Lab 2: Stealth with Graph Traversals

Due: Tuesday 01/30/2024 @ 11:59pm EST

The purpose of labs is to practice the concepts that we learn in class. 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 units (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" (in green) while enemy foot soldies ("footmen" units) will appear with a green "f" (our unit is a red footman unit). The enemy soldiers won't attack you if they spot you (yet) and won't retaliate once their townhall is destroyed (yet), so this is a rather survivable 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/lab2/lib/stealth.jar` to `cs440/lib/stealth.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/lab2/data/labs/` to `cs440/data/labs`.
  This directory contains a game configuration and map files.
- Copy `Downloads/lab2/src/labs` to `cs444/src/labs`.
  This directory contains our source code `.java` files.
- Copy `Downloads/lab2/stealth.srcs` to `cs440/stealth.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/lab2/doc/labs` to `cs440/doc/labs`. This is the documentation generated from `stealth.jar` and will be extremely useful in this assignment. After copying, if you double click on `cs440/doc/labs/stealth/index.html`, the documentation should open in your browser.

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/stealth.jar:. @stealth.srcs
java -cp lib/Sepia.jar:lib/stealth.jar:. edu.cwru.sepia.Main2 data/labs/stealth/EmptyMaze.xml

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

Task 3: Datatype Information

A note on the `Path` datatype contained within `lib/stealth.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, 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: BFS Agent (25 points)

Please take a look at the `BFSAgent.java` file located in `src/labs/stealth/agents`. This agent has two methods that you need to complete: `shouldReplacePlan` and `search`. The `shouldReplacePlan` method returns a `boolean`: `true` if the current plan is now invalid (for instance if something crosses your path), and `false` otherwise. The `search` method is where you should implement the BFS algorithm to find a path from the src vertex to the goal vertex. One thing to note is that your `search` method should produce a path that ends **before** the goal vertex: we actually don't want to try to occupy the enemy townhall's location, we just want to be next to it (so we can attack it). Please make use of the `Path` datatype found in the `lib/lab2.jar` file. Remember that BFS does **not** care about edge weights, so please make sure to use an edge weight of `1f`!
When you want to test your `BFSMazeAgent`, please open up the maze file that you want to run (either `data/labs/stealth/SmallMaze.xml` or `data/lab2/BigMaze.xml`). When you open this file 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="0">
2    <AgentClass>
3      <ClassName>src.labs.stealth.agents.BFSMazeAgent</ClassName>
4      <Argument>0</Argument>
5    </AgentClass>
6  </Player>
```

The line in my example

```
<ClassName>src.labs.stealth.agents.BFSMazeAgent</ClassName>
```

is **correct**: we want player 0 to run `src.labs.stealth.agents.BFSMazeAgent`. If this is not what that line says, you will need to change it.

Task 5: DFS Agent (25 points)

This task is the exact same as the previous one, only please implement the DFS algorithm in `src/labs/stealth/agents/DFSMazeAgent.java`. Remember to use an edge weight of `1f`: DFS also does not care about edge weights!

When you want to test your `DFSMazeAgent`, please open up the maze file that you want to run (either `data/labs/stealth/SmallMaze.xml` or `data/labs/stealth/BigMaze.xml`). When you open this file 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="0">
2   <AgentClass>
3     <ClassName>src.labs.stealth.agents.BFSMazeAgent</ClassName>
4     <Argument>0</Argument>
5   </AgentClass>
6 </Player>
```

The line in my example

```
<ClassName>src.labs.stealth.agents.BFSMazeAgent</ClassName>
```

is **wrong**: we want player 0 to run `src.labs.stealth.agents.DFSMazeAgent` instead of `src.labs.stealth.agents.BFSMazeAgent`.

Task 6: Extra Credit (25 points)

I have included an extra file `src/labs/stealth/agents/DijkstraMazeAgent.java`. Like the other two classes, you will need to implement the same methods, only this time make sure to implement Dijkstra's algorithm instead of DFS and BFS. Be sure to, and you will have to program this, consider action costs. Sepia has a `Direction` type that I have already imported for you, while the parent `MazeAgent` (of your `DijkstraMazeAgent` has a method called `getDirectionToMoveTo(Vertex src, Vertex dst)` which will return such a `Direction` (that you will need to take go to from `src` to `dst`). Be aware that this method expected `src` and `dst` to be neighbors of each other: this method will crash (on purpose) if the two vertices aren't! The `Direction` type has enum values which I want you to give individual edge weights to:

- If you need to move horizontally (`Direction.EAST` or `Direction.WEST`), please use an edge weight of `5f`.
- If you need to move down (`Direction.SOUTH`), please use an edge weight of `1f`.
- If you need to move up (`Direction.UP`), please use an edge weight of `10f`.

What we are describing is some sort of gravity, where it is easy to go down and hard to go up. When considering moving along a diagonal, please use take the two cardinal directions used in the diagonal, add the squares of their edge weights, and finally take the square root for the final edge cost. For instance, $cost(\texttt{Direction.NORTHWEST}) = \sqrt{cost(\texttt{Direction.NORTH})^2 + cost(\texttt{Direction.WEST})^2}$

When you want to test your `DijkstraMazeAgent`, please open up the maze file that you want to run (either `data/labs/stealth/SmallMaze.xml` or `data/labs/stealth/BigMaze.xml`). When you open this file 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="0">
2   <AgentClass>
3     <ClassName>src.labs.stealth.agents.BFSMazeAgent</ClassName>
4     <Argument>0</Argument>
5   </AgentClass>
6 </Player>
```

The line in my example

```
<ClassName>src.labs.stealth.agents.BFSMazeAgent</ClassName>
```

is **wrong**: we want player 0 to run `src.labs.stealth.agents.DijkstraMazeAgent` instead of `src.labs.stealth.agents.BFSMazeAgent`.

Task 7: Submitting your lab

Please submit the two java files: `BFSMazeAgent.java` and `DFSMazeAgent.java` on gradescope (no need to zip up a directory or anything, just drag and drop the files). If you choose to do the extra credit, please also submit `DijkstraMazeAgent.java`.