# **Assignment 1: A\* search**

# **Due: January 31, 2022 @ 1:**59 pm

# **Goals**

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

**The task**

Your mission is to write a 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 number 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. 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. Left / Right. Turns the agent 90 degrees, either left or right. Time required: 1/2 of the numeric value of the square currently occupied (rounded up).
3. 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., after performing Bash, the agent cannot Turn, Bash, or Demolish; it must first move Forward at least once to recover its balance.
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.

The agent cannot move Forward or Bash if doing so will take it off the map. It can Demolish at the edge of the map; blowing up squares that are off the map has no effect.

# **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 value 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, and is a relaxation of each square having a move cost of 1 (optimistic).
5. Find an admissible heuristic that dominates #4.
6. Create a non-admissible heuristic by multiplying heuristic #5 by 3.

# **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, right, forward, forward, …) taken to get to the goal, with each action separated by a newline.

# **Analysis**

You can either hand generate maps or randomly create them. Generate 10 boards of a size where Heuristic #1 can solve them in approximately 30 seconds.

Solve each world with each of the 6 heuristics. As a sanity check, for heuristics 1 through 5, the score should be identical.

Record the number of nodes expanded for each of the 6 heuristics. What is the mean effective branching factor for each of the 6 heuristics? 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?

Finally, investigate the maximum problem size your program can handle. How large of a map can you solve in 30 seconds using Heuristic 5? How much memory is needed? Perform a “back of the envelope” calculation for how large a board you could solve with 16 GB of memory using each of the 6 heuristics (you can assume the board is square for this analysis). How large of a board could you solve with Heuristic 6 and 16GB of memory? How long would it take to solve that board? How much memory would you need for a problem that requires 24 hours to solve with Heuristic 5? With heuristic 6?

# Hand **in:**

1. Your program. Include any instructions for how to execute the code.
2. Your writeup

# **Sample board**

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

# Note

The TAs have a lot of grading to do. Submissions that violate the input or output format will lose points. Code that is needlessly difficult to run will lose points. The amount of penalty will be at the discretion of the TAs and the course instructor will back them up. Make the TA’s job easy and your life happier by following the instructions.