ISTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

FOOTBALL MANAGEMENT GAME WITH ARTIFICIAL INTELLIGENCE

Graduation Project – Final Report

Ozan Ateş
150110050

Department : Computer Engineering

Advisor : Dr. Damien Jade Duff

June 2016

ISTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

FOOTBALL MANAGEMENT GAME WITH ARTIFICIAL INTELLIGENCE

Graduation Project – Final Report

Ozan Ateş
150110050

Department : Computer Engineering

Advisor : Dr. Damien Jade Duff

June 2016

Özgünlük Bildirisi

1. Bu çalışmada, başka kaynaklardan yapılan tüm alıntıların, ilgili kaynaklar referans gösterilerek açıkça belirtildiğini,
2. Alıntılar dışındaki bölümlerin, özellikle projenin ana konusunu oluşturan teorik çalışmaların ve yazılım/donanımın benim tarafımdan yapıldığını

bildiririm.

İstanbul, 03.06.2016

Ozan Ateş

FOOTBALL MANAGEMENT GAME WITH ARTIFICIAL INTELLIGENCE

(SUMMARY)

The goal of the project is to develop a football management game having a working artificial intelligence system. This project is chosen as a graduation project because game development is a very interesting area related to computer science and it is a great opportunity to demonstrate the knowledge and the skills learnt from computer engineering program.

The first work of this project was design and implementation of a graphical user interface for the game. The knowledge from Basics of Visual Composition course is used for designing the screens of the game. The artistic spirit of human cannot be underestimated and the effects of a well-constructed graphical composition can be one of the concerns of a computer scientist who is dedicated to create visual products. This is also the demonstration of that education of Department of Computer Engineering at Istanbul Technical University also covers the artistic skills and creativity towards visual compositions too.

The second work of this project was building a football simulation and a proper artificial intelligence system for the simulation. A football simulation using Bullet physics engine is built using the benefits of the object oriented programming paradigm. To simulate the 22 football players, a decision making system is constructed as a behaviour based architecture designed for artificial intelligence of football players. Many possible actions are defined for football players and few of them are implemented.

The product of the project, Caretaker Manager, is tested by 5 people who like to play football management games. According to the feedbacks of testers, the user interface and the different features of the game can be considered as good work, while the actions of footballers should be improved.

To sum up, this project was a good experience of software development and it was a good demonstration of the skills that are trained in computer engineering This project could be a lead for further possible projects about artificial intelligence. Solid simulation and working artificial intelligence architecture are built and they can be improved by another project that aims for only artificial intelligence.

YAPAY ZEKA İÇERİKLİ FUTBOL YÖNETİM OYUNU

(ÖZET)

Bu projenin amacı, çalışan bir yapay zeka mimarisine sahip olan bir futbol yönetim oyunu yapmaktır. “Caretaker Manager” adı verilen bu oyunda kullanıcı bir futbol kulübünün teknik direktörü olarak görev yapmakta ve belirlenmiş bir maçta sahaya çıkan takımı belirlemekte ve yönetmektedir. Kullanıcı futbol kulübünün sahaya çıkardığı takımı belirler, sahadaki futbolcuların hücum ve savunma oyunundaki dizilişlerini belirler, futbolcuların oyun içerisindeki tercihlerini bir strateji uğruna değiştirmek amacıyla bireysel olarak futbolculara veya tüm takıma komutlar verir. Caretaker manager; teknik direktörleri ile yollarını ayıran kulüp yönetimlerinin, yeni bir teknik direktörü takımın yönetimine getirene kadar takımı geçici olarak yönetmesi istenen kişi için futbol dünyasında bolca kullanılan bir terimdir. Oyunda kullanıcılar sadece bir adet maç için takımı yönetmektedir ve bu sebeple oyun için “Caretaker Manager” adı uygun görülmüştür.

Bitirme projesi olarak bu projenin seçilmesinin en önemli nedenlerinden bazıları, oyun geliştirmenin bilgisayar bilimiyle alakalı olan en ilgi çekici konulardan biri olması ve İstanbul Teknik Üniversitesi Bilgisayar Mühendisliği lisans programında öğrenilen çok sayıda konu hakkındaki bilgi ve becerileri gösterme imkanına sahip olmasıdır. Örneğin, Project Management (Proje Yönetimi) ve Software Engineering (Yazılım Mühendisliği) derslerinde öğrenilen bilgilerle bu projenin planlama, tasarım, uygulama ve test fazları gerçeklenmiştir. Numerical Methods (Sayısal Yöntemler) dersinde öğrenilenler sayesinde; bir gerçek dünya problemi olarak futbol topuna uygulanan itme ve topun gittiği uzaklık arasındaki ilişki incelenip, simülasyon dünyasında bir çözüm olması amacıyla formülize edilmiştir. Analysis of Algorithm (Algoritma Analizi) ve Artificial Intelligence (Yapay Zeka) derslerinde alınan eğitim, yazılım gerçeklenmesi sırasında karşılaşılan çok sayıda sorunun iyi bir çözüme kavuşturulmasına yardımcı olmuştur. Linear Algebra (Lineer Cebir) ve Robotics (Robotik) derslerindeki bilgiler, bu projedeki 3 boyutlu noktalar ve vektörler üzerindeki çalışmalarda işe yaramıştır. Discrete Event Simulation (Ayrık Simülasyon) dersinde simülasyon oluşturulma yöntemleri öğrenilmiştir ve bu projede kullanılan eğitimlerden biridir.

Bu projede sadece teknik eğitimler değil, sanatsal eğitimler de kullanılmıştır. Projenin ilk aşamasında grafiksel kullanıcı arayüzü tasarlanmıştır ve gerçeklenmiştir. Program sırasında alınan Basics of Visual Composition (Görsel Kompozisyonun Temelleri) dersinde öğrenildiği şekilde, oyundaki ekranlar tasarlanmıştır. Her ekrandaki unsurların doğru yerleştirilmeleri, renk uyumlarının sağlanması ve bilgisayar programları kullanarak yeni görsel oluşturabilme yetenekleri bu dersin öğrencilere kattığı değerlerden birkaçıdır ve bu projede kullanılmıştır. İnsanların sanatsal ruhları ve görsel kompozisyonların buna benzer görsel ürünlerdeki etkileri yadırganamaz. Bu projenin amaçlarından biri de bilgisayar mühendislerinin sadece teknik çözümler üretmekle kalmayıp yaratıcılığını ve sanatsal becerilerini gösterebileceğini az da olsa sunmaktır.

Projenin ikinci aşaması futbol simülasyonu ve bu simülasyonda kullanılan bir yapay zeka mimarisi tasarlamaktan oluşmaktadır. Proje dahilinde, Bullet adlı açık kaynaklı ücretsiz fizik motorunun sağladığı kütüphane kullanılarak başarılı sayılabilecek bir futbol simülasyonu temeli tasarlanmıştır. Simülasyon gerçeklenirken nesneye yönelik programlamanın avantajlarından yararlanmaya çalışılmıştır. Simülasyon top üzerinde farklı etkiler oluşturmayı mümkün kılmaktadır ve uluslararası futbol federasyonu olan FIFA kurallarına uygun olarak futbol oynanmasını sağlamaktadır.

Simülasyon oluşturulduktan sonraki adım olan yapay zeka tasarımında, davranış bazlı bir yapay zeka sistemi tasarlanmıştır. Bu sisteme uygun olarak futbolcuların sahada yapmayı tercih edebileceği eylemler belirlenmiştir. Bu eylemlerin bazılarından oluşan modüller oyuna eklenmiştir. Bu modüller ataklarda pozisyon alma, savunmada pozisyon alma, paslaşma, pas kontrol etme, top sürme modülleridir. Bu modüllere sahip olan oyun, kullanıcının bu modüllerle ilgili takım komutları ve futbolcu komutları vermesine imkan sağlamaktadır. Simülasyonun ve yapay zeka mimarisinin kaynak kodları, daha sonra yapılabilecek geliştirmelere ve yeni eylemlerin gerçeklenmesine olanak sağlayacak düzende yazılmıştır. Kaynak kodundaki önemli yerlerin bu raporda açıklanmasına ek olarak, kaynak kodunda yapılan işlemlerin ve oluşturulan yapıların yorum satırları ile açıklanmasına özen gösterilmiştir.

Gerçeklenen eylem modüllerine sahip olan oyun ilk oynanılabilir versiyonuyla birlikte test edilmeye hazır hale gelmiştir. İlk versiyon, futbol simülasyon oyunlarını oynamayı seven 5 adet kullanıcı tarafından test edilmiştir ve bu kullanıcıların görüşleri 7 soruluk bir anket ile alınmıştır. Bu görüşlere göre oyunun başarı kriterleri değerlendirildi ve ayrıca oyun, bu projenin gerçekleştirildiği sırada en başarılı futbol yönetim oyunu olarak bilinen Football Manager oyunuyla karşılaştırıldı. Kullanıcılardan alınan görüşlere göre bu projeyle oluşturulan Caretaker Manager'ın arayüzü iyi bir iş olarak değerlendirildi ve Football Manager'da olmayan bazı özelliklerin bu oyunda düşünülmesi beğenildi. Diğer yandan, oyunun ilk versiyonunda futbolcuların az sayıda eylemi gerçeklendiği için oyun eksik kaldı ve kullanıcılar tarafından eğlendirici bulunmadı. Futbolcuların hücum ve savunma oyunlarındaki pozisyon almaları başarılı bulunsa da, futbolcuların paslaşma ve top sürme yeteneklerinin geliştirilmesi gerektiği anlaşıldı. Ayrıca tasarlanan fakat gerçeklenmeyen eylemlerin gelecekteki muhtemel projelerle gerçeklenmesi oyunun gerçekçiliğini ve kullanıcıya hissettirdiği zevki artıracağı görüldü.

Özetle, bu proje geniş bir yazılım tasarlama ve gerçekleme tecrübesi açısından oldukça yararlı olmuştur. Ayrıca İstanbul Teknik Üniversitesi'ndeki Bilgisayar Mühendisliği lisans programında öğrenilen bilgi ve becerilerin bitirme projesinde sergilenmesi için olanakları bol bir proje olduğu görülmüştür. Proje kapsamında oluşturulan futbol simülasyonu tabanı ve yapay zeka mimarisi başarılı olabilecek kapsamlı bir oyuna sağlam bir temel sağlamıştır. Gelecekte yapılabilecek tamamen yapay zeka odaklı projelerle, karmaşık bir takım oyunu olan futbolun oldukça gerçekçi bir şekilde sanal dünyada oynanması gerçeklenebilir.

TABLE OF CONTENTS

1 INTRODUCTION

The purpose of this report is presenting and explaining the project of football management game, analyzing how the project proceeded and examining the results of the project.

1.1 Introduction for the Topic

Entertainment is a requirement for people. Humanity demands more and more entertainment in the new dynamic world of 21st century. Playing video games is one of the popular ways for people to enjoy themselves. It is not only the matter of entertaining the people. Video games are involved with training skills, creating simulation environment for real world events or testing intelligent agents in an artificial world.

Simulating sports is quite popular in the video game industry. Football simulation games are the most popular genre as football is the most popular sport in the world. According to Television Audience Report of 2014 FIFA World Cup Brazil™ produced by Kantar Media [1], approximately 1.013 billion viewers watched the final match of the biggest football competition for national teams. The popularity of football is reflected in the popularity of football games. Many football fans frequently imagine that they are the manager of their favorite football club. Football management games help football fans to achieve their dream in an imaginary world.

Video game users mostly have two kinds of abilities in video games about football. Some games allow user to control football players while some games allow user to behave as a manager who can build the team and give instructions for football players. In this project, the aim is to develop a video game where user can manage a football club for one football match.

