CSCE 580: Artificial Intelligence
Codinw HW 1: Pathfinding
Due: 1/24/2024 at 11:58pm

When debugging, use this code to set a breakpoint.

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

Your code must run in order to receive credit.

Do not change the signature of the functions. Your code must accept the exact arguments and return exactly what is specified in the documentation. You can use helper functions as long as they are contained in the same python file.

Installation

We will be using the same `conda` environment as in Homework 0.

The entire GitHub repository should be downloaded again as changes were made to other files. You can download it with the green "Code" button and click "Download ZIP".

Helper Classes and Functions

`Node`: the Node class used in the search algorithms

`get_next_state_and_transition_cost(env, state, action)`: gets the resulting state and transition cost of taking the given action in the given state.

`visualize_bfs(viz, closed_set, queue, wait)`: to visualize your search algorithm while it is running for the AIFarm (can be used for debugging).

`env.is_terminal(state)`: returns True if the state is a goal state and False otherwise.

`env.get_actions(state)`: gets the actions (a list of integers) available in that state

For this homework, you are welcome to use the AIFarm to debug your code. You can do this by using `--env aifarm` when running `run_assignment_1.py`. The cost of an optimal path for the default map is 22. You can make your own maps and use them with the `--map` switch. You can dynamically use a different heuristic function based on which environment you are using by checking the type of state. For example `type(state) is FarmState` or `type(state) is NPuzzleState`. When you turn in your code, make sure that it is optimized for the 8-puzzle.

To access the tiles of a 8-puzzle state, use `state.tiles` to obtain the numpy array. You can reshape the array into the 3x3 representation using `state.tiles.reshape((3, 3))`.

1 Finding Optimal Paths for the 8-Puzzle (80 pts)

The longest shortest path for the 8-puzzle is 31. For this exercise, you will be randomly given states from the 8-puzzle whose optimal path costs range from 0 to 31. For each state, your implementation of `search_optimal` will be called. After finding a path for all states, you should see that your search method finds an optimal path 100% of the time and the **total** time your code takes to find an optimal path for **all** states (not just one individual state) should be no longer than 3 minutes.

To run:

```
python run_assignment_1.py --env puzzle8 --type optimal
```

2 Finding Paths for the 8-Puzzle Quickly(20 pts)

This exercise will follow the same procedure as the previous one except that we no longer care about finding optimal paths. Instead, we only care about wall-clock time. For each state, your implementation of `search_speed` will be called. After finding a path for all states, you should see that the **total** time your code takes to find a path for **all** states (not just one individual state) should be no longer than 10 seconds.

To run:

```
python run_assignment_1.py --env puzzle8 --type speed
```

3 Extra Credit (10 pts)

For Exercise 1, find **optimal** paths for **all** states in 25 seconds or less. It is possible to obtain partial credit for implementations that come close to this time.

You are welcome to look up ideas (not involving code) for heuristics and other search procedures, however, you **MUST** cite any ideas in your submission and you **CANNOT** copy any code, whatsoever.

Successfully Running the Code

When successfully run on the 8-puzzle, your code should output:

```
Average difference with optimal path cost: <DIF>, %Optimal <OPT>%, Total time: <TIME> secs
```

What to Turn In

Turn in your implementation of `coding_hw/coding_hw1.py` to Blackboard.