**Battleship Project**Date:27/09/17  
Student Names: Dale Brooksby, Peter Leslie, Tom Smith, Yousaf Nasir  
  
**Statement of requirements**   
  
The aim of this project is to create a functional game of Battleships in Java. The game will involve the computer randomly placing ships on a board. The user will then select which grids to shoot at, in hopes of hitting the computer's ships. The game is finished when all the computers ships are destroyed. It will be assumed the users know the rules of battleships prior to playing. The program will be played by fellow students. These users will be able to make inputs via keyboard. The game will feature the ability to save their current state and reload that save. Furthermore the program will feature a menu from which users can adjust the programs options.

**Requirements  
Functional Requirements**  
  
R1. The program should be able to randomly place ships at the start of a game  
R2. The program should display which grid spaces have been shot or missed.  
R3. The program must be able to Save the current game state  
R4. The program must be able to reload that save state; or any other saved file.  
R5. The program should include a menu which users can adjust game options.  
R6. The program should have a graphic display window that shows the current game state.  
R7. The program must allow users to fire a shot at a grid position

**Non-functional Requirements**  
  
R1. The UI needs to be aseptically appealing with a clean design

R2. The controls must be easy to use.  
R3. Grids must change their status quickly after having been selected

**User Interface**

The program will have a console based UI. The user will enter commands or coordinates into the command line, and the board will then be displayed as a grid of characters.

At first, the grid will be full of default ‘-‘ characters, since the user doesn’t have any information yet.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |

Note: in the console, there won’t be grid lines. One programming challenge will be keeping all the rows and columns in order.

When a user makes a guess, it will either be a hit or a miss.

On a hit, the ‘-‘ character will change to an ‘x’, and the console will output “Hit!”.

On a miss, the console will output “Miss…”. For added challenge, the missed tile won’t be updated, so users must remember or keep track of their guesses.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - | - | - | - | - | - | - | - | - | - |
| - | - | x | - | - | - | - | - | - | - |
| - | - | x | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | x | x | x | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | x | - | - | - |
| - | - | - | - | - | - | - | - | - | - |
| - | - | - | - | - | - | - | - | - | - |

### Use Cases

|  |  |  |
| --- | --- | --- |
| Main menu | | |
| 1 | System: | Display menu |
| 2 | User: | Select option: |
| 2.1 |  | New game |
| 2.2 |  | Load game |
| 2.3 |  | Options |

|  |  |  |
| --- | --- | --- |
| Battleship Game: Start new game (2.1) | | |
| 1 | User: | Start a new game |
| 2 | System: | Initialize board |
| 3 | System: | Generate random locations for ships |
| 4 | System: | Display board |
| 5 | System | Start game loop |

|  |  |  |
| --- | --- | --- |
| Battleship Game: Load saved game (2.2) | | |
| 1 | User: | Load from saved file |
| 2 | System | Read save from file |
| 3 | System: | Transfer file contents to board |
| 4 | System: | Start game loop |

|  |  |  |
| --- | --- | --- |
| Battleship Game: Game loop | | |
| 1 | User: | Selects grid space to shoot |
| 2 | System: | Determine if ship was hit |
| 3 | System: | Update UI to reflect hit or miss |
| 5 | User: | Starts next turn |
| 6 | System: | repeats steps 1 - 5 until all ships hit |
| 7 | System: | Displays a Game Over screen |

### Classes

##### Candidate Classes

The aim of this project is to create a functional game of **Battleships** in Java. The **game** will involve the **computer** randomly placing **ships** on a **board**. The **user** will then select which grids to shoot at, in hopes of hitting the computer's ships. The game is finished when all the computers ships are destroyed. It will be assumed the users know the rules of battleships prior to playing. The program will be played by fellow **students**. These users will be able to make inputs with a **mouse or keyboard**. The game will feature the ability to **save** their current state and **reload** that save. Furthermore the program will feature a **menu** from which users can adjust the programs **options**.

|  |  |  |
| --- | --- | --- |
| **Candidate Classes** | **Result** | |
| Controller | Accept | Top level coordinator |
| Game | Reject | Irrelevant |
| Computer | Reject | Irrelevant |
| Ship | Accept | Ship class |
| Board | Accept | Board class |
| User | Reject | Irrelevant |
| Computers ships | Reject | Already a class (Ship) |
| Game | Reject | Irrelevant |
| Rules | Reject | Irrelevant |
| Battleships | Reject | Already covered (Ship) |
| Program | Reject | Irrelevant |
| Students | Reject | Not necessary |
| Mouse or keyboard | Reject | Irrelevant |
| **Save** | Reject | Only needs a method |
| **Reload** | Reject | Only needs a method |
| Menu | reject | Part of controller class |
| **Options** | reject | Not necessary |
| **Input Output** | Accept | Handle all file saving and loading |

