David Müller

r0827111

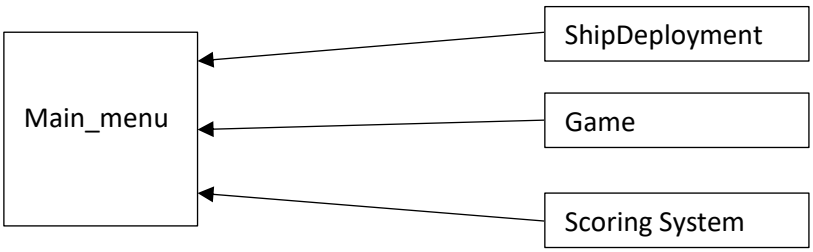
Erasmus Exchange Student

I0S75A Basic Programming

Project Report

Code Structure

The code is consisted of 4 classes. The main class is *Main_menu* where all initial parameters can be set – size of board, ship deployment or scoring system. Other classes – *ShipDeployment*, *Game* and *Scoring System* are somehow connected to the main class.



In *Main_Menu* class are first imported all packages, then are defined all variables

In method Main_menu all buttons, labels and function of buttons are adjusted, and they are added to the main frame:

```
//add elememts
frame.add(authorButton);
frame.add(exitButton);
frame.add(startGameButton);
frame.add(shipDeploymentButton);
frame.add(highScoreButton);
frame.add(rulesButton);
frame.add(chooseScoringSystemButton);
frame.add(sizeLabel);
frame.add(slider);
frame.setVisible(true);
```

If the The *authorButton* button is used, window with information about the author (me) pop up:

```
//authorButton function
authorButton.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent event){

        JOptionPane.showMessageDialog(frame, "<html>Author: David Muller <br> r0827111 <html>");
                }
            }
        );
```

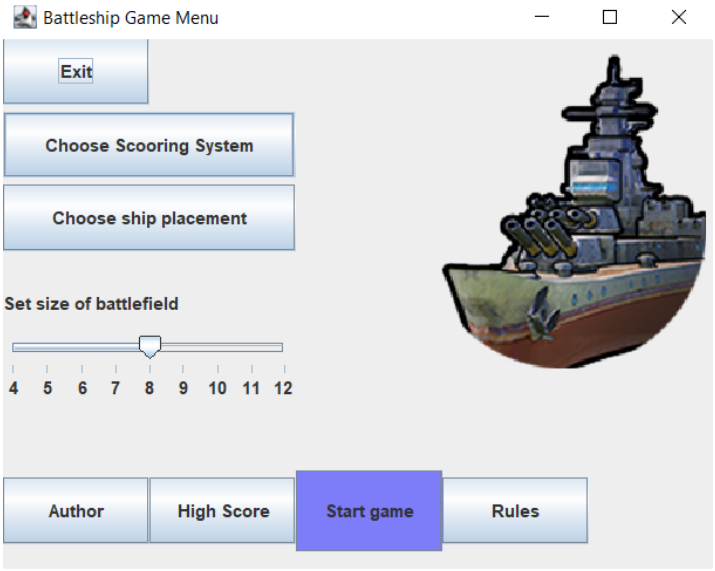
Buttons highScoreButton and rulesButton has function based on similar principle.

The *exitButton* will close the *Main method*:

```
//exitButton function
exitButton.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent event){
        frame.dispose();
                }
            }
        );
```

Buttons highScoreButton and rulesButton has function based on similar principle.

The *main* function in Main_menu is used only launch main menu GUI:




```
// main method
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            new Main_menu("");
        }
    });
}
```

The *shipDeploymentButton* calls the *ShipDeplyment* class and open 2 windows, where players can deploy their ship via GUI:

```
//shipDeploymentButton function
shipDeploymentButton.addActionListener(new ActionListener( ) {
    public void actionPerformed (ActionEvent event){
        shipDeployment.newCanvasSD(slider.getValue(),slider.getValue());
        shipDeployment.newCanvasSD2(slider.getValue(),slider.getValue());
        shipDeployment.deployShipCanvas(slider.getValue(),slider.getValue());
        shipDeployment.deployShipCanvas2(slider.getValue(),slider.getValue());
        shipDeployment.deployShip(slider.getValue(),slider.getValue());
        shipDeployment.deployShip2(slider.getValue(),slider.getValue());
    }
}
);
```

In the Battleship deployment window, player must place all the ships and save the deployment. The ship can be placed horizontally (left side of the ship will be placed in the button) or vertically (top side of the ship will be placed in the button). There is also reset button, that clears the field from ship and player can deploys ship again.

