Creating a game using digital drawing technologies

Candidate Number: 218721
Games and Multimedia Environments (BSc)
School of Engineering and Informatics
Project Supervisor: Dr Paul Newbury

Year of Submission:
2023

UNIVERSITY
OF SUSSEX

Statement of Originality

This report is submitted as part requirement for the degree of Games and Multimedia Environments at the University of Sussex. It is the product of my own labour except where indicated in text.

Signed:

[illegible]

Acknowledgements

I would like to thank my supervisor Paul Newbury for his continued academic support over the course of this project as well as my parents and friends – Niven, Rie, Eliza, Sean, Vallerie, Olly, Todd and Jo for being the emotional support that helped me reach the point where I am today.

Creating a digital game using drawing technologies

This report is part of an undergraduate project investigating current and existing solutions to making creative games which allow the user to interact with the virtual environment through digital drawing and painting techniques.

The report details the process of creating such a game and conducts background research on what current and previously used methodologies can be implemented to realise a game of such a topic as well as what aspects of these current methodologies work well and those that do not.

The report also discusses the professional considerations and ethics of publishing video games and digital media in a modern society, detailing the current systems in place to help prevent inappropriate content from reaching the wrong audience to accessibility issues and digital solutions to help those with impairments and disabilities still enjoy the same media as the rest of the world.

Following on from this discussion, the report goes into extensive detail on the design and development process of a modern game made in the Unity engine, detailing the elements which go into making a digital multimedia product, including programming and engineering solutions, graphics pipelines and shader programming, the creation of 3D assets with 3D workflows such as Blender, Cinema 4D and 3DS Max as well as miscellaneous features of game design and development, such as creating an interesting and engaging gameplay loop and giving life to a digital world and its characters through the use of narrative and dialogue systems.

The importance of testing is also discussed, in particular user testing and its importance to the ideology of iterative design, a prevalent design philosophy used in the video game industry.

This project is made in accordance with the British Computer Society (BCS) Code of Conduct and considers the standards and benchmarks laid out in order to create a project which is ethical and fair.

Contents

1.0 Introduction

1.1 Background

With this project, I intend to explore digital art technologies and their uses in video games to create an interactive and unique player experience. Due to the nature of providing a player with a creative tool and an interactive sandbox environment, the games created using said technology can be incredibly creative and personalised to the user, allowing for fun and unique player-to-player interactions when the end results of the user's experience can be shared. For this project, I would like to explore the concept in a 3D environment as I believe this will allow a unique approach which will allow for the most depth.

There is an important distinction to make, however, as if the designer drifts too far in the direction of an interactive drawing environment they can end up creating a game that lacks any substantial depth and likewise, if they end up drifting too far in the direction of gameplay or narrative, they can risk not taking full advantage of the unique opportunities and scenarios that arise from creating a game experience which can be fully customisable. Whilst both are fine in isolation, it is important to make sure that the game has a well-defined direction when it is being made as some games demand more attention in either direction. It is important to define whether the drawing technology is going to be the focus of the game or if it is going to be used to complement it.

The approach that will be taken with this project is giving the player a game world that acts essentially as a blank canvas, allowing them to paint onto existing surfaces and geometry and interact with the world in that way. There are a couple different approaches we can take but I believe this to be the richest in terms of gameplay and creative ideas. Due to the nature of the creative tool that I have developed, I believe that my game will work better exploring the painting concept in as much detail as possible, rather than using it as a footnote for other more generic gameplay elements. In this sense, my game will work more akin to *Chicory: A Colorful Tale* [1], where the gameplay is built around using the creative tools to interact with the world and create unique gameplay challenges.

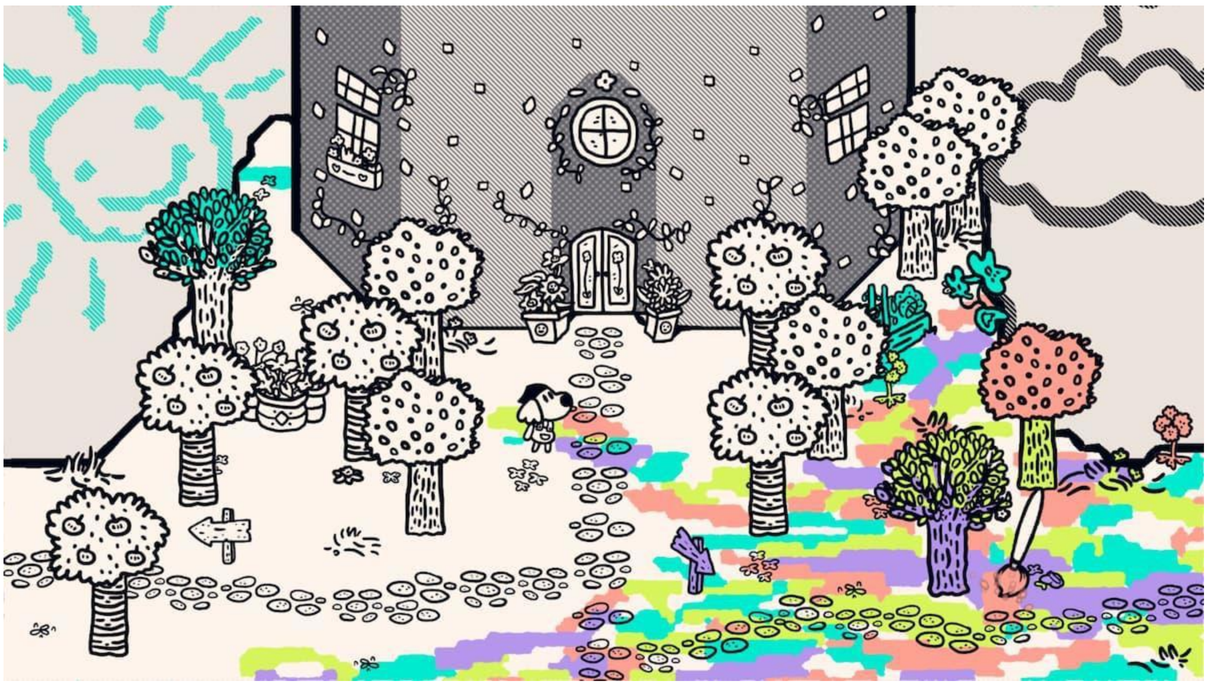


Fig. 1.1: Painting in "Chicory: A Colorful Tale" is used to interact with the environment and create a unique game world for each player.

1.2 Project Aims and Objectives

For this project, my main aim is to create a creative sandbox game themed around graffiti and urban areas, with a focus on using a painting mechanic to create a unique gameplay experience and interesting world which allows the player to express themselves fully within the game world while also providing the player with intriguing puzzles and engaging platforming challenges. I am conducting research into similar games on the market to see what works and what doesn't work in hopes to apply the knowledge I gain from this process into my own game and create the best experience possible.

