Interactive Media Teamwork

Managing Projects with GitHub

Thomas Abela

2HNDi

Task 1 – P1.1 – Identify Client Requirements

This project follows the development of my Unity computer game created for the CIDP assignment. From the beginning I already had an idea of what kind of genre of game I wanted to make, and so the functionalities I determined for the game were based on that decision. I wanted the game to be a sort of 3D platformer where you control a character that can explore a virtual world. These functionalities must reflect that genre. The following requirements were determined. The game must have -

1. A well animated character with precise movement controls.
2. A relatively large, open area of terrain for the player to explore containing a mixture of models built and textued in Maya.
3. A time-based objective. The player must complete a task before a certain amount of time expires.
4. Different collectibles spread throughout each level.
5. A smooth third person camera which allows the player to zoom in/out of the character and rotate the camera around the character.
6. A Health Bar with various collectibles affecting its value (Positive or negative).
7. While the game is intended to be played with a keyboard and mouse, an alternative set of controls should be implemented for mobile. For this reason, the option of on screen UI should be included to accommodate touch controls.
8. The main menu should reflect what kind of game this is. Therefore it should not be a static image but be based in a 3D environment with animated elements.
9. Level music is an important element to 3D platformers of the past such as Crash Bandicoot or Spyro the Dragon, which this game takes inspiration from. The game must have upbeat level music and sound effects to accommodate this requirement.
10. The game must have some convey an educational message, which has been determined to be Recycling awareness.

Task 2 – P1.2 – Define and Analyze target group to identify user needs.

In order to familiarize ourselves with GitHub, I will define the following terms in relation to GitHub tools.

**Repository**

A place where your code is stored. Each project you create must be contained within a repository. To create one, go to your GitHub account and go to the repositories tab and click New. Repositories can be public or private, however free accounts are only allowed to create public repositories, meaning anyone is able to find and access the code. So make sure your projects do not contain any sensitive information.

**Commit**

Committing is the act of taking all the changes you have made to a project and then recording them. This can be done as many times as you like, after which you can then push those changes onto the GitHub server, where you project’s repository is stored. One of the benefits of tracking commits is that it allows you to roll back to a previous version of the project if necessary.

**Issue**

Issues can be questions, tasks or suggestions made by individuals relating to your repositories. Remember that if your repository is public then anyone can see your project, and they may have queries regarding certain areas of your project. Whenever an issue is created, it opens its own discussion forum where the repository collaborators are able to discuss the issue with the person/s who submitted it.

**Sync**

Sync is the act of making sure that your local branch matches your remote branch of the project. If your local branch contains commits that your remote branch has not incorporated, then it will push those changes onto the remote branch. Likewise, if the remote branch has additions that the local branch does not, it will pull those changes onto your local branch. Sync basically ensures that your local and remote branches mirror each other.

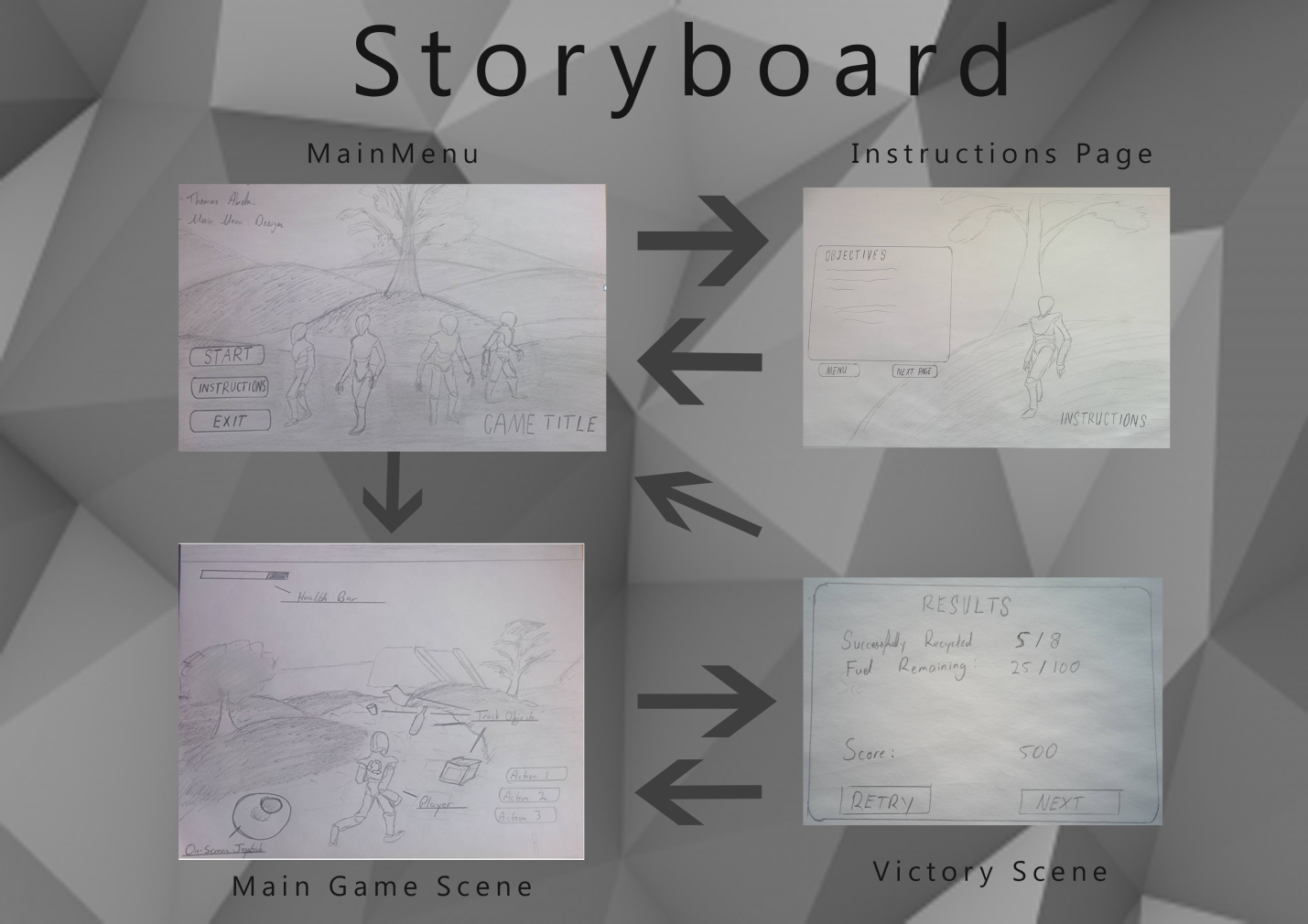
**Add**

Add is an option which allows users to add more buttons to a repository. Once this has been done, the user can commit the data and any changes to the remote branch via Sync.

**Pull Request**

Users can make changes to a project and then submit them to the repository collaborators as a proposed change to the repository. This is known as a pull request. The user who makes a change to the project can open a pull request, and the repository collaborators can then open a discussion forum to review the change. The collaborators can either accept or reject the change to the main repository.

Task 3 – P1.3 – Clarify creative intentions through recorded communication with client.



