COMP 2019 Assignment 1 – Battlefield

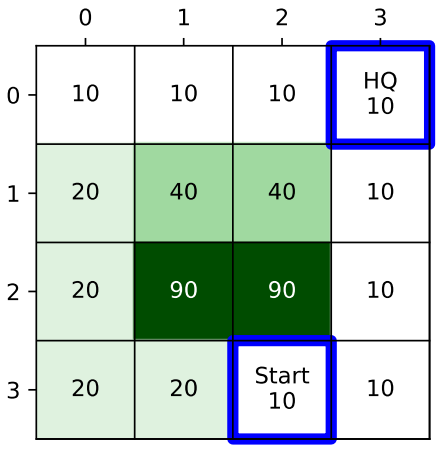
Please submit your solution via LEARNONLINE. Submission instructions are given at the end of this assignment.

This assessment is due on **Sunday, 9 April 2017, 11:55 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 a simple game where agents move on a rectangular world map:



Your agent has been dropped at the location marked 'Start' and must reach the Headquarters (the location marked 'HQ') as quickly as possible to deliver an important message to the General. 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. For this assignment, we shall assume that the map (and the locations of the enemies in Question 2) remain fixed.

The numbers shown for each location represent the difficulty of the terrain. The higher the number, the more difficult the terrain, and the more slowly your agent can traverse that location. For purposes of this assignment, the terrain difficulty indicators will all be strictly positive integers (that is, non-zero and positive whole numbers).

When moving from a location to an adjacent location, the cost associated with that move is calculated from the terrain difficulty of the two locations. Let a and b denote the terrain difficulty of the source and the target location, respectively. The cost associated with the move is calculated as $a + b$. The cost of a path is calculated as the sum of the individual move costs. Paths with lower costs are better. For example, the least-cost path from Start to HQ on the above map is (3,2)(3,3)(2,3)(1,3)(0,3).[1] The cost of this path is 80, as each move along the way costs 10+10=20, and the path requires 4 moves.

To make matters worse, your agent must carry a considerable amount of equipment and is unable to move through difficult terrain. More precisely, your agent cannot enter or traverse any location where the terrain difficulty strictly exceeds (>) a given threshold. For example, if the terrain threshold is 50, the agent can traverse the above path, but would be unable to follow the path to the north of the Start location in the above map; going West would be acceptable in this case.

[1] Each location is denoted as a pair (r,c) where r and c represent the row and column number, respectively. (0,0) is in the top left corner.

Your task for this question is to write a program that calculates the least-cost path for your agent to move from Start to HQ, while considering the terrain difficulty and the agent's inability to move through some terrain.
Implement a Java program that uses the A algorithm to find the best path, given the Start- and HQ locations, a world map showing the terrain difficulty at each location, and the agent's threshold for terrain difficulty. You will need to pick a suitable heuristic function.*

The program must calculate the path, the total path costs, and the number of explored locations.

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

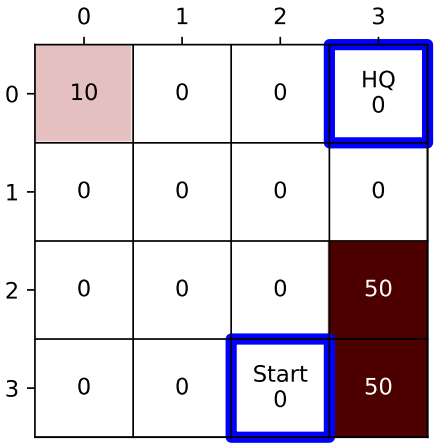
You are given sample code that reads the world map and establishes the classes and interfaces you must use for this assignment. Your code will be tested automatically and will break if you don't adhere to the given framework.

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

```
java -cp bin: comp2019_Assignment1.Question1 resources/terrain01.txt 3 2 0 3 50
```

Question 2

Unfortunately, the enemy has become aware of the agent and has dispatched troops in the area. Your agent has been given a map of the territory controlled by the enemy, in addition to the terrain map as in Question 1. Analogous to the terrain, the map shows the strength of the enemy's presence at each location. The dimensions of this map will be the same as that of the terrain map. All the strength indicators will be integers in the range [0,100] (with 0 and 100 included).



