CS 319 - Object-Oriented Software Engineering

Design Report – Revised Draft

Head Ball

Group 4

Osman Sefa İbiş	*21301693*
Mehmet Eren Turanboy	21302586
Göktuğ Özdoğan	*21301042*
Shamil İbrahimov	*21300322*

Course Instructor: Bora Güngören

Table Of Contents

1. Introduction

1.1 Purpose of the system

"HeadBall" is a 2D soccer game in which the player tries to score in the restricted time or score count against another player. The aim of the game is to successfully have score advantage against competitor. "HeadBall" is improved by adding modern and complicated features to entertain the player. In game Power-ups will generate unordinary situations by making advantageous and disadvantageous scenarios. This factors makes our game technically one step ahead.

1.2 Design goal

The ultimate design goal for a game must consider to fulfill the desire of the players to play it again and again without feeling bored. Since our game isn't scenario based we have to establish randomize scenarios in gameplay to conquer player amusement and entertainment. In game power ups and obstructers will enable the gameplay to break the monotonicity. Game Physics for "HeadBall" is important to ensure a smooth gameplay. These factors are highly considered while designing the game structure and gameplay interactions. This section clarifies the design goals of the system.

1.2.1 Maintenance criteria

Extensibility: The implementation process requires to produce a game which responds to the object-oriented software approach. Extensibility is required to cope with the changes which occur in the implementation process. The features must be easily removeable and addable therefore we considered to produce a game which is integrable and interchangeable. Our purpose is to produce a game which is maintainable.

Portability:

Java programming language is chosen to cope with the portability factor. Since Java is a platform independent programming language, our game will be executable on every operating system.

Modifiability:

Since our system is going to object oriented base and it has several subsystems with the minimum level of coupling and high level of cohesion, it should be easy for us to modify the system. The reason we want to change the system is that as every game “HeadBall” is also required to change some of its features and user interface to keep people's attention on the game. So our system has to have high modifiability goal.

1.2.2 Performance criteria

Response Time: Even though we do not have any heavy data base system, still we need to have decent amount of response time to play the game. As you know, games require a lot of reaction time so that is why when the button pressed system should recognize it immediately. Also it is same as the object interaction with each other. When the player hit the ball, ball should react to touch and change its direction immediately.

Throughput: Since the system can be played as multiplayer and single player, system should be able to accomplish several takes in a short period of time. For instance, when the both players press the button which is responsible for the jumping, both players should jump at the same time.

Smooth Graphics: Main character, power-ups, goalkeepers and the whole level environment is going to be visualized in a retrospective way. Therefore, at this point, we will not experience a noticeable loss of speed in order to process the graphics and move with the game engine.

1.2.3 End user criteria

Utility: In today's World people want to socialize while they are spending time in front of computer while they have leisure time since they need to socialize. In today's World people like to play multiplayer games since use their leisure time as socializing tool. Therefore, in today's World e-sports and other online competitive tournaments are becoming more important. Our application gives user a change user to play competitive matches with his/her friend. Therefore, it is an opportunity for users to socialize

Usability: In our application, we use very basic controllers like other games are used. Moreover, in our user interface when user enters the application they will find what they want to do in the game very easily since user can understand how to move one screen to other very easily by reading labels that are on the buttons written by very basic and understandable English.

Ease of learning: In our game we used “wasd” and arrows as initial controls like most of the other games. Therefore, user will high probably imagine how he/she can control the game even if he/she cannot imagine they can see the controls by reading instructions in the game. Moreover, instruction includes a lot of information about the game. By reading the instruction user will have enough knowledge about game. Instruction includes a lot of information that explains the purposes and process of tournaments, PvP games and single player game. If user will see that he/she should make some practice to expertise in this game he/she can practice in training mode.

2. Proposed Software Architecture

2.1 Subsystem Decomposition

For this section, the system will be broken down to ease the complexity of the understandability and to explain how the system interacts in itself. We chosed layered architecture style because since we are implementing a game it would be sufficient to employ them under different roles easily. We used three layered style to decompose the system because our game is not complicated to need more than three layers. The subsystems are Interface Subsystem, Data Management Subsystem and Game Subsystem. These subsystems accomplish high coherence and low coupling by reducing the dependencies among each other and within. As a design pattern we used Model-View-Contoller pattern. Model is data management subsystem, View is Interface subsystem and controller is the Gameplay subsystem.

Interface subsystem plays the role of controlling the user interaction and taking user input to perform the necessary functionalities. Interface subsystem will include all the panels that the game has, to segregate the interface from other subsystems. By the panels the user can navigate in the game and input necessary information via this subsystem. Interface subsystem interacts with the GamePlay subsystem to pass important data's and displays the returning data from the GamePlay subsystem to the player.

Data Management Subsystem deals with storing the scoring updates and game settings. Also stores the ".ser" file which will contain all the scores in the championship. When the user wants to continue where he/she left from the program requires to access this stored score data's of the old matches. These datas will be transmitted to the GamePlay subsystem to process them and continue the gameplay.

GamePlay Subsystem is the essential subsystem to provide major functionalities to the game. The important classes will be in this subsystem to process data. Game physics will be in this subsystem to manipulate the balls direction. Friction and gravity interacts with the ball class. The object-oriented view of the game is based in this subsystem. All the objects are created in this subsystem. Gameplay Subsytem interacts with Interface subsystem to get user input and interacts with the Data Management subsystem to update the score, get game settings information.

