Defining Operational Logics

Michael Mateas and Noah Wardrip-Fruin
Expressive Intelligence Studio
Department of Computer Science
University of California, Santa Cruz
1156 High St, MS:SOE3
Santa Cruz, CA 95064 USA
michaelm, nwf @soe.ucsc.edu

ABSTRACT
Much analysis of games focuses, understandably, on their mechanics and the resulting audience experiences. Similarly, many genres of games are understood at the level of mechanics. But there is also the persistent sense that a deeper level of analysis would be useful, and a number of proposals have been made that attempt to look toward a level that undergirds mechanics. This paper focuses on a particular approach of this sort—operational logics—first proposed by Noah Wardrip-Fruin (2005) and since then discussed by authors such as Michael Mateas (2006) and Ian Bogost (2007). Operational logics connect fundamental abstract operations, which determine the state evolution of a system, with how they are understood at a human level. In this paper we expand on the concept of operational logics, offering a more detailed and rigorous discussion than provided in earlier accounts, setting the stage for more effective future use of logics as an analytical tool. In particular, we clarify that an operational logic defines an authoring (representational) strategy, supported by abstract processes or lower-level logics, for specifying the behaviors a system must exhibit in order to be understood as representing a specified domain to a specified audience. We provide detailed discussion of graphical and resource management logics, as well as explaining problems with certain earlier expansions of the term (e.g., to file handling and interactive fiction's riddles).

Author Keywords
operational logics, mechanics, code studies, unit operations, software studies

INTRODUCTION
Games can be studied at many levels of abstraction, and in relationship to many social, cultural, psychological, media-theoretic, and formal phenomena. This multivalent nature of games contributes to the methodological pluralism seen in the game studies community. Montfort and Bogost introduce a five level model for the analysis of digital artifacts as a framework for organizing this plurality of methods and viewpoints [10]. The levels range from *reception/operation* through *interface, form/function, code* and finally *platform*. In the context of games, the first three levels are the traditional purview of game studies, with a focus, respectively, on the reception of the artifact in the social and cultural field; the game interface and the visual and auditory representational strategies; and the mechanics, rules, formal, and simulational elements of games. The last two levels, code and platform, have been far less studied, with the code level focusing on deep readings of software structures, and platform studies focusing on deep readings of the hardware and system abstractions that underlie software. In this paper we present *operational logics* as a unit of analysis centered at the code and platform levels, but that connects technical implementation strategies with authorial and audience meanings. Operational logics are the fundamental abstract operations—with effective interpretations available to both authors and players—that determine the state evolution of the system and underwrite the gameplay. As such they provide "deep cores" that bind together issues ranging from the platform to reception level for specific representational domains.

Wardrip-Fruin introduced the notion of operational logics (though did not name them as such until his 2006 dissertation [14]) in a 2005 paper exploring the different ways in which interactive texts can be made *playable* [13]. To briefly recapitulate the argument of that paper, Wardrip-Fruin argues that, rather than focusing on the question of which interactive experiences are or are not games, it is more fruitful to focus on the analytic category of the playable, and to ask of various interactive experiences "in what way is this experience playable?" He then introduces the notion of operational logics as a way of analyzing how a given interactive experience structures the space of play, particularly focusing on two families of operational logics: graphical logics and textual logics. Wardrip-Fruin further developed the idea of operational logics in his dissertation [14] and in *Expressive Processing* [15], though not providing a wholly consistent account across these three treatments. Other authors, such as Mateas [7] and Bogost [4], have further expanded on the notion of operational logics, but generally as a niche discussion in the context of a larger project; even Wardrip-Fruin, in his three different

Breaking New Ground: Innovation in Games, Play, Practice and Theory. Proceedings of DiGRA 2009

accounts, always introduces operational logics in the context of making a larger argument. In this paper we seek to define operational logics as a first-class unit of analysis, offering a more detailed and rigorous discussion than provided in earlier accounts. We discuss graphical and resource management logics in some detail, outline a number of limit cases for the concept of operational logics, and then provide and expand a full definition of the concept. Finally, we discuss how this work sets the stage for the use of operational logics in connecting a number of issues in game studies to the emerging area of software studies, offering a powerful combination of technical grounding with authorial and audience concerns.

