),
    successor(Old, Opt, New).
```

— items.lp —

```
% items.lp
% Special rules for items. See content/items/*.lp for item
% definitions.

% Item definition unpacking:
subclass(Imm, Spec) :-
  item_def(Spec, Imm, Name, Number).

concrete(Spec) :-
  item_def(Spec, Imm, Name, Number).

default_name_for(Class, Name) :-
  item_def(Class, Superclass, Name, Number).

default_number_for(Class, Number) :-
  item_def(Class, Superclass, Name, Number).

% All items are by default neuter:
default_gender_for(Class, neuter) :-
  subclass(item, Class).

% The concept of a 'nonbook tool' is used for setup
nonbook_tool(Tool, Skill) :-
  tool_for(Tool, Skill),
  0 = { item_def(Tool, book, Name, Number) }.

% Universal tool requirements:

at(N, has_tool_for(Actor, Skill)) :-
  st(N, relation(has_item, Actor, Item)),
  is_instance(N, Item, Category),
  tool_for(Category, Skill),
  story_op(N, initialize_node).

% communal property and the concept of "can trade"

at(N, can_trade(Person, Thing)) :-
  st(N, relation(has_item, Person, Thing)),
  story_op(N, initialize_node).

at(N, can_trade(Person, Thing)) :-
  st(N, state(party_member, Person)),
  st(N, state(party_member, OtherMember)),
  st(N, relation(has_item, OtherMember, Thing)),
  story_op(N, initialize_node).
```

— items.lp —

— potential.lp —

```
% potential.lp
% Rules about potential problems and opportunities.

% TODO: How to indicate resolution success?!

% Problematic/opportune states:
% -----------------------------

at(N, potential(PTyp, state(PState, Inst))) :-
  potential(PTyp, state, PState),
  st(N, state(PState, Inst)),
  story_op(N, build_options).

at(N, potential(PTyp, property(PProp, Inst, PVal))) :-
  potential(PTyp, property, PProp, PVal),
  st(N, property(PProp, Inst, PVal)),
  story_op(N, build_options).

at(N, potential(PTyp, relation(PRel, I1, I2))) :-
  potential(PTyp, relation, PRel),
  st(N, relation(PRel, I1, I2)),
  story_op(N, build_options).

% potentials can be made to dissapear in different ways:

resolution_method(resolves).
resolution_method(manifests).
resolution_method(nullifies).

at(N, consequence(Option, _not, PState)) :-
  at(N, consequence(Option, RM, potential(PType, PState))),
  resolution_method(RM).

at(N, consequence_of(Option, Outcome, _not, PState)) :-
  at(
    N,
    consequence_of(
      Option,
      Outcome,
      RM,
      potential(PType, PState)
    )
```

— potential.lp —

```
    ),
    resolution_method(RM).

% certain parties are the initiators of and/or are troubled by
% different potentials.

at(N, initiated_by(potential(PTyp, state(PState, Inst)), Inst)) :-
  at(N, potential(PTyp, state(PState, Inst))),
  initiated_by(PTyp, state, PState, inst).

at(
  N,
  initiated_by(potential(PTyp, property(PProp, Inst, PVal)), Inst)
) :-
  at(N, potential(PTyp, property(PProp, Inst, PVal))),
  initiated_by(PTyp, property, PProp, PVal, inst).

at(
  N,
  initiated_by(potential(PTyp, relation(PRel, From, To)), From)
) :-
  at(N, potential(PTyp, relation(PRel, From, To))),
  initiated_by(PTyp, relation, PRel, from).

at(N, initiated_by(potential(PTyp, relation(PRel, From, To)), To)) :-
  at(N, potential(PTyp, relation(PRel, From, To))),
  initiated_by(PTyp, relation, PRel, to).

at(N, problematic_for(potential(PTyp, state(PState, Inst)), Inst)) :-
  at(N, potential(PTyp, state(PState, Inst))),
  problematic_for(PTyp, state, PState, inst).

at(
  N,
  problematic_for(potential(PTyp, property(PProp, Inst, PVal)), Inst)
) :-
  at(N, potential(PTyp, property(PProp, Inst, PVal))),
  problematic_for(PTyp, property, PProp, PVal, inst).

at(
  N,
  problematic_for(potential(PTyp, relation(PRel, From, To)), From)
) :-
  at(N, potential(PTyp, relation(PRel, From, To))),
  problematic_for(PTyp, relation, PRel, from).

at(
  N,
  problematic_for(potential(PTyp, relation(PRel, From, To)), To)
```

— potential.lp —

```
) :-
  at(N, potential(PTyp, relation(PRel, From, To))),
  problematic_for(PTyp, relation, PRel, to).

% Potentials have categories, including "urgent" and "immediate":
at(N, category(potential(PTyp, state(PState, Inst)), Category)) :-
  at(N, potential(PTyp, state(PState, Inst))),
  pcategory(potential(PTyp, state, PState), Category),
  story_op(N, build_options).

at(
  N,
  category(potential(PTyp, property(PProp, Inst, PVal)), Category)
) :-
  at(N, potential(PTyp, property(PProp, Inst, PVal))),
  pcategory(potential(PTyp, property, PProp, PVal), Category),
  story_op(N, build_options).

at(N, category(potential(PTyp, relation(PRel, I1, I2)), Category)) :-
  at(N, potential(PTyp, relation(PRel, I1, I2))),
  pcategory(potential(PTyp, relation, PRel), Category),
  story_op(N, build_options).

% From immediate problems to mere opportunities some potentials are
% more important than others.
at(N, most_important(Pt)) :-
  at(N, importance(I, Pt)),
  0 = {
    at(N, some_importance(0)) : 0 < I
  }.

at(N, some_importance(I)) :-
  at(N, importance(I, Something)).

at(N, importance(1, potential(problem, Pt))) :-
  at(N, category(potential(problem, Pt), urgent)),
  at(N, category(potential(problem, Pt), immediate)),
  story_op(N, build_options).

at(N, importance(2, potential(problem, Pt))) :-
  at(N, category(potential(problem, Pt), immediate)),
  story_op(N, build_options).

at(N, importance(3, potential(problem, Pt))) :-
  at(N, category(potential(problem, Pt), urgent)),
  story_op(N, build_options).

at(N, importance(4, potential(problem, Pt))) :-
  at(N, potential(problem, Pt)),
```

— potential.lp —

```
    story_op(N, build_options).

at(N, importance(5, potential(opportunity, Pt))) :-
    at(N, category(potential(opportunity, Pt), urgent)),
    at(N, category(potential(opportunity, Pt), immediate)),
    story_op(N, build_options).

at(N, importance(6, potential(opportunity, Pt))) :-
    at(N, category(potential(opportunity, Pt), immediate)),
    story_op(N, build_options).

at(N, importance(7, potential(opportunity, Pt))) :-
    at(N, category(potential(opportunity, Pt), urgent)),
    story_op(N, build_options).

at(N, importance(8, potential(opportunity, Pt))) :-
    at(N, potential(opportunity, Pt)),
    story_op(N, build_options).

% The idea of "unresolved" potential:
unresolved_potential(N, option(O), potential(PTyp, Pt)) :-
    at(N, potential(PTyp, Pt)),
    at(N, option(O)),
    0 = { at(N, consequence(option(O), _not, Pt)) },
    story_op(N, build_options).

unresolved_potential(N, Opt, potential(PTyp, state(PState, Inst))) :-
    at(N, consequence(Opt, state(PState, Inst))),
    potential(PTyp, state, PState),
    story_op(N, build_options).

unresolved_potential(
    N,
    Opt,
    potential(PTyp, property(PProp, Inst, PVal))
) :-
    at(N, consequence(Opt, property(PProp, Inst, PVal))),
    potential(PTyp, property, PProp, PVal),
    story_op(N, build_options).

unresolved_potential(
    N,
    Opt,
    potential(PTyp, relation(PRel, I1, I2))
) :-
    at(N, consequence(Opt, relation(PRel, I1, I2))),
    potential(PTyp, relation, PRel),
    story_op(N, build_options).
```

— potential.lp —

```
% Potential can be hidden if something involved is off-stage:
% TODO: Hide base states/properties/relations instead/as well?
at(N, hidden(potential(PTyp, state, PState))) :-
  potential(PTyp, state, PState),
  st(N, state(PState, Inst)),
  st(N, state(off_stage, Inst)),
  story_op(N, build_options).

at(N, hidden(potential(PTyp, property(PProp, Inst, PVal)))) :-
  potential(PTyp, property, PProp, PVal),
  st(N, property(PProp, Inst, PVal)),
  st(N, state(off_stage, Inst)),
  story_op(N, build_options).

at(N, hidden(potential(PTyp, relation(PRel, I1, I2)))) :-
  potential(PTyp, relation, PRel),
  st(N, relation(PRel, I1, I2)),
  st(N, state(off_stage, I1)),
  story_op(N, build_options).

at(N, hidden(potential(PTyp, relation(PRel, I1, I2)))) :-
  potential(PTyp, relation, PRel),
  st(N, relation(PRel, I1, I2)),
  st(N, state(off_stage, I2)),
  story_op(N, build_options).
```

— settings.lp —

```
% settings.lp
% Rules about different settings.

possible_setting(town).
possible_setting(city).
possible_setting(road).
possible_setting(wilderness).

1 = {
  setting(N, Setting) : possible_setting(Setting);
  error(m("Node without setting.", N))
} :-
  story_node(N),
  story_op(N, initialize_node).

setting(Next, Setting) :-
  setting(N, Setting),
```

— settings.lp —

```
    vignette(N, V),
    vignette(Next, V),
    successor(N, Opt, Next).
```

— setup.lp —

```
% setup.lp
% Rules concerning vignette setup.

% Rules for unpacking and labelling spontaneous states:
% ----------------------------------------------------

st(N, State) :-
  sp_st(N, State),
  story_op(N, initialize_node).

spontaneous(st(N, State)) :-
  sp_st(N, State),
  story_op(N, initialize_node).

% Rules for setup arguments:
% -------------------------

% normal arguments
setup_argument_n(Setup, Arg, Class, 1, 1) :-
  setup_argument(Setup, Arg, Class).

1 >= {
  error(m("Argument␣binding␣failed.", N, Setup, Arg))
} :-
  setup(N, Setup),
  setup_argument_n(Setup, Arg, Class, Min, Max).

Min <= {
  at(N, setup_arg(Arg, Inst)) : is_instance(N, Inst, Class)
} <= Max :-
  setup(N, Setup),
  setup_argument_n(Setup, Arg, Class, Min, Max),
  not error(m("Argument␣binding␣failed.", N, Setup, Arg)),
  story_op(N, initialize_node).

% create arguments

setup_argument_create_n(Setup, Arg, Class, 1, 1) :-
  setup_argument_create(Setup, Arg, Class).
```

— setup.lp —

```
40
   1 >= {
     error(m("Creation␣argument␣binding␣failed.", N, Setup, Arg))
   } :-
     setup(N, Setup),
45   setup_argument_create_n(Setup, Arg, Class, Min, Max).

   Min <= {
     get_unique_key(s(N, sargcreate(Setup, Arg, Class), I))
       : setup_arg_id(I),
50       I <= Max
   } <= Max :-
     setup(N, Setup),
     setup_argument_create_n(Setup, Arg, Class, Min, Max),
     not error(m("Creation␣argument␣binding␣failed.", N, Setup, Arg)),
55   story_op(N, initialize_node).

   at(N, setup_arg(Arg, inst(Type, @inst(Class, K)))) :-
     unique_key(s(N, sargcreate(Setup, Arg, Class), I), K),
     category_for(Class, Type),
60   setup(N, Setup),
     story_op(N, initialize_node).

   sp_st(N, Inst) :-
     setup(N, Setup),
65   at(N, setup_arg(Arg, Inst)),
     setup_argument_create_n(Setup, Arg, Class, Min, Max),
     story_op(N, initialize_node).

   1 = {
70   sp_inst_class(N, Inst, Concrete)
       : concrete_class_of(Class, Concrete);
     error(m("Error␣with␣spontaneous␣instance␣concrete␣class.", N, Inst))
   } :-
     setup(N, Setup),
75   at(N, setup_arg(Arg, Inst)),
     setup_argument_create_n(Setup, Arg, Class, Min, Max),
     story_op(N, initialize_node).

   sp_st(N, property(type, Inst, Class)) :-
80   sp_inst_class(N, Inst, Class).

   % Add class skills:
   0 <= {
     sp_st(N, property(has_skill, Inst, Skill))
85 } <= 1 :-
     sp_inst_class(N, Inst, Class),
     is_instance(N, Inst, Superclass),
     class_skill(Superclass, Skill, sometimes).
```

— setup.lp —

```
sp_st(N, property(has_skill, Inst, Skill)) :-
  sp_inst_class(N, Inst, Class),
  is_instance(N, Inst, Superclass),
  class_skill(Superclass, Skill, always).

% Create class items

1 >= {
  error(m("Creation side-effect binding failed.", N, Setup, Arg))
} :-
  setup(N, Setup),
  setup_argument_create_n(Setup, Arg, Class, Min, Max),
  at(N, setup_arg(Arg, Inst)),
  is_instance(N, Inst, Someclass),
  class_item(Someclass, IClass, Min, Max).

Min <= {
  get_unique_key(s(N, createclassitem(Owner, CIClass), I))
    : setup_arg_id(I),
      concrete_class_of(IClass, CIClass),
      I <= Max
} <= Max :-
  setup(N, Setup),
  setup_argument_create_n(Setup, Arg, Class, Min, Max),
  not error(m("Creation side-effect binding failed.", N, Setup, Arg)),
  at(N, setup_arg(Arg, Owner)),
  is_instance(N, Owner, Someclass),
  class_item(Someclass, IClass, Min, Max),
  story_op(N, initialize_node).

sp_st(N, inst(Type, @inst(CIClass, K))) :-
  unique_key(s(N, createclassitem(Owner, CIClass), I), K),
  category_for(CIClass, Type),
  story_op(N, initialize_node).

sp_st(N, relation(has_item, Owner, inst(Type, @inst(CIClass, K)))) :-
  unique_key(s(N, createclassitem(Owner, CIClass), I), K),
  category_for(CIClass, Type),
  story_op(N, initialize_node).

sp_st(N, property(type, inst(Type, @inst(CIClass, K)), CIClass)) :-
  unique_key(s(N, createclassitem(Owner, CIClass), I), K),
  category_for(CIClass, Type),
  story_op(N, initialize_node).

% Rules for argument substitution:
% --------------------------------
```

— setup.lp —

```
s_st_n(Setup, State, 1) :-
  s_st(Setup, State).

s_st_n(Setup, State, 0) :-
  s_o_st(Setup, State).

% States
L <= { sp_st(N, state(State, Const)) } <= 1 :-
  s_st_n(Setup, state(c(State), c(Const)), L),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, state(State, Val)) } <= 1 :-
  s_st_n(Setup, state(c(State), v(Arg)), L),
  at(N, setup_arg(Arg, Val)),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, state(SVal, Const)) } <= 1 :-
  s_st_n(Setup, state(v(SArg), c(Const)), L),
  at(N, setup_arg(SArg, SVal)),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, state(SVal, Val)) } <= 1 :-
  s_st_n(Setup, state(v(SArg), v(Arg)), L),
  at(N, setup_arg(Arg, Val)),
  at(N, setup_arg(SArg, SVal)),
  setup(N, Setup),
  story_op(N, initialize_node).

% Properties
L <= { sp_st(N, property(Prop, C1, C2)) } <= 1 :-
  s_st_n(Setup, property(c(Prop), c(C1), c(C2)), L),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, property(Prop, Val, Const)) } <= 1 :-
  s_st_n(Setup, property(c(Prop), v(Arg), c(Const)), L),
  at(N, setup_arg(Arg, Val)),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, property(Prop, Const, Val)) } <= 1 :-
  s_st_n(Setup, property(c(Prop), c(Const), v(Arg)), L),
  at(N, setup_arg(Arg, Val)),
  setup(N, Setup),
  story_op(N, initialize_node).
```

— setup.lp —

```
L <= { sp_st(N, property(Prop, V1, V2)) } <= 1 :-
  s_st_n(Setup, property(c(Prop), v(A1), v(A2)), L),
  at(N, setup_arg(A1, V1)),
  at(N, setup_arg(A2, V2)),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, property(PVal, C1, C2)) } <= 1 :-
  s_st_n(Setup, property(v(PArg), c(C1), c(C2)), L),
  at(N, setup_arg(PArg, PVal)),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, property(PVal, Val, Const)) } <= 1 :-
  s_st_n(Setup, property(v(PArg), v(Arg), c(Const)), L),
  at(N, setup_arg(PArg, PVal)),
  at(N, setup_arg(Arg, Val)),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, property(PVal, Const, Val)) } <= 1 :-
  s_st_n(Setup, property(v(PArg), c(Const), v(Arg)), L),
  at(N, setup_arg(PArg, PVal)),
  at(N, setup_arg(Arg, Val)),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, property(PVal, V1, V2)) } <= 1 :-
  s_st_n(Setup, property(v(PArg), v(A1), v(A2)), L),
  at(N, setup_arg(PArg, PVal)),
  at(N, setup_arg(A1, V1)),
  at(N, setup_arg(A2, V2)),
  setup(N, Setup),
  story_op(N, initialize_node).

% Relations
L <= { sp_st(N, relation(Rel, C1, C2)) } <= 1 :-
  s_st_n(Setup, relation(c(Rel), c(C1), c(C2)), L),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, relation(Rel, Val, Const)) } <= 1 :-
  s_st_n(Setup, relation(c(Rel), v(Arg), c(Const)), L),
  at(N, setup_arg(Arg, Val)),
  setup(N, Setup),
  story_op(N, initialize_node).

L <= { sp_st(N, relation(Rel, Const, Val)) } <= 1 :-
  s_st_n(Setup, relation(c(Rel), c(Const), v(Arg)), L),
```

— setup.lp —

```
    at(N, setup_arg(Arg, Val)),
    setup(N, Setup),
    story_op(N, initialize_node).

L <= { sp_st(N, relation(Rel, V1, V2)) } <= 1 :-
    s_st_n(Setup, relation(c(Rel), v(A1), v(A2)), L),
    at(N, setup_arg(A1, V1)),
    at(N, setup_arg(A2, V2)),
    setup(N, Setup),
    story_op(N, initialize_node).

L <= { sp_st(N, relation(RVal, C1, C2)) } <= 1 :-
    s_st_n(Setup, relation(v(RArg), c(C1), c(C2)), L),
    at(N, setup_arg(RArg, RVal)),
    setup(N, Setup),
    story_op(N, initialize_node).

