Name: CLAES Irene
Student number: r0627946

ASSIGMENT: BATTLESHIP

Introduction

In this assignment, the goal was to create a 2 players Battleship, where the 2 players play on 1 board. To make this I used Java Eclipse.

The battleship game I made consists of a board with squares. There will be 4 different ships on the field, of different length. The ships can be placed randomly or with a text file.

There are several options that can be chosen (see figure 1). If you want to start the game with randomly placed boats, first choose the number of rows and columns you want to use to play. This should be a minimum of 6 and a maximum of 20. Then you choose whether you want to play the game with an equal distribution of points or with a settlement. Then press "Start Game".

Otherwise, you can also choose to place the boats on the field yourself. Before you press "Choose Ship Placement", first choose the scoring method. Then you will see a field with an indicated example. This field contains 20 rows and 20 columns and coordinates of the boats. You can edit this field. When it is adjusted press "Start Game". When the boats are on top of each other, you will get an error message, if not, the game will start.

Then the game begins. Each player calls out one shot each turn in attempt to hit one ship, by clicking on a square. When a player hits a boat, that player's score is increased. The game is over when all boats are sunk. If the high score has been improved by the winning players, it will be saved. The high scores can be viewed in 2 places, namely in the start menu or during the game itself. You can review the rules when you use the "Rules" button.

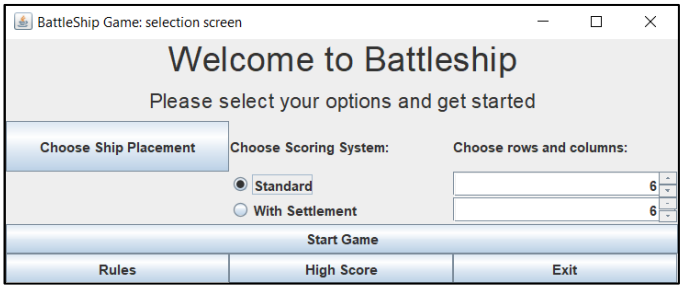

Figure 1: Start Menu

Part 1: Making a board in the console

I started creating the program by testing everything in the console first, before I made a graphical used interface.

1. <u>Randomly placing of 5 different ships on the board</u> (Class BattleshipBoard)

I first created an empty field on which boats are randomly placed. See result console figure 2.

```
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

Figure 2: Empty field

To randomize the boats, I first created a new Random. This can be used to request random numbers or booleans. Then I created 3 variables, r (int), c (int) and orientation (boolean). I request a random number or boolean for each boat separately. Next, I look at the boolean first. If this is true, the boats are placed vertically. I do this by always adding 1 to the randomly chosen c with a forloop. When it indicates the boolean as false, the boats are placed horizontally. The first boat placed is the Carrier. For the next boats, namely Battleship, Submarine and Destroyer, the program have to look at the positions of the already placed boats. I do this with a while loop. The principle is the same as by the carrier. First, a random number is chosen for r and c and a random boolean for orientation. Then it is checked in the while loop whether these new randoms have been approved. If not, new randoms are chosen. If the selected randoms are approved, the boat will be placed.

I also looked at the margin so that the boats could not be moved out of the field. I did this by putting a limit on the random integers that can be chosen by doing the number of the rows or columns minus the length of the boat.

There are also other methods in the BattleshipBoard class, such as isGameOver and isValidMove. The is GameOver method indicates when the game is over by looking at the numbers on the field. If these are all still equal to HAS_ "BOAT", then it is not game over yet, otherwise it is. The method isValidMove, I created to indicate when the field has already been pressed, but this method did not work.

The result of an example of the boats placed in the console is shown in Figure 3.

