Homework 5: Pacman

Brandon McKinzie

Problem 1: Minimax

(a) *Before you code up Pac-Man as a minimax agent, notice that instead of just one adversary, Pac-Man could have multiple ghosts as adversaries. So we will extend the minimax algorithm from class (which had only one min stage for a single adversary) to the more general case of multiple adversaries. In particular, your minimax tree will have multiple min layers (one for each ghost) for every max layer.*

Specifically, consider the limited depth tree minimax search with evaluation functions taught in class. Suppose there are $n+1$ agents on the board, $a_0, \ldots, a_n$, where a_0 is Pac-Man and the rest are ghosts. Pac-Man acts as a max agent, and the ghosts act as min agents. A single depth consists of all $n+1$ agents making a move, so depth 2 search will involve Pac-Man and each ghost moving two times. In other words, a depth of 2 corresponds to a height of $2(n+1)$ in the minimax game tree.

Write the recurrence for $V_{minmax}(s,d)$ in math. You should express your answer in terms of the following functions: IsEnd(s), which tells you if s is an end state; Utility(s), the utility of a state; Eval(s), an evaluation function for the state s; Player(s), which returns the player whose turn it is; Actions(s), which returns the possible actions; and Succ(s,a), which returns the successor state resulting from taking an action at a certain state. You may use any relevant notation introduced in lecture.

$$V_{minmax}(s,d) = \begin{cases} \text{Utility}(s) & \text{if IsEnd}(s) \\ Eval(s) & d = 0 \\ \max_{a \in \text{Actions}(s)} V_{minmax}(\text{Succ}(s,a),d) & \text{Player(s)}{=}a_0 \\ \min_{a \in \text{Actions}(s)} V_{minmax}(\text{Succ}(s,a),d) & \text{Player(s)}{\in}\{a_1,\ldots,a_{n-1}\} \\ \min_{a \in \text{Actions}(s)} V_{minmax}(\text{Succ}(s,a),d-1) & \text{Player(s)}{=}a_n \end{cases} \tag{1}$$

Problem 3: Expectimax

(a) *Random ghosts are of course not optimal minimax agents, so modeling them with minimax search is not optimal. Instead, write down the recurrence for $V_{exptmax}(s, d)$, which is the maximum expected utility against ghosts that each follow the random policy which chooses a legal move uniformly at random. Your recurrence should resemble that of Problem 1a (meaning you should write it in terms of the same functions that were specified in 1a).*

$$V_{exptmax}(s,d) = \begin{cases} \text{Utility}(s) & \text{if IsEnd}(s) \\ Eval(s) & d = 0 \\ \max_{a \in \text{Actions}(s)} V_{exptmax}(\text{Succ}(s,a), d) & \text{Player(s)}=a_0 \\ \frac{1}{|\text{Actions}(s)|} \sum_{a \in \text{Actions}(s)} V_{exptmax}(\text{Succ}(s,a), d) & \text{Player(s)} \in \{a_1, \ldots, a_{n-1}\} \\ \frac{1}{|\text{Actions}(s)|} \sum_{a \in \text{Actions}(s)} V_{exptmax}(\text{Succ}(s,a), d-1) & \text{Player(s)}=a_n \end{cases} \tag{2}$$

Problem 4: Evaluation Function (extra credit)

(b) *Clearly describe your evaluation function. What is the high-level motivation? Also talk about what else you tried, what worked and what didn't. Please write your thoughts in pacman.pdf (not in code comments).*

It's a simple linear model over some of the GameState attributes I thought would be useful. I used the number of food pellets remaining on the board, the Manhattan distance to both the nearest ghost and the nearest food pellet, and the current game score. I manually defined the weights for this linear model. I set positive weights for the game score (10) and the distance to the nearest ghost (1), in the hopes it would favor states with higher scores and further distances from ghosts. I assigned negative weights for the number of food pellets remaining (-1) and the distance to the nearest food pellet (-5), in the hopes it would favor states where food is nearby and more food has been eaten.

The challenge still remaining is that pacman likes walking back and forth a lot. This is partially due to me favoring states where he is really close (but not on top of) something like a food pellet. I wasn't able to find anything in the code that supported "is currently on top of a food pellet and is eating it". Similarly, we weren't able to utilize his previous position and direction (explicitly stated in the code). If we were, we could've discouraged changing directions repeatedly.