Homework 2
Computer Science
B351 Spring 2017
Prof. M.M. Dalkilic

Luke Doman

February 3, 2017

All the work herein is mine.

1.

- State: The current environment
- State space: All possible environments
- Search Tree: Graph to be navigated during search
- Search Node: A state within a search tree
- Goal: Desired environment in search
- Action: Transition from one state to another
- Transition Model: Logic governing which actions to take
- Branching Factor: Number of states reachable from a given state

2.

When the goal state is several nodes deep and reachable by DFS without requiring significant expansion of the frontier. Ex. A tree of depth 2- where the goal is the leftmost node at depth 10. DFS can reach this node by only expanding 10 nodes. Iterative deepening behaves like BFS so it would expand 2^9 nodes before finding it.

3.

No, because $h(B) = 4$ violates the rules for consistency. The cost of going from {(A,B) to (B,C)} only costs 3.

4.

a.

Please see implementation in rv1.py.

b.

Please see implementation in rv2.py and util.py.

a.

For problem (a) I simply utilized the DFS that we made for the last assignment. I made a Maze object that parses the maze txt file and generates an adjacency list that the Graph object from the previous assignment could parse. For problem (b) I utilized A* search. The $\hat{h}$ is calculated off of the total number of moves required for a given tile to reach the goal tile. $\hat{g}$ is calculated by adding the cost of the moving to the next tile, plus all the moves required to get there along the way. Lastly, $\hat{f} = \hat{g} + \hat{h}$

5.

By incorporating betting into the game we remove one of the possible ends states - tieing. My ai was naive in its strategy for betting, but adhered to the same logic for choosing its move as in the last assignment. By removing tieing from the game and recording stats on win percentages, it appears that the chances of winning are not significantly impacted. After running a million simulations it was almost a tie overall. Please see implementation and recorded stats in rpsg.py.