L <= { sp_st(N, relation(RVal, Val, Const)) } <= 1 :-
    s_st_n(Setup, relation(v(RArg), v(Arg), c(Const)), L),
    at(N, setup_arg(RArg, RVal)),
    at(N, setup_arg(Arg, Val)),
    setup(N, Setup),
    story_op(N, initialize_node).

L <= { sp_st(N, relation(RVal, Const, Val)) } <= 1 :-
    s_st_n(Setup, relation(v(RArg), c(Const), v(Arg)), L),
    at(N, setup_arg(RArg, RVal)),
    at(N, setup_arg(Arg, Val)),
    setup(N, Setup),
    story_op(N, initialize_node).

L <= { sp_st(N, relation(RVal, V1, V2)) } <= 1 :-
    s_st_n(Setup, relation(v(RArg), v(A1), v(A2)), L),
    at(N, setup_arg(RArg, RVal)),
    at(N, setup_arg(A1, V1)),
    at(N, setup_arg(A2, V2)),
    setup(N, Setup),
    story_op(N, initialize_node).

% Error checking:
error(m("Negative␣argument␣arity.", Setup, Arg)) :-
    setup_argument_n(Setup, Arg, Class, Min, Max),
    Min < 0.

error(m("Negative␣argument␣arity.", Setup, Arg)) :-
    setup_argument_n(Setup, Arg, Class, Min, Max),
    Max < 0.

error(m("Invalid␣argument␣arity.", Setup, Arg)) :-
```

— setup.lp —

```
    setup_argument_n(Setup, Arg, Class, Min, Max),
    Min > Max.

error(m("Excessive argument arity.", Setup, Arg)) :-
    setup_argument_n(Setup, Arg, Class, Min, Max),
    Min > max_setup_argument_arity.

error(m("Excessive argument arity.", Setup, Arg)) :-
    setup_argument_n(Setup, Arg, Class, Min, Max),
    Max > max_setup_argument_arity.

error(m("Negative argument arity.", Setup, Arg)) :-
    setup_argument_create_n(Setup, Arg, Class, Min, Max),
    Min < 0.

error(m("Negative argument arity.", Setup, Arg)) :-
    setup_argument_create_n(Setup, Arg, Class, Min, Max),
    Max < 0.

error(m("Invalid argument arity.", Setup, Arg)) :-
    setup_argument_create_n(Setup, Arg, Class, Min, Max),
    Min > Max.

error(m("Excessive argument arity.", Setup, Arg)) :-
    setup_argument_create_n(Setup, Arg, Class, Min, Max),
    Min > max_setup_argument_arity.

error(m("Excessive argument arity.", Setup, Arg)) :-
    setup_argument_create_n(Setup, Arg, Class, Min, Max),
    Max > max_setup_argument_arity.
```

— skills.lp —

```
% skills.lp
% Rules about skills and actions.

skill(unintelligent).
skill(monstrous).
restricted_skill(unintelligent).
restricted_skill(monstrous).

skill(tinkering).
skill(fighting).
skill(wilderness_lore).
skill(music).
```

— skills.lp —

```
skill(elocution).
skill(healing).
skill(thievery).
skill(literacy).
%skill(prayer). TODO: Make this useful
skill(acrobatics).
skill(storytelling).
skill(sorcery).
% TODO: sailing?
% TODO: More skills!

skill_name(unintelligent, "unintelligent").
skill_name(monstrous, "monstrous").
skill_name(tinkering, "tinkering").
skill_name(fighting, "fighting").
skill_name(wilderness_lore, "wilderness␣lore").
skill_name(music, "music").
skill_name(elocution, "elocution").
skill_name(healing, "healing").
skill_name(thievery, "thievery").
skill_name(literacy, "literacy").
skill_name(prayer, "prayer").
skill_name(acrobatics, "acrobatics").
skill_name(sorcery, "sorcery").

% Skill link types:
skill_link_type(required, positive, strong).
skill_link_type(promotes, positive, weak).
skill_link_type(avoids, negative, weak).
skill_link_type(contest, complicated).

% Universal skill links:

at(N, skill_link(Skill, Link, NeedsTool, Action, Arg, Outcome)) :-
  skill_link(Skill, Link, NeedsTool, Action, Arg, Outcome),
  story_op(N, build_options).

% Skill link unpacking:

% 'required' skill links depress expectations if a skill and/or tool
% is missing:
at(N, unlikely_outcome(X, Outcome)) :-
  at(N, action(X, Action)),
  at(N, arg(X, Arg, Actor)),
  at(N, skill_link(Skill, required, NeedsTool, Action, Arg, Outcome)),
  not st(N, property(has_skill, Actor, Skill)),
  story_op(N, build_options).

at(N, unlikely_outcome(X, Outcome)) :-
```

— skills.lp —

```
    at(N, action(X, Action)),
    at(N, arg(X, Arg, Actor)),
    at(N, skill_link(Skill, required, tool, Action, Arg, Outcome)),
    st(N, property(has_skill, Actor, Skill)),
    not at(N, has_tool_for(Actor, Skill)),
    story_op(N, build_options).

% 'promotes' as opposed to 'required' links directly increase
% expectations:
at(N, likely_outcome(X, Outcome)) :-
    at(N, action(X, Action)),
    at(N, arg(X, Arg, Actor)),
    at(N, skill_link(Skill, promotes, no_tool, Action, Arg, Outcome)),
    st(N, property(has_skill, Actor, Skill)),
    story_op(N, build_options).

at(N, likely_outcome(X, Outcome)) :-
    at(N, action(X, Action)),
    at(N, arg(X, Arg, Actor)),
    at(N, skill_link(Skill, promotes, tool, Action, Arg, Outcome)),
    st(N, property(has_skill, Actor, Skill)),
    at(N, has_tool_for(Actor, Skill)),
    story_op(N, build_options).

% 'avoids' is the inverse of 'promotes':
at(N, unlikely_outcome(X, Outcome)) :-
    at(N, action(X, Action)),
    at(N, arg(X, Arg, Actor)),
    at(N, skill_link(Skill, avoids, no_tool, Action, Arg, Outcome)),
    st(N, property(has_skill, Actor, Skill)),
    story_op(N, build_options).

at(N, unlikely_outcome(X, Outcome)) :-
    at(N, action(X, Action)),
    at(N, arg(X, Arg, Actor)),
    at(N, skill_link(Skill, avoids, tool, Action, Arg, Outcome)),
    st(N, property(has_skill, Actor, Skill)),
    at(N, has_tool_for(Actor, Skill)),
    story_op(N, build_options).

% 'contest' links using between to name two arguments & compare skill
% levels:
at(N, likely_outcome(X, Win)) :-
    at(N, action(X, Action)),
    at(N, arg(X, One, AOne)),
    at(N, arg(X, Two, ATwo)),
    at(
      N,
      skill_link(
```

— skills.lp —

```
        Skill, contest, Tool,
        Action,
        between(One, Two),
        either(Win, Lose)
    )
  ),
  st(N, property(has_skill, AOne, Skill)),
  not st(N, property(has_skill, ATwo, Skill)),
  story_op(N, build_options).

at(N, unlikely_outcome(X, Lose)) :-
  at(N, action(X, Action)),
  at(N, arg(X, One, AOne)),
  at(N, arg(X, Two, ATwo)),
  at(
    N,
    skill_link(
      Skill, contest, Tool,
      Action,
      between(One, Two),
      either(Win, Lose)
    )
  ),
  st(N, property(has_skill, AOne, Skill)),
  not st(N, property(has_skill, ATwo, Skill)),
  story_op(N, build_options).

at(N, likely_outcome(X, Lose)) :-
  at(N, action(X, Action)),
  at(N, arg(X, One, AOne)),
  at(N, arg(X, Two, ATwo)),
  at(
    N,
    skill_link(
      Skill, contest, Tool,
      Action,
      between(One, Two),
      either(Win, Lose)
    )
  ),
  not st(N, property(has_skill, AOne, Skill)),
  st(N, property(has_skill, ATwo, Skill)),
  story_op(N, build_options).

at(N, unlikely_outcome(X, Win)) :-
  at(N, action(X, Action)),
  at(N, arg(X, One, AOne)),
  at(N, arg(X, Two, ATwo)),
  at(
```

— skills.lp —

```
      N,
      skill_link(
        Skill, contest, Tool,
        Action,
        between(One, Two),
        either(Win, Lose)
      )
    ),
    not st(N, property(has_skill, AOne, Skill)),
    st(N, property(has_skill, ATwo, Skill)),
    story_op(N, build_options).

at(N, likely_outcome(X, Win)) :-
    at(N, action(X, Action)),
    at(N, arg(X, One, AOne)),
    at(N, arg(X, Two, ATwo)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(One, Two),
        either(Win, Lose)
      )
    ),
    not st(N, property(has_skill, AOne, Skill)),
    not st(N, property(has_skill, ATwo, Skill)),
    at(N, has_tool_for(AOne, Skill)),
    not at(N, has_tool_for(ATwo, Skill)),
    story_op(N, build_options).

at(N, unlikely_outcome(X, Lose)) :-
    at(N, action(X, Action)),
    at(N, arg(X, One, AOne)),
    at(N, arg(X, Two, ATwo)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(One, Two),
        either(Win, Lose)
      )
    ),
    not st(N, property(has_skill, AOne, Skill)),
    not st(N, property(has_skill, ATwo, Skill)),
    at(N, has_tool_for(AOne, Skill)),
    not at(N, has_tool_for(ATwo, Skill)),
    story_op(N, build_options).
```

— skills.lp —

```
at(N, unlikely_outcome(X, Win)) :-
  at(N, action(X, Action)),
  at(N, arg(X, One, AOne)),
  at(N, arg(X, Two, ATwo)),
  at(
    N,
    skill_link(
      Skill, contest, tool,
      Action,
      between(One, Two),
      either(Win, Lose)
    )
  ),
  not st(N, property(has_skill, AOne, Skill)),
  not st(N, property(has_skill, ATwo, Skill)),
  not at(N, has_tool_for(AOne, Skill)),
  at(N, has_tool_for(ATwo, Skill)),
  story_op(N, build_options).

at(N, likely_outcome(X, Lose)) :-
  at(N, action(X, Action)),
  at(N, arg(X, One, AOne)),
  at(N, arg(X, Two, ATwo)),
  at(
    N,
    skill_link(
      Skill, contest, tool,
      Action,
      between(One, Two),
      either(Win, Lose)
    )
  ),
  not st(N, property(has_skill, AOne, Skill)),
  not st(N, property(has_skill, ATwo, Skill)),
  not at(N, has_tool_for(AOne, Skill)),
  at(N, has_tool_for(ATwo, Skill)),
  story_op(N, build_options).

at(N, likely_outcome(X, Win)) :-
  at(N, action(X, Action)),
  at(N, arg(X, One, AOne)),
  at(N, arg(X, Two, ATwo)),
  at(
    N,
    skill_link(
      Skill, contest, tool,
      Action,
      between(One, Two),
```

— skills.lp —

```
          either(Win, Lose)
        )
    ),
    st(N, property(has_skill, AOne, Skill)),
    st(N, property(has_skill, ATwo, Skill)),
    at(N, has_tool_for(AOne, Skill)),
    not at(N, has_tool_for(ATwo, Skill)),
    story_op(N, build_options).

at(N, unlikely_outcome(X, Lose)) :-
    at(N, action(X, Action)),
    at(N, arg(X, One, AOne)),
    at(N, arg(X, Two, ATwo)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(One, Two),
        either(Win, Lose)
      )
    ),
    st(N, property(has_skill, AOne, Skill)),
    st(N, property(has_skill, ATwo, Skill)),
    at(N, has_tool_for(AOne, Skill)),
    not at(N, has_tool_for(ATwo, Skill)),
    story_op(N, build_options).

at(N, unlikely_outcome(X, Win)) :-
    at(N, action(X, Action)),
    at(N, arg(X, One, AOne)),
    at(N, arg(X, Two, ATwo)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(One, Two),
        either(Win, Lose)
      )
    ),
    st(N, property(has_skill, AOne, Skill)),
    st(N, property(has_skill, ATwo, Skill)),
    not at(N, has_tool_for(AOne, Skill)),
    at(N, has_tool_for(ATwo, Skill)),
    story_op(N, build_options).

at(N, likely_outcome(X, Lose)) :-
    at(N, action(X, Action)),
```

— skills.lp —

```
    at(N, arg(X, One, AOne)),
    at(N, arg(X, Two, ATwo)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(One, Two),
        either(Win, Lose)
      )
    ),
    st(N, property(has_skill, AOne, Skill)),
    st(N, property(has_skill, ATwo, Skill)),
    not at(N, has_tool_for(AOne, Skill)),
    at(N, has_tool_for(ATwo, Skill)),
    story_op(N, build_options).

  % An outcome value is "likely" if all other possibilities are unlikely
  % and vice versa:
  at(N, likely_outcome(X, o(OutVar, OutVal))) :-
    at(N, action(X, Action)),
    outcome_val(Action, OutVar, OutVal),
    PossibleOutcomes = { outcome_val(Action, OutVar, PossibleVal) },
    PossibleOutcomes - 1 = {
      at(N, unlikely_outcome(X, o(OutVar, UnlikelyVal)))
    },
    not at(N, unlikely_outcome(X, o(OutVar, OutVal))),
    story_op(N, build_options).

  at(N, unlikely_outcome(X, o(OutVar, OutVal))) :-
    at(N, action(X, Action)),
    outcome_val(Action, OutVar, OutVal),
    PossibleOutcomes = { outcome_val(Action, OutVar, PossibleVal) },
    PossibleOutcomes - 1 = {
      at(N, likely_outcome(X, o(OutVar, UnlikelyVal)))
    },
    not at(N, likely_outcome(X, o(OutVar, OutVal))),
    story_op(N, build_options).

  % An outcome is unlikely if it is excluded by a likely outcome:
  at(N, unlikely_outcome(X, Excluded)) :-
    at(N, action(X, Action)),
    outcome_excludes(Action, Likely, Excluded),
    at(N, likely_outcome(X, Likely)).

  % Some predicates that make gleaning a list of applicable skills/tools
  % easy:

  at(N, relevant_skill(X, Actor, has, Skill)) :-
```

— skills.lp —

```
    at(N, action(X, Action)),
    at(N, skill_link(Skill, Link, NeedsTool, Action, Arg, Outcome)),
    skill_link_type(Link, AnyValence, AnyStrength),
    at(N, arg(X, Arg, Actor)),
    st(N, property(has_skill, Actor, Skill)),
    story_op(N, build_options).

at(N, relevant_skill(X, Actor, missing, Skill)) :-
    at(N, action(X, Action)),
    at(N, skill_link(Skill, Link, NeedsTool, Action, Arg, Outcome)),
    skill_link_type(Link, AnyValence, strong),
    at(N, arg(X, Arg, Actor)),
    not st(N, property(has_skill, Actor, Skill)),
    story_op(N, build_options).

at(N, relevant_tool(X, Actor, has, Item)) :-
    at(N, action(X, Action)),
    at(N, relevant_skill(X, Actor, has, Skill)),
    at(N, skill_link(Skill, LinkType, tool, Action, Arg, Outcome)),
    at(N, arg(X, Arg, Actor)),
    st(N, property(has_skill, Actor, Skill)),
    st(N, relation(has_item, Actor, Item)),
    is_instance(N, Item, Category),
    tool_for(Category, Skill),
    story_op(N, build_options).

at(N, relevant_tool(X, Actor, missing, Skill)) :-
    at(N, action(X, Action)),
    at(N, relevant_skill(X, Actor, has, Skill)),
    at(N, skill_link(Skill, LinkType, tool, Action, Arg, Outcome)),
    at(N, arg(X, Arg, Actor)),
    st(N, property(has_skill, Actor, Skill)),
    not at(N, has_tool_for(Actor, Skill)),
    story_op(N, build_options).

at(N, relevant_skill(X, Actor, has, Skill)) :-
    at(N, action(X, Action)),
    at(N, arg(X, AArg, Actor)),
    at(
      N,
      skill_link(
        Skill, contest, Tool,
        Action,
        between(AArg, Other),
        either(Win, Lose)
      )
    ),
    st(N, property(has_skill, Actor, Skill)),
    story_op(N, build_options).
```

— skills.lp —

```
405
    at(N, relevant_skill(X, Actor, missing, Skill)) :-
      at(N, action(X, Action)),
      at(N, arg(X, AArg, Actor)),
      at(
410     N,
        skill_link(
          Skill, contest, Tool,
          Action,
          between(AArg, Other),
415       either(Win, Lose)
        )
      ),
      not st(N, property(has_skill, Actor, Skill)),
      story_op(N, build_options).
420
    at(N, relevant_skill(X, Actor, has, Skill)) :-
      at(N, action(X, Action)),
      at(N, arg(X, AArg, Actor)),
      at(
425     N,
        skill_link(
          Skill, contest, Tool,
          Action,
          between(Other, AArg),
430       either(Win, Lose)
        )
      ),
      st(N, property(has_skill, Actor, Skill)),
      story_op(N, build_options).
435
    at(N, relevant_skill(X, Actor, missing, Skill)) :-
      at(N, action(X, Action)),
      at(N, arg(X, AArg, Actor)),
      at(
440     N,
        skill_link(
          Skill, contest, Tool,
          Action,
          between(Other, AArg),
445       either(Win, Lose)
        )
      ),
      not st(N, property(has_skill, Actor, Skill)),
      story_op(N, build_options).
450
    at(N, relevant_tool(X, Actor, has, Item)) :-
      at(N, action(X, Action)),
      at(N, arg(X, AArg, Actor)),
```

— skills.lp —

```
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(AArg, Other),
        either(Win, Lose)
      )
    ),
    at(N, relevant_skill(X, Actor, has, Skill)),
    st(N, property(has_skill, Actor, Skill)),
    st(N, relation(has_item, Actor, Item)),
    is_instance(N, Item, Category),
    tool_for(Category, Skill),
    story_op(N, build_options).

at(N, relevant_tool(X, Actor, missing, Skill)) :-
    at(N, action(X, Action)),
    at(N, arg(X, AArg, Actor)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(AArg, Other),
        either(Win, Lose)
      )
    ),
    at(N, relevant_skill(X, Actor, has, Skill)),
    st(N, property(has_skill, Actor, Skill)),
    not at(N, has_tool_for(Actor, Skill)),
    story_op(N, build_options).

at(N, relevant_tool(X, Actor, has, Item)) :-
    at(N, action(X, Action)),
    at(N, arg(X, AArg, Actor)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(Other, AArg),
        either(Win, Lose)
      )
    ),
    at(N, relevant_skill(X, Actor, has, Skill)),
    st(N, property(has_skill, Actor, Skill)),
    st(N, relation(has_item, Actor, Item)),
    is_instance(N, Item, Category),
```

— skills.lp —

