# CS4341

# Assignment #1

# Due date: 1/27/17 @ 11:59 p.m.

# Goals

This assignment will familiarize you with A\* search, the use of different heuristic functions, computing effective branching factor, and writing up your results. You should use the graph version of A\* search.

# The task

Your mission is to write an agent program to help a robot navigate some inhospitable terrain. The world is represented as a rectangular array, with each cell containing one of:

1. A symbol 1 through 9, representing the complexity of the terrain at that location (higher is more complex terrain). There is no guarantee that a particular number will occur in a given map.
2. A “#” symbol indicates unnavigable terrain. The robot may not move into such a square.
3. S, representing where you start this task. You may assume there is a unique start location. You should assume the robot is initially facing “North” (towards the top of the screen). The start state has a terrain complexity of 1 by default.
4. G, representing the goal location. You may assume there is a unique goal location. The goal state has a terrain complexity of 1 by default.

# Scoring

1. The agent receives a score of +500 points for reaching the goal state and the trial terminates.
2. Each unit of time the agent spends before the trial terminates is worth -1 point.

# Actions:

1. Forward. Moves the agent one unit forward on the map without changing its facing direction. Time required: the terrain complexity of the square being moved **into**.
2. Leap. The robot powers up and makes a great leap forward. The effect is to move the robot 3 units forward on the map without changing its facing direction. Time required: 20 (ignores terrain complexity). Leap can take a robot over terrain with a “#” symbol, but is not permitted to land the robot there. The robot cannot Leap if doing so would take it off the edge of the map.
3. Turn Left / Turn Right. Turns the agent 90 degrees, either left or right. Time required: 1/3 of the numeric value of the square currently occupied (rounded up).

# Heuristics

Your heuristics will make use of the vertical and horizontal (absolute) distance between the robot’s current position and the goal.

1. A heuristic of 0. A solution for a relaxed problem where the robot can teleport to the goal. This value also provides a baseline of how uninformed search would perform.
2. Min(vertical, horizontal). Use whichever difference is smaller. This heuristic should dominate #1.
3. Max(vertical, horizontal). Use whichever difference is larger. This heuristic should dominate heuristic #2.
4. Vertical + horizontal, also known as Manhattan distance. This heuristic should dominate #3.
5. Find an admissable heuristic that dominates #4. A small tweak of #4 will work here. Hint: think about the robot’s facing direction
6. Create a non-admissable heuristic by multiplying heuristic #5 by 3. See the lecture notes on heuristics for why we might want to do such a thing.

# Program inputs and outputs

Your program should be called astar should accept a command line input of a filename, and which heuristic should be used (1 through 6). The file will be a tab-delimited file, meeting the specifications given above (see the included sample maze).

It should output on the screen:

1. The score of the path found.
2. The number of actions required to reach the goal.
3. The number of nodes expanded.
4. The estimated branching factor.
5. The series of actions (e.g., forward, turn left, forward, forward, …) taken to get to the goal, with each action separated by a newline.

# Writeup

Create 5 worlds, of varying complexity, for testing your program (computer generated is fine). To keep both you and the grader sane, the hardest world should complete on a PC in approximately 10 seconds.

You should run each world with each of the 6 heuristics. Record the score of the path found, the number of actions, and the number of nodes expanded. For heuristics 1 through 5, the score and number of actions taken should be identical.

Create a graph of number of nodes expanded for each of the 6 heuristics across the 5 worlds.

Create a graph that, for each heuristic and each world, gives the effective branching factor.

How do the 5 heuristics vary in effectiveness? How much gain is there to using *any* heuristic (#1 vs. #2)? Is #5 noticeably more effective than the other heuristics?

For heuristic #6: how does its solution quality compare with #5? Is it performing noticeably worse? How much more efficient is it?

# What you should hand in: a zip file containing

1. Your program. Include any instructions for how to execute the code.
2. The 5 worlds you created
3. Your writeup

# Sample board

There is an included sample board. The output should be something like:

Score: 17

Number of actions: 8

Number of nodes expanded: 30

Estimated branching factor: 1.53

Turn Left

Forward

Forward

Turn Right

Forward

Forward

Turn Right

Forward