PA 03 - BattleSalvo - Part 1

Title	Battleship Part 1
Issues Due Date	@May 30, 2023
EC Due Date	@June 1, 2023
Due Date	@June 3, 2023
GitHub Classroom Link	[redacted]

‼️ GitHub Classroom Link — [redacted]

📖 Background

While growing up, you may have played the game of **Battleship**! Battleship is a classic two-player board game which simulates naval warfare. The game revolves around strategy, deduction, and a bit of luck. Each player has their own private grid with various ships placed on it. The objective is to guess the location of the opponent's ships and sink them before they sink yours.

The game board consists of two grids, typically marked by letters for columns and numbers for rows. The grids are often represented by 10x10 squares, but other variations exist. Each player's grid is divided into rows and columns, allowing players to strategically place their ships on their board. The types and sizes of ships can vary, but common examples include the carrier, battleship, destroyer, submarine, and patrol boat.

Players take turns calling out coordinates on the opponent's grid, attempting to hit their ships. The opponent responds with "hit" or "miss" based on the accuracy of the guess. If a hit is recorded, the attacking player marks the location on their own tracking grid to keep track of successful hits.

As the game progresses, players use deduction and logic to narrow down the possible locations of the opponent's ships based on the hits and misses they've observed. Once all the squares of a ship have been hit, it is considered sunk. The first player to sink all of the opponent's ships wins the game.

🕹️ If you would like to familiarize yourself with the game, there is a simple, free to play version available at battleshiponline.org.

🔀 BattleSalvo, an OOD Twist!

For PA03, you will be implementing a version of Battleship, which we have nicknamed **BattleSalvo**! There are a few rule changes, which are outlined below.

📚 If you need help understanding standard Battleship, this will be a helpful resource: wikihow.com/Play-Battleship

📚 The slides from class are also available here: https://docs.google.com/presentation/d/1aruXsFgeZwS4_9jHEOi6OtNzdjeTBThL4-80FamEiWQ/view

Board Size

BattleSalvo grids, instead of being 10x10, can have height and width dimensions of any value between 6 and 15 (inclusive)! Height and width dimensions do not need to match. For example, 6x10 and 9x6 would both be valid board dimensions for a game. The size of the board for each player, however, must be identical.

Ship/Boat Sizes

In Battleship, each boat is positioned horizontally in one row or vertically in one column, and covers a specific number of cells on the board.

In traditional Battleship, the boat sizes are:

- Carrier: Size 5
- Battleship: Size 4

BattleSalvo is a bit different:

- Carrier: Size 6
- Battleship: Size 5

- Destroyer: Size 4
- Submarine: Size 3
- Patrol Boat: Size 2

- Destroyer: Size 4
- Submarine: Size 3

Fleet Size

In Battleship, there is one of each boat type. But, in BattleSalvo, there may be several of each boat type.

The total fleet size may not exceed the smaller board dimension, but must have at least one of each boat type. Therefore, for a board of size 8x11, there should be no more than 8 total boats. Fleet size and boat types will be identical between players.

Number of Shots

In traditional Battleship, a player launches a missile once per turn. In BattleSalvo, for each turn, each player launches one missile per non-sunk boat remaining in their fleet. For example, if you currently have 3 remaining ships in your fleet, you would launch 3 missiles. At the same point in time, if your opponent had 5 ships remaining, they would be able to launch 5 missiles.

Shooting Order

In BattleSalvo, both players select their shots (target locations), and the shots are exchanged simultaneously. Information about hits is then exchanged, surviving ships are updated, and the process is repeated until one (or both) players have no more surviving ships. Importantly, this means some games will end in ties!

More specifically, the steps for the shooting stage of Salvo are laid out below:

1. Both Players shoot their “Salvo’s”
2. Both Players receive the “Incoming” salvo that their opponent fired
3. Both Players update their ships accordingly, and communicate which of the incoming shots hit
4. Repeat

🚢 Your Program

The Big Picture

PA03

For this assignment, we are asking you to implement a single player version of BattleSalvo against an AI of your own creation. More specifically, a user should be able to run your program, and play BattleSalvo on the command line, against an “AI” opponent.

Where we’re going…

Over the course of PA04 and PA05, we are going to ask you to extend your program so that it allows users to play the game through a Graphical User Interface (GUI). We will also ask you to update your program to allow your “AI” to play another AI remotely over a network connection. During that assignment, you will even be able to test your AI algorithm against that of your peers…. Stay tuned 😎

Program Summary

For this assignment, we are asking you to implement a single player version of the BattleSalvo game described above. A human user should be able to run your program and play a full game against an AI player that you have implemented in the terminal.

We have largely framed our expectations for this assignment through the user experience that we expect. The way you design and implement your program is up to you, but please keep SOLID and MVC principles in mind.