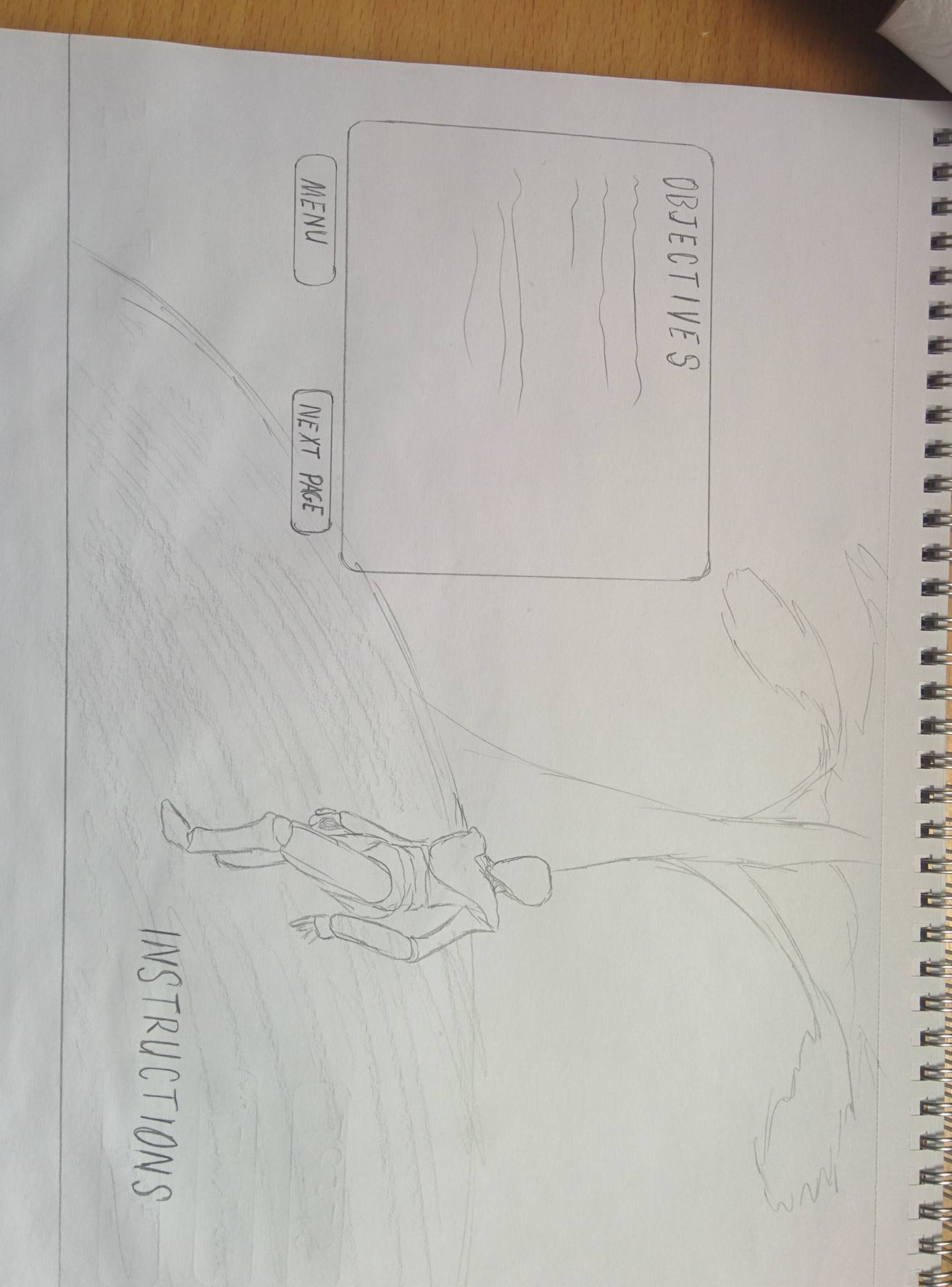
**Link to relevant commit**  
<https://github.com/Tabela91/IMT-Assignment/commit/de137b6ac687192af36e91a04b0aaf08686b5947>

**Main Menu**



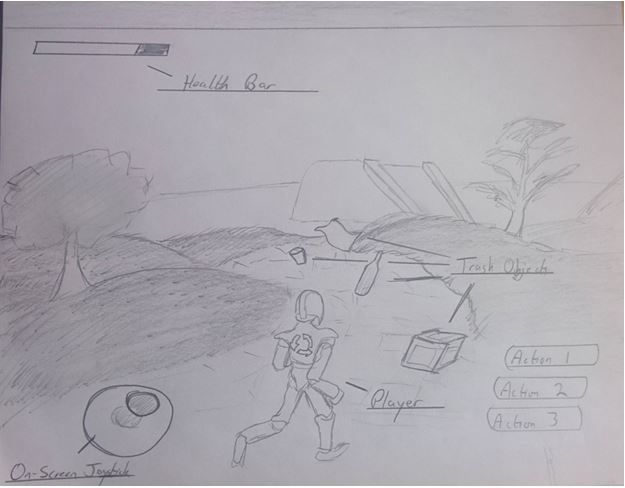
The main title screen will have animated models and offer three simple options, to start the game, go to the instructions page or quit the application.

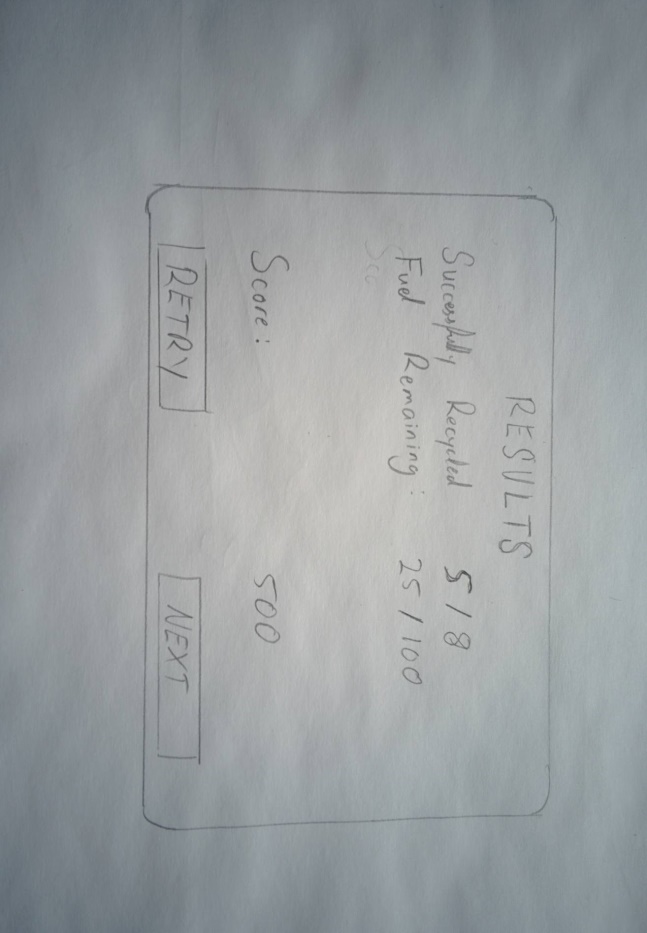
**Instructions Page**



Simple page showing game controls and objectives.

**Main Game Scene**



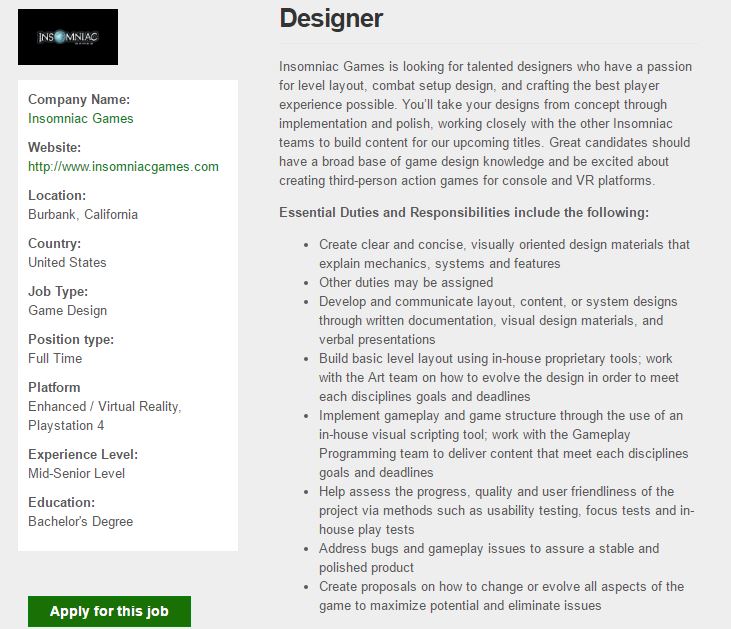
****Player controls character through the level collecting pickups and running towards the objective.

