**Battleship Game**

**Date: 18/11/2019**

**Student Names:** Miklos Mayer

**Statement of requirements**

Allow the user to play against the computer in a game of Battleships. This game is played on a 10x10 grid on which the computer will randomly generate a battle fleet which will consist of 9 ships. The player will select a square to try and locate the computers battle fleet.

The user does not need an in-depth knowledge on computing systems but will need basic knowledge on how to use a desktop computer. The user should *know how to play a basic game of battleships in order to use the program.*

*The inputs of the program are commands typed in on a keyboard.*

The outputs of the program are the battle fleet position, the players score and whether a fire was a hit or a miss.

**Requirements**

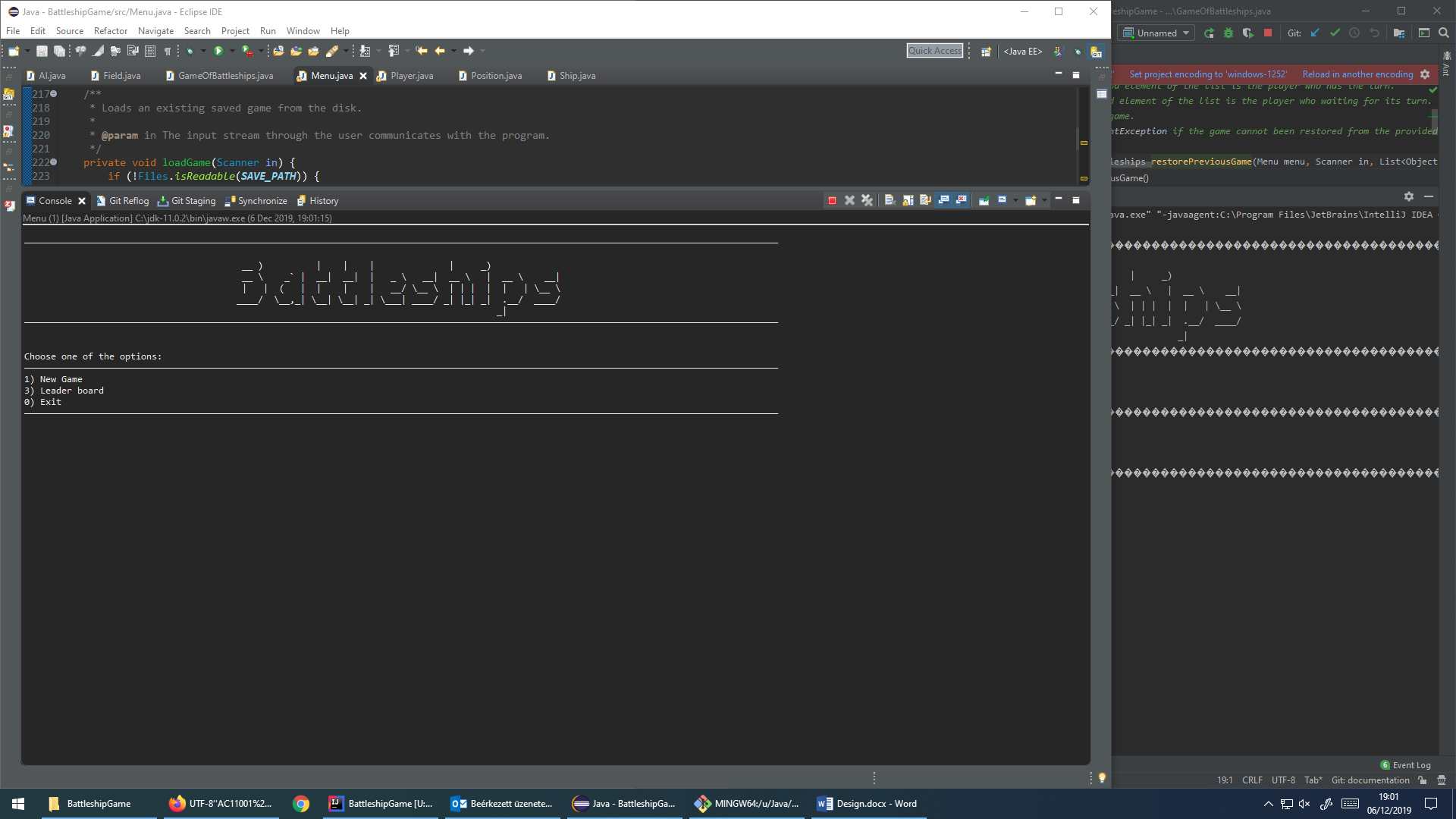
**Functional Requirement**

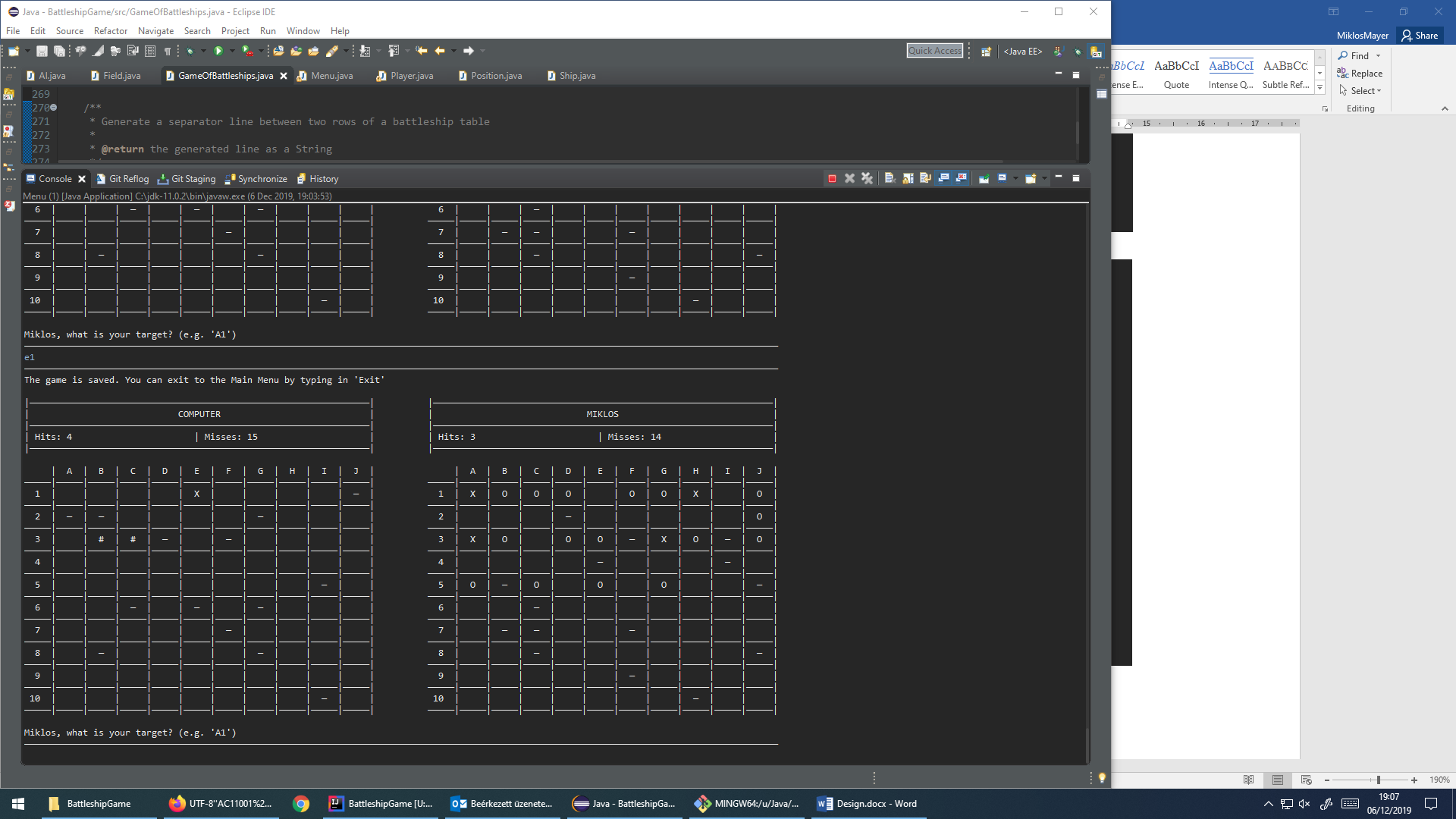
1. The system shall open with a Main Menu.
   1. This page allows the user to select the starting functions of the program.
2. The system shall contain a Start New Game option.
   1. This function is in the Main Menu. It will initiate a completely new game and activated a valid keyboard command.
3. The game is played by two players, one of them is the user, the other one is the computer called AI.
4. The system shall display two 10 x 10 square grid as the Battlefield.
   1. The x axis is labelled by the alphabet from A to J. The y axis is labelled by ascending numbers from 1 to 10.
5. The system shall display the players’ hits and misses.
6. The user shall enter a name for themselves when the game starts
7. The AI shall randomly place its fleet on the Battlefield.
   1. The ships shall not overlap each other, and they shall not touch each other (even diagonally). They should not be placed diagonally.
8. The user shall place its fleet on the battlefield with the same constricting rules.
9. A fleet consists of one 4 long, two 3 long, three 2 long and four 1 long ships
10. The system shall allow the player to choose a square in attempt to locate the computer’s battleships.
11. The system shall prompt the user to choose another field if it is previously fired upon.
12. The system shall indicate whether the fire hit or missed a battleship.
13. The system should indicate if a ship sank.
14. If a fire hit the player shall have another turn.
15. The system shall save the current game after every fire.
16. The user should have an option to Exit the game at mid-play
17. The Main Menu shall have an option to Resume to the last unfinished game.
    1. It will reveal the playfield as in New Game but with the saved scores and positions.
18. The system should not display the Resume option if there is no saved game.
19. The game shall end when one of the player hit all the ships on the battlefield.
20. The system shall save the user to the Leaderboard if their score is in the top 10.
    1. The final score will be calculated from the hits, misses and passed rounds.
21. After the game ends, the system shall display the Main Menu again
22. The Main Menu should have an option to show the Leaderboard.
23. The system should show player scores ordered from highest to lowest