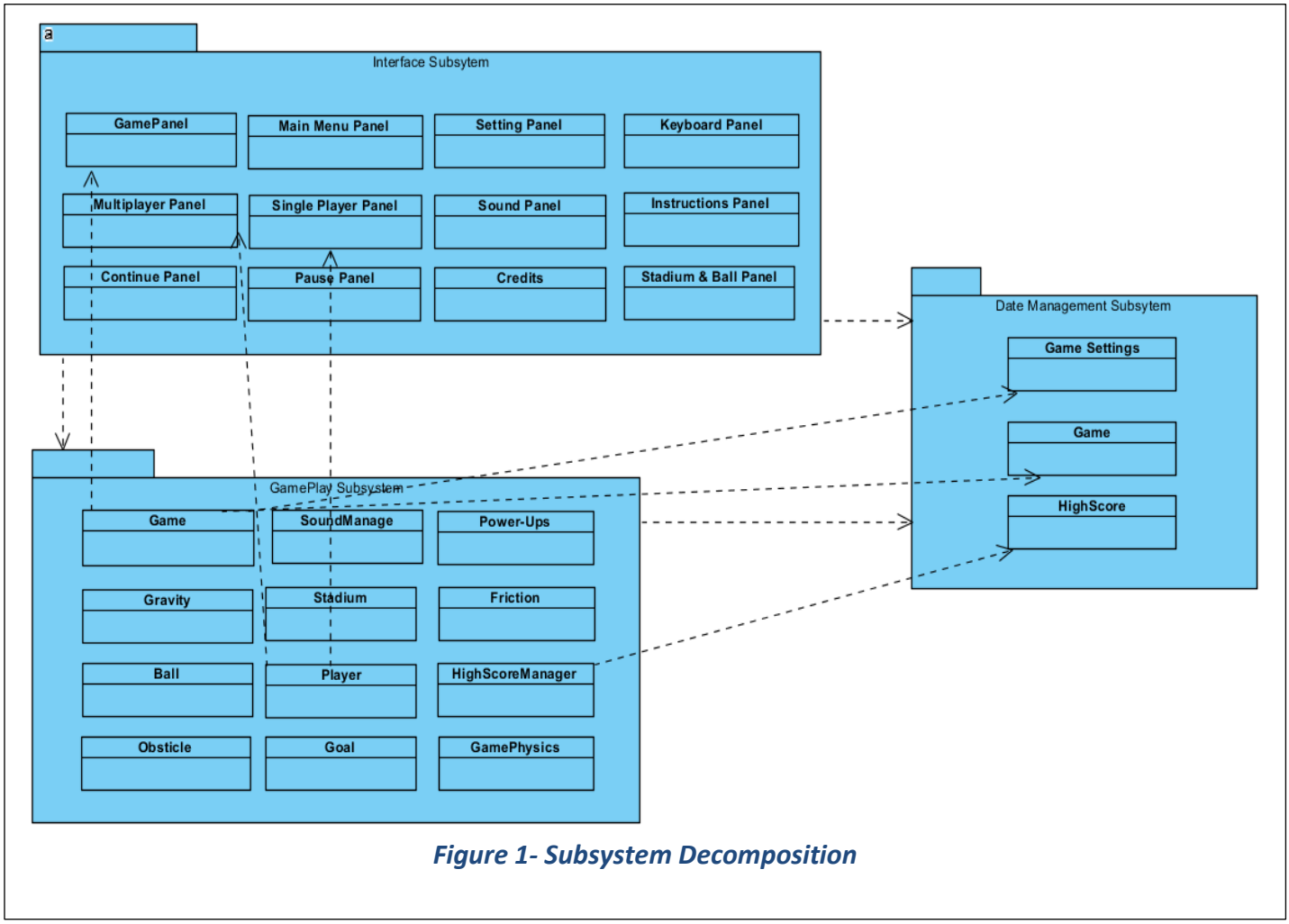


Figure 1- Subsystem Decomposition

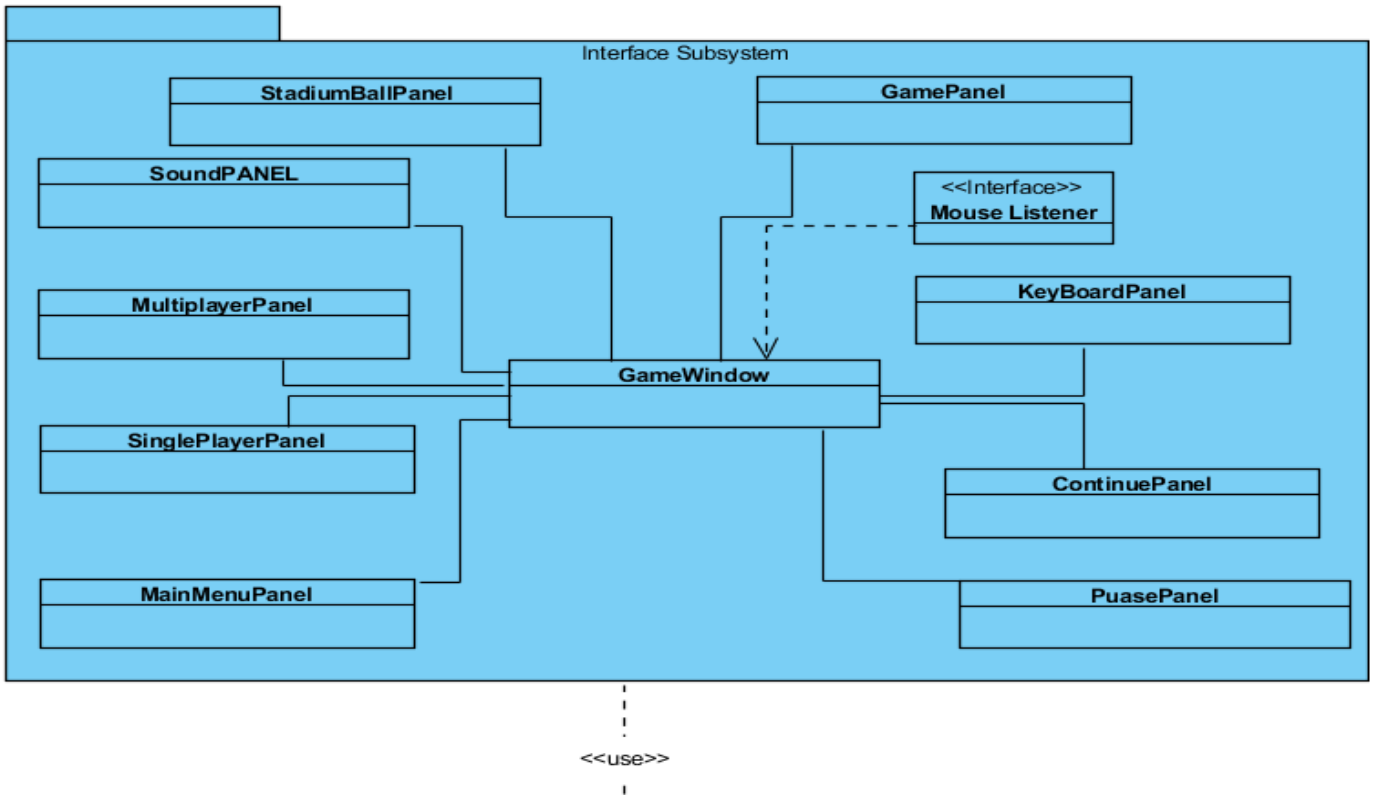


Figure 2- Interface Subsystem

Here in this subsystem, Interface related side of the system is provided. The Frame for our interface subsystem is GameWindow. All other panels are related to that main Window. These panels are not always shown in window at the same time. That why these panels are having 0.1 relations with themselves. Also, to use this GameWindow we are using mouse only that is why we have mouseListener implemented to that window.

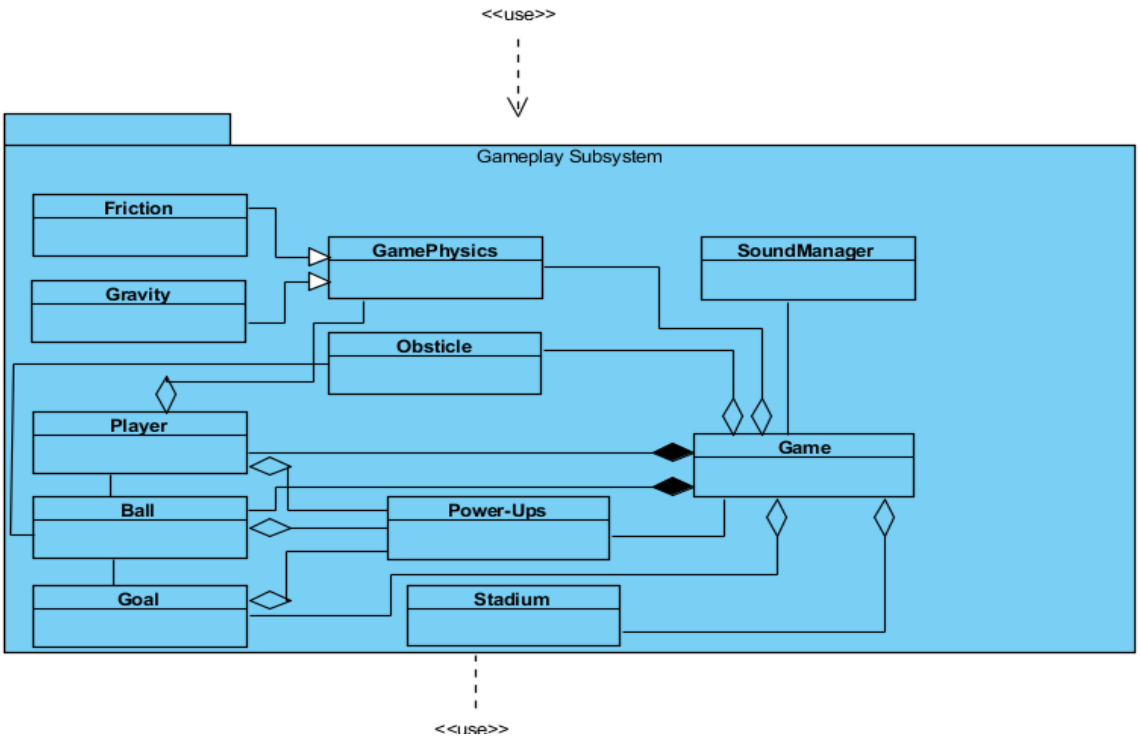


Figure 3- Gameplay Subsystem

This Subsystem is our main subsystem. In this subsystem. main class is Game class. All the classes are related to the Game class. Besides that, Friction and Gravity are related to GamePyhsics and as it is obvious this class is responsible for the physics of the game. Namely, everything important related t implementation of the system is the GamePlay Subsystem.

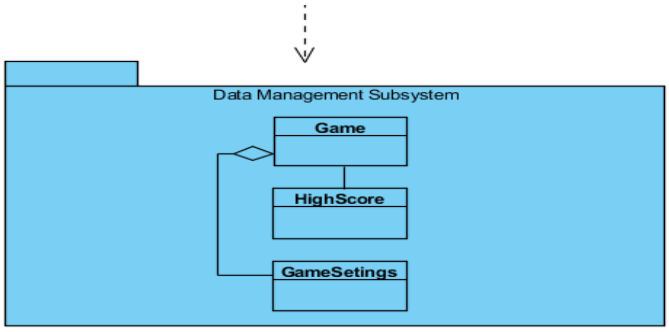


Figure 4- Data Management Subsystem

This subsystem is our small database. In the text file we are keeping the Game, HighScore and GameSetting, since the user does not arrange the Setting over and over again.

