Project 02: Multi-Agent Pac-Man

Contents

1 Description

1.1 Question 1: Reflex Agent

This function evaluates the gamestate of the Pacman and returns a score. It evaluates as:

1. If the closest scary ghost is in a range of 4 Manhattan distance, then the Manhattan distance between the closest scary ghost and the Pacman is returned.
2. If Pacman dies in the next move, then MIN value is returned, so that Pacman does not choose that move.
3. If a non scary ghost is in a range of 4 Manhattan distance, then Pacman is made to chase the non scary ghost, by returning higher scores than other moves.
4. If Pacman eats the food or the pellet in the next move, then MAX value is returned as score, so that Pacman chooses the move to eat it.

(Here, MAX is sys.maxint and MIN is -MAX).
As it can be seen in the figure, the function consistently wins in a number of games.

```
charuta:multiagent charuta$ python2.7 pacman.py --frameTime 0 -p ReflexAgent -k 1 -n 10 -q
Pacman emerges victorious! Score: 1251
Pacman emerges victorious! Score: 1164
Pacman emerges victorious! Score: 1408
Pacman emerges victorious! Score: 1590
Pacman emerges victorious! Score: 1324
Pacman emerges victorious! Score: 1219
Pacman emerges victorious! Score: 805
Pacman emerges victorious! Score: 1454
Pacman emerges victorious! Score: 977
Pacman emerges victorious! Score: 1255
Average Score: 1244.7
Scores:        1251.0, 1164.0, 1408.0, 1590.0, 1324.0, 1219.0, 805.0, 1454.0, 977.0, 1255.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
charuta:multiagent charuta$ python2.7 pacman.py --frameTime 0 -p ReflexAgent -k 1 -n 10 -q
Pacman emerges victorious! Score: 901
Pacman emerges victorious! Score: 1476
Pacman emerges victorious! Score: 1418
Pacman emerges victorious! Score: 1375
Pacman emerges victorious! Score: 1369
Pacman emerges victorious! Score: 987
Pacman emerges victorious! Score: 1531
Pacman emerges victorious! Score: 1445
Pacman emerges victorious! Score: 1398
Pacman emerges victorious! Score: 1089
Average Score: 1298.9
Scores:        901.0, 1476.0, 1418.0, 1375.0, 1369.0, 987.0, 1531.0, 1445.0, 1398.0, 1089.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

1.2 Question 2: Minimax Search Agent

A helper function called minimaxFunction has been written.
This function takes the following arguments:

Argument	**Description**
gameState	The current game state
action	The predecessor's action which leads to the current game state
currentDepth	The depth of the current game state
currentAgentIndex	The agent whose successor states have to be evaluated
isMax	True for max nodes (Pacman), False for min nodes (ghosts)

The function executes the minimax algorithm and returns [score, action], which is the best possible score and the corresponding action.

The getAction function calls minimaxFunction and returns the calculated action.

Worst case:
Time complexity = O(b^m)
Space complexity = O(bm)
where b is the branching factor and m is the depth.

(Here, b is 5, as every agent has 5 possible actions and hence 5 possible successor states.)

1.3 Question 3: AlphaBeta Search Agent

A helper function called alphabetaFunction has been written.
This function takes the following arguments:

Argument	**Description**
gameState	The current game state
action	The predecessor's action which leads to the current game state
currentDepth	The depth of the current game state
currentAgentIndex	The agent whose successor states have to be evaluated
isMax	True for max nodes (Pacman), False for min nodes (ghosts)
alpha	The alpha value propagated from the parent
beta	The beta value propagated from the parent

This function executes the minimax algorithm along with alpha-beta pruning and returns [score, action, retAlpha, retBeta], where score is the best possible score, action is the corresponding move, retAlpha is the evaluated alpha value of the current node, and retBeta is the evaluated beta value of the current node.

The getAction function calls alphabetaFunction and returns the calculated action.

Worst case:
Time complexity = O(b^m)
Space complexity = O(bm)
where b is the branching factor and m is the depth.

(Here, b is 5, as every agent has 5 possible actions and hence 5 possible successor states.)

1.4 Question 4: Expectimax Search Agent

A helper function called expectimaxFunction has been written.
This function takes the following arguments:

Argument	**Description**
gameState	The current game state
action	The predecessor's action which leads to the current game state
currentDepth	The depth of the current game state
currentAgentIndex	The agent whose successor states have to be evaluated
isMax	True for max nodes (Pacman), False for min nodes (ghosts)

The function executes the expectimax algorithm and returns [score, action], which is the best possible score and the corresponding action.

The getAction function calls expectimaxFunction and returns the calculated Action.

Worst case:
Time complexity = $O(b^m)$
Space complexity = $O(bm)$
where *b* is the branching factor and *m* is the depth.

(Here, *b* is 5, as every agent has 5 possible actions and hence 5 possible successor states.)

2 Results and Comparison

2.1 Random seed

The minimaxClassic maze was solved 5 times each using the Minimax, AlphaBeta and Expectimax agents using depths 2, 3 and 4. The following results were obtained:

Depth	Agent	Average Nodes Expanded	Number of Wins (Out of 5)	Average Score
2	Minimax	5542.2	2	-502.2
	AlphaBeta	1770.8	2	-92.2
	Expectimax	1326.2	4	313.2
3	Minimax	11282.8	3	108.4
	AlphaBeta	6445.6	1	-293.4
	Expectimax	9376.4	4	107.8
4	Minimax	30818.0	3	111.8
	AlphaBeta	21327.8	4	314.0
	Expectimax	34171.8	5	516.0

The following observations can be made from these results:

1. In general, the Expectimax agent expands the lesser number of nodes, wins the most number of games, and yields the highest average score.

2. The AlphaBeta agent expands significantly less number of nodes as compared to the Minimax agent.

3. AlphaBeta and Expectimax yield better scores than Minimax.

2.2 Fixed Seed

The smallClassic and trickyClassic mazes were solved using the Minimax, AlphaBeta and Expectimax agents, using depth 2.

The following results were obtained:

Maze	Agent	Number of Nodes Expanded	Time	Win / Loss	Score
Small Classic	Minimax	10086	0 min 1.241 sec	Loss	-50
	AlphaBeta	8559	0 min 1.075 sec	Loss	-50
	Expectimax	14582	0 min 1.707 sec	Loss	98
Tricky Classic	Minimax	540327	1 min 32.163 sec	Loss	77
	AlphaBeta	446925	1 min 16.979 sec	Loss	77
	Expectimax	209735	0 min 35.185 sec	Loss	750

The following observations can be made from these results:

1. Expectimax performs worse than the other 2 in the smallClassic maze, and better than the other 2 in the trickyClassic maze.

2. However, Expectimax consistently yields a significantly better score than the other 2.

3. AlphaBeta pruning consistently expands lesser nodes than Minimax.