CS 214: Artificial Intelligence Lab
Spring 2022-23, IIT Dharwad
Assignment-4
Artificial Intelligence Lab

Instructions

In this project, your Pacman agent will find paths through his maze world, both to reach a particular location and to collect food efficiently. You will build general search algorithms and apply them to Pacman scenarios.

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.

Files you'll edit:

- ***search.py*** Where all of your search algorithms will reside.

Files you might want to look at:

- ***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.

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
- ***searchTestClasses.py*** Project 1 specific autograding test classes

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 your code and comments. Please do not change the other files in this distribution or submit any of our original files other than these files.

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 you will wreak havoc on the autograder. However, the correctness of your implementation -- not the autograder's judgements -- will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.

Welcome to Pacman

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
```

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.

The simplest agent in searchAgents.py is called the GoWestAgent, which always goes West (a trivial reflex agent). This agent can occasionally win:

```
python pacman.py --layout testMaze --pacman GoWestAgent
```

But, things get ugly for this agent when turning is required:

```
python pacman.py --layout tinyMaze --pacman GoWestAgent
```

If Pacman gets stuck, you can exit the game by typing CTRL-c into your terminal. Soon, your agent will solve not only tinyMaze, but any maze you want.

Note that pacman.py supports a number of options that can each be expressed in a long way (e.g., --layout) or a short way (e.g., -l). You can see the list of all options and their default values via:

```
python pacman.py -h
```

Also, all of the commands that appear in this project also appear in commands.txt, for easy copying and pasting. In UNIX/Mac OS X, you can even run all these commands in order with bash *commands.txt***.**

Question 1: Varying the Cost Function (25 Points)

While BFS will find a fewest-actions path to the goal, we might want to find paths that are "best" in other senses. Consider mediumDottedMaze and mediumScaryMaze.

By changing the cost function, we can encourage Pacman to find different paths. For example, we can charge more for dangerous steps in ghost-ridden areas or less for steps in food-rich areas, and a rational Pacman agent should adjust its behavior in response.

Implement the **uniform-cost graph search algorithm** in the uniformCostSearch function in search.py. We encourage you to look through util.py for some data structures that may be useful in your implementation. You should now observe successful behavior in all three of the following layouts, where the agents below are all UCS agents that differ only in the cost function they use (the agents and cost functions are written for you):

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

Note: You should get very low and very high path costs for the StayEastSearchAgent and StayWestSearchAgent respectively, due to their exponential cost functions (see searchAgents.py for details).

Question 2: A* search (25 Points)

Implement **A* graph search** in the empty function aStarSearch in search.py. A* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information). The nullHeuristic heuristic function in search.py is a trivial example.

You can test your A* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as manhattanHeuristic in searchAgents.py).

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
```

You should see that A* finds the optimal solution slightly faster than uniform cost search (about 549 vs. 620 search nodes expanded in our implementation, but ties in priority may make your numbers differ slightly). What happens on openMaze for the various search strategies?

NOTE:

- Download *search.zip*, unzip and rename it with your *group number*.
- Write your code in respective files.
- Test your code and submit it on moodle.
- Due date for the Assignment is *14th February 2023 (11:59PM)*
- Penalty for late submission is *10%* of secured marks.
- We will run a plagiarism check for all the submissions, if found copied *100%* penalty will be applied.
- Viva and demonstration of your submitted code is mandatory and will be conducted on *15th February 2023* in lab hours, we will share the time slots for the same.