

# Project Report

## Artificial Intelligence

### Project-2, Multi Agent Packman

Ankit Dave   Rishi Josan   Abhiroop Dabral

Our submission includes report and file multiagent.py for first 3 questions. We have a separate report for Sudoku questions.

#### **Reflex Agent:**

We first compute the Manhattan distance to each ghost and verify if the ghost is active or not. If the ghost is active, it can kill pacman. We store the the Manhattan distance to each active ghost and the negative of the Manhattan distance to each inactive ghost in a list.

We store the negative distance for an inactive ghost as we would like to eat that ghost. It is not necessary to eat the inactive ghost, but we believe this strategy results in better scores.

Also if we notice the the manhattan distance to the ghost reaches 1, we run away from the ghost by returning the minimum integer value as the score for going the ghost's direction. We tried higher values for this and stuck with 1 as this gives best performance.

We also compute the sum of manhattan distances to all the food left. The sum of the distances to ghosts is divided by the the sum of manhattan distances to the food. This number is then added to the score returned.

Average score: 1059

#### **MiniMax Agent**

In this we have to implement a mini max algorithm provided in the textbook with a little variation. For this we have made changes to MiniMaxAgent stub class.

We start our algorithm from the given state and depth. At this stage we try to maximize our score by calling max agent. In this if the present state is a win/lose or if the depth is zero it returns the return the default direction. Otherwise it makes a list of all the moves that are allowed except the stop move. After that, we take every action from the above list of possible moves for our pac man and then for every such item we call the MinAgent. And this will then return the action from which max value was obtained.

Similarly, Min agent function will return stop if it is a win/lose or the max depth. Otherwise it will count the number of agents and then for every agent it will the possible moves and for each move it will try to max its value by calling Max agent. From this it will select the min move for the agent.

**Output:**

**python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4**

```
Ankit@Ankit /cygdrive/f/Source Code/projects/Homeworks/Artificial
$ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores: 516.0
Win Rate: 1/1 (1.00)
Record: win
```

**python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3**

```
Ankit@Ankit /cygdrive/f/Source Code/projects/Homeworks/Artificial In
$ python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
Pacman died! Score: -502
Average Score: -502.0
Scores: -502.0
Win Rate: 0/1 (0.00)
Record: Loss
```

In this case, our pacman always dies as it is in trapped state, and it runs towards the closest ghost, because it wants to end the game asap.

**python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 --numGames 1000 --frameTime 0 --fixRandomSeed --textGraphics**

**Running time:** 9min 25 sec

**win rate:** 678/1000 (0.68)

Observations: On larger boards such as openClassic and mediumClassic, we found that the pac man is good at not dying but very bad at winning. It can be seen thrashing without making any progress.

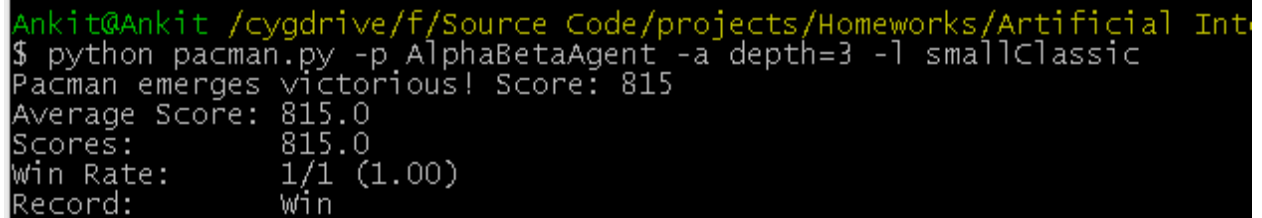
### **Alpha Beta pruning:**

In this we implemented alpha beta pruning in AlphaBetaAgent stub class. In this we initially start with alpha as minus infinity and beta as positive infinity. And we try to maximize our score by calling maxAgent. In this, we again prepare a list of all the possible moves except the stop move and we iterate through this list. For every item in this list, we try to minimize the score in that direction. We then return the direction from which we encountered the maximum value. Also, every time we check the return value with the beta value, if it is greater than beta value we just return the present value. Otherwise we update alpha with the present if alpha is less than present value.

Similarly in minAgent, for every agent, we calculate all the possible moves and for every move we calculate the maxAgent value. If the returned value by maxAgent is less than alpha we return this value else we update beta if it is greater than returned value.

#### Output:

```
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```



```
Ankit@Ankit /cygdrive/f/Source Code/projects/Homeworks/Artificial Int
$ python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
Pacman emerges victorious! Score: 815
Average Score: 815.0
Scores:      815.0
Win Rate:    1/1 (1.00)
Record:      win
```

```
python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4 --numGames 1000 --frameTime 0 -
-fixRandomSeed -textGraphics
```

**Running Time:** 7min 20sec

**win rate:** 726/1000 (0.73)