CS 386: Lab Assignment 8

(TA in charge: Vaibhav Bhosale)

Acknowledgement: This lab assignment is based on Project 2: Multi-Agent Pacman , which is a part of a recent offering of CS188 at UC Berkeley. The code and resources provided here are almost entirely drawn from the Berkeley project. We thank the authors at Berkeley for making their project available to the public.

Pacman lives in a shiny blue world of twisting corridors and tasty round treats. Navigating this world efficiently will be Pacman's first step in mastering his domain. We use this game as a model to understand how different search algorithms work. In this assignment, you will design agents for the classic version of Pacman, including ghosts. Along the way, you will implement both minimax and expectimax search and try your hand at evaluation function design.

Code

The base code for this assignment is available in this zip file. The following files are the most relevant ones for you; you will only have to edit `multiAgents.py`.

File Name	Description
multiAgents.py	Where all of your multi-agent search agents will reside.
pacman.py	The main file that runs Pacman games. This file describes a Pacman GameState type, which you use in this project.
game.py	The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.
util.py	Useful data structures for implementing search algorithms.

After downloading the code, unzipping it, and changing to the directory, you should be able to play a game of Pacman by running the following command.

```
python pacman.py
```

`pacman.py` supports a number of options (e.g. `--layout` or `-l`). You can see the list of all options and their default values via `python pacman.py -h`.

Task 1: Reflex Agent (2 marks)

Run the provided `ReflexAgent` in `multiAgents.py`.

```
python pacman.py -p ReflexAgent
```

This agent plays poorly even on simple layouts. Your task is to improve the `ReflexAgent` in `multiAgents.py` to play respectably. The provided reflex agent code provides some helpful examples of methods that query the GameState for information. A capable reflex agent will have to consider both food locations and ghost locations to perform well. Your agent should easily and reliably clear the testClassic layout.

```
python pacman.py -p ReflexAgent -l testClassic
```

Note: Some of the `get` functions from Gamestate return a data type caled `Grid`; see `game.py` to see how `Grid` can be processed. See `pacman.py` to find the list of `get` functions available from Gamestate.

Evaluation: Your agent will be run on the `openClassic` layout 10 times. You will receive 1 mark if your agent wins all 10 times, 0.5 marks if it wins more than 5 times, otherwise 0. Make sure your agent does not ever time out. Else, you will be awarded 0 marks for this task. If your agent's average score exceeds 1000, you get an additional 1 mark; if it exceeds 500, you get 0.5 marks. You can try your agent out under these conditions by running the following command.

```
python autograder.py -q q1
```

Note: Your marks for this question are half of the marks displayed by the autograder.

Task 2: Minimax (3 marks)

In this task, you will write an adversarial agent in the provided `MinimaxAgent` class in `multiAgents.py`. Your agent should work with any number of ghosts, so the algorithm you would be using should be a generalised version of the standard Minimax algorithm. Your code should be able to run the algorithm for an arbitrary depth which can be accessed from `self.depth` and score your nodes with the supplied `self.evaluationFunction`. Using these variables is a must since these are populated in response to the command line options.

Key Observations:

- The correct implementation of minimax will lead to Pacman losing the game in some tests. This is not a problem: as it is correct behaviour, it will pass the tests.
- Pacman is always agent 0, and the agents move (take turns) in the order of increasing agent index. Use the `getLegalActions(agentIndex)` function in GameState to get all the legal actions possible for agent with index `agentIndex`. Tasks 2 and 3 are based on the assumption that the ghosts will act optimally against Pacman. However, note that in reality, the ghosts pick actions uniformly at random. In Task 4, we explicitly model for this random ghost behaviour.
- All states in minimax should be GameStates, either passed in to `getAction` or generated via `GameState.generateSuccessor`.

Evaluation: Your code will be checked to determine whether it explores the correct number of game states. This is the only way reliable way to detect some very subtle bugs in implementations of minimax. As a result, the autograder will be very picky about how many times you call `GamesState.generateSuccessor`. All the nodes for which this function is called would be included in the expanded nodes set. Your agent needs to pass all the tests to earn full credit. If the last test case fails, 1 mark will be deducted, and 0.5 marks will be deducted each other failure in the test cases. To test your code, run this command.

```
python autograder.py -q q2
```

Task 3: Alpha-Beta Pruning (3 marks)

In this task, you will write an adversarial agent in the provided `AlphaBetaAgent` class in `multiAgents.py` to more efficiently explore the minimax tree. Your agent should work with any number of ghosts, so your algorithm should be a generalised version of the standard Alpha-Beta Pruning algorithm.

Evaluation: Again, your agent will be evaluated to check if it explores the correct number of states in the correct order. Therefore, it is important that you perform alpha-beta pruning without reordering children. In other words, successor states should always be processed in the order returned by `GameState.getLegalActions`. Again, do not call `GameState.generateSuccessor` more than necessary. Additionally, in order to match the set of states explored by the autograder, you must not prune on equality. Your agent needs to pass all the tests to earn full credit. If the last test case fails, 1 mark will be deducted, and 0.5 marks deducted for each other failure on the test cases. Test your code as follows.

```
python autograder.py -q q3
```

Note: The correct implementation of alpha-beta pruning will lead to Pacman losing some of the tests. This is not a problem: as it is correct behaviour, it will pass the tests.

Task 4: Expectimax (3 marks)

In this question you will implement the ExpectimaxAgent, which is useful for modeling probabilistic behavior of agents who may make suboptimal choices. Again your algorithm should be generic enough so that multiple ghosts are handled.

Evaluation: Your agent needs to pass all the tests to earn full credit. If the last test case fails, 1 mark will be deducted, and 0.5 marks will be deducted for each other failure on the test cases. Run the following command to test your solution: `python autograder.py -q q4`.

Note: The correct implementation of expectimax will lead to Pacman losing some of the tests. This is not a problem: as it is correct behaviour, it will pass the tests.

Task 5: Evaluation Function (4 marks)

Write a more effective evaluation function for Pacman in the provided function `betterEvaluationFunction`. Provide comments in your code to explain which features you are computing, and how they are combined.

Evaluation: Your agent will be run on the layout `smallClassic` 10 times and marks will be assigned as follows.

- +1 for winning at least 5 times, +2 for winning all 10 times.
- +1 for an average score of at least 500, +2 for an average score of at least 1000 (including scores on lost games).
- The additional points for average score will only be awarded if you win at least 5 times.

Test your agent as follows: `python autograder.py -q q5`.

Submission

You are not done yet! Place all files in which you have written code in or modified in a directory named your roll number (say 12345678). Tar and Gzip the directory to produce a single compressed file (la8-12345678.tar.gz). It must contain the following files.

1. `multiAgents.py`

Submit this compressed file on Moodle, under Lab Assignment 8.