```
00000000
05555500
00000300
00000300
00000300
04444000
00000000
00000220
```

Figure 3: Randomly Placement

2. <u>Placing ships with a text file</u> (Class BattleshipBoardPlacement)

To place the boats with a text file, I used the Java Scanner. With the Scanner you can go through a file, line by line, but also character by character. First, I scan the first integer, which shows the number of rows and columns. It is always a square field. Then I created a while loop so that I can go through the file to find the coordinates for each boat. I made a String (coordinate) that returns the whole line of each boat. From each line I make a selection using an array, String (coordinate []), by using Regular Expressions to find only the number of each coordinate. I used only the first two coordinates of x and y, because with these 2 you know the orientation (horizontal or vertical) and the length of each boat. Then I made a for loop to place each boat. For example, for a carrier, with length 5, I start the for loop with "for (int i = 0; i <5; i ++)", so the loop starts counting at 0 and keep adding less than 5 (= 4). This gives me a 5-block boat. I did this for every boat. As in the BattleshipBoard class, I have to check for the next 3 boats if the chosen coordinate is not already occupied. I do this by first checking whether the chosen coordinates give a square that is already occupied. For example if there is already a carrier that box is 5 instead of 0. If this is the case, an error will appear on the screen after starting the game.

The BattleshipBoardPlacement class is very similar to the BattleshipBoard class. I could also have added this method in the BattleshipBoard class by creating a boolean (true = random, false = text file). I tried this, but I didn't work. I could not make the links between other classes, so I chose to write this in separate classes. This creates a lot of duplicate codes throughout the rest of the writing process.

3. <u>Making an attack mode</u> (Class Player)

The attack mode is in the Player class. This was originally BattleshipBoard, but was later moved to Player so I could add points per player. In this method I give each hit ship a different number that can then be linked to a different color in the GUI. I made 2 attack modes because I made a different class for the random placement of boats and the placement of boats via a text file.

4. <u>Player</u> (Class Player)

In the class Player I made the 2 players ("number"), which can be linked by this class to a number of points (point = + POINT_ "SHIP"). For this I made getter method, which can display the obtained points in another class. I also made a setter for the points, so that I could make an alternative scoring method. For this I chose to start Player 1 with 5 points, because this player has to choose first, he has a disadvantage because Player 2 can make his choice depend on what Player 1 has chosen.

Part 2: Making a GUI

To create the GUI I made serveral classes that will eventually all be linked in the class StartFrame that is run in the class MainGUI. I started with setting up the player board before creating the start menu. The following breakdown is in order of how I designed it.

1. <u>Class BattleshipGrid</u>

In BattleshipGrid I started with the paintCompontent method using "Graphics g". In this I choose to make the background white and I linked this method with the method drawRect. In the drawRect method I also make the background white and I start placing the boxes in the point 0.0. Then I make a for loop with several if statements, where the first for loop ensures that I get a board with the number of rows and columns of gray boxes. The subsequent if statements change the color depending on the number I made in the console to a hit from a ship or a miss from the board. Each color is linked to a type of boat with another number, for example the carrier will be red.

Another option was to make the board from buttons. In hindsight this might have been easier because in the next method (attackBoard), I had to link the mouse clicks to a board full of pixels. These then had to be formed into an integer that must be linked to the selected rows and columns. I was able to do this with "Math.floorDiv (int, int)". This function causes the mouse click (pixel) to be divided by the height or the width of the board and then rounded down to an integer.

