Homework 09 – Battleship

Authors: Daniel
Topics: JavaFX

Problem Description

Ahoy there, captain! The enemy navy has trespassed your waters! You will command your armada in a strategic Battleship showdown, sinking your opponent's fleet to claim naval supremacy. In this assignment you will implement the popular board game *Battleship* using JavaFX.

Solution Description

You will need to complete and turn in one class: Battleship.java. This class will be the base for your GUI and you will need to meet the requirements listed below.

Provided Classes

We have provided you with various classes to help you implement this game. These classes give you the backend for your Battleship game, so you don't need to implement much of the game logic. Instead, use these provided classes to focus on making a cool user interface!

An overview of the provided classes and their methods:

- `BattleshipBackend.java`

  This class defines the backend logic of a Battleship game. For this assignment, you will implement a 1-player battleship game where you (the player) play against a computer (the enemy) that targets cells on the board at random.

  When creating your JavaFX application, create an instance of a `BattleshipBackend`. With this instance, you can simply call the methods (detailed below) to implement your game without worrying about writing the game logic yourself.

  Both sides (the player and the enemy) arrange their ships on a 10-by-10 grid and take turns guessing whether a specific cell on the opposite side's grid contains a section of a ship. If the guessed cell contains a section of a ship, the cell should be visibly marked as a "hit". If the guessed cell does not contain a section of a ship, the cell should be visibly marked as a "miss". The first side to "hit" all sections of the enemy ships wins.

  The rows on the 10-by-10 grid are 1-indexed and the columns are alphabetically indexed starting with 'A' (i.e., the top-left corner is row 1, column A, and the bottom right corner is row 10, column J).

  Layouts of the player and enemy ships are specified in `player.txt` and `enemy.txt` respectively. The `BattleshipBackend` class parses these text files for you! Feel free to modify the ship layouts – just make sure the files are 10-by-10 grids of either **~** (tilde) or **x** characters. An **x** represents a section of a ship and an **~** represents water (no ship). When running your application, make sure these two text files are in the same directory as `BattleshipBackend.java`, and **be sure to not modify the file names**.

This class provides you with all the functionality necessary to implement the gameplay loop.

- `attackEnemy(int rowIdx, char colChar)`
  - This method performs a guess on the enemy's grid at row `rowIdx` and column `colChar`.
  - This method returns a `GuessOutcome` object.
  - Remember that the grid's rows are 1-indexed, so `rowIdx` can range from 1 to 10. The grid's columns take character values from 'A' to 'J'.
- `attackPlayer()`
  - This method performs a guess on the player's grid at a random row and column.
  - This method returns a `GuessOutcome` object.
- `playerHasShip(int rowIdx, char colChar)`
  - This method returns true if the player's grid contains a section of a ship at row `rowIdx` and column `colChar`, and false otherwise.
- `hasPlayerWon()`
  - This method returns true if the player has won the game of Battleship (all the enemy's ships have been hit) and false otherwise.
- `hasEnemyWon()`
  - This method returns true if the enemy has won the game of Battleship (all the player's ships have been hit) and false otherwise.

- `Ocean.java`
  This class represents one side's grid. **You should not call any methods from this class or create instances of this class**, although two constants, `NUM_ROWS` and `NUM_COLS`, are publicly visible for use in implementing your GUI.
- `CellState.java`
  This is an enum which represents the possible states of a cell in the grid: `ALREADY_GUESSED` (a guess has already been made at this cell), `UNKNOWN` (this cell has not been guessed yet), `HIT` (there is a section of a ship at this cell), or `MISS` (there is water at this cell).
- `GuessOutcome.java`
  This class wraps the outcome of a guess (a `CellState` object), along with the coordinates of the cell that was guessed.

