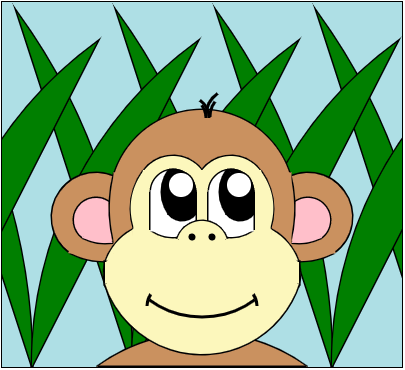
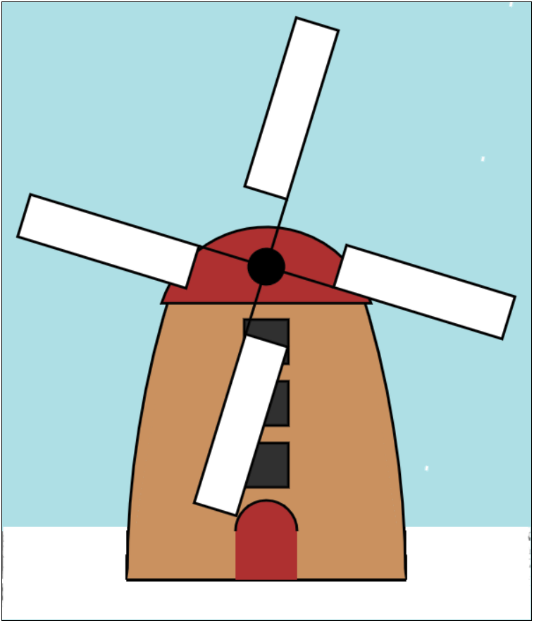
**ITEM 1 – STATIC MONKEY AND ANIMATED WINDMILL**

**Rationale**

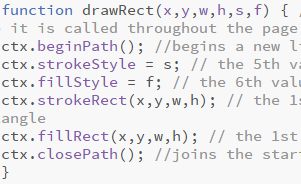
For part 1 I decided to draw a monkey as my static canvas piece because I believed it would show off my ability to draw all the different shapes and paths, these being lines, rectangles, irregular shapes and especially circles (or arcs).

For part 2 I looked online for some animation examples that I could follow and try to understand. I quickly realised most examples I was looking at, like one of planets rotating around the solar system, were very complex and would have been too challenging. I then looked for more simplistic examples and came across a rotating a rectangle which I was able to understand. Straight away I came up with the idea of drawing a windmill and identified its difficulty as suitable in the timeframe we had. I looked at the code and took the relevant parts, after hours of working I finally managed to get it working.

I knew both of these ideas would be a challenge but I felt it was achievable. The complexity allowed me to create functions and variables for each different type of path or shape, and did the same with my animated example along with setting repetitions and changing values.

**Pseudo code examples**

These are a list of simple steps to make the more complex parts of my code easier to understand for humans.

**Static - Rectangles**

Set order of letter values

Begin path

Set stroke style letter value

Set fill style letter value

Stroke rectangle letter values

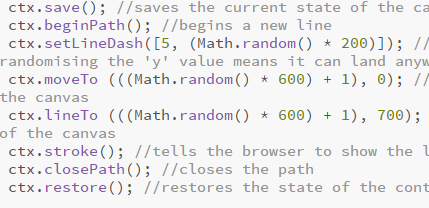
Fill rectangle letter values

Close path

Input rectangle z, y, w, h, s and f numerical/colour values

**Animation – Snow**

The line becomes dashed with ‘ctx.setLineDash’ and the ‘x’ coordinates of the start and end of the line are set to ‘Math.random’ so the line can appear anywhere within the canvas.