I have outlined the main objectives of the project below:

- Conduct research into existing solutions and market space for games that use digital drawing technology to enhance their gameplay.
- Develop a prototype for the main gameplay functionality that allows the player to draw freely onto the in-game environment.
    - This should allow the player to change the size and colour of the brush used, as well as the shape and texture of the brush.
- Use this prototype to further develop gameplay mechanics that branch off the painting mechanic, allowing for versatility in level design and puzzles using different types of paint.
- Develop a test plan and perform user research to inform design decisions while getting. feedback on what does and doesn't worth with the game
- Create and rig a fully modelled 3D character that the player can control and use to interact with the game world.

I believe these to be realistic and achievable goals, some of which I have already began to work on or have completed, giving me a good direction for the project.

1.3 Problems and Challenges

1.3.1 Development problems

On pages 3 and 4 of "*The Art of Game Design: A Book of Lenses*" by Jesse Schell, Schell lists a selection of skills needed for a game designer to possess in order to make a successful game under the sub chapter "What Skills Does a Game Designer Need?"[2]. Of the 20 skills listed, I believe the following to be most important to this project:

Animation – I plan for the gameplay of this project to switch seamlessly between first and third person, first person for focussed painting gameplay sections and third person for general exploration and platforming of the 3D environment. In order to make this work, I plan to provide the player with a 3D modelled and rigged player character which will require animation work. This animation work, however, can be resource intensive and time consuming and the timescale of this project is already rather short compared to that of professional industry projects, so while important, there will be less focus on this aspect than the core functionality and gameplay programming.

Engineering – This is a project with a relatively high engineering complexity, and as such, will require a very high level of technical proficiency in order to be fully realised. The programming in this project may require being able to write shaders and code specifically to be ran through the GPU rather than the CPU due to the graphical nature of the project.

Management – For a complex project such as this, management skills are critical to its success, whether that be for a solo designer or a team of dedicated specialists. The specifics of how this project will be managed will be covered in more detail in chapter 4.

Music and Sound Design – This is two sections combined into one – audio is a critical component of games and is a section that is relatively easy to overlook. Developers can make use of audio to make a game feel more immersive and real and this project is no exception to that rule. I, however, do not possess incredible audio skills, so this may be outsourced to a third party (the implications of which I will cover in section 1.3.3).

Visual Arts – Visuals are incredibly important to a game and its overall feel. There is little joy to be derived from a game that does not have visual appeal. In Steve Swink's "*Game Feel*"[3], he discusses the fundamentals of "game feel", a largely intangible phenomenon that describes what makes a game feel good to play. One of the three building blocks of game feel, outlined on page 2, is *polish*. Polish in the context of game feel encompasses many elements ranging from audio design to visuals. Without polish, the core functionality of the game would be all that remains. In this project, I intend to do a large amount of the graphical work myself as it is something that I have an extensive professional background with and is something I believe I will be able to provide to an appropriate standard.

1.3.2 Design Problems

Whilst not as scientific or well defined, game design is a core concern of any game development project. Poor design decisions can lead to a frustrating and unenjoyable user experience that leaves the player unattached and disengaged from the overall experience. The design of the game outlined in this project is overall more casual and laid back than most, more akin to a game like *Minecraft* or *Stardew Valley*, where there are very few restrictions and defined end goals in place, allowing for the player to progress at their own pace rather than a game like *Super Mario Bros,* where there are defined end goals for the player to reach and a set path to reach it.

Herein lies the challenge with this project's design – how can a designer create an experience that is engaging for the player whilst remaining undemanding and relaxing? The answer to this, I believe, lies in psychologist Mihaly Csikszentmihalyi's concept of the *Flow State*[4]. Csikszentmihalyi defines the flow state as "*a state in which people are so involved in an activity that nothing else seems to matter; the experience is so enjoyable that people will continue to do it even at great cost, for the sheer sake of doing it*". While there will be conventional gameplay challenges in this project, the core gameplay loop will mainly be relegated to painting and as such, I would like to achieve a state of flow in the player. This, I believe, will keep the player engaged and invested in the gameplay experience. The player's creations in the game world will impact the behaviours of the world and its inhabitants, characters will respond dynamically to the changes made by the player, and this will in turn provide the player with a motivation to continue onwards towards the overarching goal.

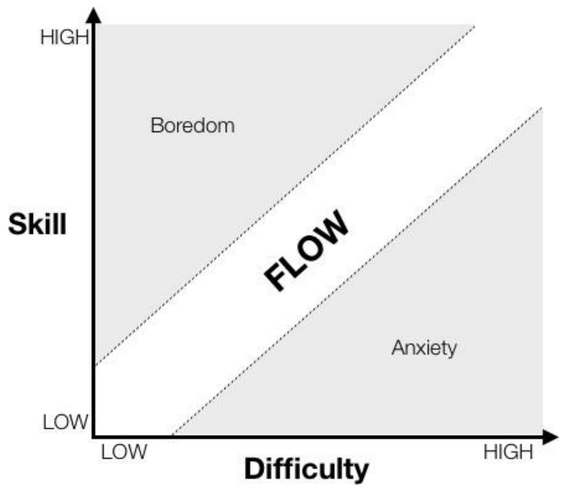


Fig. 1.2: A graphical representation of the flow state.

1.3.3 Asset Flips

Something that I am keen not to do with this project is to perform an asset flip. Whilst there may be assets and code in the game which are not of my own creation due to the time constraints, any such assets will be appropriately sourced and credited in line with section 1b of the BCS Code of Conduct (this is covered in more detail in chapter 3 of this report).

The next chapter will go over existing solutions from similar games and what a developer can learn from these solutions in order to make a game which caters well to the target audience.