1.2 Earlier Works on Football Management Games

There has been many commercial works on football management gaming such as FIFA Manager[2], Championship Manager[3] and Football Manager[4].

The most commercially successful football management games are Football Manager series developed by Sports Interactive and published by Sega. Football Manager series are the inspiration and an example work for this project. However, any kind of work of Football Manager series are not used in this project.

The similarities between the product of this project and Football Manager are using 2D bird’s eye view for the match screen and representing footballers with a circle.

2 PROJECT DESCRIPTION AND PLAN

2.1 Project Description

The objective of this project is developing a football management game with artificial intelligence. The program is decided to be written in C++ programming language. The game will work on Microsoft Windows operating systems.

Two free and open-source software development kits are used in this project:

- SFML (Simple and Fast Multimedia Library): SFML is a software development kit that provides simple application programming interface for some multimedia components such as graphics, audio and network.
- Bullet Physics: Bullet is a physics engine library that simulates soft and rigid body dynamics and handles collision detection.

SFML provides a 2D graphics module while Bullet Physics can operate on 3D world. Football is simulated in 3D but graphical user interface shows the football pitch with a bird's eye view.

The name of the game is Caretaker Manager. A caretaker manager is a temporary manager who takes charge of the management of a team for a short period.

2.2 Project Plan

2.2.1 Success Criteria

Success criteria for this project consists of three main criteria: entertainment, consistent user interface and intelligent agents.

- **Entertainment:** Users should be entertained while playing the game.
- **Consistent user interface:** User interface should be user-friendly. Errors and bugs should be avoided or handled well.
- **Correct simulation and intelligent agents:** Football players in the game should play the sport intelligently according to rules defined by FIFA (Fédération Internationale de Football Association). Footballers should reflect the instructions of user.

2.2.2 Project Scope

Project life cycle consists of three modules. The modules are defined as Graphical Design, Designing Match Engine and Testing.

- **Graphical Design:** Design and implementation of graphical elements in the game. Defining relationships between screens and implementation of
- **Designing Match Engine:** Design and implementation of match engine. Match engine is a common description for the combination of the simulation and AI engine.
- **Testing:** Testing the implemented parts of the match engine.

2.2.3 Time Management

The three modules defined in the project scope (Graphical design, match engine design and testing) are divided into smaller work packages. Durations of work packages and expected dates of milestones are described in the revised Gantt chart for the project (see Figure 2.1).

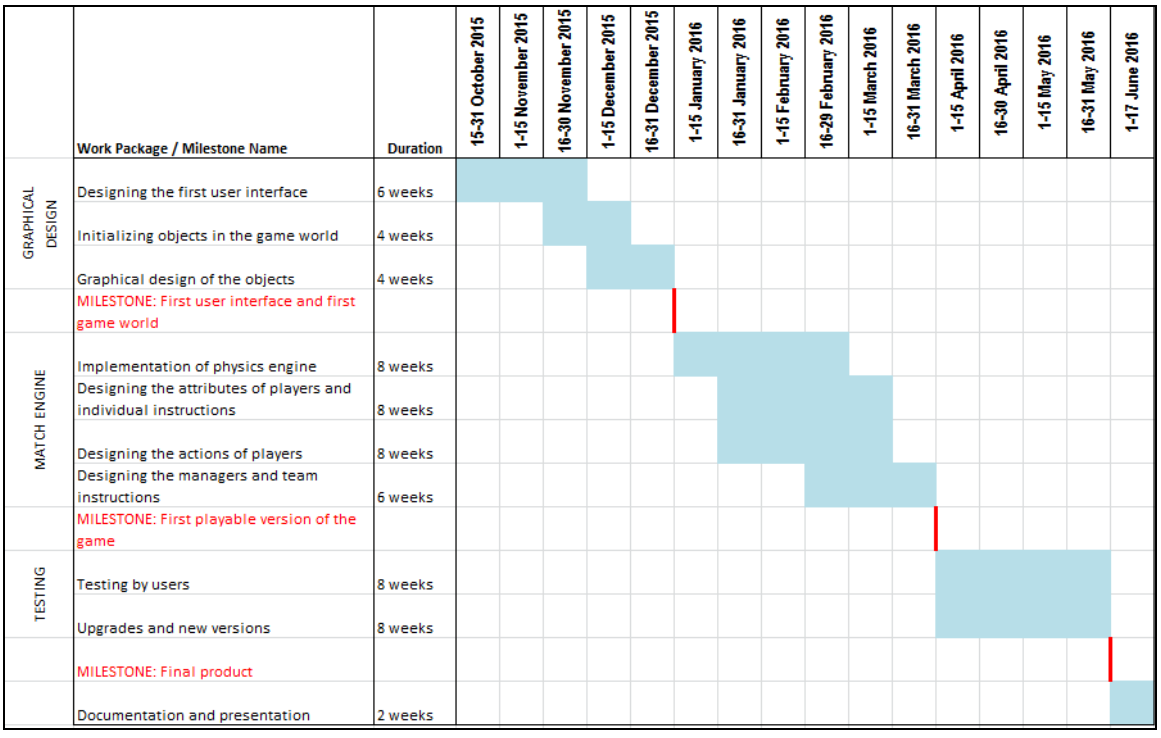

Figure 2.1 Gantt chart for the project

Graphical design for the game is completed on time.

Designing the actions of players is the only work package that is not completed on time. All possible actions of players are designed on 1 May 2016. The design of actions is discussed in the section ***"4.2 Model of Footballer Decisions"***.

First playable version of the game is released on 20 May 2016 after 8 actions are implemented. Implemented actions are ***ControlThePass***, ***DribbleForward***, ***RiskyPass***,

SimplePass, ***MoveToGatherBall***, ***TakeAttackingPosition, TakeDefendingPosition***. Users tested the first playable version of the game and contributed feedbacks for the results of the project. There were no upgrades for the first version of the game.

2.2.4 Risk Management

It is a big project that consists of very large source code. It is very important to keep the code clean and well structured. For reducing further risks, advantages of object oriented programming paradigm are used as much as possible. Modules are broke into small modules and each modules are tested both separately and together in the game environment.

It is indicated in the project plan that there was a chance that graphical design and match engine design might take longer time than expected. If first playable version of the game cannot be produced on estimated time, reduction of time for testing and upgrading was planned. If all time reserved for testing and upgrading is consumed by earlier works, testing and upgrading could be canceled as the project deadline cannot be changed.

As it is indicated in the section ***"2.2.3 Time Management"***, designing match engine took more time than expected. Therefore, duration of the work package ***"Testing by users"*** are reduced to 2 weeks. The work package ***"Upgrades and new versions"*** are canceled since the last work package ***"Documentation and presentation"*** cannot be canceled or delayed.

3 THEORETICAL INFORMATION

For the purpose of extending knowledge, theoretical information about game development and AI techniques are researched at the beginning of the project. In this chapter, life cycle of a game, game loop techniques and behaviour-based AI architectures are discussed.

3.1 Life Cycle of a Game

Before any approaches about game development, one should understand the life cycle of a game. A game is a program that is mainly responsible for three essential operations repeatedly. These operations are handling a series of user inputs, changing the state of the game and showing the changed state to the user. This cycle of three operations following each other is called game loop. Reflection of every game state on the screen is called frame.

Variables and resources for any program could be initialized at any time in the program. However, persistence and smoothness are important issues for games. It is advised that initialization of variables and resources should be done at the beginning of the life cycle, before the game loop.

Of course, games should end like any other program. Shutdown of the game should be only the choice of the user. Any errors that can occur in the life cycle of game should be handled in a way that game can continue to run if it is possible. Life of a game can because of an inevitable or fatal errors. In that case, user should be informed and the state of the game should be saved if it is possible.

3.2 Game Loop Techniques

For the main game loop, there are three kinds of organization techniques used in games [5]. Each game in the industry has its own technique and each technique has its own upsides and downsides.

First technique is the easiest one, simply executing every three essential operation sets once in each frame. This allows handling inputs and outputs simultaneously. (See Figure 3.1)

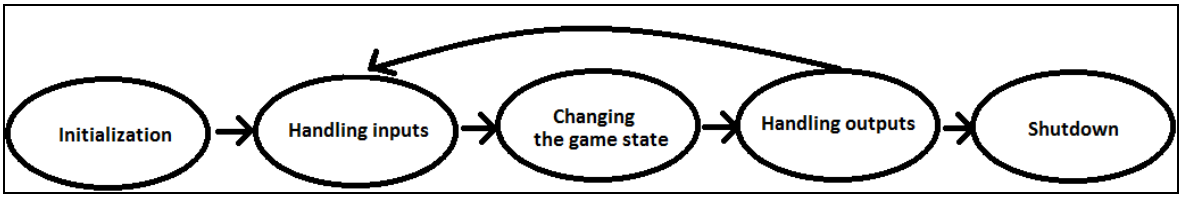

Figure 3.1: Simple structure for the game loop in the life cycle of games

Second technique is multithreaded main loops. Existence of CPU (Central Processing Unit cores and the chance of using both CPU and GPU (Graphics Processing Unit) in one

computer allows processing graphics and other things at the same time. Multithreaded games are significantly faster than single threaded games as different sections of the game life cycle can be executed at the same time in multithreaded games. CPU can continue to do its work while GPU is processing the graphical output of the earlier frame.

Third technique is a hybrid technique that merges the first two techniques for game loops. This advanced technique called cooperative multitasking is a system consist of many processes where each process uses a little CPU time for its operations.

AI agents are needed to be designed and developed in this project. Footballers have different behaviours on the football pitch. Therefore, research topics about AI are behaviour-based AI architectures and behaviour trees.

3.3 Behaviour-Based Artificial Intelligence Architectures

There are many Artificial Intelligence approaches for games and behaviour-based robotic architectures are common in game development because of the similarities between AI problems for games and AI problems for robotics. AI agents in games should perceive the world, decide its behaviour and act in the same world at each frame. Robots have the same problems and the solutions for the robotics could be used in game development too.

One example for behaviour-based AI architectures is the subsumption architecture which is one of the control mechanisms for autonomous robots having several goals and abilities. Some real world robots and some bots in games such as Quake II use this architecture [6]. Subsumption is the decomposition of an agent into different layers based on program flow and competences of the agent.

Another example for behaviour-based AI architectures is extended behaviour networks which were an architecture designed for playing RoboCup [6]. This architecture is composed of a number of behaviour modules, a set of goals and links between modules and goals. This is a goal-based approach and the most related approach to the approach needed in this project. Importance of every goal in the extended behaviour networks is defined by a strength value, static and context-independent importance of the goal, and a disjunction of propositions that is dynamic and a reflection of the game state on the goal.

4 ANALYSIS AND MODELLING

4.1 Class Relationships

The product of this project is a piece of software that consist of graphical user interface and football simulation. Model of the user interface and simulation is explained in this section. UML class diagrams and flow diagrams are introduced in this section.

4.1.1 Screen Classes

Games usually consist of many screens and transition between screens should be handled in such a way that user would not encounter excessive delay during transitions and changes. The memory usage of screens is another issue.

One of the most common tips about game programming is necessity of using object oriented design, separate classes for every screen. Every screen is rendered at some point of the game and every screen has its own resources to use when it is being rendered. This leads to an obvious solution, which is different classes inherited from an interface class with function Render( ) and resources.

Graphical user interface of this game consists of 4 different screens: Main Screen, Selection Screen, Tactic Screen, Match Screen. It was planned to create another screen named Option Screen. However, it is decided that only option for user is changing the language of the game which can be done in the main screen.