Save state of canvas

Begin path

Set line dash and its values

Set ‘x’ line values to random

Set ‘x’ line values to random

Give the line a stroke

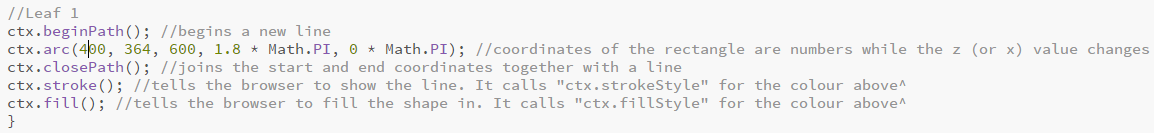
Close path

Restore state of context

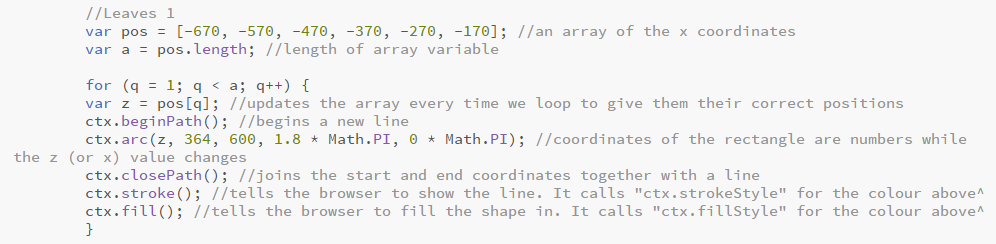
**Critique of techniques**

To keep the procedure as simple as possible, I used a technique whereby I would create each path or shape as its own entity at first. Then once I had completed all of these paths/shapes I would make a function or variable out of them after. I felt this was the best approach to take because it kept the process as simple as possible at first. I could just focus on getting my head around the drawing and understanding of how the x, y, width, height, and arc values worked in relation to the canvas. I found this progressive learning very beneficial as I now find it easy to draw shapes/paths and can get quite close to the correct position on the canvas without refreshing the page until I have completed it with a ‘closePath();’.

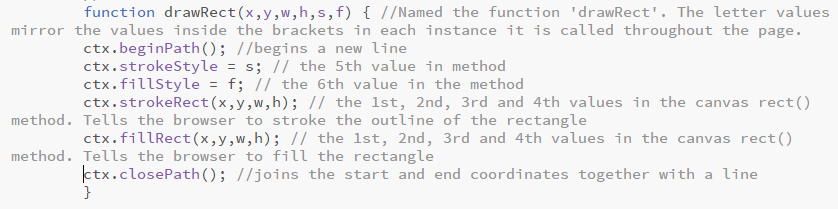
This is a coded example of how one of the background leaves looked at first (bearing in mind this was repeated 10 times with different x coordinates):

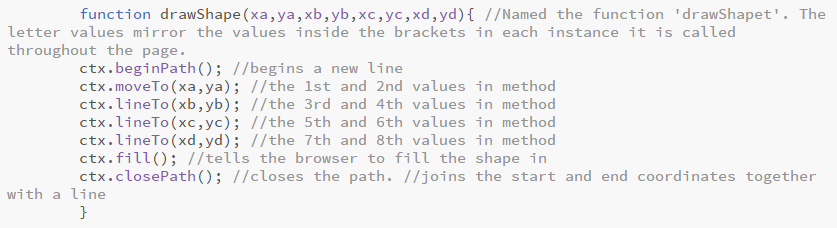


And this is how all 10 looked after:



This is clearly a much more concise and better thought out way of displaying the exact same visual. I took a similar approach with ‘ctx.rect’ and ‘ctx.beginPath’, but made a function out of them:

