**Project BATTLESHIPS (TEAM PDSD).**

**Introduction:**

**Student names/Matriculation Numbers:** Daniel Blackley/160007728, Patryk Jakubek/170011784, David Adesanya/, Sebastian Thomas/

The following document is a rough design which we will use to as a guideline when developing a game of battleships in the java programming language.

**Requirements**

**Statement of Requirements**

Our task is to create a "battleships" game which involves a user guessing where the location of where the ship is. The game includes two grids, capable of housing ships. Two players, once being a user while the other being the computer each use one of the grids to place ships. Once all the available ships are placed, the players start picking coordinates on the opposite grids. If the selected coordinate contains a ship, a point is added to the current players' score and allows the player to select a coordinate again, for all other scenarios, it changes to next players turn. The game continues until all opposite player ships are sunk.

When playing the game, there will be the possibility of two players or a vs. computer, the user will be at the age they are able to comfortably use a computer (we assumed age 7). By using a computer, we assumed that the user will have a keyboard. We assumed that by using a computer, the user will be running an operating system cable of running java executables. Finally, we also assumed that the user has access to a speaker or headphones. [Visual Clue – user feedback]

The input for the program currently is a text file. The output of the program currently is visual.

**Functional Requirements**

* The user should have the option to use mouse or keyboard to play the game.
* The user should be able to place a ship on his board.
* The ship should be visible when it is placed on the grid.
* There should be another grid that shows where you have fired missiles and whether it has been a successful hit or miss.
* The fleet must consist of 1 battleship which is 4 grid spaces long, 2 cruisers which are 3 grid spaces long, 3 destroyers which are 2 grid spaces long and 4 submarines which take up a single grid space.
* User will be able to save the current game.
* User should be able to load previous game data to run the game.

**Non-Functional Requirements**

* It should help the player improve their skills in playing battleships by giving them practice.
* The game should be easy to play and user friendly – so it must have visually appeasing buttons. In addition, the rules and instructions should be simple so everyone can understand it.
* The game should be in an enjoyable format – use of complimentary background colours to brighten up the game.
* The game will load compressed files to form the board, this will help to improve the efficiency of the program, so rather than creating the board each time, the board would be loaded. Also when saving the game, the file will in the same compressed format to save up space on the computer.

**Minimum hardware specification:**.

* Keyboard – to play the game.
* Monitor – this displays the game, so the user can see it.

**Use Cases**

**Likely Use Cases:**

Children Aged (7+)

Children Aged 7+ Are likely to use this application as anyone under the age of 6 is a lot less likely to have access to or be able to understand how a computer functions enough to navigate and operate the game. Children also need to be old enough that they can understand the rules of battleships despite it being a fairly simplistic game.

Teens (7 – 18)

Teens are likely to be the majority of the audience as this is the age that most people spend time primarily on the internet at this age due to them being old enough to understand and operate it easily. This is also when most people are interested in video games.

Young adults (18 – 21)

Young Adults who still use the internet frequently may also use this application when bored at older ages.

USER: SYSTEM

|  |  |
| --- | --- |
| USER: | Places a Ship |
| SYSTEM: | Checks if the current ship is able to fit in the current Squares provided. |
| SYSTEM: | Inform user if no space available. |

|  |  |
| --- | --- |
| USER: | targets a square on the board |
| SYSTEM: | Checks if there is an enemy battleship over that square |
| SYSTEM: | Displays appropriate information. |

|  |  |
| --- | --- |
| USER: | Clicks the ‘Save’ Button |
| SYSTEM: | The game generates a text document. |
| SYSTEM: | Writes X and Y co-ordinates of current Battleships to file. |
| SYSTEM: | Writes Where all hits and misses are to file. |

**Software:**

* [Depending on how the code is executed – either compiled or interpreted], compiled: Any operating systems that has Java installed, interpreted: any operating system that has a Java interpreter installed.