All screens are represented by separate classes that is inherited from an interface class: iScreen (see Figure 4.1).

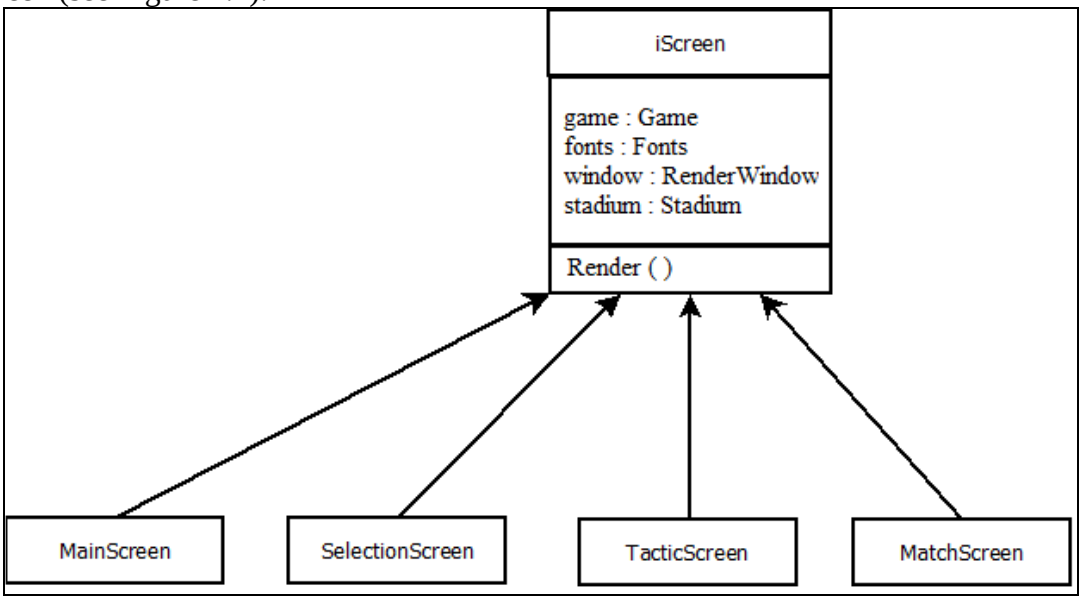

Figure 4.1: UML diagram for screen classes

Every screen class has access to one Fonts object, one Game object, one RenderWindow object and one Stadium object. All non-screen classes will be explained in the section *"**4.1.2 Important Class Relationships"***.

Possible user interaction flows between screens are represented visually in Figure 4.2.

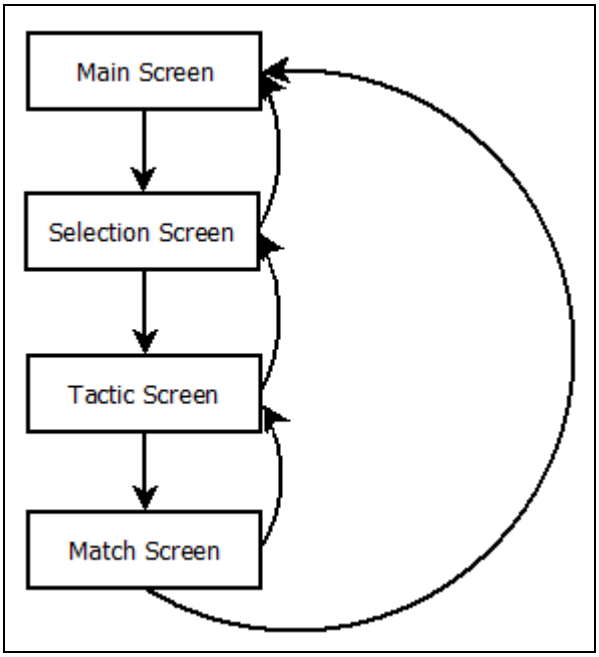

Figure 4.2: Possible user interaction flows between screens

4.1.2 Important Class Relationships

There are 11 different important classes modelled for this game. Their names, functionalities and relationships with the others are explained as follows:

- **Random:** It is the class that handles random number generation according to four kinds of distribution. Returning a random floating number or a random integer from uniformly distributed numbers is possible while normal distribution and weibull distribution are used too.

- **Language:** It is the class that owns the all text variables shown in the graphical user interface. There are two different functions named init_Turkish() and init_English(). These functions change all variables into predefined text of selected Language.

- **Footballer:** Represents a footballer with some personal properties such as name, surname, birth year and number. Holds skill values and style of play variables.

- **SquadPosition:** Represents a position in the squad. Exactly one footballer is assigned to each squad position. Every individual instruction of the manager is located in this object.

- **Club:** Represents a club and its properties. It includes teamwise functions about match engine. A club has a line up[1], substitutes[2] and reserves[3]. Starting eleven consists of 11 SquadPosition objects. Substitutes consist of maximum 12 SquadPosition objects. Reserves consist of maximum 10 SquadPosition objects. A club handles updating every player's action in every step of the game. Thus, every club has information about all possible attacking actions, defending actions and goalkeeping actions.

- **Action:** An interface class for all possible actions of football players. It has access to Random and Language objects. The reason for access to Random object is to use randomness during the selection and execution of actions. The reason for access to Language object is to provide a text value for the commentator on the stadium. Commentator is a string variable that is shown in the match screen to inform the user about the actions of footballers and the decisions of the referee.

  There are two functions of Action class. First function is checkPreconditions( ) which evaluates the availability and importance of the action for the footballer. Second function is execute( ) which executes the action for the footballer if it is selected. The list of actions will be introduced and explained in the next section of this chapter.

- **Stadium:** It is the class that keeps every object about the football match such as ball, clubs and all variables about the state of the match. It has a crucial function called StepBulletWorld( ) which lets the virtual world advance for the next frame.

- **Ball:** It represents the ball in the stadium and handles physical operations about the ball using the physics engine. Setting or getting the position of the ball, getting linear velocity or angular velocity of the ball, applying force or impulse to the ball are examples of physical operations.

- **Fonts:** It is the class that includes all fonts that are used in the game.

- **Game:** It is the class that holds information about the state of the game.

- **iScreen:** It is an interface class for all screens.

- **RenderWindow:** It is the most important class of the graphics module of the SFML. It handles the graphical operations such as drawing shapes into windows and displaying windows.

All relationships between classes can be seen in UML class diagram that is given in Figure 4.3.

[1] Eleven football players playing the match.
[2] Football players who are waiting outside of the field to replace one of the players on the field. The number of substitutes are determined by the competition. There are usually 7 substitute footballers in the competitions.
[3] Football players who are not selected for line up or substitutes.

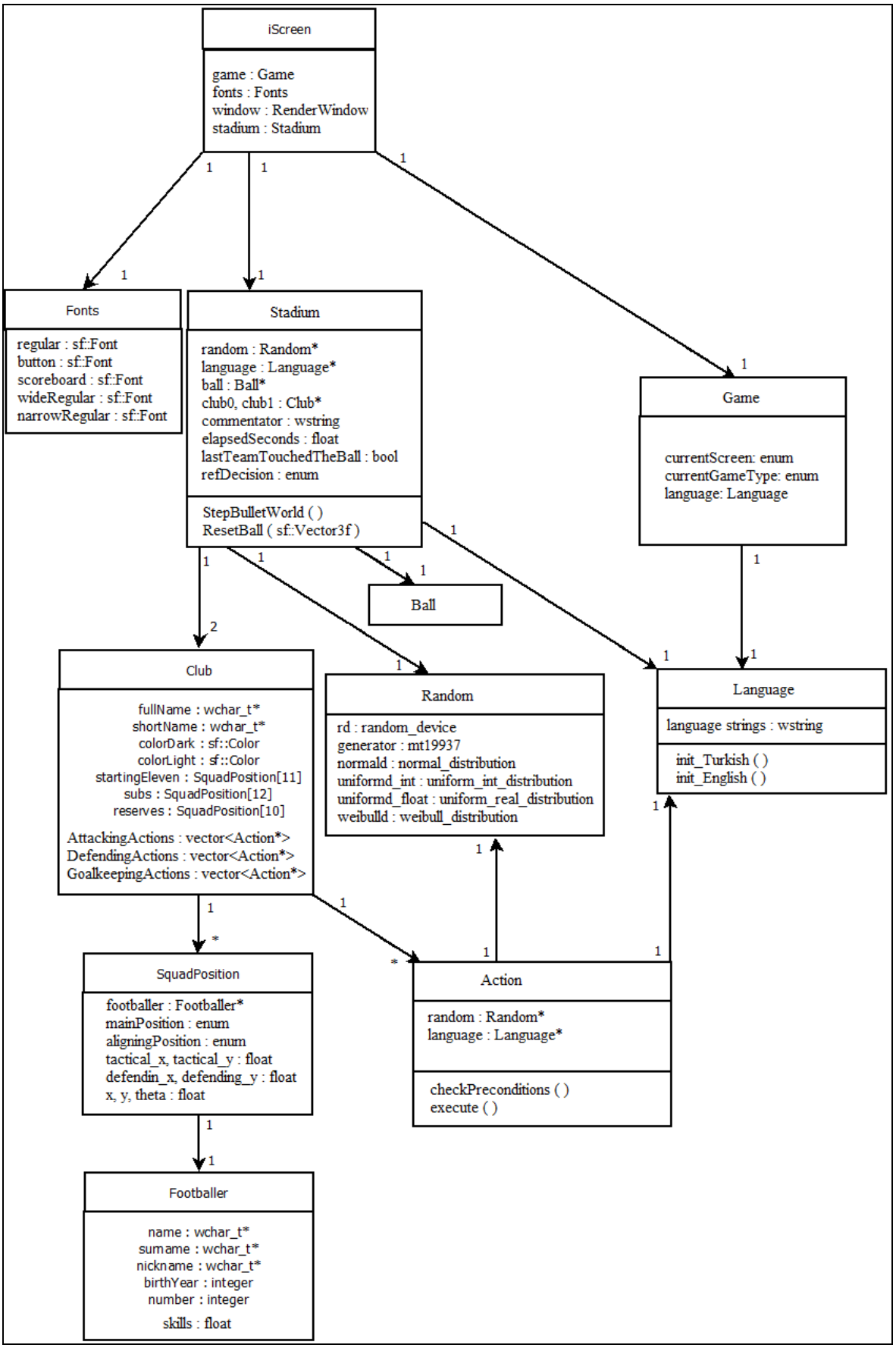

Figure 4.3: UML diagram for important classes

4.2 Model of Footballer Decisions

Footballer decisions are designed according to relationship between ball and team or footballer. Possible actions of a footballer are narrowed down into smaller parts for different circumstances. Footballer will choose their action in every moment according to their attributes, instructions given by the manager and the state of the match.