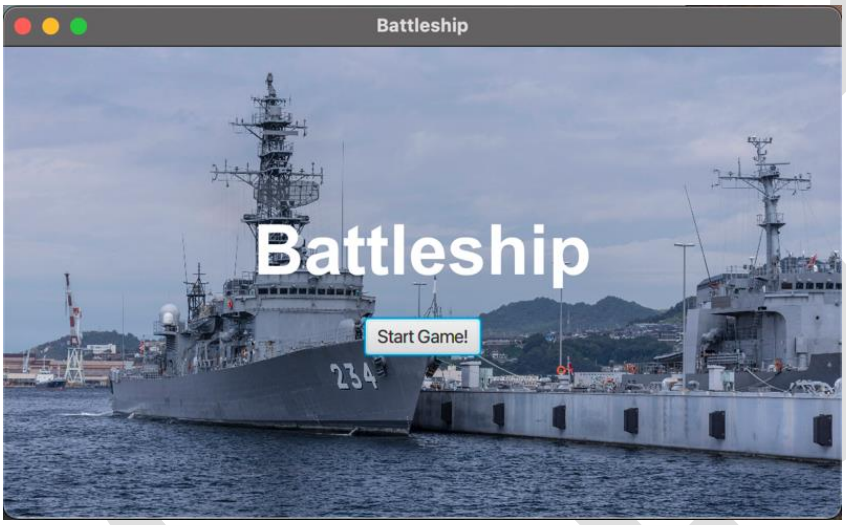
Make sure to thoroughly read through the Javadoc comments for each class so that you fully understand the provided codebase. **You are not allowed to modify any of the provided files.**

Battleship.java

This JavaFX class will be the base of your GUI. It will contain any variables or methods needed for your application to function properly.

Requirements:
Your JavaFX application must have three screens: the start screen, the game screen, and the end screen. Their requirements are as follows.

- **Start Screen**
  - The title of the window should be "Battleship". The title is displayed in the title bar at the top of a window (see example).
  - The background image should be an image of your choosing that relates to Battleship. **The image must be titled** `battleshipImage.jpg`, and you should submit this with your code.
    - If you do not want to find your own, we have provided one for you.
    - **Use a relative path for the image, not an absolute path.**
  - There should be a non-editable display of text that says "Battleship".
  - There should be a button that says "Start Game!" which takes you to the game screen.
  - **Example**

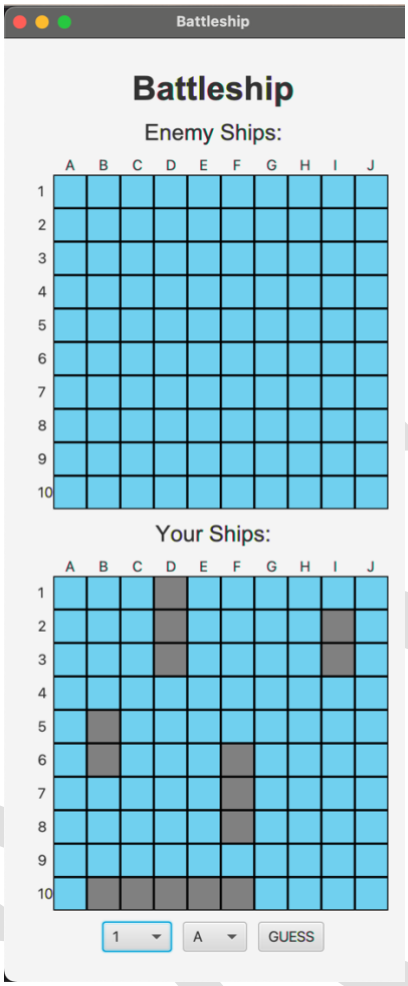


- **Game Screen**
  - The title of the window should be "Battleship". The title is displayed in the title bar at the top of a window (see example).
  - There should be a non-editable display of text that says "Battleship".
  - There are three main components:
    - An area for the enemy's grid.
    - An area for the player's grid, which must visibly display which cells contain ships.
    - An area for user input.
  - **Enemy Grid**
    - This is a 10-by-10 grid of cells, where each cell starts off as a particular color of your choosing to denote that the cell has not been guessed yet.
    - The rows should be labeled from 1 to 10, and the columns should be labeled from A to J.
    - This grid should be clearly labeled as "Enemy Ships".
      - ***Hint:*** You could implement the enemy and player grids using a grid of Rectangles.
      - ***Hint:*** Consider maintaining a 2D array of Rectangle objects to easily identify which cells need to change color after a guess.