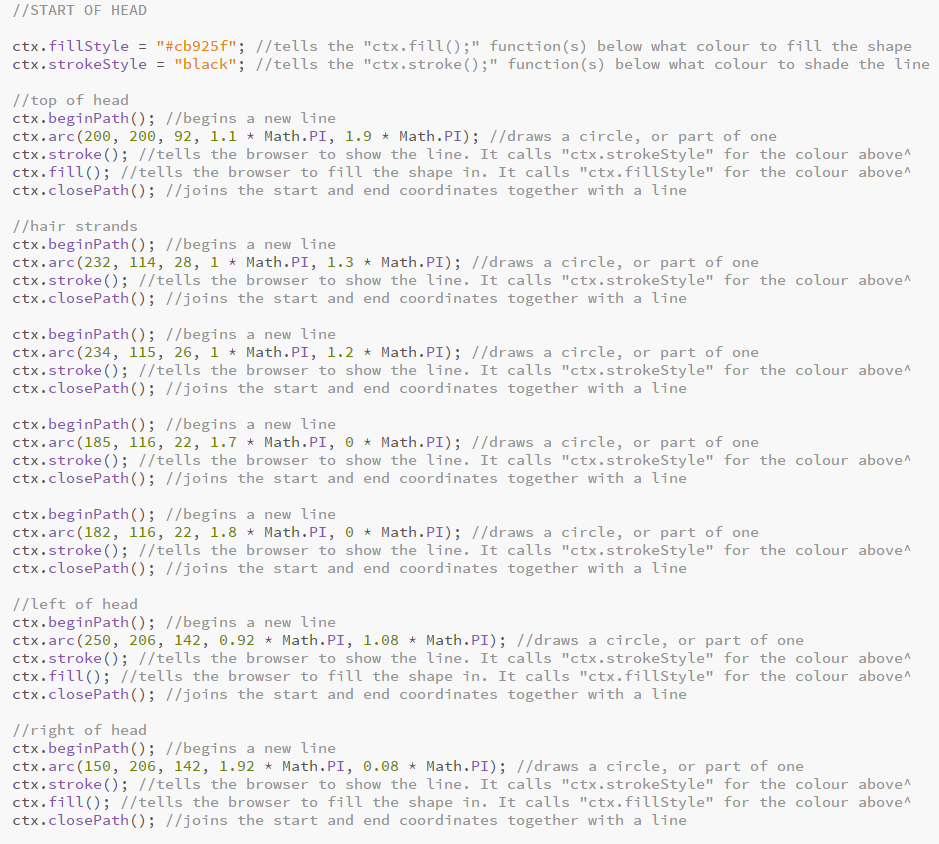




Each of these three examples drastically reduced the amount of code I had, and overall makes it much easier for anyone to locate a specific part of the canvas drawing.

\_\_

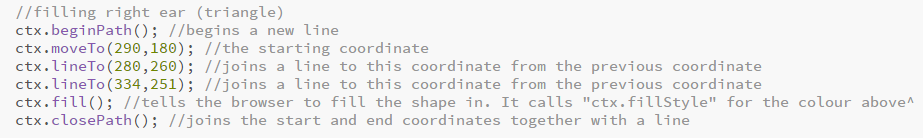
In regards to the ‘strokeStyle()’ and ‘fillStyle()’ canvas elements, I realised they did not need to be included with each path as they still work as long the path isn’t being affected by another one in between:



In this example, I would have had twelve more lines of code, which would have been unnecessary. Again, it drastically reduced the amount of code I originally had, and makes it much easier for anyone to locate a specific part of the image.

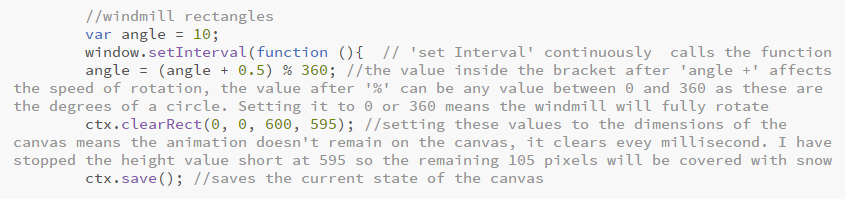
\_\_

When it came to the later stages of drawing each canvas, I realised the importance of making clear and simple code comments.



