

**COMP20300**  
**Java Programming (Mixed Delivery)**  
**2019/2020**

**(Java Project)**

**Battleships Game**

**Mohammad Asem Soufi**

Student No: 17210556

## Problem Description

Battleships is a strategy type guessing game for two players. It is played on a computer-generated grids boards on which a fleet of ships is marked randomly by the computer. The locations of the ships are concealed from both players. Players alternate turns calling "shots" at the hidden ships, and the objective of the game is to destroy the fleet by one the two players before the other.

## Game logic

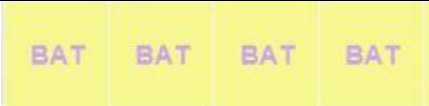

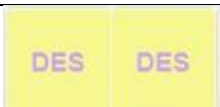
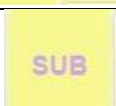
The grids board is a two-dimensional 10x10 grid with 100 available cells/buttons.

An array of all available cells is created:

{A1, B1, C1, D1, ..., A2, B2, C2, ..., F10, J10}

Then each value in the array is assigned to the corresponding Button ID on the grids board.

A ship is comprised of a one-dimensional sequential line of cells that can be placed/deployed by selecting a number of neighboring cells, either vertically or horizontally, and that number is specific to the type of ship being deployed.

Ship Type	Ship Size	Sample	Number of Ships
Battleship	4		2
Cruiser	3		3
Destroyer	2		4
Submarine	1		5

The game decides how to place ships on the grids by selecting cells and their orientation randomly.

(orientation is not considered when deploying a Submarine because it's only one cell long).

The game picks a ship of the list of ships that needs to be deployed, then first goes through a randomization processes, (but with a fair 50/50 chance) to select orientation (vertical vs horizontal).

Then it randomly selects the right number of cells for the current ship at hand, based on the size of the ship, and checks if:

1. These cells are available and were not assigned before to another ship, and,
2. That these cells are actually on the same row (if being placed horizontally), or on the same column (if being placed vertically).

Otherwise, it will keep trying by picking another set of cells until successfully deploying that ship, then moves to deploy the following ship on the list until all ships are deployed successfully.

A sample map of a deployed grids board.



Successfully deployed/assigned cells are incrementally added, as soon as they are deployed, to an ArrayList called **targetCells**, then each player is given his own copy named **myTargetCells** to be used to check if he made a hit while playing.

1. In case of a Miss:

1. A variable called **missCounter** for that player is increased by one
2. Check if the other player is still in the game
  - If so, Turn is switched and **activePlayer** points to the other player
  - Else, keep playing because you are playing solo