User Experience in Gameplay

Welcome Message and Board Parameters

- When your program begins, it should print a welcome message indicating that the game has started. It should then prompt the user to input a height (first argument) and width (second argument) of the game board. An example of this program instantiation is shown below:

```
Hello! Welcome to the OOD BattleSalvo Game!
Please enter a valid height and width below:
---------------------------------------------------
10 8
```

- Invalid inputs should be handled gracefully. If either the height or width are invalid, the user should be prompted to enter BOTH again. This is demonstrated below:

```
Hello! Welcome to the OOD BattleSalvo Game!
Please enter a valid height and width below:
-----------------------------------------------------
3 17
-----------------------------------------------------
Uh Oh! You've entered invalid dimensions. Please remember that the height and width
of the game must be in the range [6, 15], inclusive. Try again!
-----------------------------------------------------
8 8
```

Fleet Selection

After Board Parameters are entered, the user should be prompted to input the number of ships they'd like to play with.

- The user will enter an integer for the number of each type of boat, in the order [Carrier, Battleship, Destroyer, Submarine]. Please display the maximum size of the fleet that the user may enter. Here's an example:

```
Please enter your fleet in the order [Carrier, Battleship, Destroyer, Submarine].
Remember, your fleet may not exceed size 8.
--------------------------------------------------------------------------------
1 2 2 1
```

- Once again, invalid inputs should be handled gracefully. Example:

```
Please enter your fleet in the order [Carrier, Battleship, Destroyer, Submarine].
Remember, your fleet may not exceed size 8.
--------------------------------------------------------------------------------
1 2 2 0
--------------------------------------------------------------------------------
Uh Oh! You've entered invalid fleet sizes.
Please enter your fleet in the order [Carrier, Battleship, Destroyer, Submarine].
Remember, your fleet may not exceed size 8.
--------------------------------------------------------------------------------
3 2 2 1
```

Ship Placement

After the user finishes selecting their fleet, your AI should **automatically** generate placements for each ship: both on the user's board and the AI's own board. This can happen in any randomized or algorithmic manner of your choosing. We encourage creativity with this!

Keep in mind that:

- ships may not overlap
- ships must fit fully on the board
- ships must occupy the same number of coordinates (cells) as specified for each ship's type
- individual placements should be different for the user and AI (but may be the same if by random chance alone)
- individual placements may not be hard-coded

Users may NOT be prompted for ship placements. This must happen automatically.

Once the AI has generated all ship placements, the user should be presented:

1. a view of the AI's board, showing only the information known to the user (nothing, yet!)
2. the user's own board, showing all ships in their respective coordinates

An example of an initial 8x8 board is below, but please be creative with this!

```
Opponent Board Data:
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0

Your Board:
    B B B B B 0 0 0
    0 0 C 0 0 D 0 0
    0 0 C 0 0 D 0 0
```

```
    0 0 C 0 0 D 0 0
    0 0 C 0 0 D 0 0
    0 0 C 0 0 0 0 0
    0 0 C 0 0 0 0 0
    0 S S S 0 0 0 0
```

ℹ️ In PA01, the Opponent is *your* AI.

💡 **Note**: in the above placement, we included letters to represent the type of ship in order to make it easier to understand which ship is where. This is **NOT** required in your implementation. Any character or String indicating that a ship is present in a coordinate would be valid.

Game Loop

Salvo Stage

After the initial board is displayed, the user enters the game itself!

The user should be prompted to enter the shots corresponding to their remaining ships. If there are any invalid shots, please prompt the user for the entire salvo once again (example below). A “Shot” consists of an X (first argument) and Y (second argument) coordinate.

Example:

```
Opponent Board Data:
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0

Your Board:
    S S S S S 0 0 0
    0 0 S 0 0 S 0 0
    0 0 S 0 0 S 0 0
    0 0 S 0 0 S 0 0
    0 0 S 0 0 S 0 0
    0 0 S 0 0 0 0 0
```

```
    0 0 S 0 0 0 0 0
    0 S S S 0 0 0 0

Please Enter 8 Shots:
---------------------------------------------------------------
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
```

After the user fires their shots, your program should update the terminal view with representations of the following:

- shots fired by the user which hit AI ships
- shots fired by the user which did not hit ships
- shots fired by the AI which hit user ships

In the case that a user enters any invalid coordinates, please prompt them to answer an entirely new Salvo.

Example:

```
Opponent Board Data:
    M M H H H M M M
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0

Your Board:
    S S S S S 0 0 0
    0 M S 0 0 S 0 0
    0 0 H 0 0 S 0 0
    0 0 S 0 0 S M 0
    0 0 S 0 0 S 0 0
    0 0 S 0 0 0 M 0
    0 0 H 0 0 0 0 0
    0 S S H M M 0 0
```

```
Please Enter 8 Shots:
------------------------------------------------------------
```

ℹ️ **As with earlier visualizations, exactly how you represent this data is up to you!** Please be creative, just be sure to differentiate between untouched cells, hit ships, and misses. NOTE: You do NOT need to show opponent misses on your own grid.