To avoid getting caught, your agent must find a path that is not only fast, but one that also accounts for the enemy's presence. For a path of length *k* that traverses locations with enemy strengths $e1, \ldots, ek$, the probability of survival, i.e. evading detection and capture, can be calculated as $(1 - e1/100) \times (1 - e2/100) \times \ldots \times (1 - ek/100)$. [2] For example, in the map above, your agent will be captured with 50% certainty if s/he enters location (3,3) from (3,2). The probability of evading the enemy on the path (3,2)(3,3)(2,3)(1,3)(0,3) is calculated as (1-0/100)*(1-50/100)*(1-50/500)* (1-0/100)* (1-0/100)=1*0.5*0.5*1*1=0.25.

Your agent is prepared to take only some amount of risk and must find a path that does not fall below a given survival threshold. The threshold is a real number in the interval [0.0,..,1.0]. Any path that falls below the given threshold is not an acceptable solution. If your agent is prepared to accept only paths with 0.5 or higher probability of survival, the shortest path may no longer be an acceptable solution. The path (3,2)(2,2)(1,2)(0,2)(0,3) is acceptable in terms of risk (the survival probability is 1.0), however it may not be acceptable given the terrain difficulty or the path cost. Your agent must find the *least-cost* path that simultaneously meets both the acceptable risk threshold and the terrain threshold. In the example, the least-cost path for a given terrain threshold of 50 and survival probability of 0.75 is (3,2)(3,1)(3,0)(2,0)(1,0)(0,0)(0,1)(0,2)(0,3). It has a total cost of 240, and a survival probability of 0.90.

Write a program that finds the least-cost path from Start to HQ that the agent can traverse, given the restrictions on terrain difficulty (as in Question 1) and survival probability. The program must calculate the path, the total path costs, and the number of explored states. Your program should be as efficient as possible (in terms of 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.

---

[2] Intuitively, for each location, if e/100 indicates the probability of getting caught, then 1-e/100 indicates the probability that the agent can traverse undetected. We'll assume that this calculation can be carried out separately for each location. Then, the overall probability of the agent traversing the entire path undetected can be calculated as the product of the individual probabilities.

Your program can be run either via the JUnit tests, or from the terminal (all on one line):

```
java -cp bin: comp2019_Assignment1.Question2 resources/terrain01.txt resources/enemy01.txt 3 2 0 3 50 0.75
```

Hints:

You will need to modify your A* search to account for the possibility that there may be multiple paths to the same location with different costs and survival probabilities. Whereas only one such path is maintained in the "plain" A* search variant that you used in Question 1, solving this problem requires you to maintain multiple paths. For example, assume that a path with cost 100 and survival probability 0.80 to location L is found first; then a second path to L is found with cost 120 and probability 0.90. Both paths must be considered, as the path with better cost may later turn out to not reach the goal (if the probability becomes too low). The other path may however be able to reach the goal, albeit at a higher total path cost. Now consider a third path to L with cost 105 and probability 0.7. It is not useful to keep this path, as it is strictly worse in both cost and probability than another path to L. If a fourth path to L with cost 105 and probability 0.90 (or better) were found, the second path should be discarded.

Supplied Code and Implementation Notes

Download the `COMP2019-2017-Agent-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 should be able to import this into Eclipse. You may need to adjust the Build Path to point to your Java JDK and the JUnit jar library. The tests require JUnit4 version 4.12 or higher.

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 world(s) and parameters. The world 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 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.
10   10   10   10
20   40   40   10
20   90   90   10
20   20   10   10
```

Spacing around numbers within rows may vary.

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

The enemy map for Question 2 is formatted accordingly:

```
# enemy01
10    0    0    0
 0    0    0    0
 0    0    0   50
 0    0    0   50
```

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 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 Eclipse 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.

Marking Criteria

Programs that do not compile and run on Mac OS X in the Mac pool (JDK 1.8 and JUnit 4 tests) will receive zero marks.

Question 1: Correct implementation of A* and given search problem. Works for all possible input maps and threshold specified on the command line and via JUnit tests. Can find the optimal path and detect when no path exists. Implementation explores the optimal number of candidate paths. Algorithm completes within the allotted time.	60 %
Question 2: Correct implementation of A* and given search problem. Works for all possible input maps and threshold specified on the command line and via JUnit tests. Can find the optimal path and detect when no path exists. Implementation explores the optimal number of candidate paths. Algorithm completes within the allotted time.	40 %