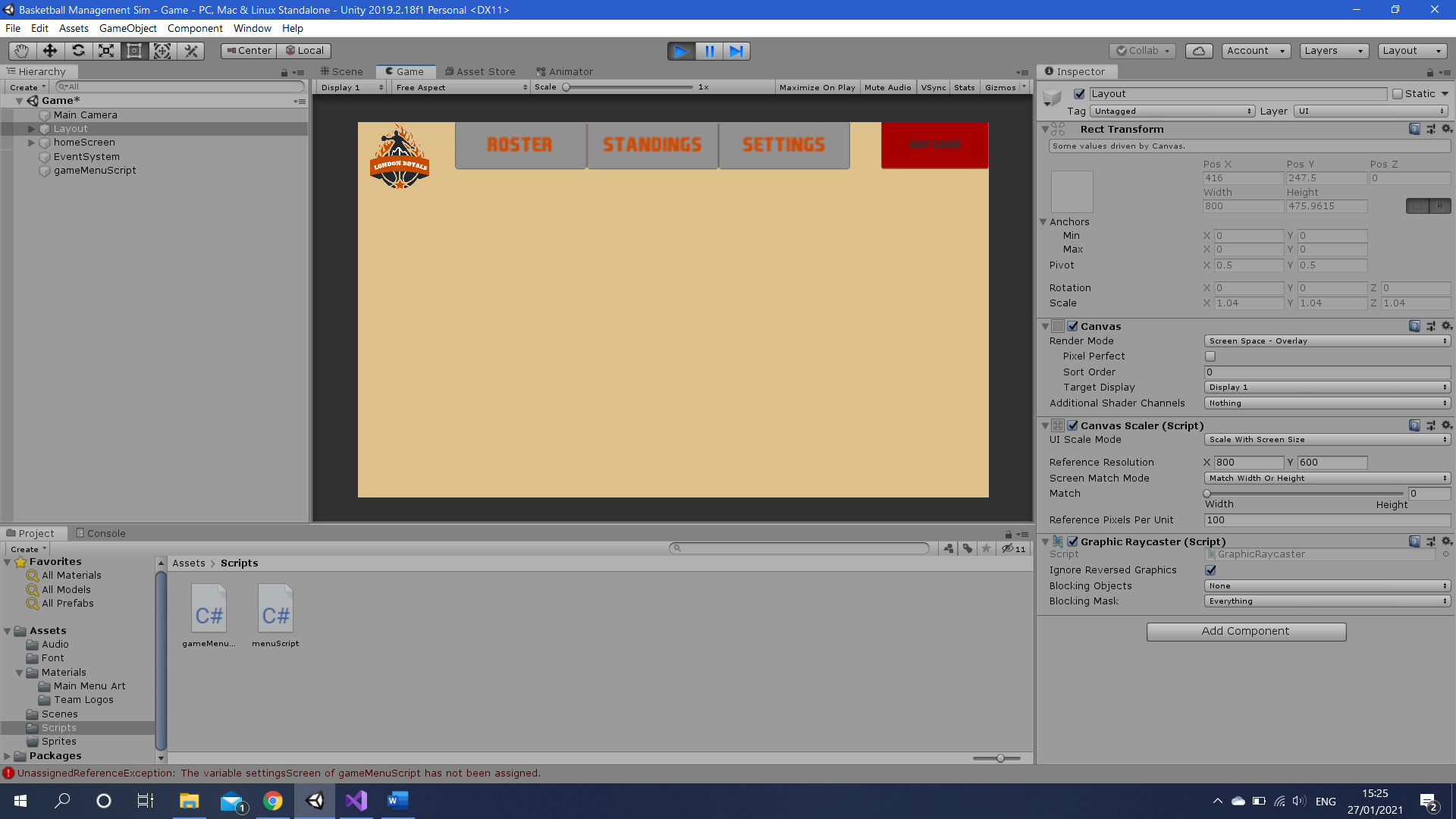
