**Introduction**

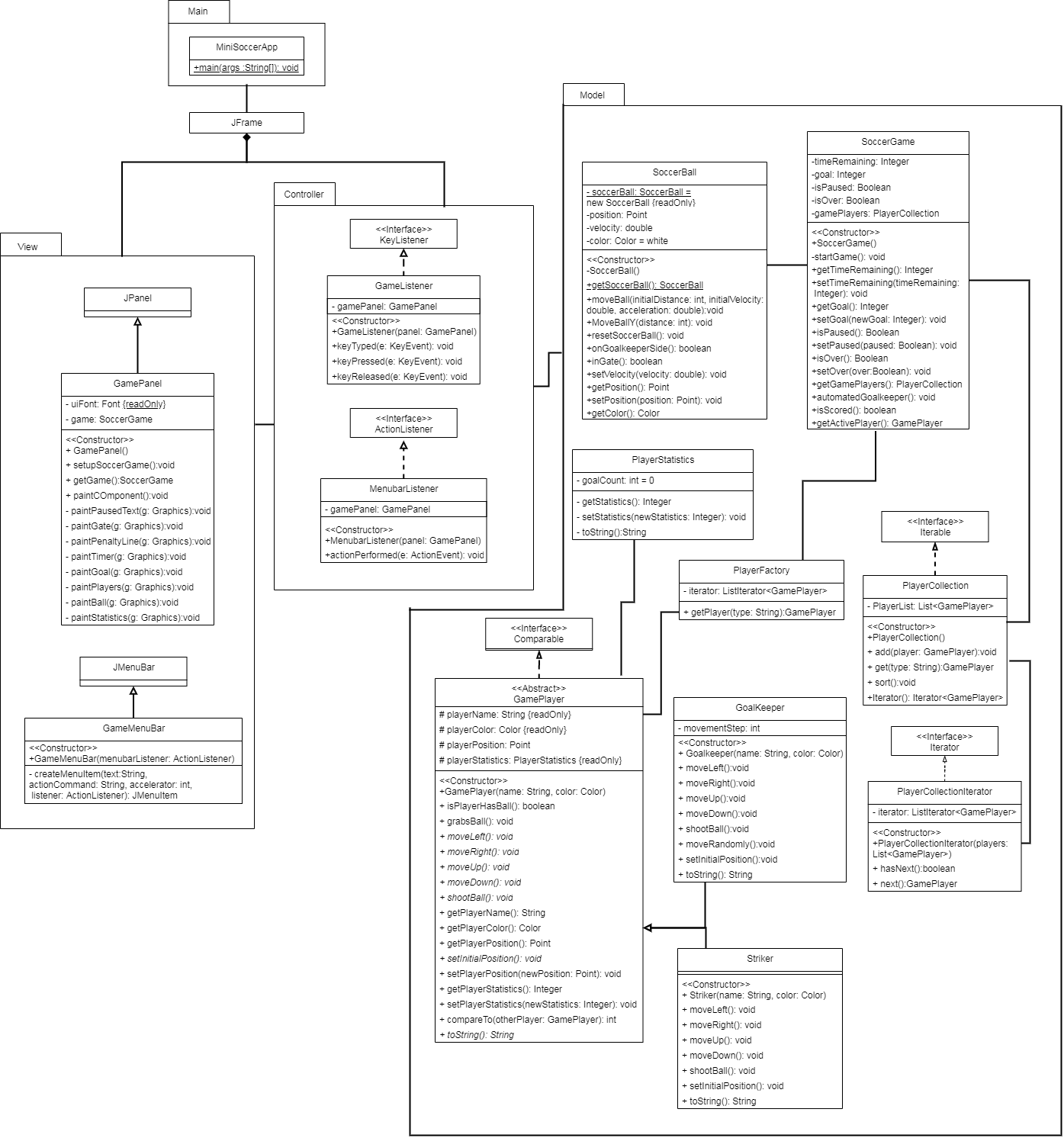
For this software project, the goal is to develop a functioning mini soccer game. The application itself will have two menus. With the Control menu, the game can be paused and resumed. The Game menu allows creating a new game or exiting the current one. The game will feature a striker and goalkeeper. The former will try and shoot the ball in the net while the latter will try to catch it and throw it back. The two players can’t cross the penalty line. Using arrow keys, the stiker can move. Spacebar can be used to shoot. The goalkeeper moves randomly. Countdown till end of game will be displayed with the application in seconds and total goals made during the period. The stiker will have 60 seconds to try and shoot as many times into the net. Each time a shot is successful, the goal count will be increased by one.

Challenges associated with this software project include not only coordinating the different classes to interact with each other to produce the final outcome, but also coordinating as a group to develop the project. Although the MVC architecture helps sort classes, there could be issues figuring out how the classes interact. However, UML might become useful to figure this out. Plus, unlike solo projects, this one requires continuous communication between the team members to successfully complete it. There could be disagreements about design until we reach consensus.

Some of the object oriented principles used to carry out the software project include polymorphism and inheritance. Plus, the design principles singleton, iterator, and factory will be used. For example, the factory pattern will be used for generating game players. Then, the singleton pattern will be used for creating a soccer ball as only one is needed. In addition, there will be sorting and collections. A collection of players will be iterable.

As this project is developed, the report will reflect the stages of development from design to implementation and modification. The final application will be created to meet the goals outlined at the start of this section. This paper will also highlight what went right, wrong, and what could have been done better to approach the task required.

**Design**



There are at least two design patterns in the UML. For starters, the Singleton pattern was used with the *SoccerBall* class as one soccer ball is needed. This pattern allows creating only one object without instantiation of the class object. Plus, the factory pattern was used to create game players for the *SoccerGame*. It requests a concrete type of abstract class *Gameplayer*’schildren such as *Striker* and *GoalKeeper*. Different players are created without exposing creation logic.

Object-oriented design principles were also used in the diagram. For one, there is inheritance from abstract class *GamePlayer* to its children *GoalKeeper* and *Striker.* There is also polymorphism as, for example, *PlayerCollection* can insert *GoalKeeper* or *Striker* when it accepts a GamePlayer with method add. In addition, there is abstraction when the game requests a player using the factory method as there is no exposed creation logic. *SoccerGame* doesn’t know how players are created, but it knows if a certain type of player is requested from PlayerFactory, a matching object will be created. Lastly, there is encapsulation with *SoccerGame* as fields were made private and only methods can be used to access and modify.

**Implementation**

**Conclusion**