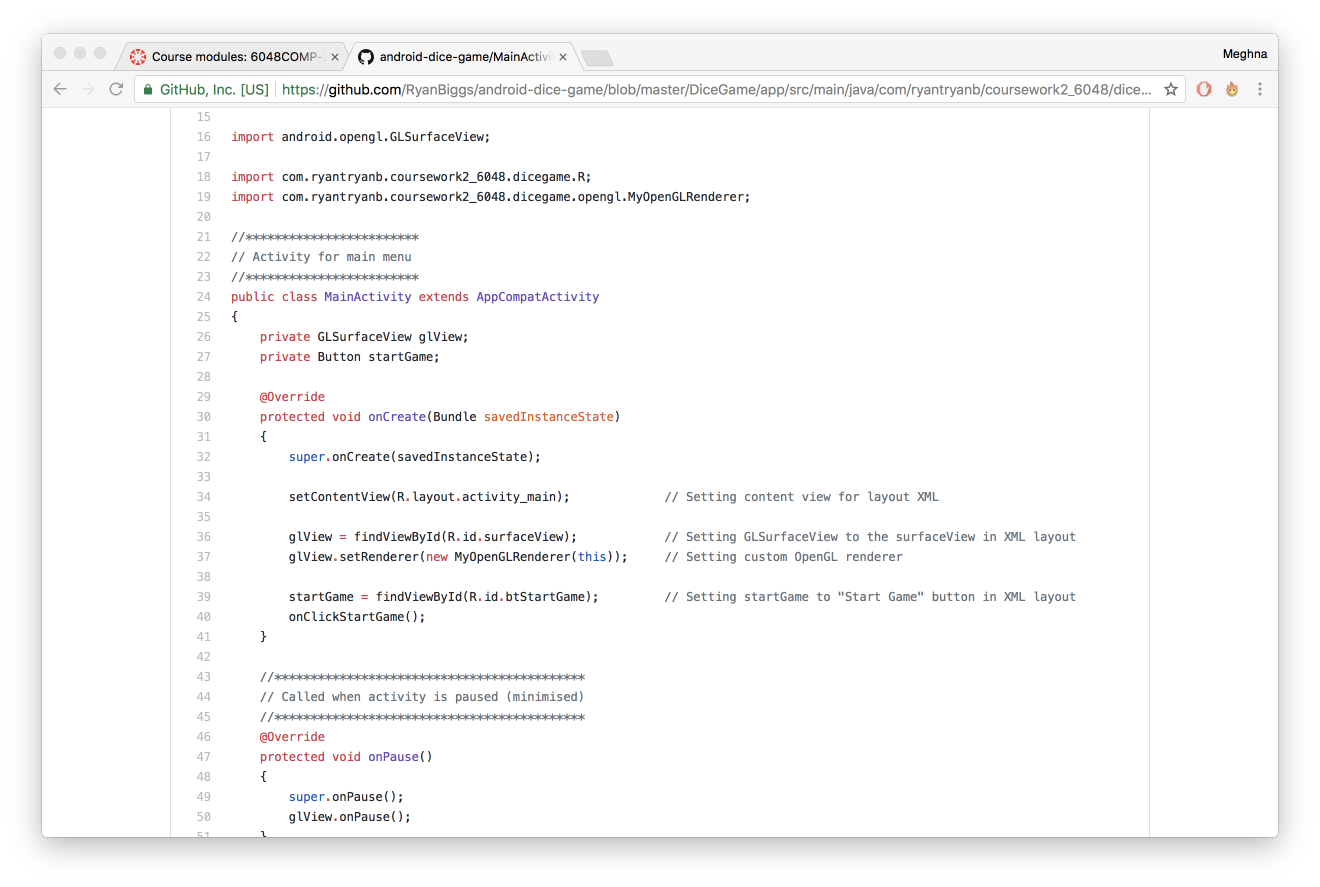
**Development Activity Report**

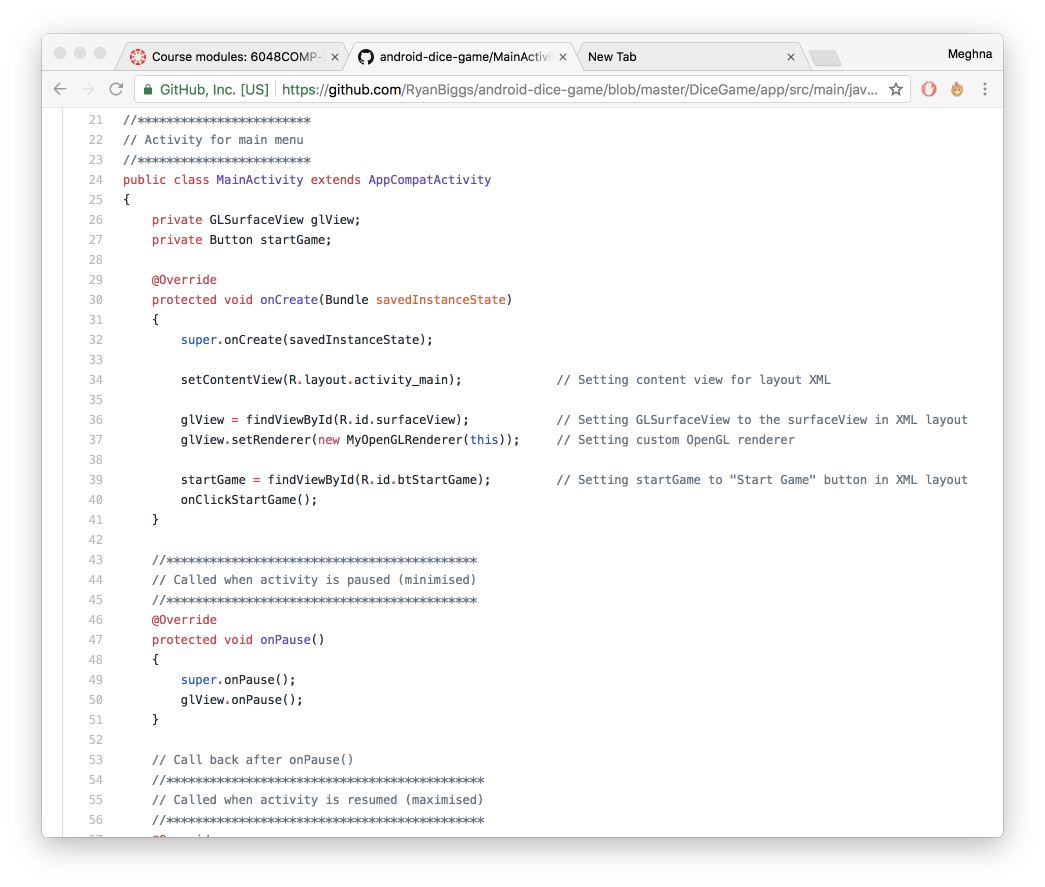
Porting is a process of adapting software. It allows us to adapt the software so it can run on a different platform that the software was not intended to be used on. In this coursework, we will be porting the existing dice game, which was developed to be used on Windows NT operating systems using .NET Framework, for Android Smart Phones.

Firstly, we examined the Graphical User Interface as this is what the user would be able to see. For the main activity, we used OpenGL ES. OpenGL is a cross platform graphics API for high performance 2D and 3D graphics and is supported by Android. We created our own OpenGL renderer class which allowed us to override the default class so we could achieve more detailed customisation when it came to things like display sizes and aspect ratios for the application.

(Look at the screenshot below)



