Nathan Simcock

CTEC3451

Computer games programming -   
final year project report

To Develop a Menu-based Sports Simulation Game

Table of Contents

[Acknowledgements 2](#_Toc71049036)

[1.0 Introduction 3](#_Toc71049037)

[2.0 Research 4](#_Toc71049038)

[2.1 Concept Research 4](#_Toc71049039)

[2.2 Development Research 5](#_Toc71049040)

[3.0 Design 5](#_Toc71049041)

[3.1 Methodology 5](#_Toc71049042)

[3.2 Analysis of Requirements 6](#_Toc71049043)

[3.3 System Concept Designs 7](#_Toc71049044)

[4.0 Implementation 8](#_Toc71049045)

[4.1 Overview 8](#_Toc71049046)

[4.2 Development Process 9](#_Toc71049047)

[5.0 Testing 16](#_Toc71049048)

[5.1 Testing Method 16](#_Toc71049049)

[5.2 Test Log 16](#_Toc71049050)

[6.0 Critical Evaluation 17](#_Toc71049051)

[6.1 Project Analysis 17](#_Toc71049052)

[6.2 Potential Improvements 18](#_Toc71049053)

[6.3 Future Work 19](#_Toc71049054)

[7.0 Conclusion 20](#_Toc71049055)

[8.0 References 21](#_Toc71049056)

[9.0 Appendices 22](#_Toc71049057)

# Acknowledgements

I would like to thank all those who have helped me throughout the research and development of my project. I would like to thank Mishrey Almarshoud, my supervisor for this project, who has always been available to offer help and guidance. I would also like to thank all tutors and staff at De Montfort University for their dedication and support in aiding my education over the past three years. Finally, thank you to Hossein Malekmohamadi, the module leader, who has given the students plenty of resources to help us in our project, and Mohammed Kabiru, the second marker of my project, for taking the time to assess my work.

# 1.0 Introduction

As part of my final year of Computer Games Programming studies at De Montfort University, I was tasked with carrying out a significant piece of work that reflects the aims and outcomes of my programme. I was able to decide upon the work that I would produce, where I decided to form the goal ‘To Develop a Menu-based Sports Simulation Game’.

To achieve this goal, I developed Basketball Management Simulator. This is a menu-based Sports Simulation game that I created using Unity Game Engine. A simulation game aims “to display the ins and outs of a certain real-life process all within the medium of the video game.” (Lerouge, 2020). The purpose of my game is to put the player in the role of the General Manager of a Basketball team, the London Royals, who are one of ten teams in a European Basketball League. It must do this whilst retaining an entertainment aspect as video games should be fun and enjoyable for the player, to retain their interest.

The aim of the game is to win the league, by having the most wins out of all teams at the end of the season. In the demonstration version of the system, the season is nine games long, so each team plays each other once, however, this would be extended for the full version.

The game does not have difficulty levels, as it is a simulation of real life, where you cannot choose how hard a task is, it is designed for each task to be as realistic as possible. Instead, difficulty can be reflected by the roster given to the player when they start the game. When the game is started, each of the ten teams is given a roster of ten players. Every player has a range of numerical stats between 1-99 to reflect their abilities in different aspects of Basketball.

A general manager cannot pick and choose the players they want when they take over a team, they must make the best of what they are given. This is reflected via random roster generation, where sometimes you may be given a team which is the strongest in the league, other times you may have the lowest rating in the league.

After taking charge of the Royals, the player has a selection of ways to improve or adjust their team. The first is to perform trades. A trade is a one-for-one swap of players between two teams in the league. Basketball Management Simulator features a realistic trade engine, which allows the user to pick the player they wish to get rid of, and then offers players of similar value from other teams.

The game also offers a detailed match engine system. As a simulator, the user does not play the matches themselves, but the result is generated based on a number of factors. Tactical plans and line-up changes made in the line-up editor can have huge effects on the result of matches. This makes it possible for an underdog to outplay a stronger opponent with a well-executed match plan.

The line-up editor allows the user to view their player’s individual stats and make a choice of who their starting line-up for the next game should be. This can massively shape the outcome of games, as the match engine considers all the player ratings compared to the opponent’s when calculating the score of a match. A well-made plan could be the difference between winning and losing in the game.

Being the general manager of a huge team like the Royals does not come without challenges, as players can become upset with your management and demand to leave the team, or their performance levels will drop.

I was convinced to create this game by the level of skill it would require to be made. Being menu-based, I knew that lots of advanced code would be required if the game were to be of high quality. This game is unlike a regular 2D or 3D game in that there is no physics or moving objects, just menus. Therefore, I knew that if the game were to be enjoyable, the code behind the menus would be vital. Not only would this game help me to demonstrate what I have learned in my time at De Montfort, but also further my coding knowledge and develop my skills.

The second reason behind my choice was a potential gap in the market. According to SteamDB, Football Manager 2021’s all-time peak for concurrent players is 93,865. Along with Bailey’s report (2021) which states that the series has now passed “33 million units sold,” this shows that Sports Simulation games can perform well in today’s market. There is also no lack of interest in Basketball video games, with the market leader NBA 2K21 selling “more than eight million copies” (Takahashi, 2021). The combination of these two genres has not yet found a way into the market, with no Basketball Simulation games surpassing 100 concurrent players according to SteamDB.

# 2.0 Research

## 2.1 Concept Research

My initial decision to create Basketball Management Simulator was inspired by Football Manager, the leading Sports Simulation game. With how popular this game and the sport of Basketball have become in recent times, I realised that there was potential for a Basketball Simulation game to succeed in this market.

The second game I drew inspiration from was the NBA 2K series, a console game that focuses more on the games themselves than the management aspect. The series does feature a mode called MyLeague, which allows the player to take charge of a team, but only scratches the surface in the details and still prioritises the match gameplay.

As part of my literature review, I studied the history of sports video games, simulation games, and the two previously mentioned games. The research performed was not only beneficial for the writing of the literature review, but also for the development of Basketball Management Simulator.

I was able to identify what would be necessary to make the game a success. In my case, a well-balanced database or roster would be essential to make the game fair in terms of difficulty. I also found that the match engine was a highly praised part of the new Football Manager instalment, so I decided to put a lot of detail into my own match engine, to make sure this was of high quality.

Not only did I research video games in the Sports world, but the sport of basketball itself. With my game intending to be a realistic simulation of Basketball, I had to further my understanding of how the game works from a coach/general manager’s perspective. This research helped me to create a match engine that would be balanced and allow a good defensive team to have as much chance of winning a game as a good offensive team. With the nature of how the engine works, comparing offensive stats against defensive stats and then generating a per quarter score, it could have been possible that a team with high offense and low defence would be more likely to win than a poor offensive side, my research informed me that this should not be the case.

## 2.2 Development Research

When developing the game, there were many times where I would reach an error in the code or not know how to create certain functions. In these cases, I used a variety of methods to overcome the obstacles.

While creating the user interface or menu code, I could often refer to previous projects I had developed over my time at DMU to gather ideas. An example of this was to create the mute sound button which is present in the settings screen.

However, as this is a project unlike any others I have created before, I would often have to look further than my previous work to find answers. In these cases, such as for designing the roster generation system, I would look towards online sources such as the Unity Forum or Stack Overflow, two places where people with similar problems to mine have asked questions and received solutions.

Another method of research I performed was watching coding tutorials on YouTube. Again, people would be performing a task like the one I was needing help with, so I was able to learn how they approached the task and adapt it to fit what I needed.

# 3.0 Design

## 3.1 Methodology

At the beginning of the project, I decided to follow the agile development methodology. This means that a component of the game would be planned, designed, developed, and tested before being deployed in the game. For the most part, I followed this structure throughout the project, however, I made a slight adjustment to the deployment stage. In the game, many features are tied together, therefore they had to be implemented before they could be fully tested. For example, the match engine required a roster to be in place to generate match results.

This methodology was useful as in situations where an error would arise, it was easy to pinpoint the cause, as the whole system would be tested and working before a new feature is added. This also saved a lot of stress on me as the developer as there were no points where I would come across a catalogue of errors all at once.

Being able to plan and design a component made development very efficient and testing easy to perform. This is because a well-designed component would already have potential errors thought out, and solutions planned for if one did arise.

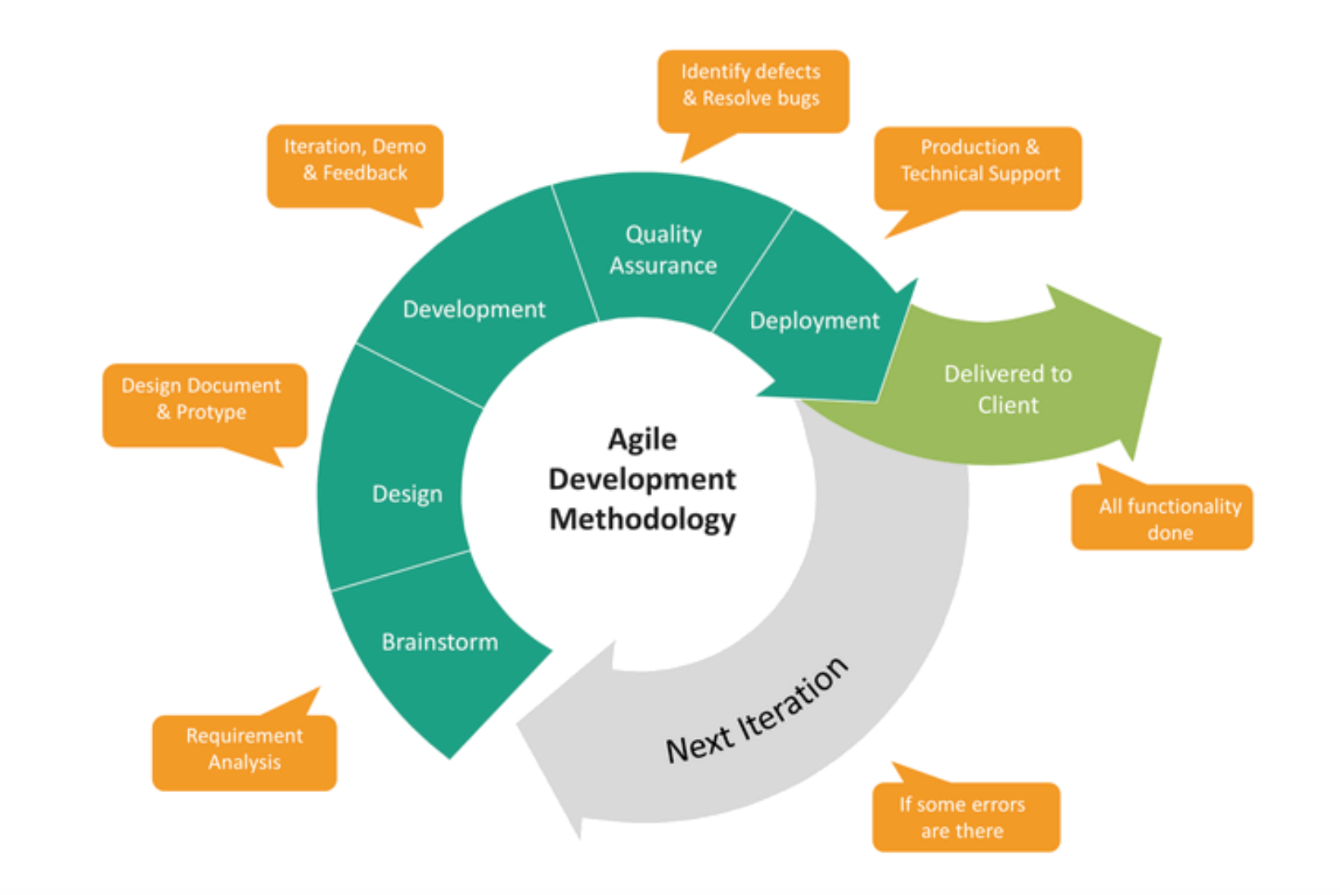


Figure - The Agile Development Methodology. Available at: <https://orionadvisortech.com/wp-content/uploads/2019/11/Screen-Shot-2019-11-04-at-1.13.10-PM.png> (Accessed 29/05/2021)

## 3.2 Analysis of Requirements

When in the initial planning stages of the project, the requirements were separated into two categories. Functional requirements, these are things which must be present in the game for the user to be able to achieve the purpose of the game. The second could be considered as non-functional requirements, additional features which I would ideally like to include in the game.