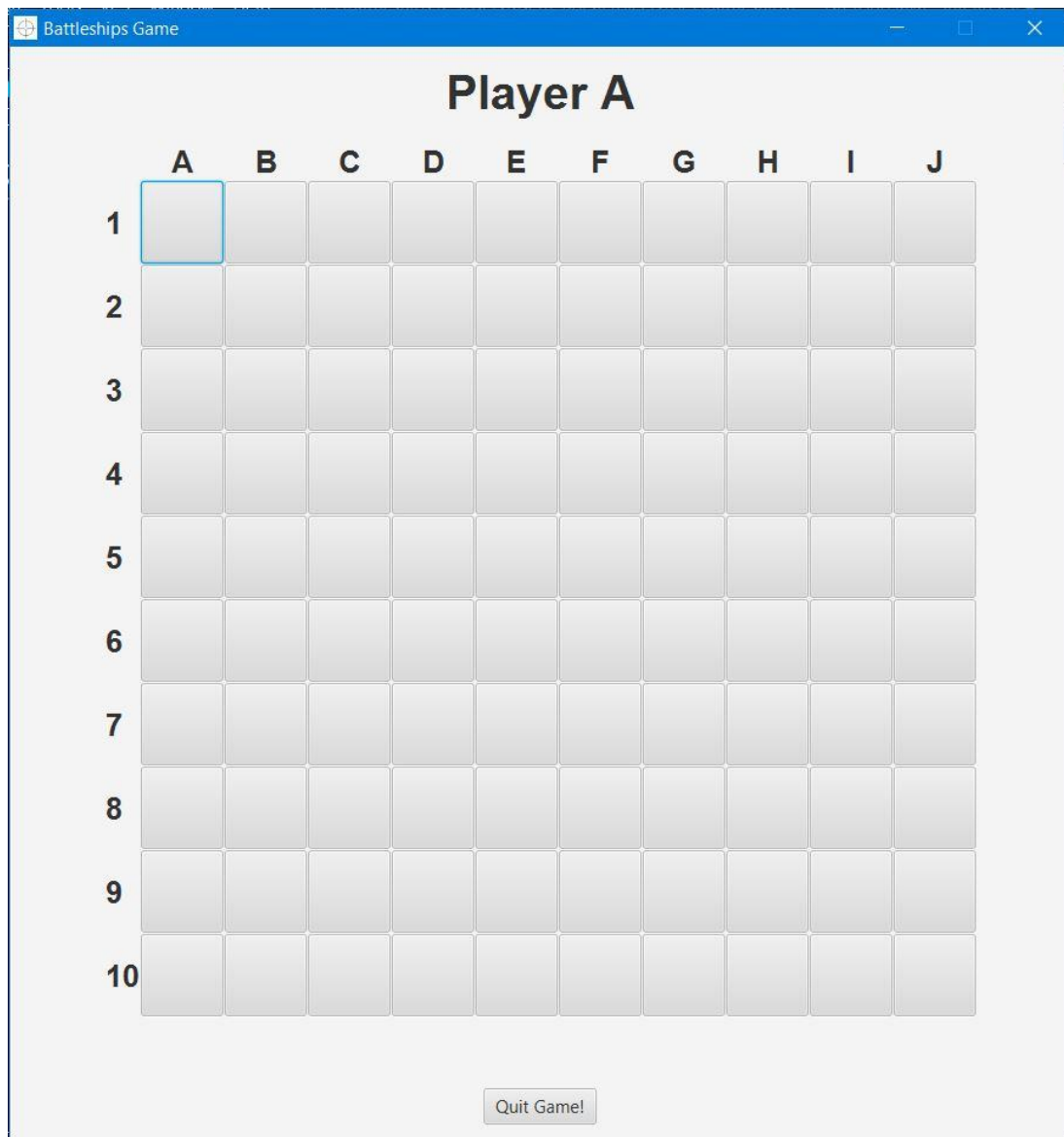
2. In case of a Hit:

1. A variable called **hitCounter** for that player is increased by one
2. The hit value (e.g. F6) is removed from **myTargetCells** list
3. **activePlayer** keeps pointing to the same player who made the hit
4. Check if Game is Over, (i.e. no more items in **myTargetCells**)
  - If so:
    - Celebrate the winner by showing a message on his grids board
    - Deactivate his grids board buttons to stop him from clicking any more buttons
    - Show the **Show Original Map** button to give him the option of seeing how the game was originally deployed in a separate window (Stage) and compare it to his own board to see how he performed
  - Otherwise:
    - Keep playing by clicking another button

To have a fair game, ships are placed, but hidden, on exactly the same locations/cells for both players, but each player gets his own grids board copy rather than aiming on the same grids board, this way one player's hit/misses are not counted for, or shown to, the other player.



Only one player can play at a time, and the active player keeps playing as long as he is hitting a target, and switching turns only happens when the current/active player makes a shot but misses to hit any possible target.

A sample of a player's own grid board with deployed cells hidden while playing.



A shot is made by clicking a button/cell on his grid and if that cell (Button ID) happens to be part of a deployed ship, it is marked as hit by displaying the word HIT with a Green background on that particular button, otherwise it will be marked as a miss by displaying the letter X with a Red background on that particular button as shown below.

Also that button will be deactivated to prevent clicking it again.

A button marked as a hit	A button marked as a miss
	

I use a third Player object called **activePlayer**, this player is solely used to keep track of who's current turn to play (i.e. Player A or Player B).