OPERATIONAL LOGICS

In this section we introduce operational logics with an informal discussion of graphical and resource management logics. The next section provides a more formal definition of operational logics.

Graphical Logics

To introduce graphical logics, consider two early videogames, *Spacewar!* (1962) and *Pong* (1972). In *Spacewar!*, two players control the flight of virtual spacecraft. Player controls are clockwise and counterclockwise rotation, thrust, fire and hyperspace (which jumps the ship to a random location on screen). Each player tries to shoot the other while avoiding being hit by enemy fire or crashing into the enemy ship, while navigating within the gravitational field of a star in the center of the screen. A ship is destroyed when it collides with a bullet, the other ship, or the star in the center.

In *Pong*, a simple table tennis simulator, two players control paddles that are able to move vertically along the goal lines at the left and right edges of the screen. A simple computer controlled opponent can be substituted for one of the players. The players volley a ball back and forth; when either player misses (the ball fails to collide with the paddle), the other player scores. The angle at which the ball reflects from the paddle on collision depends on where the ball hits the paddle, reflecting at sharper angles towards the ends of the paddle.

Though the fictional worlds of space warfare and table tennis are quite distinct from each other, and the two games would typically be classified as belonging to distinct game genres (the shoot-em-up and sports simulation respectively), there are strong similarities between them. Both games represent the *movement* of simulated objects (space ships, projectiles, balls, paddles) by constantly erasing and redrawing collections of pixels (or, in the original *Spacewar!,* collections of vectors) on the screen. Major gameplay events occur when two virtual objects *collide*, naively, when pixels/vectors belonging to one virtual object are drawn at the same screen location as pixels/vectors belonging to another virtual object. The simulated movement of objects is influenced by an underlying *physics* simulation, giving spaceships inertia and making them subject to gravity, or changing the angle of a ball's trajectory as a function of where it hits the paddle (the physics simulation does not have to obey the physics of our world). These similarities constitute *operational logics*, which we, as discussed above, define as fundamental abstract operations—with effective interpretations available to both authors and players—that determine the state evolution of the system and underwrite the gameplay. Both *Spacewar!* and *Pong* exhibit canonical graphical logics, specifically logics of movement, collision detection and physics. There's an additional logic operating in *Spacewar!*: navigation. Navigation involves player-controlled (simulated) movement of a (virtual) object in a represented space, in this case the movement of space ships in a two-dimensional field. *Pong* can be said to make use of navigation logic, though it is the degenerate case of navigation along a line segment.

Graphical logics, which underwrite the simulation of spaces and objects within spaces, are the most common logics employed in videogames, found at the heart of everything from the playful *Mario* games to the gritty *Grand Theft Auto* franchise. Consider the logic of collision detection. Collision detection is operating when Pac-Man eats a dot or power pill, or is touched by a ghost; it's operating when the player is unable to move through walls in *Doom*, picks up health packs, hits demons with weapons, and is hit by demon attacks; it's operating when the player's katamari picks up or bounces off an object in *Katamari Damacy*. Similar examples can be enumerated for movement, navigation, and physics, across the vast majority of contemporary and historical games.

Resource Management Logics

Before discussing some general properties of operational logics, we will examine one more family of logics in this section. Consider two other early computer-based games, *Hamurabi* and *Oregon Trail*. Richard Merrill wrote a land management game called *The Sumer Game* in 1969 in FOCAL for the PDP-8. David Ahl ported it to BASIC for the PDP-8, and later published an expanded version of the program, renamed *Hamurabi*, in his best-selling book *BASIC Computer Games* [2]. It is in this version that the game became well-known among personal computer hobbyists, who would type in the program to play the game (and, because the game was distributed as source code, often tinker with the program to explore variants). In *Hamurabi* the player takes the role of "Hamurabi," the ancient king of "Sumeria."[1]

In this turn-based, text-based game, the player makes a series of decisions about land management and the

[1] *Hamurabi* is presumably a shortening of Hammurabi to fit in an eight-character file name limit. The game invites players to "Try your hand at governing ancient Sumeria" (rather than Sumer).

allocation of grain. During each of 10 rounds, each representing one year, the player decides how many acres of land to buy or sell, how many bushels of grain to feed the people (20 per person required to avoid starving anyone), and how many acres of land to plant (one bushel of grain plants 2 acres)[2]. At each step, the game state consists of the population, the number of acres owned, the number of bushels of grain owned, and the current price of land in bushels per acre.

