

**Battle Ship Game**

SOEN 6441: Advanced Programming Practices **-** Summer 2019

Software Architecture Document

<Build 2>

**Submitted to:** Professor Nagi Basha

**Prepared by:**

|  |  |
| --- | --- |
| Sukhpreet Singh Bhatia | 40083564 |
| Pulkit Ghai | 40089798 |
| Avinash Damodaran | 40078258 |
| Showan Haddadi | 40074816 |
| Marjan Baghdadi | 26980198 |

1. **Abstract:**

Battleship game is a classic guessing game traditionally designed for two players. It is played on ruled grids on which each player’s fleet of ships (including battleships) are marked. The locations of the fleet are hidden from the other player. Players alternate turns in the game after each guess and the goal of the game is to destroy the opposing player’s fleet. [1]

During three builds in this term, our team will work on a desktop application including all the required source codes for the back end and front end of this game. We will present our progress on the logic behind the game (AI) written in java, as well as an interactive user interface (GUI) written in javaFX in each different milestone.

In build 1 we have implemented all the necessary infrastructure of the game including the GUI, and a simple AI. However, in build 2 we have improved all the components specifically the AI part which needed much more attention after the first build. We have improved the GUI and implemented the Drag and Drop style for putting the ships on the grid and made the game much more user friendly. The Salva variation has been added as an option. And the user is able to choose which way to play. The other additions of this build are the Timer as well as the Score system implementation

1. **Introduction:**

To implement our application as a team of developers, we use Git - a distributed version control system for tracking changes in source code - during the software development. In the following sections of the document, we discuss the Scope, Architectural Design, Assumptions, Goals and Constraints, Dependencies, Functional and Nonfunctional Requirements, Technologies used, Coding Standards, and finally Refactoring here.

1. **Architectural representation:**

Architecture of our design for build two is MVC architectural style. Our “Controller” classes are .java which contain the logics at our backend taking care of the AI of the application. Our “Model” or Data Base are coded with java as well storing the state of the players and are .java files. Our “View” are .fxml and .css files which are written in javaFx and Scene Builder which have formed the GUI of the game.

* 1. **Logical view:**

It supports the functional requirements, what exactly are the services the system should provide to its end users, in this phase we try to get better knowledge of problem by talking to domain experts, whatever decision we make here are independent of implementation decision. [2]

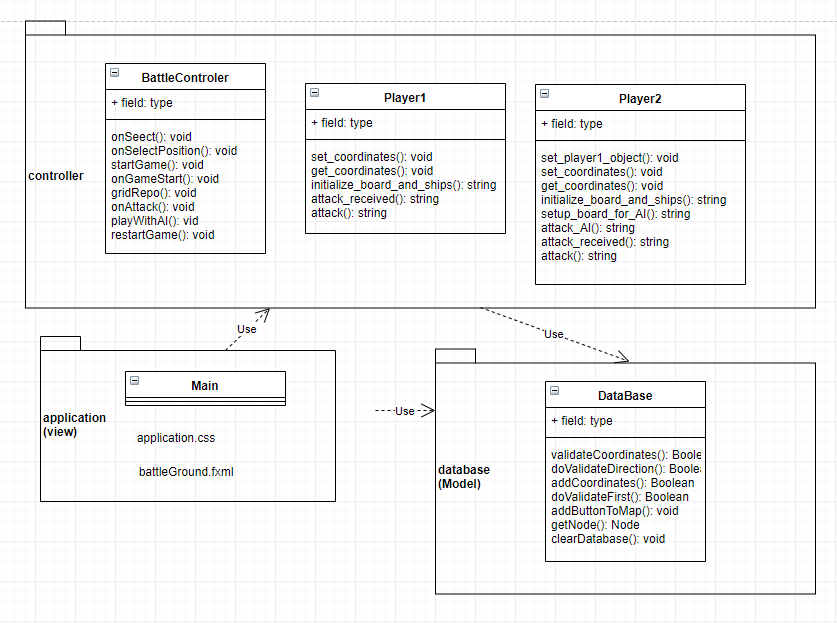


Figure 1: Package Diagram

* 1. **Process view:**

The process view focuses on what will happen during run-time and also on non-functional requirements such as performance and availability. It addresses issue of concurrency and distribution, fault-tolerance, process synchronization etc. It is illustrated using activity diagram. [2]

