The Mini-Soccer Game

EECS3311: Software Design

TA: Kazi Mridul

Bryce Cooke – 217346354 – Section A

Farnad Kazem-Zadeh – 214212856 – Section A

Jasleen Chagger - 217546938 - Section A

Angelo Alcober - 241515084 - Section A

**Part I: Introduction**

**Explain what the software project about and what are its goals:**

This project required us to implement a simple form of a soccer game. We were required to build a simple GUI soccer field in which a goalie and a striker faced off. The striker could be controlled by the user using the arrow keys and the goalie would move randomly to the left, right or center of the net. The goal for the player is to score as many goals in 60 seconds as possible, while the goalie tries to save the shots. The game comes with controls which allow the pausing and resuming of a game. If the ball goes into the net, the score is incremented by one and if the goalkeeper saves the shot his save counter is incremented by one. If the ball lands on the strikers side of the penalty line, the timer will continue and the striker can move up and take another shot. If the ball lands on the other side of the penalty line but does not end up in the gate, the keeper will “save” the ball and his counter will be incremented. We keep track of the score and time remaining on screen, and the goals saved by the keeper off screen. The game ends when the timer hits 0, at which point it can no longer be paused or resumed, only reset. At the end of the game the screen will display the goals saved and goals scored by each respective player.

**Explain the challenges associated to the software project:**

This is an interesting and unique challenge due to several factors. For one, although starter code helps point us in the right direction, it also means there’s a level of catching up and understanding to do when starting this project. We must use the given files, but in order to do that we must first comprehend what every aspect is doing and how they’re interacting. Another challenging aspect of this project is the usage of software design patterns. While helpful, they’re also relatively new to us and require some thinking about to determine where we can best implement them. We also anticipate writing JUnits for every class will be difficult and time consuming.

**Explain the concepts (e.g., OOD, OOD principles, design patterns) you will use to carry out the software project :**

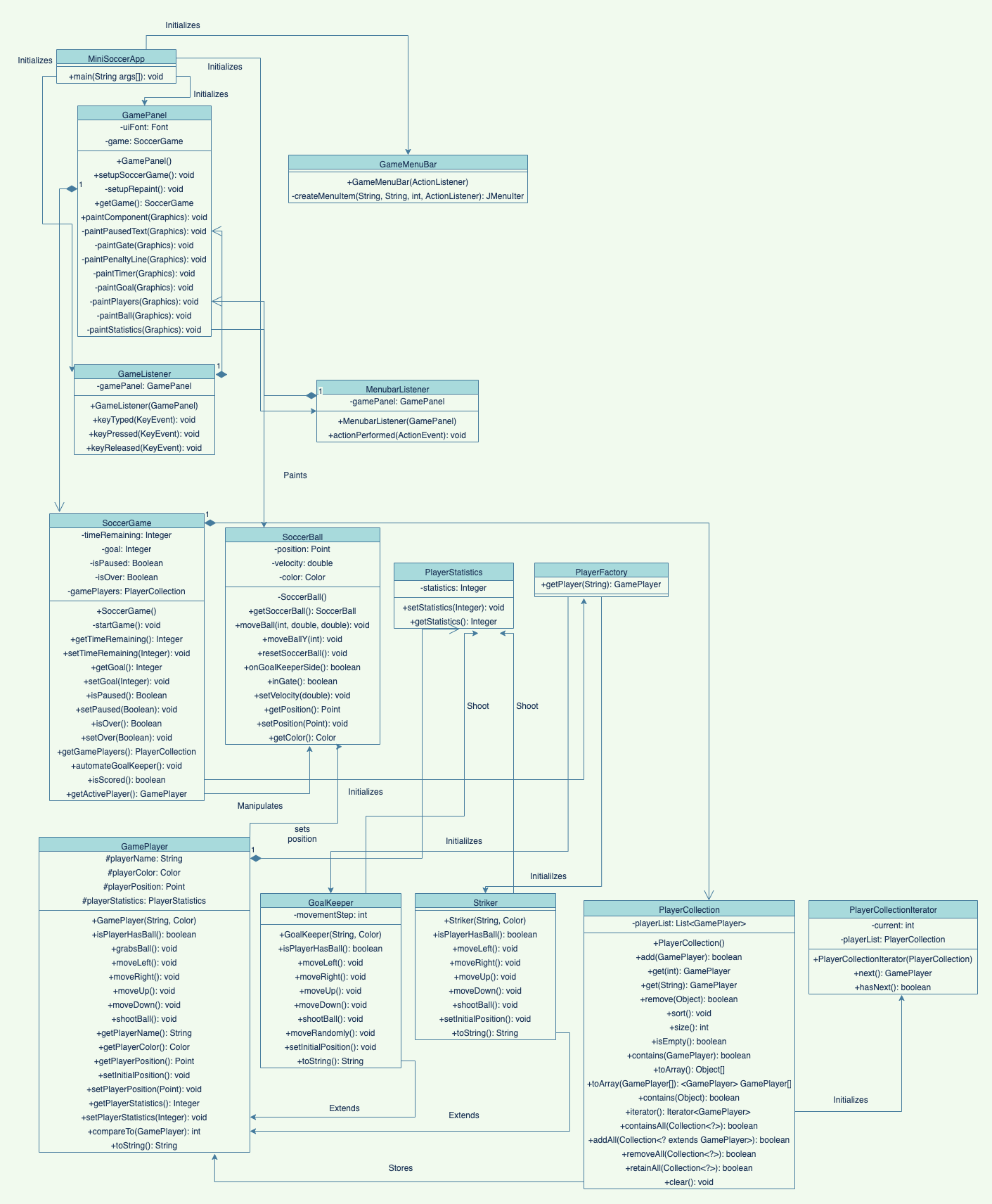
This project will require the usage of several OOD design principles. Inheritance is one example, where classes will extend others in order to use their functionality. Encapsulation is another OOD principle used in this project in order to maintain the state of the game. We want to ensure only the objects themselves can modify their states, and this is achieved using the private keyword combined with methods that allow outside classes to request modifications of the objects state. Abstraction is the concept of only sharing what is necessary with other objects which is important to our program as it ensures that we provide functionality to other classes to use methods from the object without knowing directly how it works. Polymorphism is another object-oriented principle which will be used to help store both the goalkeeper and striker in the playerCollection. Furthermore, the playerCollection must also only have one instance of itself, two extra players would be created per game played otherwise. This usage of the singleton design pattern while creating the playerCollection class will ensure this. The playerFactory class also implements the factory method to create players.