**Non-functional Requirements**

1. The Java Runtime Environment (JRE) shall be installed on the system.
2. The program shall be played by a keyboard.
3. This program cannot be played on a mobile device.
4. The program must not contain any violence or inappropriate content for children.

**User Interface**





**Use Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| New Game | | | Alternatives |
| 1 | USER | Choose new game |  |
| 2 | SYSTEM | Launches game |  |
| 3 | SYSTEM | Randomly place the AI battlefleet on a 10x10 grid |  |
| 4 | SYSTEM | Display grid on screen |  |
| 5 | SYSTEM | Ask the User to place their ships |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Firing a Shot | | | Alternatives |
| 1 | USER | Choose a square | A1, A2, A3 |
| 2 | SYSTEM | Determine whether a battleship was situated on the square that the user chose |  |
| 3 | SYSTEM | Display whether the fire hit, sank or missed a battleship |  |
| 4 | SYSTEM | Increment and display ‘Missed’ or Hit’ score |  |
| 5 | SYSTEM | If missed increment Round and switch players |  |
| 6 | SYSTEM | If all the ships found display ‘Final score’ |  |

|  |  |  |  |
| --- | --- | --- | --- |
| A1 Firing a Shot alternative | | | Alternatives |
| 1 | USER | Choose a square previously fired on |  |
| 2 | SYSTEM | Registers that the square has been previously selected |  |
| 3 | SYSTEM | Ask user to choose a different square |  |

