Description

The game involves two grids (self and attack).

Battleship is a guessing game between player and computer.

The game itself involves two grids positioned over locations in the ocean. Player may only see the ships that are in their own gird and their own attacks on the opponent's screen.

Player and Computer take turns launching missiles into individual grid location with the goal of sinking the opponent's ships.

When a missile is launched, the player is told whether the missile "**HIT" or "MISS".**

The game also says on a "hit" if the ship was "**SUNK",** meaning that all the location the ships occupies have been hit.

The game is won when all location that the enemy's ships have been "**HIT".**

Software Requirement

- **Operating System**
- **Java SDK or JRE 1.6 or higher**
- **IntelliJ IDEA Ultimate 2019.2.3**

REQUIREMENT

- 1 player
- Player has 10X10 board to place the ship.
- Player will be given 5 ships that 2,3,3,4,5 units long and 1 unit wide.
- Ships can be oriented vertically or horizontally.
- Player sees 2 grids-their own with ships placed and their opponent's grid with Record of where they have been attacked.
- Player and Computer takes turn attacking the opponent's ships.
- The result of an attack can be one of:
  -A "**HIT**" if the opponent has a ship covering the co-ordinate
  -A "**Miss**" if there is no ship covering the co-ordinate
  -"**SUNK"** if all co-ordinate a ship covers have been hit
  -"**WIN"** if all ships on the opponent's board have been sunk.

The game ends when all a player’s ship have been sunk.

FUNCTIONAL REQUIREMENTS

End user:

Start Game: The system should be capable of starting a new game when the user selects new game.

Position ships: The system should allow the user to position their ships wherever they like, and warn them when they are not allowed place a ship a certain direction or on a certain button.

Attack: The system should return the status of an attack on a JButton and indicate whether or not it was a hit.

Sink ship: The system should be able to tell when a ship has been sunk and inform the user.

Sunk all ships: Once all the ships have been sunk on one side of the board, the system should output win/loss information to the user.

System:

-The system should be able to randomly place the computer players ships so as none are overlapping and all are within the bounds of the array.

-The system should be able to count and keep track of all attacks made by both players and also the ratio of successful hits to attacks.

-The system should be able to tell when a ship has been hit.

-The system should be able to tell when win/loss point in the game has been reached.

Game Progress:

Have a **Game object.** The game object controls **the flow of the game** and meshes all the other resources to work cohesively.

Creating a **Board object**, that **holds an array of the Ships a player** has. It also has **a 2D array.**

Entity objects. These objects contain **ship data.** Several Entity objects together form a ship. They're given a unique ID such as, Battleship-1. These objects can also contain no data, which would be empty squares. The Entity objects are also stored as...

Ship objects. These **objects are stored in an array,** in **the Ship class**. The Ship itself contains an array of all the pieces that make up a ship. When an Entity is hit, and it is a **real part of a Ship**, a Boolean, such as is_destroyed toggles from false to true. When all the Entity objects that make up a ship are destroyed, the Ship is marked as destroyed overall.

The Game object will initialize the game, initialize two Board objects for the player and computer, and the Board objects will allocate the correct number and type of Ship objects in each board.

Entering the setup phase, players place their ships on the board. These modifications are made to the 2D array Board has.

Ending the setup phase and beginning the game phase, players take turns choosing a square, and the chosen square responds based on what type of Entity is in that square.

Empty Entity objects will report MISS. Ship Entity objects will report HIT. Control passes to the next player, and the same thing will happen. Loop. Once the game detects a player has lost every ship they own, display a Victory message.

1. A starter code has been provided in BattleShip.zip under files. It consists of the following files
2. **a. Battleship-**this is the main class. It creates the Player Screens.
   b. Grid-Represents the screen of a player.
   Implements the grid-layout used for self and attack grids.
   Shows player's own position of ship
   Shows attack positions and results.
   c. Randomizer- Randomly places the ships on the grid for the computer.
   d. Player- Class handles the ships placed by the player.
   e. Location- Show attack position and results

Setting up the game

Initially the game starts with just Player1's grid. The player selects the grid location Columns arranged from A to J and Row arranged from 0 to 10 and provides a valid input upon ship of that length will be placed.

Player must make sure that there are no overlaps.

For example, if there is a ship at (A,0) covering (A,0),(A,1),(A,2) the user cannot choose (A,1),(A,2) for the next ship.

Once the player has setup the game , the game begins.

Playing: when it's a player's turn, he used the attack grid to aim a missile .

Upon sinking the ship or winning the match, player will be displayed with the message ship has been "**HIT"** and that location will be marked with an x on the player attack grid.

"o" is used for the position that has been missed.

The orientation of the ships must be either horizontal or vertical (no diagonals). The coordinate system is the normal computer science system: (0,0) is at the upper left, with row number increasing downwards and column number increasing to the right. The two endpoints will occur so that the smaller row or column coordinate is always first.

To place a ship horizontally Number zero shall be given as input

To place a ship vertically Number ones shall be given as input

If any errors occur in the data used by the game upon startup, an error message is printed on standard error and the program should gracefully terminate (again). Note that these are the only error messages that should be displayed on standard error, and none should be displayed on standard out; all other messages generated while the game is being played should be displayed on the appropriate output view.

• Specification of the ship on line *n* causes it to overlap with a ship described on an earlier line. • Specification of the ship on line *n* extends beyond the boundaries of the board.	`Illegal ship specification in configuration file at line` *n*.

• Specification of the ship on line *n* is neither horizontal nor vertical.	

Selecting a game board square	`Symbolically fires a missile at the chosen square.`
	`The status of the square is  changed depending on if it is a hit or a miss. The hit on the last remaining section of a ship changes the way all of the sections of the ship are shown in the cells; it has been sunk.`
	`If a missile was already sent there, or if the game is over, does not modify the game state or the board view at all.`
	`If the missile hits a part of the sea, increment the number of tries.`
	`If the missile hits a part of a ship, increment the number of tries and the number of hits.`
	`If the missile hits a part of a ship causing it to sink, display a message to that effect.`
	`If at this point all the ships are sunk, the game will be over, and a message to that effect should be displayed.`

After every attack user placed ship will be displayed

The game finished when all ships of the computer are sunk.

When either player has no ships remaining (i.e., all of that player's ships have been sunk), that player declares victory, and the game is over.

Game further update

Add a status region to the Players Screen to show the following.

a. Number of own ships
b. Number of own ships sunk
c. Number of enemy ships sunk.
d. Current State
e. Game Over (Result)

Error messages that may occur due to problems with setup (processing the arguments, reading input, etc.) and cause the program to terminate early shall be displayed.

Game statistics shall be kept throughout the game, including keeping track of the number of hits, the total number of "tries" (hits + misses), the number of complete ships sunk.

The user should be able to save the game if they decide to quit in the middle of the game. This will save all their scores and progress of the game.