EECS 3311 - LAB5 REPORT - MINI SOCCER GAME

**PART I: INTRODUCTION**

**Goal**: This software project aims to provide a GUI for the user to interact and play a mini soccer game. The user is allowed to move a player (striker) with the arrow keys and shoot the ball to score goals within a time frame of 60 seconds. This project aims to combine the idea of pausing and resuming the soccer application in real-time. In addition, this project aims to help us understand and translate between UML Class diagrams and Java code.

**Challenges**: The challenges in this software project are completing the implementation of missing model classes and having to create tests and Javadoc for our code.

**Concepts Used**: OOD principles that we used are the Single Responsibility Principle, where each class has a specific purpose to perform. Secondly, we used the Open Close Principle, where classes are able to be extended but not modified, and thirdly, Liskov Substitution Principle. In this principle, the class that inherits from the parent class does not replicate the already performed functionality of methods. Design Patterns used for this software project are Decorator Pattern, Singleton Pattern, Factory Pattern, and Composite Pattern.

**Structure of the project report**: This project is structured in four parts, namely Introduction, Design, Implementation, and Conclusion. Each part has information about how this project is going to be implemented and concluded.

Diagram, schematic

Description automatically generated**PART II: DESIGN OF THE SOLUTION**

**Elements:** The image provided above is the UML class diagram of our project. The elements (classes) are MiniSoccerApp, PlayerStatistics, PlayerFactory, SoccerGame, SoccerBall, GamePlayer, Goalkeeper, Striker, PlayerCollection, PlayerCollectionIterator, MenuBarListener, ActionListener, GameListener, and KeyListener. Design principles used are Decorator, Composite, and Singleton. We have used the Singleton design pattern in the SoccerBall since there only needs to be one soccer ball in the game. Decorator design principle were used in the Goalkeeper and Striker classes since they extend the GamePlayer class and have getters and setters. Lastly, we have used the Composite design pattern in MiniSoccerApp which is the main class as it represents whole-part hierarchies. The most relevant operations for each class that we implemented are listed in the table below.

|  |  |
| --- | --- |
| **Class Name** | **Most Relevant Operations** |
| GamePlayer | * isPlayerHasBall() * grabsBall() * getPlayerName() * getPlayerColor() * getPlayerPosition() * setPlayerPosition() * getPlayerStatistics() * setPlayerStatistics() |
| Goalkeeper | * moveLeft() * moveRight() * moveUp() * moveDown() * shootBall() * moveRandomly() * setInitialPosition() |
| Striker | * moveLeft() * moveRight() * moveUp() * moveDown() * shootBall() * setInitialPosition() |
| PlayerStatistics | * getStatistics() * setStatistics() * toString() |
| PlayerFactory | * getPlayer() |
| PlayerCollection | * add() * remove() * getGamePlayer() * get() * sort() * iterator() |
| PlayerCollectionIterator | * hasNext() * next() |

**PART III: IMPLEMENTATION**

**How we compiled all the classes**

We have made two classes namely Goalkeeper and Striker that extend the class GamePlayer. GamePlayer class is the superclass that has the setters and getters which help in the working of the mini soccer game. The main class, MiniSoccerApp calls PlayerFactory which generates an object of type Striker and Goalkeeper. The PlayerCollectionIterator class is used by the PlayerCollection to return an iterator for the list of GamePlayer objects.

**JUnit testing**

We have created the tests for the model in the MiniSoccerAppTest class which is in the JUnitTests package in the src folder. Using the Eclipse IDE and Jacoco for code coverage, the tests were able to cover 81.1% in the model package and 95.8% in the model.players package, which is an overall 88.5% code coverage. This is enough code coverage to fulfill the requirements for this project.

**Tools used to complete this project:**

As we are working in a team, each person uses various tools. All the tools used are Github, Github Desktop, Visual Studio Code, Eclipse, JUnit to debug the code when necessary, and Jacoco for the code coverage. The JDK version that was used to compile the code is OpenJDK 16.0.2. The extensions used in Visual Studio Code are Code Runner, Eclipse New Java Project, Language Support for Java, and Extension Pack for Java. To run this project, first go to <https://github.com/mrkami1/EECS3311-Lab-5-Group-38> then watch the included video named **EECS3311 Project 2 Tutorial.mp4** in the root folder.

**PART IV: CONCLUSION**

**What went well**: Getting the game running using the incomplete demo went surprisingly quickly and was able to get working within 2 hours. Implementing the missing classes was not very difficult to do.

**What went wrong**: There was a lot of trouble trying to implement the custom PlayerCollectionIterator class since there was an issue where when it was being used in the PlayerCollection class, the game would run but the players would not show up on the screen. It was able to be resolved using generic type custom iterator methods for PlayerCollectionIterator. Also writing the JavaDoc and JUnit tests was very tedious and time consuming, it took many hours to complete.

**What we learned**: We learned about implementing java classes from the class diagram that makes it easier to develop software projects. Creating the JavaDoc was something new we learned how to do and how to create tests for our classes and check their code coverage using the Eclipse IDE. We also learned how to work in a team in order to analyze, design, implement and test a project in order to conclude it. Lastly, we learned how a 2D game using Java Swing functions.

**Advantages and Drawbacks of completing this lab in a group**:

**Advantages**: Teamwork, Less workload on each member, and Collaborative/creative thinking.

**Disadvantages**: Difficulty in getting started, Time Consuming, and Unequal Participation was one of the bigger drawbacks of group projects.

**The top three recommendations** to complete this project would be:

1. To use the code provided to start off the project for a better understanding.
2. Understand the requirements and make sure the class diagram is implemented with proper patterns.
3. Pay attention to the exceptions.

**Contributions:**

|  |  |
| --- | --- |
| Mashhood Anwar | Coding, Video, UML, Report. Was collaborative. |
| Nishaant Dhingra | UML and Report. Was collaborative. |
| Bilawal Nisar | UML and Report. Was collaborative. |
| Hasmat Sidhu | Coding. Was collaborative. |