As one would expect, feedback loops and tradeoffs exist between the various player choices. For example, you can only plant as many acres as ten times your population (each person can farm 10 acres), and your population grows as function of the number of acres owned and the amount of stored grain—though more acres and grain are required to grow the population by the same absolute value as the population gets bigger. Further, random events can strike, such as plagues that eliminate half your population and rats eating your stored grain. This introduces further tradeoffs into the system. For example, the amount of stored grain destroyed if a rat infestation strikes increases with the amount of total grain stored. This encourages the player to minimize the amount of stored grain in order to minimize exposure to this risk, yet the player must also maximize the amount of stored grain in order to grow their population. Finally, there is random fluctuation in the yield per acre (number of bushels produced per acre planted) as well as in the price per acre, introducing an element of uncertainty and encouraging the player to try strategies such as buying land low and selling high, which can become necessary in low-yield years in order to avoid mass starvation.

In *Oregon Trail* the player guides a wagon that is traversing the Oregon Trail from Independence Missouri to Oregon's Willamette Valley in 1848. The player's goal is to reach the end of the trail while minimizing the number of party members lost along the way. *Oregon Trail* was developed by Don Rawitsch, Bill Heinemann and Paul Dillenberger to teach school children about the realities of pioneer life along the Oregon Trail in the mid 19th century. The game exists in several versions, with the original mainframe versions developed in 1971 and released in 1974, and the Apple II versions that most players are familiar with first released in 1980 with an updated version in 1985.

At the beginning of the game, players are given a budget for equipping their wagon. Purchase decisions include sets of clothes (influences health as a function of weather), wagon spare parts (wagons break down), food, bullets (can be used to hunt as another way of getting food), and so on. Once the trip is underway, the player is able to adjust the pace at which the wagon moves, rate of consumption of food, whether to trade (during trading, the player is offered swaps for items in her inventory), the opportunity to hunt (use bullets to try to hit game to supplement food), the opportunity to rest (costs days, and therefore food, but can improve the health of party members and rest oxen) and, at the occasional fort, opportunities to purchase items. The player must manage tradeoffs between the health of the party members, consumption of food, and number of miles traversed per day. Additionally, the player must contend with a variety of random events, such as party members becoming sick, thieves stealing items, wagon parts breaking down, and so forth.

Though the fictional worlds of land management in ancient Sumer and 19th century pioneer life along the Oregon Trail are quite distinct, and the two games would typically be classified as belonging to distinct game genres (the tycoon game and the simulation game respectively), there are strong similarities between them. Both games represent *acquiring, using, and transforming resources* such as food and money by representing the amount of each resource currently possessed by the player as a number, and defining sources that produce resources, sinks that consume resources, and transformers that convert one resource into another. *Random events* within the fictional world can consume or produce resources, or change rate constants for production, consumption, and transformation; the probability of a random event occurring is modulated by the amount of resources possessed and/or by the current rates of production, consumption, and transformation. *Resource allocation* involves the player selecting among different sources, sinks, or transformations to apply to different resources, selecting either absolute values or rates. These similarities constitute resource management logics. Collectively, these logics provide the fundamental abstract operations and effective interpretations for computational representations of resource acquisition and spending.

Resource management logics, while not as ubiquitous in games as graphical logics, are still found at the heart of many videogames, including turn-based and real-time strategy games, tycoon games, city-management games and god games.

