# COMP 2019 Assignment 1 – A Star Search

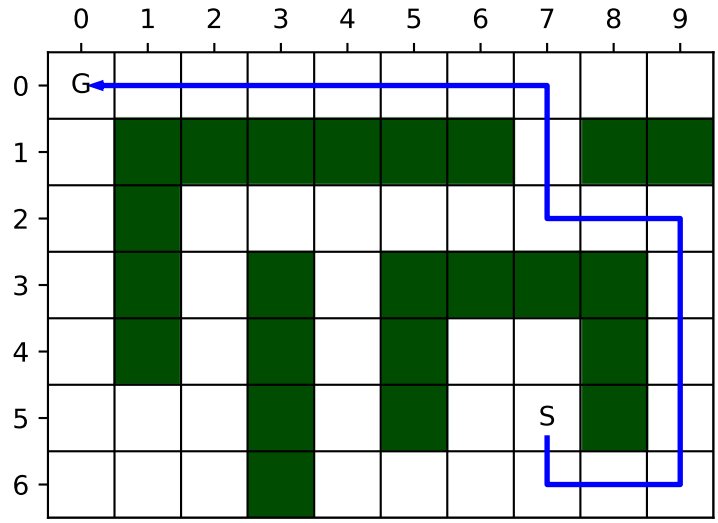
Please submit your solution via LEARNONLINE. Submission instructions are given at the end of this assignment.

This assessment is due on **Sunday, 14 April 2019, 11:59 PM.**
This assessment is worth 20% of the total marks.

This assignment consists of two parts. In the first part, you will implement a version of the A* algorithm for path finding; in the second part, you will tackle a more difficult variant of the same problem.

## Question 1

Consider the example of a simple game where agents move on a rectangular world map:




Your agent has been dropped at the location (5,7) marked 'S' and must reach the goal (the location (0,0) marked 'G') as quickly as possible. Your agent's movements are restricted to the four primary directions: North, South, East, and West, and the agent must stay within the bounds of the map.

Each location is annotated with a 0 (white background cells in the figure above) or a 1 (green background cells in the figure). Locations annotated with a 0 can be entered and traversed, whereas locations annotated with a 1 are inaccessible. Each action (moving from one cell to an adjacent cell) incurs a cost of 1. The cost for a path is the number of actions on the path. The unique best path in the example above is (5,7)(6,7)(6,8)(6,9)(5,9)(4,9)(3,9)(2,9)(2,8)(2,7)(1,7)(0,7)(0,6)(0,5)(0,4)(0,3)(0,2)(0,1)(0,0) and has a path cost of 18.

*Create a Java program that uses the A* algorithm to find a shortest path for an agent given the agent's Start- and goal locations and a world map showing the terrain accessibility at each location. If there are multiple shortest paths for a given map and agent, it is sufficient if your program finds one of the paths. You will need to pick a suitable heuristic function. The program must calculate the path and the total path cost.*

*Your program must be able to process any valid grid map and start and goal locations.*

*Your program should be as efficient as possible when using A* (in terms of the number of visited search states).*

*You are given sample code that reads the world map and establishes the classes and interfaces you must use for this assignment. Your code will be tested automatically and will break if you don't adhere to the given framework.*

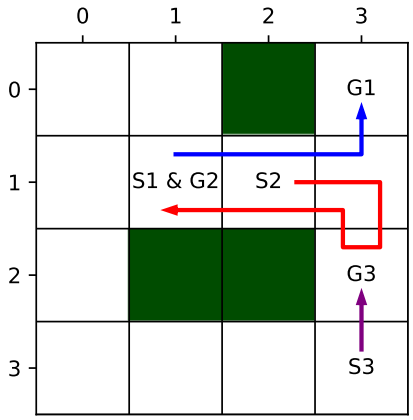
Your program can be run either via the JUnit tests, or from the terminal. For example:

```
java –cp bin: comp2019_Assignment1.Question1 resources/terrain01.txt 5 7 0 0
```

## Question 2

In this question you will implement cooperative pathfinding for multiple agents. This problem is more difficult as agents can obstruct the paths of other agents. Each agent must take the paths of other agents into account when planning its own path. We will consider the simplest variant of this idea, where each agent has a fixed priority and agents plan their paths in order of priority. Thus, the agent with the highest priority can usually follow its optimal path, whereas agents with lower priorities may need to take detours to accommodate the paths of agents with higher priority. Moreover, an agent may need to wait at a location until another agent has moved out of their way. A 'Wait' action is introduced for this purpose. Waiting at a location incurs a cost of 1.

For example, consider three agents who concurrently navigate the map given below:




Agent 0 (blue) starts in location (1,1) and aims to reach goal location (0,3). Agent 1 (red) starts in location (1,2) and aims to go to (1,1). Agent 2 (purple) starts in (3,3) and aims to go to (2,3). Suppose that Agent 0 has highest priority and gets to choose its path first, then Agent 1 gets to plan its path, and then Agent 2. According to A*, Agent 0 will travel along the path (1,1)(1,2)(1,3)(0,3) at cost 3. Although Agent 1 is only one step away from its goal location, the direct path (1,2)(1,1) is not feasible, as this would entail a head-on collision with Agent 0. Moreover, Agent 1 in its initial position obstructs Agent 0's chosen path. Hence Agent 1 must move out of the way and take a different route. The shortest path for Agent 1 is (1,2)(1,3)(2,3)(1,3)(1,2)(1,1) at cost 5. Similarly, Agent 2 waits (for 2 time steps) in (3,3) until Agent 1 has entered and exited location (2,3). Its path is therefore (3,3)(3,3)(3,3)(2,3) at cost 3. (Alternatively, Agent 2 could use path (3,3)(2,3)(3,3)(2,3) or (3,3)(3,2)(3,3)(2,3), which bear the same cost.)