```
    tool_for(Category, Skill),
    story_op(N, build_options).

at(N, relevant_tool(X, Actor, missing, Skill)) :-
    at(N, action(X, Action)),
    at(N, arg(X, AArg, Actor)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(Other, AArg),
        either(Win, Lose)
      )
    ),
    at(N, relevant_skill(X, Actor, has, Skill)),
    st(N, property(has_skill, Actor, Skill)),
    not at(N, has_tool_for(Actor, Skill)),
    story_op(N, build_options).

at(N, relevant_tool(X, Actor, has, Item)) :-
    at(N, action(X, Action)),
    at(N, arg(X, AArg, Actor)),
    at(N, arg(X, OArg, Other)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(AArg, OArg),
        either(Win, Lose)
      )
    ),
    at(N, relevant_skill(X, Actor, has, Skill)),
    not st(N, property(has_skill, Actor, Skill)),
    not st(N, property(has_skill, Other, Skill)),
    st(N, relation(has_item, Actor, Item)),
    is_instance(N, Item, Category),
    tool_for(Category, Skill),
    story_op(N, build_options).

at(N, relevant_tool(X, Actor, missing, Skill)) :-
    at(N, action(X, Action)),
    at(N, arg(X, AArg, Actor)),
    at(N, arg(X, OArg, Other)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
```

— skills.lp —

```
        Action,
        between(AArg, OArg),
        either(Win, Lose)
      )
    ),
    at(N, relevant_skill(X, Actor, has, Skill)),
    not st(N, property(has_skill, Actor, Skill)),
    not st(N, property(has_skill, Other, Skill)),
    not at(N, has_tool_for(Actor, Skill)),
    story_op(N, build_options).

at(N, relevant_tool(X, Actor, has, Item)) :-
    at(N, action(X, Action)),
    at(N, arg(X, AArg, Actor)),
    at(N, arg(X, OArg, Other)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(OArg, AArg),
        either(Win, Lose)
      )
    ),
    at(N, relevant_skill(X, Actor, has, Skill)),
    not st(N, property(has_skill, Actor, Skill)),
    not st(N, property(has_skill, Other, Skill)),
    st(N, relation(has_item, Actor, Item)),
    is_instance(N, Item, Category),
    tool_for(Category, Skill),
    story_op(N, build_options).

at(N, relevant_tool(X, Actor, missing, Skill)) :-
    at(N, action(X, Action)),
    at(N, arg(X, AArg, Actor)),
    at(N, arg(X, OArg, Other)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Action,
        between(OArg, AArg),
        either(Win, Lose)
      )
    ),
    at(N, relevant_skill(X, Actor, has, Skill)),
    not st(N, property(has_skill, Actor, Skill)),
    not st(N, property(has_skill, Other, Skill)),
    not at(N, has_tool_for(Actor, Skill)),
```

— skills.lp —

```
    story_op(N, build_options).
```

— surface.lp —

```
% surface.lp
% Rules about surface text realization

% Gender values:
gender(masculine).
gender(feminine).
gender(neuter).

number(singular).
number(plural).

% Default values for name, person, number, gender, and determination:

1 = {
  surface_property(name, inst(Type, ID), Name)
    : surface_property(name, inst(Type, ID), Name);
  surface_property(name, inst(Type, ID), Name)
    : default_name_for(Class, Name),
      st(N, property(type, inst(Type, ID), Class));
  error(m("Error␣with␣instance␣name.", inst(Type, ID)))
} :-
  st(N, inst(Type, ID)).

1 = {
  surface_property(person, inst(Type, ID), Name)
    : surface_property(person, inst(Type, ID), Name);
  surface_property(person, inst(Type, ID), third);
  error(m("Error␣with␣instance␣person.", inst(Type, ID)))
} :-
  st(N, inst(Type, ID)).

1 = {
  surface_property(number, inst(Type, ID), Number)
    : surface_property(number, inst(Type, ID), Number);
  surface_property(number, inst(Type, ID), Number)
    : default_number_for(Class, Number),
      st(N, property(type, inst(Type, ID), Class));
  error(m("Error␣with␣instance␣number.", inst(Type, ID)))
} :-
  st(N, inst(Type, ID)).
```

— surface.lp —

```
1 = {
  surface_property(gender, inst(Type, ID), Gender)
    : surface_property(gender, inst(Type, ID), Gender);
  surface_property(gender, inst(Type, ID), Gender)
    : default_gender_for(Class, Gender),
      st(N, property(type, inst(Type, ID), Class));
  error(m("Error with spontaneous instance gender.", inst(Type, ID)))
} :-
  st(N, inst(Type, ID)).

1 = {
  surface_property(determined, inst(Type, ID), Det) :
    surface_property(determined, inst(Type, ID), Det);
  surface_property(determined, inst(Type, ID), true);
  error(m("Error with instance determination.", inst(Type, ID)))
} :-
  st(N, inst(Type, ID)).
```

— the_party.lp —

```
% the_party.lp
% Rules about the player's party.

% Party Setup:
% ------------

_sr(R) :- story_root(R), story_op(R, initialize_node).

% Core setup:

starting_skill(tinkering).
starting_skill(wilderness_lore).
starting_skill(music).
starting_skill(elocution).
starting_skill(healing).
starting_skill(thievery).
%starting_skill(prayer).
starting_skill(acrobatics).
starting_skill(storytelling).
starting_skill(fighting).
starting_skill(sorcery).

st(R, inst(actor, you)) :- _sr(R).
st(R, state(party_member, inst(actor, you))) :- _sr(R).
st(R, property(type, inst(actor, you), person)) :- _sr(R).
```

— the_party.lp —

```
surface_property(name, inst(actor, you), "Dunyazad").
surface_property(person, inst(actor, you), second).
surface_property(gender, inst(actor, you), feminine).
surface_property(number, inst(actor, you), singular).
surface_property(determined, inst(actor, you), false).
% You start with literacy and three other random skills (from the list
% of "starting" skills):
3 = {
  st(R, property(has_skill, inst(actor, you), Skill))
    : starting_skill(Skill);
  error(m("Missing␣starting␣skill."))
} :- _sr(R).
st(R, property(has_skill, inst(actor, you), literacy)) :- _sr(R).
%st(R, property(has_skill, inst(actor, you), storytelling)) :- _sr(R).

% Party member parameters:

party_member_name("Arel").
party_member_name("Jain").
party_member_name("Frodde").
party_member_name("Zair").
party_member_name("Estok").
party_member_name("Ime").

party_member_rel("friend").
party_member_rel("cousin").
party_member_rel("servant").
party_member_rel("teacher").
party_member_rel("student").
party_member_rel("beloved").

% Party member selection:

1 = {
  party_size(PS) : party_size_value(PS)
} :- _sr(R).

1 = {
  party_member(member_one, Relation) : party_member_rel(Relation);
  error(m("Party␣member␣of␣no␣relation.", member_one))
} :-
  _sr(R),
  party_size(PS),
  PS >= 2.
1 = {
  party_member(member_two, Relation) : party_member_rel(Relation);
  error(m("Party␣member␣of␣no␣relation.", member_two))
} :-
```

— the_party.lp —

```
    _sr(R),
    party_size(PS),
    PS >= 3.
error(m("Can't␣handle␣party␣sizes␣larger␣than␣3.")) :-
    party_size(PS),
    PS > 3.
error(m("Multiple␣party␣members␣have␣the␣same␣relation.")) :-
    party_member(M1, Relation),
    party_member(M2, Relation),
    M1 != M2.
error(m("Polyamory␣is␣too␣complciated␣to␣handle.")) :-
    2 <= { party_member(M, "beloved") : party_member(M, R) }.
error(m("Student␣and␣master␣both.")) :-
    party_member(M1, "teacher"),
    party_member(M2, "student").

% Party member unpacking:

st(R, inst(actor, Member)) :- party_member(Member, Relation), _sr(R).

st(R, state(party_member, inst(actor, Member))) :-
    party_member(Member, Relation),
    _sr(R).

1 = {
    surface_property(name, inst(actor, Member), Name)
      : party_member_name(Name);
    error(m("Party␣member␣without␣name.", Member))
} :- party_member(Member, Relation).

error(m("Multiple␣party␣members␣have␣the␣same␣name.")) :-
    surface_property(name, inst(actor, M1), Name),
    surface_property(name, inst(actor, M2), Name),
    M1 != M2,
    _sr(R).

1 = {
    surface_property(gender, inst(actor, Member), masculine);
    surface_property(gender, inst(actor, Member), feminine);
    error(m("Party␣member␣without␣gender.", Member))
} :- party_member(Member, Relation), _sr(R).

surface_property(determined, inst(actor, Member), false) :-
    party_member(Member, Relation).

st(R, property(relationship, inst(actor, Member), Relation)) :-
    party_member(Member, Relation), _sr(R).

st(R, property(type, inst(actor, Member), person)) :-
```

— the_party.lp —

```
    party_member(Member, Relation),
    _sr(R).

  % Each party member starts with 2 skills:
  2 = {
    st(R, property(has_skill, inst(actor, Member), Skill))
      : skill(Skill),
        not restricted_skill(Skill);
    error(m("Party member missing starting skill 1.", Member));
    error(m("Party member missing starting skill 2.", Member))
  } :- party_member(Member, Relation), _sr(R).

  error(m("Party member initial skill overlap.", Skill)) :-
    skill(Skill),
    st(R, property(has_skill, inst(actor, M1), Skill)),
    st(R, property(has_skill, inst(actor, M2), Skill)),
    st(R, state(party_member, inst(actor, M1))),
    st(R, state(party_member, inst(actor, M2))),
    M1 != M2,
    _sr(R).

  % Supply unpacking:

  get_unique_key(s(supplies(Owner, Item))) :-
    supplies(Owner, Item), _sr(R).
  supply_inst(R, Owner, inst(item, @inst(Item, K)), Item) :-
    unique_key(s(supplies(Owner, Item)), K),
    supplies(Owner, Item),
    _sr(R).
  st(R, Inst) :-
    supply_inst(R, Owner, Inst, Type).
  st(R, property(type, Inst, Type)) :-
    supply_inst(R, Owner, Inst, Type).
  surface_property(name, Inst, Name) :-
    supply_inst(R, Owner, Inst, Type),
    item_def(Type, GType, Name, Number).
  surface_property(number, Inst, Number) :-
    supply_inst(R, Owner, Inst, Type),
    item_def(Type, GType, Name, Number).
  st(R, relation(has_item, inst(actor, Owner), Inst)) :-
    supply_inst(R, Owner, Inst, Type).

  % Supply details

  % You start with some basic items:
  1 = {
    supplies(you, Treasure) :
      item_def(Treasure, treasure, Name, Number)
  } :- _sr(R).
```

— the_party.lp —

```
1 >= {
  supplies(you, Charm) :
    item_def(Charm, charm, Name, Number)
} :- _sr(R).

1 = {
  supplies(you, Book) :
    item_def(Book, book, Name, Number);
  error(m("No starting book."))
} :- _sr(R).

% You can start with a tool for one of your skills:
0 <= {
  supplies(you, Tool) :
    nonbook_tool(Tool, Skill),
    st(R, property(has_skill, inst(actor, you), Skill)),
    _sr(R)
} <= 2.

% Your party members start with non-book tools for their skills:
1 = {
  supplies(Member, Tool) :
    nonbook_tool(Tool, Skill),
    item_def(Tool, Type, Name, Number);
  error(m("Member didn't start with tool for skill.", Member, Skill))
} :-
  1 <= { nonbook_tool(Tool, Skill) },
  party_member(Member, Relation),
  st(R, property(has_skill, inst(actor, Member), Skill)),
  _sr(R).

% Your literate party members might also start with some random books:
2 >= {
  supplies(Member, Book) :
    party_member(Member, Relation),
    st(R, property(has_skill, inst(actor, Member), literacy)),
    item_def(Book, book, BName, BNumber)
} :-
  _sr(R).
```

— vignettes.lp —

```
% vignettes.lp
% Rules about vignette flow.

% Setups define the start of a new vignette:
vignette(N, N) :-
  setup(N, Setup),
  story_op(N, initialize_node).
vignette(N, PrV) :-
  0 = { setup(N, Setup) : possible_setup(Setup) },
  successor(Prev, Opt, N),
  vignette(Prev, PrV),
  story_op(N, initialize_node).

% An option which resolves the all remaining potentials at a given
% node resolves that node's vignette, since a new vignette will be
% needed to start to reintroduce potential. Travel onwards is an
% exception, as it happens between vignettes.
resolves_vignette(N, option(O)) :-
  story_node(N),
  at(N, option(O)),
  0 = {
    unresolved_potential(N, option(O), Pt)
      : unresolved_potential(N, option(O), Pt),
        not at(N, hidden(Pt)),
        not at(N, category(Pt, persistent))
  },
  not at(N, action(option(O), travel_onwards)),
  story_op(N, build_options).

% The first node of a new vignette should always be a choice node:
error(m("Vignette␣starts␣with␣an␣event!")) :-
  vignette(N, N),
  not node_type(N, choice).
```

B.3 Goal Definitions

These files, from the `content/` directory, each specify how a single goal works. Note the custom code in the files `g-as_intended.lp`, `g-avoid_accusations.lp`, and `g-avoid_threats_to.lp`. These three goals extend the normal state-based goal definition format a bit by directly specifying extra conditions for certain

"expecatation" and/or "outcome_perception" predicates.

— g-as_intended.lp —

```
% as intended

at(N, goal_stakes(as_intended(Actor), low)) :-
  at(N, initiator(X, Actor)).

at(N, expectation(X, advances, as_intended(Actor))) :-
  at(N, initiator(X, Actor)),
  at(N, action(X, Action)),
  default_intent(Action, Outcome),
  at(N, likely_outcome(X, Outcome)).

at(N, expectation(X, hinders, as_intended(Actor))) :-
  at(N, initiator(X, Actor)),
  at(N, action(X, Action)),
  default_intent(Action, Outcome),
  at(N, unlikely_outcome(X, Outcome)).

at(N, outcome_perception(X, great_for, as_intended(Actor))) :-
  at(N, initiator(X, Actor)),
  at(N, action(X, Action)),
  default_intent(Action, Outcome),
  at(N, outcome(X, Outcome)).

at(N, outcome_perception(X, awful_for, as_intended(Actor))) :-
  at(N, initiator(X, Actor)),
  at(N, action(X, Action)),
  default_intent(Action, o(OutVar, OutVal)),
  not at(N, outcome(X, o(OutVar, OutVal))).
```

— —

— g-avoid_accusations.lp —

```
% avoid accusations

at(N, goal_stakes(avoid_accusations(inst(actor, ID)), high)) :-
  st(N, inst(actor, ID)).

state_hinders(
  avoid_accusations(inst(actor, Key)),
  relation(accusing, inst(actor, ThreatKey), inst(actor, Key))
) :-
  st(N, inst(actor, Key)),
  st(N, inst(actor, ThreatKey)).

% If there's an accusation, it being likely to persist seems
% dangerous:
at(
  N,
  expectation(X, hinders, avoid_accusations(inst(actor, Victim)))
) :-
  at(N, action(X, Action)),
  st(N, relation(accusing, Threat, inst(actor, Victim))),
  0 = { % No likely or neutral outcome will resolve the threat:
    at(N, likely_outcome(X, Likely)) :
      at(N, likely_outcome(X, Likely)),
      at(
        N,
        consequence_of(
          X,
          Likely,
          _not,
          relation(accusing, Threat, inst(actor, Victim))
        )
      );
    at(N, neutral_outcome(X, Neutral)) :
      at(N, neutral_outcome(X, Neutral)),
      at(
        N,
        consequence_of(
          X,
          Neutral,
          _not,
          relation(accusing, Threat, inst(actor, Victim))
        )
      )
  }.

% If there's an accusation, just letting it persist is enough to
```

— g-avoid_accusations.lp —

```
  % hinder the goal of avoiding it:
  at(
    N,
    outcome_perception(
      option(X),
      bad_for,
      avoid_accusations(inst(actor, Key))
    )
  ) :-
    at(N, option(X)),
    st(N, relation(accusing, Threat, inst(actor, Key))),
    0 = {
      at(N, outcome(option(X), Happened)) :
        at(N, outcome(option(X), Happened)),
        at(
          N,
          consequence_of(
            option(X),
            Happened,
            _not,
            relation(accusing, Threat, inst(actor, Key))
          )
        )
    }.
```

— g-avoid_threats_to.lp —

```
% avoid threats to

at(N, goal_stakes(avoid_threats_to(inst(actor, ID)), high)) :-
  st(N, inst(actor, ID)).

state_hinders(
  avoid_threats_to(inst(actor, Key)),
  relation(threatening, inst(actor, ThreatKey), inst(actor, Key))
) :-
  st(N, inst(actor, Key)),
  st(N, inst(actor, ThreatKey)).

% If there's a threat, it being likely to persist seems dangerous:
at(
  N,
  expectation(X, hinders, avoid_threats_to(inst(actor, Victim)))
) :-
  at(N, action(X, Action)),
```

— g-avoid_threats_to.lp —

```
  st(N, relation(threatening, Threat, inst(actor, Victim))),
  0 = { % No likely or neutral outcome will resolve the threat:
    at(N, likely_outcome(X, Likely)) :
      at(N, likely_outcome(X, Likely)),
      at(
        N,
        consequence_of(
          X,
          Likely,
          _not,
          relation(threatening, Threat, inst(actor, Victim))
        )
      );
    at(N, neutral_outcome(X, Neutral)) :
      at(N, neutral_outcome(X, Neutral)),
      at(
        N,
        consequence_of(
          X,
          Neutral,
          _not,
          relation(threatening, Threat, inst(actor, Victim))
        )
      )
  }.