|  |  |  |  |
| --- | --- | --- | --- |
| A2 Firing a Shot alternative | | | Alternatives |
| 1 | USER | Choose not a square |  |
| 2 | SYSTEM | Registers that the user did nor chose a square |  |
| 3 | SYSTEM | Ask user to choose a different square |  |

|  |  |  |  |
| --- | --- | --- | --- |
| A3 Firing a Shot alternative | | | Alternatives |
| 1 | USER | Type in ‘Exit’ |  |
| 2 | SYSTEM | Saves the game |  |
| 3 | SYSTEM | Return to Main Menu |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Resume Game | | | Alternatives |
| 1 | USER | Choose ‘Resume Game’ |  |
| 2 | SYSTEM | Access previously saved game |  |
| 3 | SYSTEM | Display game on screen |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Show Leaderboard | | | Alternatives |
| 1 | USER | Choose ‘Show Leaderboard’ |  |
| 2 | SYSTEM | Show the leaderboard |  |

**Classes**

**Candidate Classes**

|  |  |  |
| --- | --- | --- |
| **Candidate Classes** | **Accept / Reject** | **Reason for rejection** |
| Game of Battleships | Accept | Handle the game functions |
| Position | Reject | Better suited to be a field in the Ship class |
| Player | Accept | Contains information about a player |
| Username | Reject | Field of Player |
| Ship | Accept | Contains information about a ship |
| Battleship | Reject | Just the type of the Ship |
| Cruiser | Reject | Just the type of the Ship |
| Destroyer | Reject | Just the type of the Ship |
| Submarine | Reject | Just the type of the Ship |
| Menu | Accept | Top level coordinator |
| Option | Reject | Too vague |
| New Game | Reject | newGame could be a method in the Game of Battleships class |
| Shot | Reject | fireShot could be a method in the Game of Battleships class |
| Grid | Reject | A list of squares |
| Field | Accept | A field of the Battlefield |
| Fleet | Reject | List of Ships |
| Score | Reject | Better suited to be a field of Player |
| Leaderboard | Reject | List of players stored in a file |

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

Menu – Handle all the interactions between the user(s) and the computer

- Fields: game, hasSavedGame

- Methods: main(), newGame(), saveGame(), deleteSavedGame(), loadGame(), showLeaderboard(), saveScore()

GameOfBattleships – Handle all the interactions between the players

- Fields: menu, in, player1, player2, rounds, passivePlayer

- Methods: play(), displayGrids(), getRounds(), switchPlayers()

Player - Stores information about a player who interact with the game

- Fields: hits, misses, fleet, battleField

- Methods: calculateScore(), takeFire(), getHits(), getMisses(), placeShips(), fire()

AI – children of the Player class, controlled by the computer

- Additional/overriden methods: placeShips(), fire()

Field – One field of the battlefield. Handle hits and store information about fires and ships

- Fields: position, hasShip, isFired, isSank, shipId

- Methods: hasShip(), isFired(), isSank(), takeFire(), placeShip()

Position – Helper class for Field, ensures that fields can only have valid positions (e.g. A4)

