Lab 1: Your First Agent

Task 4. 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/lab1/lib/scripted.jar` to `cs440/lib/scripted.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/lab1/data/labs/scripted` to `cs440/data/labs/scripted`.
  This directory contains a game configuration and map files.
- Copy `Downloads/lab1/src` to `cs440/src`.
  This directory contains our source code `.java` files.
- Copy `Downloads/lab1/scripted.srcs` to `cs440/scripted.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.
- There is no documentation for this lab (all I am providing for you is the enemy agent this time).

Task 5. 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/*:. @scripted.srcs
java -cp lib/*:. edu.cwru.sepia.Main2 data/labs/scripted/game.xml

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

Task 6: Agents in Sepia

It is essential that you become familiar with the Sepia API (generally found here. Essentially, every agent in Sepia must extend the `Agent` type. `Agent`s implement a (surprisingly) simple state machine. While the `Agent` type is an abstract type (and therefore cannot be instantiated), your child type (or derived type) will be a concrete class. The state machine that every agent obeys is as follows:

1. A concrete agent type (specified by the game config .xml file) is instantiated using that types constructor.
2. All agents in the game are collected and added to internal Sepia data structures.
3. Sepia then creates the game itself (initializes a state from the map .xml file)
4. A special method that all agents must override (declared abstract in the `Agent` type) is called: `initialStep(StateView state, HistoryView history)`. This method is often used as a secondary constructor for an agent. If an agent needs to keep track of the units it controls, units the enemy(ies) control, etc., then that information is not available at the time of the constructor call. Instead, this method provides a place for fields that rely on state information to be set. This method returns a `Map<Integer, Action>`, which maps each unit (which has a unique id) to the action that unit should perform the very first turn of the game.
5. For every remaining turn in the game, a special method that all agents must override (declared abstract in the `Agent` type) is called: `middleStep(StateView state, HistoryView history)`. This method primarily contains "action logic" (i.e. what should each unit do in this turn), and also produces a `Map<Integer, Action>`.
6. Once the game has ended, a special method is called (also declared abstract in `Agent`): `terminalStep(StateView state, HistoryView history)`. This method is typically used to analyze the outcome of the game. Who won, what happened, how well did my agent do, etc. are all questions that `terminalStep` is designed to answer. Since the game is over, this method returns nothing (`void` in java).

Task 7: Scripted Agents (50 points)

In this task, I want you to complete the `src/labs/scripted/agents/ScriptedAgent.java` file that I have provided for you. I have partially implemented `ScriptedAgent::initialStep` for you to give you an idea of how to discover units, enemies, etc. I first want you to complete this method so that the only gold deposit on the map is discovered (i.e. you set the field for its id).

I then want you to implement `middleStep` so that your agent moves to a square adjacent to the enemy unit. You will need to issue `primitive move` actions (please see here for an explanation of actions in Sepia). You are welcome to follow a predetermined path to get there (i.e. you don't need to use BFS/DFS/Dijkstra/A* to find the *optimal* path) as the starting coordinates of your unit and the enemy unit will always be the same for this game. To be clear, the purpose of this assignment is not to do anything fancy, just get used to using the sepia api. So, you are encouraged to hard-code a sequence of moves (called a script) to complete this task. This script can be as simple as "go up three times then left two times" etc.

I then want you to modify your `middleStep` so that when you are adjacent to the enemy, issue `primitive attack` actions to have your unit attack the enemy unit. You will want to repeat this until the enemy unit dies (at which point the game will end). Side note, when units die, one easy way to check is to try to get the `UnitView` for a unit from the state. If the unitview is null, then the unit does not exist on the map.

Task 8: Extra Credit (25 points)

Before moving to the enemy unit and killing it, I want you to first move to a square adjacent to the gold deposit on the map. Once your agent is adjacent to the gold, I want you to issue `primitive gather` actions to gather all of the gold from the deposit (your `middleStep` should issue this gather action until the gold no longer exists on the map). Once the gold is gone, you are not free to move to the enemy unit and kill it like normal.

Task 9: Submitting your lab

Please submit your `OpenLoopAgent.java` file on gradescope (just drag and drop in the file).