Observations

Given these two different families of operational logics as examples, we can now suggest some general properties that characterize operational logics:

- Operational logics are more general, and more fundamental, than game rules or mechanics. *Doom* and *Super Mario Brothers* have very different rules, but both make heavy use of graphical logics. The graphical logics provide the more fundamental constraints and affordances on top of which rules are defined.
- Operational logics provide strategies of computational representation. Graphical logics are concerned with representing movement, virtual touch, the effects of physics, and so forth. Resource management logics are concerned with representing finite resources, resource

[2] The simulation constants reported here are based on David Ahl's 1973 BASIC version for the PDP.

production and consumption, resource tradeoffs, and so forth. As representations, operational logics operate for both authors and audiences, simultaneously providing representational tropes for authors and actionable representations for players. An algorithm or concrete implementation of an algorithm is not enough to be an operational logic, as the code on its own does not specify a representational strategy.

- Operational logics are fundamentally computational. They provide specific strategies for procedural representation. Consider the example of representing moral decision making, as is found in games such as *Star Wars: Knights of the Old Republic* or *Fable.* Simply identifying the game design trope of "moral decision making", or even, more specifically, "offer players choices between good and evil options," is insufficient to have identified an operational logic. In order to, in this case, develop logics of moral decision making, the logics would have to provide strategies for mapping representations of moral choices and their effects into a computational representation; an example of such a logic is the moral alignment logic of representing moral status as a point in a one or more dimensional space, with different player actions moving the point around within the "morality space."

In the next section, we provide a more formal definition of operational logics that further unpacks these general properties.

FORMALIZING OPERATIONAL LOGICS

Since Wardrip-Fruin's original description of operational logics [13], a number of authors have built on this concept, including both Wardrip-Fruin [14, 15] and Bogost [4]. While these authors usefully deploy operational logics in their analyses, they also provide examples of logics that muddy the concept and risk diluting its analytic power. In this section we examine a couple of problematic (and useful) examples of operational logics to motivate the need for a more formal treatment, then, extending the treatment presented in Wardrip-Fruin [14], provide a definition that sharpens its use as an analytic tool.

Overly Broad Uses

Bogost, after introducing the graphical and textual logic examples provided in Wardrip-Fruin's dissertation [14] in the context of playable media, goes on to provide additional, general computing examples [4]:

> Outside of videogames, procedural tropes often take the form of common models of user interaction. Elements of a graphical user interface could be understood as procedural tropes, for example, the scrollbar or push-button. These elements facilitate a wide range of user interactions in a variety of content domains. Operational logics for opening and saving files are also reasonable candidates; these tropes encapsulate lower-level logics for getting handles to filestreams and reading or writing byte data. We might call the former group of procedural tropes *interface logics,* and the later *input/output (I/O) logics*.

Here we see a couple of interesting moves being made. First, operational logic has slid into the more general term *procedural trope*, any commonly employed computational pattern. Second, the user interface examples provide a useful reminder that games are not the only computational media form in which an author expresses intent through providing operational interpretive affordances to an audience. Thus, we can expect operational logics to be useful wherever we find computational media.

However, the file system examples become problematic. Yes, file systems are an abstraction, one that is provided by system programmers, for use by application programmers. But every computational system is a dizzying tower of abstractions, with processes defined at one level underwriting the abstraction defined at the next level. Yet abstraction is not a unitary phenomenon—it rather involves distinct phenomena such as functional abstraction and language abstraction. If operational logic is stretched to account for the ubiquitous and multi-faceted phenomenon of abstraction, it risks losing any analytic and explanatory power.

Further, moving outside of computer science into the interdiscipline of software studies, the shift from operational logic to the more general category of "procedural trope" loses analytic power to account for specific processes of meaning making. One of the central questions of software studies is "How does computation mean?" There is no single answer to this question, but rather many different answers to how software means in different contexts, and thus many different types of "procedural tropes." In order for operational logic to be a useful analytic category, it needs to describe a specific kind of "procedural trope," a specific mechanism of meaning.

Now consider the following passage from Wardrip-Fruin [14]. In this passage, he has just introduced the text-based interactive fiction (IF) of Infocom, and is describing the logics operating in these works.

> These textual games used a different operational vocabulary from video games, supported by different logics, including arguably-literary logics that operate at relatively abstract levels (which others have discussed in terms of the literary riddle (Montfort, 2003) and the performative quest (Aarseth, 2004)). These more abstract logics had to be supported by lower-level ones, including those for textual parsing and reply, world simulation, and progress tracking.

