EECS 3311 Section B Second Software Project

Mini-Soccer Game

Xinger Yao 215106578

Chang Liu 216882813

Jiakang Chen 216952012

Yuyu Ren 216537911

TA’s Name: Naeiji Alireza

Introduction

This software project is to complete a mini-soccer game. The goal is to display the game and let the user play the game successfully. The application displays an interface with two menus “Game” and “Control”. The Control menu allows pausing or resuming the game, the game menu allows the user to start a new game or exit the current game. When the game is on pause, neither the game player nor the soccer ball can move. The interface also displays “Time” which is the remaining time in seconds of the game and “Goal” which is the number of goals scored by the striker. The game is over if and only if the remaining time is 0. Initially, when the user launches the application, the game score is zero, the remaining time starts at 60 seconds and it decreases during the game, when it reaches 0, the game will end automatically. The interface displays two game players: a striker and a goalkeeper, the goalkeeper is in front of the gate and the striker is at the bottom of the interface. During the game, the user will control the striker by using the Arrow Keys to control the movement of the striker and by using the spacebar to shoot the ball to score a goal, and the goalkeeper moves randomly in a gaussian distribution fashion on the left or on the right of the gate attempts to catch the ball. When the game is over, the statistics of each game player will be sorted and displayed, all controls, including pause and resume, will be disabled until a new soccer game is started.

There are many challenges associated with this software project. For example, the biggest challenge we have faced is to make the Junit test cover 80% of the project as required. We will explain all of them in detail as well as how to overcome them in the following paragraphs.

In this software project, we will be using many concepts from OOD, OOD principles and design patterns to carry out the project. Such as Object-oriented analysis and design workflow,

Encapsulation, Abstraction, Inheritance, Polymorphism, Singleton design pattern and Factory design pattern.

This report includes four main parts including introduction, design, implementation and conclusion, we will structure it strictly followed by the project requirement, all the mentioned details from the document will be explained. The report starting with a brief introduction, after that, we will explain our design in detail and followed by a design solution with the UML class diagram. Third, we will be presenting the details of the implementation, a short video presentation will be provided. At last, we will give an overall conclusion to this project, explaining what went well, what went wrong and what we have learned from this project as well as our recommendations to ease the completion of this software project. A table will be provided that shows the various duties allocated to each group member, as well as the percentage of the work done by each group member Indicate whether or not each group member participated.