**Explain how you are going to structure your report accordingly:**

This report will be structured as follows: Firstly we will introduce the design of our proposed solution, using UML diagrams combined with descriptions of our classes, then we will specify how we implemented the program and what tools we used to help with out implementation, then we will end the report with a conclusion summarizing our experiences and what we learnt during this project.

**Part II: Design of the Solution**

**Create a first UML class diagram of your system (use at least two design patterns), add the corresponding figure in the report and comment its elements:**

****

Elements of UML class diagram:

* MiniSoccerApp Class: This is the main class used to launch the app. It has the main method.
* GamePanel Class: This class uses the JPanel to display the interface. This class can change the appearances of different components that require changing whenever the project changes its state. This has a constructor and various methods as shown in the diagram to create or trigger visual changes. It also contains the paintComponent method. It has composition with GameListener and MenuBarListener classes.
* GameMenuBar Class: This is used to create the menu bar for the interface. It has a constructor and the createMenuItems method used to add items to the existing JMenu in the construtor.
* GameListener class: This class comes in the controller and implements KeyListener for users to play games. All the necessary key events are set up in this class.
* MenuBarListener Class: This is also a part of controller and implements actionListener for users to change state of the game. It has a constructor and function called ActionPerformed.
* SoccerGame Class: SoccerGame class comes in the model. It has data about the game. It has a constructor. This constructor grabs the players needed for the game using the PlayerCollection and PlayerFactory classes. It has composition with the PlayerCollection class. This class has methods like startGame used to start the game and getter/setter methods to hide data and ensure encapsulation is applied.
* SoccerBall Class: This class stores data about soccer ball. It has a constructor and functions like moveBall, moveBallY, resetSoccerBall. This class also contains getters/setters to encapsulate sensitive information.
* GamePlayer Class: This is an abstract class that creates objects or players for the game. It has attributes for player name, position, color and statistics. It contains a constructor and getter/setter methods along with compareTo. It has a function named grabBall that gives the player the ability to grab a ball. This class also has five abstract methods which will be defined in the classes inheriting GamePlayer class.
* Goalkeeper Class: This class extends and overrides methods from GamePlayer class. It contains additional functions named moveRandomly, setInitialPosition and toString.
* Striker Class: This class also extends the GamePlayer abstract class with additional methods like setInitialPosition and toString.
* PlayerFactory Class: This class is used to create objects of GamePlayer for the game. It has a function getPlayer to do that.
* PlayerStatistics Class: This class only has getters/setters methods and holds the information about the scores of the players.
* PlayerCollection Class: This class is implemented as a container and contains GamePlayer objects created by this project. It also has every method available in a regular Collection class.
* PlayerCollectionIterator Class: This class implements the Iterator of the objects created. It is used to traverse through the PlayeCollection.