% If there's a threat, just letting it persist is enough to hinder the
% goal of avoiding it:
at(
  N,
  outcome_perception(
    option(X),
    bad_for,
    avoid_threats_to(inst(actor, Key))
  )
) :-
  at(N, option(X)),
  st(N, relation(threatening, Threat, inst(actor, Key))),
  0 = {
    at(N, outcome(option(X), Happened)) :
      at(N, outcome(option(X), Happened)),
      at(
        N,
        consequence_of(
          option(X),
          Happened,
          _not,
          relation(threatening, Threat, inst(actor, Key))
        )
```

— g-avoid_threats_to.lp —

```
        )
    }.
```

— g-have_tool_for.lp —

```
% have tool for

% Note this is probably the most opaque goal...

at(N, goal_stakes(have_tool_for(inst(actor, ID), Skill), low)) :-
  st(N, inst(actor, ID)),
  st(N, property(has_skill, inst(actor, ID), Skill)).

state_achieves(
  have_tool_for(inst(actor, Actor), Skill),
  relation(has_item, inst(actor, Actor), inst(item, Item))
) :-
  st(N, inst(actor, Actor)),
  st(N, property(has_skill, inst(actor, Actor), Skill)),
  st(N, inst(item, Item)),
  is_instance(N, inst(item, Item), Category),
  tool_for(Category, Skill).
```

— g-preserve_health.lp —

```
% preserve health

at(N, goal_stakes(preserve_health(inst(actor, ID)), high)) :-
  st(N, inst(actor, ID)).

state_fails(
  preserve_health(inst(actor, Key)),
  state(injured, inst(actor, Key))
) :-
  st(N, inst(actor, Key)).

state_fails(
  preserve_health(inst(actor, Key)),
  state(killed, inst(actor, Key))
) :-
  st(N, inst(actor, Key)).
```

— g-preserve_health.lp —

— g-preserve_original_form.lp —

```
% preserve original form

at(N, goal_stakes(preserve_original_form(inst(actor, ID)), high)) :-
  st(N, inst(actor, ID)).

state_fails(
  preserve_original_form(inst(actor, Key)),
  property(polymorphed, inst(actor, Key), Any)
) :-
  st(N, inst(actor, Key)),
  any_class(Any).
```

— g-reclaim_property.lp —

```
% reclaim property

% This is borderline high vs. low stakes...

at(N, goal_stakes(reclaim_property(Victim, Item), high)) :-
  at(N, potential(problem, relation(stolen_from, Victim, Item))).

state_achieves(
  reclaim_property(Victim, Item),
  relation(has_item, Victim, Item)
) :-
  at(N, potential(problem, relation(stolen_from, Victim, Item))).
```

B.4 Action Definitions

These files, also from the `content/` directory, each define a single action. Table 6.1 provides a quick overview of the actions repertoire; note that the "`arrive`," "`leave`," and "`pursue`" actions are not currently enabled by any setups, and to speed up results they have been entirely commented out for now. Action definitions are

responsible for defining both individual outcome components and the skills and/or tools that make those outcomes more or less likely. They can define the consequences of individual outcome components conditionally with respect to the world state when an action occurs, although important conditional effects of an action should be reified as outcome variables to allow the system to reason about player perceptions.

— a-accuse.lp —

```
% accuse
action(accuse).

% arguments

argument(accuse, accuser, actor).
argument(accuse, target, actor).
initiator(accuse, accuser).
default_intent(accuse, o(success, accused)).

% outcomes

outcome_val(accuse, success, accused).
outcome_val(accuse, success, ignored).

% skills:

skill_link(
  elocution, contest, no_tool,
  accuse,
  between(accuser, target),
  either(o(success, accused), o(success, ignored))
).

% accuseing if successful develops an "accusing" relation

at(
  N,
  consequence_of(
    X,
    o(success, accused),
    relation(accusing, Accuser, Target)
  )
```

— a-accuse.lp —

```
) :-
  at(N, action(X, accuse)),
  at(N, arg(X, accuser, Accuser)),
  at(N, arg(X, target, Target)),
  story_op(N, build_options).
```

— a-arrive.lp —

```
%% arrive
%action(arrive).
%
%% arguments
%
%argument(arrive, subject, actor).
%off_stage_okay(arrive, subject).
%initiator(arrive, subject).
%default_intent(arrive, o(result, arrived)).
%
%% outcomes
%
%% It doesn't make sense for arriving to fail, so there's only one
%% outcome: outcome_val(arrive, result, arrived).
%
%% skills
%
%% none
%
%% (hack) prohibit threatened actors from moving back on-stage:
%% TODO: More general mechanism for this kind of thing?
%
%error(m("Threatened actor arrived.", N, X)) :-
%  at(N, action(X, arrive)),
%  at(N, arg(X, subject, Subject)),
%  st(N, relation(threatening, Someone, Subject)),
%  story_op(N, build_options).
%
%% The subject must start off-stage and arrives on-stage:
%
%error(m("On-stage subject arrived.", N, X)) :-
%  at(N, action(X, arrive)),
%  at(N, arg(X, subject, Subject)),
%  not st(N, state(off_stage, Subject)),
%  story_op(N, build_options).
%
%at(
```

— a-arrive.lp —

```
%  N,
%  consequence_of(
%    o(result,
%    arrived), _not,
%    state(off_stage, Subject)
%  )
%) :-
%  at(N, action(X, arrive)),
%  at(N, arg(X, subject, Subject)),
%  story_op(N, build_options).
%
%at(
%  N,
%  consequence_of(
%    o(result,
%    arrived), _not,
%    state(off_stage, Item)
%  )
%) :-
%  at(N, action(X, arrive)),
%  at(N, arg(X, subject, Subject)),
%  st(N, relation(has_item, Subject, Item)),
%  story_op(N, build_options).
```

— a-attack.lp —

```
% attack
action(attack).

chaotic(attack).

% arguments

argument(attack, aggressor, actor).
argument(attack, target, actor).
initiator(attack, aggressor).
default_intent(attack, o(success, victory)).

% outcomes

outcome_val(attack, aggressor_state, unharmed).
outcome_val(attack, aggressor_state, injured).
outcome_val(attack, aggressor_state, killed).

outcome_val(attack, target_state, unharmed).
```

— a-attack.lp —

```
outcome_val(attack, target_state, injured).
outcome_val(attack, target_state, killed).

outcome_val(attack, get_loot, loot).
outcome_val(attack, get_loot, nothing).

outcome_val(attack, success, victory).
outcome_val(attack, success, defeat).
outcome_val(attack, success, tie).

outcome_excludes(
  attack,
  o(success, victory),
  o(aggressor_state, killed)
).
outcome_excludes(
  attack,
  o(success, defeat),
  o(target_state, killed)
).

outcome_excludes(attack, o(success, tie), o(aggressor_state, killed)).

outcome_excludes(attack, o(success, tie), o(get_loot, loot)).
outcome_excludes(attack, o(success, defeat), o(get_loot, loot)).

outcome_excludes(
  attack,
  o(success, tie),
  o(target_state, killed)
).

outcome_excludes(
  attack,
  o(aggressor_state, unharmed),
  o(target_state, unharmed)
).

outcome_excludes(
  attack,
  o(target_state, unharmed),
  o(aggressor_state, unharmed)
).

% skills:

skill_link(
  fighting, contest, tool,
  attack,
```

— a-attack.lp —

```
    between(aggressor, target),
    either(o(aggressor_state, unharmed), o(aggressor_state, injured))
).

skill_link(
    fighting, contest, tool,
    attack,
    between(aggressor, target),
    either(o(target_state, injured), o(target_state, unharmed))
).

skill_link(
    fighting, contest, tool,
    attack,
    between(aggressor, target),
    either(o(success, victory), o(success, defeat))
).

% attacking is one way to deal with a threat

at(
    N,
    consequence_of(
        X,
        o(success, victory),
        resolves,
        potential(problem, relation(threatening, Target, Someone))
    )
) :-
    at(N, action(X, attack)),
    at(N, arg(X, target, Target)),
    st(N, relation(threatening, Target, Someone)),
    story_op(N, build_options).

% no matter the outcome, an attack that winds up injuring a threat
% resolves the threat

at(
    N,
    consequence_of(
        X,
        o(aggressor_state, injured),
        resolves,
        potential(problem, relation(threatening, Aggressor, Someone))
    )
) :-
    at(N, action(X, attack)),
    at(N, arg(X, aggressor, Aggressor)),
    st(N, relation(threatening, Aggressor, Someone)),
```

— a-attack.lp —

```
    story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(aggressor_state, killed),
    resolves,
    potential(problem, relation(threatening, Aggressor, Someone))
  )
) :-
  at(N, action(X, attack)),
  at(N, arg(X, aggressor, Aggressor)),
  st(N, relation(threatening, Aggressor, Someone)),
  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(target_state, injured),
    resolves,
    potential(problem, relation(threatening, Target, Someone))
  )
) :-
  at(N, action(X, attack)),
  at(N, arg(X, target, Target)),
  st(N, relation(threatening, Target, Someone)),
  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(target_state, killed),
    resolves,
    potential(problem, relation(threatening, Target, Someone))
  )
) :-
  at(N, action(X, attack)),
  at(N, arg(X, target, Target)),
  st(N, relation(threatening, Target, Someone)),
  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(success, Every),
```

— a-attack.lp —

```
      manifests,
      potential(problem, relation(threatening, Aggressor, Target))
    )
) :-
    outcome_val(attack, success, Every),
    at(N, action(X, attack)),
    at(N, arg(X, aggressor, Aggressor)),
    at(N, arg(X, target, Target)),
    st(N, relation(threatening, Aggressor, Target)),
    story_op(N, build_options).

% fighting is dangerous:
at(
  N,
  consequence_of(
    X,
    o(aggressor_state, injured),
    state(injured, Aggressor)
  )
) :-
  at(N, action(X, attack)),
  at(N, arg(X, aggressor, Aggressor)),
  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(aggressor_state, killed),
    state(killed, Aggressor)
  )
) :-
  at(N, action(X, attack)),
  at(N, arg(X, aggressor, Aggressor)),
  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(target_state, injured),
    state(injured, Target)
  )
) :-
  at(N, action(X, attack)),
  at(N, arg(X, target, Target)),
  story_op(N, build_options).

at(
```

— a-attack.lp —

```
    N,
    consequence_of(
      X,
      o(target_state, killed),
      state(killed, Target)
    )
  ) :-
    at(N, action(X, attack)),
    at(N, arg(X, target, Target)),
    story_op(N, build_options).

  % Looting is a possibility:

  1 = {
    at(
      N,
      consequence_of(
        X,
        o(get_loot, loot),
        relation(has_item, Aggressor, Item)
      )
    ) : st(N, relation(has_item, Aggressor, Item))
  } :-
    at(N, action(X, attack)),
    at(N, arg(X, aggressor, Aggressor)),
    1 <= { st(N, relation(has_item, Aggressor, Item)) },
    story_op(N, build_options).
```

— a-buy_healing.lp —

```
% buy_healing
action(buy_healing).

reflexive(buy_healing).
injured_can_initiate(buy_healing).

% arguments

argument(buy_healing, doctor, actor).
argument(buy_healing, patient, actor).
argument(buy_healing, buyer, actor).
argument(buy_healing, price, item).
initiator(buy_healing, buyer).
default_intent(buy_healing, o(success, healed)).

```

— a-buy_healing.lp —

```
% outcomes

outcome_val(buy_healing, success, healed).
outcome_val(buy_healing, success, still_injured).
outcome_val(buy_healing, success, killed).

outcome_val(buy_healing, deal, deal).
outcome_val(buy_healing, deal, no_deal).

outcome_excludes(
  buy_healing,
  o(deal, no_deal),
  o(success, healed)
).

outcome_excludes(
  buy_healing,
  o(deal, no_deal),
  o(success, killed)
).

% skills

skill_link(
  healing, required, tool,
  buy_healing, doctor,
  o(success, healed)
).

skill_link(
  healing, avoids, no_tool,
  buy_healing, doctor,
  o(success, killed)
).

skill_link(
  elocution, promotes, no_tool,
  buy_healing, buyer,
  o(deal, deal)
).

% Patients must be injured:
error(m("Bought␣healing␣from␣unwilling␣doctor.", N, X)) :-
  at(N, action(X, buy_healing)),
  at(N, arg(X, doctor, Doctor)),
  not st(N, property(offering_service, Doctor, treat_injury)),
  story_op(N, build_options).

error(m("Bought␣healing␣for␣uninjured␣patient.", N, X)) :-
```

— a-buy_healing.lp —

```
    at(N, action(X, buy_healing)),
    at(N, arg(X, patient, Patient)),
    not st(N, state(injured, Patient)),
    story_op(N, build_options).

% Trade constraints:
error(m("Unintelligent␣buyer.", N, X, Buyer)) :-
    at(N, action(X, buy_healing)),
    at(N, arg(X, buyer, Buyer)),
    st(N, property(has_skill, Buyer, unintelligent)),
    story_op(N, build_options).

error(m("Unintelligent␣doctor.", N, X, Doctor)) :-
    at(N, action(X, buy_healing)),
    at(N, arg(X, doctor, Doctor)),
    st(N, property(has_skill, Doctor, unintelligent)),
    story_op(N, build_options).

error(m("Buyer␣can't␣trade␣price.", N, X, Buyer, Price)) :-
    at(N, action(X, buy_healing)),
    at(N, arg(X, buyer, Buyer)),
    at(N, arg(X, price, Price)),
    not at(N, can_trade(Buyer, Price)),
    story_op(N, build_options).

% Buying healing gets rid of injuries:
at(
  N,
  consequence_of(
    X,
    o(success, healed),
    resolves,
    potential(problem, state(injured, Patient))
  )
) :-
    at(N, action(X, buy_healing)),
    at(N, arg(X, doctor, Doctor)),
    at(N, arg(X, patient, Patient)),
    st(N, state(injured, Patient)),
    story_op(N, build_options).

% However, when buying healing there's a risk of death:
at(N, consequence_of(X, o(success, killed), state(dead, Patient))) :-
    at(N, action(X, buy_healing)),
    at(N, arg(X, patient, Patient)),
    story_op(N, build_options).

% And the doctor takes the price:
at(
```

— a-buy_healing.lp —

```
    N,
    consequence_of(
      X,
      o(deal, deal),
      relation(has_item, Doctor, Price)
    )
) :-
    at(N, action(X, buy_healing)),
    at(N, arg(X, doctor, Doctor)),
    at(N, arg(X, price, Price)),
    story_op(N, build_options).

% Buying healng resolves offers to sell healing:
at(
    N,
    consequence_of(
      X,
      o(deal, deal),
      resolves,
      potential(
        opportunity,
        property(offering_service, Doctor, treat_injury)
      )
    )
) :-
    at(N, action(X, trade)),
    at(N, arg(X, seller, Doctor)),
    st(N, property(offering_service, Doctor, treat_injury)),
    story_op(N, build_options).
```

— a-deny_blame.lp —

```
% deny_blame
action(deny_blame).

% arguments

argument(deny_blame, denier, actor).
argument(deny_blame, accuser, actor).
argument(deny_blame, victim, actor).
initiator(deny_blame, denier).
default_intent(deny_blame, o(success, exonerated)).

% outcomes
```

— a-deny_blame.lp —

```
outcome_val(deny_blame, success, exonerated).
outcome_val(deny_blame, success, ignored).

% skills:

skill_link(
  elocution, contest, no_tool,
  deny_blame,
  between(denier, accuser),
  either(o(success, exonerated), o(success, ignored))
).

% constraints:

error(m("Unintelligent␣blame.", N, X)) :-
  at(N, action(X, deny_blame)),
  at(N, arg(X, Any, ShouldBeSmart)),
  st(N, property(has_skill, ShouldBeSmart, unintelligent)),
  story_op(N, build_options).

reflexive(deny_blame).

error(m("Accuser␣is␣also␣victim.", N, X)) :-
  at(N, action(X, deny_blame)),
  at(N, arg(X, accuser, SamePerson)),
  at(N, arg(X, victim, SamePerson)).

error(m("Accuser␣is␣also␣denier.", N, X)) :-
  at(N, action(X, deny_blame)),
  at(N, arg(X, denier, SamePerson)),
  at(N, arg(X, accuser, SamePerson)).

% denying blame is one way to get rid of an accusation:

at(
  N,
  consequence_of(
    X,
    o(success, exonerated),
    resolves,
    potential(problem, relation(accusing, Accuser, Victim))
  )
) :-
  at(N, action(X, deny_blame)),
  at(N, arg(X, accuser, Accuser)),
  at(N, arg(X, victim, Victim)),
  at(N, potential(problem, relation(accusing, Accuser, Victim))),
  story_op(N, build_options).
```

— a-deny_blame.lp —

— a-dispel.lp —

```
% dispel
action(dispel).

chaotic(dispel).

% arguments

argument(dispel, caster, actor).
argument(dispel, target, actor).
initiator(dispel, caster).
default_intent(dispel, o(success, dispelled)).

% outcomes

outcome_val(dispel, success, dispelled).
outcome_val(dispel, success, no_effect).

% skills:

skill_link(
  sorcery, contest, tool,
  dispel,
  between(caster, target),
  either(o(success, cursed), o(success, no_effect))
).

error(m("Attempt␣to␣dispel␣a␣non-cursed␣target.", N, X)) :-
  at(N, action(X, dispel)),
  at(N, arg(X, target, Target)),
  O = {
    st(N, property(polymorphed, Target, Any)) :
      st(N, property(polymorphed, Target, Any)),
      any_class(Any)
  },
  story_op(N, build_options).

error(m("Unintelligent␣caster.", N, X)) :-
  at(N, action(X, dispel)),
  at(N, arg(X, caster, Unintelligent)),
  st(N, property(has_skill, Unintelligent, unintelligent)),
  story_op(N, build_options).
```

— a-dispel.lp —

```
% effects:

% Note: this one effect has many rammifications; see
% content/p-polymorphed.lp
at(
  N,
  consequence_of(
    X,
    o(success, dispelled),
    resolved,
    potential(problem, property(polymorphed, Target, OriginalType))
  )
) :-
  at(N, action(X, dispel)),
  at(N, arg(X, target, Target)),
  st(N, property(polymorphed, Target, OriginalType)),
  story_op(N, build_options).
```

— a-explain_innocence.lp —

```
% explain_innocence
action(explain_innocence).

% arguments

argument(explain_innocence, explainer, actor).
argument(explain_innocence, accuser, actor).
argument(explain_innocence, victim, actor).
initiator(explain_innocence, explainer).
default_intent(explain_innocence, o(success, exonerated)).

% outcomes

outcome_val(explain_innocence, success, exonerated).
outcome_val(explain_innocence, success, ignored).

% skills:

skill_link(
  storytelling, promotes, no_tool,
  explain_innocence,
  explainer,
  o(success, exonerated)
).

```

— a-explain_innocence.lp —

```
% explaining innocence is unlikely to work if the accused is guilty:
% TODO: More specific accusations and crimes!

at(N, unlikely_outcome(X, o(success, exonerated))) :-
  at(N, action(X, explain_innocence)),
  at(N, arg(X, accuser, Justified)),
  at(N, arg(X, victim, Guilty)),
  st(N, relation(has_item, Guilty, Stolen)),
  st(N, relation(stolen_from, Anyone, Stolen)).

at(N, relevant_factor(X, Guilty, has_stolen_item, Stolen)) :-
  at(N, action(X, explain_innocence)),
  at(N, arg(X, accuser, Justified)),
  at(N, arg(X, victim, Guilty)),
  st(N, relation(has_item, Guilty, Stolen)),
  st(N, relation(stolen_from, Anyone, Stolen)).