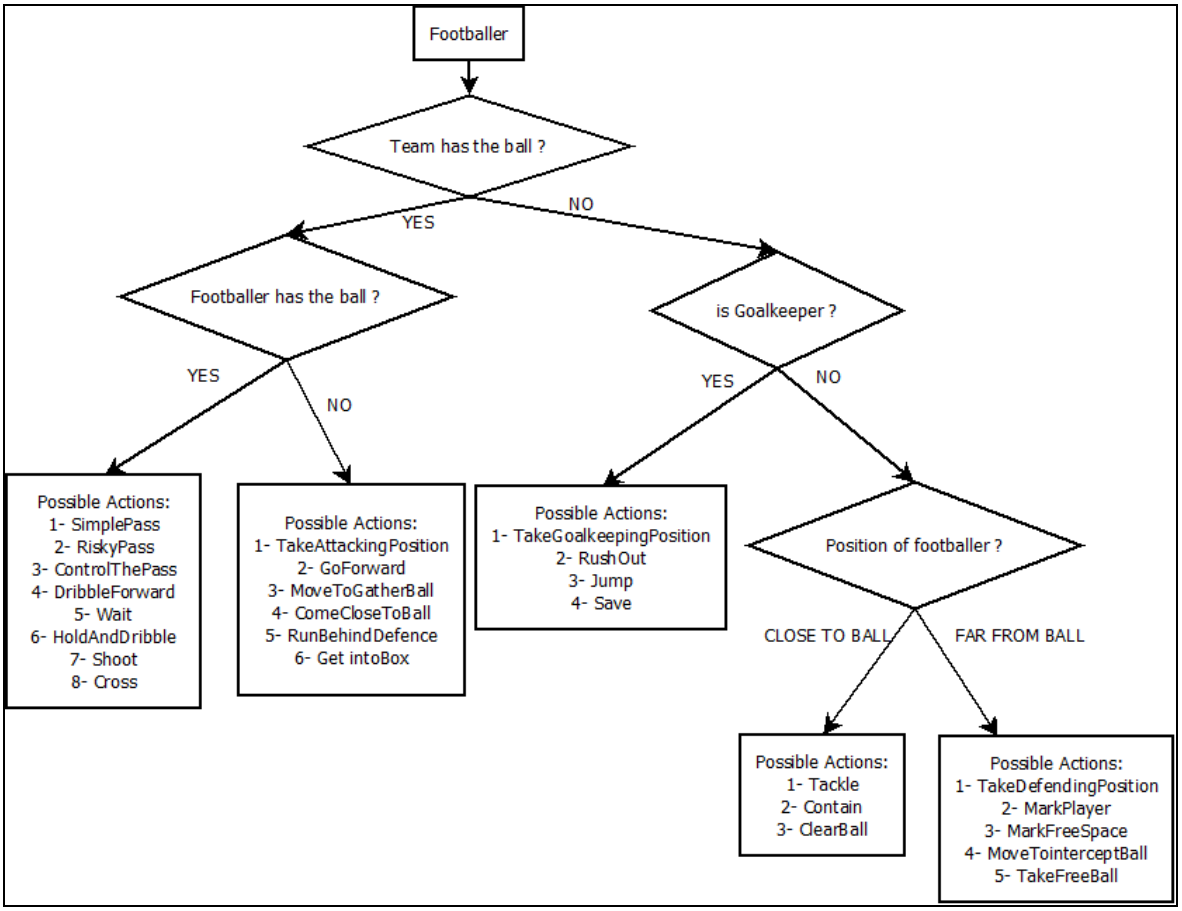

Figure 4.4: Model of footballer decisions

5 DESIGN, IMPLEMENTATION AND TESTING

5.1 Design of Screens

5.1.1 Main Screen

The Main Screen (see Figure 5.1) is the first screen of the game. The user clicks on the "Start a Game" button if he/she wants to start a new game. Main buttons disappear and game mode buttons appear (see Figure 5.2). User selects a game mode to start a new game.

- **Select a Scenario:** A game mode in which user can select a scenario to play. Clubs and all details of the football match are determined by the scenario. Scenarios are historical football matches or fictional scenarios.

- **Random match:** Clubs and all details of the football match are determined randomly by the game.

- **Custom match:** Clubs and all details of the football match are determined by the user.

There are only two football teams in the current version of the game when this report is written. They are Leicester City Football Club and Manchester United Football Club. Game mode choices are not removed so that they can be used in further studies on this project.

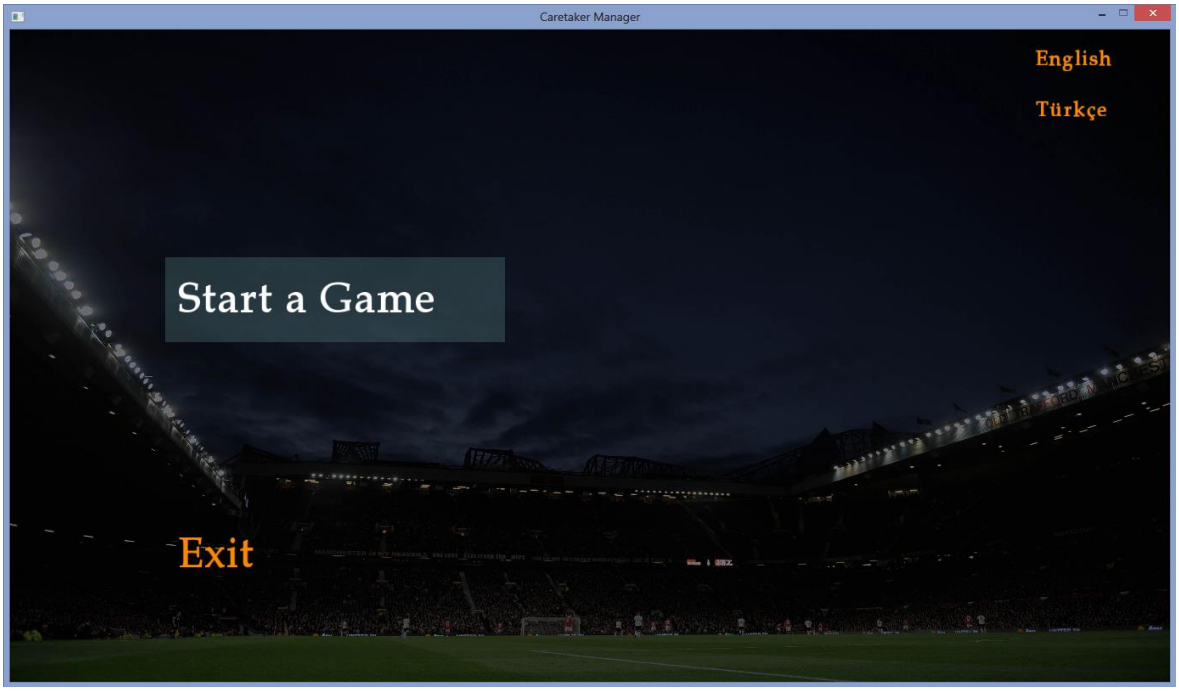

Figure 5.1: Main Screen

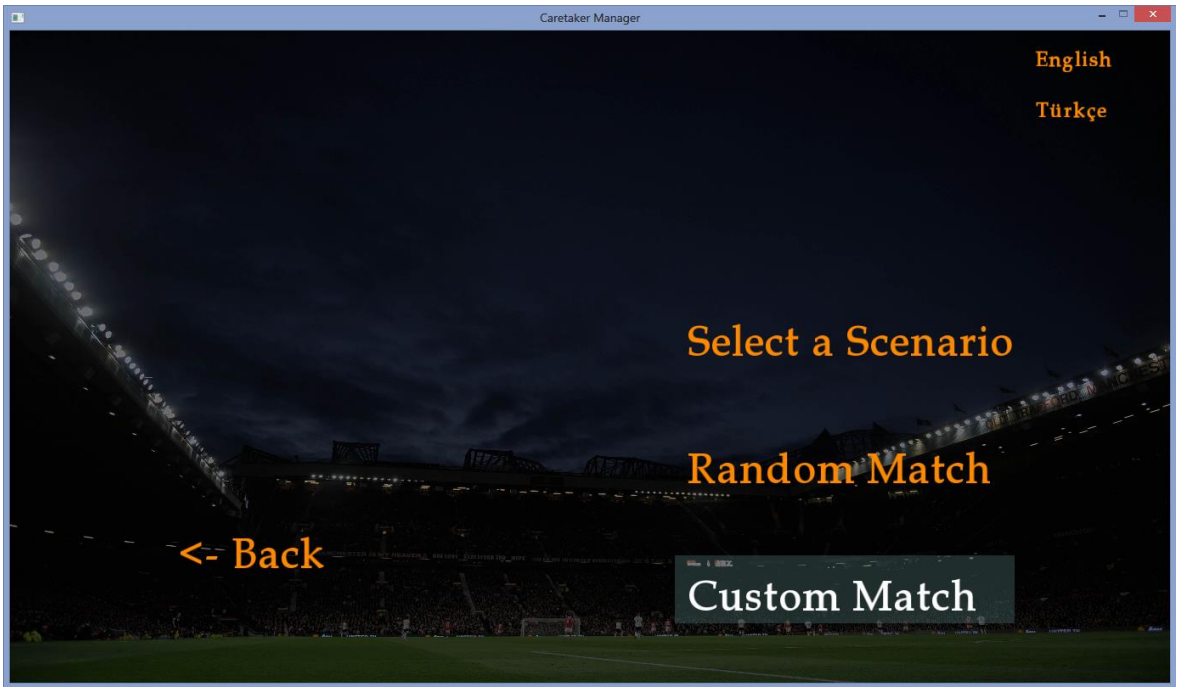

Figure 5.2: Game modes

The choice of language is available in the top-right corner of the main screen. When user clicks on a language button, all text shown in the game are replaced by new text strings of the chosen language.

5.1.2 Tactic Screen

After the ***Main Screen***, user encounters the ***Tactic Screen*** where instructions to the team or footballers can be done. There are 8 different panels for the tactic screen. Selecting one of them from the buttons on the left-hand side changes the content of the tactic screen. User can finish changing tactics and change the screen to ***Match Screen*** by clicking on ***Confirm*** button.

"Team Sheet" panel is the default panel for the ***Tactic Screen*** and it is accessible from the panel on left-hand side. The panel shows the line up, substitutes and reserves for the club. Any two footballer can be swapped by dragging with the mouse (see Figure 5.3).

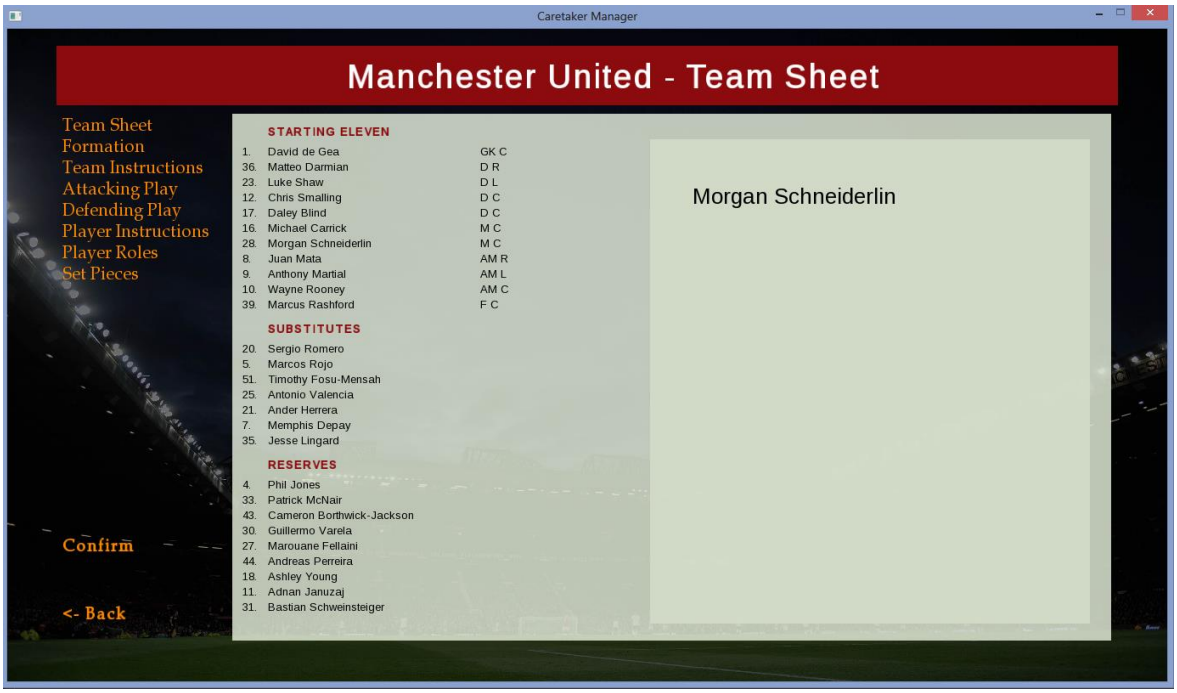

Figure 5.3: Tactic Screen: Team Sheet

"Formation" panel is accessible from the panel on left-hand side of the ***Tactic Screen***. Opaque and big circles are the tactical positions of footballers that are used for attacking positioning during the match. Transparent and small circles are the defending positions of the footballers (see Figure 5.4).