**Results Page**

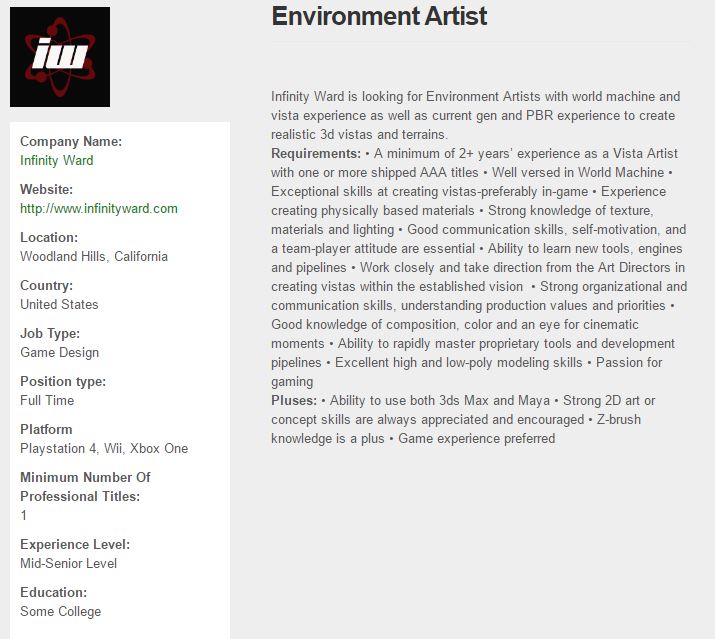
Simple canvas displaying the number of collectibles found and the score attained by the player. Player can either retry to level or move to the next level or main menu.

Task 4 – P2.1 – Identify and apply own area of expertise.

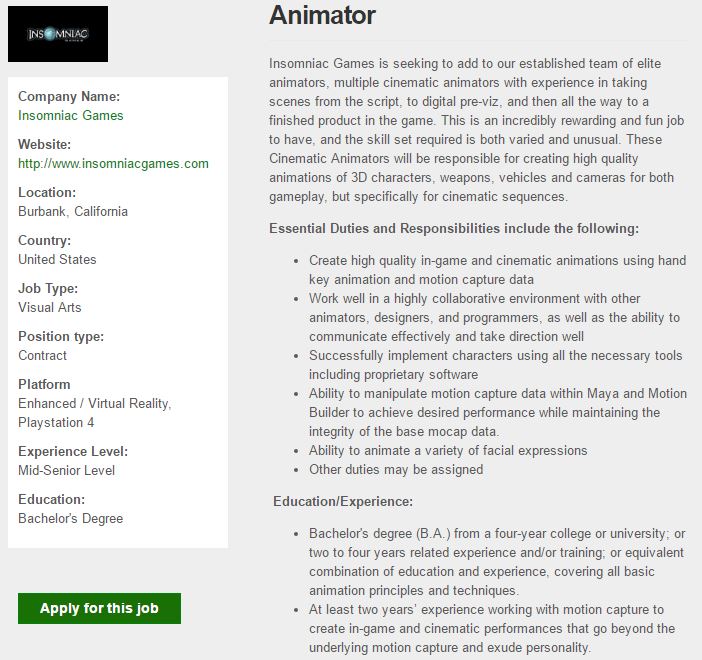
The following list outlines some job offers relevant to game development in the gaming industry. These are roles which I believe would be necessary in order to implement my game project in a professional environment.



Designers would work to build the core game mechanics, using assets from other departments such as models and animated rigs, and bringing them together to create the core game functionalities. In the case of my game project, they would be using a game engine such as Unity to bring together assets, write scripts and fix bugs to ensure that the gaming experience is as smooth and enjoyable as possible.



Environment Artists would be tasked with creating the world of the game. The main aspect of this would be building the terrain, but it is much more than that. They would have to work with lighting, textures, model building, materials and much more in order to create a seamless, living, virtual world.



Animators would be responsible for creating realistic movement for the player characters as well as any non-playable-characters (npc). They would also be responsible for any other animations such as floating pick-ups. They could use software such as Maya in order to take a 3D model and animate it realistically. In most cases, the models themselves and their movement rigs will have been created by teams dedicated to such things, the animator would have to communicate with the model creator and rigger to have a good understanding of how the character/vehicle/model is intended to move.

Personally I feel that my most suited role would be as an Environment Designer, the reason or this being that I already have a good foundation of 3D modelling, UV mapping and texture design specifically in Maya and 3DS Max. I also have some experience using ZBrush, all of which are requested requirements in the job offer listed above.

Task 5 – P2.2 – Clarify own role within team-driven development schedule.

Having chosen the role of an Environment Designer, I will list below how I feel my role would contribute to the following phases of a game’s development.

**Idea Generation –** In this phase I would be responsible for proposing ideas for the look and feel of the environment in relation to the game genre and the objectives. As this is mostly a collaborative stage where ideas are contributed by a large team, I could also consider and think of strategies which would blend together proposed ideas into a single environment concept which would work best for the overall game experience.

**Storyboarding and Game Design –** At this stage, I would take part in discussions with the environment team on creating specific sections of the environment which would be engaging for the player. As an example, if we decided to create an environment in a snowy terrain, we would try to come up with ideas for specific sections in the level where for example the player would have to move slowly/quietly in order to avoid triggering an avalanche. The idea is to design concepts for set pieces or events within the level which would be engaging and entertaining for the player.

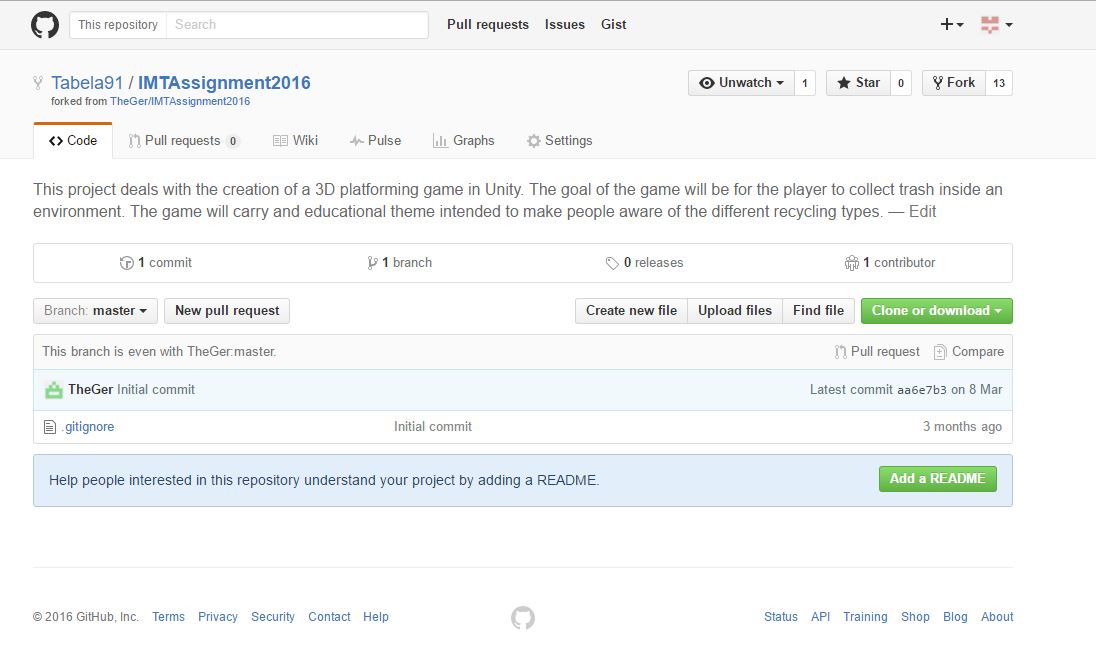
**Creation of the Game Design Document –** At this point different environments and props relevant to the level design will have already started production. Each model, prop or terrain would have to have its development documented well. The main reason for this being that game assets will be exchanged with different departments continuously, and it would be important for each hand over to be done efficiently so different departments would use those assets appropriately. Environment Designers would for example, exchange assets with 3D Artists who would be designing props for the environment or characters related to that environment. They would also be responsible for handing over terrain assets to Game Designers to place within the game engine where the actual game mechanics will be implemented.

