Battle Ship Game

SOEN 6441: Advanced Programming Practices - Summer 2019

Software Architecture Document

<Build 1>

Team K4

Yilian zhao	40050629
Xiabing Cui	40049961
Divya Pandit	40087471
Jaspinder Kaur	40059831

1. Introduction:

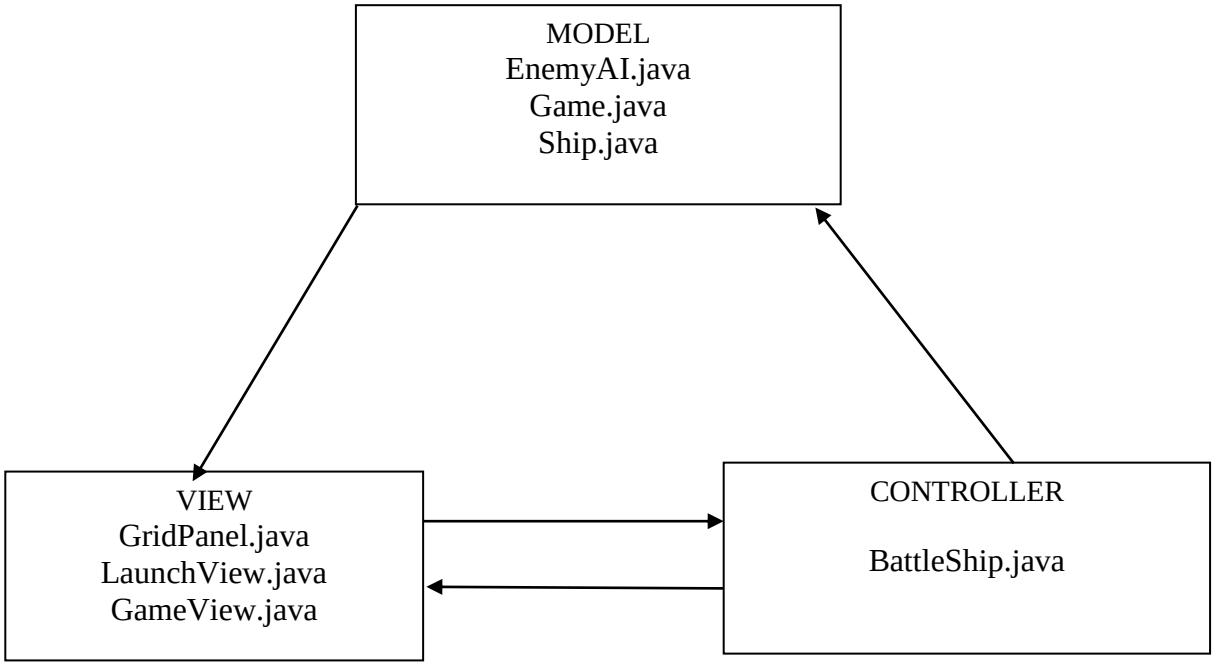
Battleship game is a two player game. In this game each player will have battleships which are placed on ruled grid and these ships and location of ships are hidden from each other. Each player will have alternative turns and goal of player is to hit all battleships first. The player who hits all ships first is winner of game.

We are preparing a Desktop based application with is based on Java. We are using Javafx for GUI.

In following sections we have discussed architectural representation,

2. Architectural representation:

We have used MVC architecture in which Our Controller class is BattleShip.java. Our Model classes are EnemyAI.java, Game.java, Ship.java. Our View classes are GridPanel.java LaunchView.java, GameView.java



2.1. Process view:

To provide a basis for understanding the process organization of the system, an architectural view called the **process view** is used in the Analysis & Design discipline. [2]