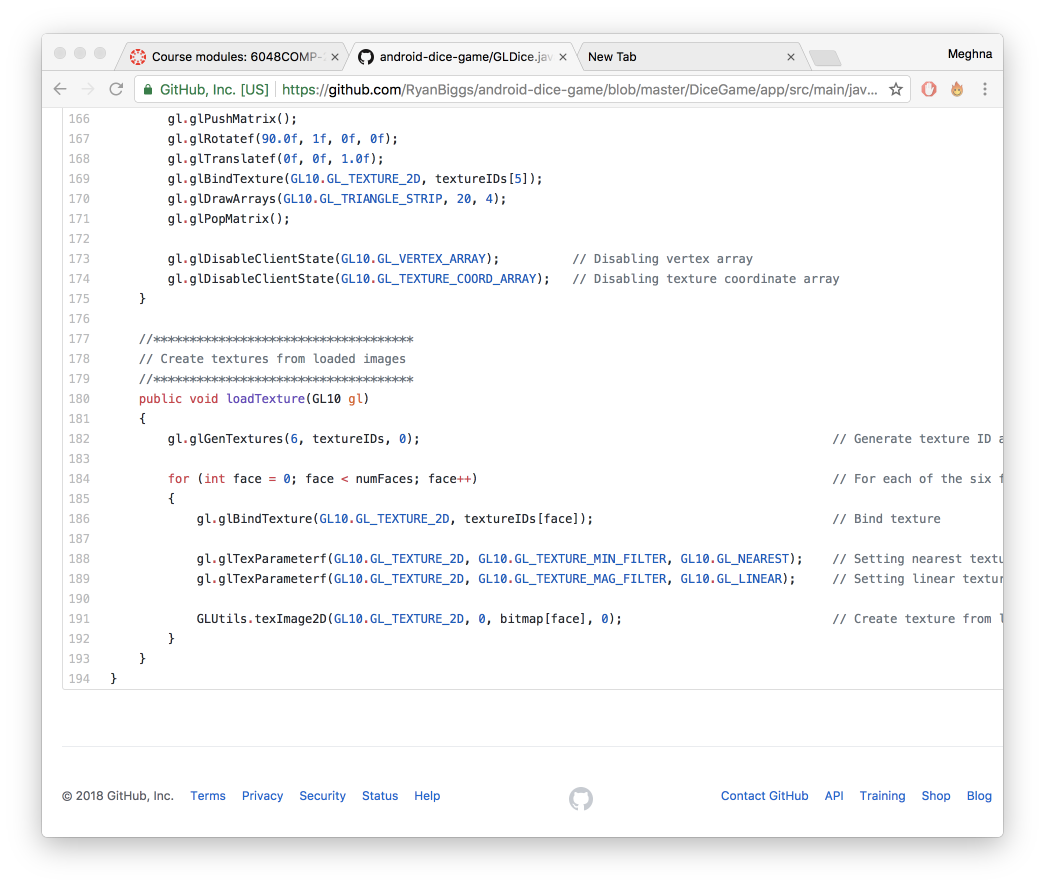
Once the OpenGL renderer was complete, we had trouble displaying it in the app’s main activity, we could only display the XML layout, or the OpenGL view, not both. This was because setting the view to the XML and to the OpenGL used the same code, meaning only one could be set. After some research, we discovered the surfaceView class. This class provides a dedicated drawing surface that is placed inside an XML view. After reading through the surfaceView page on developer.android.com, we were able to create a surfaceView within the XML layout, and link it to the OpenGL renderer we made. Now, when setting the view to the XML, the OpenGL renderer could be seen through the surfaceView, which was set to fill the screen.

(See screenshot below)

