CS 214: Artificial Intelligence Lab

Common Instructions: You have to submit a code without any syntax errors. Don't copy from others; write your own piece of code with comments as much as possible. Try to Run with the test cases before submitting the code.

Assignment 3

NOTE:

- Download *search.zip*, unzip and rename it with your group number as "group_2" for example.
- Test your code and submit it on Moodle (`https://moodle.iitdh.ac.in/user/index.php?id=992`).
- The deadline for submission is **January 31 2024 by (11:59 PM)**
- We will run a plagiarism check for all the submissions; if found penalty will be applied

Lab Instructions:

In this project, you will design agents for the classic version of Pacman, including ghosts. Along the way, you will implement both minimax search and alpha-beta pruning.

This project includes an autograder for you to grade your answers on your machine. This can be run with the command: ***python autograder.py***

The code for this project consists of several Python files, some of which you will need to read and understand in order to complete the assignment and some of which you can ignore. You can download all the code and supporting files as a zip archive.

File to edit:

- **multiAgents.py** All your multiagent search functions will reside here.

Can look at files:

- **Pacman.py:** The main file that runs Pacman game. This file describes a Pacman GameState type, which you use in this project.
- **game.py:** This file describes several supporting types like AgentState, Agent, Direction, and Grid.
- **util.py:** Useful data structures for implementing search algorithms.

You can ignore all other supporting files

Files to Edit and Submit:
You will fill in portions of **search.py** and **searchAgents.py** during the assignment. You should submit these files with complete code and comments. Please do not modify other files in this distribution.

Evaluation:
Your code will be autograded for technical correctness. **Please do not change the names of any provided functions** or classes within the code, or your code can not be evaluated. However, the marks will be awarded based on the judgment of autograder and manual evaluation.

The Pacman Game:

After downloading the code (search.zip), unzipping it, and changing to the directory, you should be able to play a game of Pacman by typing the following at the command line:

python pacman.py

using the arrow keys to move. Now, run the provided ReflexAgent in multiAgents.py

python pacman.py -p ReflexAgent
Try to run the ReflexAgent on simple layout and see how poorly it plays using command

python pacman.py -p ReflexAgent -l testClassic

make sure you understand the code in multiAgents.py.

Question 1: Minmax

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. 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 a depth 2 search will involve Pacman and each ghost moving two times.

make sure that your code explores the correct number of game states. 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

Question 2: Alpha-Beta Pruning

Make a new agent that uses alpha-beta pruning to more efficiently explore the minimax tree, in AlphaBetaAgent. Again, your algorithm will be slightly more general than the pseudocode from the lecture, so part of the challenge is to extend the alpha-beta pruning logic appropriately to multiple minimizer agents.

You should see a speed-up (perhaps depth 3 alpha-beta will run as fast as depth 2 minimax). Ideally, depth 3 on smallClassic should run in just a few seconds per move or faster.

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

The *AlphaBetaAgent* minimax values should be identical to the *MinimaxAgent* minimax values, although the actions it selects can vary because of different tie-breaking behavior. Again, the minimax values of the initial state in the *minimaxClassic* layout are 9, 8, 7, and -492 for depths 1, 2, 3, and 4, respectively.

Because we check your code to determine whether it explores the correct number of states, 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.

You must not prune on equality in order to match the set of states explored by our autograder. (Indeed, alternatively, but incompatible with our autograder, it would also be to allow for pruning on equality and invoke alpha-beta once on each child of the root node, but this will not match the autograder.)

Follow the pseudocode from class to implement the algorithm. To test and debug your code, run

python autograder.py -q q3

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 q3 –no-graphics

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 behavior, it will pass the tests.