2.2 Architectural Patterns

Our design choices concluded to use the MVC (Model-View-Controller) design pattern. We constructed three subsystems to adapt the design pattern. Each subsystem belongs to one of the MVC component which are controller, model or view. The reason of choice relies to manage the complexity of our system. While managing subsystem choices we considered the low coupling and high cohesion factors to decrease the dependencies to benefit system complexity. MVC design patter allowed to system to be more simplified. MVC provided our game to attain some of it's design goals such as reusability, flexibility, extensibility and modifiability. Due to MVC our development process increased since MVC enables rapid and parallel development. Division of labor benefitted progress due to MVC.

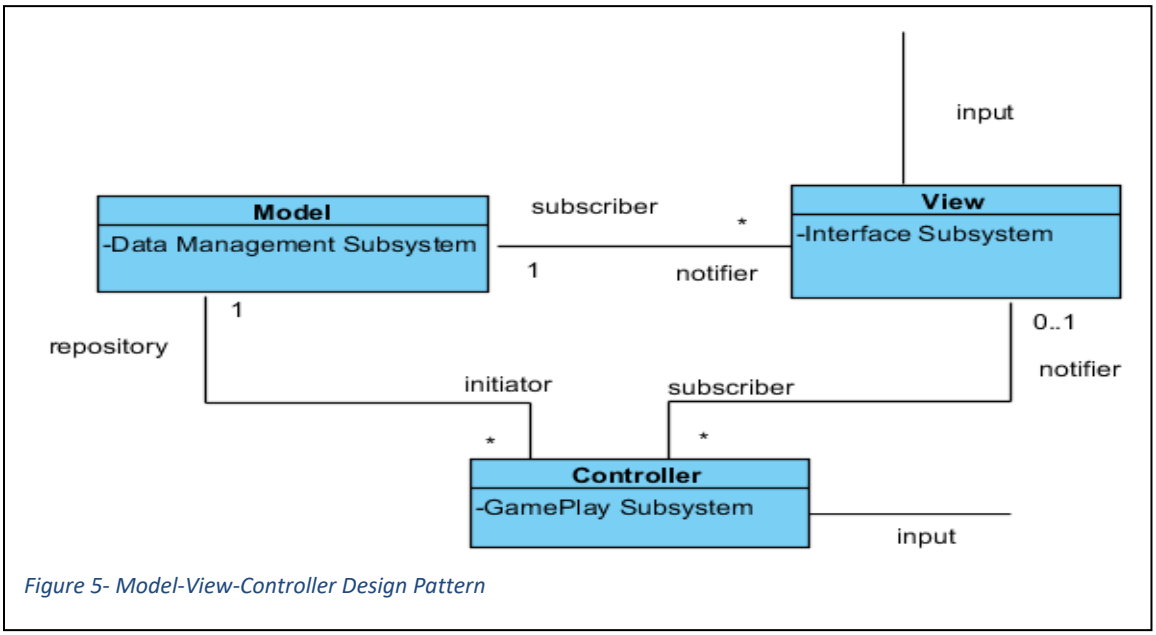


Figure 5- Model-View-Controller Design Pattern

Below diagram depicts the sequence of events in MVC. It shows how model, view and controller respond to a user input during HeadBall.

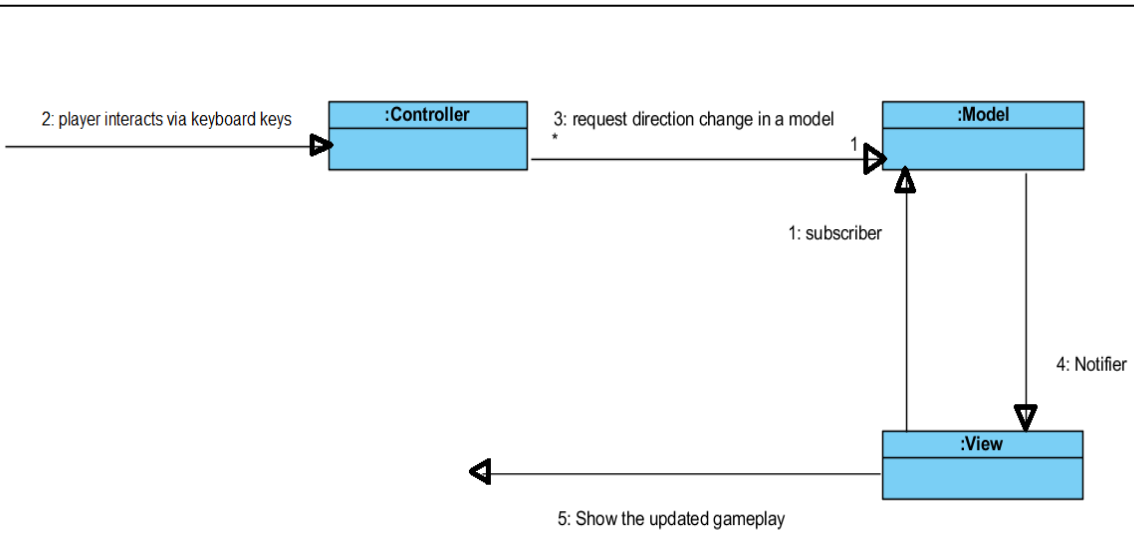


Figure 6- Shows The Sequence of events in MVC

2.3 Hardware/software mapping

Our game is developed by using Java programming language as development language. Therefore, in order to run this game user should have java run time environment in his/her computer. To control the game, user will need a keyboard and mouse. The game will successfully run on almost every computer that are produced in last decade, since the game requirements are very low.