Design

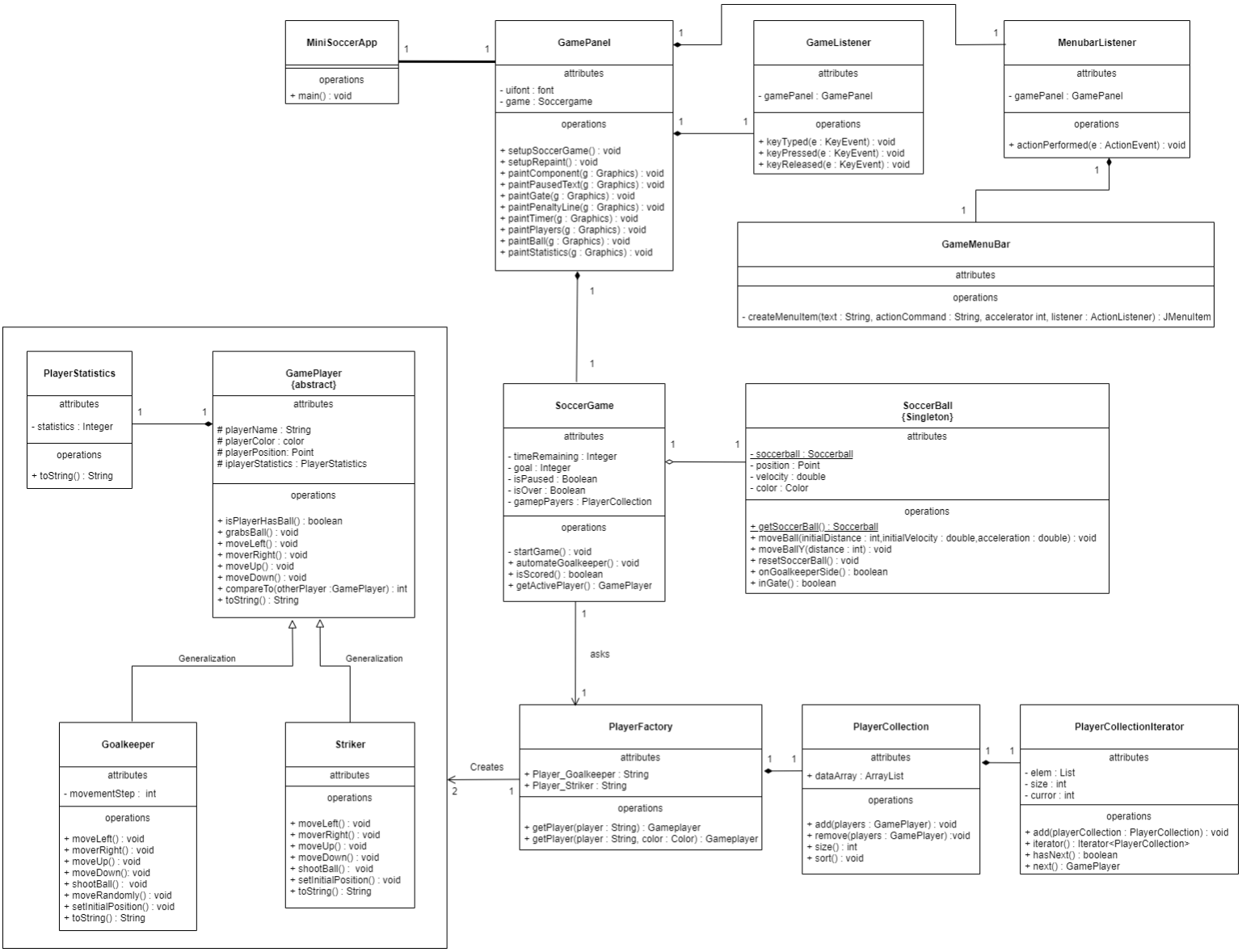

Figure 1: UML diagram of project Minisoccergame with Abstract Factory and Singleton design pattern

In this project, our group implemented two design patterns which are Factory design pattern and Singleton design pattern. We want a factory design pattern because we would want a method to return one of several possible classes that share a common superclass. In our Minisoccergame project, we want to choose a game player for the soccer game and we do not know what type of game player that is going to show on the frame. In this case, the

playerFactory class will pick the string assigned to a specific game player and then return the value to the user. The UML class diagram we created has a client class which is MiniSoccerApp. It contains the main method which is going to call the game panel object, and then the game panel will create the soccer game. After that, it will call the factory object and ask to make Gameplayer and provide a String to go over the abstract Gameplayer superclass. All the common different methods that we need in superclass will be implemented. According to the two subclasses which are striker and goalkeeper, they are all going to implement this abstract Gameplayer class by inheriting the method. The factory design pattern allows us to create objects without specifying the exact class of an object that will be created.

The singleton design pattern can be used to design the soccer ball because there is only one required in this project. This pattern could be designed for a single class which is the Soccerball class to create a soccer ball object and make sure that it is the only single object that gets created. In order to implement the Soccerball class using the singleton design pattern; we make the class constructor private. A singleton instance called Soccerball is created and we also need a static method to return it. Therefore, A public static getSoccerBall method is implemented to allow users to see and return that instance.

According to the UML class diagram that we created. We implement Abstraction, Encapsulation, Polymorphism and Inheritance in our project. We design an abstract class called Gameplayer. It contains several abstract methods such as moveUp(), moveRight(), shootBall(), etc. These methods do not have a body. The body is provided by the inherited subclass Striker and Goalkeeper. Talks about inherited methods that increase the reusability of our project, Inheritance principle has been applied through these three classes. Gameplayer is the superclass and it has two child classes which are Striker and Goalkeeper. The next principle is Polymorphism. Not only the inherited classes that I mentioned before are benefiting from this

principle, but also the getPlayer() methods are shown in PlayerFactory class. These two methods have the same name but with different parameters. We implemented Encapsulation in many of the classes such as SoccerGame, GamePlayer, GamePanel, etc. Each of these classes has its own mutator and accessor. The attributes in these classes are either private or protected in order to hide from other classes to avoid unwanted access by clients.

Implementation

The whole code starts at the MiniSoccerApp class. This class's main purpose is to create an application in order to generate an interface to display the entire element which could form a soccer game including goalkeeper, striker, Gate, etc. The main method in this class will paint the required elements by creating objects. It also paints the number of goals scored by the striker and the remaining time starts at 60 seconds from the GamePanel class. There are also three objects created from the GameListener and MenubarListener class based on the GamePanel objects. GameListener is able to let the user control the active player which is the striker's movement. A menu bar is created by GameMenuBar class which allows pausing or resuming game and starting or exiting the game. The MenubarListener class will perform the action from the menu bar. The SoccerGame class is called by the GamePanel class. The main purpose of this class is to generate soccer balls from the SoccerBall class and call the PlayerFactory class to get one striker and one goalkeeper object. Moreover, it provides the attribute and method to function the timer and tract the goal. PlayerColletion class instantiates an Arraylist which contains one striker and one goalkeeper's behavior. The PlayerCollectionIterator class will then make these elements able to iterate over the PlayerCollection. GamePlayer class is a super class which contains the base behavior which its subclass striker and goalkeeper has. The two sub classes will inherit all GamePlayer class's

attributes and methods, plus having its own additional method such as moveRandomly(). Furthermore, the GamePlayer class will create objects from the PlayerStatistics class and store them in its own playerStatistics attribute. After all the classes are compiled, the JPanel will create an interface which displays all the requirements asked in the project. The goalkeeper will move randomly on the left or right in front of the gate and try to catch the ball. The user can control the striker moves randomly but cannot cross the penalty line. The other function such as timer and goal point which displays on the left top corner will be correctly computed.