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

**Controller**

* + **Description:** Top level class. Calls main method, and coordinates all other classes in the program.
  + **Fields:** Board[2] userBoard – Two board objects.
  + **Methods:**
    - main(): Main method, called when program runs
    - gameLoop(): Handles the flow of the game.
    - processUserInput(String stringInput): Handles the users input, including converting string coordinates into integers, and calling save/load methods when inputted.
    - processHit(int[] intInput): Determines whether a ship has been hit, and outputs a message accordingly.
    - checkVictory(): Determines whether the user has won.

**Ship**

* + **Description:** Stores the fields and methods about the ships that are used in the game
  + **Fields:**
    - Length (int): How many tiles the ship occupies
    - Destroyed (Boolean): Whether the user has hit the ship
  + **Methods:**
    - getDestroyed: returns a Boolean value for whether the ship has been hit
    - setDestroyed: sets a ships destroyed field to true (cannot be set to false)
    - getLength: returns an int value of the length of the ship

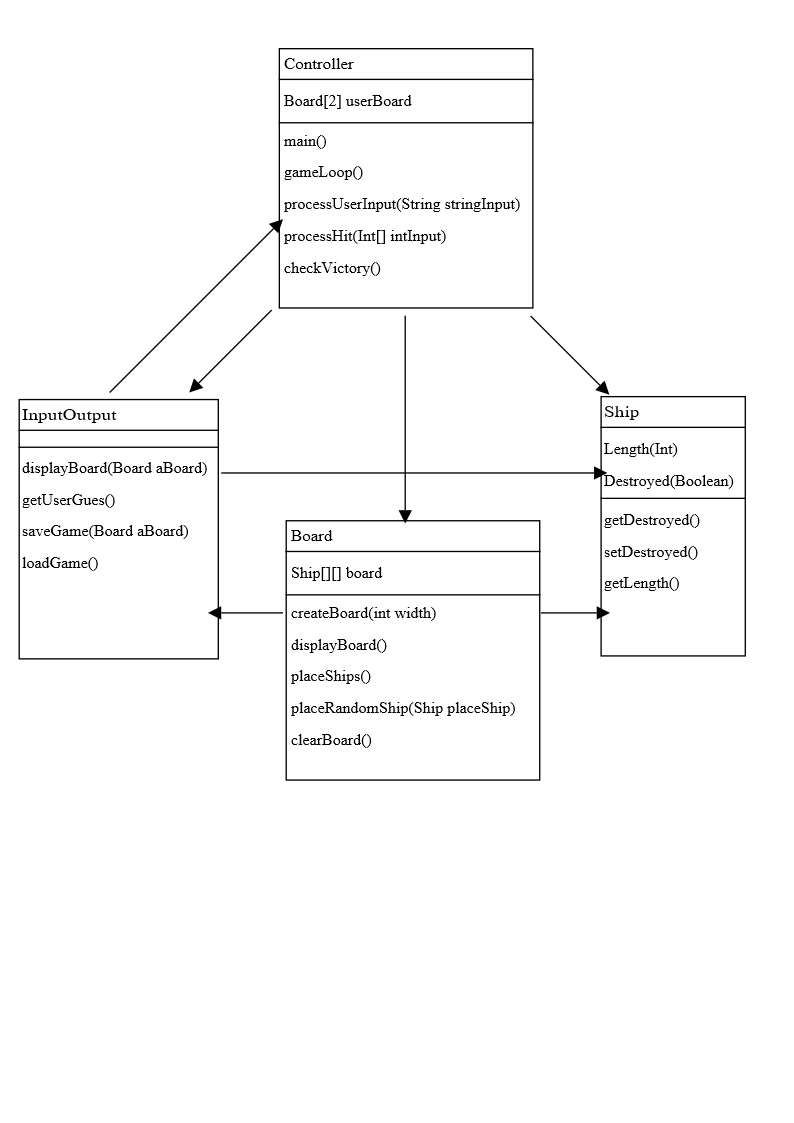
**Board**

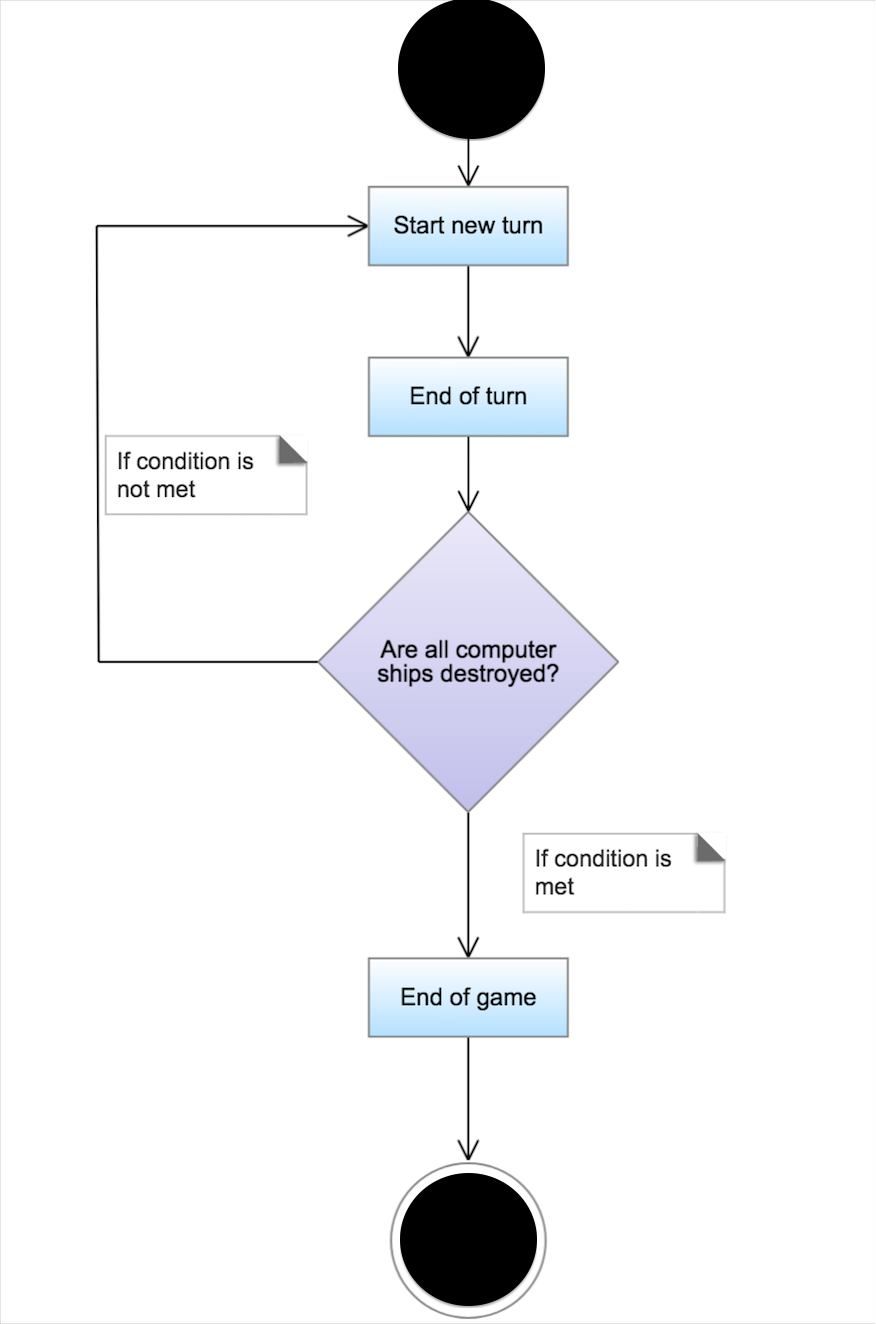
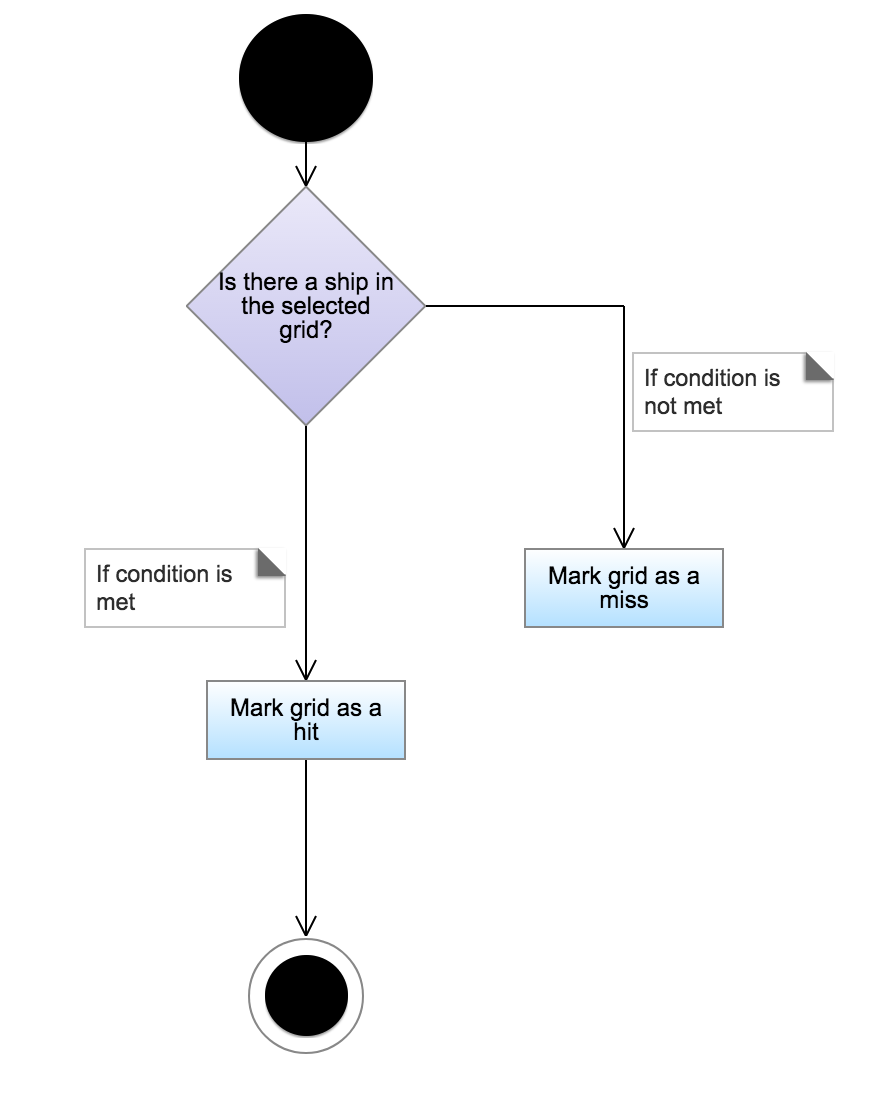
* + **Description:** saves information about the board and which grid spaces are occupied by a ship.
  + **Fields:**
    - Ship[][] board: A 2d grid of Ship objects, to be used as a board
  + **Methods**
    - createBoard(int width): initializes the *board* field, with the height and width values set to the width parameter.
    - displayBoard(): Calls the display method in the IO class.
    - placeShips(): Initializes the ships to be placed, and passes them into the placeRandomShip method
    - placeRandomShip(Ship placeShip): Places the ships in random places on the board
    - clearBoard: Writes empty lines to the console, to make the current game state clear.