Any footballer can be selected by mouse button. Tactical position of selected footballer can be moved by arrow keys on the keyboard. Goalkeeper cannot move upwards or downwards. All outfield footballers can be moved as long as the desired destination is inside of the pitch. Defending position of selected footballer can be moved by **W**, **A**, **S**, **D** keys on the keyboard.

Figure 5.4: Tactic Screen: Formation Panel

"Team Instructions", ***"Attacking Play"***, and ***"Defending Play"*** are the tactic panels that provide an interface for teamwise instructions valid for whole team. Every instruction for the user team can be changed using minus (–) or plus (+) buttons located in both sides of the instruction bars (See Figure 5.5).

Figure 5.5: Tactic Screen: An example of team instructions

"Player Instructions" is the tactic panel for the individual instructions for footballers. User can change the footballer by clicking on the arrow buttons located in both sides of the footballer name. According to chosen footballer, the instructions is shown in the panel and can be changed by clicking on minus (–) or plus (+) buttons located in both sides of the instruction bars (see Figure 5.6).

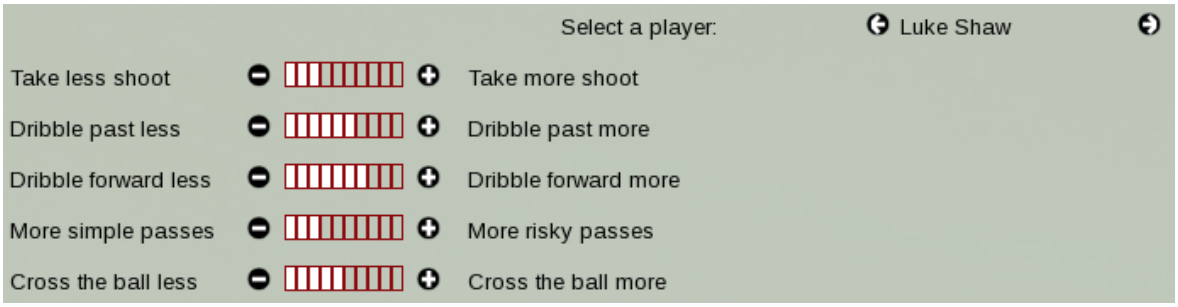

Figure 5.6: Tactic Screen: An example of player instructions

"Set Pieces" is the tactic panel for changing the footballers that are going to take free kicks, corner kicks and penalty kicks. User can change the chosen footballer by clicking on the arrow buttons located in both sides of the footballer name (see Figure 5.7).

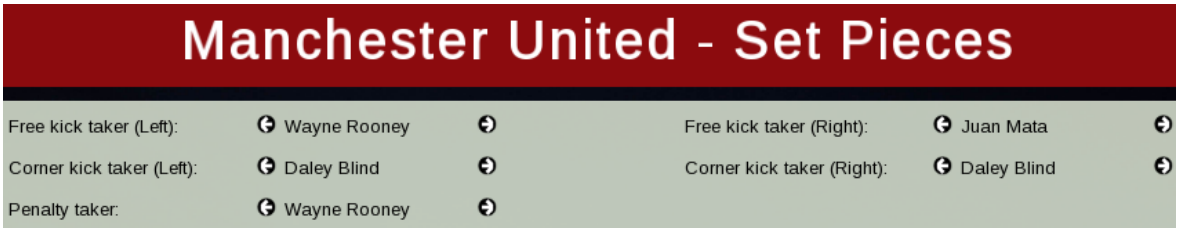

Figure 5.7: Tactic Screen: Set Pieces

5.1.3 Match Screen

The ***Match Screen*** is rendered after tactical changes are completed. This is the screen where the football match is shown to the user. There are three statistical information panels on the left-hand side. Scoreboard is located at the top of the screen. There are two buttons that a user can interract with. The ***"START/PAUSE"*** button starts or pauses the game. The

"CHANGE TACTICS" button changes the screen to the Tactic Screen and allows user to change tactics during the match (see Figure 5.8).

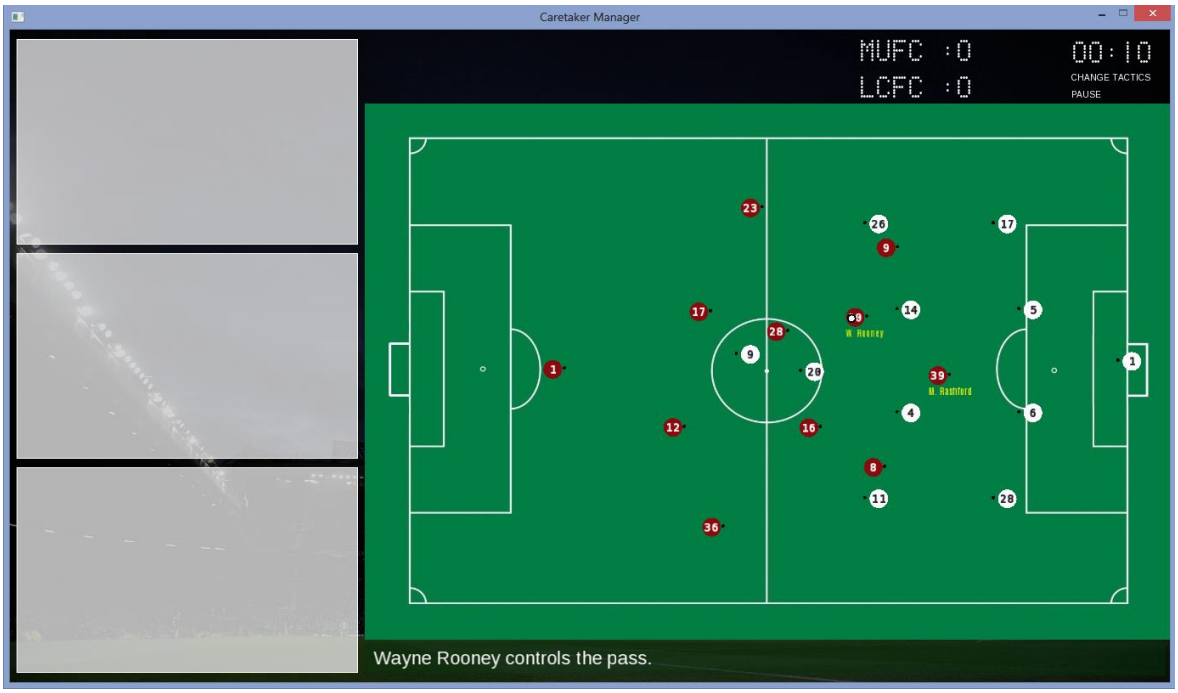

Figure 5.8: The Match Screen

5.2 Implementation of Graphical User Interface

The graphical user interface is designed and implemented using SFML library functions. Main function of the program starts with the initialization of visual window that is going to appear in the operating system. The wndow for the game has a fixed size of 1366x768. The second operation is the initialization of objects of important classes such as ***Random***, ***Language***, ***Game***, ***Fonts*** and ***Stadium*** (See Figure 5.9).

```
int main()
{
    sf::RenderWindow window(sf::VideoMode(1366, 768), "Caretaker Manager", sf::Style::Close);
    window.setPosition(sf::Vector2i(0,0));
    window.setFramerateLimit(60);
    window.setKeyRepeatEnabled(false);

    Random random;
    srand(time(NULL));
    Language *language = new Language;
    Game game(language);
    Fonts fonts;
    Stadium* stadium = new Stadium(&random,language);
```

Figure 5.9: Initialization of window and important classes

All screen objects are initialized after other objects. Screen classes are derived from base class called ***iScreen***. A vector contains every screen class. Game starts with rendering main screen and continues with other screens. Return value of the ***Render( )*** function defines the next screen that is going to be rendered. When the return value is ***exitSignal***, program is terminated (See Figure 5.10).

```
std::vector<iScreen*> Screens;
int screen = DEBUG_STARTMATCH?3:0;

MainScreen screen0(&window,&game,&fonts,stadium);
Screens.push_back(&screen0);
SelectionScreen screen1(&window,&game,&fonts,stadium);
Screens.push_back(&screen1);
TactiiScreen screen2(&window,&game,&fonts,stadium);
Screens.push_back(&screen2);
MatchScreen screen3(&window,&game,&fonts,stadium);
Screens.push_back(&screen3);

while (screen >= 0 && screen!=exitSignal)
{
    screen = Screens[screen]->Render();
}
window.close();

return EXIT_SUCCESS;
```

Figure 5.10: Initialization of screens and rendering screens

A screen is rendered by a while loop in the ***Render( )*** function. Interrupts for a user input are handled by polling events from the window object of SFML. According to the type of the event, different functions that performs necessary changes are called. After polling event and changing the state of the game, window from the earlier frame is cleared and all graphical elements of the game is drawn into window. A loop of the screen is completed with window being displayed. An example of ***Render( )*** function can be seen in Figure 5.11.

```
e_Window MatchScreen::Render()
{
    float mouse_x, mouse_y; wstring ballPosition = L"";
    while (window->isOpen())
    {
        while (window->pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
            {
                if (RenderClosing())
                    return exitSignal;
            }
            else if (event.type == sf::Event::MouseButtonPressed)
            {
                if (event.mouseButton.button == sf::Mouse::Left)
                {
                    if( ControlMousePressed(event.mouseButton.x,event.mouseButton.y) )
                        return tacticScreen;
                }
            }
        }
        if (!paused) stadium->StepBulletWorld();
        window->clear();
        drawBackground();
        drawStadium();
        drawPanels();
        drawScoreboard();
        drawButtons();
        drawCommentator();
        window->display();
    }
    return exitSignal;
}
```

Figure 5.11: Render( ) function for the Match Screen

5.3 Simulating Football

The football match is simulated by ***StepBulletWorld( )*** function of ***Stadium*** class. This function is called before rendering every next frame of the game.

The first duty of the function is to check all decision possibilities of the referee. In every decision, necessary changes are made (see Figure 5.12). Game starts with a kick-off. The initial decision of the referee is kick-off. If the ball crosses the line between goal posts, referee decides it is a goal and score of the scorer team is increased by one. Game starts again with kick-off. If the ball crosses one of touchlines, it is a throw in. A goal kick or a corner kick is awarded according to last team that touches the ball if the ball goes out of field by crossing other edges of the field.

The second duty of the function is changing the match state according to decision that referee gives (see Figure 5.13). This change is performed after some delay that counted with ***countdown_refDecision*** value because the user has to see the world changing for a couple of seconds before new adjustments in the football field is in use.