% constraints:

error(m("Unintelligent␣blame.", N, X)) :-
  at(N, action(X, explain_innocence)),
  at(N, arg(X, Any, ShouldBeSmart)),
  st(N, property(has_skill, ShouldBeSmart, unintelligent)),
  story_op(N, build_options).

reflexive(explain_innocence).

error(m("Accuser␣is␣also␣victim.", N, X)) :-
  at(N, action(X, explain_innocence)),
  at(N, arg(X, accuser, SamePerson)),
  at(N, arg(X, victim, SamePerson)).

error(m("Accuser␣is␣also␣explainer.", N, X)) :-
  at(N, action(X, explain_innocence)),
  at(N, arg(X, explainer, SamePerson)),
  at(N, arg(X, accuser, SamePerson)).

% explaining innocence is one way to get rid of an accusation:

at(
  N,
  consequence_of(
    X,
    o(success, exonerated),
    resolves,
    potential(problem, relation(accusing, Accuser, Victim))
  )
) :-
  at(N, action(X, explain_innocence)),
```

— a-explain_innocence.lp —

```
    at(N, arg(X, accuser, Accuser)),
    at(N, arg(X, victim, Victim)),
    at(N, potential(problem, relation(accusing, Accuser, Victim))),
    story_op(N, build_options).
```

— a-flee.lp —

```
% flee
action(flee).

chaotic(flee).

% arguments

argument(flee, fearful, actor).
argument(flee, from, actor).
initiator(flee, fearful).
injured_can_initiate(flee).
default_intent(flee, o(success, escape)).

% outcomes

outcome_val(flee, success, escape).
outcome_val(flee, success, failure).

outcome_val(flee, get_injured, injured).
outcome_val(flee, get_injured, safe).

% skills

at(
  N,
  skill_link(
    acrobatics, contest, no_tool,
    flee,
    between(fearful, from),
    either(o(success, escape), o(success, failure))
  )
) :-
  1 <= {
    setting(N, city);
    setting(N, town)
  },
  story_node(N).
```

— a-flee.lp —

```
at(
  N,
  skill_link(
    acrobatics, contest, no_tool,
    flee,
    between(fearful, from),
    either(o(get_injured, safe), o(get_injured, injured))
  )
) :-
  1 <= {
    setting(N, city);
    setting(N, town)
  },
  story_node(N).

at(
  N,
  skill_link(
    wilderness_lore, contest, no_tool,
    flee,
    between(fearful, from),
    either(o(success, escape), o(success, failure))
  )
) :-
  1 <= {
    setting(N, road);
    setting(N, wilderness)
  },
  story_node(N).

at(
  N,
  skill_link(
    wilderness_lore, contest, no_tool,
    flee,
    between(fearful, from),
    either(o(get_injured, safe), o(get_injured, injured))
  )
) :-
  1 <= {
    setting(N, road);
    setting(N, wilderness)
  },
  story_node(N).

% Fleeing gets rid of threats
at(
  N,
  consequence_of(
```

```
      X,
      o(success, escape),
      resolves,
      potential(problem, relation(threatening, From, Fearful))
    )
  ) :-
    at(N, action(X, flee)),
    at(N, arg(X, fearful, Fearful)),
    at(N, arg(X, from, From)),
    st(N, relation(threatening, From, Fearful)),
    story_op(N, build_options).

% Fleeing gets rid of accusations
% TODO: Complicate this somewhat!!
at(
  N,
  consequence_of(
    X,
    o(success, escape),
    resolves,
    potential(problem, relation(accusing, From, Fearful))
  )
) :-
  at(N, action(X, flee)),
  at(N, arg(X, fearful, Fearful)),
  at(N, arg(X, from, From)),
  st(N, relation(accusing, From, Fearful)),
  story_op(N, build_options).

% Fleeing puts things off-stage, unless they're a party member, in
% which case it puts everything not in your party or belonging to it
% off-stage:

at(
  N,
  consequence_of(X, o(success, escape), state(off_stage, Fearful))
) :-
  at(N, action(X, leave)),
  at(N, arg(X, fearful, Fearful)),
  not st(N, state(party_member, Fearful)),
  story_op(N, build_options).

at(
  N,
  consequence_of(X, o(success, escape), state(off_stage, Item))
) :-
  at(N, action(X, leave)),
  at(N, arg(X, fearful, Fearful)),
  not st(N, state(party_member, Fearful)),
```

— a-flee.lp —

```
    st(N, relation(has_item, Fearful, Item)),
    story_op(N, build_options).

% All of the things you are moving away from:

move_away_from(N, X, inst(Type, Inst)) :-
  at(N, action(X, flee)),
  at(N, arg(X, fearful, Fearful)),
  st(N, state(party_member, Fearful)),
  st(N, inst(Type, Inst)),
  not st(N, state(party_member, inst(Type, Inst))),
  0 = {
    st(N, relation(has_item, PartyMember, inst(Type, Inst))) :
      st(N, state(party_member, PartyMember))
  },
  story_op(N, build_options).

at(
  N,
  consequence_of(X, o(success, escape), state(off_stage, Thing))
) :-
  at(N, action(X, leave)),
  at(N, arg(X, fearful, Fearful)),
  st(N, state(party_member, Fearful)),
  move_away_from(N, X, Thing),
  story_op(N, build_options).

% You can be injured if you fail to escape:

at(
  N,
  consequence_of(X, o(get_injured, injured), state(injured, Fearful))
) :-
  at(N, action(X, flee)),
  at(N, arg(X, fearful, Fearful)),
  story_op(N, build_options).
% TODO: something about the attacker's intent?
```

— a-gossip.lp —

```
% gossip
action(gossip).

% arguments

argument(gossip, interested, actor).
argument(gossip, knowledgeable, actor).
initiator(gossip, interested).
default_intent(gossip, o(utility, useful)).

% outcomes

outcome_val(gossip, utility, useful).
outcome_val(gossip, utility, useless).

% skills

skill_link(
  elocution, required, no_tool,
  gossip, interested,
  o(utility, useful)
).

error(m("Unintelligent␣gossip.", N, X)) :-
  at(N, action(X, gossip)),
  at(N, arg(X, interested, Unintelligent)),
  st(N, property(has_skill, Unintelligent, unintelligent)),
  story_op(N, build_options).

error(m("Unintelligent␣gossip.", N, X)) :-
  at(N, action(X, gossip)),
  at(N, arg(X, knowledgeable, Unintelligent)),
  st(N, property(has_skill, Unintelligent, unintelligent)),
  story_op(N, build_options).

% Either way you can't gossip with the same actor again.
at(
  N,
  consequence_of(
    X,
    o(utility, useful),
    resolves,
    potential(opportunity, state(knows_gossip, Knowledgeable))
  )
) :-
  at(N, action(X, gossip)),
```

```
    outcome_val(gossip, utility, OVal),
    at(N, arg(X, knowledgeable, Knowledgeable)),
    st(N, state(knows_gossip, Knowledgeable)),
    story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(utility, useless),
    nullifies,
    potential(opportunity, state(knows_gossip, Knowledgeable))
  )
) :-
  at(N, action(X, gossip)),
  outcome_val(gossip, utility, OVal),
  at(N, arg(X, knowledgeable, Knowledgeable)),
  st(N, state(knows_gossip, Knowledgeable)),
  story_op(N, build_options).


% TODO: More effects!
```

— a-leave.lp —

```
%% leave
%action(leave).
%
%% arguments
%
%argument(leave, subject, actor).
%initiator(leave, subject).
%default_intent(leave, o(result, gone)).
%
%% outcomes
%
%outcome_val(leave, result, gone).
%outcome_val(leave, result, detained).
%
%% skills
%
%% none
%
%% You can't leave if you're being threatened (you should flee
%% instead):
```

— a-leave.lp —

```
%
%error(m("Walked out of a threatening situation.", N, X)) :-
%  at(N, action(X, leave)),
%  at(N, arg(X, subject, Subject)),
%  st(N, relation(threatening, Someone, Subject)),
%  story_op(N, build_options).
%
%% Since it's all the same in the end, the initator of leaving should
%% always be % you.
%
%error(m("Party member other than you left...")) :-
%  at(N, action(X, leave)),
%  at(N, arg(X, subject, Subject)),
%  st(N, state(party_member, Subject)),
%  Subject != inst(actor, you),
%  story_op(N, build_options).
%
%% Leaving puts things off-stage, unless they're you, in which case it
%% puts % everything not in your party or belonging to it off-stage:
%
%at(
%  N,
%  consequence_of(X, o(result, gone), state(off_stage, Subject))
%) :-
%  at(N, action(X, leave)),
%  at(N, arg(X, subject, Subject)),
%  Subject != inst(actor, you),
%  story_op(N, build_options).
%
%at(
%  N,
%  consequence_of(X, o(result, gone), state(off_stage, Item))
%) :-
%  at(N, action(X, leave)),
%  at(N, arg(X, subject, Subject)),
%  Subject != inst(actor, you),
%  st(N, relation(has_item, Subject, Item)),
%  story_op(N, build_options).
%
%% All of the things you are moving away from:
%
%move_away_from(N, X, inst(Type, Inst)) :-
%  at(N, action(X, leave)),
%  at(N, arg(X, subject, inst(actor, you))),
%  st(N, inst(Type, Inst)),
%  not st(N, state(party_member, inst(Type, Inst))),
%  0 = {
%    st(N, relation(has_item, PartyMember, inst(Type, Inst))) :
%      st(N, state(party_member, PartyMember))
```

— a-leave.lp —

```
%  },
%  story_op(N, build_options).
%
%at(
%  N,
%  consequence_of(X, o(result, gone), state(off_stage, Thing))
%) :-
%  at(N, action(X, leave)),
%  at(N, arg(X, subject, inst(actor, you))),
%  move_away_from(N, X, Thing),
%  story_op(N, build_options).
```

— a-pacify.lp —

```
% pacify
action(pacify).

% arguments

argument(pacify, pacifier, actor).
argument(pacify, aggressive, actor).
initiator(pacify, pacifier).
default_intent(pacify, o(mood, relaxed)).

% outcomes

outcome_val(pacify, mood, relaxed).
outcome_val(pacify, mood, enraged).

outcome_val(pacify, get_injured, injured).
outcome_val(pacify, get_injured, safe).

outcome_excludes(pacify, o(mood, relaxed), o(get_injured, injured)).

% skills

skill_link(
  music, required, tool,
  pacify, pacifier,
  o(mood, relaxed)
).

skill_link(
  music, promotes, tool,
  pacify, pacifier,
```

— a-pacify.lp —

```
    o(mood, relaxed)
  ).

% TODO: Should we really have such high expectations?
%skill_link(
%  music, promotes, tool,
%  pacify, pacifier,
%  o(mood, relaxed)
%).

skill_link(
  music, required, no_tool,
  pacify, pacifier,
  o(get_injured, safe)
).


% you can only pacify unintelligent beings: TODO: Change this?
error(m("Pacification␣of␣intelligent␣target.", N, X)) :-
  at(N, arg(X, aggressive, Aggressive)),
  not st(N, property(has_skill, Aggressive, unintelligent)),
  story_op(N, build_options).

% pacifying is one way to deal with a threat
at(
  N,
  consequence_of(
    X,
    o(mood, relaxed),
    resolves,
    potential(problem, relation(threatening, Aggressive, Someone))
  )
) :-
  at(N, action(X, pacify)),
  at(N, arg(X, aggressive, Aggressive)),
  st(N, relation(threatening, Aggressive, Someone)),
  story_op(N, build_options).

% if things go badly the subject threatens you
at(
  N,
  consequence_of(
    X,
    o(mood, enraged),
    relation(threatening, Aggressive, Pacifier)
  )
) :-
  at(N, action(X, pacify)),
  at(N, arg(X, aggressive, Aggressive)),
```

— a-pacify.lp —

```
    at(N, arg(X, pacifier, Pacifier)),
    story_op(N, build_options).

% and ignores anyone else
at(
  N,
  consequence_of(
    X,
    o(mood, enraged),
    nullifies,
    potential(problem, relation(threatening, Aggressive, Someone))
  )
) :-
  at(N, action(X, pacify)),
  at(N, arg(X, aggressive, Aggressive)),
  at(N, arg(X, pacifier, Pacifier)),
  st(N, relation(threatening, Aggressive, Someone)),
  Someone != Pacifier,
  story_op(N, build_options).

% You can get injured if things go badly.
at(
  N,
  consequence_of(
    X,
    o(get_injured, injured),
    state(injured, Pacifier)
  )
) :-
  at(N, action(X, pacify)),
  at(N, arg(X, pacifier, Pacifier)),
  story_op(N, build_options).
```

— a-pay_off.lp —

```
% pay_off
action(pay_off).

% arguments

argument(pay_off, asking, actor).
argument(pay_off, listening, actor).
argument(pay_off, victim, actor).
argument(pay_off, price, item).
initiator(pay_off, asking).
```

— a-pay_off.lp —

```
default_intent(pay_off, o(deal, deal)).

% outcomes

outcome_val(pay_off, deal, deal).
outcome_val(pay_off, deal, no_deal).

% skills

skill_link(
  elocution, promotes, no_tool,
  pay_off, asking,
  o(deal, deal)
).

error(m("Pay␣off␣without␣price.", N, X)) :-
  at(N, action(X, pay_off)),
  at(N, arg(X, asking, Asking)),
  at(N, arg(X, price, Price)),
  not at(N, can_trade(Asking, Price)),
  story_op(N, build_options).

error(m("Price␣is␣not␣a␣treasure.", N, X)) :-
  at(N, action(X, pay_off)),
  at(N, arg(X, asking, Asking)),
  at(N, arg(X, price, Price)),
  not is_instance(N, Price, treasure),
  story_op(N, build_options).

error(m("Unintelligent␣asker.", N, X)) :-
  at(N, action(X, pay_off)),
  at(N, arg(X, asking, Asking)),
  st(N, property(has_skill, Asking, unintelligent)),
  story_op(N, build_options).

error(m("Unintelligent␣listener.", N, X)) :-
  at(N, action(X, pay_off)),
  at(N, arg(X, listening, Listening)),
  st(N, property(has_skill, Listening, unintelligent)),
  story_op(N, build_options).

error(
  m("Bribe␣target␣isn't␣threatening␣or␣accusing␣the␣victim.", N, X)
) :-
  at(N, action(X, talk_down)),
  at(N, arg(X, listening, Listening)),
  at(N, arg(X, victim, Victim)),
  0 = {
    st(N, relation(threatening, Listening, Victim));
```

— a-pay_off.lp —

```
    st(N, relation(accusing, Listening, Victim))
  },
  story_op(N, build_options).

% the threat is withdrawn:
at(
  N,
  consequence_of(
    X,
    o(deal, deal),
    resolves,
    potential(problem, relation(threatening, Listening, Victim))
  )
) :-
  at(N, action(X, pay_off)),
  at(N, arg(X, listening, Listening)),
  at(N, arg(X, victim, Victim)),
  st(N, relation(threatening, Listening, Victim)),
  story_op(N, build_options).

% or the accusation is withdrawn:
at(
  N,
  consequence_of(
    X,
    o(deal, deal),
    resolves,
    potential(problem, relation(accusing, Listening, Victim))
  )
) :-
  at(N, action(X, pay_off)),
  at(N, arg(X, listening, Listening)),
  at(N, arg(X, victim, Victim)),
  st(N, relation(accusing, Listening, Victim)),
  story_op(N, build_options).

% but the price must be paid:
at(
  N,
  consequence_of(
    X,
    o(deal, deal),
    relation(has_item, Listening, Item)
  )
) :-
  at(N, action(X, pay_off)),
  at(N, arg(X, listening, Listening)),
  at(N, arg(X, price, Item)),
  story_op(N, build_options).
```

— a-pay_off.lp —

— a-play_song.lp —

```
% play_song
action(play_song).

% arguments

argument(play_song, musician, actor).
argument(play_song, audience, actor).
initiator(play_song, musician).
default_intent(play_song, o(quality, harmonious)).
% TODO: the instrument?

% outcomes

outcome_val(play_song, quality, harmonious).
outcome_val(play_song, quality, plain).
% TODO: add discordant?

% skills

skill_link(
  music, required, tool,
  play_song, musician,
  o(quality, harmonious)
).

error(m("Unintelligent musician.", N, X)) :-
  at(N, action(X, play_song)),
  at(N, arg(X, musician, Unintelligent)),
  st(N, property(has_skill, Unintelligent, unintelligent)),
  story_op(N, build_options).

error(m("Unintelligent audience.", N, X)) :-
  at(N, action(X, play_song)),
  at(N, arg(X, audience, Unintelligent)),
  st(N, property(has_skill, Unintelligent, unintelligent)),
  story_op(N, build_options).

% effects

at(
  N,
  consequence_of(
```

— a-play_song.lp —

```
      X,
      o(quality, harmonious),
      resolves,
      potential(opportunity, state(bored, Audience))
    )
  ):-
    at(N, action(X, play_song)),
    at(N, arg(X, audience, Audience)),
    st(N, state(bored, Audience)),
    story_op(N, build_options).

  at(
    N,
    consequence_of(
      X,
      o(quality, plain),
      nullifies,
      potential(opportunity, state(bored, Audience))
    )
  ):-
    at(N, action(X, play_song)),
    at(N, arg(X, audience, Audience)),
    st(N, state(bored, Audience)),
    story_op(N, build_options).


  % TODO: angering the audience?

  % TODO: Other effects?
```

— a-polymorph.lp —

```
1 % polymorph
  action(polymorph).

  chaotic(polymorph).
5
  % arguments

  argument(polymorph, caster, actor).
  argument(polymorph, target, actor).
10 initiator(polymorph, caster).
  default_intent(polymorph, o(success, cursed)).

  % outcomes
```

— a-polymorph.lp —

```
outcome_val(polymorph, success, cursed).
outcome_val(polymorph, success, no_effect).

% skills:

skill_link(
  sorcery, required, tool,
  polymorph, caster,
  o(success, cursed)
).

skill_link(
  sorcery, promotes, tool,
  polymorph, caster,
  o(success, cursed)
).

error(m("Unintelligent␣caster.", N, X)) :-
  at(N, action(X, polymorph)),
  at(N, arg(X, caster, Unintelligent)),
  st(N, property(has_skill, Unintelligent, unintelligent)),
  story_op(N, build_options).