**Implementation of Functionalities –** At this point the focus would be on the direct interaction between the player and the environment. If events within the level are triggered by the player, it would be important for the game designers and environment designers to collaborate and ensure that those events occur as intended. The environment designer would explain to the programmers how specific sections of the environment would react or behave to a change triggered by the player.

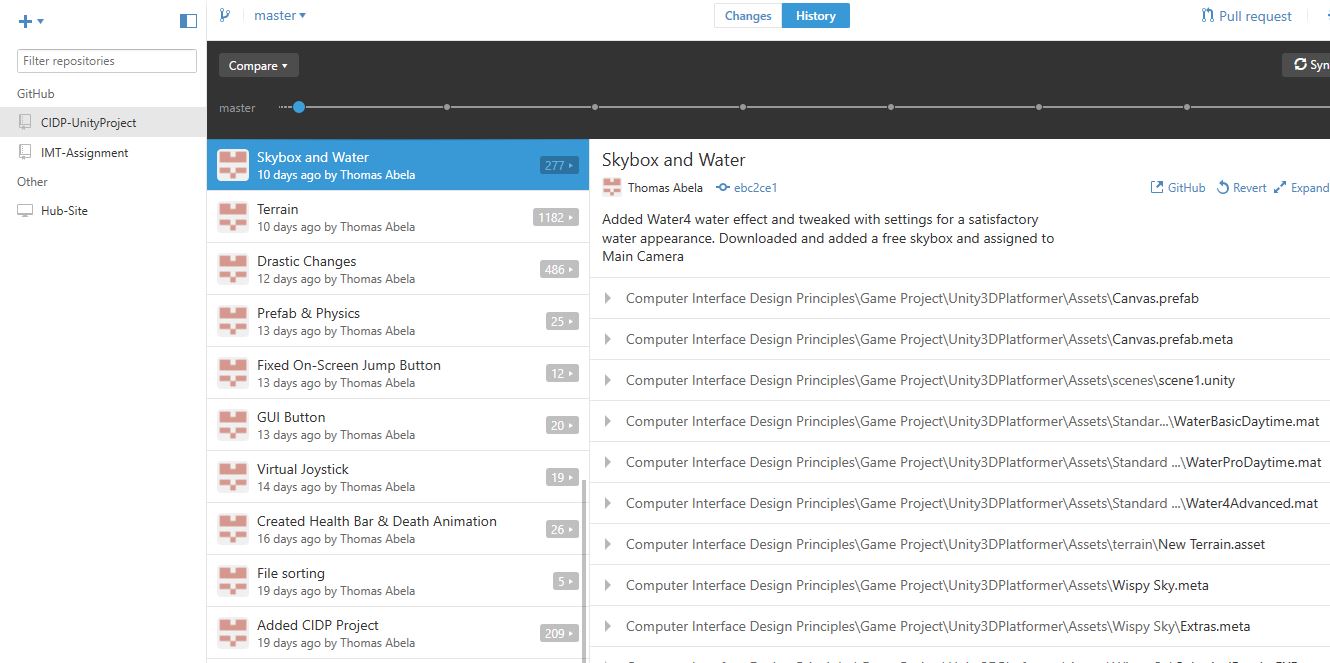
**Deployment and Support of the Game –** At this point, all functionalities should be implemented, but before deployment, it would be important to ensure that the entire player experience blends those functionalities well, and that all functionalities work together well. Events should not conflict with each other. Environment Designers would be responsible for ensuring that the Environment behavior in each level works as intended.

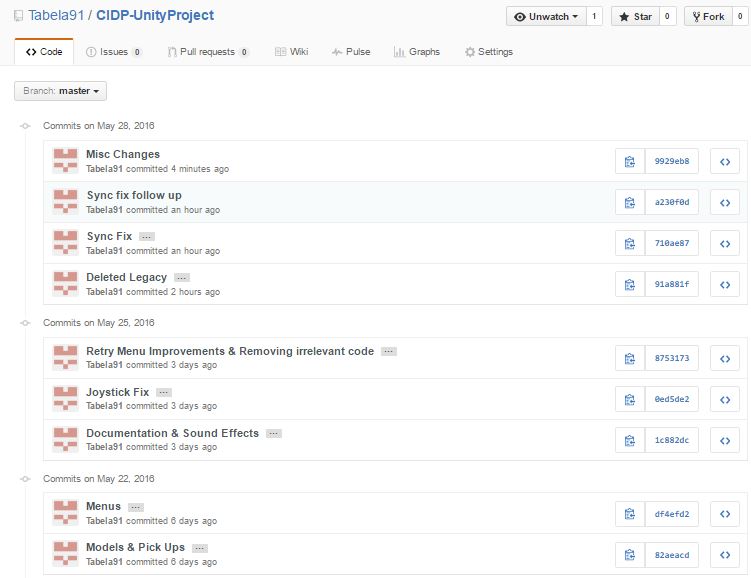
Task 6 – P3.1 – Produce preliminary components for an initial prototype.

The following screenshot shows the newly forked project in the assignment repository.



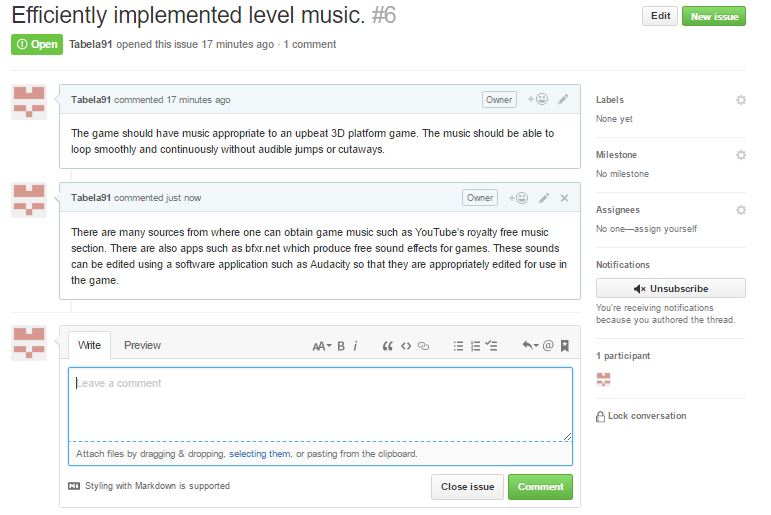
Below you can see the local game project repository open in the Git Client and the remote online repository respectively, with several commits that have been implemented throughout the project’s development.

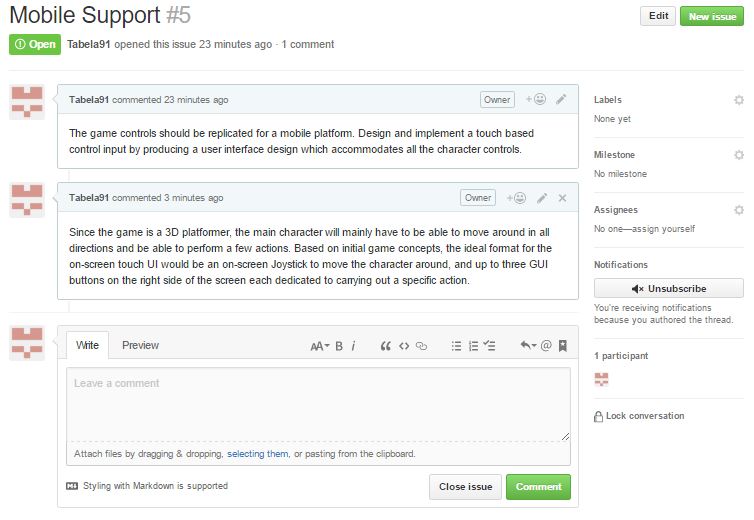


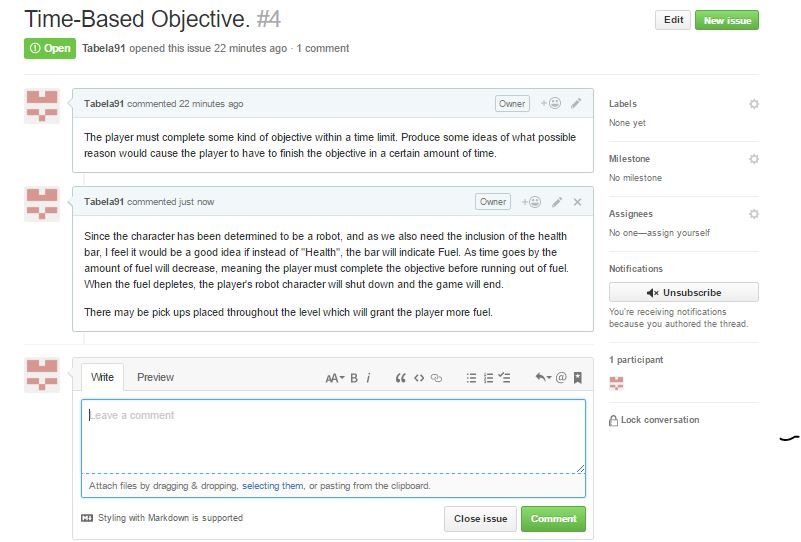


Task 7 – P3.2 – Evaluate and confirm prototype in relation to constraints.

