Programming Assignment 1: Infiltration & Exfiltration

Due: Friday 10/19/2023 @ 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 unit (the red one) 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 ("archer" units) will appear with a "a". The enemy soldiers **will** attack you if you are seen, 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/pa1.jar` to `cs440/lib/pa1.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/pa1/data/pa1` to `cs440/data/pa1`.
  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/pa1.srcs` to `cs440/pa1.srcs`.
  This file contains the paths to the `.java` files we are working with in this lab. Just like last 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.
- I would recommend taking a look at the documentation for the provided jarfile. I have included it in this package and will also post it on Piazza.

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/Sepia.jar:lib/pa1.jar:. @pa1.srcs
java -cp lib/Sepia.jar:lib/pa1.jar:. edu.cwru.sepia.Main2 data/pa1/OneUnitSmallMaze.xml

# Windows. Run from the cs440 directory.
javac -cp lib/Sepia.jar;lib/pa1.jar;. @pa1.srcs
java -cp lib/Sepia.jar;lib/pa1.jar;. edu.cwru.sepia.Main2 data/pa1/OneUnitSmallMaze.xml
```

3. Information on the provided files

- Directory `src/pa1/agents/` contains one .java file called `StealthAgent.java`. This is where I would like you to put your implementation. There is also another directory with a single file in it called `src/pa1/ec/DisrespectfulStealthAgent.java`. This is your extra credit (see more on the extra credit below).
- `cs440/lib/Sepia.jar`. The main type in Sepia is called an `Agent`, and is the type we want to extend in order to write our own agents that can interact with Sepia. I encourage you to become familiar with the Agent api as well as the Sepia api, because I have **not** implemented many of those methods for you. I have left some examples of how to discover units and so on in the `initialStep` method, but the rest of this is up to you. Here are the methods you will need to implement in order for the `Agent` to interact with Sepia. However, I have also left some helper methods that I believe you will find useful as a template for designing your implementation:
  - a Constructor. The `Agent` type in `Sepia` does not have a public constructor, so if you extend this class you **must** write your own. This constructor must take the player number (an `int`), and optionally may take a `String[] args` (just like a main method).
  - `initialStep(StateView, HistoryView)`. This method is called on the first turn of a game (and **only** called on the first turn of a game). We typically use it to discover quantities about the world we want to know (for instance, what are the units under my control, enemy control, etc.). It is common to set the fields of an `Agent` to some invalid state in the constructor and then to actually populate them in `initialStep`. This method returns a `Map<Integer, Action>`. Every entity in the game (unit or resource) has a unique integer id, and this return value is how your `Agent` tells each unit what to do. If you want a unit to take an action that turn, you put an entry in the map for that unit (using its id as a key) along with the action you want it to take.
  - `middleStep(StateView, HistoryView)`. This method is called every turn and should contain the main logic of your `Agent`. Just like `initialStep`, this method should return a mapping from (your) unit id(s) to the actions you want those units to take this turn.
  - `terminalStep(StateView, HistoryView)`. This method is called once (and only once) at the end of a game. To be clear, this method is called after the game has **completed**, so this method returns nothing (there is no game to perform actions in). This method is typically used for other purposes such as logging info, saving things, etc.
  - `savePlayerData(OutputStream)`. This method is used to save your `Agent` to disk. We won't make use of it very often.
  - `loadPlayerData(InputStream)`. This method, like its counterpart, is to load an `Agent` from disk. We won't make use of it very often.

  More information on `Agent` methods can be found in the Sepia Agent Tutorial.

Task 1: `StealthAgent.java` (100 points)

Your agent must use the A-star 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 (called a *primitive move* in Sepia) 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.

Note: their is a method built into `StealthAgent.java` that will tell you how many squares away an archer can detect enemy units (i.e. you). Be sure to account for this when planning your routes.

Notes

- You may create whatever helper methods you want in order to accomplish this goal. For instance, I would suggest, like last time, implementing a method to get the neighbors of the current `Vertex`.
- Please use **manhattan** distance as your heuristic for all `Agent`(s) you implement!
- When you want to test your `Agent`, please open up `data/pa1/[BigMaze | OneUnitSmallMaze | TwoUnitsSmallMaze].xml` in a text editor of some kind (your machine may default to opening it in your browser, we don't want that), you will need to look at line 3 in the following section:

```
1 <Player Id="1">
2    <AgentClass>
3      <ClassName>src.pa1.agents.StealthAgent</ClassName>
4      <Argument>0</Argument>
5           <Argument>2</Argument>
6    </AgentClass>
7  </Player>
```

  Change the ClassName line to the `Agent` that you want to run.

```
<ClassName>src.pa1.agents.StealthAgent</ClassName>
<ClassName>src.pa1.agents.DisrespectfulStealthAgent</ClassName>
```

  Do **NOT** change any of the arguments passed into any of the agents (the `<Argument></Argument>` tags) unless you know what you are doing.

Task 2: Extra Credit (100 points)

Your `src/pa1/ec/DisrespectfulStealthAgent.java` agent will work identically to your `StealthAgent` except for one key difference. In addition to an enemy townhall, the enemy has also collected a fair amount of gold. In the extra credit, you must achieve the same mission as the `StealthAgent`, but this time you must also steal all of the gold that the enemy team possesses. 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 3: Submitting your assignment

Please submit `StealthAgent`, and optionally, `DisrespectfulStealthAgent` on gradescope (no need to submit a directory or anything, just drag and drop the files into gradescope).