Any player can quit the game at any point by clicking the **Quit Game** button, but playing the game can only continue as long as at least one player keeps playing and doesn't quit the game, and until all targets had been successfully hit.

Another Player object is used and called **winner** to point to the winner of the game (either Player A or Player B), that is if there was a winner.

The winner's name and score is then recorded in the game's score board, basically a table in a MySQL database.

Game number is just used as a unique index and is filled automatically by MySQL along with the Date.

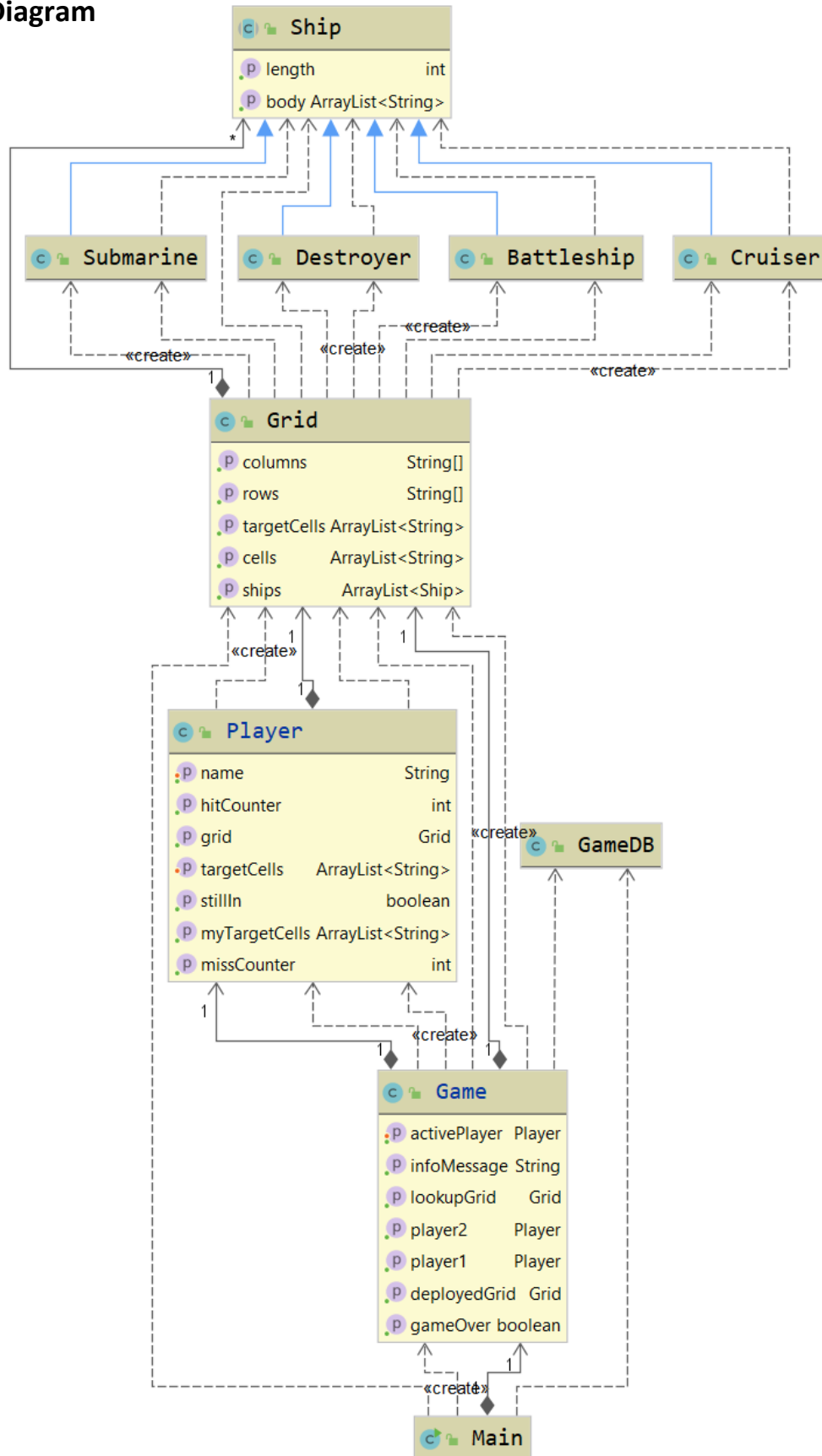
A sample score board.

