JAVA project: Battleship

Stijn Staring (r0620003)

December 17, 2020

1 The Game

1.1 Introduction story

It is the year 1659 and two rivalling Italian towns are competing for dominance over sea. However, attracted by the wealth of both city states, pirates from the far east have come to raid the prospering towns. In order to repel the attackers the cities have made an agreement to set their difference aside and face the common enemy. The leaders of both cities know however that the whole of Italy is watching them and as soon as the pirates will be defeated, the rivalry will be resumed. In order to gain the most prestige, it is of vital importance to be the clear saviour of Italy and be the one that conflicts the most damage upon the pirates.

1.2 Rules

Setting up the game: When the game is started a selection screen will pop up where the players can set their preferences. The rules of the game can again be requested from this frame. During the game the players have to guess where they think the pirate ships are located on a board with a grid layout. Each cell in the grid matches with a certain coordinate defined for example as $5 * 0$. Note that the first row and column have the index zero. Under each cell of the grid a part of a pirate boat can be located.
The "**Choose Ship Placement**" button allows the players to load a text file that specifies the location of the ships. A template file can be seen in Figure . If no such file was uploaded, the placement of the ships is automatically done at random. The first row of the text file should specify the board size by defining respectively the amount of rows and columns of the board. Next, the ships are specified. The order in which this is done is irrelevant, but the ship sizes and names should correspond with the ones as shown in Table 1. There should be only four different ships on the board. Also, it should be taken into account that the ships can't be separated. In order to be more robust the system will only take the first two coordinates given when the length of the ship is wrong. When other errors occur e.g. the ship has overlap with other ships or is not laying in the board, the user will be noted that random placement of the ships will apply.

```
13;13
Carrier;3*1;3*2;3*3dres;3*4;3*5
Battleship;4*6;5*6;6*6;7*6
Submarine;5*2;6*2;7*2
Destroyer;1*6;1*7
```

Figure 1: An example text file that can be provided to define the ships on the board.

Ship name	**Length**	**Colour**
Carrier	5	*red*
Battleship	4	*green*
Submarine	3	*yellow*
Destroyer	2	*white*

Table 1: Overview of the characteristics of the different ships.

Next, the "**Choose Scoring System**" button allows the option between an equal score system or an adjusted one. The adjusted score system takes into account that the second player also has information of the move of the first player. Therefore, player one will receive a higher score when a ship is hit. Table 2 gives an overview of the

points that can be used. As explained in the "Game Play" a player receives double points when he hits the last unrevealed cell of the ship.

Ship name	**Standard score**	**Higher score**
Carrier	10	11
Battleship	15	16
Submarine	20	21
Destroyer	25	26

Table 2: Overview of the scores that can be used.

The spinners that are located on the selection screen can be used to change the dimensions of the board which specifies the amount of cell in the game. The initial value is set to eight and the maximum is thirteen.

Game Play: When the players start playing the game, a board will appear with grey cells. The players take turn by clicking a grey cell where they think a pirate ship is located. The score and whose turn it is, will be indicated at the top of the screen. The players keep clicking the tiles until all four ships have been found whereafter the game automatically ends. When a player hits the last tile of a ship the ship will sink and a double of the points will be received. After a game is finished, the score of the winner will be assessed if it is high enough to be added to the current list of high scores. Only the top five results are being saved. Enjoy!

2 Code description

2.1 Define classes

Figure 2 shows the different packages in braun and the classes that they include. Now, a short description is given for each individual class.

