Lab 3: Infiltration & Exfiltration

Due: Tuesday 09/27/2024 @ 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 green one) is tasked with infiltrating enemy territory to destroy the enemy base (called a "Townhall" in Sepia), and escaping afterwards. 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. Once the enemy base is destroyed, the enemy soldiers will stop what they are doing and return to the site of their (now former) base to investigate. If you are still there, they will kill you! So, not only do you have to get to and destroy the enemy base without being seen, you must escape (back to the square your unit originally started at) without being seen. This is a very dangerous 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/lab3/lib/infexf.jar` to `cs440/lib/infexf.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/lab3/data/labs` to `cs440/data/labs`.
  This directory contains a game configuration and map files.
- Copy `Downloads/lab3/src/labs` to `cs440/src/labs`.
  This directory contains our source code `.java` files.
- Copy `Downloads/lab3/infexf.srcs` to `cs440/infexf.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.
- Copy `Downloads/lab3/doc/labs` to `cs440/doc/labs`. This is the documentation generated from `infexf.jar` and will be extremely useful in this assignment. After copying, if you double-click on `cs440/doc/labs/infexf/index.html`, the documentation should open in your browser.
- **NOTE:** for your benefit I have also included the file `Downloads/lab3/SpecOpsAgent.java`. This is the source code for the type your agent will extend. I included it if you wanted to see how the state machine works, my implementation of A*, etc. You don't need to copy it anywhere.

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/*:." @infexf.srcs
java -cp "./lib/*:." edu.cwru.sepia.Main2 data/labs/infexf/OneUnitSmallMaze.xml

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

Task 3: Datatype Information

A note on the `Path` datatype contained within `lib/infexf.jar`. A `Path` here is implemented as a reverse singly-linked list. The reason for this is twofold: to make it easier to "extend" paths, and to make comparison logic easier. One form of comparison logic in java is the `.equals(Object other)` method, which returns `true` if the `other` object is equal to `this` object. When creating a custom datatype, and you want to use it in, say a `Queue` or a `Stack`, you will need to implement this method for `contains()` to work correctly (by default `.equals(Other object)` will only check for *shallow copies* of the object, not *deep copies*). Two `Path` objects are considered equal if their *destinations* are the same, rather than the entire set of edges being equal. This is so that when you implement your methods, you can easily compare `Path`s together.
The other reason, as mentioned previously, is to make it easier to "extend" a `Path`. Creating a new `Path` here is as easy as this:

```
new Path(newDstVertex, edgeWeightFromOldDstToNewDst, oldPath)
```

where `oldPath` is the path you are trying to extend (i.e. "grow" by one edge). When doing search (I implemented this for you in this lab), we will be expanding paths a **lot**, so formulating paths like this is convenient to our needs *and* lets us use shallow copies of the shared paths (rather than deep copies so it also saves us memory).

Task 4: `InfilExfilAgent` (50 points)

Please take a look at `InfilExfilAgent.java` located in `src/labs/infexf/agents`. This agent has two methods that you need to complete: `getEdgeWeight` and `shouldReplacePlan`. Like last time, the `shouldReplacePlan` method returns `true` if the current plan is invalid (for instance if something blocks you), and `false` otherwise. My code will use this method to recalculate a plan whenever this method returns `true`.

The majority of the work for this lab is the `getEdgeWeight` method. This method returns the weight of an edge between two vertices, and is extremely important to the success of the mission. If your edge weights do not account for "danger" or "risk" (for instance, if an edge brings you within the attack radius of the enemy, you certainly should discourage A* from using that edge in a path). Basically, we want to assign edges that are bad large weights (so A* wont use them), and assign edges that are good small weights (so A* will use them). This will involve quite a bit of creativity on your part, which is why I have implemented A* (and the state machine of the agent) for you.

When testing, the two smallers maps `OneUnitSmallMaze.xml` and `TwoUnitSmallMaze.xml` are **always** winnable. For full credit, your agent must win 100% of these games that it plays (I will play a bunch of these games on the autograder).

Task 5 Extra Credit (50 points)

`BigMaze.xml` is **not** always winnable. The culprit for this (surprisingly high mortality rate) are *tunnels*. Tunnels have a high chance of being deathtraps: not only is there no manuverability for your agent, but the tunnel walls are made of trees which the enemy soldiers are tasked with cutting down (so soldiers have a large chance of flocking to tunnels). This makes `BigMaze.xml` a hard map. For full extra credit, your agent must beat `BigMaze.xml` at least 40% of the time.

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 6: Submitting your assignment

Please submit your `InfilExfilAgent.java` on gradescope (no need to submit a directory or anything, just drag and drop the files into gradescope).