Here a different problematic emerges in identifying operational logics in the reference to the literary riddle. Montfort argues that the pleasures of the puzzle-like

situations presented to players of IF can be understood in reference to the figure of the literary riddle [8]. That is, the same kinds of conceptual shifts required of readers to solve a literary riddle (or understand a presented solution) are required of players of IF. In this quote, the riddle-like operation of IF is identified as an abstract operational logic. But in what sense is there computational support for this riddle-like nature?

Within interactive fiction there are operational logics supporting navigation, object manipulation, and other aspects of a simulated world. Whether a particular piece of interactive fiction requires the player to make the conceptual shifts associated with the riddle, however, is "merely" an issue of authorial skill in deploying the operational logics in combination with good writing. If all developed authoring approaches for computational media fall under the sign of operational logic, then the concept devolves, losing explanatory power. Putting it crudely, if the filesystem abstractions fail at being operational logics because they are code without meaning (they are decoupled from author and audience), the IF riddle fails at being an operational logic because it is meaning without code.

An Expanded Definition

We are now ready to state a more formal definition of operational logic that captures the intuitive properties of logics described in the previous section, and can make principled distinctions to avoid the problems described above:

> An operational logic defines an authoring (representational) strategy, supported by abstract processes or lower-level logics, for specifying the behaviors a system must exhibit in order to be understood as representing a specified domain to a specified audience.

We unpack the elements of this definition below, starting with abstract process. An abstract process is a specification for how a process operates. An abstract process for determining whether two visual representations (collections of pixels) overlap is to declare an overlap has occurred if any of the pixels of the two objects occupy the same location. An abstract process for randomly adding or subtracting an amount from a numeric value is to define a set of positive and negative numbers and non-deterministically select one of these numbers to add to the value (we will make use of this abstract process in a discussion of the random event resource management logic below). Almost any abstract process could be carried out through ("implemented in") human effort as well as automatic computation, but for many contemporary works of digital media the calculations would be intractable to manually carry out. Abstract processes describe sets of algorithms whose behavior meets the abstract specification.

Implemented processes are concrete realizations of abstract processes. Some implementations are through human effort, others through automatic computation. implementations make different tradeoffs in the amount of memory and processing time they require, which can determine whether work of a particular sort is possible with the available resources (e.g., certain approaches for representing statistical models balloon much more quickly than others) and fast enough to be responsive (e.g., fast 3D rendering enables fluid interactive navigation of virtual space). Implementation specifics may also alter the results of processes in ways that can appear profoundly different on the surface (e.g., using a piece of data as a model of its own statistics may meet the abstract definition of a statistical technique while producing very different results from an external model). An implemented process is a specific algorithm that meets the specifications of an abstract process.

An operational logic is not just a naked process, but provides a strategy for mapping a desired representational effect onto the process; that is, it defines a unit of authorial and interpretive affordance [6]. Interpretive affordances support the interpretations an audience makes about the operations of a computational system, conditioning the meanings negotiated between author and audience. Interpretive affordances provide resources both for creating a mental model of the operation of the system, and additionally, in the case of an interactive systems, for supporting intentions for action. The authorial affordances of a computational system are the "hooks" that the system architecture (processes and data) provides for an author to inscribe their authorial intention in the machine. Different architectures provide different relationships between authorial control and the combinatorial possibilities of computation. Operational logics are the units that provide effective authorial affordances for specific representational tasks. To define an effective authorial affordance, we first need to understand the double meanings that all computational systems participate in.

Every computational system can be read as a static text and executed as a process. As a static text, a computational system is a description of an implemented process. The process may be described using code in a particular programming language, a particular configuration of hardware elements, or precise natural language description. The execution of a process description is purely mechanical, that is, it requires no processes of human meaning making. As a pre-interpreted machine, an executing process consists entirely of complex causal flows mediating changes in abstract state.