Then I started linking the following elements, such as the different players. For this I made an array of 2 players (0 and 1), which is linked to the player number (numPlayers). The for loop in the constructor makes sure that 2 players are created.

The subsequent if statement ensures that there are 2 different scoring methods. For this I use the setter of the points, which allows me to start Player 1 with a lead of 5 points. Later this method will be linked to the JRadioButtons in the StartFrame.

One of the most important parts of this constructor is the mouseClicked method. This allows cells to change when indicated with the mouse by the attackBoard method, but also the players are changed by the mouse click, from 1 to 2. Furthermore, the score of each player is continuously updated by a mouse click and also the method isGameOver from BattleshipBoard is linked. After this a showMessageDialog is always displayed, which says which player has won. After pressing "ok" the game is closed.

When the game is over, the high score is saved in a text file. Here reference is made to the last method of the BattleshipGrid class. In the saveHighScore method, the text file is read and updated. First the text file is read by BufferedReader and a string that is going to "read" every line of the file. In this an if statement is made that if the score of player 1> than the score of player 2, the score of player 1 is saved as high score, otherwise the score of player 2 is saved as high score. The BufferedReader is closed by reader.close (). A try and catch is used so that when errors occur, it will allow the game to start. Then a BufferedWriter is called to add the new high score to the file. A try and catch is also used here.

2. Class QuitHscore

The class QuitHscore is provided to create the 2 JButtons of the game board, namely "Quit Game" and "High Scores". These are later linked to a frame is the Class StartFrame. That is why no JFrame is created in this class and immediately starts writing setPrefferedSize, ... An actionListener is created for both buttons. The actionListener for the "Quit Game" button is easier because it simply has to exit the game. At the actionListener of the "High Scores" button you have to go to another JFrame, in which all scores are displayed. This is done by a WindowEvent. For showing the high score there must be created a JFrame. This is done in the hScoreF method called "Highscore". In this a JTextArea is provided in which the text file of the high scores can be displayed. I set "setEditable" false, because otherwise things can be rewritten. Then a BufferedReader, a BufferedWriter and an ArrayList are used. First the BufferedReader will read the text file, by looking at it line by line. Each line is added to the Arraylist "line" by line.add(currentLine);. Then we can sort the array line by 2 functions (Collections.sort (line); and Collections.reverse (line);), which first sort the array from lowest to highest and then reverse that the highest number is at the top. Then the BufferedWriter is used to add the newly sorted array back to the text file. Finally, all the BufferedWriter and BufferedReader are closed and a scanner is used again that will read the text and display it in the JTextArea.

3. Class PanelLabels

In the class PanelLabels the 3 labels (JLabels), "Score Player 1", "Score Player 2" and "Turn", are created. These labels are also linked to a JFrame in the class StartFrame. Also 3 other JLabels are created (pl1, pl2 and turnsPs). These are linked in BattleshipGrid and BattleshipGridPlacement and are therefore continuously updated for each player via the mouseClicked method. For this we use an empty string ("") for pl1 and pl2 because they always get a different period. TurnPs gets a fixed string "Player" followed by a number, depending on which player is on the turn.

4. BattleshipGridPlacement

BattleshipGridPlacement is a copy of the BattleshipGrid class. I did this so that I could link my BattleshipBoardPlacement class here. This is actually the only difference between the 2 classes, just like the attack2 method with the class Player. This could probably have been done in a more efficient way by using a boolean as mentioned above. With the saveHighScore method, the same text file is called, so the high score method remains exactly the same.

5. Class StartFrame