- Fields: position

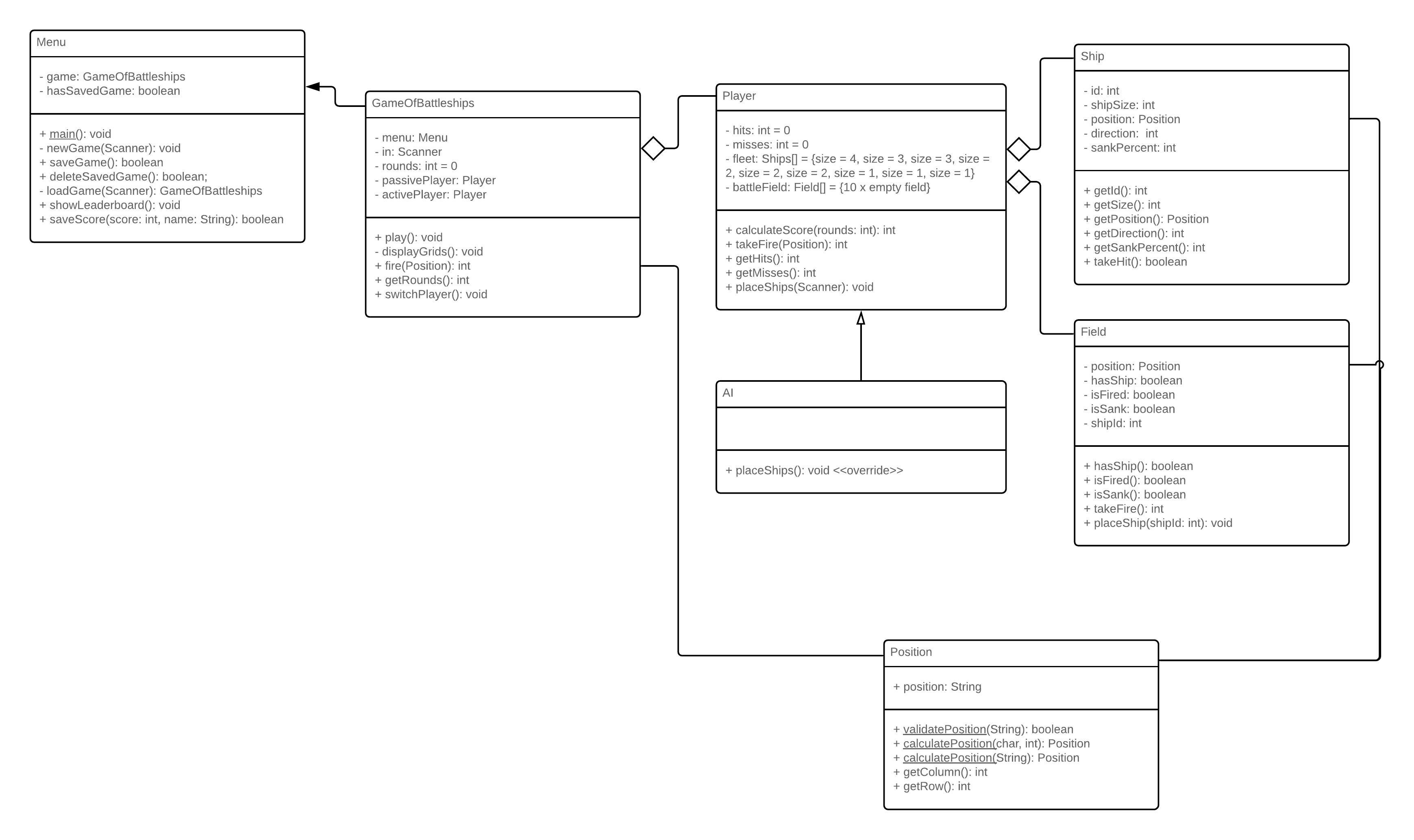
- Methods: validatePosition(), calculatePosition(),getColumn(), getRow()