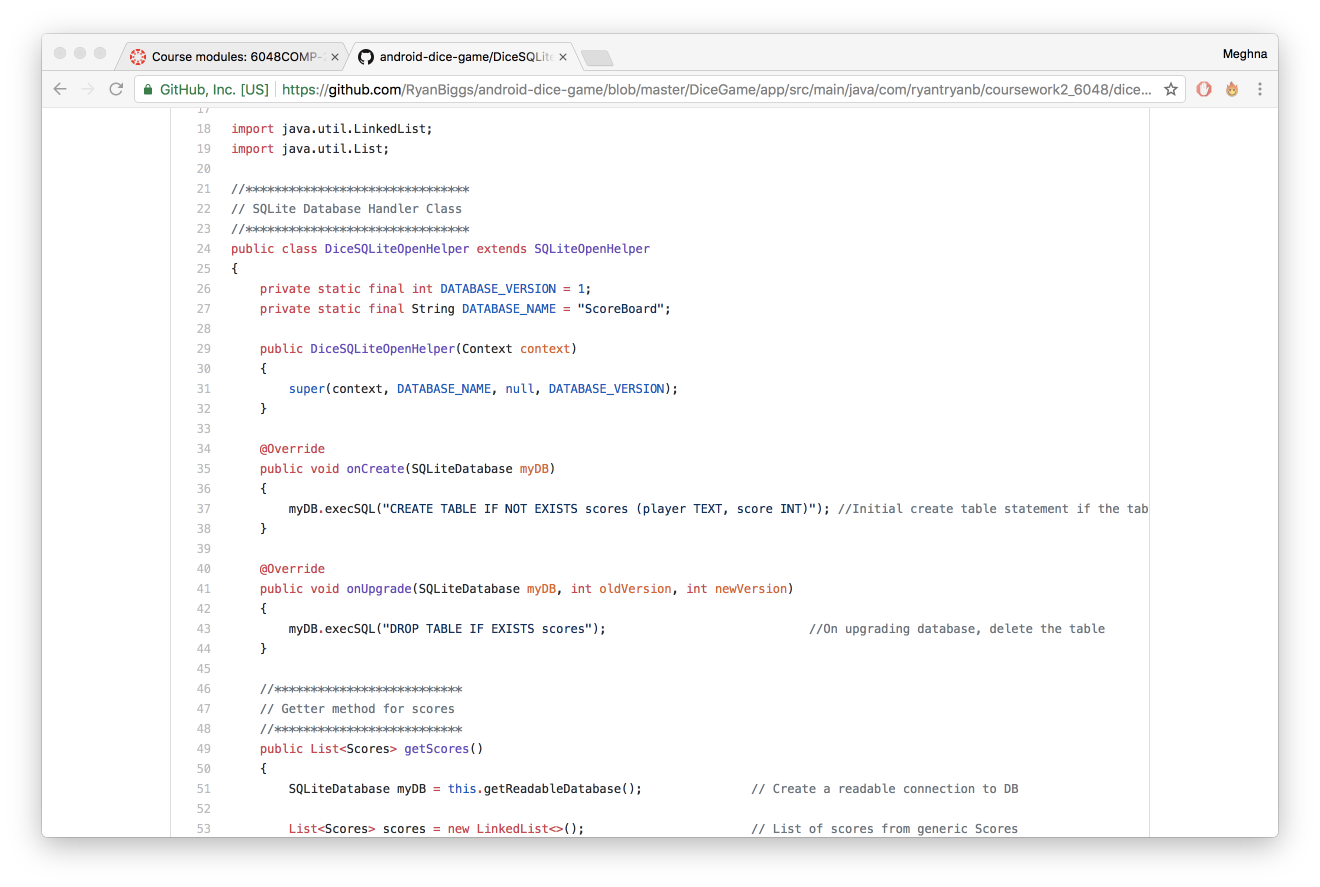
Once the custom renderer class was set up, we made a class to handle the functionality for each die.

The OpenGL renderer class handles the rendering side of things, like creating the surfaces, what happens when a surface is changed, or when each frame is drawn. The Dice class handles the functionality for the creation, drawing and texturing of the dice.