To decide upon the functional requirements, which are outlined in Appendix 1, I took a step-by-step approach through the idea for the game. By doing this, I could determine what would be necessary for the user to get from the start screen of the game to the final screen at the end of the season. As the game was still in the planning stage, and nothing had been written or designed at this point, it was possible that the requirements could evolve throughout the development cycle. While most stayed the same and were met in the final product, the losing condition was changed from an owner happiness level, to not winning the championship at the end of the season.

The research stage was vital for the creation of the system requirements, as I was able to identify the key functions of similar games. While my game is not the same as those I researched, sharing a genre with them means that some aspects will be shared across most games in that genre. In the Sports Simulation genre, I found that every game must have some form of continue button to progress to the next event, and a match engine to simulate games.

Once functional requirements were set, the focus could be moved to features that are not necessary for the game but add depth and enjoyment. This included components such as a realistic trade system, line-up editor, and player happiness system. The game can be played without these features, but they are in place to help the player reach the end goal of winning the championship.

By setting out functional requirements, this not only gives the game criteria to have its success judged upon, but it also helps in the planning and developing stages. Knowing that the project has a strict deadline, I understood that not every component I desired could be added to the game, therefore I had to manage my time efficiently. To produce a working product, I prioritised the functional requirements, developing these as early as possible, and then moving my attention to additional features.

## 3.3 System Concept Designs

As part of the first deliverable, a collection of designs was created as plans for how the game could look later in development. These included interface designs, an entity-relationship diagram, and a system architecture plan. At that stage in development, most of the designs were complete, however, since then some extra items such as the team logo were designed.



Figure - The player team logo design.

Similar to the system requirements, the designs and plans made the development process much smoother and efficient. The menu designs gave a template to follow whilst creating the game’s UI, meaning this did not have to be improvised and thought of during development. The entity-relationship diagram was extremely beneficial as when the roster database was being created, this plan ensured there were no missing items as the code could be checked against the plan. The system architecture plan, which laid out the scripts that would be required throughout the game, was not followed completely as some functions were placed together in one script. For example, the trade system was inside the roster script. However, this did not make the plans useless as they detailed much of the data that would be needed to perform each function.

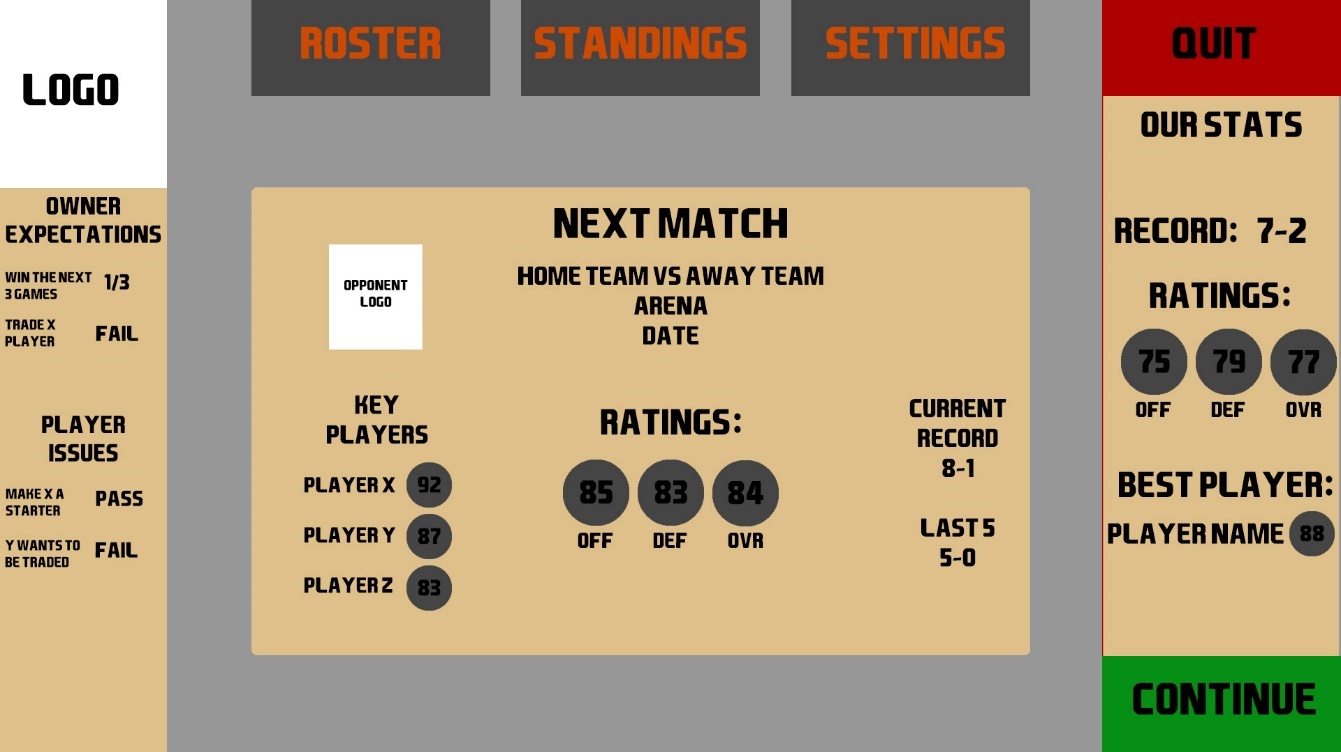


Figure - Home screen design concept.

Figure 3 shows a concept of the home screen of the game, which is arguably the most important screen as it is the first the user comes across after starting a new game. It is also the screen with the most important information such as next match information, happiness details, and the continue to match button. As I knew this screen was going to be visited many times in a playthrough, it was important to make sure it looked professional and provided all necessary details.

Being the first design of the game, this was where I decided upon the colour scheme and layout of the game, choosing a primary colour of beige with black writing, and secondary colour of light grey. The reasoning behind these choices was to not have the colour stand out of the screen, attracting the eye more to the information than the colours. The quit and continue buttons are clearly coloured red and green to make these buttons stand out.

Despite planning to have at least six screens that the user can navigate to, this design only features three buttons across the top of the screen. These buttons act as categories which will then create a drop-down list of menus in that category. I chose to do this as having lots of buttons on the screen may overcrowd the screen space, confusing the player.

# 4.0 Implementation

## 4.1 Overview

I chose to use Unity Game Engine to implement the system I had designed. Developing a game that focuses on UI to present itself to the player, I needed software which would help me to design professional UI elements. Unity not only offers pre-made objects such as a canvas or buttons with functions included, but it allows for the addition of original code to manipulate these objects to work how I intend them to. For example, a button has a built-in on-click function, which I adapted to perform any tasks I wanted to occur when the button is clicked.

If I did not use a game engine with existing components, I would have had to create code to record the button press, create hover and click animations, and even render the button to the screen. All these tasks would have taken time away from creating the game, so while it would have been completely developed from the ground up, I think the time lost making these functions would have been damaging to the quality of the product.

## 4.2 Development Process

When developing the game, the front-end would always be the first part I had to work on, this is because a piece of code requires an object to function. For instance, a script for a button would not work without a button to attach the script to. Once a feature had been implemented on the front end, it could be coded and adjusted to perform the functions desired.

I worked in a logical order, from the first screen of the game the user would come across, to the end of the game. This meant that I could easily perform integration testing, where you investigate how a feature fits into the existing system.

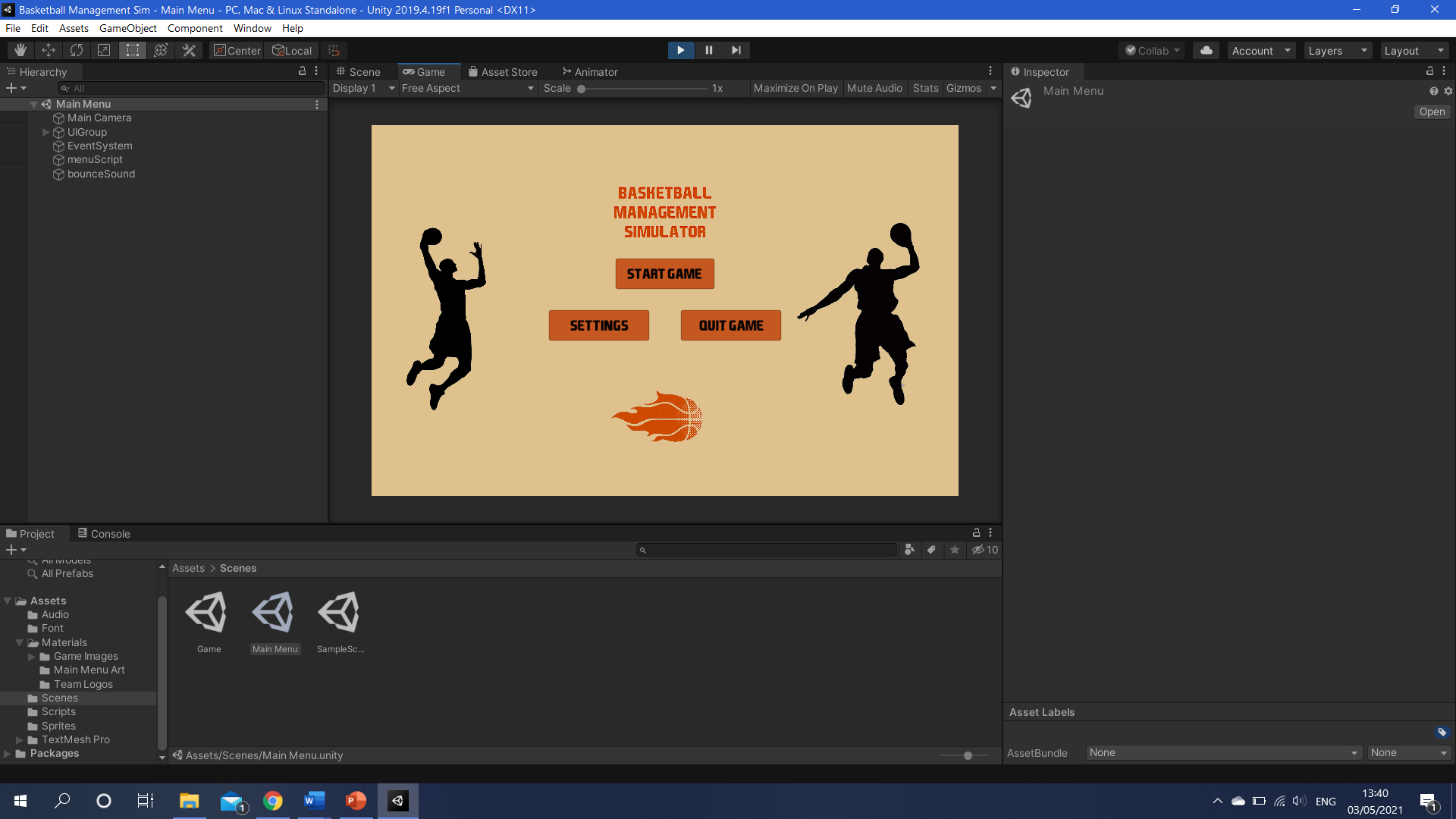
With this approach, the first step in the development process was to create the main menu, as this is the first screen the player sees when the game starts. This screen was also the easiest to build as it does not require anything else to be in place before it can be built.

