Programming Project - Documentation (Team 8)

Team Members

Heena Kashyap - 300305305
Sukhleen Kaur - 300310052
Sehajpreet Singh Kingra - 300304180
Kunal Ajaykumar Jeshang - 300328339

Repository Link

https://github.com/kjeshang/CSIS3275-BattleshipGame

Description

We created the battleship game using the Java programming language using the Eclipse IDE. Java Swing was utilized to create the user interface. The MVC structure and approach was implemented to the best of our ability. The project contains a single package with four classes. The UI class incorporates the swing elements for user interface, and instantiates the Model, View, and Control classes. In turn, the UI class is responsible for displaying elements of the game. It also validates the player input before the input is utilized by the other classes. The Model class sets up the battle area grid and all of the battle area's elements & locations (e.g., water, enemy ship, etc). The View class takes the elements from the Model class and finalizes the format of how it will be displayed to the player. The Control class takes & validates user input and updates the battle area accordingly. The View class takes battle area updates from the Control class and finalizes the format that would be displayed to the player.

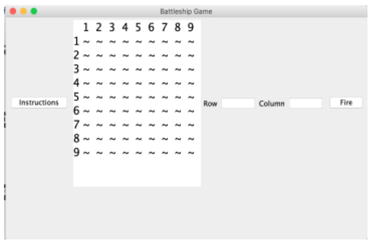
Added Features

- Random allocation of enemy ships upon initialization of battleship game.
- Enemy Ships are of now 2 types small ships and Big Ships.
- A single player ship is allocated in the battle area, but location is unknown to the player. If the player fires a missile and they destroy their own ship, it is an instant game over.

The BattleShip Game

Overview

Here is how the game will work:



```
  1 2 3 4 5 6 7 8 9
1 ~ ~ ~ ~ ~ ~ ~ ~ ~
2 ~ ~ ~ ~ ~ ~ ~ ~ ~
3 ~ ~ ~ ~ ~ ~ ~ ~ ~
4 ~ ~ ~ ~ ~ ~ ~ ~ ~
5 ~ ~ ~ ~ ~ ~ ~ ~ ~
6 ~ ~ ~ ~ ~ ~ ~ ~ ~
7 ~ ~ ~ ~ ~ ~ ~ ~ ~
8 ~ ~ ~ ~ ~ ~ ~ ~ ~
9 ~ ~ ~ ~ ~ ~ ~ ~ ~
```

1) Firstly, we have placed a 9X9(can be changed) grid. Which is an array that will act as the sea and will hold all our enemy and friendly ships. On the default the grid will be covered by '~' indicating water.
2) There will be two text input boxes for the player labelled as Row and column in which the user will type in the coordinates of the MISSILE. After entering the user hits Fire which then unlocks the object behind the ~.
3) There is also an Instructions button which provides the instructions to the user on how to play the game.

Backend

Step 1: The user executes the UI class which imports all the other java classes (Model, View, Control).

Step 2: A New UI object is created which further calls the Model class to build the Map.
Step 2.1: In the Model Class everything is done under the default parameterized constructor which calls in below methods and serves following purposes.

setDimension(dimension)	Builds the Sea grid with dimension X dimension size.
setWater(),setShip(),setBigShip(),setFriendlyShip()	Generates and stores the coordinates of water, ship, big ships, Friendly ships on the respective locations.
setBattleArea()	Uploads the updated grid information to the View

Step 3: An array is created which gets everything in the battleArea from Model class
String[][] battleArea = model.getBattleArea();

Step 4: A view is created using the defined Model and generated battle area
View(battleArea,model).

Step 5: An action listener is used at the fire button which performs the operations of the game step by step each time it is clicked.

Step 6: Once the user Clicks the fire button the following process is repeated every time.

Step 6.1: The user input for row and column is received and then verified and once it is verified. It is passed to the control class
Step 6.1.1: In the control class the logic of the game is processed. First it is checked what is at the user coordinates checkAndEvaluateTarget(int[] ..).
Step 6.1.2: Once we know what is at the coordinates the battleArea is updated and a respective message is returned to the function call.
Step 6.2: The process is repeated until either the undetectedShips reaches 0 or we hit the friendly ship.
Step 6.3: the process is looped on a Boolean variable gameOver which remains false until the above condition is met.

General explanation of the classes

Control Class: the class acts as the controller processor and logic executor in the game. The class is responsible for Validating the responses from the user and performing respective actions.

Model Class: The Model class is responsible for holding all the elements and the data associated in those which will be utilized by the control class to perform logic and View class to display the required information to the user.