**Candidate Classes:**

|  |  |  |
| --- | --- | --- |
| Candidate Classes | Accept / Reject | Reason for rejection |
| Ship | **Accept** |  |
| Board | **Accept** | **Duplicate of grid** |
| User | **Reject** | **Too vague, Instance of Player Class** |
| Task | **Reject** | **Too vague** |
| Game | **Accept** |  |
| Computer | **Reject** | **Too vague, Instance of Player Class** |
| Battleship | **Reject** | **Sub class of Ship** |
| Grid | **Accept** |  |
| Ships | **Reject** | **Plural of ship** |
| Coordinate | **Reject** | **Too vague, Field of Grid** |
| Turn | **Reject** | **Too vague, Field of Game** |
| Player | **Accept** |  |
| Sunk | **Reject** | **Too vague, Instance** |
| Score | **Reject** | **Field of Class Player** |

**Class Descriptions including Responsibilities, Fields and Methods**

**Class: Game**

It is responsible for all information needed to start the game and keep track of who's turn it is currently in the game. It also needed to when one needs to reset/load the game.

Fields:

Type: boardSize: holds the size of the current board.

ShipNumber: holds the number of ships in play

Methods:

SetType(String): string - Sets the type of game it is, e.g. against computer.

playerShot: lets the player take a shot

Savegame: saves the game to a text file.

LoadGames: loads a previous game.

getTurn: string - Fetches whose turn it is.

**Class: userInput**

Userinput class is responsible for all information to do with the battleships in play.

Variables:

ShipLength: int - Contains the current ship length

Methods:

playerShot: tells the computer what to do with the players shots

ValidateInput: makes sure the player is actually using integers

**Class: Computer**

The computer is responsible for all random information concerning the current board in play

Variables:

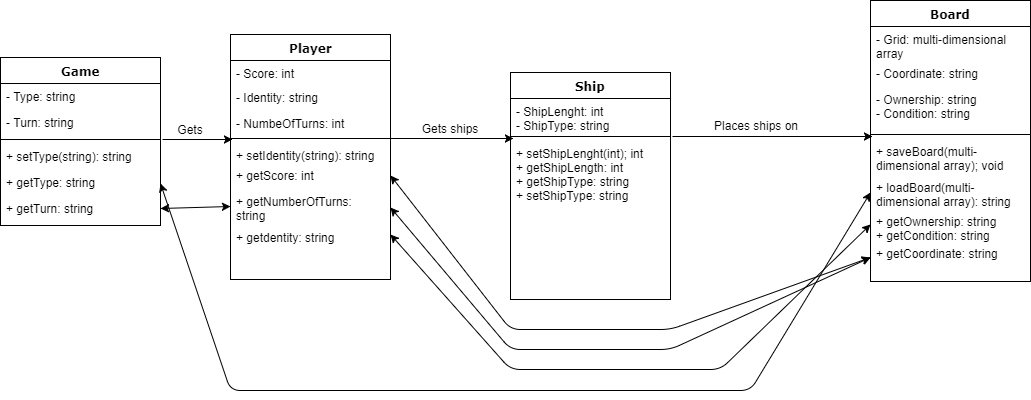
Random: used to generate random numbers

Methods:

Setcomputerships: sets the computers ships to be in the correct place

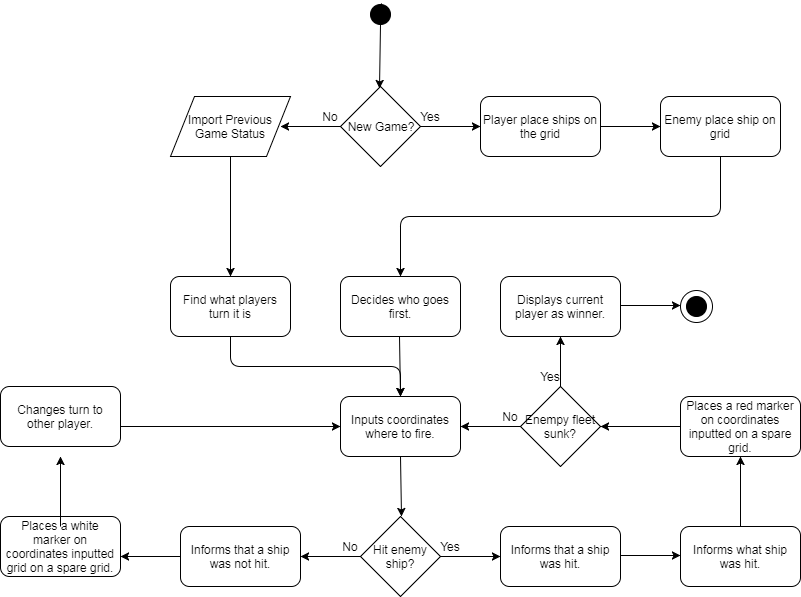
SetcomputerShot: sets the computers shot to hit a random place on the board.