Figure - The final version of the start screen.

When starting a project in Unity, the first step was to create a new scene, which contains the objects of a game. Individual scenes are commonly created for the different levels of a game, so in this case, the main menu and the game scene were split into different scenes.

Once the scene was in place, the user interface for this menu could be created. As seen in Figure 4, the main menu was kept basic, to fit with the earlier designs. The ‘menuScript’, which is a C# coding file, was then created to perform the functions of the main menu. This script holds each button, as well as canvasses, which is a background holding UI elements. In total there are four canvasses, the main menu, the settings screen, the quit screen, and a how to play screen.

Each screen is tied to a button press, so when the corresponding button is clicked, the screen is loaded and other screens are closed, see Appendix 2 for the canvas switch code which is used throughout the game. A sound clip of a basketball bouncing is attached to button presses on this menu; however, this can be toggled off in the settings screen. The settings screen also allows for the game to be toggled between full screen and windowed.

When developing in Unity, you are provided with a preview window of your game, this shows you how the game looks to the user. This was the cause of my first problem, as I built the scene to fit the preview window. This meant that when the game was played in a larger display size, objects did not scale to the size of the new window. This was fixed via the Canvas Scaler, where you can set the objects to scale with the screen size.

Figure - My first error, the main menu does not scale to the window size, leaving lots of blue space.

Later in the development stage, the same issue came up with buttons falling in the wrong places, this was fixed by anchoring. This sets an object to be set in place no matter the screen size. For example, the continue button would be anchored to the bottom right of the screen, this meant that in any screen size, the continue button would be locked to the bottom right.

Once the main menu was in place, the game scene had to be made so that when the start-game button is pressed, the player is taken to the game. The initial canvas of the game scene was the home screen, where the user can navigate to any other screen in the game and is presented with information about their next match and their team. As there was no database holding the roster in place at this time, the details could not be added, but the layout and the navigation system could be created.

Whilst creating the navigation buttons, I came across the second issue of my system (Figure 6). When a button with text was scaled in size, the text would lose quality, becoming blurry. This would be fixed by increasing the size of the button, then adjusting the font size, rather than scaling the button. At first this issue was also happening across text boxes, where any slight adjustment would cause a loss of quality in text. I then switched text type from regular to TextMesh Pro across the project, which uses advanced rendering techniques to modify the style of text, instead of having “to “improvise” and attempt to generate the missing pixels on the fly.” (Nigretti, 2018).

Every screen was separated into its own canvas to enable navigation between menus, and the navigation buttons themselves were separated into a separate canvas, so they could be used on every screen without duplication. This caused an issue when switching the active canvas, as the new canvas would be placed over the buttons. This was solved by adjusting the sort order, which places the lower number on top of the scene as it is rendered after.

Figure - Once the buttons were created, if they were made bigger the text would become blurry.

Once a menu system was in place that would allow the navigation between every screen, the game itself was made. The first component to be developed was the database, or roster. As outlined in the entity-relationship diagram in the first deliverable, the roster would hold most data that is needed, including player names, stats, etc.

The first step to creating a roster is the ‘rosterScript’, which holds the database and all related functions. The initial plan for the game was to have a pre-made roster with all player stats determined on start-up. However, to increase replayability and make the game more sporadic, I created a roster generator. This creates a new roster every time the game is started, which can make the game more easy/difficult depending on how good the team you receive is.

The roster consists of a series of lists, each list holds a specific attribute e.g., player name or inside scoring attribute. Every list is in the same order, so entry 32 in each list relates to one player.

Each player attribute works via a for loop and a number generator. The name generator has 30+ first names and last names which are then picked from and combined to make each player’s full name.

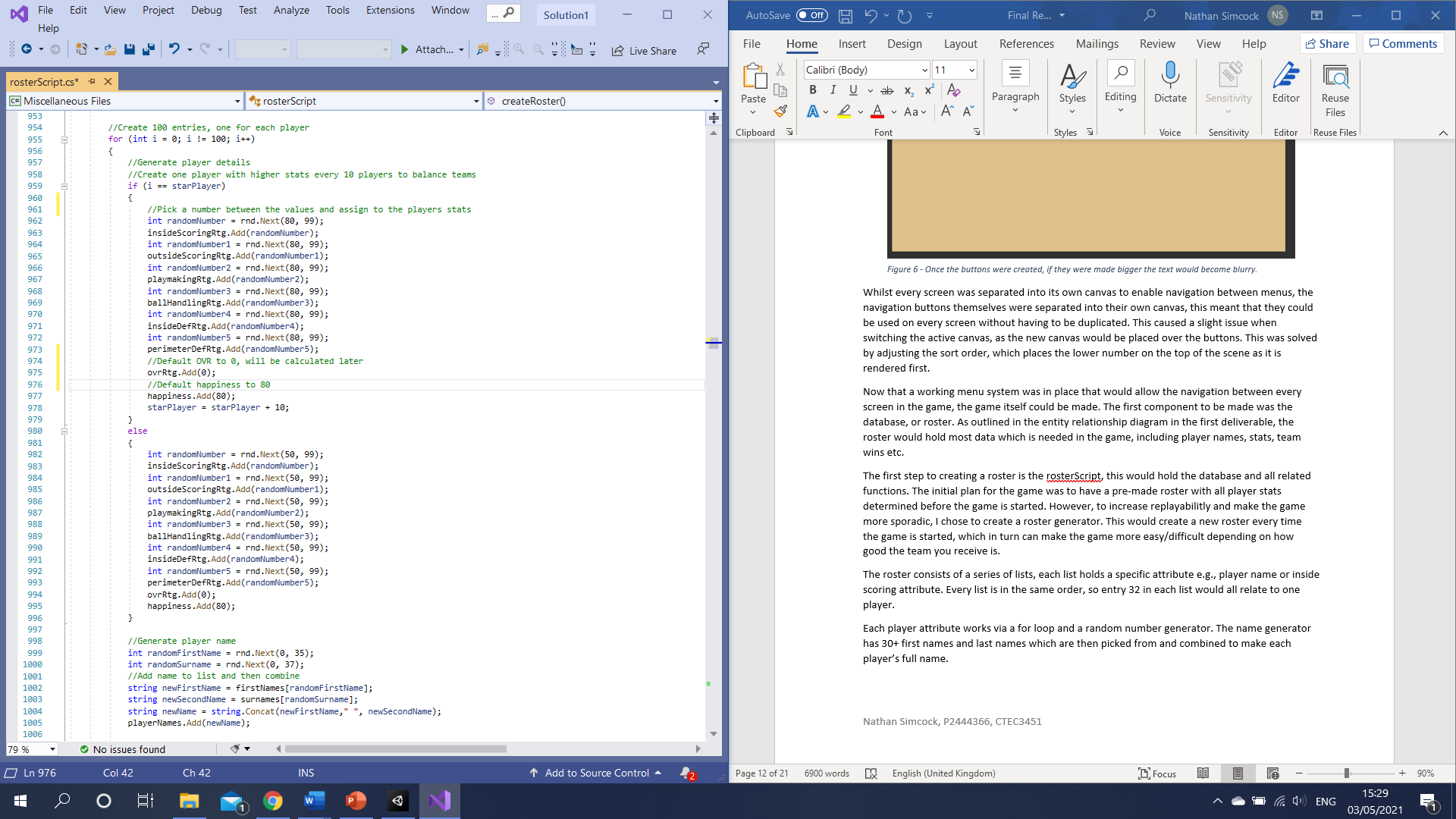
The player generator, seen above, creates a 100-player loop to fill each team with 10 players. It then has a ‘starPlayer’ value which starts at 5 and increases by 10 each time the loop reaches it. The regular loop generates stats between 50 and 99, but the ‘starPlayer’ loop picks numbers between 80 and 99 to ensure each team has at least one good player. After a number is picked, it is then added to the corresponding list. Once the stats for a player are added, two names are picked and combined to form the full name. A range of if statements then assign each player to a team based on their index number from 0-99, 0-9 is London, 10 – 19 Paris, etc.

Figure - The player creation system.

A function called ‘ratingCalculate’ then adds each attribute for a player and divides by the number of attributes (6) to give an overall rating. After this, the ‘calcTeamRatings’ function runs which does the same but instead for an entire team to create team ratings.

The first screen which had details added was the roster screen, this takes each player on the user team’s attributes and displays them on screen. This screen is useful for analysing the players you have available.

The trade finder system was then developed. This screen prints out a list of the user’s players along with their rating, happiness, and trade value. Trade value is a combined rating of a player’s overall and happiness and is displayed as a star rating. This system makes a player who is not happy with their team less valuable in trade deals, as trade partners will know that the player does not want to play for their team anymore.