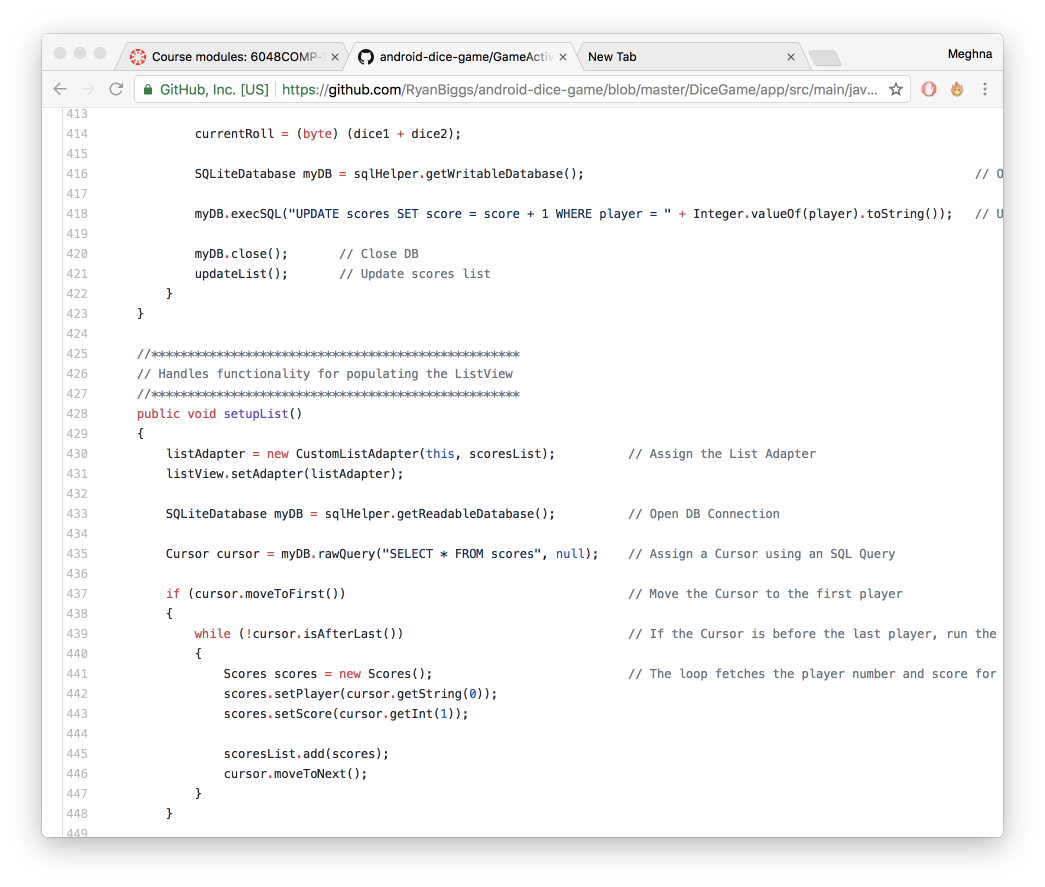
Here is where we ran in to the main problem, which also turned out to be the easiest to fix, although it took quite some time to come to the solution. The dice were being created perfectly, everything worked exactly as it should have, but the textures didn’t display. Originally, we used a single texture map for the dice faces, each face using different co-ordinates, so they would use a different part of the texture. Because of this, we thought the UV co-ordinates were wrong, because the faces were white, even when colouring the texture black. However, this was not the case, and after many hours of work and research, we discovered that we hadn’t set up any texture filtering for the cubes, and this was crucial to getting the textures to display. We had already changed the code to use six individual texture images, rather than the single texture map when trying to fix the problem. So we just left it like that, save spending more time changing it when it already worked.

(See screenshot below)