**InputOutput**

* + **Description:** Handles all information going in and out.
  + **Fields: None**
  + **Methods:** 
    - displayBoard(Board aBoard): Outputs the contents of a board object to the console
    - getUserGuess(): returns a String value of the users input.
    - saveGame(Board aBoard): Writes the contents of the board to a text file
    - loadGame(): returns a Board object. Reads a text file, and writes the contents to a board.

##### Class Diagram



**Activity Diagrams**

**Pseudocode**

CREATE boardArray of type ship[9][9]

GET menu input FROM mouse

**New game**

-Place random ships-

**Load game**

GET file data FROM system

SET boardArray TO file data

**Game loop**

WHILE playerWon = FALSE

IF there are enemy ships left

GET user input FROM keyboard

seperate input into x and y

IF boardArray[x][y] contains ship

ship.shipType = destroyed

END IF

ELSE

SET playerWon TO true

END IF

END WHILE

**Place random ships**

Ships should be placed from largest to smallest

WHILE shipPlaced = FALSE

Generate 2 random numbers xy, from 0-9

SET boardArray[x][y] TO ships origin square

Store coordinates of ship

Generate random direction, 0-3

FOR length of ship

Try placing next ship square

IF error

DELETE all stored coordinates for current ship

ELSE

shipPlaced = true

store coordinates of ship

END FOR

move onto next ship