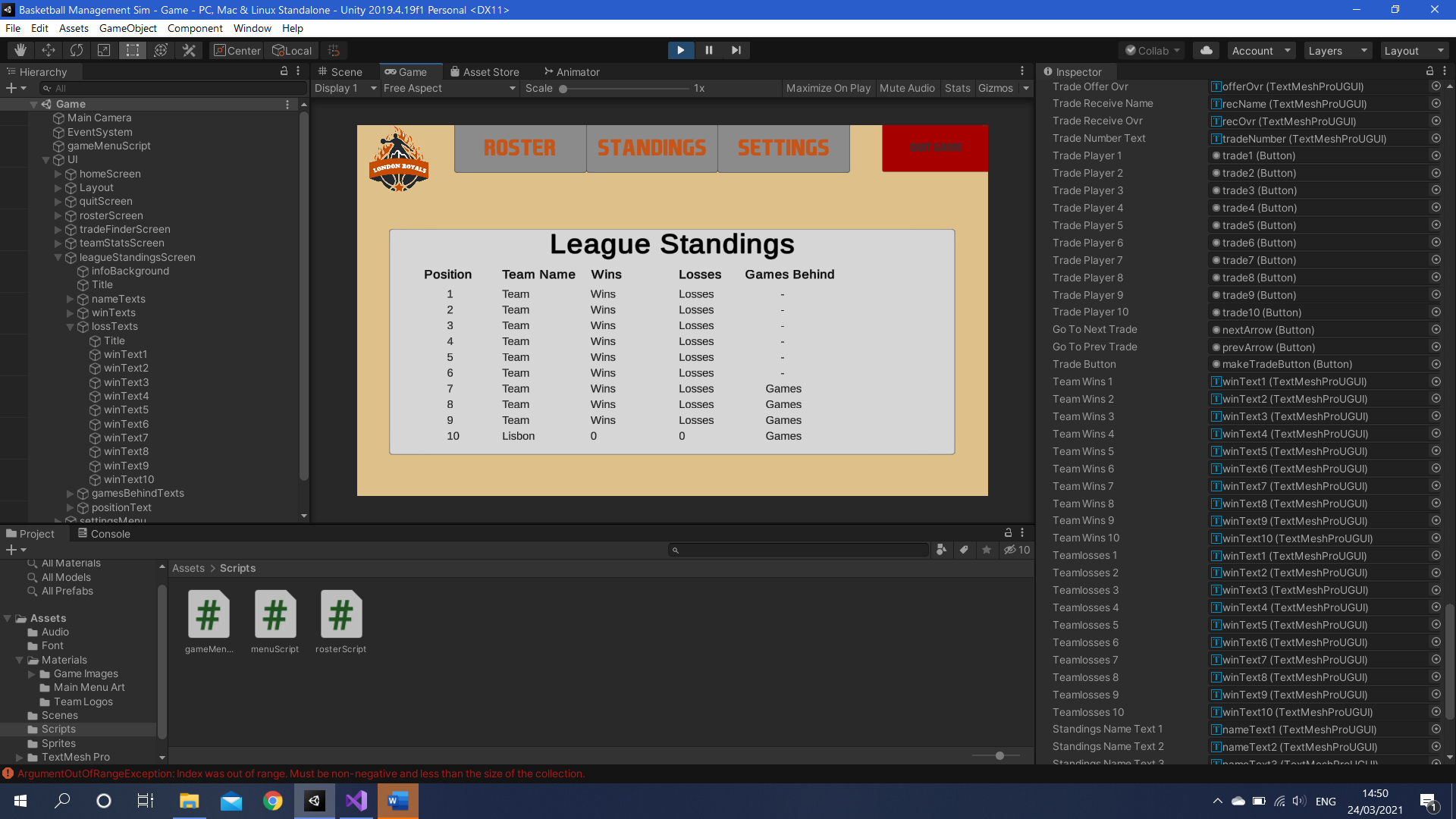
**Development Journey**

