COMP 2019 Assignment – Heuristic Search

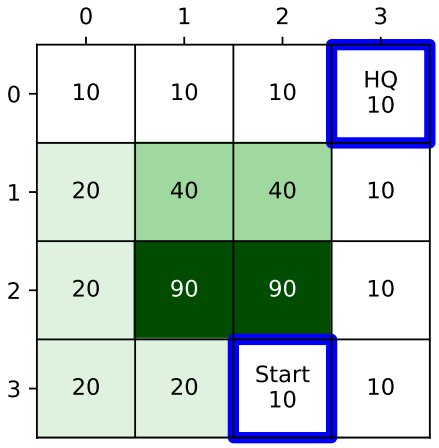
Please submit your solution via LEARNONLINE. Submission instructions are given at the end of this assignment.

This assessment is due on **Sunday, 25 September 2022, 11:55 PM.**
This assessment is worth 30% 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 an agent moves 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. 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 throughout the search. However, your program must work with any given maps, not just the example given in this document.

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 is (3,2)(3,3)(2,3)(1,3)(0,3) in the above map.[1] The cost of this path is 80, as each move along the way costs 10+10=20, and the path requires 4 moves.

Moreover, the agent 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 aforementioned 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 Python program that uses the A algorithm to find the best path, given the Start- and HQ locations, a terrain 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. Please write a separate function for your heuristic and embed a function comment in your source code in which you describe what heuristic you have chosen, whether the function is consistent (why/why not?), and why you have chosen this heuristic. Remember that choosing a heuristic function is essential for obtaining an efficient search algorithm.

The program must calculate the path and the total path cost. Moreover, your algorithm will need to call specific functions throughout the search that will be used to assess the correctness of your implementation. A list of these functions is given in the Appendix.

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

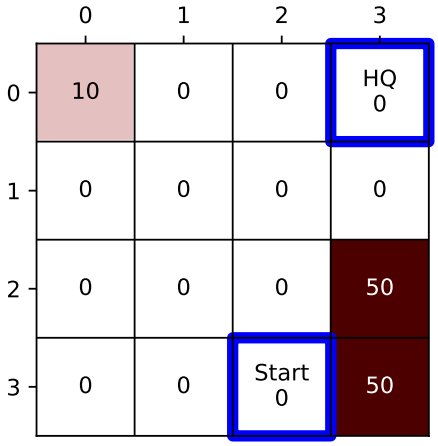
You are given sample code that reads the world map and provides the classes and interfaces that you must use for this assignment. Your code will be tested automatically and break if you don't adhere to the framework.

Your program can be run either via the python unit tests or from the terminal:

```
python pathfinding_task1.py 3,2 0,3 resources/terrain01.txt 50
```

Question 2

The 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 are the same as that of the terrain map. Each cell on the map provides an indicator of the enemy's presence in the form of an integer number in the range [0,…,100].



To avoid getting caught, the agent must find a path that not only has low cost but one that also accounts for the enemy's presence. The agent must account for the enemy's presence and choose a path that evades detection and capture to ensure mission success. For a path of length *k* that traverses locations with enemy strengths $e1, \ldots, ek$, the probability of success can be calculated as $(1 - e1/100) \times (1 - e2/100) \times \ldots \times (1 - ek/100)$. [2] For example, in the map above, the agent will be captured with a 50% probability if the agent enters the location (3,3). The probability of success on the path (3,2)(3,3)(2,3)(1,3)(0,3) is calculated as (1-0/100)*(1-50/100)*(1-50/100)* (1-0/100)* (1-0/100)=1*0.5*0.5*1*1=0.25.

The agent is prepared to take only some amount of risk and must find a path that does not fall below a given success threshold. The success 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 a 0.5 or higher probability of success, 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 success threshold and the terrain threshold. In the example, the least-cost path for a given terrain threshold of 50 and success 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 success 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 success probability (Question 2). The program must calculate the path, the total path costs, and success probability. Moreover, your algorithm will need to call specific functions throughout the search that will be used to assess the correctness of your implementation. A list of these functions is given in the Appendix.

[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.

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

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

```
python safe_pathfinding_task2.py 3,2 0,3 resources/terrain01.txt 50 resources/enemy01.txt 1.0
```

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 success 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 a cost of 100 and a success probability of 0.80 to a location L is found first; then a second path to L is found with a cost of 120 and a success probability of 0.90. Both paths must be considered, since the path with better cost may later turn out to not reach the goal (if the success 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 a cost of 105 and a probability of 0.7. It is not useful to keep this path, since it is strictly worse in both cost and probability than another path to L. If a fourth path to L with a cost of 105 and probability of 0.90 (or better) were found, the second path should be discarded.

Supplied Code and Implementation Notes

Download the `COMP2019-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 unit tests that your program must pass. The file requirements.txt lists the prerequisite packages.

It is essential that you adhere to the given APIs, as your code will be assessed using additional unit tests, like the ones supplied with the assignment specification. Your program must work with Python version 3.10 or higher and must not depend on any packages other than those listed in requirements.txt and those that are part of the standard 3.10 python runtime environment.

Each 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).

Your program must be able to process any valid grid maps and parameters. The maps given in this assignment specification and the unit tests are only examples. Other examples will be used to assess your implementation.

Your program must be well-behaved. Your programs must run to completion when invoked via unit tests. Ensure that your code does not wait for user input, that it does not read files other than the terrain and enemy maps, 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 the program state in global variables or use concurrent programming techniques, since this may cause failures if more than one unit test is run.

Implementation Requirements

Your implementation must invoke the following functions throughout the search:

- `log_visit_state(location, cost, p_success)` each time a candidate path is visited.
- `log_enqueue_state(location, cost, p_success)` each time a candidate path is added to the frontier.
- `log_ignore_state(location, cost, p_success)` each time a candidate path is ignored because it cannot be the best answer.

Please refer to the provided source code to learn the meaning of these functions' parameters.

Submission Instructions

Submit a single ZIP file containing the following:

The source code (*.py) files for Questions 1 and 2. Include all supplied source code and any other code needed to compile and run the programs. Do *not* include compiled bytecode, test files, version control metadata, or IDE configuration files.

Please ensure that the submission is complete, that all files run without error when invoked from the command line and unit tests, and that all unit tests pass without requiring any user input. Maintain the file and function names used in the supplied code.

Please structure your code well and include comments in your code where appropriate.

Marking Criteria

Programs that do not run using python 3.10 and those that make use of packages other than those set out in this document will receive zero marks.

Question 1: Correct implementation of A* and given search problem. Choice of heuristic function described and justified Works for all possible input maps and thresholds specified on the command line and via unit tests. Can find the optimal path and detect when no path exists. Visits the optimal number of states. Complies with the specified implementation requirements. Clean code, well-structured.	40 %
Question 2: Correct implementation of A* and given search problem. Works for all possible input maps and thresholds specified on the command line and via unit tests. Can find the optimal path and detect when no path exists. Visits the optimal number of states. Complies with the specified implementation requirements. Clean code, well-structured.	60 %