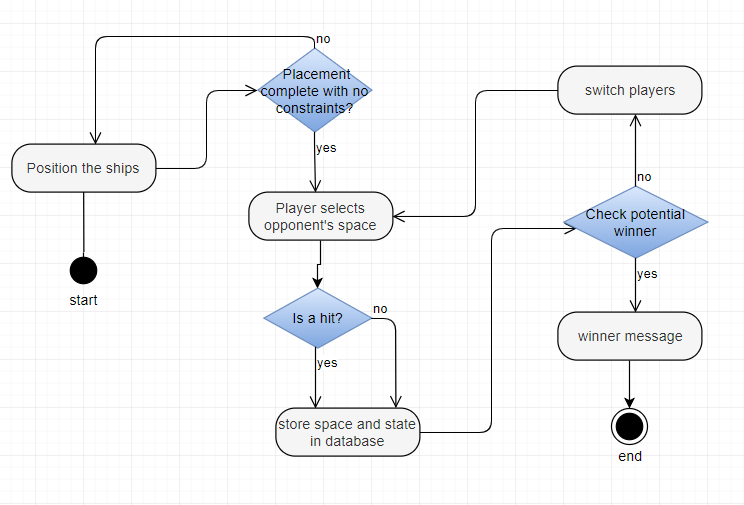


Figure 2: Activity Diagram

* 1. **Use case view:**

This view shows the interactions between the player and the system as a whole. It shows the main processes of the application. In this view, we use a set of scenarios to depict the description of the system architecture.

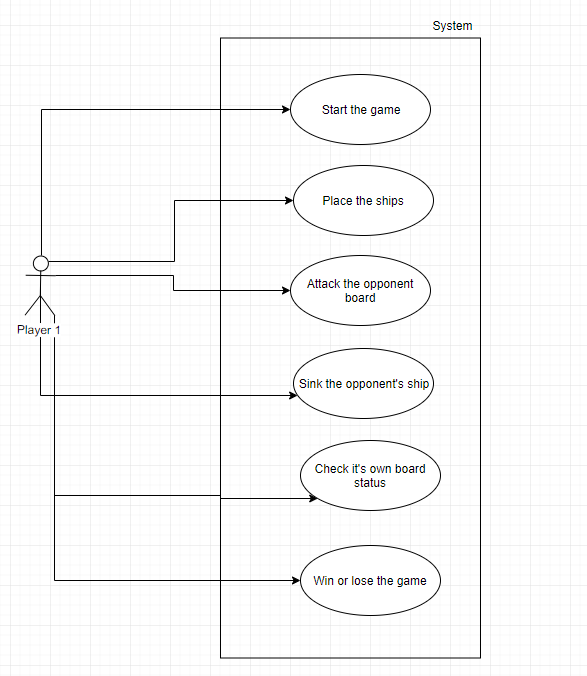


Figure 3: Use Case Diagram

1. **Functional Requirements:**

‘The functional requirement describes, “How it should work”. [3]

Functional requirements describe the desired end function of a system operating within normal parameters, so as to assure the design is adequate to make the desired product and the end product reaches its potential of the design in order to meet user expectations. ‘[4]

The functional requirements of this system are the following:

|  |  |  |
| --- | --- | --- |
| **Use Case Name** | **Actor** | **Description** |
| Start Game | Player | The player should be able to start a new game. |
| Position Ships | Player | The player should be able to position the fleet on the grid (considering the constraints) |
| Attack | Player | The player should be able to start attacking (hitting/missing) the opponent fleet |
| Sink a ship | Player | The player should be able to sink a ship by hitting all the right spots and receive an alert indicating he/she sank a ship |
| Win the game | Player | The player should be able to sink the whole opponent’s fleet by hitting all the right spots and receive an alert indicating he/she sank all of the ships and won the game. |
| Placement of ships | System | The system should be able to randomly select positions of the fleet for the AI player after each start/restart of the game |
| Respecting the constraints on placement of ships | System | The system should be able to only allow positioning of ships on acceptable positions and directions based on the rules of the game. |
| Showing alerts and messages | System | The system should show messages for each hit from the opponent, each ship sink from the opponent, indicating the winner of the game, etc.. |