Battleships Score Board				
Game	Player	Score	Date	
1	Jack Sparrow	100%	2019-11-28 00:35:40	
2	Asem Soufi	65%	2019-11-28 00:35:40	
15	Black Pirate	45%	2019-11-28 23:11:53	
3	Lloyd Christmas	40%	2019-11-28 00:35:40	
19	Player B	37%	2019-11-29 12:59:03	
20	Player A	35%	2019-11-29 13:07:50	
4	Joey Tribbiani	30%	2019-11-28 15:19:55	

If both players quit the game before winning, the game ends without having a winner and a score will not be recorded.

A score is basically the ratio of (Successful Hits) to (Total Shots made) formatted as a percentage. (i.e. between 30%-100%).

## Class Diagram



## Class design choices

- **Class Main**

Handles GUI objects and starts a new game.

On the GUI side, I use 3 stages:

1. **primaryStage**

Handles all GUI objects that are related to playing the game

2. **originalStage**

Handles all GUI objects that are related to showing the game map as created when the game was initiated

3. **scoreBoardStage**

Only shows a table view of the scores table in the database sorted by Score

- **Class GameDB**

Handles MySQL database connection

- **Class Player**

Handles players and their information:

- Names
- Are they still playing or did they quit
- Their copies of the playing grid and target cells
- Number of shots(hits and misses) they make

- **Class Ship**

The parent class of all ships and sets the blueprint for building a ship

- **Class Battleship, Cruiser, Destroyer, and Submarine**

Inherit Ship class and create objects of the specific type based on size

- **Class Grid**

Handles building all grids boards and randomly deploying ships

- **Class Game** (game engine)

Handles setting-up a new game, with all required non-GUI objects, and game logic



## Unit Testing

I test all classes except **Main** the GUI class, and **GameDB** which is straight forward and already has sufficient exception handling.

The game is built around Arrays and ArrayLists, that's why most of my unit testing targets primarily making sure all are created and searched correctly.

### Unit Testing Coverage Summary for Project

Project	Class, %	Method, %	Line, %
Battleships	88.2% (15/ 17)	61.1% (58/ 95)	38.9% (244/ 628)

<u>Class</u>	<u>Class, %</u>	<u>Method, %</u>	<u>Line, %</u>
Battleship	100% (1/ 1)	100% (2/ 2)	83.3% (5/ 6)
BattleshipTest	100% (1/ 1)	100% (2/ 2)	100% (6/ 6)
Cruiser	100% (1/ 1)	100% (2/ 2)	83.3% (5/ 6)
CruiserTest	100% (1/ 1)	100% (2/ 2)	100% (6/ 6)
Destroyer	100% (1/ 1)	100% (2/ 2)	83.3% (5/ 6)
DestroyerTest	100% (1/ 1)	100% (2/ 2)	100% (6/ 6)
Game	100% (1/ 1)	64.3% (9/ 14)	29.9% (26/ 87)
GameTest	100% (1/ 1)	100% (6/ 6)	91.7% (22/ 24)
Grid	100% (1/ 1)	71.4% (10/ 14)	74.2% (98/ 132)
GridTest	100% (1/ 1)	100% (4/ 4)	100% (14/ 14)
Main	0% (0/ 1)	0% (0/ 19)	0% (0/ 256)
Player	100% (1/ 1)	46.2% (6/ 13)	65.5% (19/ 29)
PlayerTest	100% (1/ 1)	100% (3/ 3)	64.7% (11/ 17)
Ship	100% (1/ 1)	100% (4/ 4)	90.9% (10/ 11)
Submarine	100% (1/ 1)	100% (2/ 2)	83.3% (5/ 6)
SubmarineTest	100% (1/ 1)	100% (2/ 2)	100% (6/ 6)
GameDB	0% (0/ 1)	0% (0/ 2)	0% (0/ 10)