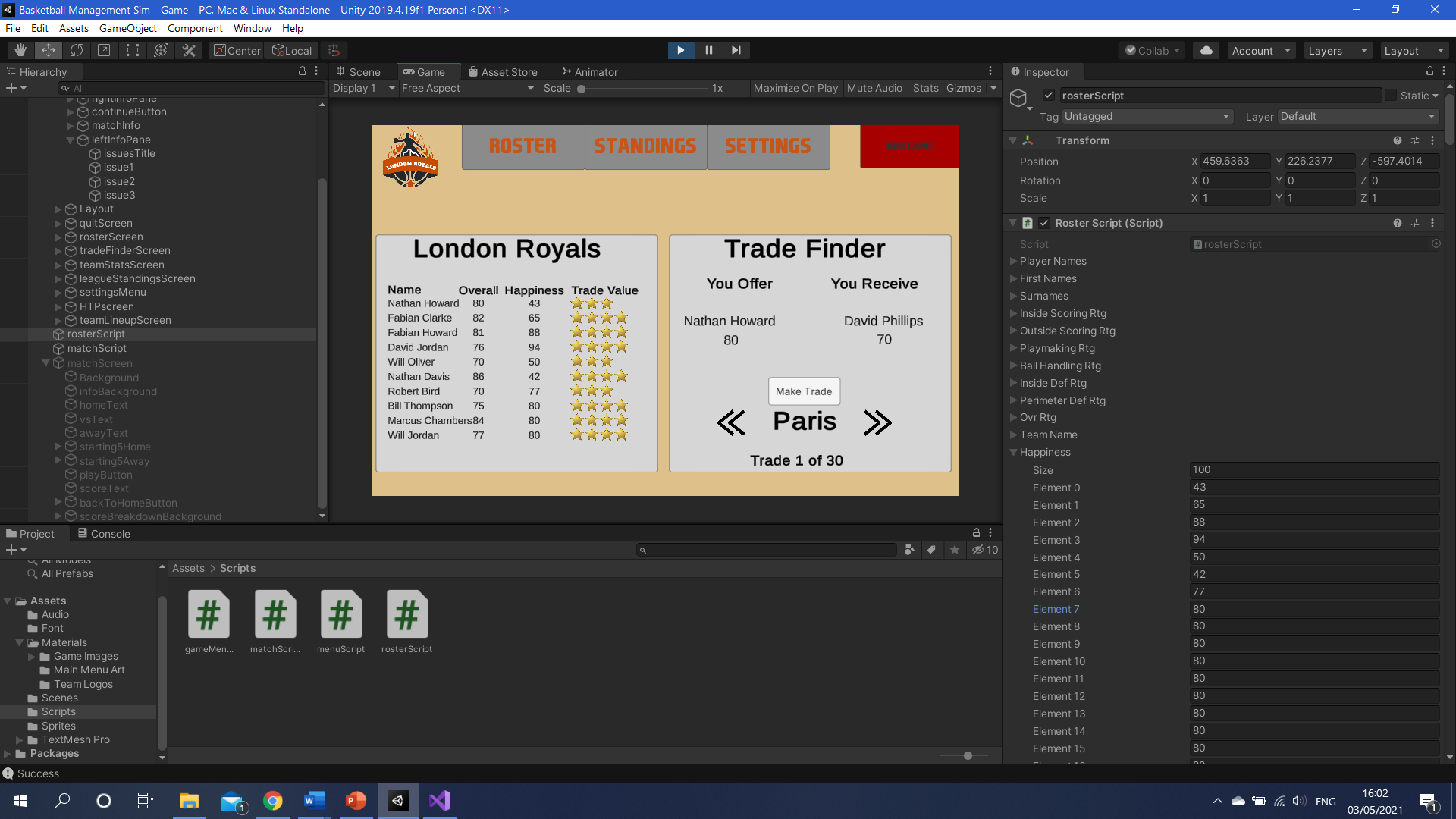


Figure - The trade finder system.

Figure 8 shows the trade finder screen and how a player who is unhappy holds less value than a player who is happy. The user can then select the player they wish to trade and are provided with trade offers on the right panel of the screen. The system is built to find players of similar value from each team and offer a suitable trade. You can cycle through each available trade by clicking the arrows, and then press the make trade button to swap the players. This then creates some temporary placeholders for each player’s attributes. The player joining the team fills the temporary values, before the playing leaving occupies that player’s space in the database, and their old space is filled with the temporary values. This basically switches the unique index number of the two players, so they are then associated with their new team.

The next screens to be developed were the two standings pages, team stats, and league standings. The team stats screen takes all the team’s average ratings and presents them for use in making tactics and gameplans. The league standings table lists each team in order of their number of wins.

This presented several issues where teams would be tied for wins. On start-up, where all teams have 0 wins, only the last team to be checked would have their details displayed. Here I developed a system with additional conditions, which is shown in Figure 9. This system takes each team and compares them to the team currently occupying each place in the table. If the current team has more wins than the selected team, they will take their place. If the two teams are tied on wins, the new system awards the team with less losses the higher slot, if tied again, the team with more total points.

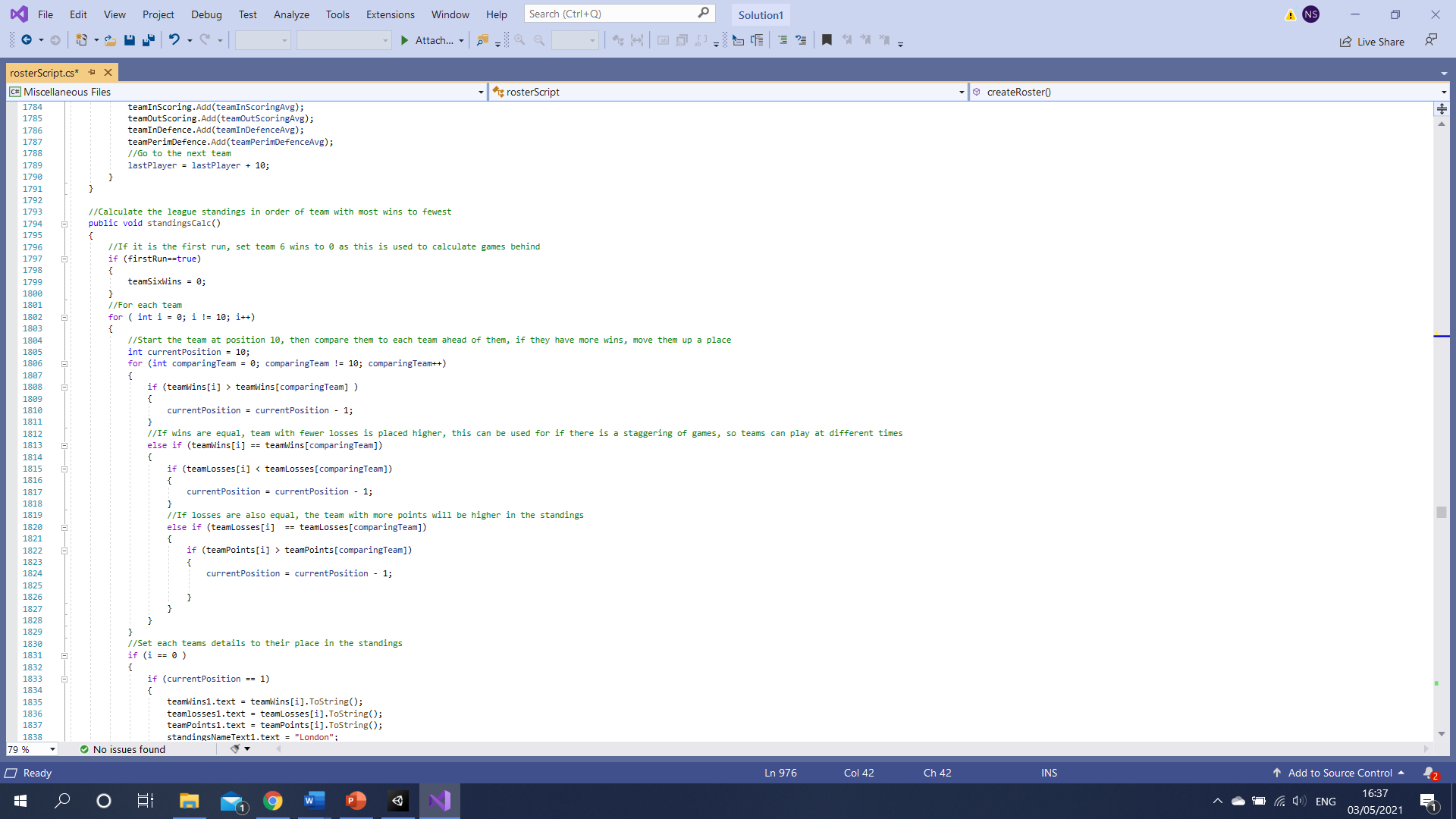
Once the roster was in place and the league standings could be viewed, the match engine was developed. The first version of the match engine, which can be seen in Appendix 3, was very basic, simply comparing a team’s defensive rating to the other’s offense, then generating a score. This system worked but produced a lot of upsets, where a weaker team beats a stronger opponent. This is because there is an equal chance of a number at the bottom of a score bracket being picked as a higher number, so strong teams were regularly scoring low points. I then developed a new system (Appendix 4) with 4 quarters, to reflect a real Basketball match, and generate a score per quarter. By having four different scores, it is more likely that the stronger team wins, as there are four simulations, reducing the luck aspect.

Figure - The improved standings calculator.

This new system made it very hard for an underdog to beat a stronger team, as the minimum score for a team with an offense to defence difference of +5 was 25, and half of the brackets are unable to score this as their maximum.

This caused a third iteration of the match engine to be developed (Appendix 5). This system removes a lot of luck through a max score value. The system now compares more values and adds to the max score. The outside scoring is compared to the perimeter defence, and inside scoring is compared against inside defence. There is then a fixed minimum score of 10 and a maximum depending on the stat differentials which has a random number generated from. Each quarter’s scores are then added, and if the scores are equal, an overtime period is played until a winner is found.

Now that the match engine considered multiple stats, this could be paired with a line-up editor. The engine was developed to weigh the starting 5 players more heavily when calculating team ratings, as these will have more impact on the game. Therefore, the game needed to allow the user to change who their starting line-up is.

The line-up editor works similar to the trade engine, where the player selects who they want to move in and out, and the system temporarily stores one player and replaces them with the other. The editor was designed to be used alongside the team stats page so that a player can plan the best line-up for an upcoming opponent. For example, if the player sees their next opponent is weak at perimeter defence and strong at inside scoring, they may choose to start their best outside scorers and inside defenders to counter this. This feature gives the player a chance to shape the outcome of matches.

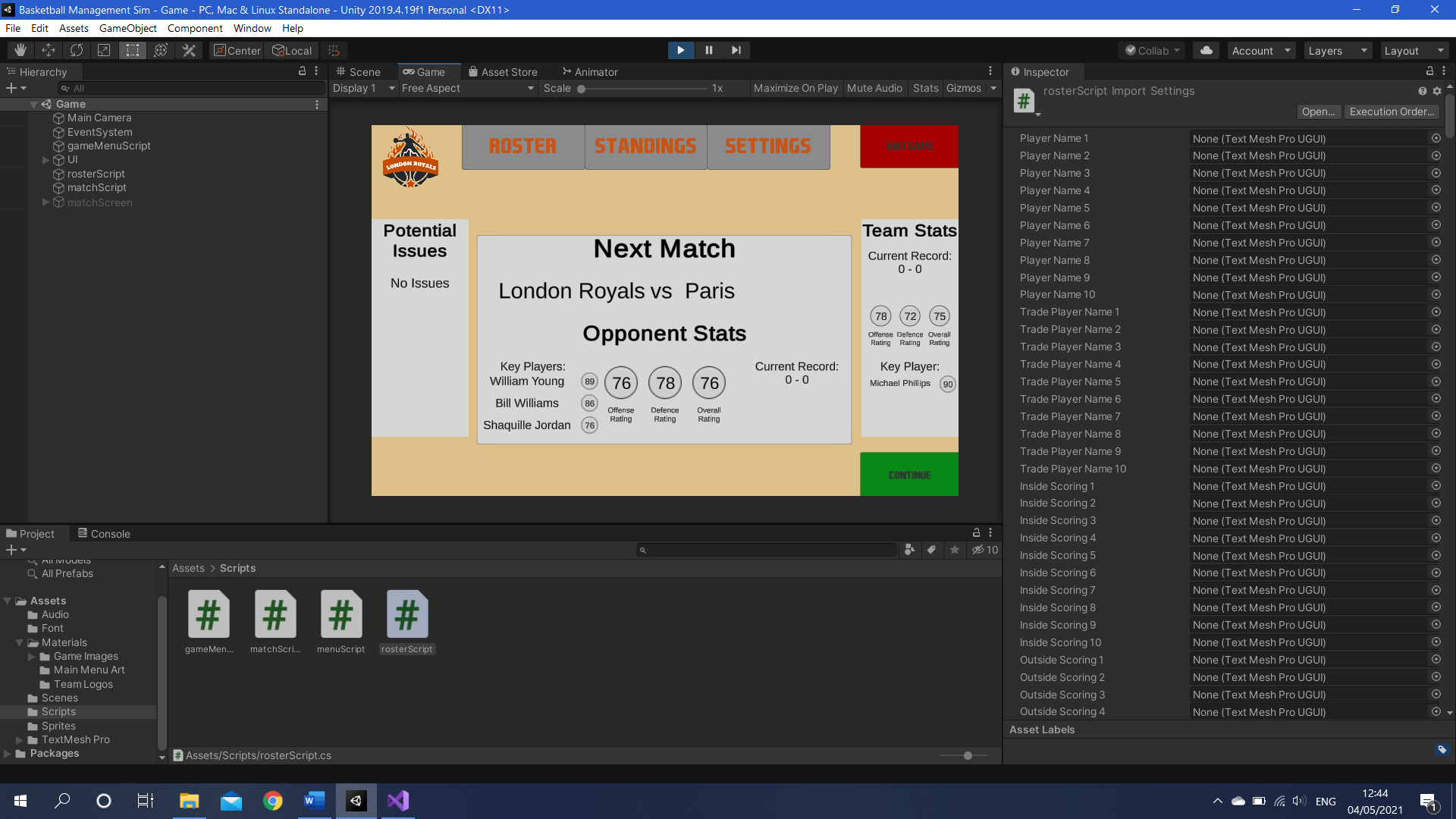
The next feature that required developing was the simulation of a whole season, rather than a single game. Once the user leaves the match, the home screen is updated with information about their upcoming opponent, including team ratings and current record. This is filled out through a ‘gameNumber’ variable which keeps track of what the next match will be. The game then cycles through the team for the next game and presents the relevant details.