In the *ShipDeployment* class are first defined all the variables similarly as in in *Main_Menu* class. *ShipDeplyment* class contains methods:

- *deployShipCanvas*, that generates grid of buttons
- *addCoordinates,* that saves coordinates of deployed ships
- *ship,* that is used for generating any ship and consists of variables

- *numberOfMoves*, that is used to decide in which order will the ship be deployed
- coordA and coordB, that set first coordinates (x and y) in the grid
- lengthOfShip, that set length of the ship
- *type*, that assigns type of ship

- deployShip, that generate specific ship in specic order of deployment

If all the ships are deployed, the game can be launched

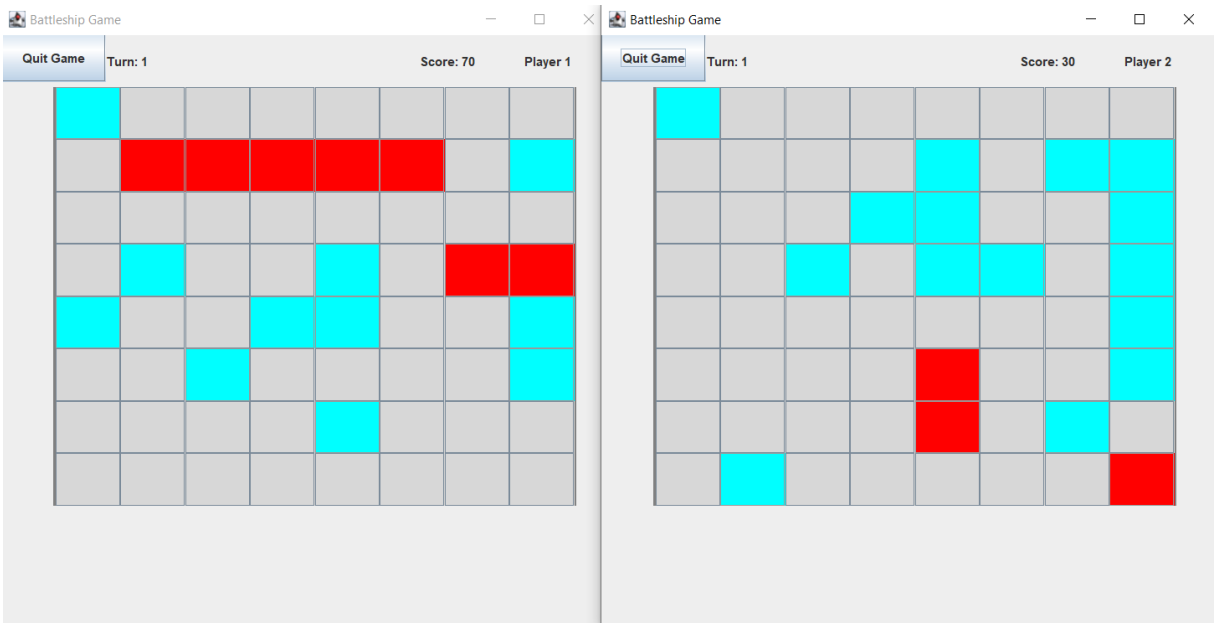
```
    //startGameButton function
    startGameButton.addActionListener(new ActionListener( ) {
        public void actionPerformed (ActionEvent event){
            if (ShipDeployment.shipDeployed == true && ShipDeployment.shipDeployed2 == true) {
            playGame.play(slider.getValue(),slider.getValue());
                    }
            else {
                System.out.println("Ship was not deployed");
            }
                }
    }
            );
```

Class for game itself is *Game*. In *Game* class are first defined all variables. Class Game consists of methods *playerXBattlefield, playerXMove, scoreXAdd* for each player (X = 1 or 2), *player ,setTurn and getHighScore*. In method *player* is set the GUI of the game. The method *player* is launched by the *play* method. Methods *player1Battlefield* and *player2Battlefield* set grid into GUI. Methods *Player1Move* and *Player2Move* compare saved coordinates from class *ShipDeployment* with coordinates obtained by clicking on arbitrary button in grid. If they are same the button changes its colour to red and add score to opposite player.

```
if(a ==setX[i] && b ==setY[i]){
    grid1[a][b].setBackground(Color.RED);
    grid1[a][b].setEnabled(false);
    score1Add();
    break;
}
else {
    grid1[a][b].setBackground(Color.CYAN);
    grid1[a][b].setEnabled(false);
}
```

Methods *scoreAdd* and *move* are used for adding score and number of turn.

Method *play* is used to launch the Game window and it is used in *Main_Menu* in button *startGameButton*.



Method *setTurn* allows switching game between Player 1 and Player 2 after a turn is played.

Method *getHighScore* returns hight score.

Game will ends when player reveals all opposing ships

Last class ScoringSystem is for choosing scoring system.

In first case is scoring system equals for both players. In second case the points-per-hit are different for each player in order to compensate the player going second.

Strength and weaknesses of the code

From all the points if the assignment I wasn´t able to implement different scoring for the specific ship because I didn't have enough time for it. The reason is that I spend lot of time to implement better ship placement via GUI (see class ShipDeployment).

I would say that the strength of this code, that the game itself works really good. On the other hand weakness of the code is, that the code is too long, because some method I had to used 2 times for each player (I spend lot of time to try implement only one method for each player but I didn´t work).

Overall I spend on this project really lot of time, because I didn´t have had any prior experience with coding before and even If the code is not far perfect and could be more simplified, I have learned a lot on this project.