Repeat

This “Salvo Stage” will continue until either:

1. all of the AI’s ships have been sunk, or
2. all of user’s ships have been sunk

Game End

When the game ends, your program should display to the user whether they have won, lost, or tied. After displaying that, the program should exit — without requiring any additional input from the user.

🌷 Starter Code

```
/**
 * Represents a single player in a game of BattleSalvo.
 */
public interface Player {

    /**
     * Get the player's name.
     * NOTE: This may not be important to your implementation for PA03, but it will be later
     *
     * @return the player's name
     */
    String name();

    /**
     * Given the specifications for a BattleSalvo board, return a list of ships with their locations
     * on the board.
     *
     * @param height the height of the board, range: [6, 15] inclusive
     * @param width the width of the board, range: [6, 15] inclusive
     * @param specifications a map of ship type to the number of occurrences each ship should
     *        appear on the board
     * @return the placements of each ship on the board
```

```
     */
    List<Ship> setup(int height, int width, Map<ShipType, Integer> specifications);


    /**
     * Returns this player's shots on the opponent's board. The number of shots returned should
     * equal the number of ships on this player's board that have not sunk.
     *
     * @return the locations of shots on the opponent's board
     */
    List<Coord> takeShots();

    /**
     * Given the list of shots the opponent has fired on this player's board, report which
     * shots hit a ship on this player's board.
     *
     * @param opponentShotsOnBoard the opponent's shots on this player's board
     * @return a filtered list of the given shots that contain all locations of shots that hit a
     *         ship on this board
     */
    List<Coord> reportDamage(List<Coord> opponentShotsOnBoard);

    /**
     * Reports to this player what shots in their previous volley returned from takeShots()
     * successfully hit an opponent's ship.
     *
     * @param shotsThatHitOpponentShips the list of shots that successfully hit the opponent's ships
     */
    void successfulHits(List<Coord> shotsThatHitOpponentShips);

    /**
     * Notifies the player that the game is over.
     * Win, lose, and draw should all be supported
     *
     * @param result if the player has won, lost, or forced a draw
     * @param reason the reason for the game ending
     */
    void endGame(GameResult result, String reason);

}
```

‼️ Yes, we know that some of the types included in this `Player` interface neither exist in Java, nor are defined in the starter code! These are up to you to define as you wish.

We HIGHLY recommend that you do not update or change this interface.

🗓️ Deliverables + Deadlines

Design

- Due: Tuesday, May 30, 2023 at 10pm to Github project repo
- Backlog of Tasks entered as Issues (Similar stipulations as with A1 and A2)
- UML Class Diagram
    - On your UML Class Diagram, it will help to label each class with either Model, View, or Controller in your class and interface names. This will help you think about how to structure your project, and will help the TA's better interpret your code to give you better feedback!

Program

- Due: Friday, June 2, 2023 at 10pm to Github project repo.
- Your BattleSalvo program with all classes EXCEPT your `Driver` in one of three packages: `model`, `controller`, `view` and associated tests.
- An updated UML diagram showing the final (submitted) design of your program.
- Ensure your entry-point class is named `Driver`. This is the class which contains your `main()` method.
- As with PA01 & PA02, you should have at minimum 90% test coverage. We will be looking at both Missed Instructions and Missed Branches.

💬 FAQ

- How advanced does my AI player need to be?
    - Not very. Our only specification is that the AI player should not shoot at the same coordinate twice. Beyond that, any implementation is fine. This is a great opportunity to go above and beyond, though! There may even be some benefits for exceptional algorithms in the future.
- What if the number of ships I have exceeds the number of remaining Coordinates on the opponent's board, or vice versa for the AI?
    - You should allow the user to shoot the number of shots corresponding to the number of empty coordinates remaining.
- See additional FAQs on Piazza, 

✨ Helpful Hints

- Before starting to design your solution, remind yourself of the SOLID principles. These articles (DigitalOcean and FreeCodeCamp) an overview of them if you need a very brief reminder. Consider not only how this can help your implementation but also your testing.
- The assignment is designed such that you should make use of the MVC design pattern. Be sure to review your lecture notes and Lab 06 for guidance.
- If you need help creating a UML diagram, this is a reference sheet you may find useful. We are not expecting you to label packages, but we are expecting classes, interfaces, fields, and methods to be described, as well as their relationships.
- We may come back to this project later (or very soon), so consider a few ways we may expand the scope.
- Finally, don't forget your essential problem solving concepts. This JavaDoc (this reference page could be helpful) and following the design recipe to break down problems will help you greatly.