The following screenshots show several issues that were made on the remote repository for the game project. These issues reflect functionalities which must be implemented into the game based on client requirements.

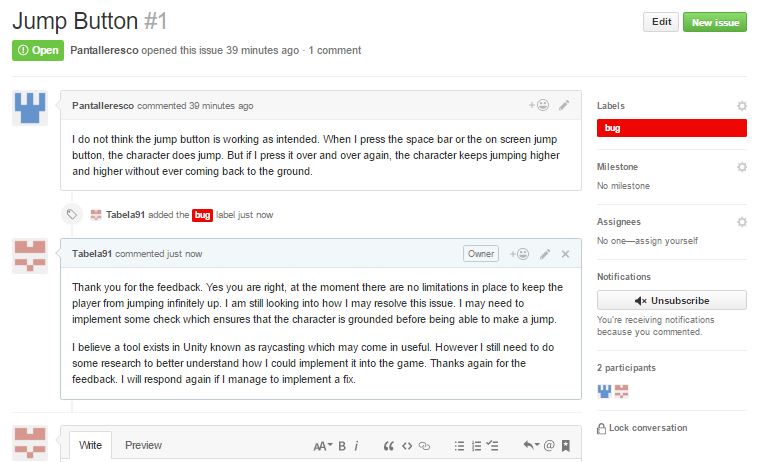


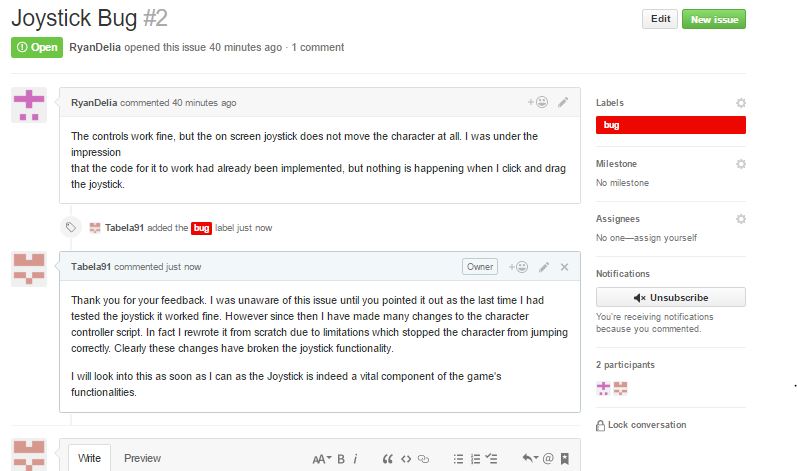


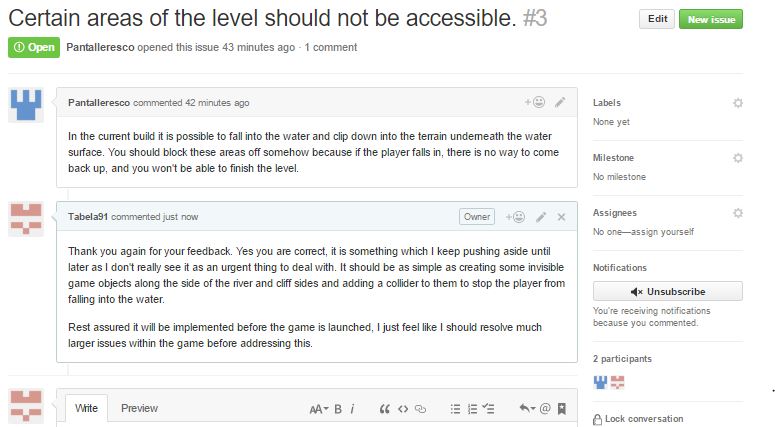


Task 8 – P3.3 – Reflect and record on feedback from prototype phases.

The game’s prototype was given to two other students who were able to test and provide feedback for the game. Between them they produced three issues with the game that they felt needed urgent attention. The following screenshots show the outcome of these issues.





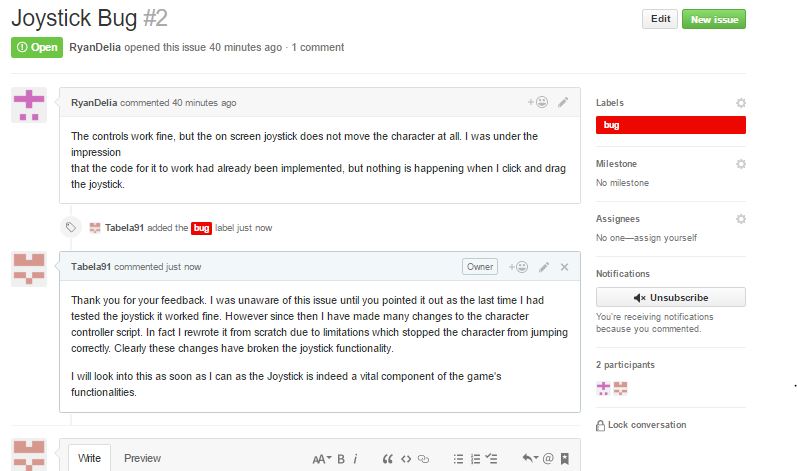


Task 9 – P4.1 – Develop a fully working interactive media product that meets clients’ needs.

This task will document how a bug was detected, investigated and resolved over a period of time, particularly what steps were taken to fix the issue.

In this case I will be investigating a bug with the on-screen joystick.

Here is the Issue related to the bug (shown in the previous task).



As explained before, the joystick is a vital part of the UI as adapting the controls for mobile touch input is an important client requirement. I had already implemented a working version of the joystick, but after several drastic modifications were made to the character controller script, the joystick seemed to stop functioning completely.

To better understand this issue, I will explain how I implemented the virtual joystick in the first place.

Here is a screenshot of the main game level. As you can see, the UI has been designed for a simple mobile touch input. On the bottom right you can see the Jump button, and on the bottom left is the virtual joystick.



As the game is being built on PC, they currently work by using the mouse cursor. You click the Jump button to jump, and you click and drag the knob on the joystick to move the character around. However when played on PC, you can simply move around with the WASD or Arrow keys and jump using the Space bar instead.

Unfortunately at the moment, the joystick function isn’t working, and this is the bug I will attempt to resolve.