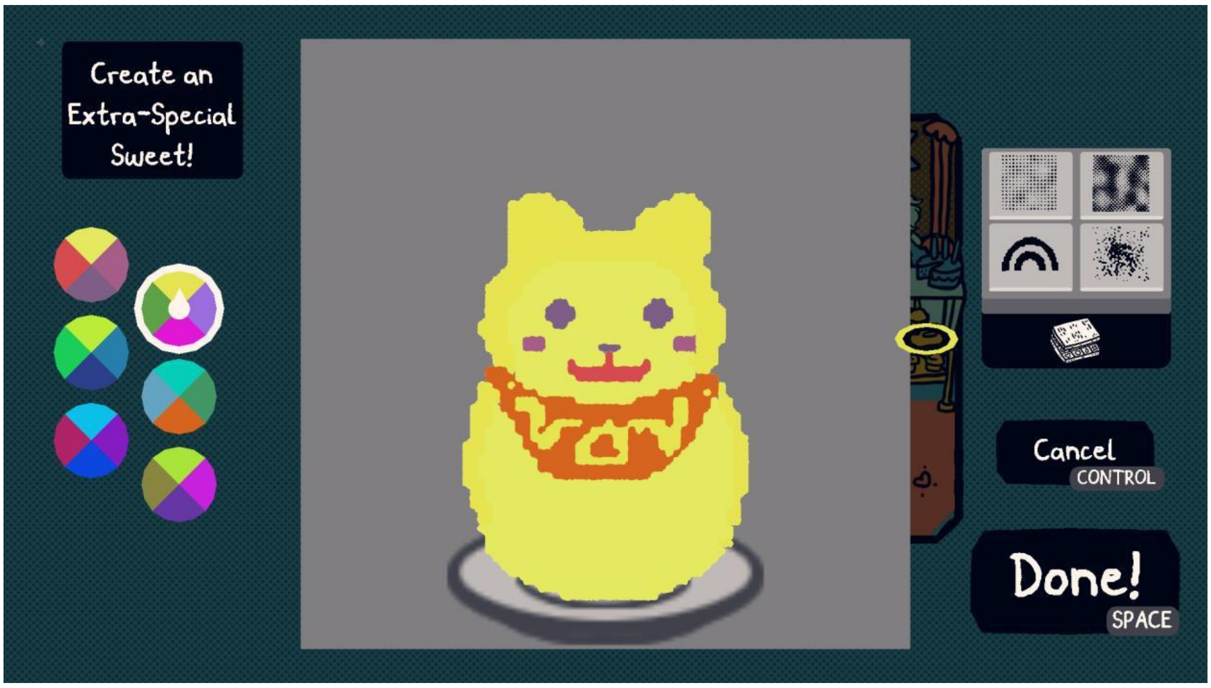
2.0 Requirements Analysis

2.1 Background Research and Existing Solutions

Whilst a creative and unique idea, a game where the main gameplay concept revolves around painting either onto or around the game environment is not one without history. There are a couple of examples in the gaming sphere which approach the problem raised in this project. The first example I want to talk about is Chicory: A Colorful Tale, the main inspiration for the project and a game which I have referenced earlier in this report. Chicory takes a different approach to the painting mechanic, as it is created in the GameMaker engine so there will be some differences between their solutions and mine. In his 2019 article for gamedeveloper.com[5], Greg Lobanov (main developer for *Chicory: A Colorful Tale*) talks about the solutions he used to make the painting mechanics in his game. In Chicory, Greg makes use of a grid-based system containing an integer value from 0-4 where 0 represents an empty square and 1 through 4 represent squares which contain one of the 4 available colours in each area. These 12x12 pixel squares are made to look more realistic and believable through the use of marching squares[6].

The solution I intend to implement makes use of unity's SetPixels[7] method, which takes a provided RGB value at a given position (in the case of this project, this position is determined from the mouse position using a raycast hit) and overwrites the pixel value at that given position with that of the provided RGB value. This can be a problematic solution if not performed correctly as going through a given list of pixel values and manually changing the colour data for them one by one can be very CPU intensive and create lag, detracting from the overall experience. If possible, I would like to find a solution for this problem that allows these operations to be completed using the GPU as it will be more efficient. This will most likely be achieved through unity's compute shaders[8].

In terms of game design, the project will employ a similar solution to that used in Chicory, where the player is offered interactions with NPCs asking them to create something that fits to a certain theme, which is then used in the game world to inform character interactions.



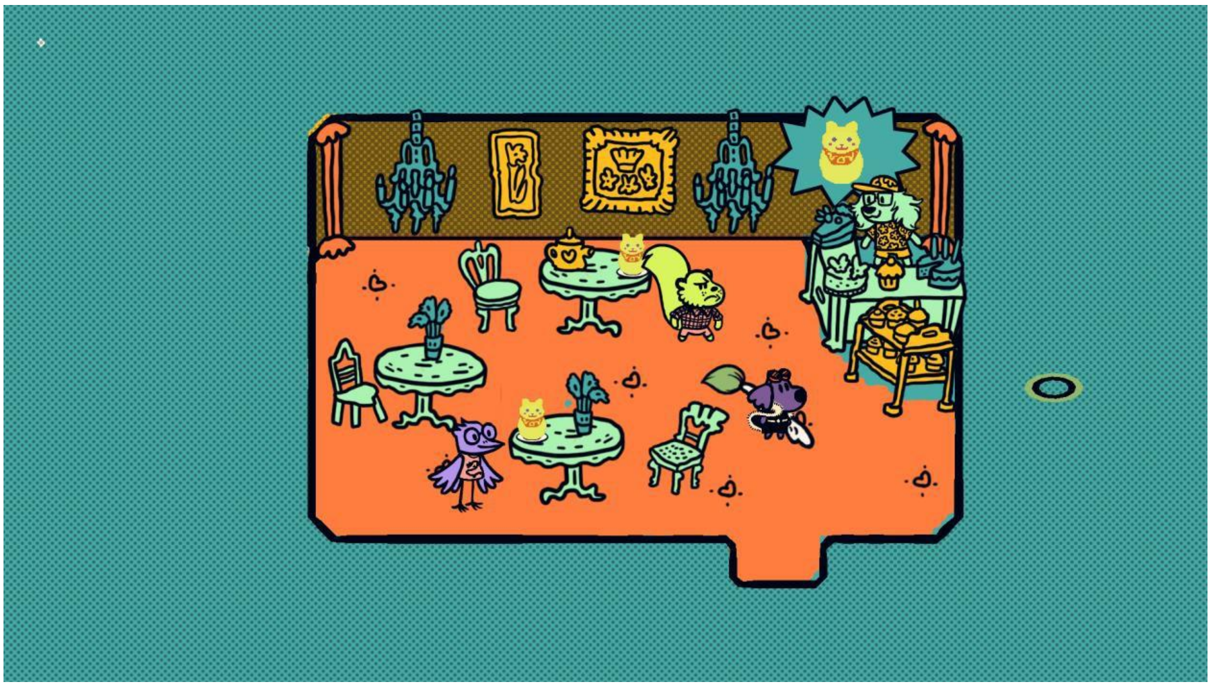


Fig. 2.1 and 2.2

The player is asked to make a sweet which is then placed in the game world and interacted with by the NPCs.

Since this example is in a 2D game, there will be some differences in execution, as it would not look correct if an NPC asked the player to create an object in 2D that would then get placed into a 3D world and I do not believe it to be within the scope of the project to provide the player with 3D art tools. A game that makes a similar approach, more likely to be in line with that of this project is *SuchArt: Genius Artist Simulator*[9], a game which aims to provide a first person 3D simulation of being an artist. In SuchArt, the player is provided with a range of different objects to create art with, ranging from canvas to sculpture that they can paint and edit how they see fit. Once finished, the player can display their pieces for NPCs to comment on in a gallery, as seen in figure 2.3.