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

‘In systems engineering and requirements engineering, a non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.’ [5]

The non-functional requirements of the system are the following: [2]

* Performance requirements: The game has to maintain an acceptable response and execution time suitable for the nature of game industry
* Operating constraints: on build 3 of this project we need 2 players and a reliable network system to run the game, however there won’t be much more operating constraints than the requirement of having at least one player available.
* Platform constraints: This game uses JavaFX for GUI which makes it capable of running on multiple platforms.
* Modifiability: This software is highly modifiable as we are using a VCS on git to let us be able to apply any required changes 24/7.
* Portability: Our game is highly portable as we are using JavaFX for the GUI which is suitable for multiple mobile operating systems, including android, windows mobile, and etc. ...
* Reliability: This game is highly reliable as we run multiple tests to catch any bugs at each build.
* Testability: This game will be tested efficiently and carefully by Junit starting from build 2.
* Usability: This system is highly user friendly as we have built an interactive and highly descriptive UI for it.

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

|  |  |
| --- | --- |
| **Name** | **Description** |
| Git (Github) | Distributed version control system used for tracking changes in the source code during software development |
| Eclipse | IDE used for the game development |
| JavaFX (and Scene Builder) | Software platform for creating and delivering the GUI. JavaFX is library for Java SE. |
| Java | Programming language used for the source codes |
| Junit | Unit testing framework for the java programming language. Will be used in Build 2 to create unit tests |
| UML | Unified Modeling Language developed by drawit.io |

1. **Implementation:**

**Main.java:** Main class file is used to launch the application by calling the fxml file and setting it up to the scene.

**BattleController.java:** Controller class is used to handle the action event of the user. On each key press or performing any functionality the associated the Action Event is called to carry out the respective function.

**Database.java:** Database is used here to storing the coordinates of the ship placed on the grid and do the validation of the coordinates.

**Player1.java:** Here we have implemented the functionality for the Player 1 i.e. is the for ship hit or miss checks by the AI

**Player2.java:** Here we have implemented the functionality for the Player 2 i.e. is the AI for ship hit or miss checks by the user

**battleground.fxml:** Complete GUI is built through scene builder and this is represented using fxml file which is in turn called in controller to handle the action listener.

1. **Coding Standards:**

We have used java coding standards in this project. [7]

1. **Refactoring:**

Between the first and second build of this project, we had to do several refactoring to improve the cleanness and quality of our code. Refactoring has been done on classes related to the AI, GUI, and database and coding standards, more relevant comments and etc. have been enforced in all the previous Implementation from the first build.

1. **Abbreviations:**

RUP: Rational Unified Process

UML: Unified Modeling Language

SAD: Software Architecture Document

GUI: Graphical User Interface

AI: Artificial Intelligence

UI: User Interface

VCS: Version Control System

NFR: Non-Functional Requirements

FR: Functional Requirements

MVC: Model-View-Controller

1. **Resources:**

[1] https://en.wikipedia.org/wiki/Battleship\_(game)

[2] http://users.csc.calpoly.edu/~jdalbey/SWE/QA/nonfunctional.html

[3] https://www.sciencedirect.com/topics/engineering/functional-requirement

[4] https://whatis.techtarget.com/definition/functional-requirements

[5] https://en.wikipedia.org/wiki/Non-functional\_requirement

[6] <https://www.academia.edu/23707562/Battleship_Software_feasibility_analysis_design_testing_and_debugging_Report>

[7] https://android.jlelse.eu/java-coding-standards-ee1687a82ec2