All previous classes come together in the class Start Frame. To keep things clear, I started making the Start menu frame (constructor). For this I made several JButtons, JLabels, JSpinners, JRadioButtons and JPanels. Then I made ActionListeners, so that the buttons, spinners and radio buttons that were made could perform an action. In one actionListerner I just used an ActionEvent, which aims to just run something. With other actionListeners I also used a WindowEvent, so that the previous JFrame can be closed and a new Frame can be opened. Finally, I linked all buttons, spinners, … to the frame.

Then I started working on different methods. The first method is rulesFrame, the new frame that has to be opened when the button "Rules" is pressed. I used a scanner to read the text file in which I wrote in the rules. The next method

is hScoreFrame, which is the same as the high score Frame, as described in the class QuitHscore. Since I have not found a method to link the created frame to this button, I also have duplicate code here.

The boardFrame method is the frame to which the classes PanelLabels, QuitHscore and BattleshipGrid are attached. As mentioned, no JFrame was created in these classes. The frame is created in this method, with the classes positioned on the frame. Since the constructor has a number of conditions, I was able to link these conditions to all other buttons of the GUI, such as scoringMode. This is linked to the JRadioButtons, so the scoring mode has 2 options. With an if statement an integer is given to the JRadioButtons, this integer is then placed in the conditions of the link to the constructor of BattleshipGrid. This is then read in the BattleshipGrid class and thus the link is made between the JRadioButton and the choice of the correct scoring method. Also important is that these terms say "new BattleshipBoard (row, col)", since we want to play on one specific board, I had to make sure this is always done during the same game. Here the “row” and “col” will be linked to the JSpinners in startFrame, by saving the value of the JSPinner.

The shipPlacementFrame method creates a frame with a JTextArea, like hscoreFrame, but this time I set "setEditable" to true. This allows the players to rewrite the text. Then I added the text file ShipPlacement to the text area and I created an actionEvent that ensures that firstly the changed text in the text area is saved in a new text file (newShipPlacement.txt), which can then be read in the class BattleshipBoardPlacement, which places the boats on the board, after this BoardFrame2 is displayed. Which is actually similar to boardFrame, with an extra scanner so that the rows and columns are added to the board.

6. Class MainGUI

Ultimately, all classes are linked to the class StartFrame in a way, so that in the main method we only have to refer to the class StartFrame. We do this by adding *SwingUtilities.invokeLater*. By using the invokeLater method of the SwingUtilities class, I make sure that all GUI related elements are executed on the Event Dispatch Thread. When we finally run everything, we get a working game. The result after you press "Start Game" is shown in figure 4.

Part 3: Relationships between the classes

I have described most of the relationships in the brief explanation of each class. The figure below (figure 5) is a representation of the classes used and in which they are linked.

Strengths and weaknesses in the project

As mentioned before, I often have duplicate codes in my project. This was because I could not make the link between 2 parts that were similar. An example are the 2 classes BattleshipBoard and BattleshipBoardPlacment. The big problem with this was that since these classes were already different, I also had to copy a lot of code during the rest of the project. I also found it difficult to keep everything organized so that it looked clear. Since I have gone through the code many times myself, I know what the code means and how it works, but I think an outsider needs time to understand everything. One way to make everything more transparent was to create a class for all fixed variables, such as PANELWIDHT or HAS_CARRIER. This could have meant that I had to write less code. In the end I am satisfied with the end result because I have been able to implement all parts. By starting the project on time, I had time to develop everything and make it work.

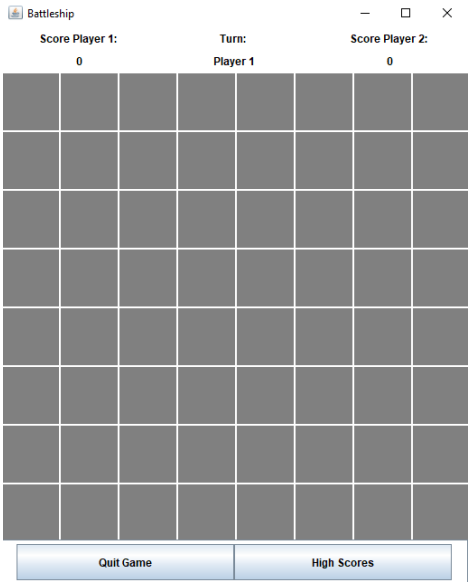

Figure 4: game board

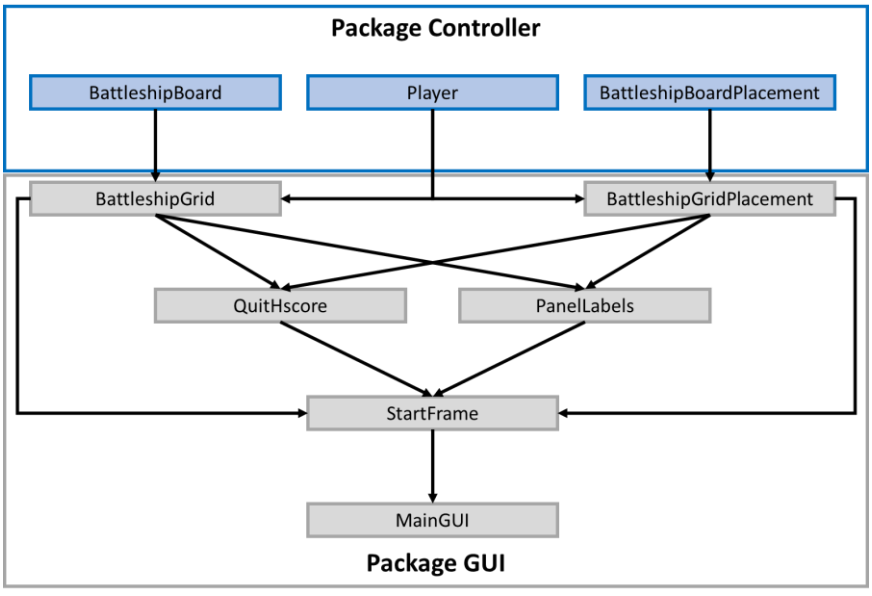

Figure 5: simple diagram of the used classes