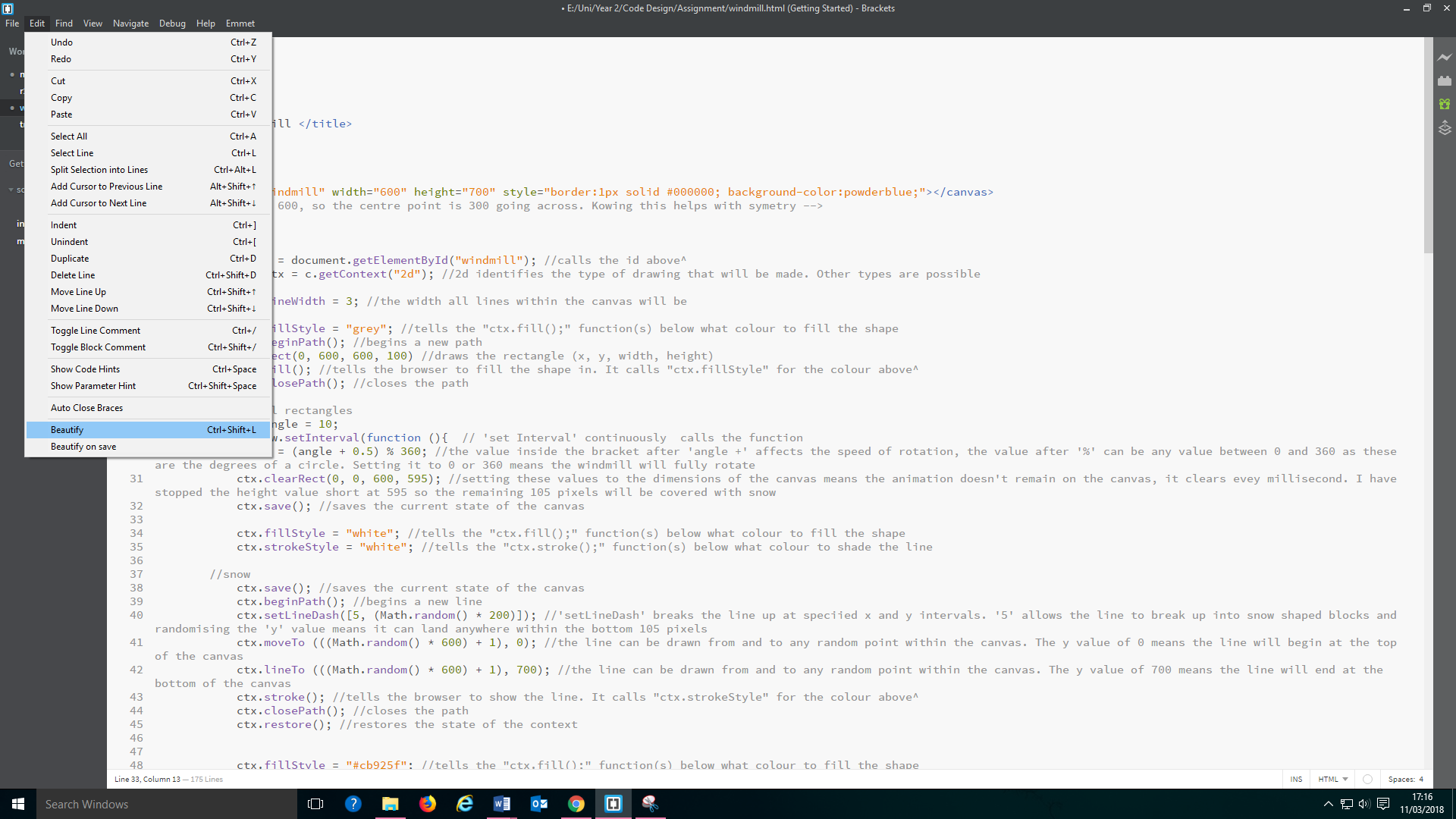
Whereas beforehand I might have called this canvas element ‘Right ear 3’ or something else unhelpful, now I have called it ‘filling right ear (triangle)’ which tells me exactly what and where the path is affecting in as little information as possible. This not only helps me locate that part of the canvas, it also helps anyone else who is viewing my code who otherwise would find difficult to know where to start looking. Utilising good practice like this is beneficial when working in groups and sharing work with people, as I found out in the paired part of this assignment.

I took a similar approach to my animated windmill but was even more careful about giving my code the appropriate explanations as code comments, as I knew it was much more complicated to understand than the static example.



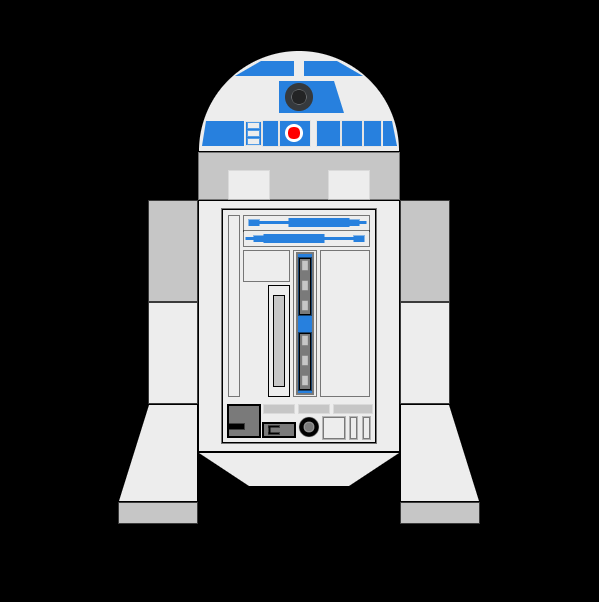
I didn’t want to write lines of comments but I found it difficult to keep it all on one line given the amount of writing I felt was necessary in order to explain what was going on properly.

