Programming Assignment 1: Battleship

Due: Friday 03/29/2024 @ 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. We will not be using Sepia for this assignment: I have developed a game engine for us to use. In this assignment we will be writing agents to play battleship.

Battleship is a two player game where each player is in control of a fleet of ships. Each player is in control over a discrete board (i.e. a board with discrete coordinates), and both boards are the same size. Players place their ships one at a time on their board in secret: ships are allowed to touch but cannot overlap (i.e. they can be next to each other but not intersect). Once a player has finished placing their ships on their board, the ships cannot move afterwards. Finally, players take turns guessing coordinates on their opponents board. If the guessed square is occupied by a ship, that ship is "hit". If all of the squares that a ship occupies are hit, that ship is sunk. The player who sinks their opponents entire fleet wins.

After a square is guessed, the opponent provides feedback as to the outcome of the guess. There are three outcomes:

- `MISS`. If the guessed square does not contain a ship, the outcome of the guess is a `MISS`.
- `HIT`. If the guessed square does contain a ship but the ship is not sunk, the outcome of the guess is a `HIT`.
- `SUNK`. If the guessed square does contain a ship and that ship is sunk, the outcome of the guess is a `SUNK`. The player who guessed also receives the type of the ship that was sunk.

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/battleship.jar` to `cs440/lib/battleship.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/pa1/lib/argparse4j-0.9.0.jar` to `cs440/lib/argparse4j-0.9.0.jar`.
  This is a jarfile that `battleship.jar` depends on. It provides similar functionality to Python's `argparse` module. The documentation for `argparse4j` can be found here.
- Copy `Downloads/pa1/src` to `cs440/src`.
  This directory contains our source code `.java` files.
- Copy `Downloads/pa1/battleship.srcs` to `cs440/battleship.srcs`.
  This file contains the paths to the `.java` files we are working with in this assignment. Just like in the past, 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/pa1/doc/pas` to `cs440/doc/pas`. This is the documentation generated from `battleship.jar` and will be extremely useful in this assignment. After copying, if you double-click on `cs440/doc/pas/battleship/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 following code. A window should appear:

```
# Mac, Linux. Run from the cs440 directory.
javac -cp "./lib/*:." @battleship.srcs
java -cp "./lib/*:." edu.bu.battleship.Main

# Windows. Run from the cs440 directory.
javac -cp "./lib/*;." @battleship.srcs
java -cp "./lib/*;." edu.bu.battleship.Main
```

NOTE: The commands above will not run your code. There are several command line options available to you. I will describe them briefly here, but if you forget you can always add a `-h` or `--help` argument to the command line and see the help message!

- The first command line argument you can give is `--p1Agent`. This argument specifies the fully-qualified path to the Java class of the agent you want to run. For example, if you want player1 to be your agent, you would need to provide the argument like follows:

  `--p1Agent src.pas.battleship.agents.ProbabilisticAgent`

  If left unspecified, this defaults to `edu.bu.battleship.agents.RandomAgent` (i.e. an agent which randomly guesses the coordinate to attack).

- The next command line argument you can give is `-d` or `--difficulty`. This specified the difficulty of the game. Harder difficulties have bigger maps and larger fleet sizes. There are three choices with the following map and fleet sizes:

`EASY`: This map is a 10x10 with 1 aircraft carrier (5 squares), 1 battleship (4 squares), 1 destroyer (3 squares), 1 submarine (3 squares), and 1 patrol boat (2 squares).

`MEDIUM`: This map is a 20x20 with 1 aircraft carrier (5 squares), 1 battleship (4 squares), 2 destroyers (3 squares each), 2 submarines (3 squares each), and 3 patrol boats (2 squares each).

`HARD`: This map is a 30x30 with 2 aircraft carriers (5 squares each), 3 battleships (4 squares each), 4 destroyers (3 squares each), 4 submarines (3 squares each), and 5 patrol boats (2 squares each).