The executing process gains a layer of human meaning through interpretations of outputs and of the relationship between inputs and outputs. The static, textual description of the process simultaneously specifies an uninterpreted, meaningless machine (the executing process), and represents properties of the (desired) human interpretations of potential executions. This raises a conundrum: how can process descriptions be simultaneously amenable to the

uninterpreted manipulations of computational systems (execution) and to serving as signs for human subjects? The answer is that the literal process description (the "code machine") must be coupled with a collection of rhetorical strategies for talking about the static process description and its executions (the "rhetorical machine"). For example, the rhetorical machine associated with the process description of a planner supports the use of language such as "goal," "plan," and "knowledge" to simultaneously refer to specific formal entities within the process, and to make use of the systems of meaning these words have when applied to human beings. In fact, these rhetorical structures are also important to the initial construction of the planner itself, by its author(s).

Now we can define an *effective authorial affordance*: a computational media system exhibits effective authorial affordances for a specific representational task when the internal structures and processes made available by the system are coupled with rhetorical strategies such that the author is able to represent desired interpretive affordances in the static process description, and, when executed, the process does indeed provide the desired interpretive affordances for the audience. *An operational logic is precisely such a packaging of a rhetorical strategy*—"an authoring (representational) strategy"—*with a process*—"supported by abstract processes or lower-level logics"—*in order to provide an effective authorial affordance*—supporting the specification of "the behaviors a system must exhibit in order to be understood as representing a specified domain to a specified audience."

Examples Revisited

Now we will look again at a couple of our previous examples of operational logics in light of this more formal definition. Consider the movement graphical logic.

- The abstract process is "continuously redraw collections of pixels, erasing the previous drawing between each redraw, while applying a small, identical offset to the screen location at which each pixel is drawn."
- The domain is the representation on the screen of the movement of physical objects.
- The representational strategy is "in order to represent on the screen a physical object moving along a trajectory, make the collection of pixels be an image of the object, specify a sequence of offsets along the desired trajectory, using larger offsets to represent faster movement."
- The specified audience is an audience that is primed to interpret the continuous redrawing of an image as the movement of an object. While this interpretation can be scaffolded, like any representational system it is ultimately conventional.

Fortunately for the designers of videogames, this representation is a so broadly understood convention that the specified audience is in some sense "everybody" (one can make the argument that this representational convention is supported by features of the human vision system). But operational logics can partake of interpretive conventions that are understood by narrow audiences. Consider the random event resource management logic.

- The abstract process is "given a collection of one or more labeled numeric values, define a set of positive and negative numbers each associated with a label and a sensory representation that will make use of other operational logics, define a probability distribution over these numbers, and non-deterministically select a number according to the distribution, displaying its associated representation and adding it to the number associated with its label."
- The domain is the representation of random events within a fictional world that impact the amount of managed resources.
- The representational strategy says "in order to represent randomly occurring fictional events impacting managed resources, associate each resource with a numeric value, associate 'good' fictional events with positive numbers, 'bad' fictional events with negative numbers, create a sensory representation of the fictional event, and define a probability of the event occurring, generally associating smaller probabilities with larger positive and negative resource changes."
- On the audience interpretive side, this representation is entirely conventional for players of simulation games (and thus most videogame players).

It is possible to now clearly see how both the file system and riddle examples introduced at the beginning of this section fail to qualify as operational logics. For the file system operations, while they provide abstractions useable by programmers, abstractions that in fact might be used as part of a process description, they do not participate in the computational media ecosystem of authorial intention mediated through a computational representation to an audience. While one can talk about meaning in the context of file systems, it is not this kind of meaning making. On the other hand, while the trope of the riddle is certainly a representation an author can intend, there is no abstract process nor representational strategy supporting the riddle. When the trope of the riddle successfully operates in an IF, it is because the author has made idiomatic use of many logics that are conventional in IF, but these idioms are not yet codified to the point of providing effective authorial affordances.