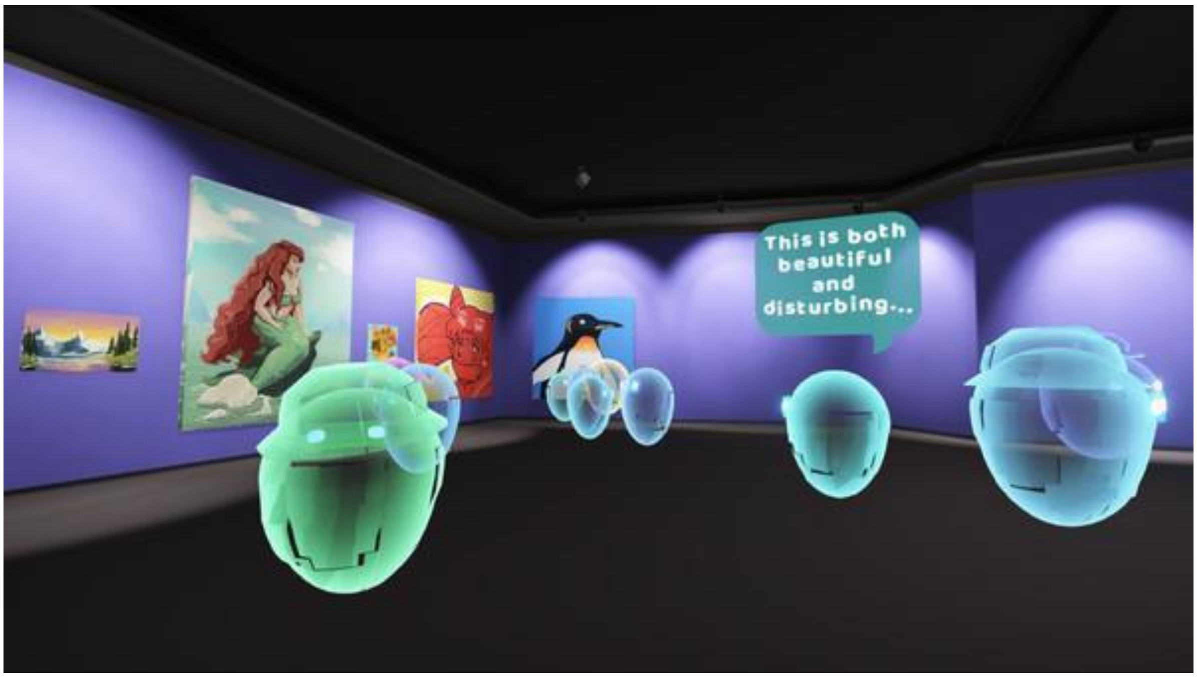


Fig. 2.3: User created art pieces in SuchArt set up for display in an in-game gallery for comment by NPCs.

This project will attempt a solution that combines these two existing solutions into one. The project will be set in a city, with a focus on street art and graffiti, which will inform the environment, UI and characters. In this regard, it will have a similar visual style to that of *Jet Set Radio*[10], while deviating from Jet Set Radio's input-based arcade style painting in favour of attempting to incorporate the more in-depth painting mechanics of Chicory and SuchArt. It is important to follow in the examples of Chicory and SuchArt with a game like this and provide the player with varying scenarios to create art in the game, asking them to do more than just paint walls, instead giving them a range of activities ranging from designing billboard adverts to painting a sculpture. This way the gameplay loop can be explored in depth and remain within the scope of the project, but still provide the player with ever-changing gameplay that doesn't leave them bored and outside of the flow state, as mentioned in chapter 1.

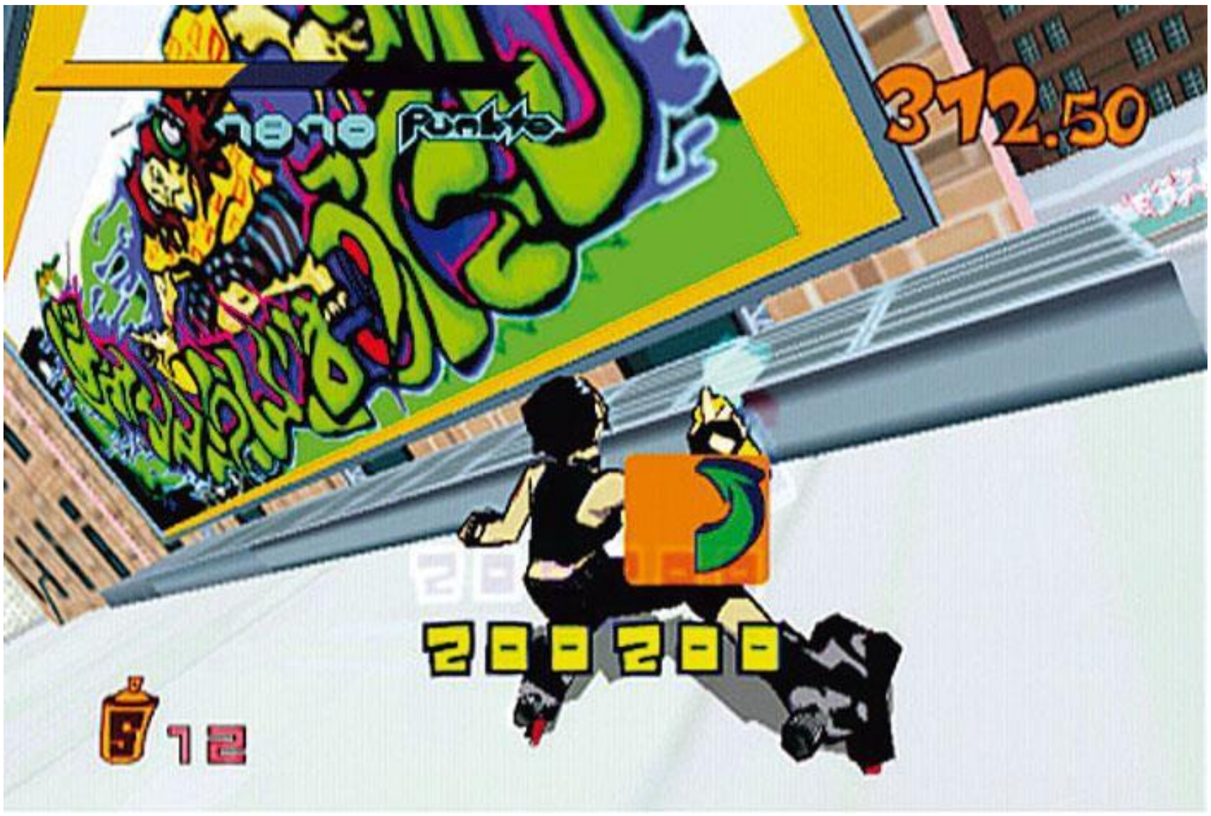


Fig. 2.4: Painting graffiti onto the game world in Jet Set Radio through pre-defined inputs.