% effects:

% Note: this one effect has many rammifications; see
% content/p-polymorphed.lp
at(
  N,
  consequence_of(
    X,
    o(success, cursed),
    property(polymorphed, Target, OriginalType)
  )
) :-
  at(N, action(X, polymorph)),
  at(N, arg(X, target, Target)),
  st(N, property(type, Target, OriginalType)),
  story_op(N, build_options).
```

— —

— a-pursue.lp —

```
%% pursue
%action(pursue).
%
%chaotic(pursue).
%
%% arguments
%
%argument(pursue, subject, actor).
%argument(pursue, object, actor).
%off_stage_okay(pursue, object).
%initiator(pursue, subject).
%default_intent(pursue, o(result, caught_up)).
%
%% outcomes
%
%outcome_val(pursue, result, caught_up).
%outcome_val(pursue, result, lost).
%
%% skills (same as flee)
%
%at(
%  N,
%  skill_link(
%    acrobatics, contest, no_tool,
%    pursue,
%    between(subject, object),
%    either(o(result, caught_up), o(result, lost))
%  )
%) :-
%  1 <= {
%    setting(N, city);
%    setting(N, town)
%  },
%  story_node(N).
%
%at(
%  N,
%  skill_link(
%    wilderness_lore, contest, no_tool,
%    pursue,
%    between(subject, object),
%    either(o(result, caught_up), o(result, lost))
%  )
%) :-
%  1 <= {
%    setting(N, road);
```

— a-pursue.lp —

```
%    setting(N, wilderness)
%  },
%  story_node(N).
%
%% You can't pursue if you're being threatened (you should flee
%% instead):
%
%error(m("Pursued out of a threatening situation.", N, X)) :-
%  at(N, action(X, pursue)),
%  at(N, arg(X, subject, Subject)),
%  st(N, relation(threatening, Someone, Subject)),
%  story_op(N, build_options).
%
%% The object of pursuit must be off-stage:
%
%error(m("Pursued on-stage actor.", N, X)) :-
%  at(N, action(X, pursue)),
%  at(N, arg(X, object, Object)),
%  not st(N, state(off_stage, Object)),
%  story_op(N, build_options).
%
%% Pursuit puts things off-stage, unless they're you, in which case it
%% puts % everything not in your party or belonging to it off-stage,
%% while putting % previously off-stage things on-stage (unless you
%% get lost).
%
%at(
%  N,
%  consequence_of(X, o(result, caught_up), state(off_stage, Subject))
%) :-
%  at(N, action(X, pursue)),
%  at(N, arg(X, subject, Subject)),
%  not st(N, state(party_member, Subject)),
%  story_op(N, build_options).
%
%at(
%  N,
%  consequence_of(X, o(result, caught_up), state(off_stage, Item))
%) :-
%  at(N, action(X, pursue)),
%  at(N, arg(X, subject, Subject)),
%  not st(N, state(party_member, Subject)),
%  st(N, relation(has_item, Subject, Item)),
%  story_op(N, build_options).
%
%at(
%  N,
%  consequence_of(X, o(result, lost), state(off_stage, Subject))
%) :-
```

— a-pursue.lp —

```
%  at(N, action(X, pursue)),
%  at(N, arg(X, subject, Subject)),
%  not st(N, state(party_member, Subject)),
%  story_op(N, build_options).
%
%at(
%  N,
%  consequence_of(X, o(result, lost), state(off_stage, Item))
%) :-
%  at(N, action(X, pursue)),
%  at(N, arg(X, subject, Subject)),
%  not st(N, state(party_member, Subject)),
%  st(N, relation(has_item, Subject, Item)),
%  story_op(N, build_options).
%
%% All of the things you are moving away from:
%
%move_away_from(N, X, inst(Type, Inst)) :-
%  at(N, action(X, pursue)),
%  at(N, arg(X, subject, Subject)),
%  st(N, state(party_member, Subject)),
%  st(N, inst(Type, Inst)),
%  not st(N, state(party_member, inst(Type, Inst))),
%  0 = {
%    st(N, relation(has_item, PartyMember, inst(Type, Inst))) :
%      st(N, state(party_member, PartyMember))
%  },
%  not st(N, state(off_stage, inst(Type, Inst))),
%  story_op(N, build_options).
%
%at(
%  N,
%  consequence_of(X, o(result, caught_up), state(off_stage, Thing))
%) :-
%  at(N, action(X, pursue)),
%  at(N, arg(X, subject, Subject)),
%  st(N, state(party_member, Subject)),
%  move_away_from(N, X, Thing),
%  story_op(N, build_options).
%
%at(
%  N,
%  consequence_of(X, o(result, lost), state(off_stage, Thing))
%) :-
%  at(N, action(X, pursue)),
%  at(N, arg(X, subject, Subject)),
%  st(N, state(party_member, Subject)),
%  move_away_from(N, X, Thing),
%  story_op(N, build_options).
```

— a-pursue.lp —

```
%
%at(
%  N,
%  consequence_of(
%    X,
%    o(result, caught_up),
%    _not,
%    state(off_stage, Thing)
%  )
%) :-
%  at(N, action(X, pursue)),
%  at(N, arg(X, subject, Subject)),
%  st(N, state(party_member, Subject)),
%  st(N, state(off_stage, Thing)),
%  story_op(N, build_options).
%% if you're lost instead, you wind up alone
```

— a-reach_destination.lp —

```
% reach_destination: a special party-only action that ends the story
action(reach_destination).

% arguments

argument(reach_destination, subject, actor).
initiator(reach_destination, subject).

% outcomes

outcome_val(reach_destination, journey, over).

% no skills

% The argument of reach_destination is fixed:

at(N, arg(Opt, subject, inst(actor, you))) :-
  at(N, action(Opt, reach_destination)).

% reach_destination cannot happen except at an ending:

error(m("Reached␣destination␣before␣end␣of␣story", N, X)) :-
  at(N, action(X, reach_destination)),
  not node_type(N, ending).

% reach_destination gets rid of all existing potentials:
```

— a-reach_destination.lp —

```
at(
  N,
  consequence_of(
    X,
    o(journey, over),
    nullifies, potential(Any, Pt)
  )
) :-
  at(N, action(X, reach_destination)),
  at(N, potential(Any, Pt)),
  story_op(N, build_options).

% reach_destination doesn't need to have any consequences as it can
% only happen at the end of a story.
```

— a-shift_blame.lp —

```
% shift_blame
action(shift_blame).

% arguments

argument(shift_blame, shifter, actor).
argument(shift_blame, accuser, actor).
argument(shift_blame, old_victim, actor).
argument(shift_blame, new_victim, actor).
initiator(shift_blame, shifter).
default_intent(shift_blame, o(success, shifted)).

% outcomes

outcome_val(shift_blame, success, shifted).
outcome_val(shift_blame, success, suspected).
outcome_val(shift_blame, success, ignored).

% skills:

skill_link(
  elocution, contest, no_tool,
  shift_blame,
  between(shifter, new_victim),
  either(o(success, shifted), o(success, suspected))
).
```

— a-shift_blame.lp —

```
% Shifting the blame to yourself is always easy:

at(N, likely_outcome(X, o(success, shifted))) :-
  at(N, action(X, shift_blame)),
  at(N, arg(X, accuser, Accuser)),
  at(N, arg(X, shifter, SamePerson)),
  at(N, arg(X, new_victim, SamePerson)).

at(
  N,
  relevant_factor(
    X,
    SamePerson,
    shift_blame_via_confession,
    SamePerson
  )
) :-
  at(N, action(X, shift_blame)),
  at(N, arg(X, accuser, Accuser)),
  at(N, arg(X, shifter, SamePerson)),
  at(N, arg(X, new_victim, SamePerson)).

% constraints:

error(m("Unintelligent␣blame.", N, X)) :-
  at(N, action(X, shfit_blame)),
  at(N, arg(X, Any, ShouldBeSmart)),
  st(N, property(has_skill, ShouldBeSmart, unintelligent)),
  story_op(N, build_options).

reflexive(shift_blame). % see custom role overlap constraints below

error(m("Shifter␣is␣also␣accuser.", N, X)) :-
  at(N, action(X, shift_blame)),
  at(N, arg(X, shifter, SamePerson)),
  at(N, arg(X, accuser, SamePerson)).

error(m("Accuser␣is␣the␣old␣accused.", N, X)) :-
  at(N, action(X, shift_blame)),
  at(N, arg(X, accuser, SamePerson)),
  at(N, arg(X, old_victim, SamePerson)).

error(m("Accuser␣is␣the␣new␣accused.", N, X)) :-
  at(N, action(X, shift_blame)),
  at(N, arg(X, accuser, SamePerson)),
  at(N, arg(X, new_victim, SamePerson)).

error(m("Old␣and␣new␣victims␣are␣the␣same.", N, X)) :-
  at(N, action(X, shift_blame)),
```

— a-shift_blame.lp —

```
    at(N, arg(X, old_victim, SamePerson)),
    at(N, arg(X, new_victim, SamePerson)).

error(m("Tried␣to␣shift␣non-existent␣blame.", N, X)) :-
    at(N, action(X, shift_blame)),
    at(N, arg(X, accuser, Accuser)),
    at(N, arg(X, old_victim, OldVictim)),
    not st(N, relation(accusing, Accuser, OldVictim)).

% shifting blame if successful changes an "accusing" relation

at(
  N,
  consequence_of(
    X,
    o(success, shifted),
    resolves,
    potential(problem, relation(accusing, Accuser, OldVictim))
  )
) :-
  at(N, action(X, shift_blame)),
  at(N, arg(X, accuser, Accuser)),
  at(N, arg(X, old_victim, OldVictim)),
  at(N, potential(problem, relation(accusing, Accuser, OldVictim))),
  story_op(N, build_options).

% The accusation just spreads rather than shifting if you fail...
%at(
%  N,
%  consequence_of(
%    X,
%    o(success, suspected),
%    resolves,
%    potential(problem, relation(accusing, Accuser, OldVictim))
%  )
%) :-
%  at(N, action(X, shift_blame)),
%  at(N, arg(X, shifter, Shifter)),
%  at(N, arg(X, accuser, Accuser)),
%  at(N, arg(X, old_victim, OldVictim)),
%  at(N, potential(problem, relation(accusing, Accuser, OldVictim))),
%  Shifter != OldVictim,
%  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(success, shifted),
```

— a-shift_blame.lp —

```
      relation(accusing, Accuser, NewVictim)
    )
  ) :-
    at(N, action(X, shift_blame)),
    at(N, arg(X, accuser, Accuser)),
    at(N, arg(X, new_victim, NewVictim)),
    story_op(N, build_options).

  at(
    N,
    consequence_of(
      X,
      o(success, suspected),
      relation(accusing, Accuser, Shifter)
    )
  ) :-
    at(N, action(X, shift_blame)),
    at(N, arg(X, shifter, Shifter)),
    at(N, arg(X, accuser, Accuser)),
    story_op(N, build_options).
```

— a-steal.lp —

```
% steal
action(steal).

chaotic(steal).

% arguments

argument(steal, thief, actor).
argument(steal, victim, actor).
argument(steal, target, item).
initiator(steal, thief).
default_intent(steal, o(get_item, get)).
default_intent(steal, o(get_caught, safe)).

% outcomes

outcome_val(steal, get_item, get).
outcome_val(steal, get_item, dont_get).

outcome_val(steal, get_caught, caught).
outcome_val(steal, get_caught, safe).
```

— a-steal.lp —

```
% skills:

skill_link(
  thievery, contest, no_tool,
  steal,
  between(thief, victim),
  either(o(get_item, get), o(get_item, dont_get))
).

skill_link(
  thievery, contest, no_tool,
  steal,
  between(thief, victim),
  either(o(get_caught, safe), o(get_caught, caught))
).

error(m("Stole␣from␣wrong␣victim.", N, X)) :-
  at(N, action(X, steal)),
  at(N, arg(X, victim, Victim)),
  at(N, arg(X, target, Item)),
  not st(N, relation(has_item, Victim, Item)).

% consequences

at(
  N,
  consequence_of(
    X,
    o(get_item, get),
    relation(has_item, Thief, Item)
  )
) :-
  at(N, action(X, steal)),
  at(N, arg(X, thief, Thief)),
  at(N, arg(X, target, Item)),
  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(get_item, get),
    relation(stolen_from, Victim, Item)
  )
) :-
  at(N, action(X, steal)),
  at(N, arg(X, victim, Victim)),
  at(N, arg(X, target, Item)),
  0 = {
```

— a-steal.lp —

```
      st(N, inst(actor, Anyone)) :
        st(N, inst(actor, Anyone)),
        st(N, relation(stolen_from, inst(actor, Anyone), Item))
  },
  story_op(N, build_options).

% if you get caught stealing something that doesn't belong to you
% you'll be either threatened or accused (TODO: accusations!).

at(
  N,
  consequence_of(
    X,
    o(get_caught, caught),
    relation(threatening, Victim, Thief)
  )
) :-
  at(N, action(X, steal)),
  at(N, arg(X, thief, Thief)),
  at(N, arg(X, victim, Victim)),
  at(N, arg(X, target, Item)),
  not st(N, relation(stolen_from, Thief, Item)),
  story_op(N, build_options).
```

— a-talk_down.lp —

```
% talk_down
action(talk_down).

% arguments

argument(talk_down, asking, actor).
argument(talk_down, listening, actor).
argument(talk_down, victim, actor).
initiator(talk_down, asking).
default_intent(talk_down, o(attitude, convinced)).

% outcomes

outcome_val(talk_down, attitude, convinced).
outcome_val(talk_down, attitude, unconvinced).

outcome_val(talk_down, is_enraged, enraged).
outcome_val(talk_down, is_enraged, not_enraged).
```

— a-talk_down.lp —

```
outcome_excludes(
  talk_down,
  o(attitude, convinced),
  o(is_enraged, enraged)
).

% skills

skill_link(
  elocution, required, no_tool,
  talk_down, asking,
  o(attitude, convinced)
).

skill_link(
  storytelling, promotes, no_tool,
  talk_down, asking,
  o(attitude, convinced)
).

skill_link(
  elocution, required, no_tool,
  talk_down, asking,
  o(is_enraged, not_enraged)
).

skill_link(
  storytelling, promotes, no_tool,
  talk_down, asking,
  o(is_enraged, not_enraged)
).

error(m("Unintelligent␣asker.", N, X)) :-
  at(N, action(X, talk_down)),
  at(N, arg(X, asking, Asking)),
  st(N, property(has_skill, Asking, unintelligent)),
  story_op(N, build_options).

error(m("Unintelligent␣listener.", N, X)) :-
  at(N, action(X, talk_down)),
  at(N, arg(X, listening, Listening)),
  st(N, property(has_skill, Listening, unintelligent)),
  story_op(N, build_options).

error(m("Victim␣not␣being␣threatened␣or␣accused.", N, X)) :-
  at(N, action(X, talk_down)),
  at(N, arg(X, listening, Listening)),
  at(N, arg(X, victim, Victim)),
  0 = {
```

— a-talk_down.lp —

```
      st(N, relation(threatening, Listening, Victim));
      st(N, relation(accusing, Listening, Victim))
    },
    story_op(N, build_options).

  % success convinces the threatener to back down
  at(
    N,
    consequence_of(
      X,
      o(attitude, convinced),
      resolves,
      potential(problem, relation(threatening, Listening, Victim))
    )
  ) :-
    at(N, action(X, talk_down)),
    at(N, arg(X, listening, Listening)),
    at(N, arg(X, victim, Victim)),
    st(N, relation(threatening, Listening, Victim)),
    story_op(N, build_options).

  at(
    N,
    consequence_of(
      X,
      o(attitude, convinced),
      resolves,
      potential(problem, relation(accusing, Listening, Victim))
    )
  ) :-
    at(N, action(X, talk_down)),
    at(N, arg(X, listening, Listening)),
    at(N, arg(X, victim, Victim)),
    st(N, relation(accusing, Listening, Victim)),
    story_op(N, build_options).

  % but the the target might become enraged instead
  at(
    N,
    consequence_of(
      X,
      o(is_enraged, enraged),
      relation(threatening, Listening, Asking)
    )
  ) :-
    at(N, action(X, talk_down)),
    at(N, arg(X, asking, Asking)),
    at(N, arg(X, listening, Listening)),
    at(N, arg(X, victim, Victim)),
```

— a-talk_down.lp —

```
    st(N, relation(threatening, Listening, Victim)),
    story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(is_enraged, enraged),
    relation(accusing, Listening, Asking)
  )
) :-
  at(N, action(X, talk_down)),
  at(N, arg(X, asking, Asking)),
  at(N, arg(X, listening, Listening)),
  at(N, arg(X, victim, Victim)),
  st(N, relation(accusing, Listening, Victim)),
  story_op(N, build_options).
```

— a-tell_story.lp —

```
% tell_story
action(tell_story).

% arguments

argument(tell_story, teller, actor).
argument(tell_story, audience, actor).
initiator(tell_story, teller).
default_intent(tell_story, o(quality, entertaining)).

% outcomes

outcome_val(tell_story, quality, entertaining).
outcome_val(tell_story, quality, boring).

% skills

skill_link(
  storytelling, promotes, no_tool,
  tell_story, teller,
  o(quality, entertaining)
).

error(m("Unintelligent storyteller.", N, X)) :-
  at(N, action(X, tell_story)),
```

— a-tell_story.lp —

```
    at(N, arg(X, teller, Unintelligent)),
    st(N, property(has_skill, Unintelligent, unintelligent)),
    story_op(N, build_options).

error(m("Unintelligent audience.", N, X)) :-
    at(N, action(X, tell_story)),
    at(N, arg(X, audience, Unintelligent)),
    st(N, property(has_skill, Unintelligent, unintelligent)),
    story_op(N, build_options).

at(
    N,
    consequence_of(
      X,
      o(quality, entertaining),
      resolves,
      potential(opportunity, state(bored, Audience))
    )
) :-
    at(N, action(X, tell_story)),
    at(N, arg(X, audience, Audience)),
    st(N, state(bored, Audience)),
    story_op(N, build_options).

at(
    N,
    consequence_of(
      X,
      o(quality, boring),
      nullifies,
      potential(opportunity, state(bored, Audience))
    )
) :-
    at(N, action(X, tell_story)),
    at(N, arg(X, audience, Audience)),
    st(N, state(bored, Audience)),
    story_op(N, build_options).

% TODO: Other effects?
```

— —

— a-trade.lp —

```
% trade
action(trade).

% arguments

argument(trade, buyer, actor).
argument(trade, seller, actor).
argument(trade, price, item).
argument(trade, goods, item).
initiator(trade, buyer).
default_intent(trade, o(deal, deal)).

% outcomes

outcome_val(trade, deal, deal).
outcome_val(trade, deal, no_deal).