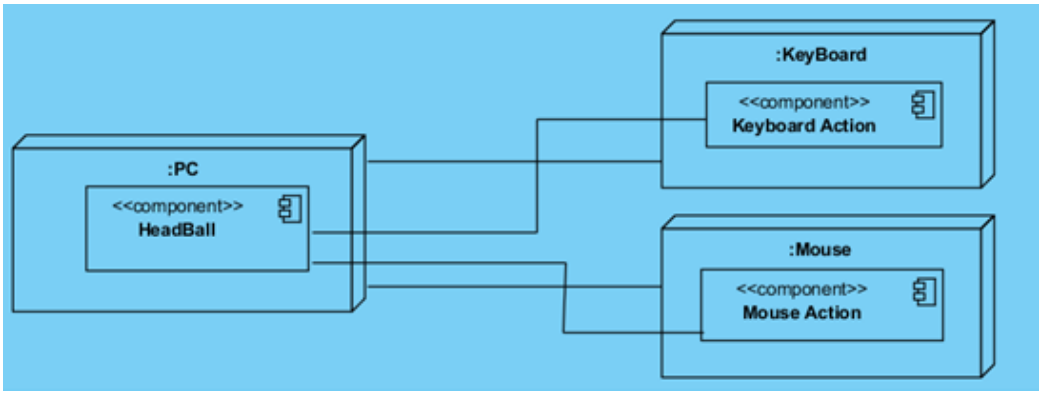


Figure 7- Shows the deployment diagram

As it is obvious from the deployment diagram HeadBall game is controlled by the Keyboard and Mouse actions.

Besides that we are using ".ser" file as a small database to save the Game Settings, High Score and the Game object itself. Game Settings are High Score is the feature of the game and that is why we are obliged to save them. However, saving the Game object is our decision since we think that it will make easier to implementation of our system.

2.4 Persistent Data Management

Since our application does not use any online database system, we use users' hard disk to store the data. Most of the data that should be stored are images and sounds and it's stored inside the project folder. Moreover, scores of the games is stored inside a txt file inside the project folder. Besides, as Seriasible objects we save the Game object in the same data base.

2.5 Access control and Security

Our program is a simple game even without any login page. That is why we do not need any kind of security system for our system. Also, if take into consideration our data base system, we do not save there any kind of important information and it is totally safe to keep data in our ".ser" documents.

Other than that, our system is not connected to any other kind of system to get data like internet, Bluetooth and etc. that is why it is totally safe against any kind of outside interferences.

2.6 Boundary Conditions

Initialization

-When user chooses to play the game all information that application needs to know are sent from the game mode, that is sent according to game mode that has been chosen. For instance, if a user chooses to play a multiplayer game the information will be sent from the game data that includes information according to the multiplayer game mode.

-Before playing the game user should choose the player, stadium, goal limits, time limits, game mode "whether it will be time limited or goal limited" and team. If user doesn't choose

them, game would cannot be playable therefore to prevent this failure we make them initially chosen. Therefore, player will be able to change them during playing the game but if they would not chosen , game will not be terminated.

Termination

- User will be able to terminate the game via the exit button on the main menu.

-If user is playing a game ,user will be able to pause the game and in the pause screen user can terminate the game by clicking the exit button in pause screen.

Failure

The system is very unlikely to fail since all the information that can lead to crash the game are initially initialized in order to diminish the possibility of the failure.

- Player, stadium, goal limits, time limits, game mode “whether it will be time limited or goal limited” and team are initially initialized so it would not lead the crash.

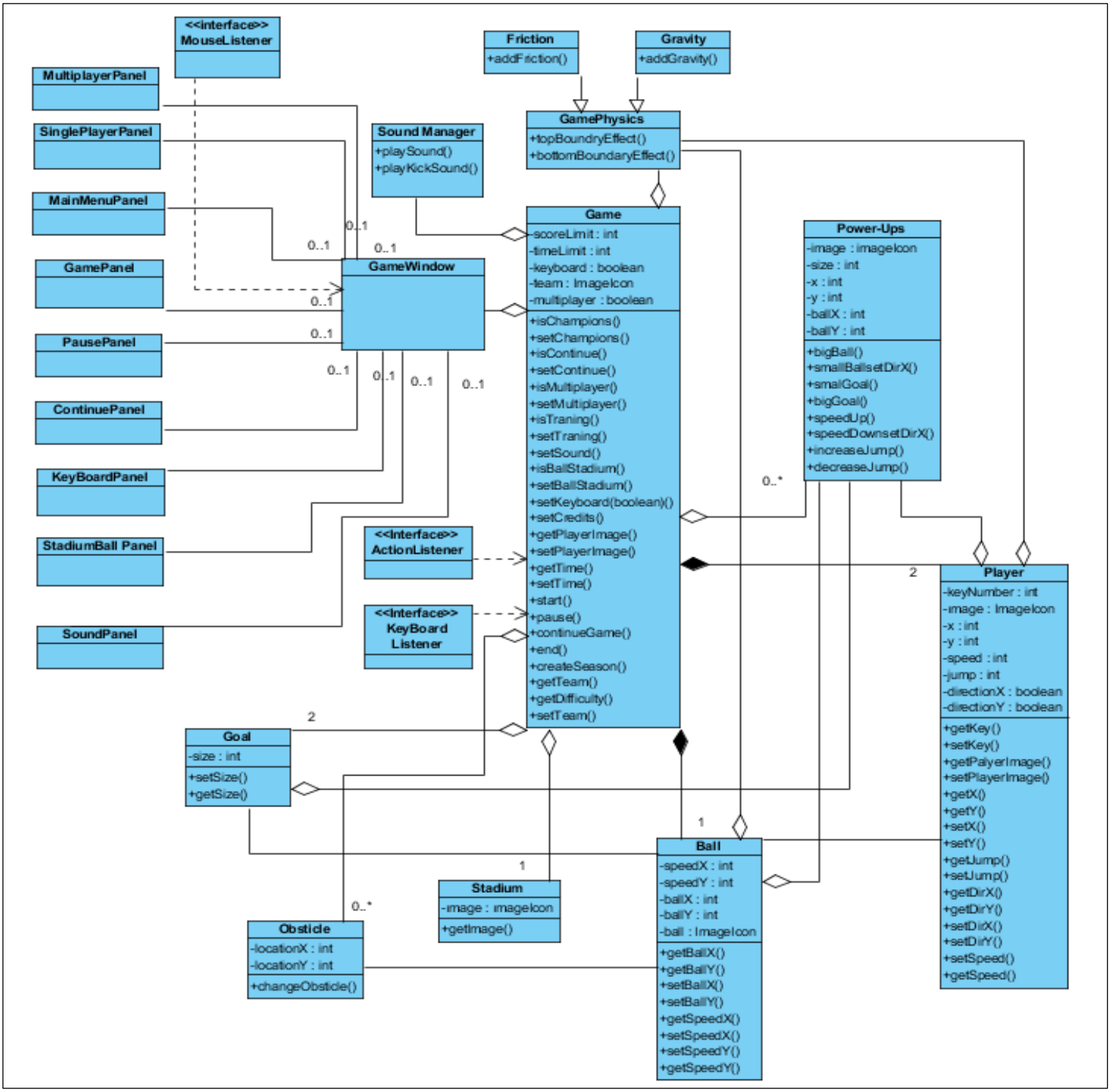
Only possible scenario seems in the game that can lead to failure is if user terminates the game by using termination button “X” on the window that game is played in or if player’s window is shouted down the game that user was playing before termination will be lost. In this case user cannot play the game from where they should continue, so they should play the game from begining.

