Intro to AI & AI, 2023 Spring
Multi-agent Pacman

This program assignment is based on the project of UCB's AI course. You can refer to the full project via https://inst.eecs.berkeley.edu/~cs188/sp23/projects/proj3/#q2-5-pts-minimax. Here, we only try to implement "**Q2: Minimax**. You may download the source code from https://inst.eecs.berkeley.edu/~cs188/sp23/assets/projects/multiagent.zip .

Files:

Files you'll edit:	
`multiAgents.py`	Where all of your multi-agent search agents 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 type, which you will use extensively 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. You don't need to use these for this project, but may find other 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
`autograder.py`	Project autograder
`testParser.py`	Parses autograder test and solution files
`testClasses.py`	General autograding test classes
`test_cases/`	Directory containing the test cases for each question
`multiagentTestClasses.py`	Project 3 specific autograding test classes

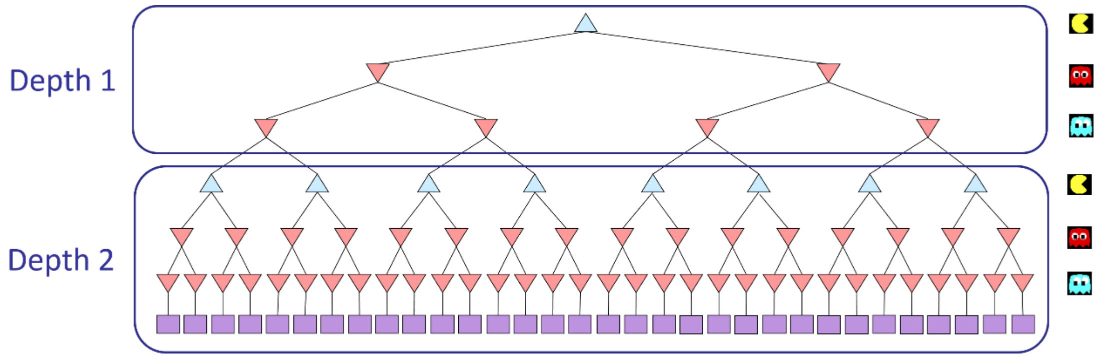
Problem Description:

Minimax

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, so you'll have to write an algorithm that is slightly more general than what you've previously seen in lecture. 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 `self.evaluationFunction`, which defaults to `scoreEvaluationFunction`. `MinimaxAgent` extends `MultiAgentSearchAgent`, which gives access to `self.depth` and `self.evaluationFunction`. Make sure your minimax code makes reference to these two variables where appropriate as these variables are populated in response to command line options.

Important: A single search ply is considered to be one Pacman move and all the ghosts' responses, so depth 2 search will involve Pacman and each ghost moving two times (see diagram below).



Grading: We will be checking your code to determine whether it explores the correct number of game states. This is the only 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 `GameState.generateSuccessor`. If you call it any more or less than necessary, the autograder will complain. To test and debug your code, run

```
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
```

Hints and Observations

- Implement the algorithm recursively using helper function(s).
- 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.

- The evaluation function for the Pacman test in this part is already written (`self.evaluationFunction`). You shouldn't change this function, but recognize that now we're evaluating states rather than actions, as we were for the reflex agent. Look-ahead agents evaluate future states whereas reflex agents evaluate actions from the current state.
- The minimax values of the initial state in the `minimaxClassic` layout are 9, 8, 7, -492 for depths 1, 2, 3 and 4 respectively. Note that your minimax agent will often win (665/1000 games for us) despite the dire prediction of depth 4 minimax.

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

- 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`. In this project, you will not be abstracting to simplified states.
- 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. Don't worry if you see this behavior, question 5 will clean up all of these issues.
- When Pacman believes that his death is unavoidable, he will try to end the game as soon as possible because of the constant penalty for living. Sometimes, this is the wrong thing to do with random ghosts, but minimax agents always assume the worst:

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

Make sure you understand why Pacman rushes the closest ghost in this case.

Requirements:

1. Your program should pass by the autograder. (80%)

$ python autograder.py -q q2

```
### Question q2: 5/5 ###


Finished at 18:08:53

Provisional grades
==================
Question q2: 5/5
------------------
Total: 5/5

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

2. Write comments of your code to describe functionalities of your code segments and the reason why you implement it. (20%)
3. DO NOT copy the code from others. (0 points instead.)

Submissions:

1. multiAgents.py (Your comments should be included.).
2. Any other files you modified.

More Hints:

Code framework explained by TA (in English): https://youtu.be/ZmwQkRsHOyw
The starting point of your implementation may be getAction() of MinimaxAgent class and you may follow to the state dispatch in the lecture slide for the implementation.