**Explain in your report how you have used design patterns: name the corresponding classes, interfaces, and if possible most relevant operation:**

Design patterns:

* Singleton (PlayerCollection):
  + The PlayerCollection class holds the objects GamePlayer created. It contains the methods which are present in a regular Collection class. It is implemented as a container.
  + It is initialized by PlayerCollectionIterator only once to implement the singleton pattern. This restricts the instantiation of PlayerCollection to one "single" instance.
* Factory Method (playerFactory):
  + The factory pattern is implemented using the PlayerFactory class. The GamePlayer class is extended by two classes i.e. Goalkeeper and Striker class. The PlayerFactory class generates instances of goalkeeper and striker for this game i.e GamePlayer objects for this game.

**Use OO design principles in your class diagram. Explain in your report how you have used them: name the corresponding classes, interfaces, and if possible most relevant operations:**

OO design principles:

* Abstraction:
  + Data abstraction can be achieved with either abstract classes or interfaces. The GamePlayer is an abstract class.
  + Also, we provide functionality to other classes to use methods without knowing directly how the methods actually work.
* Encapsulation:
  + Encapsulation is used to maintain the state of the game.
  + It can be achieved by keeping the attributes of the class private or protected and public getter/ setter methods are provided to manipulate the attributes.
  + All the classes have private attributes in this project to keep the logic inside the classes.
* Polymorphism:
  + When a method needs to change depending on the object using it or on its parameters.
  + A class can inherit from another class and change what the inherited methods did in the parent class.
  + The methods that are inherited from the GamePlayer class are overridden in GoalKeeper and Striker classes. These are implemented according to the requirement. These methods can take many forms as they can be called from GamePlayer but perform the function of the desired class.
* Inheritance:
  + The classes GoalKeeper and Striker extend GamePlayer abstract class. This shows inheritance.
  + These classes have the same functions/methods as the parent class with addition to their own states and operations. The child classes override the methods in parent class.

**IMPLEMENTATION**

In this lab report, we have the task to modify and add classes for the project file MiniSoccerGameProject for it to execute. The project is pre-coded and we only have to add some classes to the model part of the MVC pattern. There are 4 packages for use in this project.

These are the packages that contains the classes that are provided:

Main package: contains the main class and initializes the program.

MiniSoccerApp.java

View package: contains the view of the project and creates the menu bar for the users to use.

GameMenuBar.java, GamePanel.java

Controller package: contains the controller of the project and provides the keyListener and actionListener for users to interact with the program.

GameListener.java, MenubarListener.java,

Model package & Model.Players: contains the model of the project and controls the state of the program while being run.

SoccerGame.java, SoccerBall.java, GamePlayer.java

For the project to run, some classes are required to be created. These are the classes that we implemented for the project to run and test.

PART OF THE MODEL – IMPLEMENTED CLASSES:

*PlayerFactory.java* – This is the class that creates the objects of GamePlayers for the game. This has one function which is:

* getPlayer – this is the function to return a player. The goal is to create two players. A striker and a goalkeeper. This function returns either when requested by the game.

*PlayerStatistics.java* – This is the class that holds the information about the scores of the players. It records either the number of goals the striker hits or the number of blocks the goalkeeper made. This class has only a getter and setter for the statistics.

* getStatistic - returns the statistics when needed like displaying it when the game finishes when the timer goes down to zero.
* setStatistic - set the statistics of goalkeeper and striker that are to be displayed at the end of the game.

*PlayerCollection.java* – This is the class that holds the objects GamePlayer created by this project. This class is implemented as a container. In this case, it is implemented as a Collection class. So, every function or method that is available is the one that a regular Collection class has.

*PlayerCollectionIterator.java* – This is the class that implements the Iterator of the objects that was created. To be able to traverse through the PlayerCollection, we implemented an Iterator which is this class.