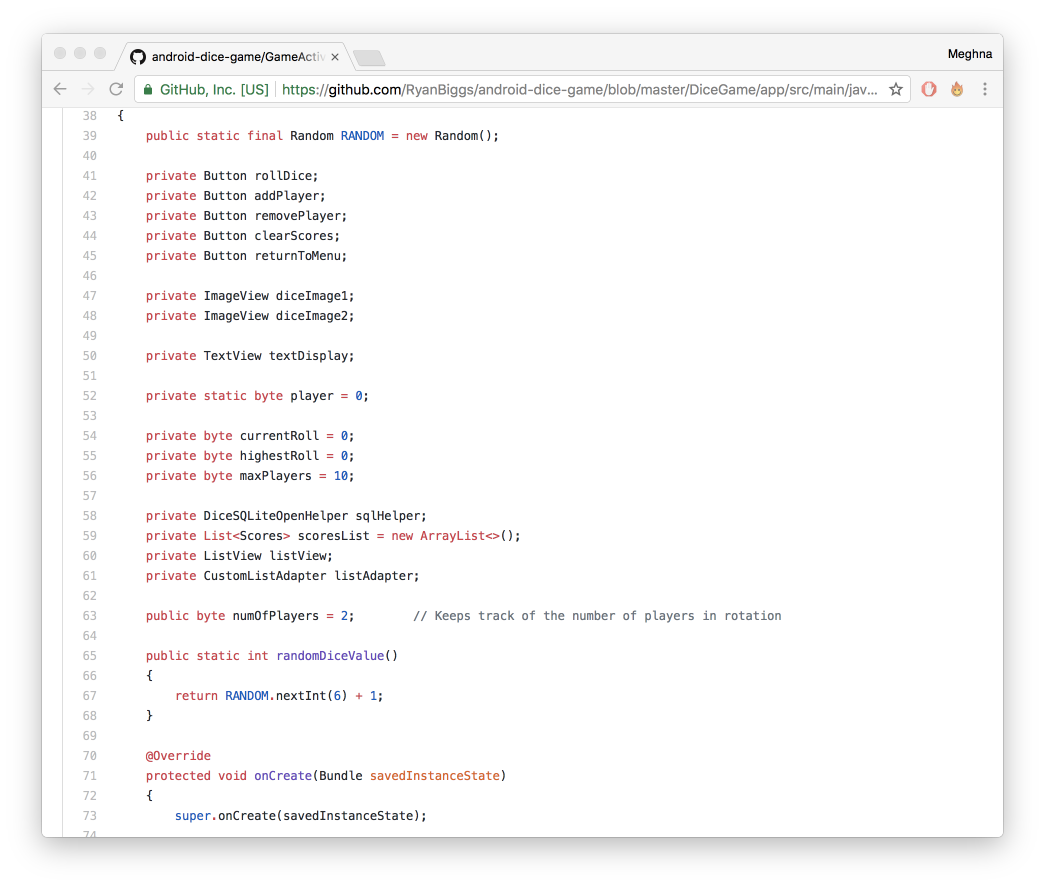
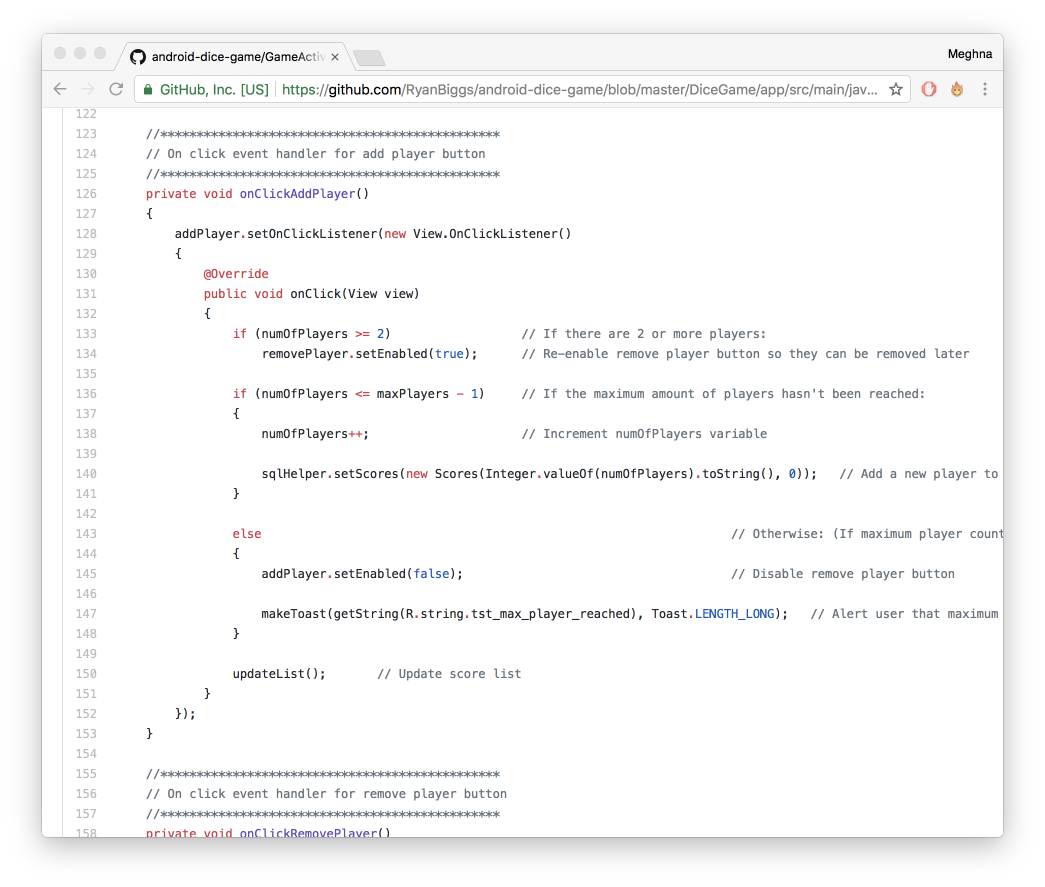
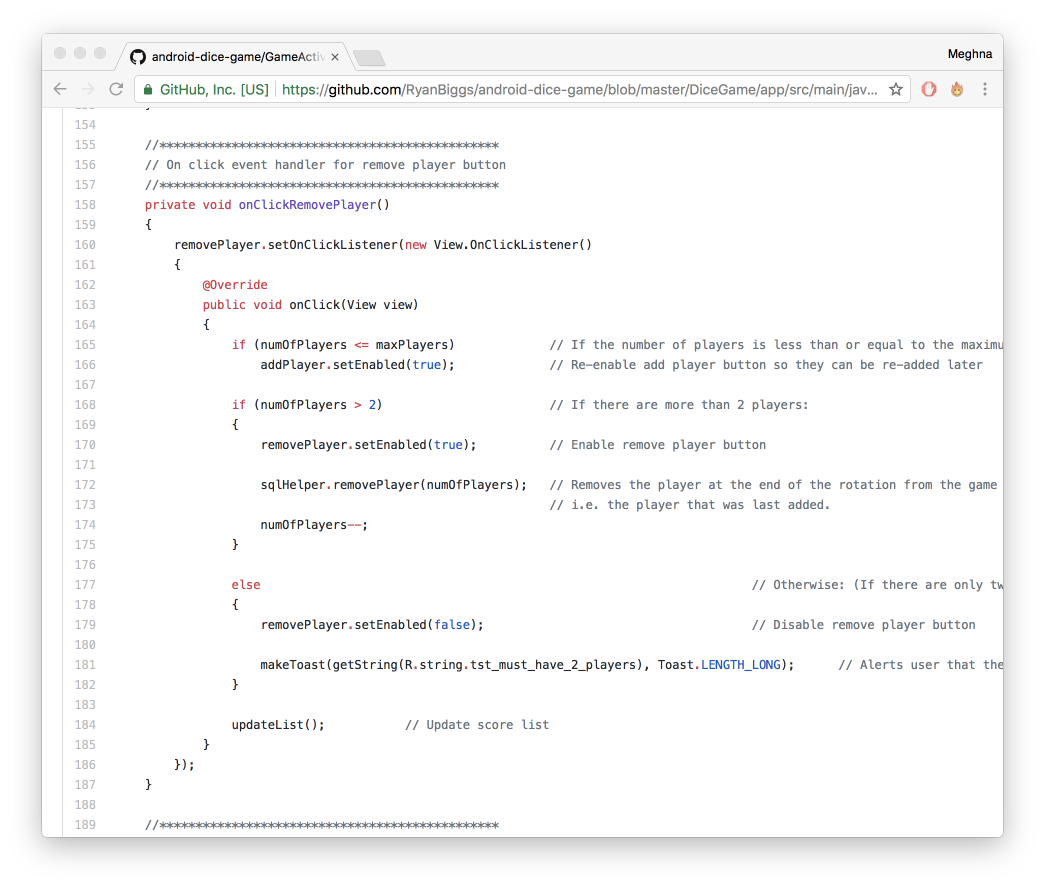
We used SQLite for the database. This is because it keeps track of the dice roll score for each player and stores them into the database so they can be accessed during the game.

(See screenshot below)

We came across a problem when making the score list update when one of the players won. We had trouble with the notifydatasetchanged(). So we tried to reuse the setup list method to update the list view however it wouldn’t work as the setup list method didn’t work for both. Instead we had to make two separate methods to handle the set up functionality and the update functionality. The update functionality is essentially a duplicate of the setup list we initially tried to use, but didn’t work. We found that having two methods, although it meant more code, worked better to update the score list when a player had won.

(See screenshot below)

When developing, we added some extra methods to make our dice game even better. They are listed below with screenshots of the code to show how we implemented them.

We added the ability to add and remove players with a 10 player limit compared to the original application which wanted a fixed number of two players:

we also added localisation for four different languages as well as English to the application. This means if the devices language is German, Spanish, French or Japanese, then the whole application will be displayed in that language.

