CS221 Summer 2019 Homework 5

SUNet ID: jaigupta
Name: Jai Gupta

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1

(a) If N is the number of ghosts and D is the max depth. The $V_{minimax}(s, D)$ gives the solution, where $V_{minimax}(s, d)$ follows the following recursion:

$$V_{minimax}(s,d) = \begin{cases} Utility(s) & \text{if } IsEnd(s) \\ Eval(s) & \text{if } d = 0 \\ \max_{a \epsilon Actions(s)} V_{minimax}(Succ(s,a), d) & \text{if } Player(s) = a_0 \text{ (Pacman)} \\ \min_{a \epsilon Actions(s)} V_{minimax}(Succ(s,a), d) & \text{if } Player(s) = a_i; i \epsilon \{1, 2, ..., N-1\} \\ \min_{a \epsilon Actions(s)} V_{minimax}(Succ(s,a), d-1) & \text{if } Player(s) = a_N \text{ (last ghost)} \end{cases}$$

Note that $Actions(s)$ is assumed to return only the list of legal possible actions at state s.

Problem 3

(a) If N is the number of ghosts and D is the max depth. The $V_{expectimax}(s, D)$ gives the solution, where $V_{expectimax}(s, d)$ follows the following recursion:

$$V_{expectimax}(s,d) = \begin{cases} Utility(s) & \text{if } IsEnd(s) \\ Eval(s) & \text{if } d = 0 \\ \max_{a \epsilon Actions(s)} V_{expectimax}(Succ(s,a), d) & \text{if } Player(s) = a_0 (Pacman) \\ \frac{\sum_{a \epsilon Actions(s)} V_{expectimax}(Succ(s,a), d)}{|Actions(s)|} & \text{if } Player(s) = a_i; \ i \epsilon \{1, 2, ..., N-1\} \\ \frac{\sum_{a \epsilon Actions(s)} V_{expectimax}(Succ(s,a), d-1)}{|Actions(s)|} & \text{if } Player(s) = a_N \text{ (last ghost)} \end{cases}$$

Note that $Actions(s)$ is assumed to return only the list of legal possible actions at state s.

Problem 4

(a) in code

(b) The evaluation function works as follows:

i. Use currentGameState.getScore() as this is a score that is certainly avaiable.

ii. Add score of 7*number_of_remaining_food as an approximation of amount max score achievable. Here 7 is a manually trained hyper-parameter. Note that because we keep the score lower than the food score of 10, this score starts prefering states that have eaten more in the 2 depths before calling the evaluation function.

iii. In order to favor food that is nearest, we deduct the distance (calculated using BFS) from the current position to the nearest food as penalty. This essentially favors positions where food is nearer.

iv. We also penalize states that have more capsules left as capsules are really important for creating scared ghosts.

v. If a ghost is in scared position and its timer is greater than or equal to the distance, we favor moving in its direction by adding "50 - distance_to_ghost" as an additional score. This moves in the direction of the nearest scared ghost.