First issue – when building the UI, it wouldn’t scale to the screen size. When run in full screen or anything other than Unity game view, everything would be small and background wouldn’t cover the whole screen. Fixed via the use of the canvas scaler. This takes everything on the canvas and scales it to the size of the screen.

When building the home screen, I separated the buttons which would appear on every screen into their own canvas, so they wouldn’t have to be duplicated. This caused a problem with new items being placed behind the background and out of view. For example, the home screen would load up without the continue button and info panels. This was resolved through the use of the sort order function of the canvas, which provides a layered effect.



Another problem I came across was the loss of quality across text in the game. This was in text boxes as well as buttons. The cause of this problem was that I had been enlarging text to fit the size I wanted, rather than changing the font size.



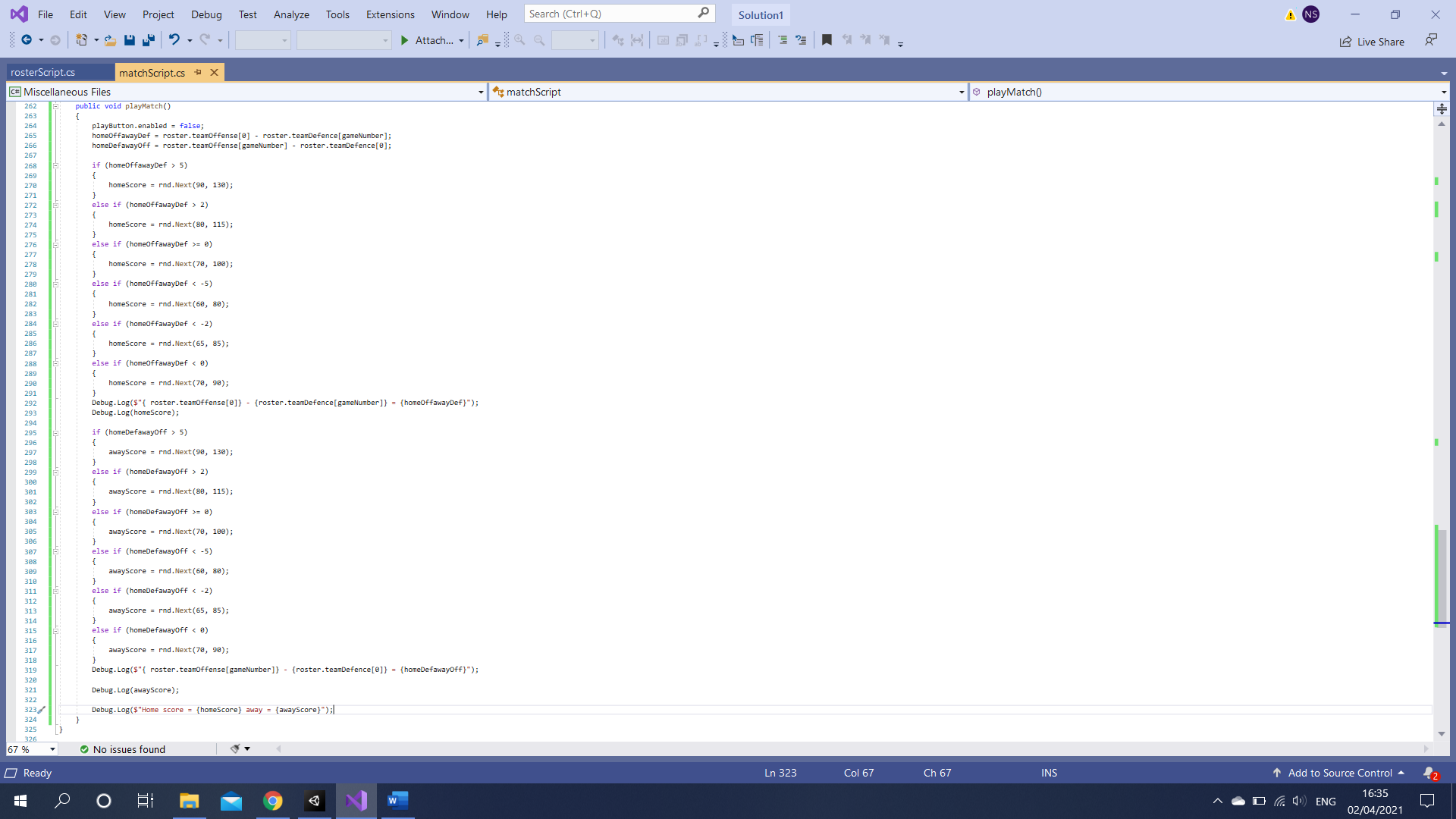
When developing league standings, if teams wins were equal, only the last team to be checked would be placed in a position. Lisbon is the last team being checked, all teams on 0 wins.