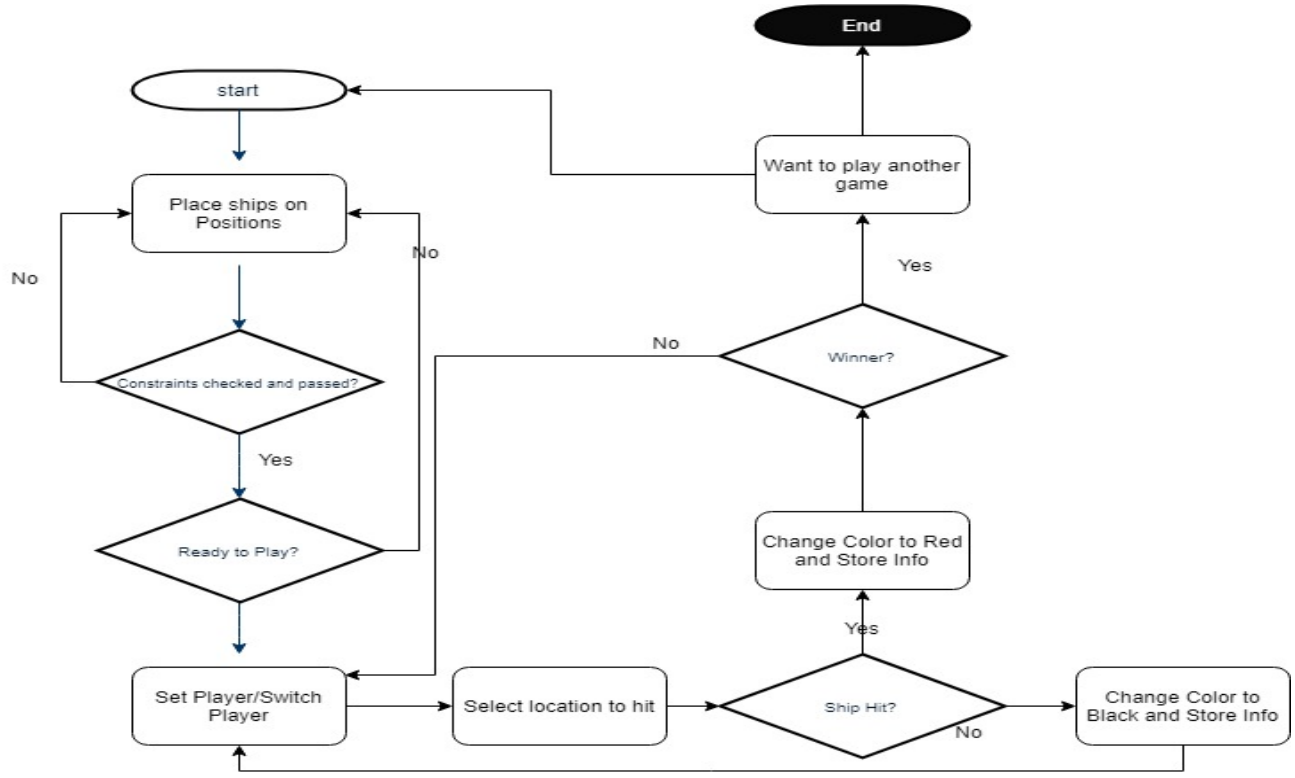


Figure 1: Progress View Diagram

2.2. Use case view:

In the diagram below we are showing interactions of player and system. It gives us internal view of the system. Here player1, player2 and System are three actors. Below is the UML diagram for our project.