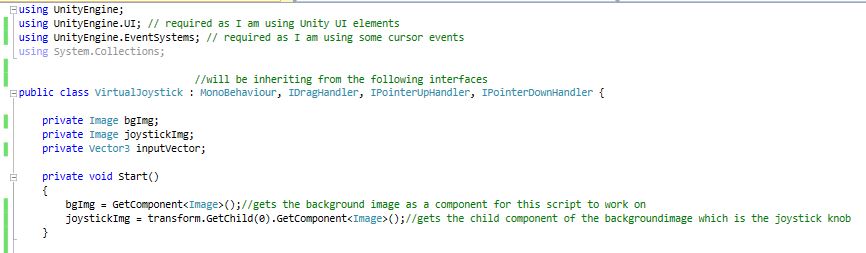
**Creating a Touch-Input Joystick**

**Creating the Joystick UI**

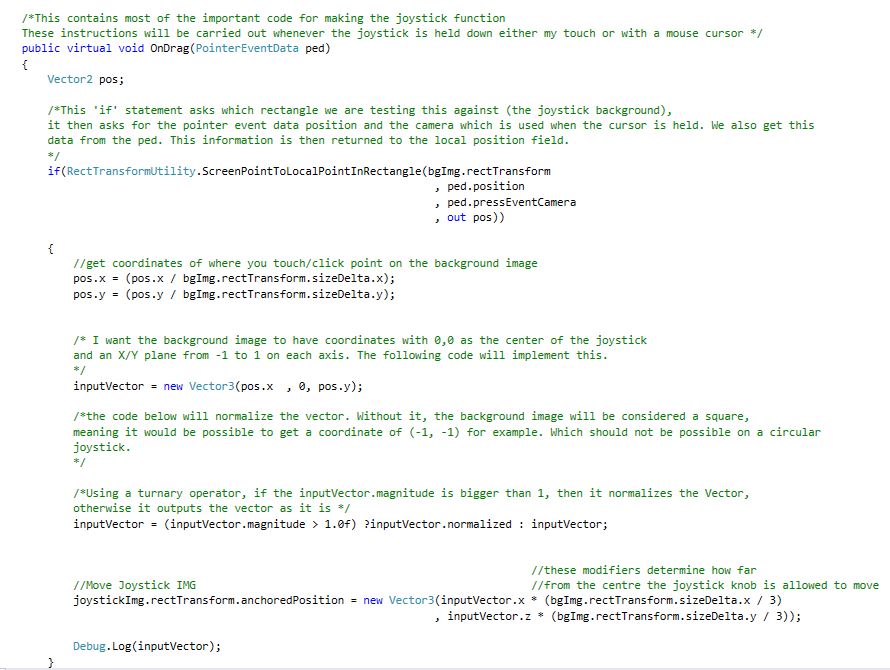
I created the joystick by adding an Image to the main canvas and anchored it to the bottom left of the screen with a width and height of 100x100. I used a Unity default sprite called “Knob”, which is simply a white circle. I gave the Knob a Green color to match the rest of the UI design.

I then added another UI Image as a child of the Green circle. This child image was again given the Knob sprite but made smaller to make it appear like a small joystick knob.

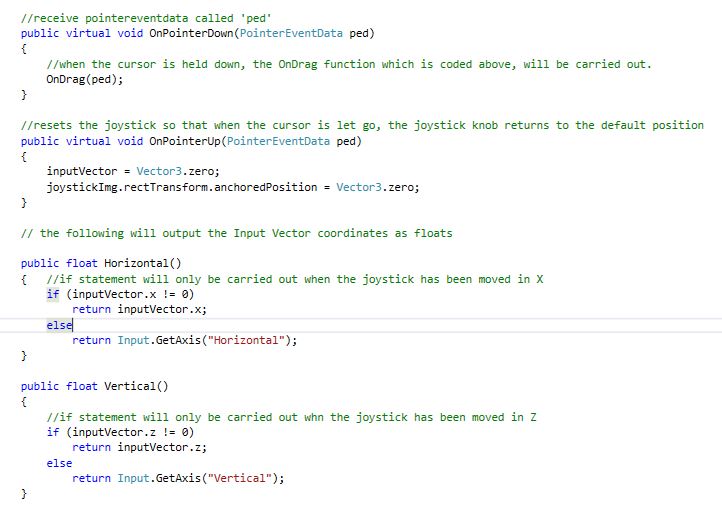
**Virtual Joystick Script**

I needed a C# script to control the joystick’s behavior. The code is explained through in line comments within the code below:

At this point, we have called the required interfaces and elements which will control all the events throughout the script. The joystick background image and knob image have been called as components.

This is the biggest chunk of the joystick code. The OnDrag function, controls the movement of the joystick knob and creates an inputVector which contains the coordinates of where the cursor is moved around the joystick background image.

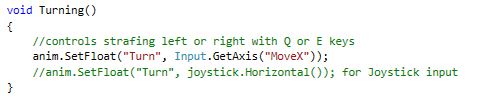
This final part of the script controls simple things such as executing the OnDrag function and resetting the joystick position when we stop dragging the knob.



The last two functions, Horizontal() and Vertical(), will return the X and Z coordinates of the input Vector. As they are set to public, these values can be called from the character movement script to determine which direction the character should move.

Here you can see several functions in the character controller script where we have called for the joystick.Horizontal() and joystick.Vertical() float values.

The character’s run animation is determined by two float values: MoveZ and MoveX. These values go from -1 to 1, and so all I have to do for joystick input to work, is assign the joystick.Horizontal() and joystick.Vertical() values to the floats which control the running animations.





Note that at this point I was unable to have both Touch and Keyboard input working together at the same time, so I would have the appropriate code un/commented depending on which input I wanted to work at the time.

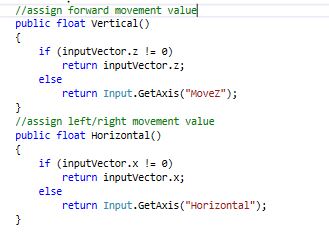
**Resolving the Issue**

After implementing the joystick, it had worked fine, however I eventually had to make some drastic changes to the character controller script. It seems that after I made these changes, the joystick stopped working.

After spending a long time evaluating and debugging the code, I realized that for some reason, the float values of joystick.Horizontal() and joystick.Vertical() were not being correctly assigned to the MoveX and MoveZ animation floats. They would always return a value of zero, even though the inputVector from the joystick script was retrieving the X and Z coordinates correctly.

I had also added some custom Project Input Settings, as a result of the new Movement script, called MoveX and MoveZ which controlled the values of the float animations with keyboard input and they were conflicting in the code with the joystick float values.

**Fixes – 1. Simultaneous Keyboard/Joystick controls.**

First of all, I found a way to keep both the Joystick and Keyboard controls active at the same time.

By adjusting the Vertical() and Horizontal() functions in the Virtual Joystick script, here we can see that the code is checking to see if the joystick has been moved, in which case it returns the input vector value.

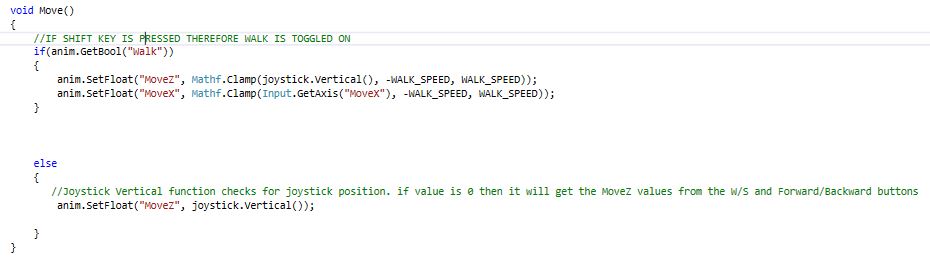
However if the joystick is not moved, the functions will instead assign the float a value based on the keyboard input. In other words, the game will always revert to keyboard controls unless it detects Joystick movement.

**Fixes – 2. Reassigning Input Settings**

The animator makes use of three primary float values which control character movement. These are:

* MoveZ – Controls forward and backward movement.
* MoveX – Controls strafing from left to right.
* Turn – Controls left and right rotations.

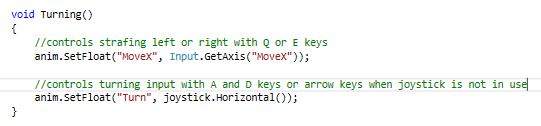
The previous version of the character controller did not cater for strafing. With the way the scripts and the blend trees are now set up, we can make sense of how the keyboard and joystick affect these three float values to move the character.

MoveZ is always affected by Input.GetAxis(“MoveZ”), which is the Input setting I set up for forward movement, controlled by the W/S, Up Arrow/Down Arrow keys.

As you can see, the Move() function in the Movement script, now gets all the Z movement from joystick.Vertical(), which now caters for both joystick and keyboard input.