**Class Diagram**

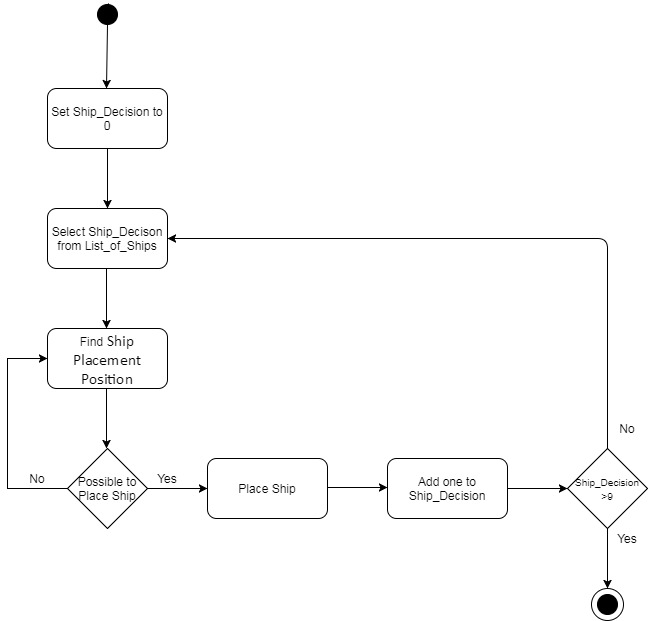


**Activity Diagrams / Pseudocode**

**Activity Diagram**



**Enemy place ship on grid**



**Pseudocode**

**Enemy place ship on grid**

1. Select Ship

2. Find Position

2. Place Ship

Top Down Refinement

1. Select Ship

1.1 Select First ship from List of Ships

2. Find Position

2.1 Select mini-grid from 9 available

2.2 Select grid-box within the mini-grid

2.3 Check whether the ship can be positioned in the selected grid-box

2.4 Check whether enough space to position the whole ship

3. Place Ship

3.1 Decide whether to be placed horizontal or vertical

3.2 Place Ship

3.3 Lists of ships + 1

Further Refined

2.1 Select mini-grid from 9 available

2.1.1 Randomly pick number between 1 and 9

2.2 Select grid-box within the mini-grid

2.2.1 Randomly pick number between 1 and 9

2.2.2 Check if the grid-box is in use

2.2.3 Repeat Step 2.2.1 if grid-box is in use

2.3 Check whether the ship can be positioned in the selected grid-box

2.3.1 Check if the grid-box is in use

2.3.2 Repeat Step 2.2 if grid-box is in use

2.4 Check whether enough space to position the whole ship

2.4.1 Find the O\_Number which is the size of the ship from center point

2.4.2. Check if number of empty grids to the left is greater than or equal to O\_Number

2.4.2.1 If true then Left\_Side\_Empty = true

2.4.3 Check if number of empty grids to the right is greater than or equal to O\_Number

2.4.3.1 If true then Right\_Side\_Empty = true

2.4.4 Check if number of empty grids to the Top is greater than or equal to O\_Number

2.4.4.1 If true then Top\_Side\_Empty = true

2.4.5 Check if number of empty grids to the Bottom is greater than or equal to O\_Number

2.4.5.1 If true then Bottom\_Side\_Empty = true

2.4.6 If Left\_Side\_Empty = True and Right\_Side\_Empty = True then Horizontal-Placement = True

2.4.7 If Top\_Size\_Empty = True and Bottom\_Size\_Empty = True then Vertical-Placement = True

3.1 Decide whether to be placed horizontal or vertical

3.1.1 Check if Vertical-Placement = True & Horizontal-Placement = True

3.1.2 Select one that is true

3.1.3 If both are true then select randomly