2.2 Project Requirements

2.2.1 Mandatory Requirements

There are some essential requirements that the project should adhere to. These are more in line with technical requirements to ensure that the game runs smoothly and is as accessible as possible. These are not gameplay requirements; those will be outlined in the next section. The game must meet the following criterium:

- The game must run on Windows 10 systems.

- The game must take input from common control types, including keyboard and mouse as well as gamepad input (controller).
- The game must not experience substantial frame drops and must run at a consistent frame rate of at least 30fps (ideally 60fps).
- The game should have a mean time to failure (MTTF) of at least 1 hour, e.g. no bugs or crashes should inhibit the gameplay experience and cause the player to be unable to progress or lose progress.
- The game should be able to run on all up-to-date modern hardware for accessibility purposes.
- The game should provide the player with accessibility options for people with disabilities such as colour blindness or those hard of hearing.

2.2.2 Mechanical Requirements

Due to the nature of game development, it is important to have an idea of where you want a project to go before completing too much work on it as otherwise this can lead to a game feeling aimless and incomplete. In chapter 1.2, *Project Aims and Objectives*, I defined the objective of "*use this prototype to further develop gameplay mechanics that branch off the existing painting mechanic"*. Whilst this objective is vague, I have included it like this as ideas and mechanics tend to develop naturally over the course of a game's development and this project should be no different, especially due to the open-endedness of the concept. It is important, however, to ensure that a project's development doesn't fall victim to *feature creep* and *scope creep*, phenomena where developers continuously build on ideas in a game to a point where the game loses sight of its end goal and ends up in a perpetual cycle of development. To avoid this, the project should clearly set out what features, mechanics and levels are going to be included before going deep into development and stick to them rigidly so that there is always a definitive end goal in mind.

With that in mind, these are all the features that I currently aim to implement:

Feature	Description	Difficulty
Painting (core)	User will be able to freely edit and draw onto most textures in the game – essential to development	Medium (2-4 weeks)
Brushes (core)	Allow the user to change the brush shape and texture used to draw	Easy (1-2 weeks)
Dialogue System (core)	Create a dialogue system that allows the player to interact with characters in game and accept quests and have conversations	Medium (2-4 weeks)
Create an animated and rigged player model (core)	Create a player model which has animations and basic movement such as walking, running and jumping	Medium (2-4 weeks)
Create a small open world sandbox environment (core)	Create a small but open environment for the player to explore and create in – this can include different buildings and sub-sections with hidden secrets and challenges, but must all be connected	Hard (4-6 weeks)
User content sharing system (optional)	Create a system that allows users to share screenshots of their paintings to a web server where other players can browse user content	Hard (4-6 weeks)
Camera/Screenshot mode (optional)	Allow the player to take pictures of their game world with a dedicated and tailor-made camera mode with in-depth controls and filters	Medium (2-4 weeks)

Ability to store and save changes made by the user (core)	Allowing the user to save the edits they have made to the game world locally to their computer so that their world doesn't get reset after each play session, could also allow for players to share their game worlds with each other	Easy (1-2 weeks)
More realistic paint texture with marching squares (optional)	Current solution to painting is done with raw pixels which makes paintings look like they were made with a binary brush, marching squares can be used to create a more natural looking paint shape and texture	Medium (2-4 weeks)
Ability to switch between a first and third-person perspective (core)	The user must be able to play the game in either a first or third-person perspective and should be able to switch between them at any point in runtime	Easy (1-2 weeks)
NPCs should be able to comment on the user's creations (optional)	The player should be able to interact with NPCs and get feedback on their creations	Medium (2-4 weeks)
NPCs must be able to give the player tasks to complete (core)	The NPCs should provide the player with objectives to complete, this will take the form of asking the player to create or paint something in the game world which can then be interacted with as per the previous point	Hard (4-6 weeks)

I have marked each of these requirements as core or optional, indicating whether development can be completed without them or not, I have also given each of them an estimated duration for completion, ranging from 1 to 6 weeks. I will be working on multiple features at the same time so I have given them more time for completion than I would expect as there are multiple factors which can affect development and the time it takes to complete a feature. I aim to complete all the features marked as core and to finish as many of the features marked as optional as possible.

The following chapter will discuss the ethical implications and the professional considerations of the project, what to consider in terms of risks and potential issues such as privacy and offensive user created content.

3.0 Ethics and Considerations

3.1 Professional Considerations

Since I am making a game for my final year project, there are far fewer professional considerations for me to consider – I am not handling potentially sensitive user data, neither am I not doing any user testing or research that could potentially bring up ethical issues and I am not developing any product or solution that could potentially do harm to any involved bodies. The user research I will be conducting on this project will be entirely observational, observing how the user interacts with the game and the game environment in a generalised fashion rather than collecting specific or personalised user data.

There are, however, other issues to consider with this project. For starters, we must consider potential age ratings and the content included within the game that would give it such a rating. My game will be relatively tame, with maybe a few pieces of dialogue that could be considered inappropriate for younger kids, so it's safe to imagine that the game would probably receive a rating around PEGI 7 or PEGI 12. Currently, there is no plan for any form of violence in the game as I don't think it would fit the theme, however due to other factors such as potentially inappropriate dialogue, the game may reach a 12 rating, although the dialogue of the game is largely undecided at this time.

The main point of consideration for ratings, however, is user shared content. Should the feature allowing for user content to be shared online with others be completed, there is a possibility that users could create malicious or offensive content, which is inappropriate for younger audiences, which could be shared online. In an ideal scenario, a system would be put in place to analyse content and reject content which is potentially harmful, but I do not think I have the time or knowledge to properly implement something like this. There is, however, a descriptor used by the ESRB for online content in situations like this where the online content based on user interaction is separated from the content of the game itself and does not contribute towards its rating. My game will not include any of the descriptors specified by the PEGI ratings board, all of which can be seen below.