The third duty of the function is to make both clubs update their team variables and individual variables according to the state of the match (see Figure 5.14).

```
void Stadium::StepBulletWorld()
{
    if (countdown_refDecision>0) // don't change the state yet. visual delay is needed.
        countdown_refDecision--;
    else
    {
        int goal_check = IsGoal();
        if (goal_check>=0)
        {
            if (goal_check) score1++;
            else score0++;
            refDecision = DECISION_GOAL;
            countdown_refDecision = 180;
            commentator += language->COMMENTATOR_00006;
            club0->sinceBallIsPassed=0; club1->sinceBallIsPassed=0;
            gameIsOn = false;
        }
        else if (IsThrowIn())
        {
            countdown_refDecision = 180;
            refDecision = DECISION_THROWIN;
            commentator += L" " + language->COMMENTATOR_00001;
            club0->sinceBallIsPassed=0; club1->sinceBallIsPassed=0;
            gameIsOn = false;
        }
        else if (IsGoalKick())
        {
```

Figure 5.12: StepBulletWorld: example of referee decisions

```
if (countdown_refDecision == 0)
{
    switch(refDecision){
    case DECISION_NO:
        cout << "Error: Countdown is over but there is no ref decision." << endl;
        break;
    case DECISION_GOAL:
        matchState=kickoff;
        break;
    case DECISION_CORNERKICK:
        matchState=cornerKick;
        break;
    case DECISION_GOALKICK:
        matchState=goalKick;
        break;
    case DECISION_THROWIN:
        matchState=throwIn;
        break;
    }
    countdown_refDecision = -1;
}
```

Figure 5.13: StepBulletWorld: example of state changes

```
if (matchState==openPlay)
{
    if (whoHasTheBall)
    {
        club1->updateAttackingPlay();
        club0->updateDefendingPlay();
    }
    else
    {
        club0->updateAttackingPlay();
        club1->updateDefendingPlay();
    }
}
else if (matchState==kickoff)
{
    if (elapsedSeconds>60)
    {
        elapsedSeconds += random->get_int(50,80);
        commentator = language->COMMENTATOR_00003;
    }
    else
        commentator = language->COMMENTATOR_00002;

    ResetBall(sf::Vector3f(centerSpot.x,0.0,centerSpot.y));
    club0->takeKickOffPositions(!whoHasTheBall);
    club1->takeKickOffPositions(whoHasTheBall);
    matchState=openPlay;
    gameIsOn = true;
}
else if (matchState==goalKick)
{
```

Figure 5.14: StepBulletWorld: updating clubs and the simulation world

Last duty of the StepBulletWorld function is to step the dynamics world by one frame (See Figure 5.15). Dynamics world is the virtual world that is run by Bullet physics engine.

```
dynamicsWorld->stepSimulation(1/60.0,10);
elapsedSeconds += 1/60.0;
```

Figure 5.15: StepBulletWorld: updating the physics engine.

5.4 Bullet Physics Engine

Bullet physics engine is used for handling ball movements and handling collisions between ball and goal posts.

Three kinds of objects are defined in the Bullet world. Only the ball is a moving object among them. Other two kinds of objects are immovable.

The gravity of the world is set to 9.8 downwards. Ground is a static plane shape with the restitution of 0.79, hit fraction of 1.5 and rolling fraction of 12.0.
3 goal posts for each goal, total of 6 goal posts are located in the football pitch. The width of the goal is 7.32 meters and the height of the goal is 2.44 meters. Radius of every goal post is 0.22 meters. Dimensions of objects and all pitch measurements are determined according to standard regulations of FIFA (International Federation of Association Football).

While meter is the unit of length in the physics engine, pixels were needed for rendering the world in 2D. Mapping between physics engine and drawn window is performed in the drawing functions according to proportion of two domains.

5.5 Decision Making System

All possible actions for footballers are implemented in different class definitions derived from a base class called ***Action***. Every ***Club*** object is responsible for handling the actions of footballers in every frame. Thus, ***Club*** holds all possible actions under three vectors called ***attackingActions***, ***defendingActions***, ***goalkeepingActions*** (see Figure 5.16).

```
typedef std::vector<Action*> ActionVector;
typedef std::map<float,Action*> ActionMap;
typedef std::pair<float,Action*> ActionPair;

class Club
{
    ActionVector attackingActions;
    ActionVector defendingActions;
    ActionVector goalkeepingActions;
    void initializeActions();
```

Figure 5.16: Action vectors of the class Club.

Clubs update states of their footballers using different functions determined by the state of the match (see Figure 5.14). If the match is in a new state such as kick off, goal kick, corner kick, free kick or throw in, one of the 5 specialized functions are called (see Figure 5.17). ***updateAttackingPlay( )*** and ***updateDefendingPlay( )*** functions are called in case of open play.

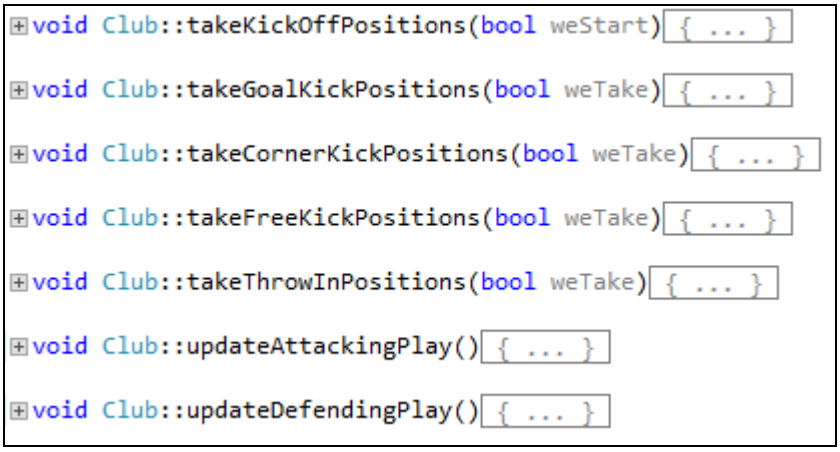

Figure 5.17: Club functions regarding necessary updates of footballers

Every function about set pieces only determine the position of the footballers in the world. However, open play functions are involved with decision selection algorithm. Clubs need to update team values such as current defensive line, current leftmost position of the team and current uppermost position of the team (see Figure 5.18). These are the clubwise values that are used by action classes.

Defensive line of the team is calculated using the x & y positions of the ball and ***defensiveLine*** instruction which is defined by the user. Defensive line describes the horizontal position of the most defensive outfield player in the team. Defined logarithmic function provides a decreasing change of defensive line over the ball position. The more advanced position the ball has, the less encouraged footballers to take defensive line much forward.

All footballers have a tactical position defined by the formation that user decided. These tactical positions are used for attacking movements and they consist of ***tactical_x*** and ***tactical_y*** values between 0 and 1. ***currentLeftMost*** is the horizontal position of an imaginary footballer having a ***tactical_x*** value of 0. ***currentUpperMost*** is the vertical position of an imaginary footballer having a tactical_y value of 0. currentWidth is the maximum width of the team which is possible by having two footballers with having tactical_y values of 0 and 1. Attacking and defending width of the team would be different and they are all instructions of the user.

The effects of team values is explained under the section ***"5.6.1 Default Attacking and Defending Positions"***.

```
void Club::updateAttackingPlay()
{
    if (!stadium->gameIsOn) return;

    float stadium_width = stadium->playArea.width;
    float stadium_height = stadium->playArea.height;
    float goalEffect = (onTheRightGoal)?(-1.0):(1.0);
    sf::Vector2f ball_pos = stadium->ball->getPosition2();
    sf::Vector2f relative_ball_pos = ball_pos;
    if (onTheRightGoal)
    {
        relative_ball_pos.x = stadium->playArea.width - relative_ball_pos.x;
        relative_ball_pos.y = stadium->playArea.height - relative_ball_pos.y;
    }

    // Update currentDefensiveLine
    currentDefensiveLine = 0.12 * log(31.0*(relative_ball_pos.x/stadium->playArea.width)+1.0) /
        ( log(2) * ((1.0/-0.9) * (defensiveLine-1.0) +1.0) );
    if(onTheRightGoal) currentDefensiveLine = 1.0 - currentDefensiveLine;
    currentDefensiveLine *= stadium->playArea.width;

    // Update currentLeftmost
    float minX = 2.0;
    for (int i=1; i<11; i++)
    {
        if (lineUp[i]->tactical_x<minX) minX = lineUp[i]->tactical_x;
    }

    float howFar_defLine = onTheRightGoal?currentDefensiveLine:(stadium->playArea.width-currentDefensiveLine);
    currentLeftmost = currentDefensiveLine - goalEffect * ( (minX * howFar_defLine) / (0.9-minX));
    currentWidth = 0.5 + 0.4 * attackingWidth;
    currentUppermost = ball_pos.y * (1.0-currentWidth);
    currentWidth *= stadium_height;
```

Figure 5.18: updateAttackingPlay( ) - Updating team values.

After team values are updated, every footballer decides his next action. Since it is an attacking play, all attacking actions are evaluated and decision is made according to return values of ***checkPreconditions( )*** functions of all attacking actions. Each footballer chooses the action with maximum return value of all. Action is executed by the ***execute( )*** function of the action (see Figure 5.19).

Each action class has a ***checkPreconditions( )*** function that evaluates the current state of the footballer and the current state of match. This function returns the weight of the action that is also affected by randomness.

Each action class has an ***execute( )*** function that changes the state of the footballer and the state of the match. Every effect of the action is applied in this function. It also changes the commentator text that is defined in ***Stadium*** class for showing updates about the match like a real commentator. The content of an Action class can be seen in Figure 5.20.

```
SquadPosition* player;
for (int j=0; j<11; j++)
{
    player = lineUp[j];
    ActionMap possibleActions;
    for (int i=0; i<attackingActions.size(); i++)
    {
        float weight = attackingActions[i]->checkPreconditions(player,stadium);
        if (weight>0)
            possibleActions.insert( ActionPair(-1.0*weight,attackingActions[i]) );
    }
    if (possibleActions.size()>0)
        possibleActions.begin()->second->execute(player,stadium);
}
```

Figure 5.19: updateAttackingPlay( ) - Evaluating possible attacking actions

```
class Action
{
public:
    Random* random;
    Language* language;

    Action(Random* r,Language* lang):random(r),language(lang){};
    virtual int checkPreconditions(SquadPosition*, Stadium*)=0;
    virtual bool execute(SquadPosition* , Stadium*)=0;
};
```

Figure 5.20: Action class

5.6 Implementation of Actions

A number of possible actions are defined for a footballer. Each action is implemented with a new class that is derived from ***Action*** class. The object oriented programming paradigm allows the possibility of implementing and testing new actions independently.

Decision making system relies on the variety of actions. The more actions defined, the more realistic the simulation gets. The scope of this project is not limited to constructing the game environment and the match simulation. It also includes the implementation of some actions. In this section, implemented actions will be explained using visual demonstrations.

5.6.1 Default Attacking and Defending Positions

TakeAttackingPosition and ***TakeDefendingPosition*** are the default actions for the footballers that don't have the ball and the footballers that are not near the ball. Importance of these actions are always returned by the value 100 while other actions are trying to make the footballer decide on a different action by returning different importance values.

Execution of the action TakeAttackingPosition and TakeDefendingPosition consist of three stages (see Figure 5.21).

First of all, target position is calculated using team values such as currentLeftmost, currentUppermost and currentWidth. The calculations of team values are explained under the section ***"5.5. Decision Making System"***. The target position of the footballer is basically determined by their tactical positions with the restriction of an imaginary frame that is defined by team values. An example of the graphical representation of the frame and default positions can be seen in Figure 5.22.