Tools:

The IDE that we used is Eclipse IDE 2021-09 version 16.0.2

Libraries:

java.util.ArrayList

java.util.Collect

java.util.Iterator

java.util.List

java.util.Timer

java.util.TimerTask

JUnit tester:

org.junit.Assert.asserEquals

**Part IV: Conclusion**

**What went well in the software project?**

As with all things in life this project had its fair share of challenges while providing us with unique learning experiences. Before the design or implementation part of the project we hit the ground running and had figured out good conceptualizations of what each class was meant to do, as well as a basic understanding of what we would have to do during implementation to make this happen. In other words, our requirement analysis step was a smooth process. Following the lead of our requirement analysis step was our design step, in which we created UML class diagrams. While this UML class diagram was constantly updated and progressed alongside the implementation of our program, the initial design covered many of the classes and methods eventually implemented which made our lives a great deal easier down the road, something every team considers a success!

d

**What went wrong in the software project?**

While we had our fair share of success stories during this lab, as mentioned previously, we faced significant challenges together as well. The large concentration of these was during the implementation phase of the project, with many technical problems popping up during the process. Some of the biggest obstacles in our path included the overriding of the implementations of the methods that the collection class defines as we first had to understand how these functions worked, as well as do research into overriding these methods. Another aspect of this lab was one we’re not often exposed to at york; JUnit testing. JUnit allowed us to test our factory and collection implementations during that phase and identify where our errors were. However, as stated, York does not expose students to much JUnit, and as a result, the usage of JUnit itself was a challenge. We not only had to figure out the syntax used in JUnit but also how to make the unit tests examine what specific aspect of the program we needed it to. This required some critical thinking and proved to be time consuming.

**What have you learned from the software project?**

This software project provided us with a lot of insight into several aspects of the software development process, different design patterns, and JUnit testing. The first aspect of designing our solution to this project was understanding the development process better, where to start and how to move forwards. This required revisiting the Software development life cycle, allowing us to gain a better understanding of the steps involved. Furthermore, we had to review content from the course and do our own external research for extra help understanding some of the design patterns we could implement to help our program. These design patterns proved very helpful as one of our bugs was solved by implementing the Singleton pattern. Lastly this project helped us learn a lot about how to test a program, how to make sure after computing our results were the ones we were looking for.

**What are the advantages and drawbacks of completing the lab in group?**

Group work is a great tool for learning how to work collaboratively as a team. This was a major advantage during this project as it allowed us to compartmentalize and work on parts of the project individually, before combining them all to create the full project. This not only eased the workload on each individual greatly; but also allowed us to seek out each other for help when confused on how to continue our work. Another advantage was having more than one brain to analyze what we were required to do. This allowed us to have several perspectives on the approach we should take, as well as how certain methods should be implemented; which both proved critical to our completion of this software project. However, as with all team work activities, some problems were encountered as well. While our group worked well together and everyone contributed fairly, one problem we consistently ran into was figuring out timing for everyone to have a team meeting. As all of our team members lead busy lives outside of school, it was often difficult finding times for us to meet and collaborate all at once. Another drawback we faced was that while group work provided us with several perspectives as mentioned earlier, we also had to find a middle ground between our ideas to implement, as everyone had a fair say. This proved a little challenging in some aspects but we managed to persevere and work well together.

**What are your top three recommendations to ease the completion of the software**

**project ?**

There are several things one could do to help ease the implementation of such a project. Our first and most important recommendation is pre-planning. While the software development life cycle is something new to all of our group, we can not overstate the importance of using this approach to plan ahead very methodically in every aspect of this project. It will make the developers life so much easier, allowing them to include every functionality required, reducing headaches massively. Another recommendation we have is aimed at the group work aspect of this project, it's important to be open to ideas and be willing to go out of your comfort zone to try and accommodate everyone's ideas. It's very important all team members feel equal for them to do their best work. Lastly we suggest every team compartmentalizes this project and allows people to play on their strengths, each member working on that they feel the most comfortable with. This will allow for better quality overall on the project, as people will be on tasks they’re most comfortable with.

**Add a paragraph (or a table) to indicate the different tasks of the work that were**

**assigned to each group member, and the portion of the work completed by each**

**group member. Also indicate if each group member was collaborative.**

As mentioned beforehand, the work on this project was compartmentalized. We each chose tasks we were most comfortable with and came together occasionally to put together work we’d done. All members of the group were collaborative and contributed equally to this project. The requirement analysis and design steps were performed by all members of the team together, the implementation of the program done by Bryce Cooke and Angelo Alcober, and the report and UML Class diagrams created by Farnad Kazem-Zadeh and Jasleen Chagger. These are rough guidelines however, as everyone helped wherever they could when somebody got stuck on a problem encountered during the project.