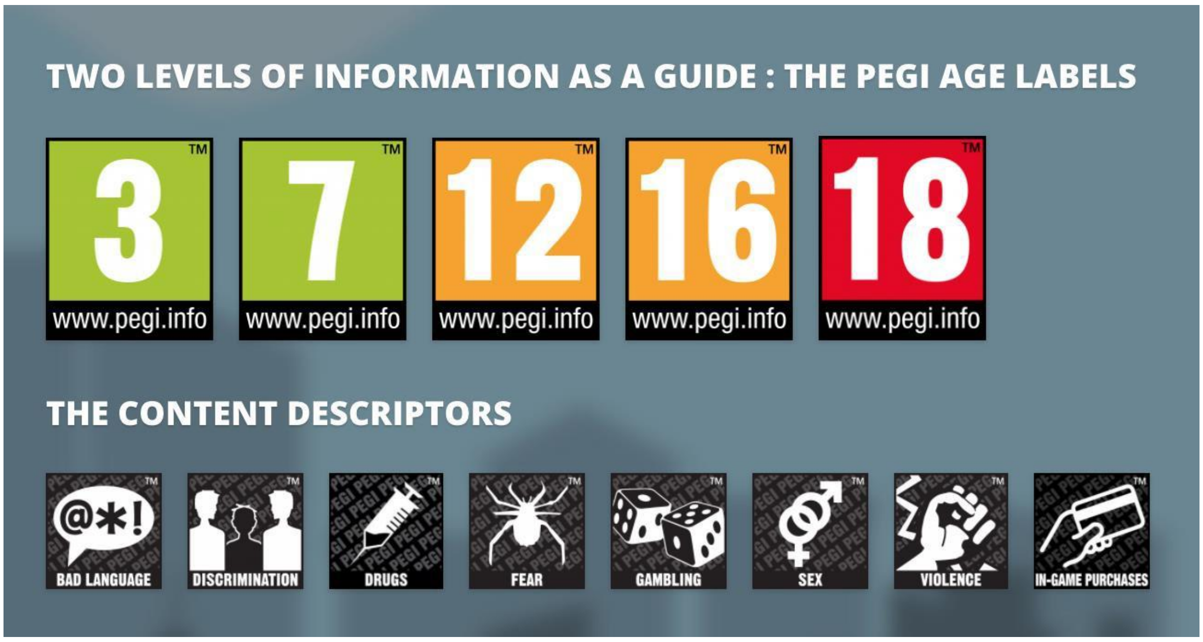


Fig. 3.1: The PEGI ratings board labels and descriptors.

3.2 Ethical Considerations

3.2.1 General Ethics

Since the project is a game development project, there are important ethical considerations to consider, but they are slightly different to those of a more general computer science project. This project will not be handling any personal data and data relating to the user will be entirely observational and will focus on how the user interacts with the game as a generalised entity. There will be no information that can be used to identify individual users.

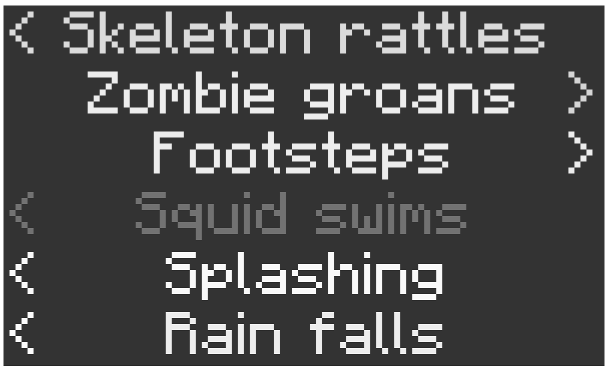
3.2.2 BCS Code of Conduct

The project will conform to the ethical testing standards set out in the BCS Code of Conduct. Section 1a of the BCS Code of Conduct states "*You shall have due regard for public health, privacy, security and wellbeing of others and the environment.*" – in terms of the project this means that it will take into consideration any potential issues that could be caused by the gameplay. Such considerations include potential epilepsy triggers, headaches and dizziness from extended play sessions as well as other generalised health problems in relation to extended video game usage.

Section 1b states that "*You shall have due regard for the legitimate rights of Third Parties*" – in the context of this project this can relate to any assets or code used in the project that are created by anyone other than myself, any referenced code or assets will be credited to the relevant party.

In section 1c, it is stated "*You shall conduct your professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement.*" – this is a section of the code of conduct that doesn't have a lot of relevance to the project as by the nature of the project I won't be conducting any tests based on users' personal traits, just observing their behaviour in relation to the game and specific scenarios.

Likewise with section 1d "*promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise*" - in relation to this section, the project can make sure that there are options for accessibility wherever possible. This can include colour blind options, changes to field of view and audio description as seen in games like Minecraft (as seen in figure 3.2). It is important to keep the project as open and accessible to players as possible as we don't want to keep people from being able to enjoy the game because of factors outside of their control. Likewise, the game should be made to run as well as possible on all currently available hardware so that people from less privileged backgrounds aren't prevented from playing the game because of an inability to run the game from outdated hardware.



< Skeleton rattles
Zombie groans >
Footsteps >
< Squid swims
< Splashing
< Rain falls

Fig. 3.2: Subtitles (audio description) in Minecraft.

Section 2d claims "*You shall ensure that you have the knowledge and understanding of Legislation* and that you comply with such Legislation, in carrying out your professional responsibilities.*" – Once all the main functionality of the game and the content planned to be in the final release of the game has been completed, a consultation with PEGI will occur to decide what audience best fits the game and its contents, as outlined earlier in section 3.1.

In section 2e, it is stated "*You shall respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work."* – A core aspect of game development is to take feedback from all testers, regardless of their background and competence in the field. It is important to value feedback from all users since they are the audience you are trying to impress, whether they have a deeper understanding of the design and theory behind one's decisions or not.

Finally, section 2g states that "*You shall reject and will not make any offer of bribery or unethical inducement."* – There is no benefit to tampering with the views and feedback of any involved parties, false positives cannot change fundamental issues with a game or its mechanics so it is important to get honest, uninfluenced feedback and responses from user testing regardless of if it is positive or negative as this can identify issues which can be worked upon and improved in order to create an overall better product.

These are the points in the BCS code of conduct which I believe to be most relevant to the project, the full code of conduct can be found in the appendix attached.