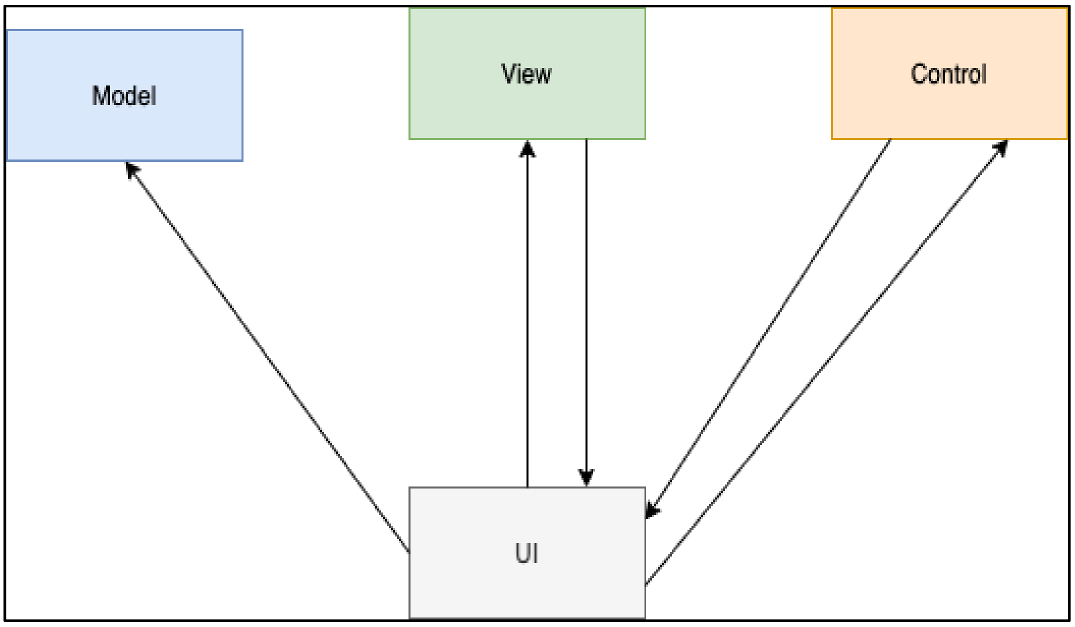
View Class: The View Class depicts the information and the format in which the user or the player see it. This class holds the elements like the battleground map, how the ships and the water look like before and after the coordinates have been hit.

UI Class: This Class glues everything in Control, Model, View and user inputs together. UI stands for User Interface acts as a central hub where all the other classes in the MVC structure are imported and their methods are respectively called, and operations are performed. This very class starts the game and ends the game too.

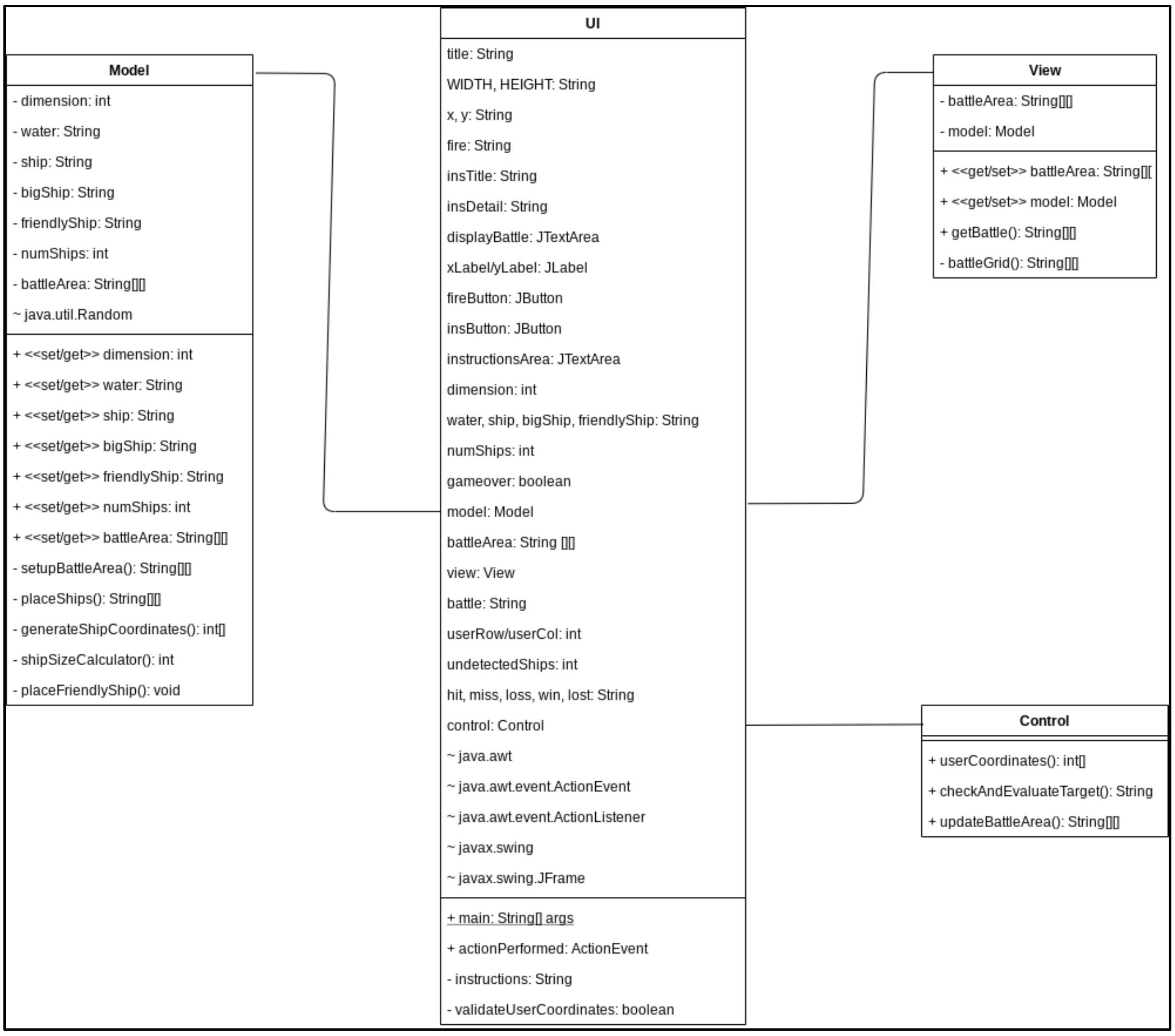
We have built our project according to MVC structure which makes it easier:

- To find errors and fix it.
- To change the look of the game
- To add new features
- To change the logic of the game.

Class Relationship Diagram



UML Diagram


Model
- dimension: int
- water: String
- ship: String
- bigShip: String
- friendlyShip: String
- numShips: int
- battleArea: String[][]
~ java.util.Random
+ <<set/get>> dimension: int
+ <<set/get>> water: String
+ <<set/get>> ship: String
+ <<set/get>> bigShip: String
+ <<set/get>> friendlyShip: String
+ <<set/get>> numShips: int
+ <<set/get>> battleArea: String[][]
- setupBattleArea(): String[][]
- placeShips(): String[][]
- generateShipCoordinates(): int[]
- shipSizeCalculator(): int
- placeFriendlyShip(): void
UI
title: String
WIDTH, HEIGHT: String
x, y: String
fire: String
insTitle: String
insDetail: String
displayBattle: JTextArea
xLabel/yLabel: JLabel
fireButton: JButton
insButton: JButton
instructionsArea: JTextArea
dimension: int
water, ship, bigShip, friendlyShip: String
numShips: int
gameover: boolean
model: Model
battleArea: String [][]
view: View
battle: String
userRow/userCol: int
undetectedShips: int
hit, miss, loss, win, lost: String
control: Control
~ java.awt
~ java.awt.event.ActionEvent
~ java.awt.event.ActionListener
~ javax.swing
~ javax.swing.JFrame
+ main: String[] args
+ actionPerformed: ActionEvent
- instructions: String
- validateUserCoordinates: boolean
View
- battleArea: String[][]
- model: Model
+ <<get/set>> battleArea: String[][
+ <<get/set>> model: Model
+ getBattle(): String[][]
- battleGrid(): String[][]
Control
+ userCoordinates(): int[]
+ checkAndEvaluateTarget(): String
+ updateBattleArea(): String[][]

Legend:

-	Private variable or method
~	Imported java framework package
+	Public variable or method
<<set/get>>	Indicates both getter and setter of a particular variable

Note: References provided in APA7 format

References

Freeman, E., &amp; Robson, E. (2014). Head first JavaScript programming: a brain-friendly guide. O'Reilly.

UML Class Diagram Tutorial. Lucidchart. https://www.lucidchart.com/pages/uml-class-diagram.

UML Class Diagram Tutorial. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/.

Zeitvergessen. (2021, January, 29). How to create Battleship with Java (from scratch!) Part (1 of 3). YouTube. https://www.youtube.com/watch?v=DNK-2j4k2bg&list=WL&index=14

Zeitvergessen. (2021, January, 29). How to create Battleship with Java (from scratch!) Part (2 of 3). YouTube. https://www.youtube.com/watch?v=VEPJI2eh9oY&list=WL&index=15

Zeitvergessen. (2021, January, 29). How to create Battleship with Java (from scratch!) Part (3 of 3). YouTube. https://www.youtube.com/watch?v=581KPBVasZQ