**Dev Process:**

Started with the main menu. This is the first screen the user comes across and enables the playing of the game so is a logical place to start. Doesn’t rely on other features being in place.

Once this was built I moved on to the menu system of the game. Once in a game, this is the first thing that will be required, so the player can navigate screens. Built a homescreen and the navigation with sub menus. Most pages don’t have data yet as they rely on the database to be put it in place but they have templates for where data will be shown.

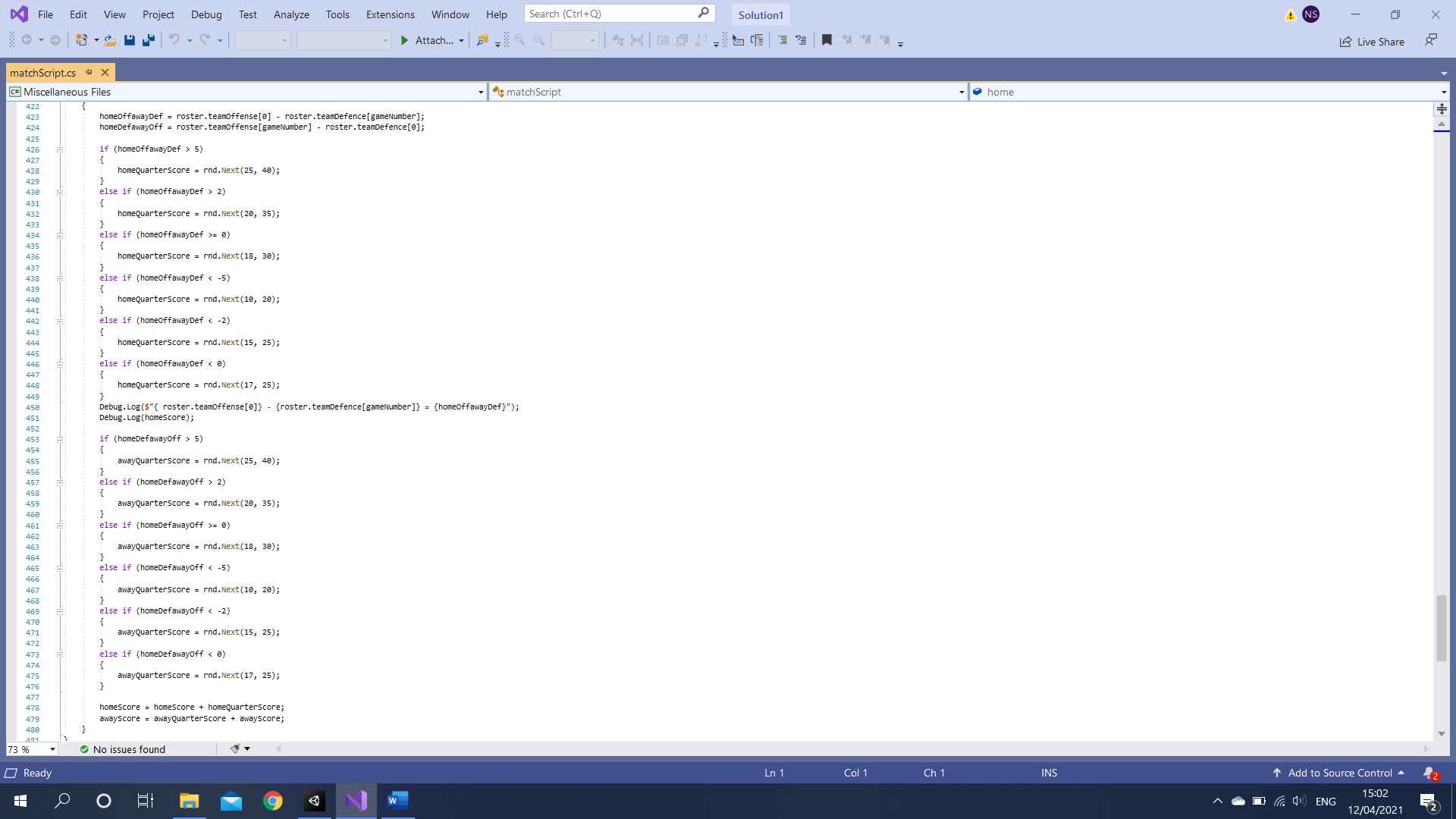
Next step is the database. I started by creating a series of lists. Each list contains an attribute about a player in the database, e.g. name or playmaking ability. The way the lists work together is via their index number, a player’s data will be stored in the same slot on each list. If there were to be a trade, player a’s data will be swapped with player b’s data, as a team will consist of players in a certain index range on the list. This is where I developed an efficient way of creating the database. Rather than hand entering every single player’s data, I created a random generator to do this for me. Not only does this turn 100+ add statements for each list in one loop per list, it adds replayabilitly and varying levels of difficulty to the game, as no game will be the same when you start it.