% skills

skill_link(
  elocution, promotes, no_tool,
  trade, buyer,
  o(deal, deal)
).

error(m("Unintelligent␣buyer.", N, X, Buyer)) :-
  at(N, action(X, trade)),
  at(N, arg(X, buyer, Buyer)),
  st(N, property(has_skill, Buyer, unintelligent)),
  story_op(N, build_options).

error(m("Unintelligent␣seller.", N, X, Seller)) :-
  at(N, action(X, trade)),
  at(N, arg(X, seller, Seller)),
  st(N, property(has_skill, Seller, unintelligent)),
  story_op(N, build_options).

error(m("Buyer␣can't␣trade␣price.", N, X, Buyer, Price)) :-
  at(N, action(X, trade)),
  at(N, arg(X, buyer, Buyer)),
  at(N, arg(X, price, Price)),
  not at(N, can_trade(Buyer, Price)),
  story_op(N, build_options).

error(m("Seller␣can't␣trade␣goods.", N, X, Seller, Goods)) :-
  at(N, action(X, trade)),
```

— a-trade.lp —

```
    at(N, arg(X, seller, Seller)),
    at(N, arg(X, goods, Goods)),
    not at(N, can_trade(Seller, Goods)),
    story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(deal, deal),
    relation(has_item, Buyer, Goods)
  )
) :-
  at(N, action(X, trade)),
  at(N, arg(X, buyer, Buyer)),
  at(N, arg(X, goods, Goods)),
  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(deal, deal),
    relation(has_item, Seller, Price)
  )
) :-
  at(N, action(X, trade)),
  at(N, arg(X, seller, Seller)),
  at(N, arg(X, price, Price)),
  story_op(N, build_options).

% trading is how you buy stuff:
at(
  N,
  consequence_of(
    X,
    o(deal, deal),
    resolves,
    potential(opportunity, relation(selling, Seller, Goods))
  )
) :-
  at(N, action(X, trade)),
  at(N, arg(X, seller, Seller)),
  at(N, arg(X, goods, Goods)),
  st(N, relation(selling, Seller, Goods)),
  story_op(N, build_options).
```

— a-trade.lp —

— a-travel_onwards.lp —

```
% travel_onwards: a special party-only action that moves to the next
% vignette
action(travel_onwards).

% arguments

argument(travel_onwards, subject, actor).
initiator(travel_onwards, subject).
injured_can_initiate(travel_onwards).

% outcomes

outcome_val(travel_onwards, onwards, onwards).

% no skills

% The argument of travel_onwards is fixed:

at(N, arg(Opt, subject, inst(actor, you))) :-
  at(N, action(Opt, travel_onwards)).

% travel_onwards as an option removes all problems and opportunities
% related to things left behind, but it can't be an option if doing so
% would remove a non-hidden problem.

removed_by(N, X, potential(PType, state(PState, Subject))) :-
  at(N, action(X, travel_onwards)),
  at(N, potential(PType, state(PState, Subject))),
  travel_away_from(N, X, Subject),
  story_op(N, build_options).

removed_by(N, X, potential(PType, property(PProp, Subject, PVal))) :-
  at(N, action(X, travel_onwards)),
  at(N, potential(PType, property(PProp, Subject, PVal))),
  travel_away_from(N, X, Subject),
  story_op(N, build_options).

removed_by(N, X, potential(PType, relation(PRel, Subject, Object))) :-
  at(N, action(X, travel_onwards)),
  at(N, potential(PType, relation(PRel, Subject, Object))),
  travel_away_from(N, X, Subject),
  story_op(N, build_options).
```

— a-travel_onwards.lp —

```
removed_by(N, X, potential(PType, relation(PRel, Subject, Object))) :-
  at(N, action(X, travel_onwards)),
  at(N, potential(PType, relation(PRel, Subject, Object))),
  travel_away_from(N, X, Object),
  story_op(N, build_options).

error(m("Left␣a␣non-hidden␣problem␣behind.", N, Prb)) :-
  at(N, action(X, travel_onwards)),
  removed_by(N, X, potential(problem, Prb)),
  not at(N, hidden(potential(problem, Prb))),
  1 <= {
    st(N, state(party_member, Member))
      : at(N, problematic_for(potential(problem, Prb), Member))
  },
  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    o(onwards, onwards),
    nullifies, potential(PType, Pot)
  )
) :-
  at(N, action(X, travel_onwards)),
  at(N, potential(PType, Pot)),
  removed_by(N, X, potential(PType, Pot)),
  story_op(N, build_options).

% travelling onwards gets rid of all spontaneous instances and
% properties thereof unless the party has picked up the instance to
% take with them:

travel_away_from(N, X, inst(Type, Inst)) :-
  at(N, action(X, travel_onwards)),
  vignette(N, V),
  spontaneous(st(V, inst(Type, Inst))),
  st(N, inst(Type, Inst)),
  not st(N, state(party_member, inst(Type, Inst))),
  0 = {
    st(N, relation(has_item, PartyMember, Inst)) :
      st(N, state(party_member, PartyMember))
  },
  story_op(N, build_options).

at(N, consequence_of(X, o(onwards, onwards), _not, Inst)) :-
  travel_away_from(N, X, Inst).
```

— a-travel_onwards.lp —

```
at(
  N,
  consequence_of(
    X,
    o(onwards, onwards),
    _not, state(State, Inst)
  )
) :-
  st(N, state(State, Inst)),
  travel_away_from(N, X, Inst).

at(
  N,
  consequence_of(
    X,
    o(onwards, onwards),
    _not, property(Prop, Inst, Value)
  )
) :-
  st(N, property(Prop, Inst, Value)),
  travel_away_from(N, X, Inst).

at(
  N,
  consequence_of(
    X,
    o(onwards, onwards),
    _not, relation(Rel, Inst, Other)
  )
) :-
  st(N, relation(Rel, Inst, Other)),
  travel_away_from(N, X, Inst).

at(
  N,
  consequence_of(
    X,
    o(onwards, onwards),
    _not, relation(Rel, Other, Inst)
  )
) :-
  st(N, relation(Rel, Other, Inst)),
  travel_away_from(N, X, Inst).
```

— a-treat_injury.lp —

```
% treat_injury
action(treat_injury).

reflexive(treat_injury).
injured_can_initiate(treat_injury). % TODO: really this?
chaotic(treat_injury).

% arguments

argument(treat_injury, doctor, actor).
argument(treat_injury, patient, actor).
initiator(treat_injury, doctor).
default_intent(treat_injury, o(success, healed)).

% outcomes

outcome_val(treat_injury, success, healed).
outcome_val(treat_injury, success, still_injured).
outcome_val(treat_injury, success, killed).

% skills

skill_link(
  healing, required, tool,
  treat_injury, doctor,
  o(success, healed)
).

skill_link(
  healing, avoids, no_tool,
  treat_injury, doctor,
  o(success, killed)
).

% Patients must be injured:
error(m("Treated␣uninjured␣patient.", N, X)) :-
  at(N, action(X, treat_injury)),
  at(N, arg(X, patient, Patient)),
  not st(N, state(injured, Patient)),
  story_op(N, build_options).

error(m("Unintelligent␣doctor.", N, X)) :-
  at(N, action(X, treat_injury)),
  at(N, arg(X, doctor, Unintelligent)),
  st(N, property(has_skill, Unintelligent, unintelligent)),
  story_op(N, build_options).
```

— a-treat_injury.lp —

```
% Treating injuries gets rid of them
at(
  N,
  consequence_of(
    X,
    o(success, healed),
    resolves,
    potential(problem, state(injured, Patient))
  )
) :-
  at(N, action(X, treat_injury)),
  at(N, arg(X, doctor, Doctor)),
  at(N, arg(X, patient, Patient)),
  st(N, state(injured, Patient)),
  story_op(N, build_options).

% However, when treating injuries there's a risk of death:
at(N, consequence_of(X, o(success, killed), state(dead, Patient))) :-
  at(N, action(X, treat_injury)),
  at(N, arg(X, patient, Patient)),
  story_op(N, build_options).
```

B.5 Setup Definitions

These files define the individual setup possibilities. Each setup includes fixed elements but may also include some variable elements. The core file `setup.lp` defines the implications of most of the predicates used in these definitions. Note that the "`thiefs`"etup is currently inactive (and is commented out for efficiency).

— —

— s-healer.lp —

```
% a healer
% TODO: Merge this into a larger setup?

possible_setup(healer).

setup_argument_create(healer, doctor, healer).
setup_argument_create(healer, supplies, medicine_chest).

s_st(healer, relation(c(has_item), v(doctor), v(supplies))).

s_st(
  healer,
  property(c(offering_service), v(doctor), c(treat_injury))
).

error(m("Used␣healer␣setup␣without␣any␣patients", N)) :-
  setup(N, healer),
  0 = {
    st(N, state(injured, Inst))
  }.
```

— s-market.lp —

```
% a market

possible_setup(market).

setup_argument_create_n(market, merchant_one, merchant, 1, 1).
setup_argument_create_n(market, merchant_two, merchant, 0, 1).
setup_argument_create_n(market, noble, aristocrat, 0, 1).
setup_argument_create_n(market, peasant, laborer, 0, 1).
setup_argument_create_n(market, doctor, healer, 0, 1).
setup_argument_create_n(market, lowlife, bad_guy, 0, 1).

% Some stuff for the merchant(s) to sell:
setup_argument_n(market, goods_one, item, 0, 3).
setup_argument_n(market, goods_two, item, 0, 3).

error(m("Market␣population␣too␣low.", N)) :-
  setup(N, market),
  3 > {
    at(N, setup_arg(Arg, Anyone)) :
      at(N, setup_arg(Arg, Anyone)),
```

— s-market.lp —

```
        setup_argument_create_n(market, Arg, Typ, L, U)
    }.

error(m("Too much boredom", N)) :-
  setup(N, market),
  3 <= {
    sp_st(N, state(bored, inst(actor, A))) : st(N, inst(actor, A))
  },
  story_op(N, initialize_node).

error(m("Too much gossip", N)) :-
  setup(N, market),
  3 <= {
    sp_st(N, state(knows_gossip, inst(actor, A)))
      : st(N, inst(actor, A))
  },
  story_op(N, initialize_node).

error(m("Noble accusing multiple parties.", N)) :-
  setup(N, market),
  at(N, setup_arg(noble, Noble)),
  2 <= {
    sp_st(N, relation(accusing, Noble, inst(actor, 0)))
      : st(N, inst(actor, 0))
  },
  story_op(N, initialize_node).

error(m("Lowlife threatening multiple parties.", N)) :-
  setup(N, market),
  at(N, setup_arg(lowlife, Lowlife)),
  2 <= {
    sp_st(N, relation(threatening, Lowlife, inst(actor, 0))) :
      st(N, inst(actor, 0))
  },
  story_op(N, initialize_node).

error(m("Merchant one doesn't own their goods", N)) :-
  setup(N, market),
  at(N, setup_arg(merchant_one, Merchant)),
  at(N, setup_arg(goods_one, Goods)),
  not st(N, relation(has_item, Merchant, Goods)),
  story_op(N, initialize_node).

error(m("Merchant two doesn't own their goods", N)) :-
  setup(N, market),
  at(N, setup_arg(merchant_two, Merchant)),
  at(N, setup_arg(goods_two, Goods)),
  not st(N, relation(has_item, Merchant, Goods)),
  story_op(N, initialize_node).
```

— s-market.lp —

```
70
   % Potentials:
   s_st(market, relation(c(selling), v(merchant_one), v(goods_one))).
   s_st(market, relation(c(selling), v(merchant_two), v(goods_two))).
   s_st(
75   market,
     property(c(offering_service), v(doctor), c(treat_injury))
   ).

   s_o_st(market, state(c(bored), v(merchant_one))).
80 s_o_st(market, state(c(bored), v(merchant_two))).
   s_o_st(market, state(c(bored), v(noble))).
   s_o_st(market, state(c(bored), v(peasant))).
   s_o_st(market, state(c(bored), v(lowlife))).
   s_o_st(market, state(c(knows_gossip), v(merchant_one))).
85 s_o_st(market, state(c(knows_gossip), v(merchant_two))).
   s_o_st(market, state(c(knows_gossip), v(peasant))).
   s_o_st(market, state(c(knows_gossip), v(lowlife))).
   s_o_st(market, relation(c(accusing), v(noble), c(inst(actor, you)))).
   s_o_st(market, relation(c(accusing), v(noble), v(peasant))).
90 s_o_st(market, relation(c(accusing), v(noble), v(merchant_one))).
   s_o_st(market, relation(c(accusing), v(noble), v(merchant_two))).
   s_o_st(market, relation(c(threatening), v(lowlife), v(merchant_one))).
   s_o_st(market, relation(c(threatening), v(lowlife), v(merchant_two))).
   % TODO: find some way to run the text binding for four cases and then
95 % enable this.
   %s_o_st(
   %  market,
   %  relation(c(threatening), v(lowlife), c(inst(actor, you)))
   %).
```

— —

— s-monster_attack.lp —

```
% monster attack

possible_setup(monster_attack).

setup_argument_create(monster_attack, monster, monster).

s_st(
  monster_attack,
  relation(c(threatening), v(monster), c(inst(actor, you)))
).
```

— s-on_sale.lp —

```
% goods for sale

possible_setup(on_sale).

setup_argument_create(on_sale, merchant, businessperson).
setup_argument_create_n(on_sale, goods, item, 2, 3).

s_st(on_sale, relation(c(has_item), v(merchant), v(goods))).

s_st(on_sale, relation(c(selling), v(merchant), v(goods))).
```

— s-tavern.lp —

```
% a tavern

possible_setup(tavern).

setup_argument_create_n(tavern, merchant, merchant, 0, 1).
setup_argument_create_n(tavern, noble, aristocrat, 0, 1).
setup_argument_create_n(tavern, peasant, laborer, 0, 1).
setup_argument_create(tavern, innkeeper, innkeeper).

% The stuff that the merchant is selling:
setup_argument_n(tavern, goods, item, 0, 2).

error(m("Tavern␣population␣too␣low.", N)) :-
```

```
    setup(N, tavern),
    2 > {
      at(N, setup_arg(merchant, Merchant));
      at(N, setup_arg(noble, Noble));
      at(N, setup_arg(peasant, Peasant))
    }.

error(m("Too␣much␣boredom", N)) :-
  setup(N, tavern),
  3 <= {
    sp_st(N, state(bored, inst(actor, A))) : st(N, inst(actor, A))
  },
  story_op(N, initialize_node).

error(m("Too␣much␣gossip", N)) :-
  setup(N, tavern),
  3 <= {
    sp_st(N, state(knows_gossip, inst(actor, A)))
      : st(N, inst(actor, A))
  },
  story_op(N, initialize_node).

error(m("Noble␣accusing␣multiple␣parties.", N)) :-
  setup(N, tavern),
  at(N, setup_arg(noble, Noble)),
  2 <= {
    sp_st(N, relation(accusing, Noble, inst(actor, O)))
      : st(N, inst(actor, O))
  },
  story_op(N, initialize_node).

error(m("Merchant␣doesn't␣own␣the␣goods", N)) :-
  setup(N, tavern),
  at(N, setup_arg(merchant, Merchant)),
  at(N, setup_arg(goods, Goods)),
  not st(N, relation(has_item, Merchant, Goods)),
  story_op(N, initialize_node).

% Potentials:
s_o_st(tavern, relation(c(selling), v(merchant), v(goods))).
s_o_st(tavern, state(c(bored), v(merchant))).
s_o_st(tavern, state(c(bored), v(noble))).
s_o_st(tavern, state(c(bored), v(peasant))).
s_o_st(tavern, state(c(knows_gossip), v(noble))).
s_o_st(tavern, state(c(knows_gossip), v(peasant))).
s_o_st(tavern, state(c(knows_gossip), v(innkeeper))).
s_o_st(market, relation(c(accusing), v(noble), v(peasant))).
s_o_st(market, relation(c(accusing), v(noble), v(merchant))).
s_o_st(market, relation(c(accusing), v(noble), v(innkeeper))).
```

— s-tavern.lp —

— s-thief.lp —

```
% a thief steals an item

%possible_setup(thief).

%setup_argument_create(thief, perp, inst(actor, thief)).
%setup_argument(thief, victim, actor).
%setup_argument(thief, goods, item).

%% The thief has the goods:
%s_st(thief, relation(c(has_item), v(perp), v(goods))).
%s_st(thief, relation(c(stolen_from), v(victim), v(goods))).
%
%error(m("Thief stole wrong item.")) :-
%  setup(N, thief),
%  at(N, setup_arg(goods, Goods)),
%  at(N, setup_arg(victim, Victim)),
%  0 != #sum {
%    1,story_node(Prev) : story_node(Prev), successor(Prev, Opt, N);
%    -1,story_node(Prev) :
%      story_node(Prev),
%      successor(Prev, Opt, N),
%      st(Prev, relation(has_item, Victim, Goods))
%  }.
```

— s-threatened_innocents.lp —

```
% innocents being threatened

possible_setup(threatened_innocents).

setup_argument_create(
  threatened_innocents,
  attacker,
  tough
). % might be armed
setup_argument_create(threatened_innocents, victim, businessperson).
setup_argument_create(threatened_innocents, goods, treasure).
setup_argument_create_n(threatened_innocents, goods, item, 0, 1).
```

— s-threatened_innocents.lp —

```
% The victim owns the goods:
s_st(
  threatened_innocents,
  relation(c(has_item), v(victim), v(goods))
).