Second stage is calculating the speed of player. Footballers have different physical abilities such as Acceleration, Maximum Speed and Agility. While acceleration defines the ability to change the speed, maximum speed defines the maximum speed that the footballer can reach. Agility is a term for the ability of changing directions easily. These three physical abilities of the footballer and latest speed from the earlier frame defines the current speed of the footballer (see Figure 5.21).

Final stage of the default positioning actions is to make the footballer go towards desired position and to make necessary changes in the state of the footballer.

```
// Calcuate position
float target_x, target_y;

float howFar_leftmost = (hisclub->onTheRightGoal)?
    (hisclub->currentLeftmost):(stadium_width - hisclub->currentLeftmost);
target_x = hisclub->currentLeftmost + howFar_leftmost * player->tactical_x * goalEffect;

float y_pos = (hisclub->onTheRightGoal)?(1.0-player->tactical_y):player->tactical_y;
target_y = hisclub->currentUppermost + hisclub->currentWidth * y_pos;

stadium->makeItInside(&target_x,&target_y);

// Calculate speed.
float target_theta = atan2( (target_y-player->y), (target_x-player->x) );
float difference_theta = abs(target_theta - player->theta);

while (difference_theta > M_PI){ difference_theta -= 2*M_PI; }
while (difference_theta < -M_PI){ difference_theta += 2*M_PI; }

float max_speed = footballer->getMaxSpeed();
float agility_effect = 0.2 + 0.3 * footballer->agility / 100;
max_speed *= (agility_effect + (1-agility_effect)* abs(M_PI-difference_theta) / M_PI);

float change_of_speed = footballer->getMaxAcceleration();
travelOverFrame = player->lastMotion + change_of_speed;
if (travelOverFrame>max_speed) travelOverFrame=max_speed;

// Go towards the calculated position
player->lastMotion = player->goTowardsPoint(target_x,target_y,travelOverFrame);
player->theta = (player->club->onTheRightGoal)?M_PI:0;
```

Figure 5.21: Execution of the action ***TakeAttackingPosition***

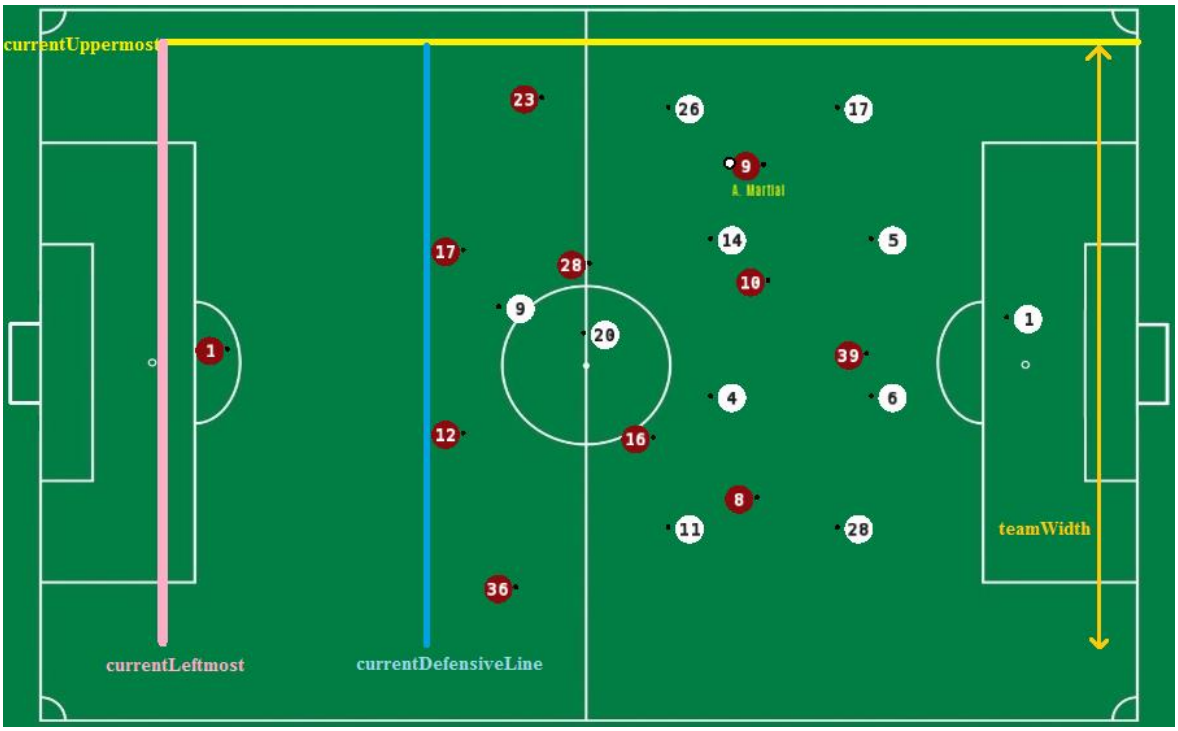

Figure 5.22: Graphical representation of team values and default positions

5.6.2 Passing

One of the most important goals of the footballers is to get the ball forward for the purpose of increasing the score of their team. Footballers have to pass the ball to each other in order to get the ball forward quickly.

The routine of passing is implemented with 4 actions in this game:

1. **SimplePass:** The action of attacking footballer passing to a teammate who is most probably close to the source of the pass. The height of these passes are usually low. Footballer always tries to perform such a pass that nobody from the opposite team can intercept. The purpose of this pass is keeping possession and not giving the ball away. The preconditions of this action are having the ball nearby and being a member of attacking team at the current state.

2. **RiskyPass:** The action of attacking footballer passing to a teammate who has most advanced position in the field. The purpose of this pass is getting the ball forward and creating a scoring chance. The preconditions of this action are having the ball nearby and being a member of attacking team at the current state.

3. **MoveToGatherBall:** The action of an attacking footballer moving towards a position to get the ball. If a pass is given from a teammate, every footballer checks the availability to intercept the ball in time. The preconditions of this action are anticipating the availability of future interception and being a member of attacking team at the current state.

4. **ControlThePass:** The action of an attacking footballer controlling the ball after a pass is given from his teammate. The preconditions of this action are having the ball nearby and being a member of attacking team at the current state.

Decision making between trying a simple pass or trying a risky pass depends on the instruction that manager gives and an two random numbers. Weight of both action modules are return when they are evaluated. The formula for the calculation of weights can be seen in Figure 5.23. The variable named ***freq_riskyPassing*** is the instruction that manager gives. This is a floating number between 0 and 1.

Action	**Formula**
`SimplePass`	`750 + 10*(0.5-player->freq_riskyPassing) + random->get_int(0,100)`
`RiskyPass`	`750 - 10*(0.5-player->freq_riskyPassing) + random->get_int(0,100)`

Figure 5.23: Formula for return values of checkPreconditions( ) of the actions SimplePass and RiskyPass

The action of passing is executed by applying an impulse to the ball. The ideal angle is calculated by using the inverse tangent function after taking the difference between the position of the source and the position of the destination. However, the ideal angle is not the angle of impulse for these actions. Footballers are not perfect and passes can be wrong placed. In order to implement the factor of human error, angle is a bit changed with a random number that is taken from a normal distribution. Standard deviation of the normal distribution is determined according to passing skill of the footballer. An example of how errors are handled, Figure 5.24 demonstrates the change of angle in the action SimplePass.

```
float angle = atan2(dify,difx); // it is the ideal angle.
float sd = 0.4 - (0.35 * player->get_footballer()->shortPassing * 0.01);
angle += random->get_normald(0,sd);
```

Figure 5.24: Implementation of errors in the action SimplePass

It is not only the angle that is needed to be calculated for passing. The power of the pass should be calculated as well. The correlation between the amount of impulse and the distance between source and destination is examined by doing some experiments on the physics engine. 12 different impulse values are applied to the ball and the distances that ball reaches with a low speed are observed. The blue points in the Figure 5.25 represents the observed distances. A logarithmic function is created for calculating the necessary impulse that is needed to applied in order to make the ball go to some distance (see Figure 5.26). A direction vector from the source to the destination is the only parameter of this function. The necessary impulse is calculated by the formula “impulse = (100*log(1+distance/450))+1”. The function returns the impulse vector that will take the ball to desired location on the football pitch.

Distance (m)	Expected Impulse	Formulated Impulse
1	1	1,221975674
4	2	1,884961528
8	3	2,762160135
12	4	3,631730832
17	5	4,70816749
21	6	5,561051125
26	7	6,617027147
31	8	7,661968734
38	9	9,106782309
46	10	10,7328344
53	11	12,13425873
59	12	13,32004338

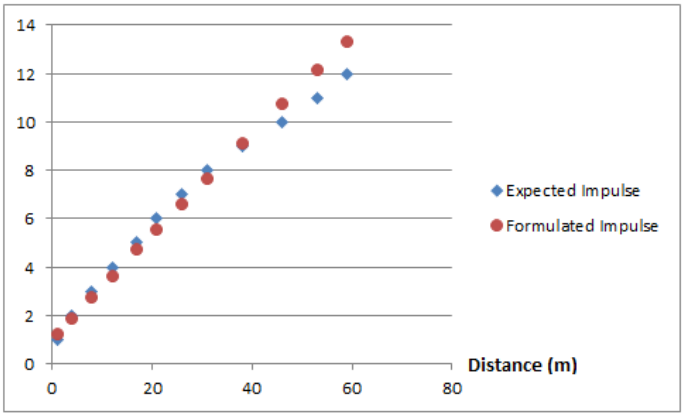

Figure 5.25: Comparison between expected impulses and formulated impulses.

```
sf::Vector2f CalculatePassImpulse(sf::Vector2f direction)
{
    float impulse;
    float distance_direction = sqrt( std::pow(direction.x,2.0) + std::pow(direction.y,2.0) );
    impulse = (100.0*log((distance_direction/450.0)+1.0)) +1.0;

    return sf::Vector2f(direction.x*impulse/distance_direction, direction.y*impulse/distance_direction );
}
```

Figure 5.26: Calculating the approximate impulse for passing the ball.

5.6.3 Dribbling

Footballers don't always try to get the ball forward with passing. Dribbling, running with the ball, is another action that a footballer can take.

DribbleForward is the first dribbling action of the game. The preconditions of this action are having the ball nearby, being a member of attacking team at the current state and finding a free space in front of him.

Checking if there is a free space in front of the footballer is done by a function called ***isPointBetween( )*** which takes the test point, two points and a margin value as parameters (see Figure 5.28). The function calculates the slope of the line between two points and the slope of a line perpendicular to the earlier line. The purpose of these operations are constructing 4 points that define a tetragon of the dribbling space (see Figure 5.27).