In this project, the JUnit tests are written into 11 different files which are the GameMenuBarTest.java, GamePanelTest.java, GamePlayerTest.java, GoalKeeperTest.java, MenubarListenerTest.java, PlayerCollectionIteratorTest.java, PlayerCollectionTest.java, PlayerFactoryTest.java, PlayerStatisticsTest.java, SoccerGameTest.java, StrikerTest.java, Each test file tests its own java file.

The code is written with Eclipse 2021-09, the version of JDK is JDK-13.0.1.

All the other requirements can be found in our code.

Conclusion

For this project, our group did an excellent job. Refusing to just satisfy the basic requirements, we try to make this to be a perfect software. The software can be executed well, with functions to shoot the ball and catch it, pause or resume the game, restart a new game and so on. The software is implemented with design principles. Abstraction, Inheritance, encapsulation and polymorphism are all used in the implementation of classes. The design pattern, factory pattern

and singleton pattern are used to make the process efficient. And the test code is compiled well to check every method in the class to ensure the software can be executed successfully.

The game is implemented and works in the correct way, however, some shortcomings of this software might be caused by the original given code and the data we set. For example, when we shoot the ball to the gate in this game, the ball will always be thrown back by the goalkeeper. We can shoot it to the gate in the situation that the x positions of the striker and the goalkeeper that in the left-most and right most-position of the gate. Moreover, during the Lab, we all struggled with how to implement JUnit and spent most of our time on it to try to reach the requested 80% coverage. Obviously, we need to do more practice on the testing part in the future. We learned a lot from this project about the implementation of JUnit and how to find its coverage with Jacoco.

From this project, we learned that we should have an overview of the project. It is the first thing we should consider when designing the software. We also should put more attention to the details. For example, the data, the type of attributes and methods. The mistake in these details may destroy the whole project. Besides, the generalization relationship should be considered carefully. The code of the software is always with lots of superclasses and subclasses. And in the process of compiling the subclass, some points like the attributes and the overriding of some methods should be carefully considered. Moreover, we learned the importance of the JUnit test. The result of the test is necessary for the software design. It will ensure that we provide the game with no mistakes to the users.

There are definitely benefits to group work. Since the work can be assigned to group members, the stress for everyone would be reduced. A member can concentrate on the part he is skilled in, then finishes it as perfectly as it can be. Otherwise, when collaborating with others, each

member would realize the missing points or his wrong thought for the course material and then fix it. In other words, collaboration can direct group members to the correct place for the study.

However, every coin has two sides. Assigning the work would result in a negative infection to the ability of software design. To finish a project, the most critical thing I think is to have an overview. The code and UML diagram are depending on each other, and the introduction is usually another form of the overview for the project. If any part is done badly, the whole project would possibly face a situation with failure. So that it should be necessary for everyone to be skilled in all parts and have the ability to provide a useful overview. Another point is that each member would always choose the part he is good at. It always says that “Practice makes perfect”. Instead of the part assigned, each one needs more practice to the parts he is weak in. But in most cases the group work will direct us in the opposite direction.

To ease the tasks facing the software project, our top 3 recommendation is that:

1. Understanding the design. To have a good overview of the project, an understanding of the overall design can make communication within the group easier.

2. After finishing a part, always do another part related to it. For example, we should always implement the subclass, superclass, or other classes associated with the class we just finished.

3. Learn while working. When facing tasks we can’t deal with, ask for suggestions from the Professor or TA. We can also use some useful tools like the W3schools to learn what is not covered by course materials.

Xinger Yao	Works on the designing and coding part. Works as the team leader clarifies the team's objectives, makes sure every member understands their role and assigns tasks to members so they can help the team achieve the goals. Gather ideas and lead group meetings.
Chang Liu	Works on the report part. Contribute ideas and suggestions for resolving problems within the group. Monitor interactions and progress, set deadlines to keep members on task, ensure everyone is on track and the project continues to meet its objectives. Check and modify the overall quality of the project.
Jiakang Chen	Works on the report part. Contribute ideas and suggestions for resolving problems within the group. Encouraging communication between group members, ensures that all group members have a chance to participate and learn. Recording team meetings and maintaining documentation of group activities.
Yuyu Ren	Works on the code and report. Contribute ideas and suggestions for resolving problems within the group. Communicating the project’s goals throughout the entire project, providing feedback and concerns on progress as well as identifying and eliminating them. Help the team overcome challenges.