In the two examples above, pulling apart the abstract process from the representational strategy resulted in awkward, and perhaps overly abstruse-sounding, descriptions of the processes and strategies. This is precisely because an operational logic actually binds process and strategy into a unified whole; the strategy provides the language for talking about the process. When you pull the two halves apart, it reveals the hidden, inner complexity of the union. Especially for logics that are as deeply conventional in computational media production as graphical and resource management logics, it can feel as if the process is intrinsically about the representational domain. Thus "move objects by continuously redrawing them" and "random events add and subtract from managed resources" can feel like unproblematic descriptions of processes. But processes, on their own, have no intrinsic representational power (and can thus be quite difficult to talk about in the abstract). A representational strategy functions precisely because it provides a way of talking and thinking about a process. But the original establishment of this mapping between process and rhetoric takes work. Fundamental innovation in computational media involves doing the work of establishing new strategies for mapping representational effects onto (potentially new) processes.

DISCUSSION

In this paper we have gone to some length to give examples of operational logics (and examples that aren't logics), define the term, and expand on the definition. We do this not from a love of terminology, but because we believe that a relatively precise notion of operational logics makes it possible to discuss important issues with connections "under the hood" of games. In particular, we believe operational logics provide one of the most potentially fruitful ways to bring game studies together with software studies. We sketch a few directions for possible work of this sort here.

Operational logics provide a useful analytical tool for understanding constraints on game mechanics and game rules, two of the central topics of game studies. (While these topics are not always clearly distinguished, Sicart [12] and others define mechanics in relation to player/agent actions, while rules are more general.) Discussion in game design and game scholarship often identifies mechanics and rules as central sites for innovation. But the space of possible innovation is not free—it is fundamentally constrained by the operational logics available. This is because operational logics deeply underwrite mechanics and rules.

Given this, considering operational logics can clarify what differentiates certain types of alternative game creation. Wardrip-Fruin et al's *Regime Change* (2004) is driven by the serial-ordering logic of Markov chains, which he argues is an appropriate logic for textual play. Mateas and Andrew Stern's *Façade* (2005) required the development of new interpersonal logics that could support structures such as its "affinity game." Arguably, the definition and development of new types of logics presents an important alternative to creating games, commercially or otherwise, that primarily depend on longstanding spatial and resource management logics. In undertaking such expansions, some necessary logics may already have formalized abstract and implemented processes (further work in textual logics may draw on computational linguistics) while others may require novel computational models (e.g., to broaden interpersonal logics to include friendship or humor).

This is not to say that interesting innovation can't involve working within established logics. It is certainly possible to define interesting new kinds of mechanics and rules on top of existing logics. In fact, such work can arguably lead to the production of new logics. Consider *Passage* (2007) by Jason Roherer. In this game graphical logics are used as the basis of spatial mechanics associated with metaphors about life. For example, collision detection is used to determine whether the character's journey will take place with a partner or alone. This isn't just part of the fictional world—solo characters can explore parts of the world that couples can't. As time passes, the player character inevitably grows old and dies. Nick Montfort argues that in *Passage* choosing to do things like explore the world, perhaps searching for hidden treasure, become as much about how one lives one's life as about spatial exploration and game accomplishment [9]. One can imagine such currently-unusual uses of graphical logics eventually becoming well-understood, to the point that the underlying abstract processes become recognized as participating in two kinds of operational logics: both the current graphical/spatial logics and another in which the shifting position of elements on the screen is actually understood as the making of non-spatial life decisions. This seems unlikely in the specific case of *Passage,* but it explains part of what seems unusual and full of potential about the work.

On a different note, there is a non-obvious issue with the fact that operational logics are defined in terms of abstract rather than implemented processes. It raises the specter that bogus operational logics can be defined which are not actually underwritten by implementable computational processes. However, in order to qualify as an abstract process, there must be an implemented process that meets the specification of the abstract process. We are not allowed to cheat by defining an operational logic around "magic" abstract processes such as "generate the same kinds of emotional responses to interpersonal interactions that people do." Unless an operational logic has actually been demonstrated to *operate* by creating a computational media artifact that achieves the claimed representational effect using the claimed process and a strategy, then it doesn't exist. Once demonstrated, operational logics support a level of analysis that is safely computational (no cheating), without having to drop to the level of the code in which the processes are implemented. This differentiates the approach from one such as Mark Marino's "Critical Code Studies"

[5], eliminating the need to acquire (rarely available) source code access. On the other hand, operational logics are more specific to computational systems than concepts such as Bogost's of the "unit operation," which creates a foundation for "any medium—poetic, literary, cinematic, computational" to be "read as a configurative system, an arrangement of discrete, interlocking units of expressive meaning" [3]. Instead, operational logics are an implementation-independent way of talking about system architectures and their fundamental actions.

