# CS4341

# Assignment #1

# Due date: Friday, September 11, 2015 @ 5:00 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.

# 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. 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.
3. 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 +100 points for reaching the goal state and the trial ends.
2. Falling off the edge of the world is worth -100 points and the trial ends.
3. Each unit of time the agent spends before reaching the goal is worth -1 point.

# Actions:

1. Forward. Moves the agent 1 unit forward on the map without changing its facing direction. Time required: the terrain complexity of the square being moved **into**.
2. Bash. The robot powers up, and charges forward crashing through obstacles in its path. The effect is to move the agent 1 unit forward on the map without changing its facing direction. Time required: 3 (ignores terrain complexity), and the next action taken by the agent **must** be Forward. I.e., the after Bashing, the agent cannot turn, Demolish, or Bash; it must first move Forward at least once to recover its balance.
3. Turn. Turns the agent 90 degrees, either left or right. Time required: 1/3 of the numeric value of the square currently occupied (rounded up).
4. Extra credit: Demolish. The robot uses high-powered explosives to simplify the task. The explosives clear all 8 of the adjacent squares (excluding the square inhibited by the robot, fortunately) and replaces their terrain complexity with 3 due to residual rubble. Time required: 4. Note: this action can increase terrain complexity if the initial complexity of the square is less than 3. Also note that if the agent considers using Demolish, but the search backtracks, you must ensure that the correct terrain complexity is restored to the map.

# 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. Sum the differences together. This heuristic should dominate #3.
5. Find an admissable heuristic that dominates #4. A small tweak of #4 will work here.
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 series of actions (e.g., forward, turn, forward, forward, …) taken to get to the goal, with each action separated by a newline.

# Analysis

Create 5 worlds, of varying complexity, for testing your program (computer generated is fine). 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.

Given the number of actions required, compute the effective branching factor for each of the 5 worlds.

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

A sample board is provided as further documentation as a means of testing your code. For this board, the shortest solution is: bash (3), forward (4), turn (2), forward (1), for a total cost of 10, and a score of 90.