4.0 Design and Implementation

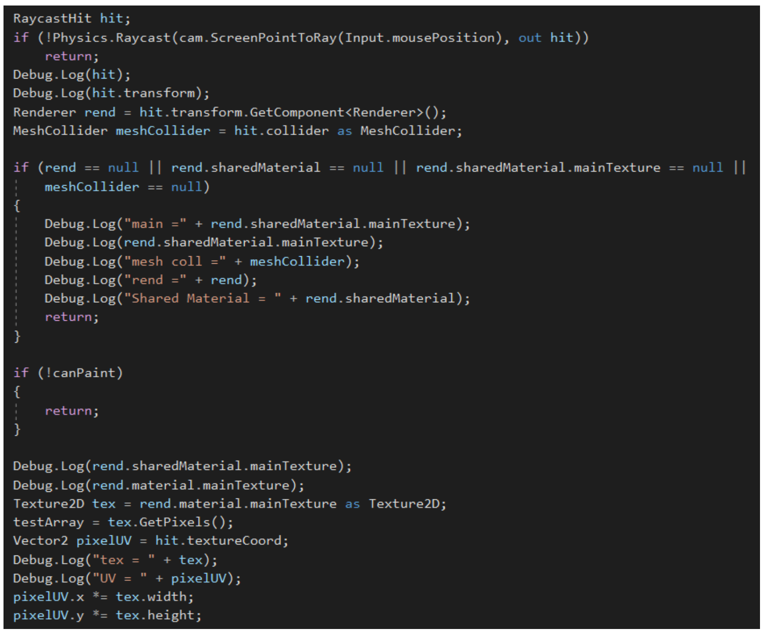
Due to the nature of this project, multiple areas had to be tackled to achieve the desired outcome. This included work and research into areas from different disciplines including:

- Dynamic Texture Programming
- 3D Modelling and asset creation
- Shader Coding
- Dialogue System
- Gameplay

4.1 Dynamic Texture Programming

The game's main feature is its painting mechanic. Throughout the development of the project up to this point, multiple different methods of implementing this mechanic were researched from creating a custom shader to Unity's decal feature. However, in the end it was decided that the project would use the *SetPixel* method as this would be the easiest way to implement the feature with my skillset.

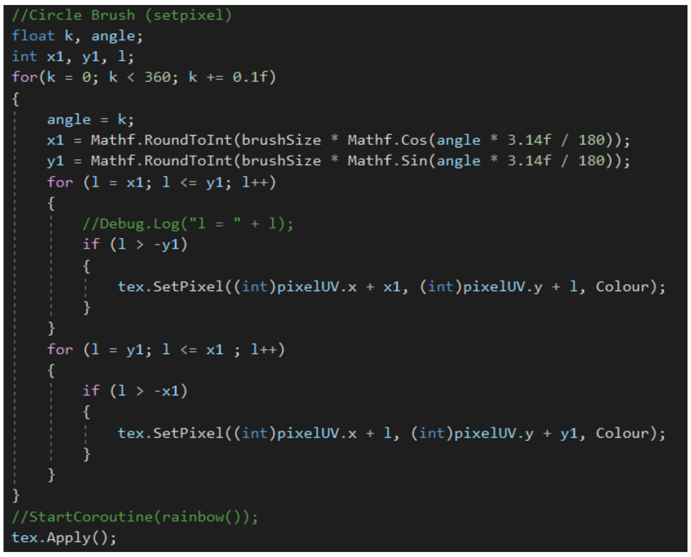
The painting mechanic works by making use of raycasts. When the left mouse button is pressed, the camera sends out a raycast to the centre of the screen. If this raycast returns a hit on an object with a renderer attached, it will access this renderer and use it to access data relating to the object's texture for editing later in the function as shown in figure 4.1.



```
RaycastHit hit;
if (!Physics.Raycast(cam.ScreenPointToRay(Input.mousePosition), out hit))
    return;
Debug.Log(hit);
Debug.Log(hit.transform);
Renderer rend = hit.transform.GetComponent<Renderer>();
MeshCollider meshCollider = hit.collider as MeshCollider;

if (rend == null || rend.sharedMaterial == null || rend.sharedMaterial.mainTexture == null ||
    meshCollider == null)
{
    Debug.Log("main =" + rend.sharedMaterial.mainTexture);
    Debug.Log(rend.sharedMaterial.mainTexture);
    Debug.Log("mesh coll =" + meshCollider);
    Debug.Log("rend =" + rend);
    Debug.Log("Shared Material = " + rend.sharedMaterial);
    return;
}

if (!canPaint)
{
    return;
}

Debug.Log(rend.sharedMaterial.mainTexture);
Debug.Log(rend.material.mainTexture);
Texture2D tex = rend.material.mainTexture as Texture2D;
testArray = tex.GetPixels();
Vector2 pixelUV = hit.textureCoord;
Debug.Log("tex = " + tex);
Debug.Log("UV = " + pixelUV);
pixelUV.x *= tex.width;
pixelUV.y *= tex.height;
```

Figure 4.1: The raycast implementation used to gather data about the texture to be written to.

Information taken about the associated object includes the texture to be edited, where on the texture the raycast landed and the array of pixel data from the texture. It is important to note that this texture must be set to enable read/write operations for the function to work. Following this, the function makes use of some simple mathematics to create a circle of pixels of the current size, which is then applied to the target texture through the use of the SetPixel() method. Once the calculations are all complete, the changes to the texture are written to it through the Apply() method, which is a crucial step as otherwise the texture would not visibly change.

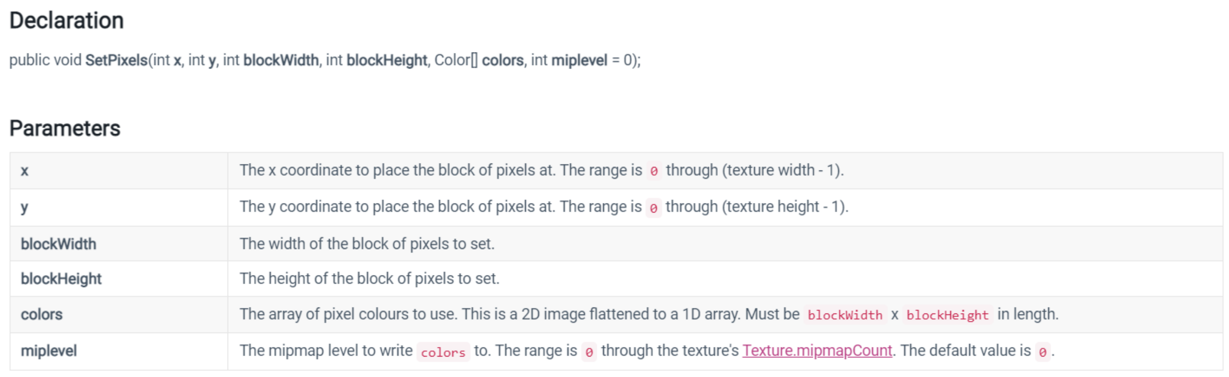


```
//Circle Brush (setpixel)
float k, angle;
int x1, y1, l;
for(k = 0; k < 360; k += 0.1f)
{
    angle = k;
    x1 = Mathf.RoundToInt(brushSize * Mathf.Cos(angle * 3.14f / 180));
    y1 = Mathf.RoundToInt(brushSize * Mathf.Sin(angle * 3.14f / 180));
    for (l = x1; l <= y1; l++)
    {
        //Debug.Log("l = " + l);
        if (l > -y1)
        {
            tex.SetPixel((int)pixelUV.x + x1, (int)pixelUV.y + l, Colour);
        }
    }
    for (l = y1; l <= x1 ; l++)
    {
        if (l > -x1)
        {
            tex.SetPixel((int)pixelUV.x + l, (int)pixelUV.y + y1, Colour);
        }
    }
}
//StartCoroutine(rainbow());
tex.Apply();
```

Figure 4.2: The SetPixel() implementation, with maths used to create a circular brush.