\_\_



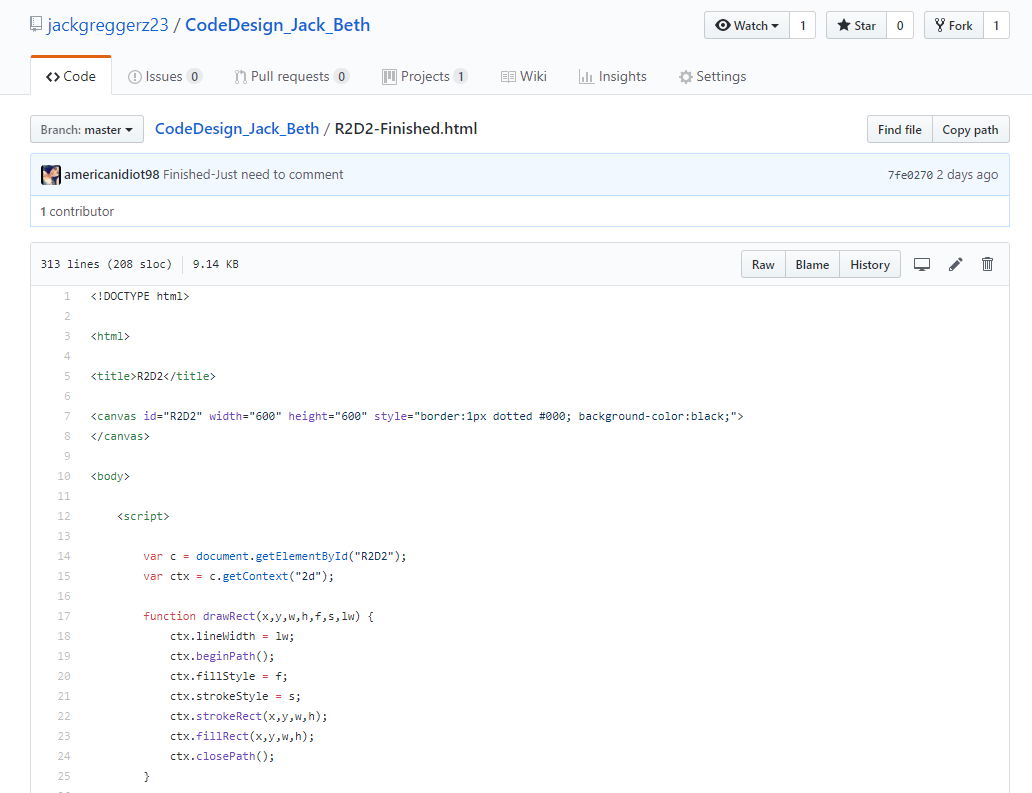
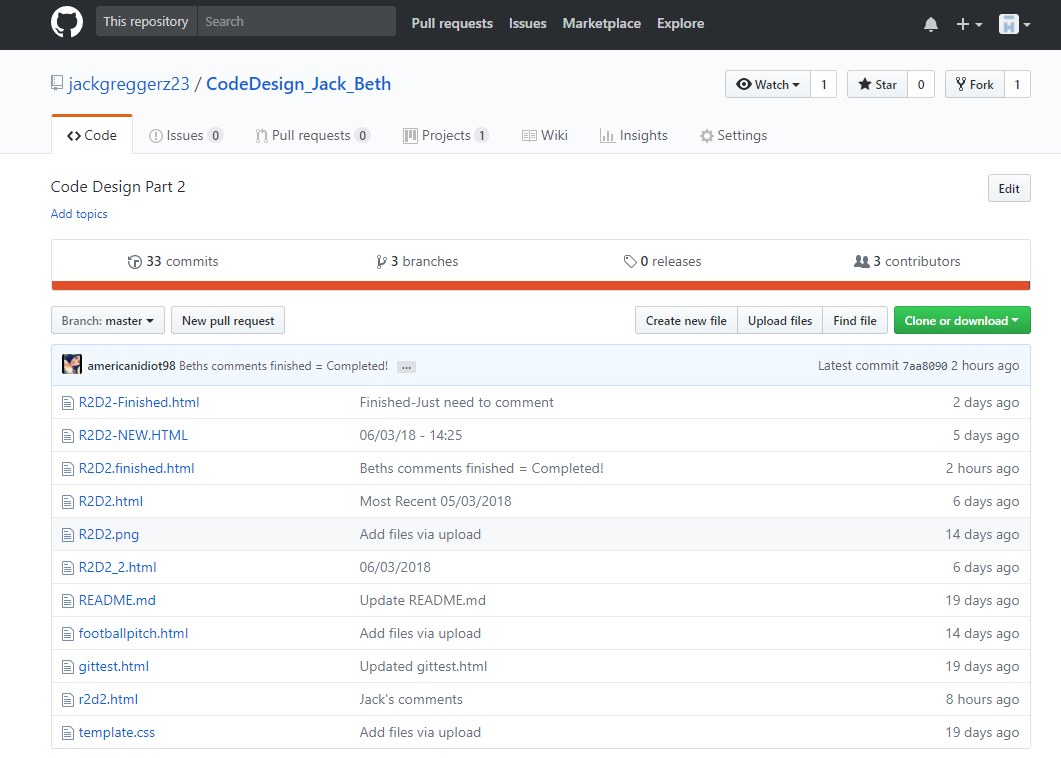
I used the beautifier tool in brackets to keep my code neat and easy to identify the elements that are affected by other elements and the ones that are not (parent-child relationships).