Checking if a test point is inside a tetragon is done by a function called ***isPointInTetragon( )*** which takes the test point and the 4 points of the tetragon as parameters (see Figure 5.28). This function uses the angle summation method to find the solution. The angle summation method is checking the summation of the angles between the test point and each edge's end-points if it is near the 2 times PI which means the test point is inside the tetragon. [7]

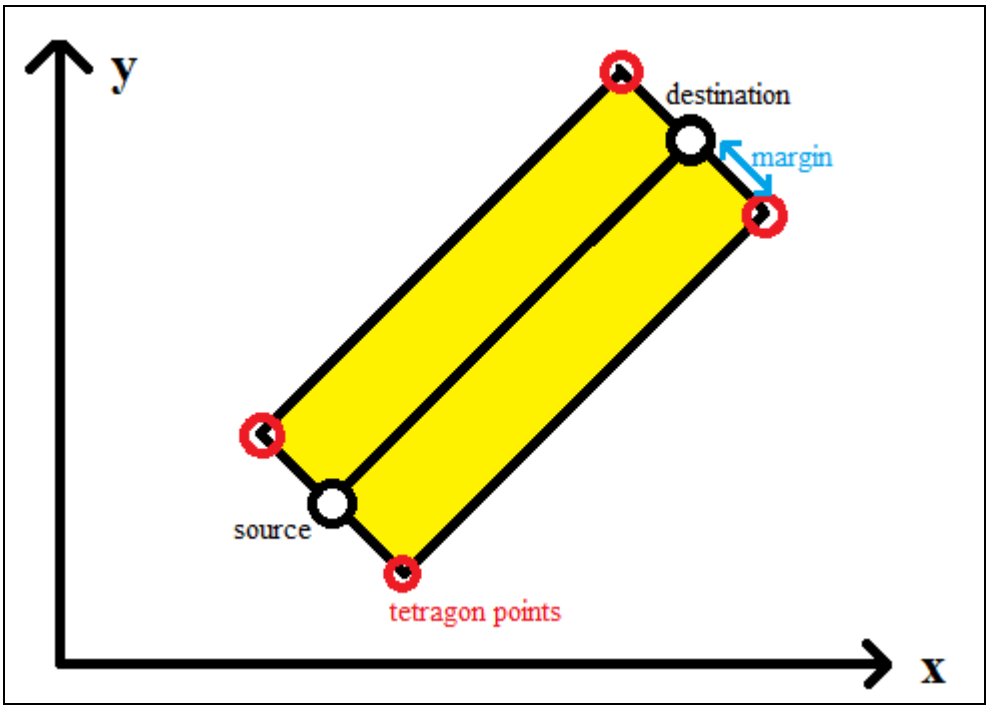

Figure 5.27: Illustration of the tetragon constructed by two points and a margin.

```
bool isPointInTetragon(sf::Vector2f point, sf::Vector2f* tetragon)
{
    float angle; sf::Vector2f p1,p2;
    for (int i=0;i<4;i++) {
      p1.x = tetragon[i].x - point.x;
      p1.y = tetragon[i].y - point.y;
      p2.x = tetragon[(i+1)%4].x - point.x;
      p2.y = tetragon[(i+1)%4].y - point.y;
      angle += getAngle(p1.x,p1.y,p2.x,p2.y);
   }
   if (abs(angle) < M_PI) return false;
   else return true;
};

bool isPointBetween(sf::Vector2f testPoint, sf::Vector2f point1, sf::Vector2f point2, float margin)
{
    float slope1 = (point2.y-point1.y)/(point2.x-point1.x);
    float slope2 = -1.0/slope1;
    float angle = atan(slope2);
    float x_dif = margin * cos(angle);
    float y_dif = margin * sin(angle);
    sf::Vector2f tetragon[4] = {
        sf::Vector2f(point1.x+x_dif,point1.y+y_dif),
        sf::Vector2f(point1.x-x_dif,point1.y-y_dif),
        sf::Vector2f(point2.x-x_dif,point2.y-y_dif),
        sf::Vector2f(point2.x+x_dif,point2.y+y_dif)
    };
    return isPointInTetragon(testPoint,tetragon);
};
```

Figure 5.28: Checking if a point is between two given points.

The dribbling action is executed with two possible changes following each other. If the ball is on the footballer's feet, he kicks the ball towards desired location. If the ball is not on the footballer's feet, he goes towards the ball (see Figure 5.29). Repeated execution of these two possible changes makes the impression of dribbling the ball.

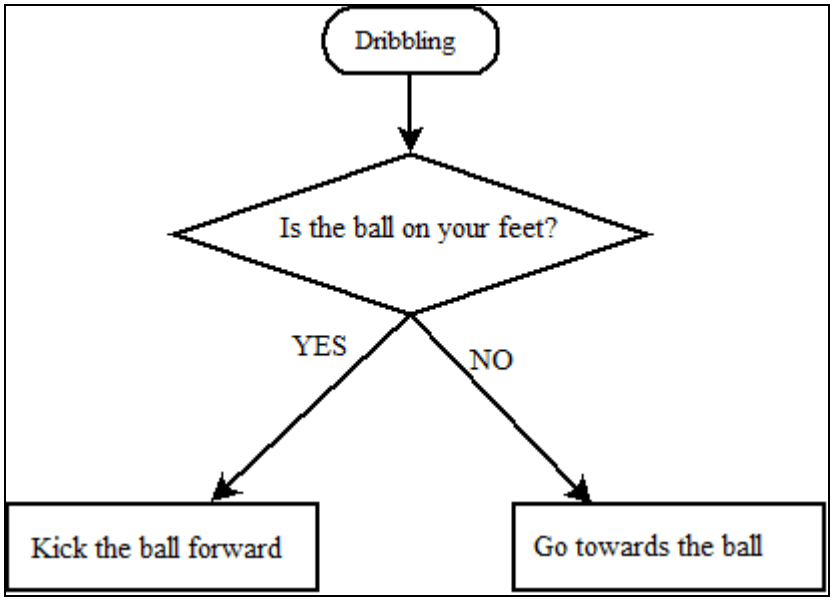

Figure 5.29: Illustration of dribbling with a simple decision tree.

6 EXPERIMENTAL RESULTS

A questionnaire is prepared with 7 questions for examining the results of the project (see Figure 6.1). The software product of this project is tested by 5 people who like playing football management games. Participants played the game for 30 minutes and answered all questions in the questionnaire.

The main purpose of the questionnaire is questioning whether the project is meeting success criteria which is defined at the beginning of the project. The second purpose of the questionnaire is comparing the game with the most successful football management game, Football Manager [6]. The questionnaire consist of 5 multiple choice questions and 2 short answer questions.

Question 1: How do you rate the user interface of the game in terms of functionality and consistency?
1) Very poor 2) Poor 3) Average 4) Good 5) Excellent

Question 2: How do you rate the entertainment you experienced from this game?
1) Very poor 2) Poor 3) Average 4) Good 5) Excellent

Question 3: How realistic do you think that default positioning of footballers is?
1) Very poor 2) Poor 3) Average 4) Good 5) Excellent

Question 4: How realistic do you think that passing actions between footballers are?
1) Very poor 2) Poor 3) Average 4) Good 5) Excellent

Question 5: How realistic do you think that dribbling actions of footballers are?
1) Very poor 2) Poor 3) Average 4) Good 5) Excellent

Question 6: What are the positive aspects of this game compared to Football Manager?
Answer:

Question 7: What are the negative aspects of this game compared to Football Manager?
Answer:

Figure 6.1: The questionaire prepared for experimental results

QUESTION	P1	P2	P3	P4	P5	AVERAGE
1. How do you rate the user interface of the game in terms of functionality and consistency?	4	4	5	4	4	4.2
2. How do you rate the entertainment you experienced from this game?	2	1	2	2	2	1.8
3. How realistic do you think that default positioning of footballers is?	4	3	4	3	4	3.6
4. How realistic do you think that passing actions between footballers are?	3	2	3	3	3	2.8
5. How realistic do you think that dribbling actions of footballers are?	2	1	3	2	3	2.2

P: Participant

1: Very poor
2: Poor
3: Average
4: Good
5: Excellent

Figure 6.2: The answers of multiple choice questions in the questionaire

All answers to 5 multiple choice questions are given in Figure 6.2. While the user interface and the positioning of footballers have recieved good response from participants, the responses for passing and dribbling actions are below average. It can be understood that participants experienced poor entertainment from the game since the number of actions implemented is pretty low.

Success criteria for the project is described in the section "2.2.1 Success Criteria". Three criteria is evaluated according to user feedbacks and personal examination of the system as below:

- **Entertainment:** Users were not very entertained while playing the game due to lack of actions of footballers. Further projects based on this project can satisfy the entertainment criteria by improving the existing actions and implementing new actions.
- **Consistent user interface:** User interface is considered user-friendly, functional and consistent by users. The source code of user interface is clean and suitable for new additions. This criteria is satisfied.
- **Correct simulation and intelligent agents:** Correct simulation of football is developed with correct football pitch dimensions and correct referee decisions after the movements of the ball. However, some aspects of football are missing as few actions could be implemented at the completion of the project. Implemented decision making system is working properly and it is suitable for developing truly intelligent agents that can play football.

The answers to 2 short answer questions were about comparison between the product of this project and Football Manager.

Participants indicated that the positive differences of Caretaker Manager are the freedom of determining footballers' tactical positions, simplicity of the user interface and the quality of Turkish language support. Football Manager allows user to choose one of the 25 defined tactical positions for footballers. The user interface of Football Manager is considered as complicated by some users. The translation for Turkish language is sometimes criticized by users. The negative differences of Caretaker Manager are defined as the lack of actions for footballers and the quality of actions.

As a result, the simulation and AI architecture looks solid. The graphical user interface of the game is adequate. However, more actions are needed to be implemented in order to have a realistic game that can simulate football which is one of the most complicated team sports in the world.

7 CONCLUSION AND SUGGESTIONS

The project started with the idea of developing a football management game. The project scope is defined as three following parts: Creating a graphical user interface, building a football simulation and building an artificial intelligence system for playing football.

Solutions for the three parts of the project are designed and implemented. The product of the project is tested by users. According to feedbacks of users, solution for user interface problem is satisfying. The action modules are not good enough both quantitatively and qualitatively to simulate the football realistically. Action modules need to be reviewed and new actions should be implemented as a better solution. One

The performance of the implemented parts of solutions is satisfying. Football simulation and artificial intelligence architecture are both working properly and they are compatible with each other. The cost of adding new action modules are pretty low in terms of time and effort. The product of this project can be considered as a well-constructed software. This project can be used as a base project for further projects.

My suggestions for people who will be working on building simulations and developing games are as follows:

1- Project plan should be clear and well constructed. The plan is the path that developers will follow all the way through.

2- Analysis and modelling parts of the project should be finished as early as possible. One of the reasons why some of the success criteria of this project could not be satisfied is the lack of time for implementation of new actions.

3- Building a simulation is a complicated task and it is needed to be well planned. Following the object oriented programming paradigm is a good way to keep things clean and understandable. It is always good to construct the necessary systems as a whole that consists of smaller modules. Implementation and testing are two of the most decisive phases of game development. If there is a good model consists of small modules, implementation and testing of the software gets easier and cause less time consumption. Building a football simulation is even more complicated than most of simulations. Therefore, the importance of a good model should not be underestimated.

4- The impact of graphical user interface on users is important. A good simulation game can be considered by many users as an unsuccessful simulation if the interface is not user-friendly. Composition of visual elements is also important. The colors of shapes and the sizes of words on the screen cannot be determined separately. A screen should be constructed and analyzed as a whole.

8 REFERENCES

[1] Kantar Media, “2014 FIFA World Cup Brazil™ Television Audience Report”, 2014, http://resources.fifa.com/mm/document/affederation/tv/02/74/55/57/2014fwcbraziltvaudiencereport(draft5)(issuedate14.12.15)_neutral.pdf
[2] FIFA Manager 11. (n.d.). Retrieved June 03, 2016, from http://www.ea.com/fifa-manager-11
[3] Championship Manager. (n.d.). Retrieved June 03, 2016, from http://www.championshipmanager.co.uk/
[4] Football Manager 2016 - The Official Site. (n.d.). Retrieved June 03, 2016, from http://www.footballmanager.com/
[5] Graham. D. (2012). *Game Coding Complete, Fourth Edition.* Boston, MA: Course Technology.
[6] Pinto. H., & Alvares L. O. (2006). *Game Programming Gems 6.* Boston, MA: Charles River Media. pp.235-242.
[7] Heckbert P. S. (Ed.). (1994). *Graphics Gems IV.* San Diego, CA: Academic Press. p.42.