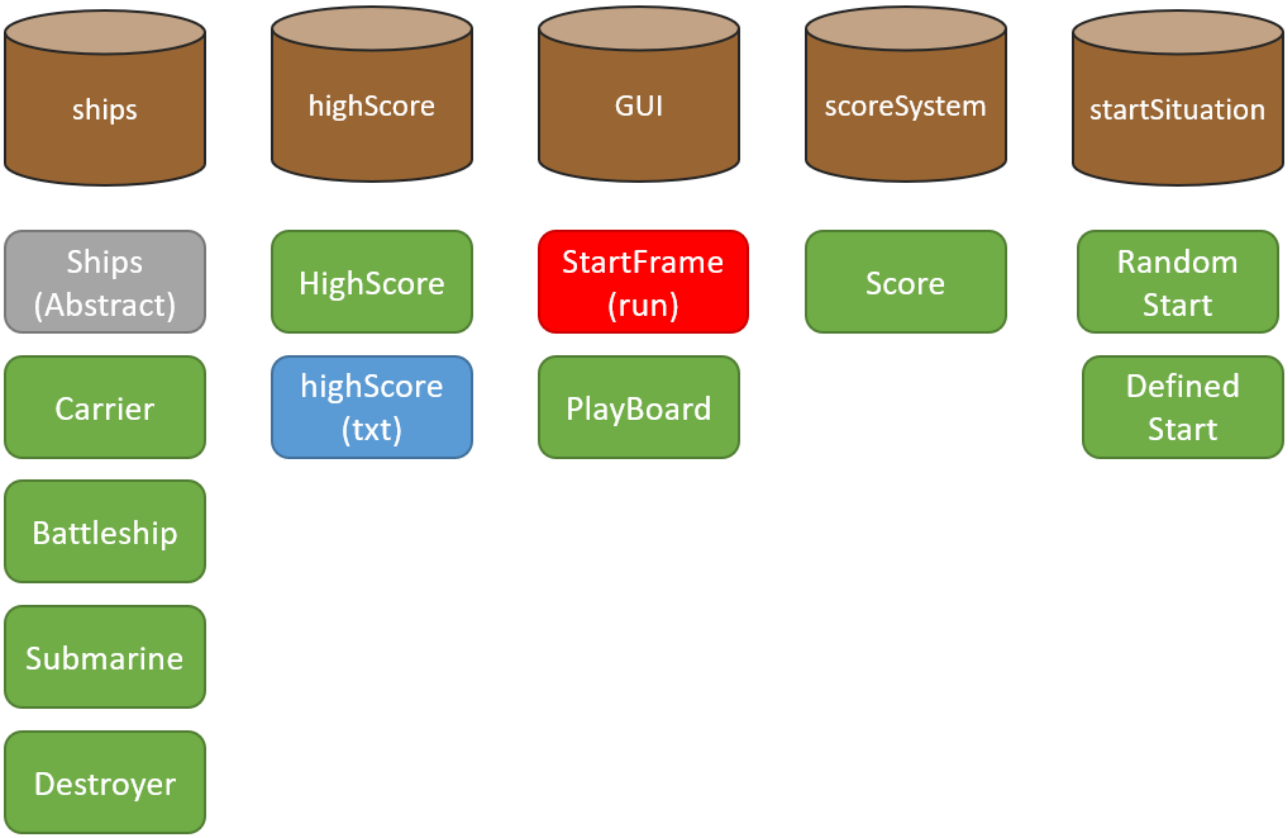

Figure 2: An overview of the different packages (Braun), classes (Green), abstract classes (Gray) and text files (Blue).

Ships
The Ships class is an abstract class that contains the functionality that the specific ships have in common. Functionality that is included is:

- getName
- setStartIndex → coordinates of the start of the ship
- setDirection → direction of the ship on the board
- getLength of the ship → abstract method because the length differs for each ship but the method is already needed in this class
- checkShot → is there a ship under the clicked tile or not if so increase the damage
- isSunk → is the ship completely damaged
- allUsedIndices → get from the start coordinates and direction all the coordinates occupied by the ship

Carrier/Battleship/Submarine/Destroyer
This class specifies the length of each unique ship and can implements the abstract function getLength.

HighScore
At the start of the program the current high scores that were saved in the highScore text file are read in by a buffered reader which provides buffering of data for fast reading. The functionalities that this class overs are:

- updateHighScore → this method takes the score of the winner into account and checks if it should be included in the list of high scores
- convertToString → converts the ArrayList which stores the high scores to a string format
- saveHighScore → when a game is played, the text file on the hard drive is updated

StartFrame
In the StartFrame class the selection panel is build. This is also the only file that contains a main method.

PlayBoard
In the PlayBoard class the play board is build.

Score
In this class the scores are given for each hit are assigned. The constructor is overloaded when an adjusted score is used. The updateScore method update the score after a player clicked a tile.

RandomStart
This class places the ships in a randomly way on the board. Its methods consist out of:

- noOverlap
- inBoard
- generateLegitimatePlace → which produces a set of start coordinates and a direction to define where a ship is placed on the board.

Define start
Here the user provided text file is read in to define the placement of the ships on the board. It is checked if the user input is feasible. A method defined here is to get the direction of the ship out of two sets of coordinates.

2.2 Relations between classes

Table 3 shows the connection between the different classes. In green is the connection between the specific ships indicated, that extends the abstract ship class. The classes defined on the rows composes (red) or imports (orange) the classes shown at the top. Composition in this context means that the an object of a different class is used as an instance variable. Importing a class means that an object of a different class is used but not as instance variable.

	Sh	**Ca**	**Ba**	**Su**	**De**	**Hi**	**St**	**Pl**	**Sc**	**Ra**	**De**
Ships											
Carrier											
Battleship											
Submarine											
Destroyer											
HighScore											
StartFrame											
PlayBoard											
Score											
RandomStart											
DefinedStart											

Table 3: Overview of relations between the different classes. The figure is read from the rows to the columns. extends (Green), composes (Red) or imports (Orange)

3 Conclusion

An advantage of using the code which makes use of an object oriented programming language, is that it is easy to separate the different tasks and bundle them in a class. When the functionality is needed, it can be called and stored by the means of an object. This also makes the code more clear and will therefore easier to maintain. During the project I could rely on a vast amount of explanation and a there are a lot of libraries available for JAVA. After some search work, you will most likely find what you want. Because the different tasks of the project are nicely separated and stored in classes and objects, the functionality can be called on different places and this reusing of code, makes in my opinion the development of code faster.

Some disadvantages were however the steep learning curve. Not everything was directly clear and I had to put the needed time and effort to understand the different concepts. For example key techniques such as inheritance and polymorphism (overloading and overwriting) in Java were first not that intuitive to understand.

References