Programming Assignment 1: Infiltration & Exfiltration

Due: Friday 02/28/2025 @ 11:59pm EST

The purpose of programming assignments is to use the concepts that we learn in class to solve an actual real-world task. To that end you will be writing java code that uses a game engine called Sepia to develop agents that solve specific problems. In this lab we will be invading enemy territory. Our footman unit (i.e. meele unit) is tasked with infiltrating enemy territory to destroy the enemy base (called a "Townhall" in Sepia). A Townhall will appear with the letter "H" while enemy archers (i.e. ranged units) will appear with an "a". The enemy archers **will** attack you if you get too close, so you are not guaranteed to survive this mission.

1. Copy Files

Please, copy the files from the downloaded lab directory to your cs440 directory. You can just drag and drop them in your file explorer.

- Copy `Downloads/pa1/lib/pas-stealth.jar` to `cs440/lib/pas-stealth.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/pa1/data/pas` to `cs440/data/pas`.
  This directory contains a game configuration and map files.
- Copy `Downloads/pa1/src` to `cs440/src`.
  This directory contains our source code `.java` files.
- Copy `Downloads/pa1/pas-stealth.srcs` to `cs440/pas-stealth.srcs`.
  This file contains the paths to the `.java` files we are working with in this lab. Files like these are used to speed up the compilation process by preventing you from listing all source files you want to compile manually.
- Copy `Downlaods/pa1/doc` to `cs440/doc`. If you don't have a `cs440/doc` directory, please make one. This folder contains the autogenerated java documentation (javadoc for short) for the custom jarfile I made for you. After copying, if you double click `cs440/doc/pas/stealth/index.html`, it should open in your browser. This documentation can be treated just like the official java documentation, and will be extremely useful when using the apis I have created in the custom jarfile.

2. Test run

If your setup is correct, you should be able to compile and execute the given template code. You should see the Sepia window appear.

```
# Mac, Linux. Run from the cs440 directory.
javac -cp "./lib/*:." @pas-stealth.srcs
java -cp "./lib/*:." edu.cwru.sepia.Main2 data/pas/stealth/OneUnitSmallMaze.xml

# Windows. Run from the cs440 directory.
javac -cp ./lib/*;. @pas-stealth.srcs
java -cp ./lib/*;. edu.cwru.sepia.Main2 data/pas/stealth/OneUnitSmallMaze.xml
```

Task 3: Datatype Information

A note on the `Path` datatype contained within `lib/pas-stealth.jar`. A `Path` here is implemented as a reverse singly-linked list. The main reason for doing so is that we can forgo the use of a "parents" or "backpointers" table that you're probably used to seeing. To be clear, you are welcome to use backpointers, however you don't *have* to, the `Path` datatype will let you grow paths and remember where they came from on your behalf.
The way the `Path` datatype achieves this is through two strategies:

- It is built as a reverse singly-linked list. The way to build a path looks like this:

  ```
  Path p = new Path(src); // empty path for the src vertex
  p = new Path(v1, 1, p);    // grow the path using edge (src -> v1) w/ weight 1
  p = new Path(v2, 1.5, p);    // grow the path using edge (v1 -> v2) w/ weight 1.5
  ...
  ```

  After these three lines, variable `p` which has type `Path` will contain the entire path from `src` to `v2` with a total observed path cost of `2.5`.

- comparing two `Path` datatypes only compares the destination vertices of the two paths. So even though a `Path` contains the entire path from the source vertex, if you want to compare one `Path` against another, they pretend to only be their destination vertices. Graph search algorithms need to compare vertices against each other, which you can now do with `Path` objects directly.

Task 4: `StealthAgent.java` (100 points)

Your agent must use the A* algorithm to find routes between desired coordinates on the map. The goal for this agent is to infiltrate enemy territory without being seen, destroy the enemy townhall and then escape. This mission is fraught with peril: the enemy has archers (i.e. units that can attack from a distance) that will shoot you on sight while they are doing their daily task of chopping down trees. The enemy archers are incredibly strong: you will die if you try to engage with them. Therefore, you must be sneaky and get to the townhall location without being seen. Once you get to the townhall, you will need to repeatedly attack it until it is destroyed.

The second that the townhall is destroyed, you will need to escape and return to the square on the map that you started at. Once the enemy townhall is destroyed, the enemy archers will be alerted to your presence (presumably because you trip an alarm or something with the act of destroying the townhall) and will immediatly return to the location of the (former) townhall to investigate. While on the way, the archers can still see you if you get too close to them, so it is imperative that you find a safe path back to the starting square. After a certain amount of time after reaching the starting square, the enemy archers will die of grief: they simply cannot live without their townhall, at which point the mission is successful.

Much of the success of your agent will depend on how you calculate your edge weights in the **`getEdgeWeight`** method. You should design your edge weights to capture "danger" or "risk" (for instance, if an edge brings you closer to an enemy unit, you should certainly discourage A* from using that edge in the shortest path). Remember, A* is searching for the shortest path, so you should try and encode more "danger" or "risk" as larger and larger edge weights. This will involve quite a bit of creativity on your part, as there are many different forms of risk in this game (not just proximity to the enemy).

The `OneUnitSmallMaze.xml` game is **always** winnable, the `TwoUnitSmallMaze.xml` game is **mostly** winnable, and the `BigMaze.xml` game is **barely** winnable. For full credit, you must implement a correct A* algorithm, and your agent must meet some performance criteria on the three maps: your agent must win 100% of the games it plays on the `OneUnitSmallMaze`, 95% of the games it plays on the `TwoUnitSmallMaze`, and 20% of the games it plays on the `BigMaze`. Note that the autograder that checks performance will run many hundreds of games. This autograder will time out after 40min of runtime, so if the autograder times out, you need to make your code more efficient!

Rather than run the game by hand over-and-over again, I recommend creating a script on your machine (could be a bash script, a bat script on Windows, etc.). What this script should do is play the game a large amount of times with the rendering turned **off**. You can turn off the rendering by commenting out the lines for the `VisualAgent` in the `.xml` file you want to run. The lines to comment look something like this:

```
<Player Id="0">
    <AgentClass>
        <ClassName>edu.cwru.sepia.agent.visual.VisualAgent</ClassName>
        <Argument>true</Argument>
        <Argument>false</Argument>
    </AgentClass>
</Player>
```

If you comment these lines out and run the game, you won't see anything but the game will run much quicker. You can tell the outcome of the game based on the console output. Personally, I would include counting the number of successes in the script you write and then print out the success rate at the end, but I leave those design decisions up to you.

Task 5: Extra Credit (50 points)

In addition to an enemy townhall, the enemy has also collected a fair amount of gold. To earn the extra credit, you must achieve the same winrates as the normal credit, but additionally steal the gold from the enemy along the way. Don't worry, the gold reserve is sitting close to the enemy townhall, but be warned: stealing the gold **after** destroying the townhall is a bad idea! I would recommend stealing the gold and then destroying the townhall, which adds a little more complexity to your agent. But remember, when playing video games, being disrespectful is always worth it!

Task 7: Submitting your assignment

Please submit `StealthAgent` on gradescope (no need to submit a directory or anything, just drag and drop the file into gradescope). There will be two entries on gradescope for this pa. One entry will test the correctness of your A* algorithm, and another will test the performance of your agent on the three maps.