Homework 5: Pacman versus the Ghosts

Due October 12th at 10pm

Part 1: Adversarial bread search

Concentration is a game where the players take turns listing items in a particular category. If you cannot think of an item in the category within a time limit, you lose. You also lose if you say an invalid item or repeat an item that has previously been named.

There are many variants of this game. We will focus on a variant where each player's item must start with the same letter as the last item ended with.

Task 1: Playing bread

Assume that the category is bread. You and your fellow player know only 6 kinds of bread: pistolet, wrap, paratha, arepa, toast, tortilla. **Draw out the game tree for this small vocabulary list.** Assume that the start state is the empty string: you can play any of the words after this.

Task 2: Minimaxing bread

The basic minimax strategy works fine for Concentration if the vocabulary is small. Can you think of a way of improving it for larger vocabulary sizes?

Part 2: Pacman and enemies

We return now to our Pacman game. In Part 3, you will implement adversarial search strategies for Pacman. Unlike the Pacman game you played in Part 1, you now have adversaries: ghosts that will eat you if you make contact with them. **Part 2 involves the multiagent folder.**

Refer to the UC Berkeley assignment for instructions on how to run the Pacman game simulation: UC Berkeley Project 2.

Task 1: Minimax

Implement minimax for Pacman in multiAgents.py. This involves generalizing our minimax algorithm to games with two players: there are now multiple min layers for each max, since there are multiple ghosts.

You'll probably need to write some helper functions. I structured my approach as follows:

getAction: calls self.minimax and returns the result minimax: handles the logic for switching between layers in the tree. Takes self, gameState, player, and depth as arguments. Figures out if the game is going to continue and calls getValue to find the best value to take. Returns a pair consisting of the best value and its associated action.

getValue: handles a single layer in the tree. Takes self, gameState, player, and depth as arguments. For the given state, considers all actions and recursively calls minimax to calculate their values. Returns a pair consisting of the best value and its associated action.

You can follow this approach, or split the minimax function into separate min and max functions.

Part 3: *You Look Like a Thing and I Love You* reflections

Question 1

In Chapter 4, Shane discusses an issue that data-driven approaches to AI face: people talk more about things that are surprising than about things that seem commonplace. For instance, people mention are more likely to mention a giraffe than a tree in a caption of a picture containing both. Similarly, question-answering datasets often suffer from the absence of questions that humans deem irrelevant, like "How many giraffes are there?" when the picture does not contain giraffes.

Is this missing data problem an issue in symbolic approaches to AI, like the Pacman game that we have been working with? Why or why not?

Question 2 (10 points)

In Chapter 3, Shane discusses a sandwich-making neural network. Revisit this application from the lens of reinforcement learning.

A. Discuss how you could set up a reward function and what a policy would be.

B. Do you think a reinforcement learning approach is better than the neural network that Shane discusses? Why or why not?