Figure - The home screen showing data about the next match, against Paris. In the right panel, the user team's stats can be seen as well.

Once the next match details are filled out, the team stats panel is also updated. This retrieves the player team’s current record, stats, and best player. It was important to include this as a snapshot of the player’s current situation without navigating to the league standings page.

It was also important to simulate every team’s fixtures during the match simulation, otherwise, the only games being played would be the users. This simulation uses a simplified version of the match engine, finding a match score based on team ratings, rather than a per quarter score, as the user will not be watching these games happen. After a round of matches, the league standings are updated to reflect the result of each match.

The trade finder shows how an unhappy player is less valuable in trades, however, there was no way of decreasing a player’s happiness. Two methods of doing so were then created, the first being every time a game is lost, players lose 3 happiness. A star player over 85 rated would lose 5 happiness, to reflect how the best players want to win as many games as possible, and will become disgruntled quicker if they do not.

The second method is not being in the starting line-up. The average player overall rating is in the low 70’s, so anyone 77 or higher is considered someone who should be starting matches. If these players are left out, their happiness drops by 3, meaning the player must create tactical plans whilst maintaining player happiness, especially if they drop a star player to the bench and lose, as this would subtract 8 happiness.

Happiness is implemented across every team, meaning the user should watch for bargain trades for disgruntled stars on losing teams.

When the whole game was working, the final screen could be added for an end game. This is run when the last game of the season is played and simply tells the user who won the league, and asks whether they want to replay or quit. If they choose to replay the scene will be reloaded, giving a brand new game with a different roster.

# 5.0 Testing

## 5.1 Testing Method

To make sure the game worked as intended, a range of testing methods was performed on the system. The first was Unit Testing, where an isolated component of the game is checked to make sure it functions. This is performed with pre-determined tests (see test log), comparing an expected result to the actual result. Here the boundaries of inputs are also tested, such as whether the rating generator outputs results between 50 and 99. Unit testing is efficient as you know what you are looking for and can give a definitive pass or fail to the test.

The component is then tested as to how it works alongside the rest of the system, this is called integration testing. This ensures every function is connected and works together smoothly. An example of how I used this would be the integration of the trade engine with the roster generation, making sure the generated players were available in the trade finder.

Once the whole system is in place and each function has been tested individually, the whole system can be tested at once. Known as system, or beta testing, this is where the product is tested in every aspect from start to finish. These tests will make sure that the system fits the functional requirements that were outlined at the start of the project.

Most of the above are examples of black-box testing, where the functionality is tested, but not the internal structure of the game. It is also important to perform white box testing, as something may look fine on the screen, but the calculation behind it could be wrong, or a different piece of data is being presented than expected.

Visual Studio and Unity have their own built-in white box testing. This comes in the form of checking syntax for any errors, and not allowing you to run the program if any do exist that may harm the program. They then point you towards the line of the error to try and help. I also performed some of my own white box testing, through unit tests which make sure the correct calculations are made. Also, by commenting the code throughout the scripts, this acts as testing as it allows the developer to fully understand what they have written.

## 5.2 Test Log

The testing log, seen in Appendix 7, was taken from the first deliverable and adapted to include additional features which were not planned at that stage in development. It outlines what test was being performed, followed by the expected result of the test, and then what happened. Each test is then given a pass or fail depending on the result, and if failed, I attempted to fix the error.

A test that failed was that the league standings would calculate how many games behind the sixth playoff spots teams 7, 8, 9, and 10 were. Appendix 6 shows how the test brought up an erroneous result. This was fixed by a new calculation which is performed after the table is set, rather than whilst the standings are still being calculated. This fixed the issue so the test can now pass.

# 6.0 Critical Evaluation

## 6.1 Project Analysis

When initially designing my system, I set out a list of functional requirements. These are the components that are needed for the game to function and serve its purpose. My final product meets every one of these requirements, therefore it serves the purpose it was designed for. However, this does not mean that the game is perfect as there were some features that I planned to implement but are not present in this version of the game.

I believe I was slightly over-ambitious in the plan I had for the game given the time limit available, but I am happy the game works from start to finish with most of the intended features included. My goal for the system was to have every feature implemented well and to focus on quality over quantity of the components, and I think I have stayed true to this thought process.

One of the features which was missing from my final product is a playoff system. My original plan for the game included an additional tournament between the top 6 teams in the league after all regular season matches were played. The winner of this tournament would then be crowned the champion of the league. I left this feature out of the current version of the game as it was very similar to the regular season matches, and I thought my time would be better spent on a new feature which is unlike something which is already existing. The game was also adapted to award the team with the most wins at the end of the season with the championship.

A feature I am particularly proud of is the match engine. As stated in my development, I created several iterations of this component, making it more detailed and realistic each time. The combination of both the match system and line-up editing I think is a very important feature as it allows the user to play the game with more planning, like a real-world manager. If utilised correctly, a player could take charge of a team which has the worst overall rating of the league, but still win every game if they plan strategically and adjust their team to exploit each opponents’ weaknesses.

In retrospect, my research could have been wider and covered more games made by smaller development teams. I think this is what led to me setting slightly over-ambitious goals as I wanted to build a game on a level with others made by huge studios. If I had looked at other games, I might have gained a better idea of what would be a realistic product given the time and resources available.

My choice of development software (Unity and C#) was good as I was comfortable coding in this language and using the interface and tools provided by Unity. Unity provides a wide range of user interface features, which was vital for my game as it uses UI from start to finish. I have also used Unity to build several projects in the past, so I was able to look back at past projects to draw influence or inspect areas of code that I could possibly use in this game.

A small issue I came across while building the UI in Unity was the assigning of objects to placeholders in scripts. The software forces you to manually assign each object to the placeholder via click and drag. When creating the roster screen, for instance, I would have to click and drag each rating slot for each player to the corresponding placeholder in the roster script. There was a total of 80 placeholders to fill for the roster screen alone. The process of filling the placeholders was time-consuming and required a lot of focus as if the wrong placeholder were to be filled, the player data on the screen would be inaccurate. If there was a method to automatically assign each placeholder, this process would have saved a lot of development time as this had to be repeated for the trade and team stats screens.

## 6.2 Potential Improvements

Although I stated the line-up system was one of my game’s strengths, I believe it could be further developed to be even better. At present, the player can view their opponents’ stats and have a rough idea of how to beat them if they play a team to exploit any weaknesses. They do so by adjusting the players in their starting line-up as these are weighted more heavily when determining a match outcome.

The way I would improve this system is through the addition of some artificial intelligence. A player can create a plan on how to beat the opposing team, so I think the opponent should be able to adjust their own roster to match the player’s team. This would take quite a complicated system to try and determine what line-up the player may decide, but it would make the game more unpredictable and difficult.

Another feature that I believe could be improved is the trade system. While it is a good way of acquiring new players or getting rid of unwanted ones, it can be easily exploited. The current system works well in that it only offers players which are of similar value to the player offered by the user. However, this system can be exploited by repeatedly trading up in player value. There will always be a team which gains the stronger player in a trade, as that is how sports work, no players are identical. The problem with the system is that you can always be on the stronger end of the deal, for instance trading an 80 rated player for an 82, then trading the 82 for an 84. By doing this you could gradually build an unstoppable team which would remove any difficulty from the game whatsoever.

This could be solved by another use of AI, by calculating whether the team the trade is being offered should accept the deal. For example, if the team is winning a lot of games and is being offered a player who is worse than the one they give up, they should not accept the trade as it does not help them. On the other hand, a team that is losing a lot of games may be more likely to give up a better player to try something new. Also, trades between AI teams could be developed so that each team is actively attempting to improve, rather than only trading with the user team.

I think the addition of AI would be very important, as at this point the game revolves around the user team. Whereas the simulation should be about the player going into the world of a Basketball league.

## 6.3 Future Work

Generally, I am very proud of the work I have produced over the course of this project, but that does not mean I think that the game is a finished product. Looking at similar games in the industry like Football Manager, they are built by huge teams over many years, with yearly iterations offering new features and improvements. Further developing Basketball Management Simulator is something I have a lot of interest in as it has been fascinating to make.

The first feature I would introduce if I were to keep working on the game would be the implementation of seasons. Something that draws a lot of players into games like this is the aspect of a long-term plan to achieve success, for example, taking some of the worst teams in the game right to the top of the world. This feature would create many new scenarios and problems for a manager in the game.

A players age would factor into their trade value, as you would not want to give up a lot of assets for a player who is at the end of their career. You could also watch young prospects in the game turn into stars over the course of several seasons. This would pose the question of whether to sacrifice young potential to push to win matches now, or whether to be patient and build a better team in a few season’s time.

In professional basketball, there are three ways to acquire new players. Those are through trading players, signing them in free agency, or drafting a young player who is entering the league for their first season. As my game only runs from the start to the end of one season, there is no off-season period for the latter two methods to occur. The addition of a multiple season system would allow these to be introduced and give the player more ways of acquiring players without giving away any that they currently have. In fact, this would open a whole new branch of features such as scouting a potential draft pick to gain an idea of their stats or having to manage following salary rules when signing a free agent.

An example of these rules would be a salary cap. To be fair to all teams and create the most competitive league possible, the National Basketball Association (NBA) of America implemented a salary cap of $109 million per team. This means that if a team wants to sign a player through free agency, their total team salary cannot cross this amount. This increases fairness in the league as the richest teams cannot outspend their competitors to win more games and allows teams that cannot generate as much money to keep hold of their star players.

The implementation of this rule in my game would lead to interesting scenarios where a player will have to plan out their budget and decide whether to spend all their money on one big signing or spread it over several players.