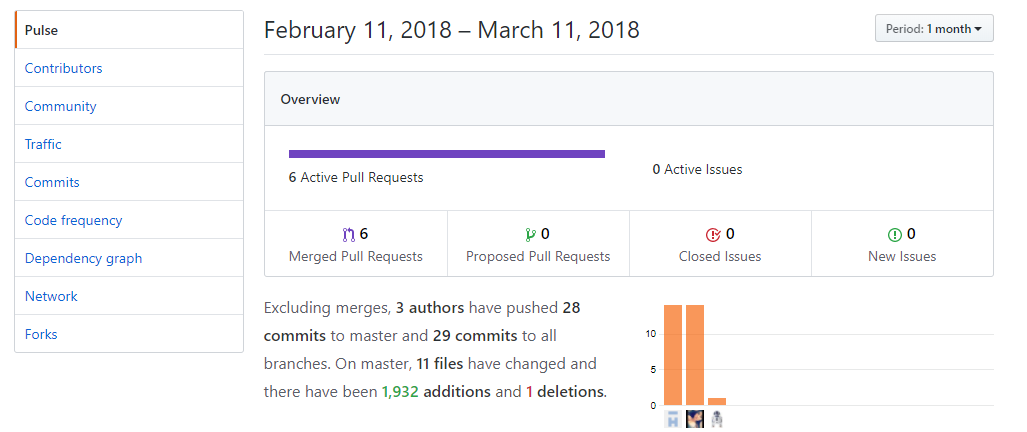
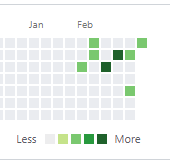
It definitely helped me keep my code neat and saved me a lot of time because instead of having to manually change each line, all I had to do was press the button whenever I had written a new block of code. It did the parent-child relationship for me so eradicated the possibility of human error.