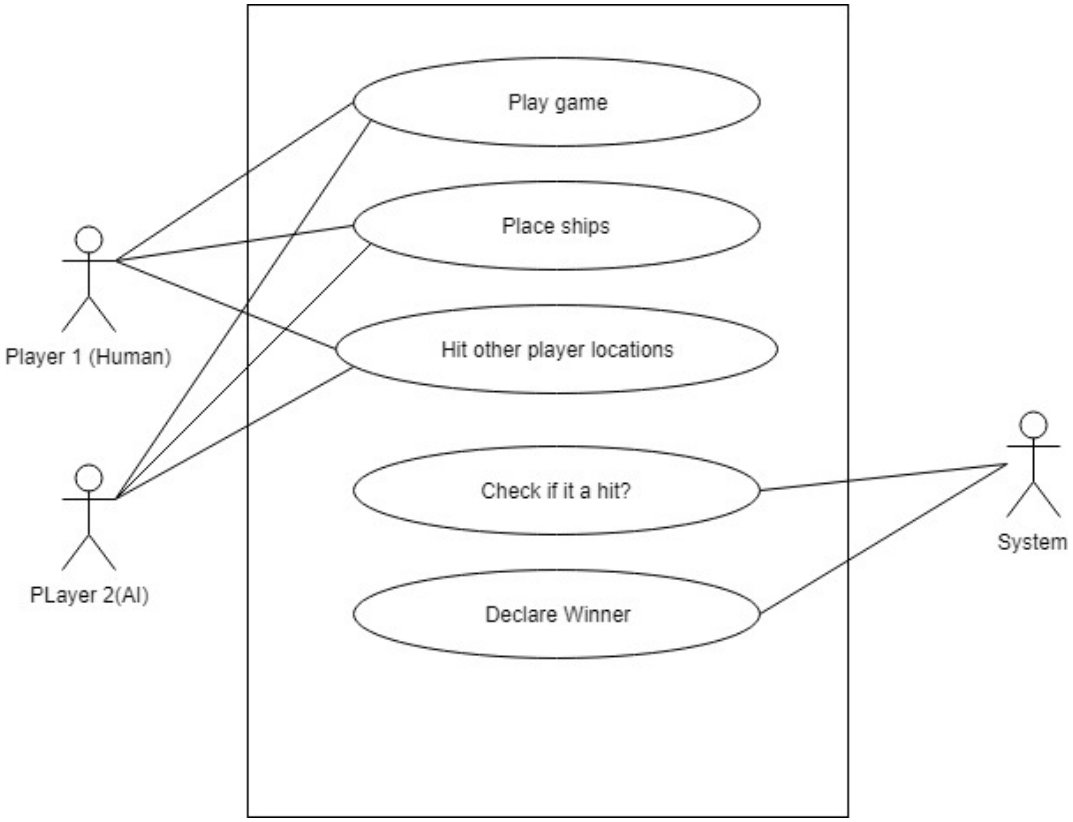


Figure 2: Use Case Diagram

3. Functional Requirements:

'The functional requirement describes, "How it should work". [1]

Functional requirements are those which we actually want our system to do in normal conditions. Functional requirements are subtype of user requirements. Below is detailed functional requirement [1]

Functional	**Description**

Requirement	
Start Game	The player should be able to start new game.
Place ships	The player should be able to position the Battleships on the grid
Attack	The player should be able to attack opponent's Battleship.
Result	System should be able to announce the winner who first destroy all ships
Placement of ships (AI)	Player 2 i.e. AI should be able to place ships randomly on grid
Constraints for Placement of ships	Both AI and player should be allowed to place ships according to constraints only.

4. **Non-Functional Requirements (Quality Attributes):**

Non functional requirements are criteria that judge the operation of system rather than specific behavior. These are majorly quality attributes defined by users for system. [1]

The non-functional requirements of the system are the following:

- Performance requirements: AI and system respond in real-time.
- Platform constraints: The system is platform independent for Desktop based systems as it runs on Java.
- Modifiability: System uses MVC model which makes system highly modifiable for later stages.
- Reliability: System is reliable and proper tests are carried out to check its reliability.
- Usability: GUI of system is highly user friendly with Help option for Users.

5. **Tools and Technologies used in the project:**

Name	Description
Eclipse	Interactive Development Environment
Java	Programming language

Java Docs	For API Documentation
Draw.io	For UML Diagrams
MS Word	For Documentation

6. Implementation:

BattleShip.java is the Initial Start of the game i.e. it will call the launchview.java class

LaunchView.java creates the start view ,when you proceed the player gets two options:

1)If the player clicks the help button, it will show a message telling player how to play it.

2)If the player clicks the start button, and the Game.java class will be called and the player start playing the game

GameView.java will be called and it will create complete game layout.

Ship.java is called to support the placement of ships.

GridPanel.java it is called because it handles mouse events for opponent and player 1 and it also checks the placement validity for ships.

EnemyAI.java is called after the player (human) has placed the ships already, the game

Cell.java is called to change the color of the grid for hit for miss.

7. API Documentation

- For API documentation we have Summarized the tasks carried out in a particular class.
- We have defined different parameter used and values returned in class with @param and @return.
- We have also mentioned throws exception for classes with @ throws comment
- Finally we generated html documents for all classes.

8. Resources:

[1] U. Eriksson, “Functional Requirements vs Non Functional Requirements ,” *ReQtest*, 29-Oct-2018. [Online]. Available: https://reqtest.com/requirements-blog/functional-vs-non-

functional-requirements/. [Accessed: 12-Jul-2019].

[2] J. C. Helm, “Rational Unified Process,” *Concepts: Process View*. [Online]. Available: https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/process/workflow/ana_desi/co_pview.htm. [Accessed: 12-Jul-2019].