I believe the random roster generation is a good component for this version of the game. As the user is limited to playing as one team, having the same players every time would become repetitive and boring and makes the game easy to win as you could make the same moves every time you play. However, in the long run, I would like to introduce the ability to play as any team in the game and have one specific roster, or a choice of rosters available. This would create ‘cult heroes’ among the players in the game, who look to be an average player but somehow become legends. Chris Wright (ESPN, 2020) wrote an article on the “All-time Football Manager cult XI”, all of which had quiet careers in real life, but in the game, were very good.

Another addition I would like to make is a conversation system. At the moment, if a player is unhappy, an issue will pop up in the sidebar on the home screen warning the user of the player’s low happiness level, they can then either attempt to help the player by winning matches, or playing them in the starting line-up, or they can trade the player away to get rid of the possible bad influence. A conversation system would give more options for solving a player’s problems, such as promising to fix the issue, causing a short-term happiness jump.

The second conversation system would be with the team’s owner. The owner would be able to set the user short- and long-term goals, including winning x number of games this season, trade x player away, or win a championship within 3 years. This would not only give the player the objective to win games but to also keep the owner happy whilst doing so, as they may be fired otherwise.

All the above additions not only add to the enjoyment and range of features in the game, but also to the realism. Whilst the purpose of a game is to be fun, a simulation game is supposed to be “one that simulates an experience” (Maximum Games, 2016). Therefore, it is not only important that each feature added to the game is enjoyable, but also reflective of real-life basketball management.

# 7.0 Conclusion

Overall, I think the project was successful. As stated in the project analysis, the game meets all the functional requirements outlined in my first deliverable and achieved a lot of the additional features which I wanted to include to increase the quality of the game.

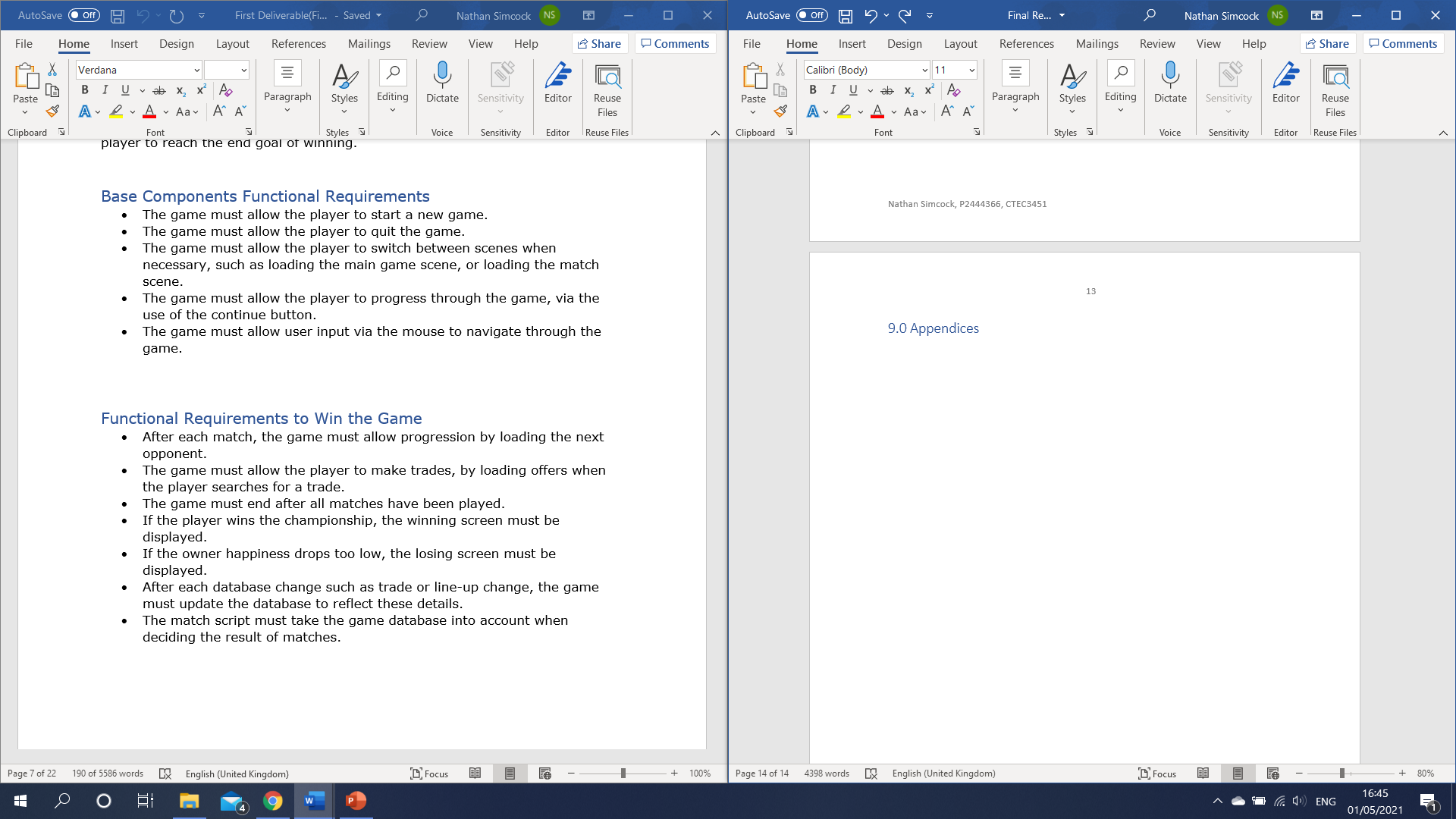
While the game is fun to play, it does not deviate from its purpose of being a realistic simulation of the job a Basketball general manager performs in real life. I think the game finds a good balance between realism and entertainment, which is vital for keeping the player interested.

As I mentioned in the introduction, the purpose of this project was to reflect the aims and outcomes of my course, computer games programming. It would be impossible to cover every aspect I have learned throughout my studies, as we were taught many different techniques, using different software, for different situations. However, I feel that the system I have created demonstrates a lot of things that I have learned in the last three years. I think Unity was a good engine to develop the game with as on the front-end it provides tools that helped me to build the UI to the designs I had created, and on the back end, it allowed me to write efficient code to make the UI work as intended with C# and Visual Studio.

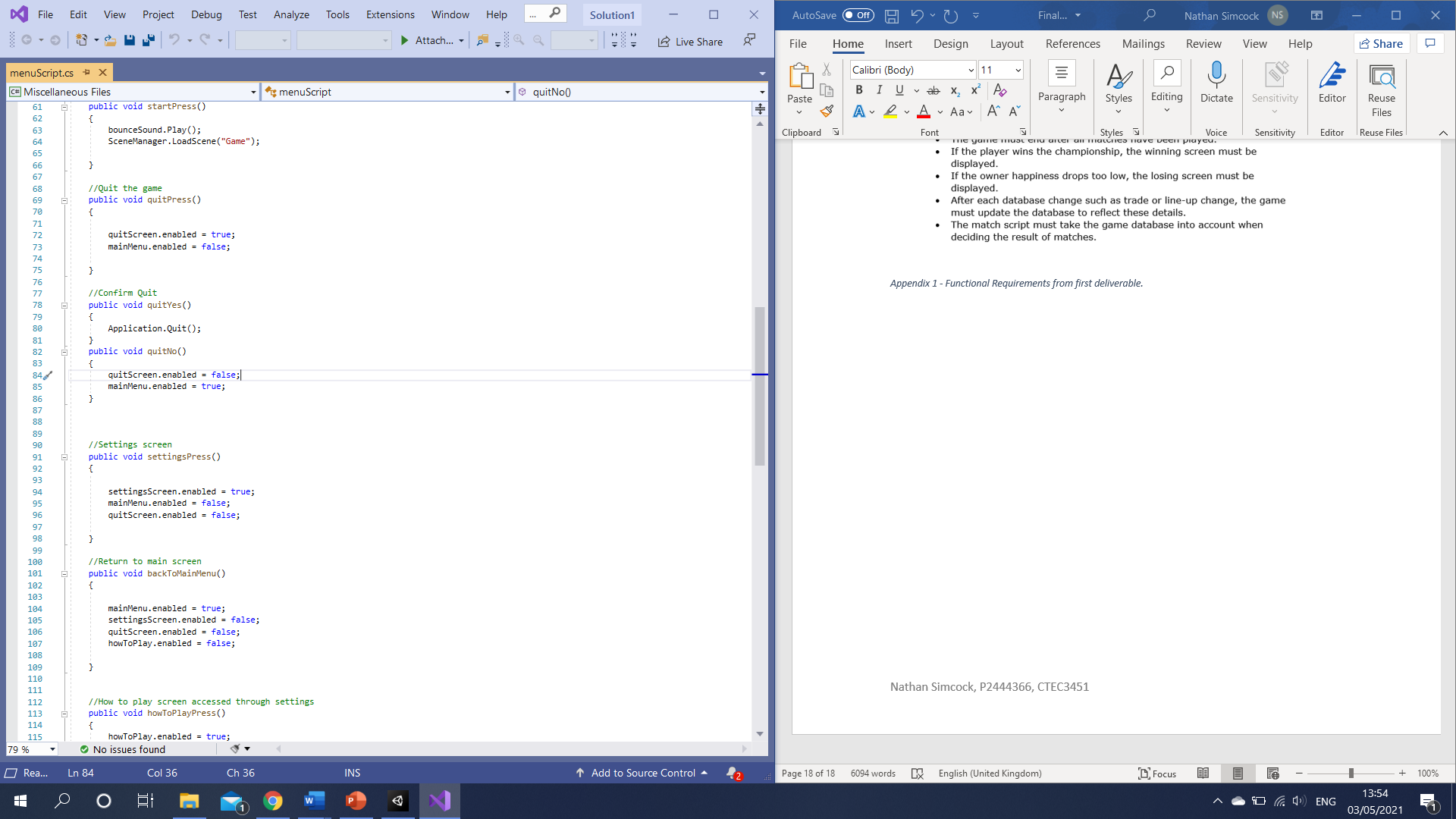
# 8.0 References

1. Bailey, D. (2021) Football Manager sales top 33 million. Available at: <https://www.pcgamesn.com/football-manager-2021/football-manager-sales> (Accessed 01/05/2021)
2. Lerouge, A. (2020) What Makes a Good Simulation Game? An Analysis. Available at: <https://thevaultpublication.com/2020/03/13/what-makes-a-good-simulation-game-an-analysis/> (Accessed 30/04/2021)
3. Maximum Games (2016) PLAYING WITH REALITY: WHAT MAKES A SIMULATION GAME? Available at: <https://www.maximumgames.com/press/playing-with-reality-what-makes-simulation-game/> (Accessed 29/04/2021)
4. Nigretti, A. (2018) Making the most of TextMesh Pro in Unity 2018. Available at: <https://blogs.unity3d.com/2018/10/16/making-the-most-of-textmesh-pro-in-unity-2018/> (Accessed 02/05/2021)
5. Takahashi, D. (2021) NBA 2K21 pushes Take-Two to $860.9 million in Q4 2020 revenue, down 7.5%. Available at: <https://venturebeat.com/2021/02/08/nba-2k21-pushes-take-two-to-860-9-in-q4-2020-revenue-down-7-5/#:~:text=In%20an%20analyst%20call%20today,player%2C%20said%20CEO%20Strauss%20Zelnick>. (Accessed 01/05/2021)
6. Wright, C. (2020) *‘Who makes the all-time Football Manager cult XI?’*, ESPN. Available at: <https://www.espn.co.uk/football/blog-the-toe-poke/story/3995925/who-makes-the-all-time-football-manager-cult-xi> (Accessed 29/04/2021)