**ITEM 2 – R2D2**

**Code management tool – Github**

<https://github.com/jackgreggerz23/CodeDesign_Jack_Beth>

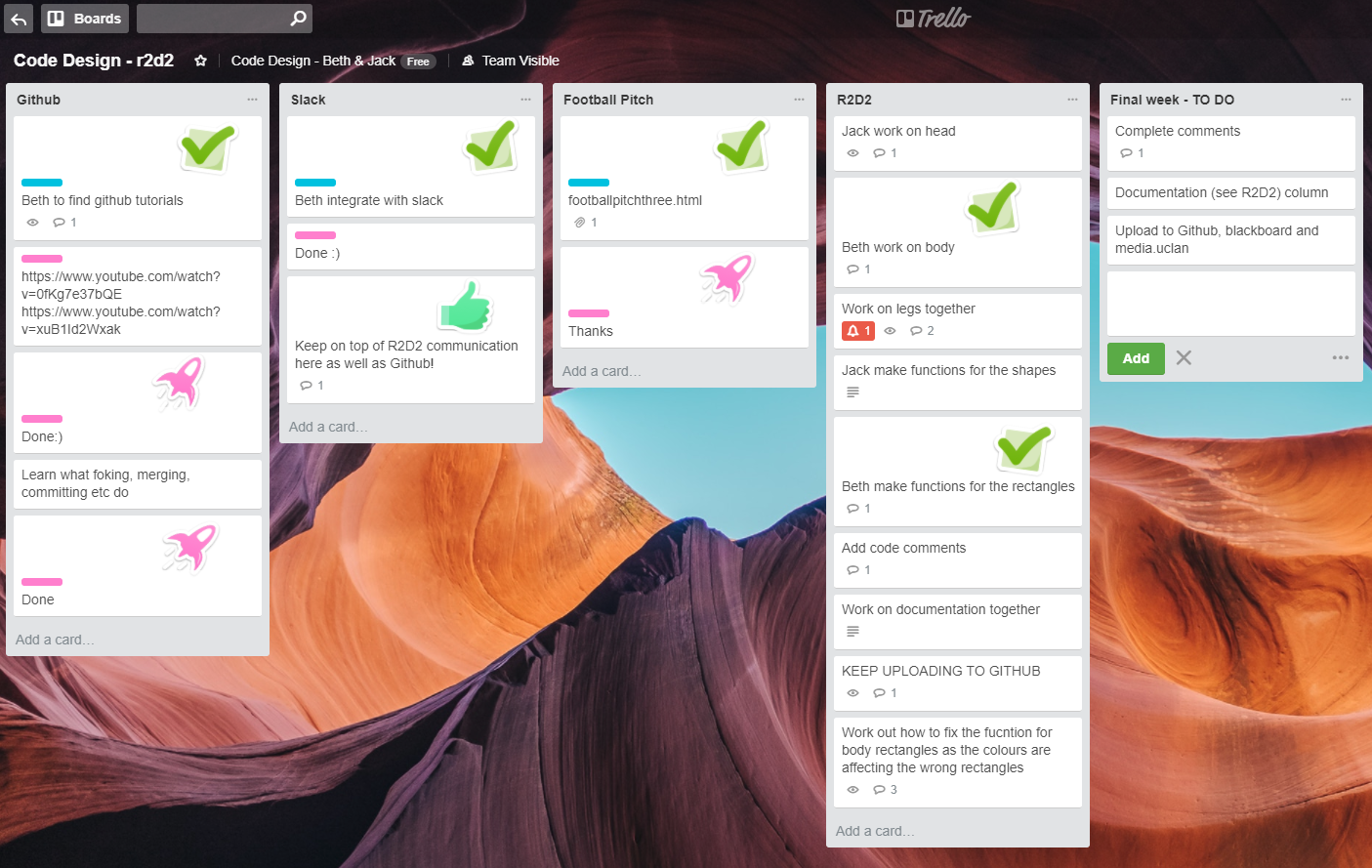


This is the evidence of my use of Github as the code management tool. Me and my teammate Beth uploaded our work here each time we had completed a section of the work. In order to version control, we committed, merged, pulled and forked eachother’s work. This eliminated the chances of work being lost and reduced the amount of time it takes to combine eachothers work together.

