Jianwei Zheng
jianwei@ucsb.edu
3290962

Machine Problem 2 Report

In this problem, we are asked to build a search agent to make the computer win the Pacman.

- Architecture:
  Based on the requirements of the agents, I only add getAction() and myGetScore(). In getAction, I define minMax(), a mini-max algorithm to determine which action should be chosen.

- Preprocessing:
  All data structures are implemented in the source code.

- Model Building:
  The agent is based on getAction() and myGetScore(). The myGetScore() calculates the score by the given state of the game. The getAction() makes a depth of the given depth minMax tree with alpha and beta pruning and compares the score calculated by myGetScore() to choose the suitable action Pacman should do.

- Results:
  Running time: about one minute each game for given depth is 3
  Pacman could win at least once for 10 trials for the default maze.

- Challenges:
  To get the higher probability of Pacman's win, I revise the myGetScore() multiple times. Since the point of capsules is proportional with the remaining number of food, I increase the weight of capsules to let Pacman eat capsules as soon as possible. For the default myGetScore(), Pacman are highly likely to be caught by ghosts or get stuck or loop at some locations. I added the weights of ghosts distances, number of walls and food distances to get better performance.
  When I tested the MultiPacmanAgent on a corner maze, I found that the Pacman would get stuck and loop forever. To solve this problem, I added an 2-dimensional array to represent the number of states that are visited, and add repeated states penalty in myGetScore().

- Weaknesses:
  There are two main weaknesses. One is that we assume that ghosts' actions are optimal but they are random. This sometimes makes the Pacman's action not optimal. The other one is that the evaluation function is not perfect. One way to improve this is to implement heuristic.