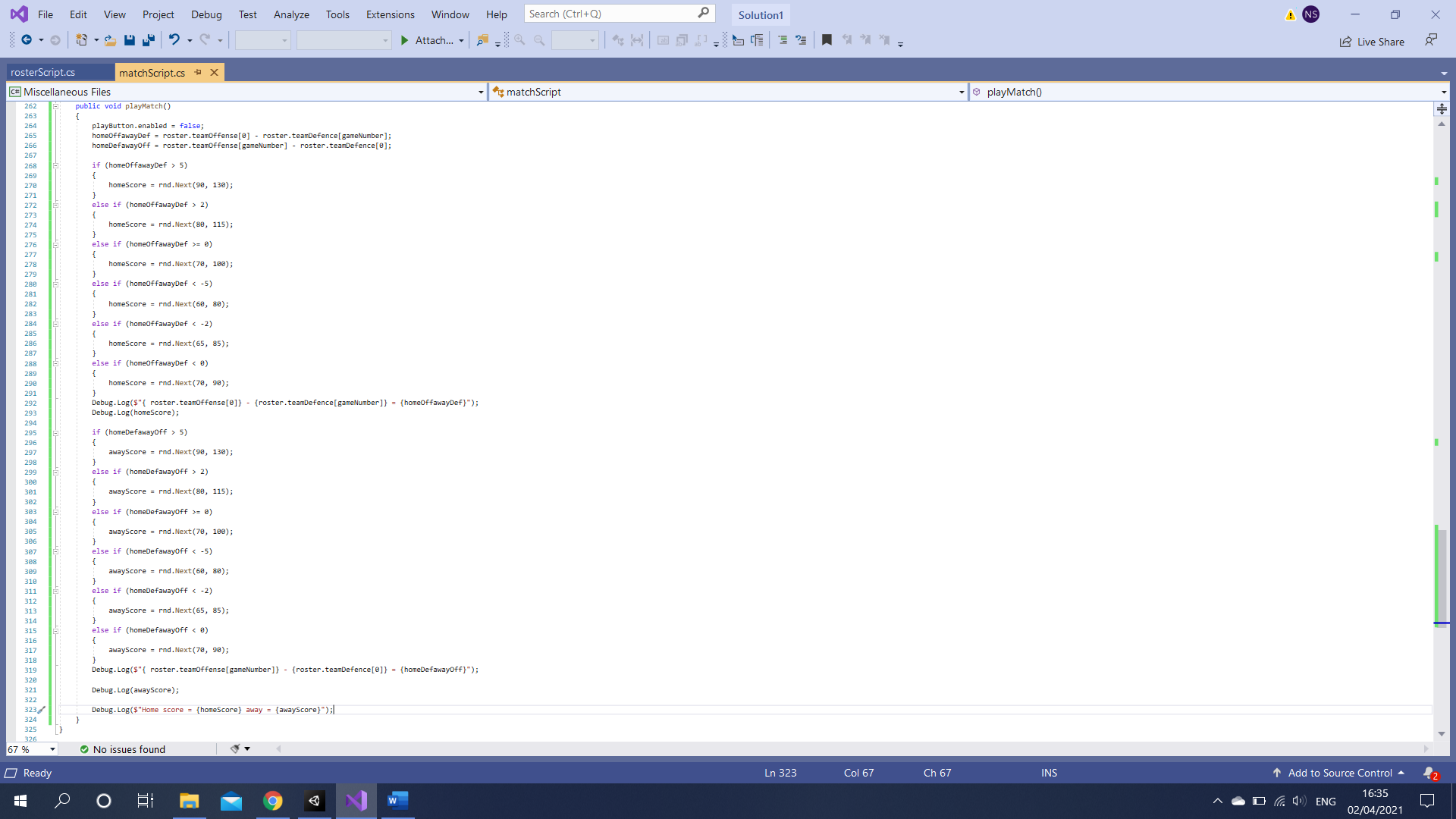
# 9.0 Appendices



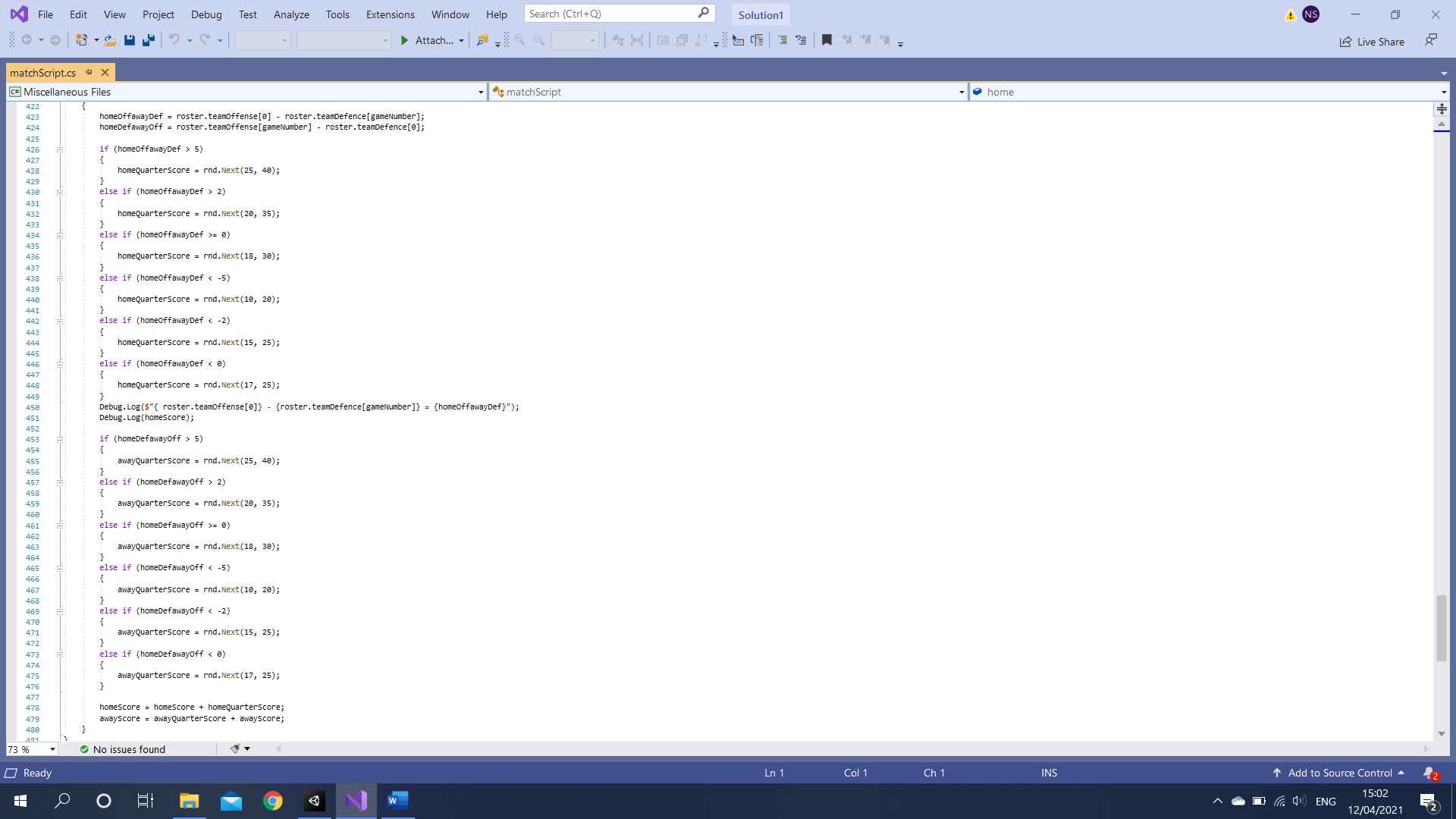
Appendix - Functional Requirements from first deliverable.



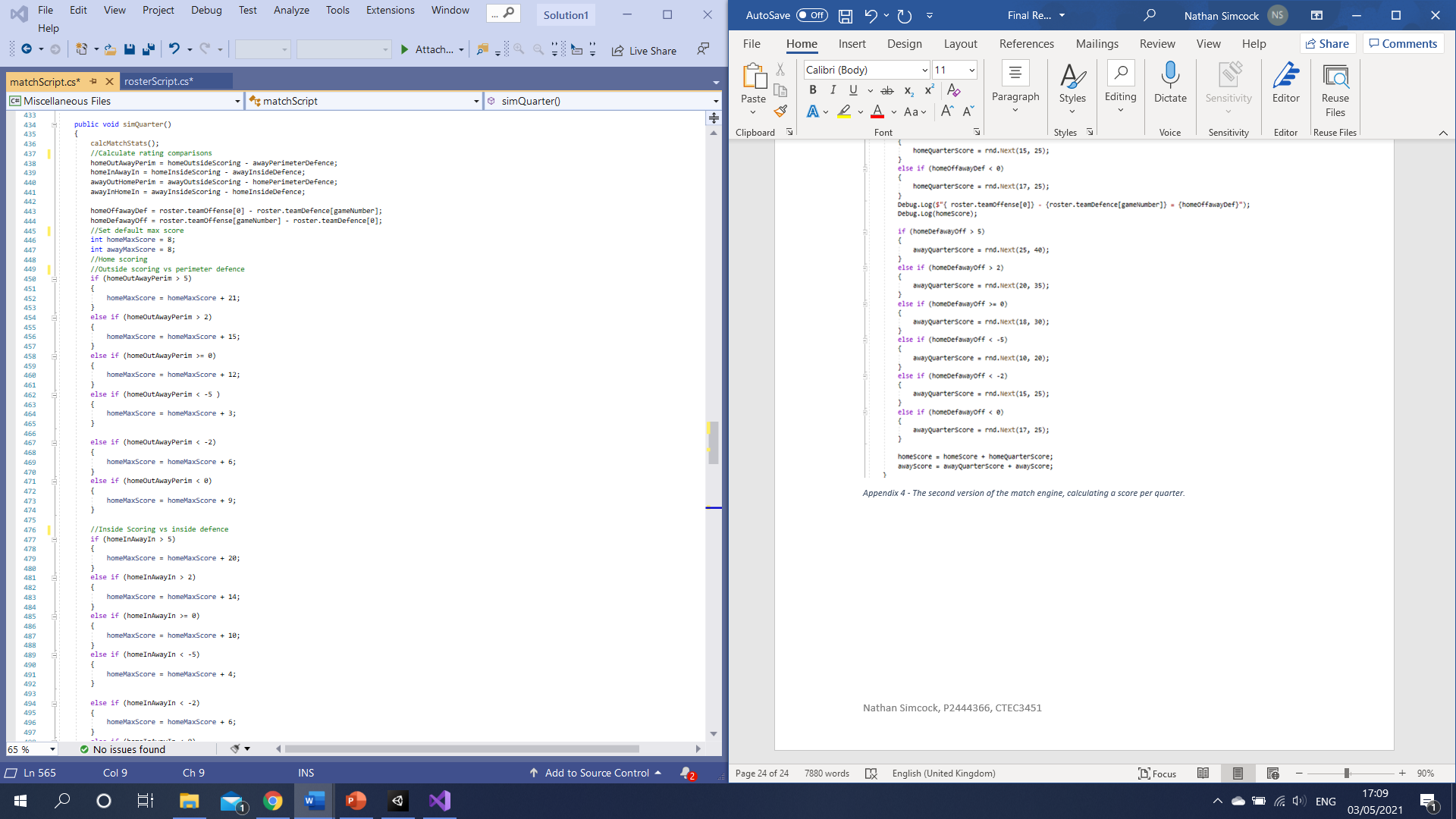
Appendix - Some of the canvas switch functions which were applied to button presses.