2.7 Global software Control

Our control flow is procedure driven control, since it waits for every single setting to be made. These are teams, footballers, ball, time or score based settings, multiplayer or single player settings. Besides that, nothing is happening spontaneously. User has to press the buttons to control the player during the game, which also shows that our system is procedure driven control.

3 Subsystem Services

The services and interactions are already discussed in the subsystem decomposition section. To provide greater clarity for “HeadBall” interactions and basic structures, we included the detailed class diagram.



To detail the services we decomposed the class diagram into sub-sections to provide better information. Descriptional information is provided about the methods and classes use.

3.1 Detailed Object Design

- **SoundManage class**

➢ In this class game's sound related jobs are handled.

- In this class playSound() method plays a champions league sound that is opened at the beginning of the game.
- In playKickSound() method when player kicks the ball a kick sound is played.
-

- **Goal Class**

➢ It is a class that represents goals.

- setSize(): Sets the goals size.
- getSize(): Gets the goal's size.

- **Game Class**

- In this class game's main logic are gotten together and get ready to transfer to frame.

- rigthPlayerJump(): This method interacts with an action listener and it is to make football player in right to jump.

- leftPlayerJump(): This method interacts with an action listener and it is to make football player in left to jump.

- leftPlayerRun(): This method interacts with an action listener and it is to make football player in left to run.

- rigthPlayerRun(): This method interacts with an action listener and it is to make football player in rigth to run.

- stopLeftPlayer(): this method important since when the when ronaldo should start to fall this method determines the time

- **Player Class**

getKey() gets the int that represents the keyboard settings of the game.

getDirX() returns whether it is in positive x direction or not.

getDirY() returns whether it is in positive y direction or not.

getPlayerImage(): returns the string that represents path of the image of the player

setPlayerImage(String path): sets the string that represents path of the image of the player

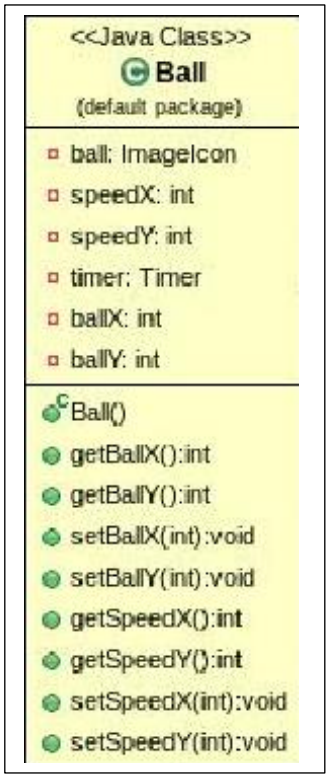
getSpeedX() returns the speed of the player in X direction.

setSpeedX() sets the speed of the player in X direction.

getSpeedY() returns the speed of the player in Y direction.

SetSpeedY() sets the speed of the player in Y direction.

- **Ball Class**



getSpeedX() returns the speed of the ball in X direction.

setSpeedX() sets the speed of the ball in X direction.

getSpeedY() returns the speed of the ball in Y direction.

setSpeedY() sets the speed of the ball in Y direction.

- **GamePhysics Class**

topBottomBoundryAffect(): Changes balls speed by multiplying it with -1 in Ydirection.

SideBoundryAffect(): changes balls speed reverse direction on X cordinate.

- **GravityListener Class**

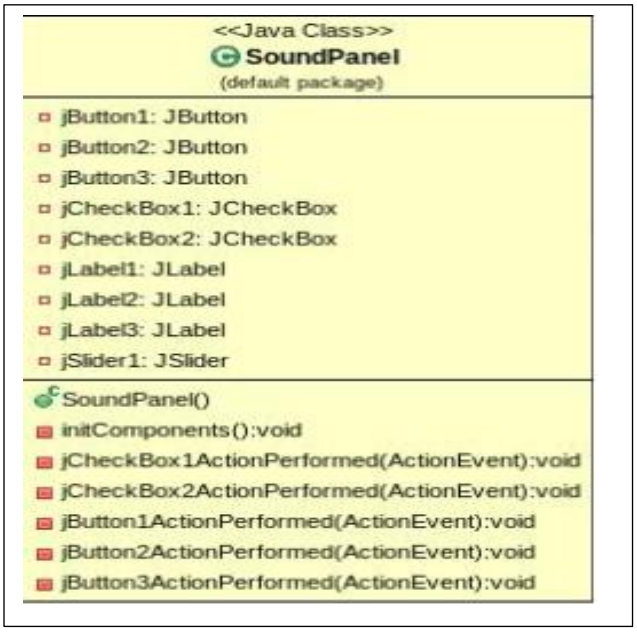
- ➢ Handles friction related physic calculation
- addGravity() : generates gravity for ball and players