Notice that MoveX, which controls strafing, is still set to Input.GetAxis(“MoveX”), as strafing is controlled by the Q and E keys and is not catered for by the virtual joystick.

As for the Turn() function, the code was simplified as show here.



Note that the previous code for strafing was to be triggered only when walking is toggled, which is why it is typed again here for regular running speed.  
As with the MoveZ float, I am relying on the joystick game object to control the turning animation, which will be controlled by the A/D or Left/Right Arrow keys when the joystick is not available.

In truth it is difficult to point out exactly what caused the joystick functionality to break, as the joystick script worked correctly. The Movement script however had seen drastic changes, so it was clear that this was the root of the problem. However through a lot of deduction and simplification of the code, I was able to ensure that the correct inputs were being taken and assigned to the three floats controlling the animations, resulting in a smooth and easy to use on-screen joystick.

Task 10 – P4.2 – Evaluate and record Interactive Media outcomes against constraints & requirements of brief.

From the start of the project I was confident that I would be able to implement all the functionalities I had planned for the game. Personally I set my goals at a high level of quality. I wanted the game to be beautiful to look at, I wanted the environments to be impressive, the animations as seamless as possible and to implement several models in the environment to make the game seem like it had a semi-professional quality about it.

That being said, I underestimated how much time would need to be invested to reach those goals. The development of the character controller script for example, took much longer than I expected and at one point had to be completely rewritten to accommodate the animation system I wanted to use. The third person camera was also something I wanted to work in a very specific way, similar to how a 3rd person camera works in MMOs, and since no included Unity scripts catered to this, I had to write my own camera script from scratch which took some time to get right. By the time I got these vital components to work the way I wanted them to, I was left with little time to focus on things such as:

* Fine-tuning character movement.
* Higher quality/quantity of 3D models made and imported from Maya.
* Larger range of player actions.
* Improved UV mapping of textured models.
* Hand refining textures in Photoshop.
* Optimizing graphics/textures for Mobile version.

Task 11 – M1.1 – Identify & Apply Strategies to find appropriate solutions.

There are many other kinds of Version Control Software like GitHub. The following is an evaluation and comparison of other Version Control Software which one might find as a suitable alternative to GitHub.

**Beanstalk**

Unlike GitHub, Beanstalk requires payment in order to be used. It supports Git and SVN Version control software and allows you to edit your project code within the browser, similar to GitHub. It sells its service in a variety of packages, the cheapest alternative allowing you up to 5 collaborative users and up to 10 Repositories.

**BitBucket**

One of GitHub’s disadvantages is that it requires payment in order to create private repositories, which isn’t ideal if your project involves handling sensitive information. BitBucket is an alternative solution which offers both public and private repositories for free, up to a certain number of users. It is a popular code hosting website for Mercurial and Git version control software. You can also use whichever Git Client your team is comfortable with in order to sync your projects with the remote repository.

**SourceForge**

This web-based source code repository allows developers to control and manage their software development much like GitHub, however it has been in use far longer than Git. It requires that each project has a unique name and lets users host static or dynamic pages on their servers. It supports PHP, Perl, Python and other scripting languages, and it also allows for uploads via SFTP clients.

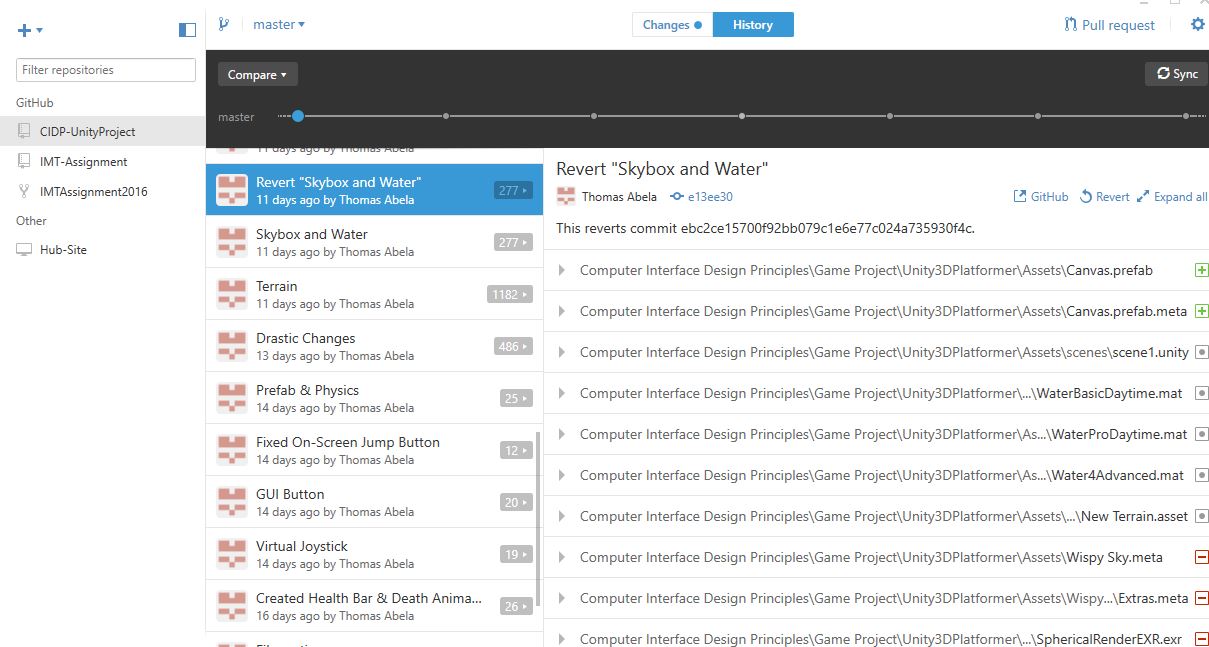
Task 12 – M2.1 – Select/Design and apply appropriate methods/techniques.

This task will explain the concept of rolling back a commit, and show case how such a thing can be performed.

As we have already discussed, GitHub allows users to track changes made to a project over the entire duration of the software’s development. Each change is documented as a commit, with a description of the changes made for each one. But the concept behind version control is that it allows users to keep multiple versions of the software while it was being built.

It’s entirely possible that at some point during development, users will realize that they may need to roll back their project to a previous version. This can be for several reasons, whether for reference or perhaps an undesirable change was made which can cause critical issues for the software. By tracking these commits over time, GitHub allows users to select any of these commits and revert the changes made.

Below is an example of a roll back that was made during the development of my game project where I wanted to change the skybox that I used to another.



Task 13 - M3.1 – Present and Communicate appropriate findings.

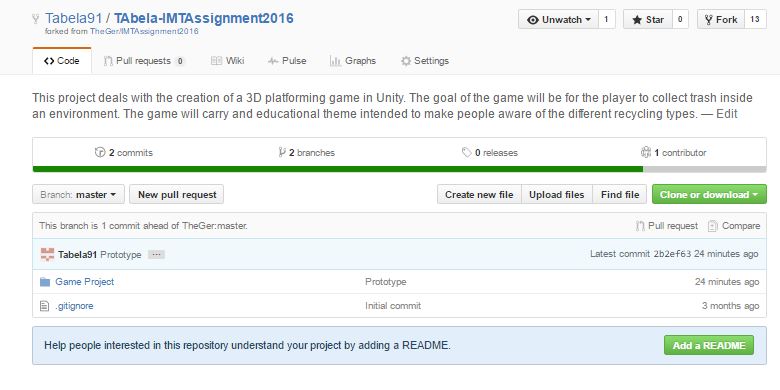
Whenever a developer is developing a new feature of a software project, or fixing a specific bug, it would be ideal to carry out such tasks on a new ‘branch’ of the master project.

The Master project is considered to be the Master branch, and developers should be very careful when making changes to the Master, as it is basically meant to be considered the most advanced, working stage of the project at any point in time. So in order to ensure the Master Branch does not suffer any undesirable changes, what we do instead is create a new branch of the Master project.