Finally, operational logics are also a tool for comparing and examining individual works. For example, while at the level of interface actions the mechanics of *Façade* may seem similar to those of Joseph Weizenbaum's (1966) *Eliza/Doctor*—each supports conversational interaction, with the player allowed arbitrary textual input—an examination of the underlying logics reveals a stark contrast. *Eliza/Doctor*'s conversational logic is one of transformation, turning each audience statement into its own reply (though designed to avoid immediate detection). *Façade,* on the other hand, interprets audience text as discourse acts within the game's social space. Further, looking at the relationships between operational logics within a system can reveal where interpretive energy is best spent (and help avoid error). For example, while Janet Murray (1997) and Espen Aarseth (1997) both understand the interaction mechanics of the *Tale-Spin* system, their interpretations ignore the fact that planbox-based character simulation is its central operational logic, leading both to missed opportunities and inaccuracies.[3]

In short, operational logics can help us see the structure of games and the field more deeply and broadly, imagine the future in constructive new ways, and interpret individual works more accurately. By employing an analytic unit that cuts across multiple levels of analysis to describe the relationship between implementation and representation, we can begin to map the space of possibility for procedural representation that underlies games and all computational media work.

[3] In particular, Murray critiques *Tale-Spin* for plot structures that are too abstract (in fact, it contains no plot structures) and devotes the next chapter to simulated characters (but makes no mention of *Tale-Spin*'s interesting planning logic for character behavior) [11]. Aarseth, on the other hand, uses *Tale-Spin* as an example in an argument that machines should not be forced to simulate human narrators, when the absence of simulated narration is a primary critique of *Tale-Spin* from those who engage its operations. Aarseth's missed opportunity is to consider *Tale-Spin* as a step toward the new genres for which he is calling [1].

REFERENCES

1. Aarseth, E. *Cybertext: Perspectives on ergodic literature.* Johns Hopkins University Press, Baltimore, 1997.
2. Ahl, D. *Basic Computer Games.* Workman Publishing, New York, 1978.
3. Bogost, I. *Unit Operations: An approach to videogame criticism.* MIT Press, Cambridge MA, 2006.
4. Bogost, I. *Persuasive Games: The expressive power of videogames.* MIT Press, Cambridge MA, 2007.
5. Marino, M. 2006a. "Critical code studies." *Electronic Book Review* 2006. Available at www.electronicbookreview.com/thread/electropoetics/codology.
6. Mateas, M. "Expressive AI: A semiotic analysis of machinic affordances." 3rd Conference on Computational Semiotics and New Media, University of Teeside, UK. September 10-12, 2003.
7. Mateas, M. "Making games about people: AI and game design." Keynote Speaker, Medi@Terra: Gaming Realities, Athens, Greece, October 4-8, 2006.
8. Montfort, N. *Twisty Little Passages: An approach to interactive fiction.* MIT Press, Cambridge MA, 2003.
9. Montfort, N. "PvP: *Portal* versus *Passage.*" Grand Text Auto 2008. Available at grandtextauto.org/2008/02/24/pvp-portal-versus-passage/
10. Montfort, N. and Bogost, I. *Racing the Beam.* MIT Press, Cambridge MA, 2009.
11. Murray, J. *Hamlet on the Holodeck*. Free Press, New York, 1997.
12. Sicart, M. "Defining game mechanics." *Game Studies* 8. 2. Available at gamestudies.org/0802/articles/sicart
13. Wardrip-Fruin, N. "Playable media and textual instruments." *Dichtung Digital* 1, 2005. Available at http://www.dichtung-digital.com/2005/1/Wardrip-Fruin
14. Wardrip-Fruin, N. "Expressive Processing: On process-intensive literature and digital media," PhD dissertation, Brown University, 2006.
15. Wardrip-Fruin, N. *Expressive Processing: Digital fictions, computer games, and software studies.* MIT Press, Cambridge MA, 2009.