The Hierarchical Cooperative A* (HCA*) algorithm is capable of calculating paths for multiple moving agents. The HCA* algorithm improves upon the basic A* algorithm by considering which locations are occupied by agents at each time step. These locations are then treated as (temporary) obstacles when searching for other agents' paths.

Please read the paper by Silver (2005) that describes the core ideas of HCA*. Consider only HCA* and leave aside the more complicated windowed-variant, WHCA*. Hence the sections on WHCA* and the experimental evaluation are not relevant for this assignment.

*Create a Java program that finds a feasible shortest path for each agent from the agent's start to its goal location. The path must respect the restrictions on terrain accessibility and presence of other agents on the map at each time step. The program must calculate the path and path cost for each agent. Assume that all agents commence moving along their paths at the same time instant, one action per time step. A limit on the number of time steps to consider will be given so that your program can detect easily if no path exists for an agent. You can assume that the given limit is always greater (and can be much greater) than the actual maximal time step needed to find a path.*

*Your program should be as efficient as possible (in terms of the number of visited search states).*

*It is recommended that you tackle reservation table described in Silver (2005) first and use pre-computed distances between each map location. Once that works correctly, improve the efficiency of the search by replacing the pre-computed distances with on-demand calculations using the Reverse Resumable A* (RRA*) algorithm. Please note that, contrary to Algorithm 1 sketched in the paper, you may need to add the successors to the Open queue prior to the goal test in line 12. Otherwise, the distance calculation may be incorrect.*

*You are given sample code that reads the world map and establishes the classes and interfaces you must use for this assignment. Your code will be tested automatically and will break if you don't adhere to the given framework.*

Your program can be run either via the JUnit tests, or from the terminal. For example:

```
java –cp bin: comp2019_Assignment1.Question2 resources/terrain01.txt resources/agents01.txt 100
```

The 100 in the above line represents the maximum time step that your program shall consider when searching for paths.

## Supplied Code and Implementation Notes

Download the `COMP2019-2019-HCAStar-Students.zip` archive from the course website. This supplied code already provides the input and output routines, the main method, and a set of JUnit tests that your program must pass. You may need to adjust the Java Build Path to point to your Java JDK and the JUnit jar library (download if necessary). The tests require JUnit4 version 4.12.

**It is essential that you adhere to the given APIs**, as your code will be assessed using additional JUnit tests, like the ones supplied with the assignment specification.

A location on a grid map is represented as a pair (r,c) where r denotes the row number and c the column number. In this coordinate system, the top left corner of the map is at (0,0). See class `Location`.

**Your program must be able to process any valid grid map and agent list.** The scenarios given in this assignment specification and the JUnit test code are only examples. Other examples will be used to assess your implementation. A specification of the file format is given below. Code to read this format has been provided.

**Your program must be well-behaved.** Your programs must run to completion when invoked via JUnit tests. Ensure that your code does not wait for user input, that it does not read files other than the terrain and agent lists, it does not write to any files, and that it does not make use of networking APIs. Read only the files supplied as arguments to the program, and do not assume the presence or absence of other files and directories. Do not hard-code paths to files in the code, as this will cause the program to fail when run on another machine. Finally, do not keep program state in global/static variables, as this may cause failures if more than one JUnit test is run.

### Input File Format – Grid Map

The input file format is as follows:

```
# Zero or more lines starting with a hash char
# These are comments and are ignored.
# This is the map presented in the Question 2 example
0 0 1 0
0 0 0 0
0 1 1 0
0 0 0 0
```

Spacing around numbers within rows may vary.

The above file corresponds to the terrain map shown in the description of Question 1.

### Input File Format – Agent List

The agents list is formatted as follows:

```
# Zero or more lines starting with a hash char
# These are comments and are ignored.
#
# The fields are as follows (separated by one or more spaces):
# AgentPriority StartRow StartColumn GoalRow GoalColumn
#
# The following lines correspond to the three agents in the Question 2 example
0 1 1 0 3
1 1 2 1 1
2 3 3 2 3
```

You can assume that the agents will be listed in order of priority and that each agent has a unique priority. Priorities are consecutive integers starting at 0, where 0 denotes highest priority.

## Submission Instructions

Submit a single ZIP file containing the following:

The source code (*.java) files for Questions 1 and 2. Include all supplied source code plus any other code that is needed to compile and run the programs. Do *not* include compiled bytecode, jar archives, the JUnit framework, test files, or Eclipse/IntelliJ project files.

Please ensure that the submission is complete, and that all files compile and run without error when placed in the directory structure given in the project, and that all JUnit tests pass without user input. Maintain the package and class names used in the supplied code.

Please structure your code well and comment your code where appropriate.

## References

Silver, D. (2005) Cooperative pathfinding. In Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (pp. 117-122). AAAI Press.
http://www.aaai.org/Papers/AIIDE/2005/AIIDE05-020.pdf

## Marking Criteria

Programs that do not compile and run using JDK 1.8 (or newer) and JUnit 4 tests will receive zero marks.

| | |
|---|---|
| **Question 1:**<br>Correct implementation of A* and given search problem.<br>Works for all possible input maps specified on the command line and via JUnit tests.<br>Can find the optimal path and detect when no path exists.<br>Implementation explores the optimal number of candidate paths.<br>Algorithm completes within the allotted time. | 40 % |
| **Question 2:**<br>Correct implementation of HCA*.<br>Works for all possible input maps and agent lists specified on the command line and via JUnit tests.<br>Can find the optimal path for each agent and detect when no path exists.<br>Efficient implementation of the abstract distance function.<br>Algorithm completes within the allotted time. | 60 % |