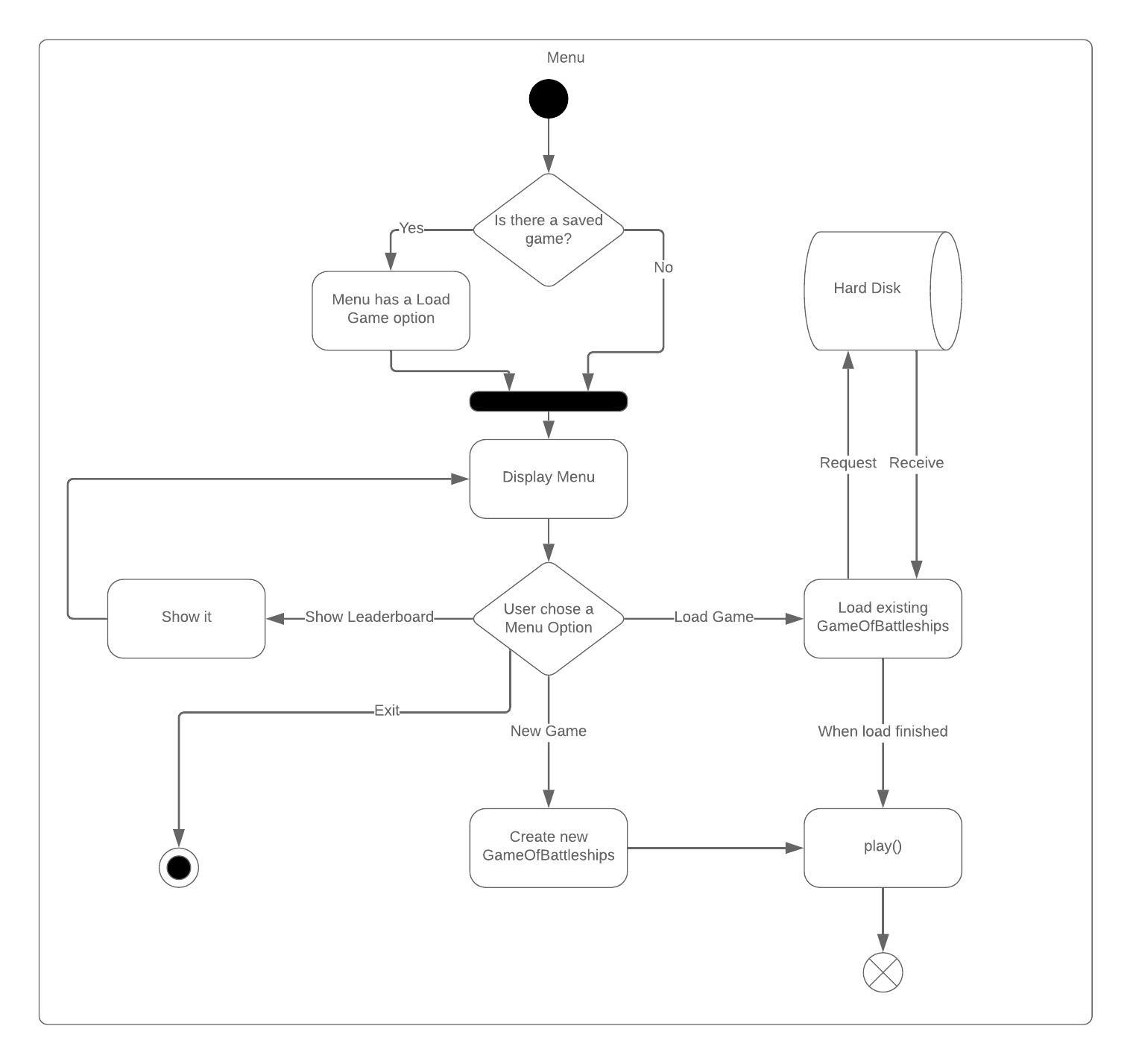
Ship – contains information about a ship