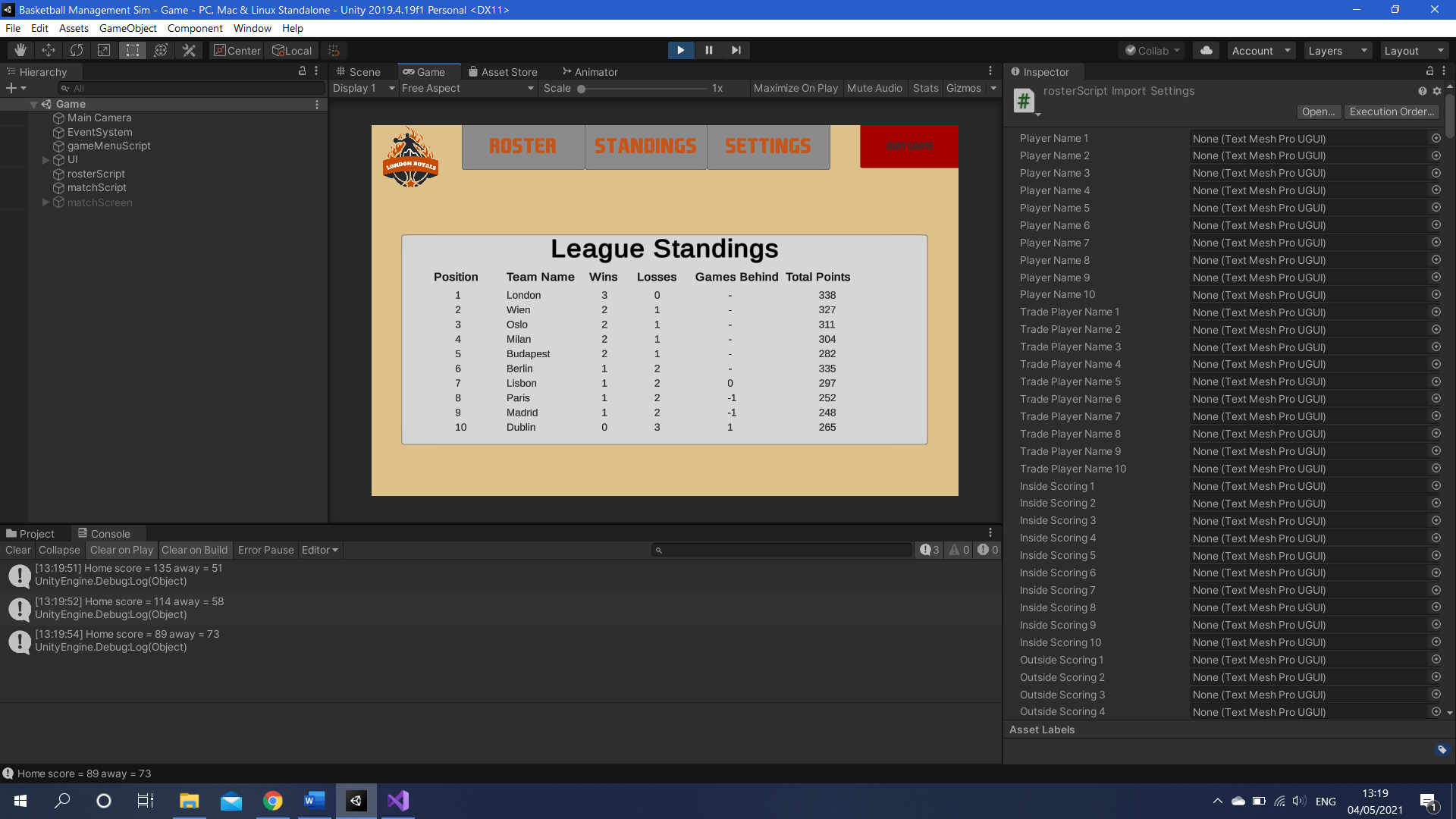
Appendix - The first match engine, which simply compares team offense to team defence and generates a score for the match.



Appendix - The second version of the match engine, calculating a score per quarter.



Appendix - The final version of the match engine. Incorporates more team stats and then adds to the maximum possible score for a quarter. This is then repeated for inside scoring. These scores are then added to decide a winner.



Appendix - An example of where testing can spot issues, in some playthroughs the games behind was giving erroneous results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **Process** | **Expected Outcome** | **Actual Outcome** | **Pass/Fail** |
| Clicking the button for the current page doesn’t reload the page. | Click the button corresponding to the page which is currently displayed. | Current page is not reloaded. | The screen is not reloaded and nothing happens. | Pass |
| Quit Confirmation screen, to prevent accidental quitting. | Click the quit game button. | A menu pops up asking if user wants to quit. | Menu asks “are you sure you want to quit?” user can then select yes/no. | Pass |
| Every button across the top bar leads to the correct page. | Click the button for each page across top bar. | Game navigates to the correct screen. | When each button is pressed, the corresponding page is loaded. | Pass |
| Records are updated after each match according to result. | Play a match and return to home screen. | Team record is updated with new result added. Records for all other teams are updated in standings. | After every match, the updated record can be seen on the home page and league standings. | Pass |
| Menu screens are updated to match the next opponent after each match. | Finish a match and return to home screen. | The next opponent section of screen is updated. | When you return from a match, the details on the home page are altered for the next match in the schedule. | Pass |
| After a trade, each teams ratings are recalculated and roster is adjusted. | Make a trade. | On the roster screen, the players are switched and ratings are updated. | The two selected players exchange places in the database. Team ratings are then updated to reflect the change. | Pass |
| When a goal is met or fails, this is updated on the home screen. | Pass or fail a goal. | When a goal is passed, this is reflected and happiness for relating player or owner increases. If failed the happiness falls. | No owner in final version of game. | N/A |
| Once all regular season games are finished, the game advances to the playoffs if the user team qualified. | Play all matches up to the playoffs. | If team qualifies, next match screen is updated with the playoff games. If not, game over. | No playoffs, however after the last game of the season, the championship is displayed and presented to the player if they have the most wins. | Pass |
| If owner happiness falls too low, user is fired. | Make the owner happiness fall below threshold. | Pop up conversation with owner where player is fired and game over screen appears. | No owner in final version of game. | N/A |
| The result of winning the playoffs. | Win all playoff games. | Team wins the championship and winning screen appears. | Winning screen appears if player has the most wins in the league, if not, losing screen appears. | Pass |
| Player happiness is updated after each game. | Play a game and then navigate to player happiness tab. | If the team won the game, player happiness rises, if they lose then it falls. Higher rated players who are not in the starting lineup also have happiness fall. | When a team wins, happiness rises, when they lose it falls. Higher rated players fall quicker. If a star player isnt in the starting lineup, happiness falls. | Pass |
| The manager receives a warning when a player’s happiness is low. | Bench a star player and lose games to make happiness fall. | An issue will appear telling the user that the player is unhappy. | Up to 3 players with happiness under 50 are shown to the user, along with their happiness level. | Pass |
| When a player’s happiness falls too low, they demand to be traded. | Continuously bench a star player until their happiness falls below threshold. | A message appears in player issues with player demanding a trade. | No conversation system in final version. | N/A |
| When trying to trade a player, the user is presented with a set of offers. | Go to the trading screen and select player to trade. | At least 3 offers appear for the player, user can then select which trade they want to perform. | All players who fit the trade value are presented, and the user can click through the options. Offers are based upon trade value. | Pass |
| When a game is played, results follow set boundaries. | Play 5 matches and observe results. | Teams scores must be between 70 and 140, star player can’t score more than 70, in most cases the stronger team wins, but there can be upsets. | Updated match engine has new limits, but results always fit the boundaries. Any team can win any game no matter their strength. | Pass |
| Testing the full screen option. | Going to settings and pressing the full screen button. | Game goes into full screen mode. | On full screen click, full screen toggles on/off. | Pass |
| Mute button can toggle sound. | Go to settings and press mute button. | If sound is on, it is toggled to off and vice versa. | On mute click, sound toggles on/off. | Pass |
| Quitting the game closes the application. | Go to main menu and press quit game. | If user presses yes to quit confirm, the game closes. | On confirmation, the application is closed. | Pass |
| Game can return to the main menu | In game, press quit game in top right corner. | Confirmation screen pops up, if yes pressed, main menu is loaded. | User confirms quit, main menu scene loads. | Pass |
| How to play screen explains all features. | Go to settings then how to play. | Each feature is explained in basic terms. | There are multiple categories, one for each part of the game, each explains the use and how to use. | Pass |
| As matches are played, roster is updated with new trade values. | Play some matches then view the trade finder. | Unhappy players have lower value than those with no issues. | Before matches are played, trades are always within 3 rating. After a few games, this widens as the number of unhappy players grows. | Pass |
| The trade finder cycles through offers. | Reach the end of the trade offers and press next/previous offer. | Return to the first/last offer. | Once you reach the last or first offer, pressing next/previous goes to the other end of the loop. | Pass |
| The league standings table sorts the teams in order of wins. | Go to league standings table after some games are played. | Most wins are at the top, least at the bottom. | Teams shown in order, if tied the team with most points comes first. | Pass |
| The league standings shows how many wins the 7th, 8th, 9th and 10th teams are behind the final playoff spot (6th). | Go to league standings after some games are played. | The number of games each team behind 6th is calculated and shown. | Incorrect results being displayed, some show minus numbers. | Fail |
| (RETEST)  The league standings shows how many wins the 7th, 8th, 9th and 10th teams are behind the final playoff spot (6th). | Go to league standings after some games are played. | The number of games each team behind 6th is calculated and shown. | The game now calculates accurately how many games each team is away from the 6th spot. | Pass |
| The team stats page shows the average ratings for every team. | Go to team stats screen. | Each team’s average ratings are displayed. | After originally showing just team offense, defence and overall, the new match engine requires averages of every stat so these are all shown here. | Pass |
| The line-up editor allows the user to change their active starting line-up. | Go to line-up editor and make some changes. | The incoming and outgoing player are switched on click. | You select the incoming and outgoing players and their places are switched. This is then updated for the match engine. | Pass |
| No ratings are generated outside of the selected boundaries. | Run the game a few times and check player ratings. | No player has ratings higher than 99 or lower than 50. | No player has ratings higher than 99 or lower than 50. | Pass |
| Each team is balanced and given at least one star player. | Run the game a few times and check player ratings. | Each team has at least one player with ratings over 80. | Every 10th player (one per team) in the game is given ratings over 80. | Pass |
| Once the final game of the season finishes, the player is taken to an end of season screen. | Finish a season. | The winning or losing screen is displayed. | The player is told who won the championship, with a unique screen if it was them. They are then able to replay or return to the main menu. | Pass |
| If a match is tied, an overtime period will be played. | Play some matches. | The tied score will be shown, then the overtime will take place and decide a winner. | An overtime period takes place and the score is updated. | Pass |

Appendix - Test Cases and results.