- **FrictionListener Class**

➢ Handles friction related physic calculation

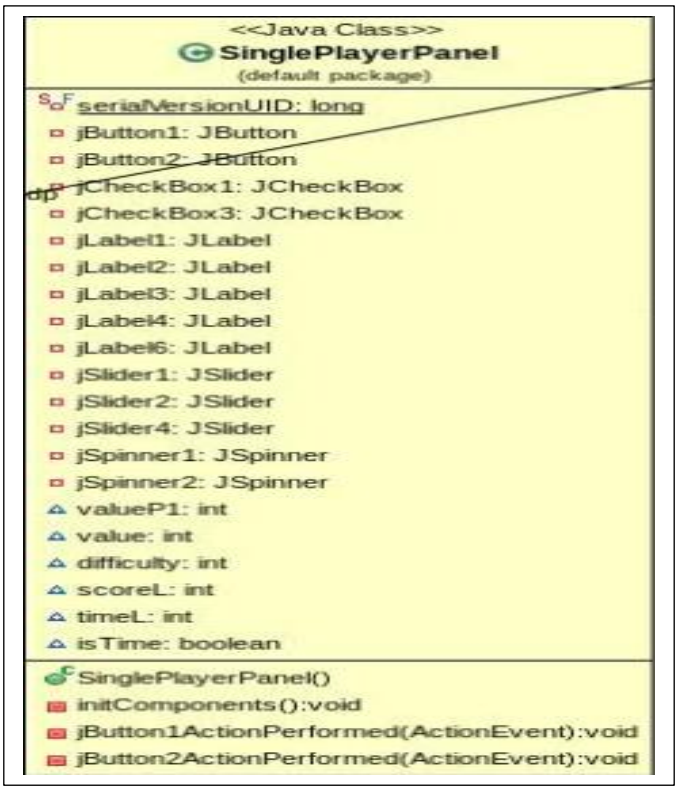
addFriction(): adds friction that affects ball.

- **SoundPanel**



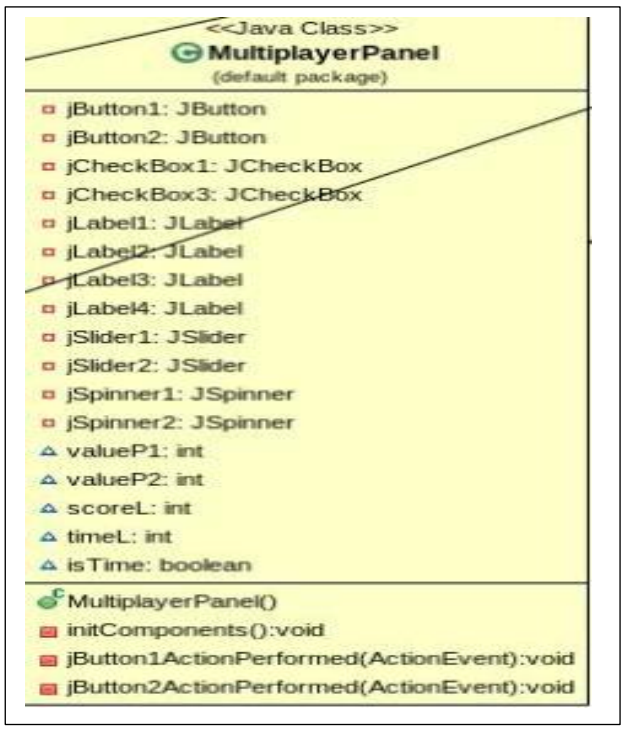
- Sound Panel is responsible for the sounds. For example, Champions League song can be turned off or turned on in this Panel.

- **SinglePlayerPanel**



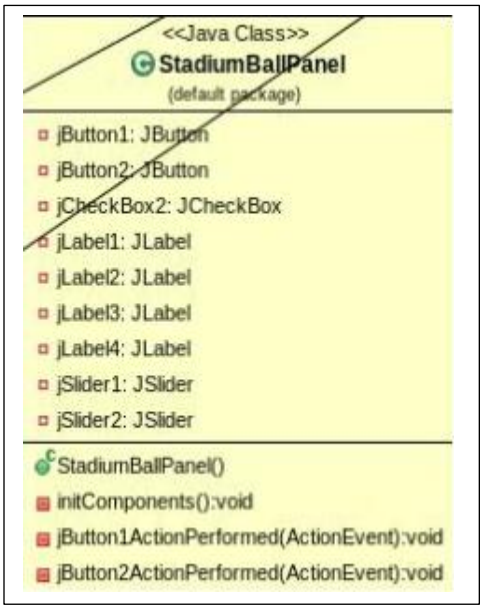
- This panel is responsible for the choosing the difficulty, time, score etc. of the game. When the user wants to play single player game that is what he or she encounters.

