Homework 1 (Due January 26)

CS3642: Artificial Intelligence
Kennesaw State University
Spring 2024

In this project, we will model a maze solving problem as a state space problem, which can then be solved using search.

Consider the following maze, drawn out using ASCII text:

```
###############
#+.........#o#
#####.#.#####.#
#.....#....o#.#
#.#####.#####.#
#.....#.....#x#
###############
```

This maze is an $m \times n$ grid of cells, where we use the following symbols:

- a plus (`+`) denotes a player in the maze;
- a cross (`x`) denotes the goal cell;
- a dot (`.`) denotes an empty cell;
- a hash (`#`) denotes a wall that the player cannot enter;
- a circle (`o`) denotes a portal/teleporter which the player can use to teleport to another part of the maze

Our goal is to find a shortest path that allows the player to reach the goal cell. The player can perform five different actions (each action has a cost of one):

- move `UP` a cell
- move `DOWN` a cell
- move `LEFT` a cell
- move `RIGHT` a cell
- or `TELEPORT` from one portal (`o`) to another portal (`o`)

The player cannot move into a cell containing a wall. The player can only teleport when they are occupying a cell containing a portal. In this project, a maze may either contain exactly two portals, or none at all.

This project is based on the python code repository of the book "AI: A Modern Approach" (AIMA) by Russel & Norvig, which is available at `https://github.com/aimacode/aima-python`. We will use their abstract class `Problem` which allows us define a search problem, which can then be subsequently solved with one of a number of provided search algorithms. Note that their python code is based on Python version 3.7 and up. If you have not coded in python before, I suggest downloading the free version of PyCharm, which is an IDE for developing and debugging python programs.

Skeleton code is provided to help you get started. It already includes the relevant files from the AIMA repository, so you do not have to download the AIMA repository yourself. `maze.py` is the file that you will modify and turn in. `test.py` is an included testing script that is the main script to run. `search.py` includes the abstract class `Problem`, as well as other examples (including `EightPuzzle` for the sliding tile puzzle), which I also strongly suggest that you read. `maze.py` includes three classes:

- a `Maze` class which is a simple class that will take in the definition of a maze as a string, and convert it into a matrix, which can be indexed using the `at` function. The `at` function takes in a coordinate (x, y) as input, and returns a symbol representing the cell of the maze (i.e., whether it is empty, a wall, etc.)
- a `MazeState` class which is simply just a maze and a position for the player (this suffices to define the state of a player in a maze).
- a `MazeProblem` class, which is incomplete. The goal of this project is to complete this definition of a maze problem, so that it can be solved by one of the search algorithms already included in the AIMA repository. Once properly implemented, the included `test.py` script will run tests over multiple different mazes, each time showing the maze, the solution found, and the number of steps (actions) required to solve the maze (each action counts as one step).

Turn in: your modified version of `maze.py` onto the course website under `Assignments` and `Homework 1`. Assignments are due Friday, January 26 by 11:59pm. Please start early in case you encounter any unexpected difficulties.

Included files:

- `homework01.pdf`: this document
- `maze.py`: this includes a skeleton of the `MazeProblem` class that you want to complete.
- `test.py`: this file includes several tests of your `MazeProblem` class. **Make sure every test passes before you submit your assignment.** This file contains the public tests, whereas your assignment will be graded based on private tests that will be revealed when the solutions are posted.
- `search.py` and `util.py` come from the AIMA python repository.

Hint: Read the code. USE THE DEBUGGER. For example, insert this line in your code:

```
import pdb; pdb.set_trace()
```

and whenever your code runs this line, it will start the command-line debugger at that point in the code.