% The attacker is threatening the victim:
s_st(
  threatened_innocents,
  relation(c(threatening), v(attacker), v(victim))
).
```

B.6 Potential Definitions

These files define individual potentials, which are state predicates that suggest actions (for example, being injured suggests getting treatment). These files refine underlying state definitions by specifying things like who is responsible for the state, and who it is good or bad for. They also define urgency and immediacy of potentials, so that characters can have believable priorities (e.g., seeking treatment for an injury being prioritized over buying an available item as opposed to the opposite).

— p-accusing.lp —

```
% accusing
potential(problem, relation, accusing).
pcategory(potential(problem, relation, accusing), urgent).
pcategory(potential(problem, relation, accusing), immediate).
initiated_by(problem, relation, accusing, from).
problematic_for(problem, relation, accusing, to).
```

— —

— p-bored.lp —

```
% bored
potential(opportunity, state, bored).
initiated_by(opportunity, state, bored, inst).

error(m("Bored while embroiled in something interesting.")) :-
  st(N, state(bored, Actor)),
  1 <= {
    at(N, potential(PType, state(S, Actor))) :
      at(N, potential(PType, state(S, Actor))), S != bored;
    at(N, potential(PType, property(P, Actor, V)));
    at(N, potential(PType, relation(R, Actor, Other)));
    at(N, potential(PType, relation(R, Other, Actor)))
  }
  story_op(N, initialize_node).
```

— p-injured.lp —

```
% injured
potential(problem, state, injured).
pcategory(potential(problem, state, injured), urgent).
pcategory(potential(problem, state, injured), persistent).
problematic_for(problem, state, injured, inst).
```

— p-knows_gossip.lp —

```
% knows_gossip
potential(opportunity, state, knows_gossip).
initiated_by(opportunity, state, knows_gossip, inst).
```

— —

— p-offering_service.lp —

```
% offering_service
potential(opportunity, property, offering_service, Action) :-
  action(Action).
initiated_by(opportunity, property, offering_service, Action, inst) :-
  action(Action).
% TODO: auto-cancel

error(m("Offered␣service␣without␣required␣skill.", N)) :-
  st(N, property(offering_service, Offering, Service)),
  at(
    N,
    skill_link(Skill, required, Tool, Service, SInitArg, SOutcome)
  ),
  default_intent(Service, SOutcome),
  initiator(Service, SInitArg),
  not st(N, property(has_skill, Offering, Skill)),
  story_op(N, build_options).

error(m("Offered␣service␣without␣promoting␣skill.", N)) :-
  st(N, property(offering_service, Offering, Service)),
  at(
    N,
    skill_link(Skill, promotes, Tool, Service, SInitArg, SOutcome)
  ),
  default_intent(Service, SOutcome),
  initiator(Service, SInitArg),
  not st(N, property(has_skill, Offering, Skill)),
  story_op(N, build_options).

error(m("Offered␣service␣without␣contest␣skill.", N)) :-
  st(N, property(offering_service, Offering, Service)),
  at(N,
    skill_link(
      Skill, contest, Tool,
      Service,
      between(SInitArg, Opponent),
      either(SOutcome, OtherOutcome)
    )
  ),
  default_intent(Service, SOutcome),
  initiator(Service, SInitArg),
  not st(N, property(has_skill, Offering, Skill)),
  story_op(N, build_options).

error(m("Offered␣service␣without␣contest␣skill.", N)) :-
  st(N, property(offering_service, Offering, Service)),
```

— p-offering_service.lp —

```
    at(
      N,
      skill_link(
        Skill, contest, Tool,
        Service,
        between(Opponent, SInitArg),
        either(OtherOutcome, SOutcome)
      )
    ),
    default_intent(Service, SOutcome),
    initiator(Service, SInitArg),
    not st(N, property(has_skill, Offering, Skill)),
    story_op(N, build_options).

error(m("Offered service without required tool.", N)) :-
    st(N, property(offering_service, Offering, Service)),
    at(
      N,
      skill_link(Skill, required, tool, Service, SInitArg, SOutcome)
    ),
    default_intent(Service, SOutcome),
    initiator(Service, SInitArg),
    not at(N, has_tool_for(Offering, Skill)),
    story_op(N, build_options).

error(m("Offered service without promoting tool.", N)) :-
    st(N, property(offering_service, Offering, Service)),
    at(
      N,
      skill_link(Skill, promotes, tool, Service, SInitArg, SOutcome)
    ),
    default_intent(Service, SOutcome),
    initiator(Service, SInitArg),
    not at(N, has_tool_for(Offering, Skill)),
    story_op(N, build_options).

error(m("Offered service without contest tool.", N)) :-
    st(N, property(offering_service, Offering, Service)),
    at(
      N,
      skill_link(
        Skill, contest, tool,
        Service,
        between(SInitArg, Opponent),
        either(SOutcome, OtherOutcome)
      )
    ),
    default_intent(Service, SOutcome),
    initiator(Service, SInitArg),
```

— p-offering_service.lp —

```
    not at(N, has_tool_for(Offering, Skill)),
    story_op(N, build_options).

error(m("Offered␣service␣without␣contest␣tool.", N)) :-
  st(N, property(offering_service, Offering, Service)),
  at(
    N,
    skill_link(
      Skill, contest, tool,
      Service,
      between(Opponent, SInitArg),
      either(OtherOutcome, SOutcome)
    )
  ),
  default_intent(Service, SOutcome),
  initiator(Service, SInitArg),
  not at(N, has_tool_for(Offering, Skill)),
  story_op(N, build_options).
```

— p-polymorphed.lp —

```
% polymorphed
potential(problem, property, polymorphed, Any) :- any_class(Any).

pcategory(potential(problem, property, polymorphed, Any), urgent) :-
  any_class(Any).

pcategory(
  potential(problem, property, polymorphed, Any),
  persistent
) :-
  any_class(Any).

problematic_for(
  problem,
  property,
  polymorphed,
  Any,
  inst
) :- any_class(Any).

% When you become polymorphed, you become a chicken if you weren't one
% already:
% Changing the type and/or number of an entity is NOT GOOD!
% TODO: Make this work?
```

— p-polymorphed.lp —

```
%at(N, consequence_of(X, O, property(type, Target, chicken))) :-
%  at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
%  not st(N, property(type, Target, chicken)),
%  story_op(N, build_options).

% You also gain the "unintelligent" skill if you didn't already have
% it:
at(
  N,
  consequence_of(X, O, property(has_skill, Target, unintelligent))
) :-
  at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
  not st(N, property(has_skill, Target, unintelligent)),
  story_op(N, build_options).

% You also drop all of your items:
% TODO: Some way to pick them back up!
% TODO: Some concept of item ownership!
at(N, consequence_of(X, O, _not, relation(has_item, Target, Item))) :-
  at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
  st(N, relation(has_item, Target, Item)),
  story_op(N, build_options).


% When you're polymporhed back, you regain your original type and lose
% the unintelligent skill unless it's a class skill for your original
% type:

% See above
% TODO: Make this work?
%at(N, consequence_of(X, O, property(type, Target, Original))) :-
%  at(
%    N,
%    consequence_of(
%      X,
%      O, _not,
%      property(polymorphed, Target, Original)
%    )
%  ),
%  Original != chicken,
%  story_op(N, build_options).

at(
  N,
  consequence_of(
    X,
    O, _not,
    property(has_skill, Target, unintelligent)
  )
```

— p-polymorphed.lp —

```
  ) :-
75  at(
      N,
      consequence_of(
        X,
        O, _not,
80      property(polymorphed, Target, Original)
      )
    ),
    O = {
      class_skill(SomeClass, unintelligent, always) :
85      subclass(Original, SomeClass),
        class_skill(SomeClass, unintelligent, always);
      class_skill(Original, unintelligent, always) :
        class_skill(Original, unintelligent, always)
    },
90  story_op(N, build_options).

  % Becomming polymorphed cancels several other potentials:

  at(
95  N,
    consequence_of(
      X,
      O,
      nullifies,
100   potential(opportunity, state(knows_gossip, Target))
    )
  ) :-
    at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
    at(N, potential(opportunity, state(knows_gossip, Target))),
105 story_op(N, build_options).

  at(
    N,
    consequence_of(
110   X,
      O,
      nullifies,
      potential(opportunity, state(bored, Target))
    )
115 ) :-
    at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
    at(N, potential(opportunity, state(bored, Target))),
    story_op(N, build_options).

120 % yes, polymorphing someone is an extreme way of healing them...
  at(
    N,
```

— p-polymorphed.lp —

```
    consequence_of(
      X,
      O,
      nullifies,
      potential(problem, state(injured, Target))
    )
  ) :-
    at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
    at(N, potential(problem, state(injured, Target))),
    story_op(N, build_options).

  at(
    N,
    consequence_of(
      X,
      O,
      nullifies,
      potential(
        opportunity,
        property(offering_service, Target, Anything)
      )
    )
  ) :-
    at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
    at(
      N,
      potential(
        opportunity,
        property(offering_service, Target, Anything)
      )
    ),
    story_op(N, build_options).

  at(
    N,
    consequence_of(
      X,
      O,
      nullifies,
      potential(opportunity, relation(selling, Target, Anything))
    )
  ) :-
    at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
    at(N, potential(opportunity, relation(selling, Target, Anything))),
    story_op(N, build_options).

  at(
    N,
    consequence_of(
```

— p-polymorphed.lp —

```
      X,
      O,
      nullifies,
      potential(problem, relation(accusing, Target, Anyone))
    )
  ) :-
    at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
    at(N, potential(problem, relation(accusing, Target, Anyone))),
    story_op(N, build_options).

  at(
    N,
    consequence_of(
      X,
      O,
      nullifies,
      potential(problem, relation(threatening, Target, Anyone))
    )
  ) :-
    at(N, consequence_of(X, O, property(polymorphed, Target, Any))),
    at(N, potential(problem, relation(threatening, Target, Anyone))),
    story_op(N, build_options).
```

— p-selling.lp —

```
% selling
potential(opportunity, relation, selling).
initiated_by(opportunity, relation, selling, from).

error(m("Selling␣more␣than␣one␣of␣the␣same␣item␣type.")) :-
  story_node(N),
  st(N, relation(selling, Seller, Item1)),
  st(N, relation(selling, Seller, Item2)),
  Item1 != Item2,
  st(N, property(type, Item1, SameType)),
  st(N, property(type, Item2, SameType)).

at(
  N,
  consequence_of(
    X,
    O,
    nullifies,
    potential(
      opportunity,
```

— p-selling.lp —

```
        relation(selling, Merchant, Item)
      )
    )
  ) :-
    at(
      N,
      consequence_of(X, O, _not, relation(has_item, Merchant, Item))
    ),
    at(N, potential(opportunity, relation(selling, Merchant, Item))),
    story_op(N, build_options).
```

— p-stolen_from.lp —

```
% stolen_from
potential(problem, relation, stolen_from).
pcategory(potential(problem, relation, stolen_from), urgent).
pcategory(potential(problem, relation, stolen_from), immediate).
problematic_for(problem, relation, stolen_from, from).

% Whoever has the item is the thief:
at(
  N,
  initiated_by(
    potential(problem, relation(stolen_from, Victim, Item)),
    Thief
  )
) :-
  at(N, potential(problem, relation(stolen_from, Victim, Item))),
  st(N, relation(has_item, Thief, Item)).

% taking back the item resolves the stolen_from potential:
at(
  N,
  consequence_of(X, O, resolves, relation(stolen_from, Victim, Item))
) :-
  at(N, consequence_of(X, O, relation(has_item, Victim, Item))),
  st(N, relation(stolen_from, Victim, Item)),
  story_op(N, build_options).
```

— —

— p-threatening.lp —

```
% threatening
potential(problem, relation, threatening).
pcategory(potential(problem, relation, threatening), urgent).
pcategory(potential(problem, relation, threatening), immediate).
initiated_by(problem, relation, threatening, from).
problematic_for(problem, relation, threatening, to).
```

B.7 Actor Definitions

These files define a simple hierarchical ontology of actors. The highest levels of the ontology are actually defined in the core file `actors.lp`. Actor definitions can also specify things like skills or items that can be automatically associated with certain types of actors. Actor types are used mainly in the setup files.

— animals.lp —

```
actor_def(chicken, animal, "chicken", singular, either).
actor_def(chickens, animal, "chickens", plural, neuter).
```

— bad_guys.lp —

```
subclass(person, bad_guy).
powerful(bad_guy).

subclass(bad_guy, stealer).
class_skill(stealer, acrobatics, sometimes).
class_skill(stealer, thievery, always).

subclass(bad_guy, tough).
class_skill(tough, fighting, always).
class_item(tough, artificial_weapon, 0, 1).
```

— bad_guys.lp —

```
actor_def(bandits, tough, "bandits", plural, neuter).
class_skill(bandits, thievery, sometimes).

actor_def(thief, stealer, "thief", singular, either).
```

— commmoners.lp —

```
subclass(person, commoner).
powerless(commoner).

% General commoners
actor_def(innkeeper, commoner, "innkeeper", singular, either).
class_skill(innkeeper, literacy, sometimes).
class_skill(innkeeper, storytelling, sometimes).

% Businesspeople
subclass(commoner, businessperson).
class_skill(businessperson, literacy, always).

actor_def(merchant, businessperson, "merchant", singular, either).
class_skill(merchant, tinkering, sometimes).
class_skill(merchant, music, sometimes).
class_skill(merchant, elocution, sometimes).
class_skill(merchant, storytelling, sometimes).
class_item(merchant, treasure, 0, 1).
class_item(merchant, book, 0, 1).
class_item(merchant, artificial_weapon, 1, 1).

% Skilled workers
subclass(commoner, skilled).
class_skill(skilled, literacy, sometimes).

actor_def(healer, skilled, "healer", singular, either).
class_skill(healer, healing, always).
class_item(healer, medicine_chest, 1, 1).


% Laborers
subclass(commoner, laborer).
class_skill(laborer, storytelling, sometimes).

actor_def(peasant, laborer, "peasant", singular, either).
class_skill(peasant, wilderness_lore, sometimes).
```

— —

— monsters.lp —

```
subclass(animal, monster).
powerful(monster).
class_skill(monster, fighting, always).
class_skill(monster, monstrous, always).
class_skill(monster, wilderness_lore, sometimes).

%actor_def(dragon, monster, "dragon", singular, neuter).
%item_def(dragon_scale, treasure, "dragon scale", singular).
%item_def(dragon_claws, natural_weapon, "claws", plural).
%class_item(dragon, dragon_claws, 1, 1).
%class_item(dragon, dragon_scale, 0, 1).
%trophy(dragon, dragon_scale).

%actor_def(leviathan, monster, "leviathan", singular, neuter).
%item_def(
%  leviathan_ambergris,
%  treasure,
%  "leviathan ambergris",
%  singular
%).
%trophy(leviathan, leviathan_ambergris).

% TODO: "group" number
%actor_def(ripper_pack, monster, "pack of rippers", singular, neuter).

actor_def(ogre, monster, "ogre", singular, either).
item_def(sack_of_gold, treasure, "sack␣of␣gold", singular).
%item_def(dragon_claws, natural_weapon, "claws", plural).
%class_item(dragon, dragon_claws, 1, 1).
class_item(ogre, artificial_weapon, 0, 1).
class_item(ogre, sack_of_gold, 0, 1).
trophy(ogre, sack_of_gold).
```

— —

— nobles.lp —

```
subclass(person, aristocrat).
powerful(aristocrat).
class_skill(aristocrat, literacy, always).
class_skill(aristocrat, elocution, sometimes).
class_skill(aristocrat, music, sometimes).
class_item(aristocrat, treasure, 0, 1).
class_item(aristocrat, book, 0, 1).

actor_def(noble, aristocrat, "noble", singular, either).
```

B.8 Item Definitions

These files define a hierarchical ontology of items beneath the high-level categories defined in the core file `items.lp`. Item types are mainly relevant because they can be tools for associated skills. The "`teaches_skill`" predicates in the `books.lp` file are unfortunately impotent at the moment, although once a skill-learning system exists these would provide an additional incentive for acquiring these items.

— books.lp —

```
subclass(item, book).

%item_def(holy_book, book, "holy book", singular).
%teaches_skill(holy_book, prayer).
%tool_for(holy_book, prayer).

item_def(plants_book, book, "book␣of␣herbal␣lore", singular).
teaches_skill(plants_book, wilderness_lore).
tool_for(plants_book, wilderness_lore).
teaches_skill(plants_book, healing).
tool_for(plants_book, healing).

item_def(ancient_grimoire, book, "ancient␣grimoire", singular).
teaches_skill(ancient_grimoire, sorcery).
tool_for(ancient_grimoire, sorcery).
```

— books.lp —

```
%item_def(animals_book, book, "guide to wild beasts", singular).
%teaches_skill(animals_book, wilderness_lore).
%tool_for(animals_book, wilderness_lore).

%item_def(speech_book, book, "book of speeches", singular).
%teaches_skill(speech_book, elocution).

%item_def(music_book, book, "music book", singular).
%teaches_skill(music_book, music).

item_def(medicine_book, book, "book␣of␣medicine", singular).
teaches_skill(medicine_book, healing).
tool_for(medicine_book, healing).

%item_def(legends_book, book, "book of legends", singular).
%teaches_skill(legends_book, storytelling).
%tool_for(legends_book, storytelling).
```

— charms.lp —

```
subclass(item, charm).

item_def(ring, charm, "ring", singular).
%item_def(amulet, charm, "amulet", singular).
%item_def(bracelet, charm, "bracelet", singular).
%item_def(necklace, charm, "necklace", singular).
%item_def(anklet, charm, "anklet", singular).
%item_def(earrings, book, "earrings", plural).
%item_def(hair_clip, charm, "hair clip", singular).
%item_def(belt, charm, "belt", singular).
%item_def(vest, charm, "vest", singular).
%item_def(comb, charm, "comb", singular).
```

— instruments.lp —

```
subclass(item, instrument).

%item_def(harp, instrument, "harp", singular).
%item_def(mandolin, instrument, "mandolin", singular).
%item_def(flute, instrument, "flute", singular).
item_def(oboe, instrument, "oboe", singular).
%item_def(trumpet, instrument, "trumpet", singular).
%item_def(accordian, instrument, "accordian", singular).

tool_for(IType, music) :- item_def(IType, instrument, Name, Number).
```

— tools.lp —

```
subclass(item, tool).

%item_def(hammer, tool, "hammer", singular).
%tool_for(hammer, tinkering).

%item_def(pliers, tool, "pliers", plural).
%tool_for(pliers, tinkering).

item_def(medicine_chest, tool, "medicine chest", singular).
tool_for(medicine_chest, healing).
```

— treasure.lp —

```
subclass(item, treasure).

%item_def(coins, treasure, "coins", plural).
%item_def(gem, treasure, "gem", singular).
%item_def(spices, treasure, "spices", plural).
item_def(perfume, treasure, "perfume", plural). % TODO: Mass nouns?
```

— —

— weapons.lp —

```
subclass(item, weapon).

subclass(weapon, artificial_weapon).
subclass(weapon, natural_weapon).

%item_def(sword, artificial_weapon, "sword", singular).
item_def(spear, artificial_weapon, "spear", singular).
%item_def(javelin, artificial_weapon, "javelin", singular).
%item_def(dagger, artificial_weapon, "daggers", plural).
%item_def(axe, artificial_weapon, "axe", singular).
%item_def(mace, artificial_weapon, "mace", singular).
%item_def(glaive, artificial_weapon, "glaive", singular).
%item_def(bow, artificial_weapon, "bow", singular).
%item_def(rapier, artificial_weapon, "rapier", singular).

tool_for(IType, fighting) :-
  item_def(IType, artificial_weapon, Name, Number).
tool_for(IType, fighting) :-
  item_def(IType, natural_weapon, Name, Number).
```