- **MultiplayerPanel**



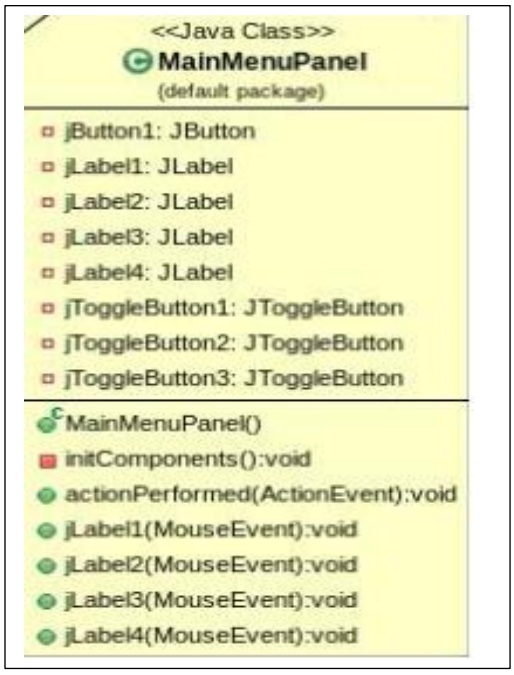
- If the user wants to play Multiplayer game he comes to this panel and choose the time, number of the goals and difficulty of the game that he or she wants to play.

- **StadiumBallPanel**



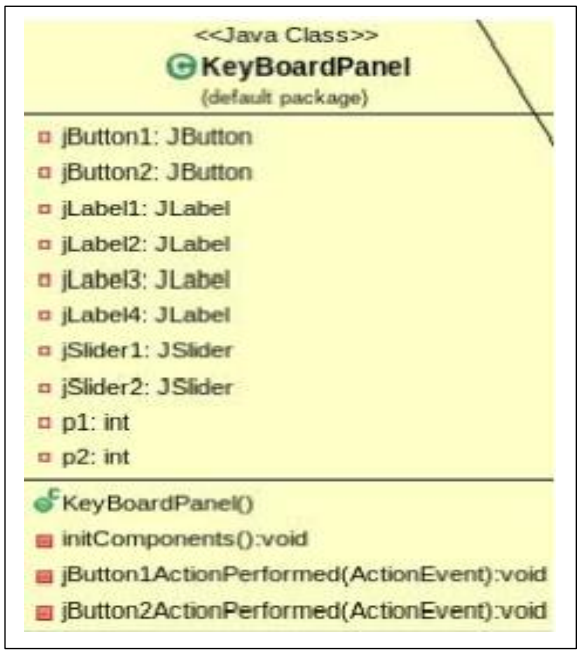
➢ In this panel user will be able to choose the ball and stadium images

- **MainMenuPanel**



➢ In main menu panel user will decide what he/she want to do. In this panel user can choose to play single player or multiplayer game, go keyboard settings menu, continue a game or go to sound settings menu.

- **KeyBoardPanel**

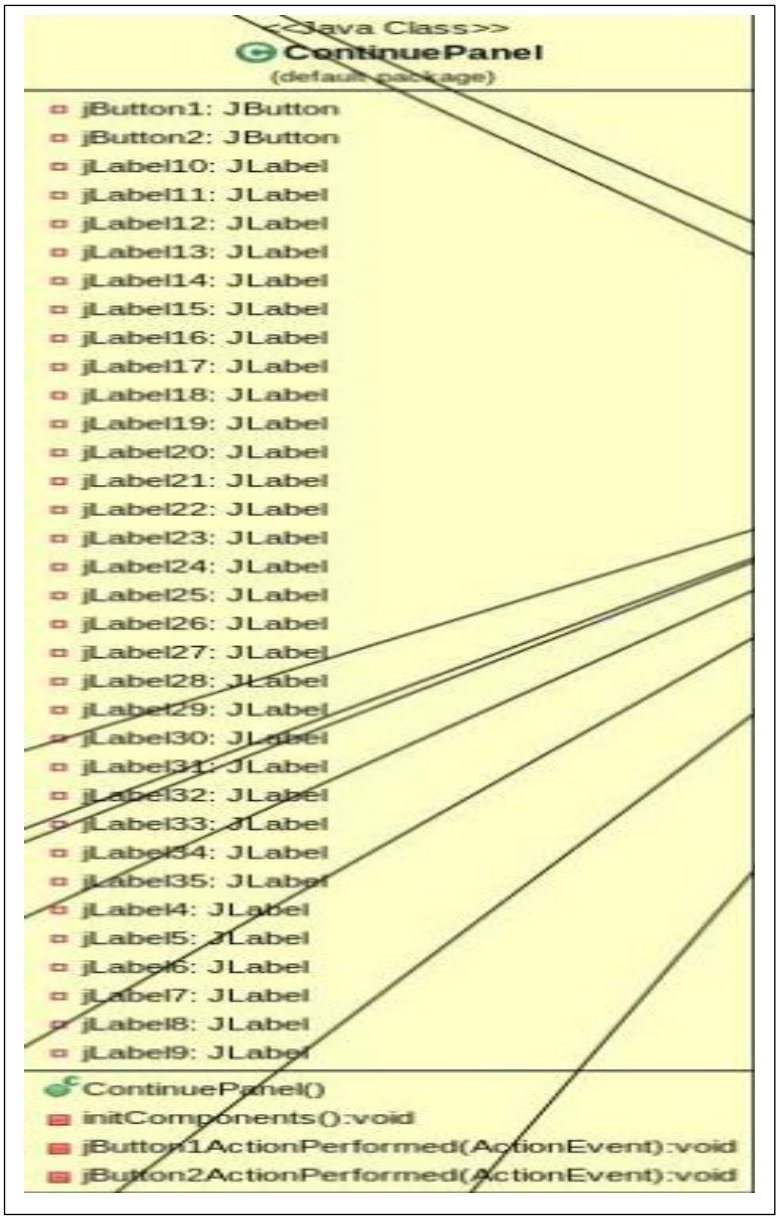


- In this panel user will be able to change key board controls. In this panel user can change the key board setting by changing specified settings.

- **GamePanel**

- In this panel user will play game it basically a panel that multiplayers and single player games are played in.

- **Continue Panel**



- In this panel user can exit the game, continue the game or go to instructions and learn game mechanics.