CSCE 790-001: Deep Reinforcement Learning and Search
Coding Homework 3
Due: 11/16/2021 at 11:59pm

Your code must run in order to receive credit.
For grading purposes, do not change the signature of the function. Your code must return exactly what is specified in the documentation.

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".

A* Search (100 pts)

For this homework, you will implement A* search and, with only a one-line modification, you will also implement weighted A* search, greedy best-first search, and uniform cost search. The A* search algorithm is outlined in Algorithm 1.

The only function to implement is `astar` in `code_hw3.py`, however, you are free to implement helper functions, if you like.

You will be using A* search to solve the 8-puzzle. The heuristic function given to the A* search algorithm comes from a deep neural network that was trained with the deep approximate value iteration in Coding Homework 2. The loading of this deep neural network and conversion to a callable heuristic function is already done for you. The initialization of A* search is also done for you. Use this to understand the functionality of the functions and data structures.

A `Node` class is already implemented and has the following properties:
`state`: the state associated with the node
`path_cost`: the cost to go from the start state to the current state
`parent_action`: the action taken from the parent node to get to the current node
`parent`: the parent node

When you find a goal node, use `node.parent` and `node.parent_action` to find a solution. Keep in mind that, for the root node, `node.parent is None` and `node.parent_action is None`.

There are 287 states of varying difficulty. Each solution is checked to make sure it indeed finds a path to the goal. If not, an exception is thrown.

To run A* search: `python run_code_hw_3.py --weight_g 1.0 --weight_h 1.0`
The average cost of your solutions should be close to 16.36 and this should take around 13 seconds.

To run weighted A* search: `python run_code_hw_3.py --weight_g 0.9 --weight_h 1.0`
The average cost of your solutions should be close to 16.43 and this should take around 10 seconds.

To run another weighted A* search: `python run_code_hw_3.py --weight_g 0.5 --weight_h 1.0`
The average cost of your solutions should be close to 17.36 and this should take around 8 seconds.

To run greedy best-first search: `python run_code_hw_3.py --weight_g 0.0 --weight_h 1.0`
The average cost of your solutions should be close to 22.08 and this should take around 8 seconds.

To run uniform cost search: `python run_code_hw_3.py --weight_g 1.0 --weight_h 0.0`
You do not have to wait for this to terminate as the time to find a path for all states will be too long.

Algorithm 1 A* Search

```
1: procedure A* Search(s,h,$\lambda_g$, $\lambda_h$)
2:     Initialize OPEN, CLOSED
3:     Create node root with state s
4:     push root to OPEN with cost $\lambda_g$*root.path_cost + $\lambda_h$*h(root)
5:     CLOSED[root] = root.path_cost
6:     while OPEN is not Empty do
7:         pop highest priority node n from OPEN
8:         if n is a goal node then
9:             return get_soln(n)
10:        end if
11:
12:        for child in expand(n) do
13:            if child not in CLOSED or child.path_cost < CLOSED[child] then
14:                CLOSED[child] = child.path_cost
15:                push child to OPEN with cost $\lambda_g$*child.path_cost + $\lambda_h$*h(child)
16:            end if
17:        end for
18:    end while
19:    return failure
20: end procedure
```

What to Turn In

Turn in your implementation of to `code_hw/code_hw3.py` to Homework/Coding Homework 3 on Blackboard.