My match engine was developed in several stages. The first stage can be seen in the code above, where the game takes the overall offense of each team and compares it to the overall defence of their opponent. These two calculations then effect how many points each team scores, which is decided by a random number in a set range. The team with more points will then win the game. However, I plan to develop this further, firstly by implementing 4 quarters to match the structure of a basketball game.

It is understandable that the stronger team in a match will be expected to win, however there can be some upsets. The nature of the current system will be more likely to produce an upset as there is an equal chance of the bottom number in a bracket being picked than a higher number. By implementing 4 quarters, I can create 4 simulations, with a score for each quarter, then add them together. This will still leave the possibility of an underdog winning a match, however in 4 simulations, the stronger team should have more points in at least 2-3, making the game more realistic.

The current system is quite simple in that it just compares the home team offense to the away team defence, and then the opposite. While this does reflect the method of winning, by outscoring your opponent, it completely ignores a huge factor, tactics. There are countless instances in basketball of an underdog beating a stronger team due to a successful game plan. Therefore, in a future update to the match engine, I would like to rework the scoring system to take more stats into account, such as a team 3-point rating or team perimeter defence. This would allow for extra planning in games, such as using a more outside scoring-oriented line up to exploit an opponent who are poor at defending on the perimeter.



When switching to the quarter-based approach, I used the above values for minimum-maximum scores. While this succeeded in making the stronger team more likely to win due to performing 4 simulations instead of one, the stronger team won almost every game. This is because of the minimum scores, for example, a team whose offense is 5 or more higher than their opponents defence is guaranteed to score 25 per quarter, whereas a team with a differential of 0 or less cannot score more than 25. No matter the team strength, a team should never be guaranteed to win a match before it starts, as this removes the excitement and realism of the game where an underdog can win at any time. To counteract this, I balanced the score ranges, lowering the minimums for some of the higher bands, as even the best teams can have a bad game.