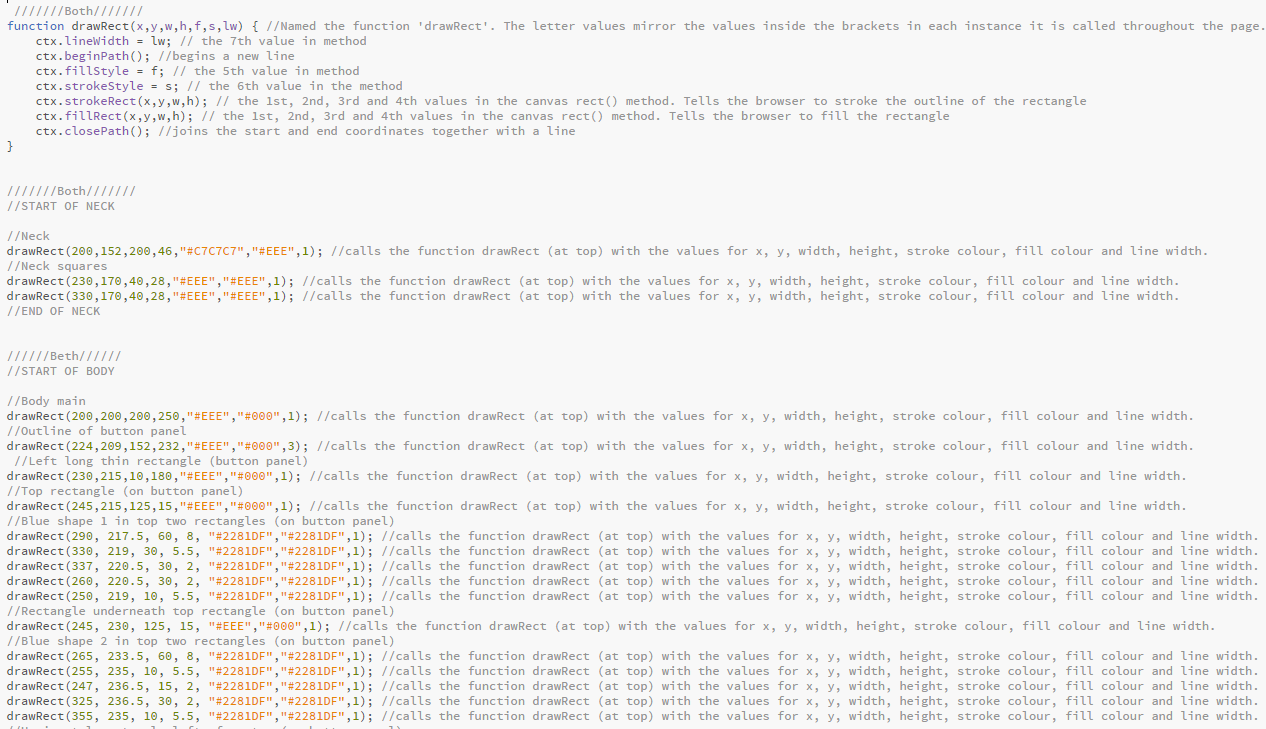
**Communication tool – Trello**

https://trello.com/b/Kxet66sN/code-design-r2d2



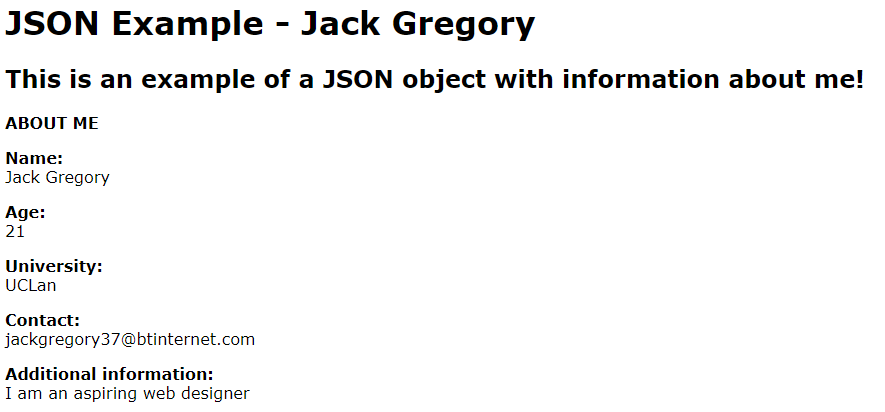
We used Trello as our communication tool throughout the process. We updated it when necessary with tasks to complete and responses when they were completed. It worked well as it gave us some structure to the process.

**Code Structure**

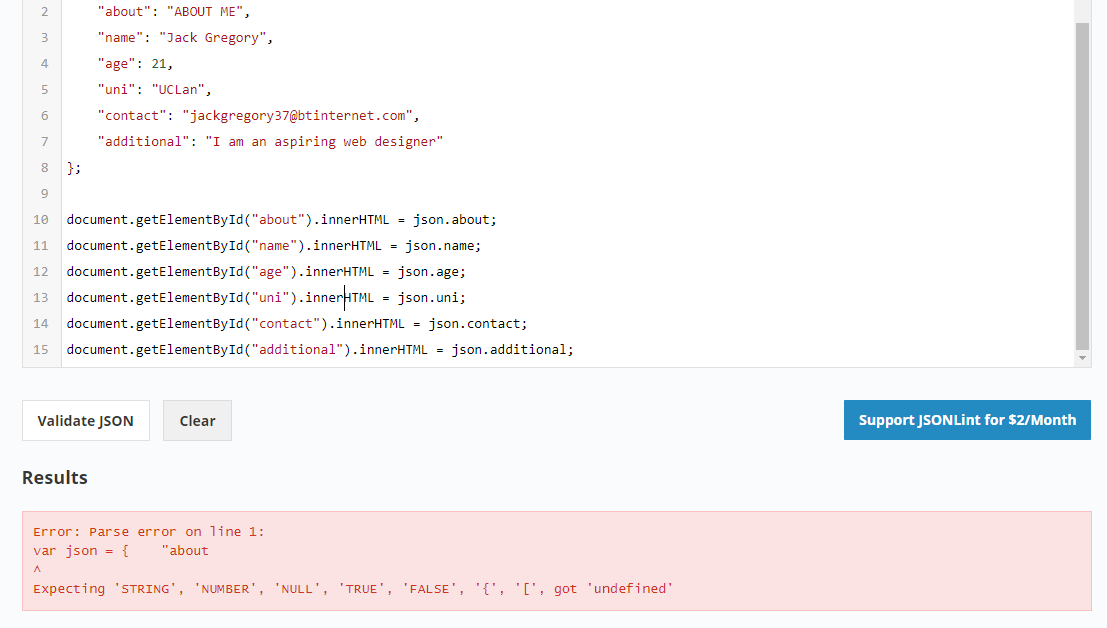


In order to keep a mutual understanding of the work we agreed on a structure to the code prior to starting. We agreed to give each section of code a short, but understandable name, for example, ‘Arse2D2’. We put these comments in block capitals, as we knew we would want to find these comments before we looked for the more specific comments in the section in question. We called each section we were the main contributors of by our name or both if we felt we were equal contributors, but we both contributed to all sections in some way.

**ITEM 3 – JSON**



**Linting**

I used <https://jsonlint.com/> to demonstrate quality checking. It identifies the problems within my page with validation and gives some explanation on how to fix these problems.