COEN 266 Artificial Intelligence

Group Homework #2 Minimax Game

Introduction

In this GROUP assignment, you will design a minimax algorithm for Pacman (player-MAX), who plays a game with three ghosts (player-MINs). The game tree will have MAX nodes and MIN nodes. In general, MAX nodes will call a "max" function to collect the largest value from its successors, and MIN nodes will call a "min" function to collect the minimum value from its successors.

You can download and unzip all the code and supporting files from **multiagent.zip**.

Implement the following function in **Class MinimaxAgent,** in multiAgents.py

def getAction(self, gameState):

```
    …

    # this function will

    • recursively call a "max" function and a "min" function.

    • propagate the "leaf" node values to upper layers, until the root node is reached.

    • finally, return the best action for player-max (Pacman) at the root node.
```

Files you'll edit:	
multiAgents.py	Where your multi-agent search agent will reside.
Files you might want to look at:	
pacman.py	The main file that runs Pacman games. This file also describes a Pacman GameState class, which you will use extensively in this assignment.
game.py	The logic behind how the Pacman world works. This file describes several supporting classes like AgentState, Agent, Direction, and Grid.
util.py	Useful data structures for implementing search algorithms. You may find some functions defined here to be useful.

Supporting files you can ignore:	
graphicsDisplay.py	Graphics for Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
ghostAgents.py	Agents to control ghosts
keyboardAgents.py	Keyboard interfaces to control Pacman
layout.py	Code for reading layout files and storing their contents

Task: Minimax Agent

Now you will write an adversarial search agent in the provided MinimaxAgent class stub in multiAgents.py. Your minimax agent should work with any number of ghosts. In particular, your minimax tree will have multiple min layers (one for each ghost) for every max layer.

Your code should also expand the game tree to an arbitrary depth. Score the leaves of your minimax tree with the supplied scoreEvaluationFunction. MinimaxAgent extends MultiAgentSearchAgent, which gives access to self.depth (the number of search plies of the game tree). A single search ply (self.depth=1) is considered to be one Pacman move and all the ghosts' responses, so a search with self.depth=2 will involve Pacman and each ghost moving two times.

Make sure your minimax code makes reference to variable self.depth where appropriate as this variable is populated in response to command line options.

Experiment:

```
python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 --frameTime 0
```

Hints and Observations

- The evaluation function for the Pacman test in this part is already written (scoreEvaluationFunction), which evaluates the *states*. You shouldn't change this function.
- The minimax values of the initial state in the minimaxClassic layout are 9, 8, 7, -492 for self.depth=1, 2, 3 and 4 respectively.
- Pacman is always agent 0, and the agents move in order of increasing agent index.
- All states in minimax should be GameStates, either passed in to getAction or generated via GameState.generateSuccessor.

- On larger boards such as openClassic and mediumClassic (the default), you'll find Pacman to be good at not dying, but quite bad at winning. He'll often thrash around without making progress. He might even thrash around right next to a dot without eating it because he doesn't know where he'd go after eating that dot.
- You can check the correctness of your code by running the following test:

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

- This will show what your algorithm does on a number of small trees, as well as a pacman game. To run it without graphics, use:

```
python autograder.py -q q2 --no-graphics
```

- The correct implementation of minimax will lead to Pacman losing the game in some tests of the aforementioned autograder.py. This is not a problem: as it is correct behavior, it will pass the tests.

Submission:

1. Submit a pdf file to Camino (for the format of the file, please refer to GroupHW2_submission_sample.pdf).

   The following items are essential components of your code, and you will explain them in your pdf report.

   - How does the "max" function work?
   - How does the "min" function work?
   - How to recursively call the "max" function and "min" function?
   - How is depth=4 reached?
   - How does the root node return the best action?
   - How do you set the PlayerIndex for Pacman and for three ghosts?
   - When depth<4, how to go from the 3rd ghost to the next player-MAX, and at the same time increase the depth by 1?

2. Submit all source code needed (with multiAgents.py modified by you) to generate the result of the **Experiment** as a .zip file to Camino. We will test run your submitted code, so make sure it works. We expect Pacman to win at least 5 out of 10 successive games. Your code should also pass 5/5 tests of the following: python autograder.py -q q2 --no-graphics