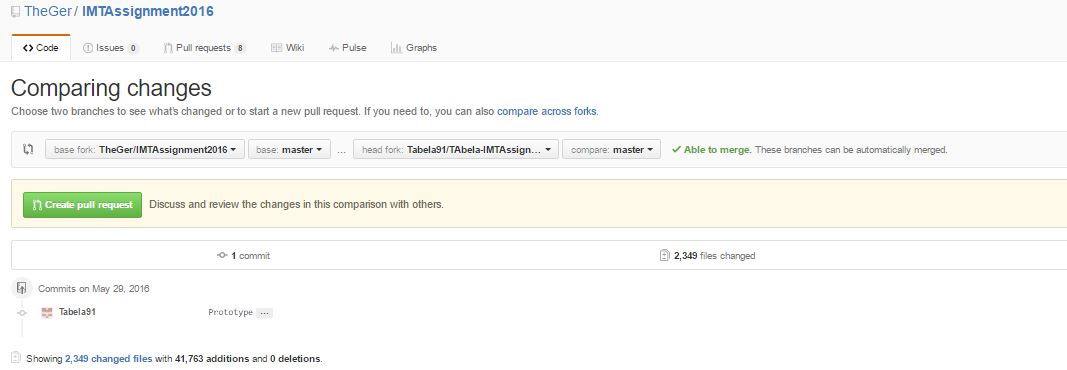
The features or fixes that the developer wants to apply to the project, will be carried out on this new branch. Once the feature or fix is complete, the developer can open a pull request, which opens that branch up for evaluation by a repository collaborator. This opens a discussion thread where the changes made in the new branch can be discussed, suggestions or further changes may be proposed. If the repository collaborator approves of the changes made by the new feature or fix, then that branch can be merged to the Master Branch, and the features or fixes will now have been committed to the main project. Of course it is also possible for the pull request to be outright rejected if it is not seen as a suitable change.

The fork taken from the IMTAssignment2016 project for example, is a branch of the original project on which we can add updates to the game project we are developing.

Here you can see the ‘master’ branch of the TAbela-IMTAssignment 2016, which was forked from TheGer/IMTAssignment2016.



Here we can see the option for a pull request, where the option to merge the TAbela branch with the master branch should the project moderator approve of the request.



Task 14 – D1.1 – Use Critical Reflection to evaluate own work and justify valid conclusions.

A Git rebase, is the ability to take existing commits in a project and place them in a new branch that you may have just created.

When creating a new branch of a project, it’s entirely possible that you create a branch while other users are close to finalizing some hotfixes on the main project. This could result in the branch you have just created becoming instantly outdated within a few days, once that hot fix has been made to the main branch.

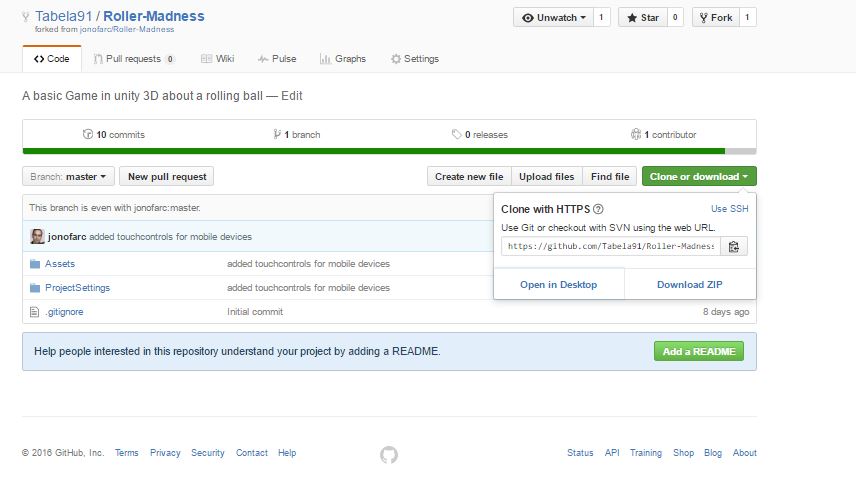
With Git however, users can go ahead and create a new branch, make the changes they wish to add to the project. Rebase allows your branch to be relocated to an older step in the history of the master branch. So all the changes you make to your branch can be added to the master branch as if they were implemented to the project after those hotfixes had already been added.

The advantage of this method is that it allows the history of the project to appear a lot clearer.

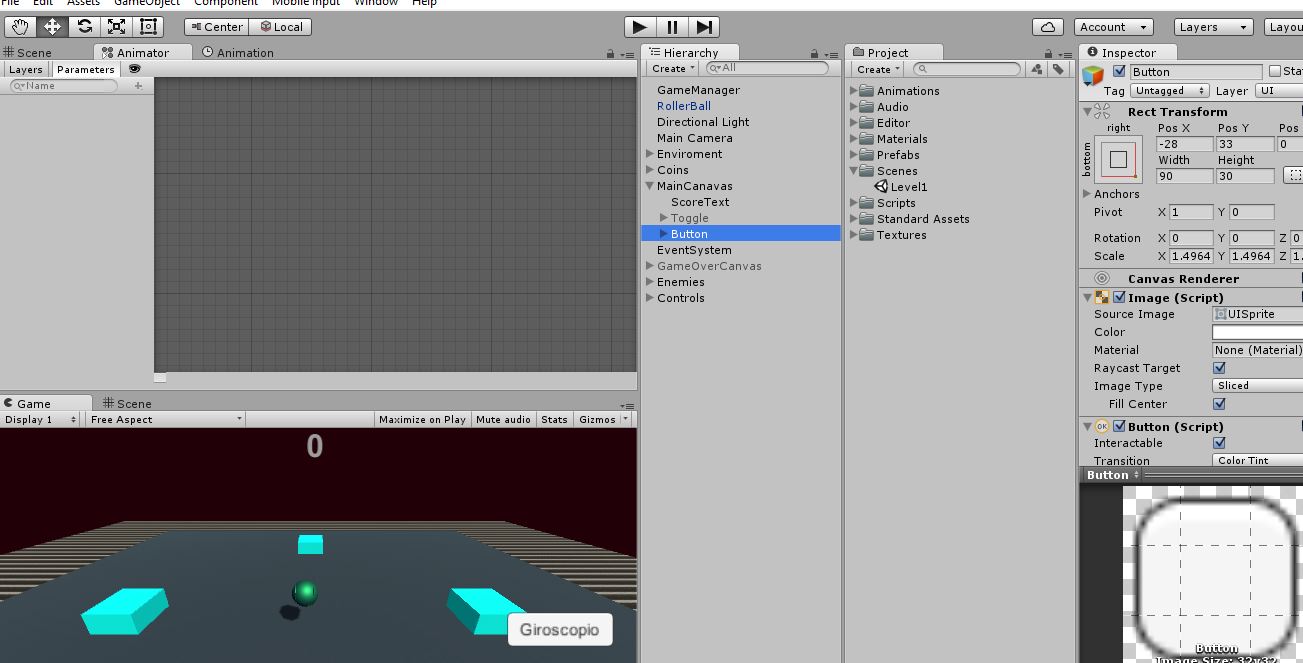
Task 15 – D3.1 – Demonstrate Convergent/lateral/creative thinking.

This task will show how an existing Unity project on GitHub was forked and modified in a new branch.

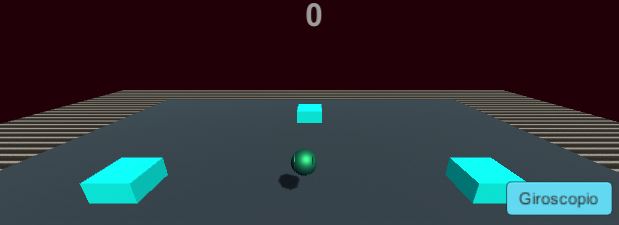
I found a Unity project called Roller-Madness and forked the project.



The project was then cloned onto my desktop and added to a new local repository using the Git Client. I then opened the game project using Unity.



I made a few changes to the project. The first thing I did was change the background color of the button in the canvas to a light blue.



I then opened the Rotate Script and renamed the variable ‘way’ to ‘direction and increased the value of the public float ‘speed’ to 70.f.



You can see here the list of commits in the Roller Madness game project, with my commit “Various Changes”, listed as the most recent commit with a short description of the changes I made.