  This argument defaults to `EASY`, but if you wanted to play on `MEDIUM` difficulty, you would need to provide the argument like follows:

  `-d MEDIUM`

  You could also provide it like this too:

  `--difficulty MEDIUM`

- The next command line argument you can give is `-s` or `--silent`. If specified, the game will execute but no rendering window will appear. This can be used to play the game much faster, but you will not be able to view it. This argument does not take a value, simply adding `-s` or `--silent` to the command line when you execute battleship is enough.

- The final command line argument you can give is `-t` or `--thinkingTimeInMS`. This argument specifies how long each agent will have to think for the entire game in milliseconds. The amount of time your `makeMove` method takes to run (i.e. the method your agent must implement to play the game) will be subtracted from this number every time it is called (i.e. once per turn). If a player's time budget ever drops to zero, they automatically lose the game. The value defaults

to 480000 (8min worth of time), but if you want to specify the amount of time yourself, you will need to provide the argument like follows (in this example we will allocate both agents 10000ms of time each. This is roughly equivalent to 0.167 minutes):

`-t 10000`

You could also provide it like this too:

`--thinkingTimeInMS 10000`

3. Information on the provided files

- Directory `src/pa1/agents/` contains one .java file called `ProbabilisticAgent.java`. This file is the file I want you to implement. In particular, the method you need to implement for your agent to interact correctly with `battleship.jar` is the method `makeMove`. This method returns a `Coordinate` (i.e. the coordinate you want to attack) and takes a `GameView` object as input. `GameView` objects contain everything you want to know regarding the state of the game from your perspective: it contains your fleet information, what you know about the enemy fleet, a grid of outcomes (one per coordinate), some hyperparameters of the game (such as board dimensions, difficulty, etc.), and how much time you have remaining. I highly suggest you take a look at the `GameView` documentation to see what you have at your disposal.

This game is implemented using alternating turns, and I had to engineer quite a few things for this to work. If you are curious about how I actually solved these problems, feel free to ask! I spent many hours engineering around problems like this.

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

Please implement `ProbabilisticAgent.makeMove`. Your agent should think probabilistically in that we want to compute which coordinates (also called cells) contain ships given our evidence on the board. Formally:

Let $Y = \bigcup_{i=1}^{k} Y_i$ be the event of the grid configuration, where Y_i is the event of a particular ship configuration.

Let $X_i = \{\texttt{there is a ship in cell } i | Y\}$ be the event that there is a ship in cell i given the configuration of the board. We want to find these probabilities. Of course, the probability of this event is really a superposition of other probabilities. Using the law of total probabilities, we find that:

$$X_i = \{\texttt{aircraft carrier in cell } i|Y\} \cup \{\texttt{battleship in cell } i|Y\} \cup \cdots \cup \{\texttt{patrol boat in cell } i|Y\}$$

Your code should calculate these probabilities and then calculate the superposition of them (hint: they're dijoint events!). Your agent should guess the coordinate with the highest probability!

Inside the jarfile there are a few agents that are provided for you to play against. The one you probably will play the most against is the `RandomAgent` as you debug. However, there are others called `EasyAgent.java`, `MediumAgent`, and `HardAgent.java`. I will be calculating your assignment grade based on how well you do against the easy and medium agents in `EASY` and `MEDIUM` difficulties.

Task 2: Extra Credit (50 points)

If you agent can beat the `HardAgent` on `HARD` difficulty, I will award you full extra credit.

Task 3: Tournament Eligibility

In order for your submission to be eligible in the tournament, your submission must satisfy all of the following requirements:

- Your submission must be on time.
- You do not get an extension for this assignment.
- Your agent compiles on the autograder.
- Your agent can beat the `MediumAgent.java` more than 50% of the time (out of 10 trials) on `MEDIUM` difficulty. Your agent must also be able to play a game (win or lose) on `HARD` difficulty against the `RandomAgent`