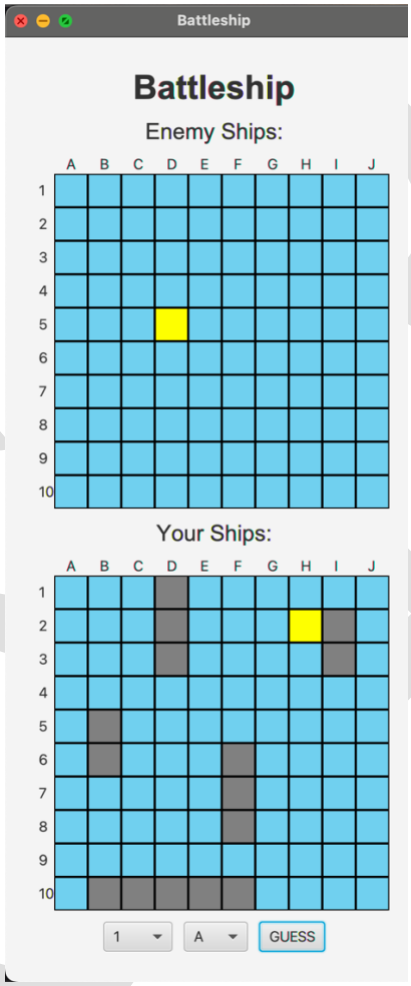
- **Player Grid**
  - This is a 10-by-10 grid of cells, where each cell starts off as a particular color of your choosing to denote that the cell has not been guessed yet. This color should be the same as the color of enemy cells that have not been guessed. Use another color to denote cells that contain a section of one of your ships.
    - ***Hint:*** `BattleshipBackend`'s `playerHasShip` method may come in handy here!
  - The rows should be labeled 1-10, and the columns should be labeled A-J.
  - This grid should be clearly labeled as "Your Ships".
- **User Input**
  - This area allows the player to select the cell in the enemy grid they would like to guess. A guess is a row number (1-10) and column letter (A-J) corresponding to the cell to guess.
  - This area should have:
    - A dropdown menu with the numbers 1-10 which corresponds to the row to guess.
      - This menu should have a default value of 1.
    - A dropdown menu with the letters A-J which corresponds to the column to guess.
      - This menu should have a default value of 'A'.
    - A button which says "GUESS" that places the guess.
  - Upon pressing the button, both dropdown menus should be set to default values (1 and 'A').
  - When the player presses the button, the appropriate cell in the enemy's grid should be set to a particular new color which denotes either a HIT or a MISS. These two colors must be distinct from each other (e.g., red for hit, yellow for miss) as well as any other cell colors used.
    - You should call `BattleshipBackend`'s `attackEnemy` method and update the enemy's grid according to the outcome of the guess.
    - ***Hint:*** Refer to the documentation for the return type of `attackEnemy`, `GuessOutcome`, for details on how you can get the result of the player's guess!
    - **Note:** You should not be able to guess a cell that has already been guessed. If the player attempts to guess a cell that has already been guessed, nothing should happen. The enemy should not guess either.
  - If, after placing the guess, the player wins the game of Battleship, the screen should switch to the end screen, ending the game.
    - ***Hint:*** `BattleshipBackend`'s `hasPlayerWon` method may be helpful!
  - If the player does not win the game after their guess, the enemy places their guess. You should call `BattleshipBackend`'s `attackPlayer` method and update the player's grid according to the outcome of the guess.
    - Update the cell the enemy guessed at in the player's grid with the corresponding color for a hit or a miss.
    - ***Hint:*** Refer to the documentation for return type of `attackPlayer`, `GuessOutcome`, for details on how you can get the result of the enemy's guess!

- If, after placing the guess, the enemy wins the game of Battleship, the screen should switch to the end screen, ending the game.
  - ***Hint:*** `BattleshipBackend`'s `hasEnemyWon` method may be helpful!
- Otherwise, the player should guess again, and this repeats until there is a winner!