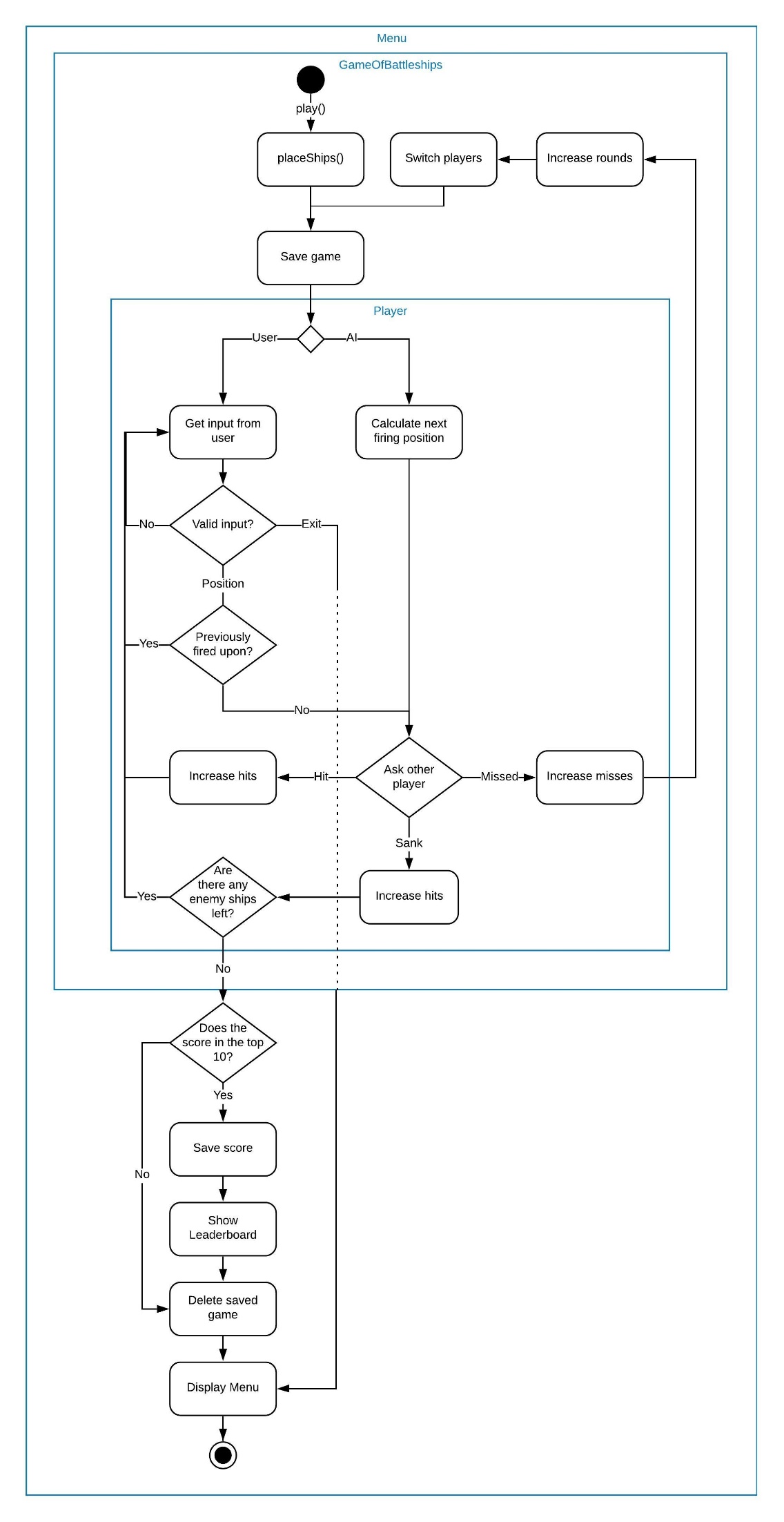
- Fields: id, shipSize, position, direction, sankPercent

- Methods: getSize(), getDirection(), getSankPercent(), getPosition(), takeHit(), getId()

t

**Class Diagram**

**Activity Diagrams / Pseudocode**



**Self Evaluation**

I took the advice of the teacher and started with the Class Designs and Activity Diagram. It was quite useful because I barely had to refactor my code during the development process. I left this early design in this document for reference and I modified it only slightly during writing my code.

Of course the final game is different in some aspects. I have new methods compared to the Class Design or don’t have them if they were not used.

But the program flow is the same as I planned on the Activity Diagram. This was the first I programmed, only with placeholder outputs and inputs with error handling. I have an empty game without any developed game logic, but I could navigate from the Main Menu to the end of the game. I did not have to modify this flow during the development and it prevented any future refactoring apart from debugging the logical errors inside my methods.