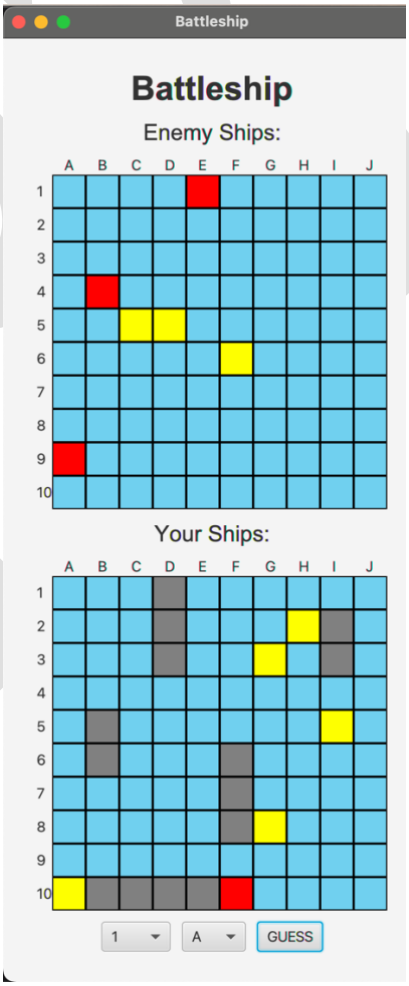
- ***Hint:*** You should create an instance of `BattleshipBackend` in your application. With this instance, you can simply call the methods provided to create the gameplay loop instead of writing all the game logic from scratch.
- **Examples**



What your application may look like initially, without performing any guesses.



The player guessed row 5 and column D, which was a miss. The enemy guessed row 2, column H immediately after (also a miss).



Several guesses later, the player guessed row 9, column A, which was a hit! The enemy guessed row 10, column F, which was also a hit.

- **End Screen**
  - The title of the window should be "Battleship".
  - This screen should display "YOU WIN!" if the player wins, or "YOU LOSE!" if the enemy wins.
  - **Example**

- When implementing the **base functionality** of your UI, you **MUST** use the following:
  - At least one anonymous inner class
  - At least one lambda expression

Tips and Tricks

- **IMPORTANT: Do not use absolute path for your images/files. Use relative path as the autograder will not find your files if you use the absolute path.**
- The JavaFX API can provide you with a lot of helpful information! Use it to your advantage.
- Make sure JavaFX is properly installed and runnable on your computer before starting this assignment. Do not wait until the last minute to configure your JavaFX!
- Work incrementally. Get a basic UI up and running. Add the buttons, cells for display, etc., piece by piece. It may be helpful to plan out your layout on paper before implementing it.

Extra Credit

- We will award **up to 25 points** of extra credit based on the following criteria:
  - Adding a non-trivial addition to the application's graphics that clearly extends its look (this cannot be something like just changing the background image). **[5 points]**
    - E.g., adding images to represent water/ships/hits/misses as the cells of the grid instead of colors, sound effects on a ship hit/miss
  - Adding a non-trivial addition to the program's controls or functionality that clearly extends its capabilities. **[5-15 points]**
    - E.g., adding the ability to toggle between dark mode and light mode
  - An impressive, simply amazing solution that blows the TAs away. (We've seen some students do very cool things before. Just run away with the assignment and make it awesome!) **[5 points]**
  - You can choose to do one, two, all of the above, or none.
- **Note:** Credit for your 1 lambda expression and 1 anonymous inner class comes from the code for the base functionality of your application. You are free to use any event handling for your extra credit.
- **If you choose to do extra credit,** include an **`extra_credit.txt`** file to your submission that explains exactly what additional features you have added.
  - If your code requires a different command to compile and run it (i.e., using different JavaFX modules), then put the command in `extra_credit.txt`.
- **Extra credit features should not affect any of the base functionality!**

Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 50 points.** This means there is a maximum point deduction of 50. If you don't have Checkstyle yet, download it from Canvas → Modules → Checkstyle Resources → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar YourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the number of points we would take off (limited by the Checkstyle cap). In future assignments we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Battleship.java`
- `battleshipImage.jpg`
- `extra_credit.txt` (optional, see "Extra Credit" section for details)
- **Any other files needed for your code to compile and run.**

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the assignment. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g., non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

- Any imports from any package that starts with `java` or `javafx` that do not belong to the AWT or Swing APIs.
  - ***IMPORTANT: You are NOT allowed to use FXML or any of the associated classes or packages.***
- Note that JavaFX is different than AWT or Swing. If you are trying to import something from `javax.swing` or `java.awt`, you are probably importing the wrong class.
- The only modules you should use are javafx.controls and javafx.media.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

The only code you may share are test cases written in a Driver class. If you choose to share your Driver class, they should be posted to the assignment discussion thread on the course discussion forum. We encourage you to write test cases and share them with your classmates, but we will not verify their correctness (i.e., use them at your own risk).

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items.
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope.
- Submit every file each time you resubmit.
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points.
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.