A notable concern with this method is the fact that is has a substantial overhead as it must complete all these operations and rewrite the target texture every frame. Over the course of development, it was attempted to mitigate this performance hit with an array of different solutions including trying to make use of Unity's decal system, however, after enough experimentation and testing, SetPixels() proved to be the most effective method. The SetPixels() method is different from SetPixel() as instead of setting each individual pixel one at a time, it takes an array of pixel data and sets all of the pixels from the array into the texture at the same time, making it considerably faster, up to an entire order of complexity faster under the correct circumstances.

SetPixels() has two declarations, the first one edits the entirety of the mipmap level – with a mipmap level being a specific resolution of a texture. For the purposes of this project, the mipmap level is set to zero, as we only want to edit the default resolution texture that we see as a player. We then feed the function the co-ordinates that we want to write to and the width and height of the block of pixels to set, which we just provide as the current brush size. Since the array starts at the top left corner, we offset the x and y co-ordinates by half of the brush size so that the pixels are edited around the centre of the raycast.

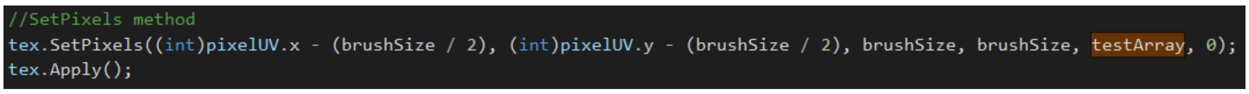


Declaration

public void **SetPixels**(int **x**, int **y**, int **blockWidth**, int **blockHeight**, Color[] **colors**, int **miplevel** = 0);

Parameters

x	The x coordinate to place the block of pixels at. The range is `0` through (texture width - 1).
y	The y coordinate to place the block of pixels at. The range is `0` through (texture height - 1).
blockWidth	The width of the block of pixels to set.
blockHeight	The height of the block of pixels to set.
colors	The array of pixel colours to use. This is a 2D image flattened to a 1D array. Must be `blockWidth` x `blockHeight` in length.
miplevel	The mipmap level to write `colors` to. The range is `0` through the texture's Texture.mipmapCount. The default value is `0`.



```
//SetPixels method
tex.SetPixels((int)pixelUV.x - (brushSize / 2), (int)pixelUV.y - (brushSize / 2), brushSize, brushSize, testArray, 0);
tex.Apply();
```

Figure 4.3.1 and 4.3.2: The unity documentation for the SetPixels() method and its implementation in the codebase for the project.

4.2 3D Asset Creation

The primary tool used to create the 3D assets seen in the game was Blender.[11] Every asset seen in the game is completely original and made by the author within the scope of the project. This includes modelling, texturing and in some cases rigging and animating every 3D object showcased in the game. For the sake of demonstration, this section will be showcasing the workflow used to create the animated NPC seen in the game's tutorial area.

4.2.1 Blender Workflow

The creation of most assets begins with the use of primitives (objects such as cubes and spheres) to block out a silhouette of the end goal. Since the art style of the game is relatively simple and low poly, there is not too much work that needs to be done to move the objects from this stage to their final finished product in most cases. In the case of the NPC character, a quickly sketched model sheet was made to compare the rough form and proportions of the model against.

4.2.2 Character Rigging

Once the character model is made to an acceptable standard, it needs to be rigged. Rigging allows an animator to grab individual parts of the model's mesh and animate them, which can be achieved by giving a skeleton or armature to a mesh. [12]

Creating an armature for a model is done by first adding a root bone, this is the bone that is at the origin of the character mesh, usually placed at the hip region of the model as it does not deform the mesh in any way. The root bone is used to edit the transform of the entire armature as all bones in the object's armature are connected back to the root bone. From here, extra bones are added onto the root bone, slowly building up a full rig, with each bone corresponding to a different part of the mesh.

To get each bone to deform their respective part of the mesh, they must be *weight painted*[13], a process where each bone is assigned a different weight from zero to one for each vertex in the mesh (zero being no influence over the given vertex and one being complete control over said vertex).

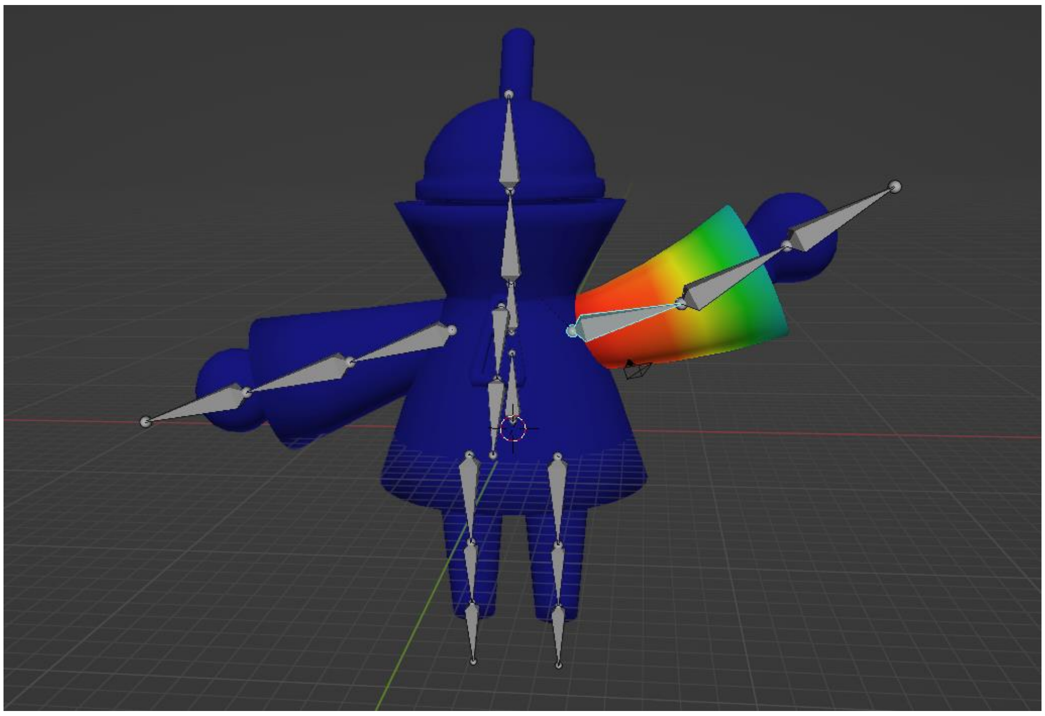
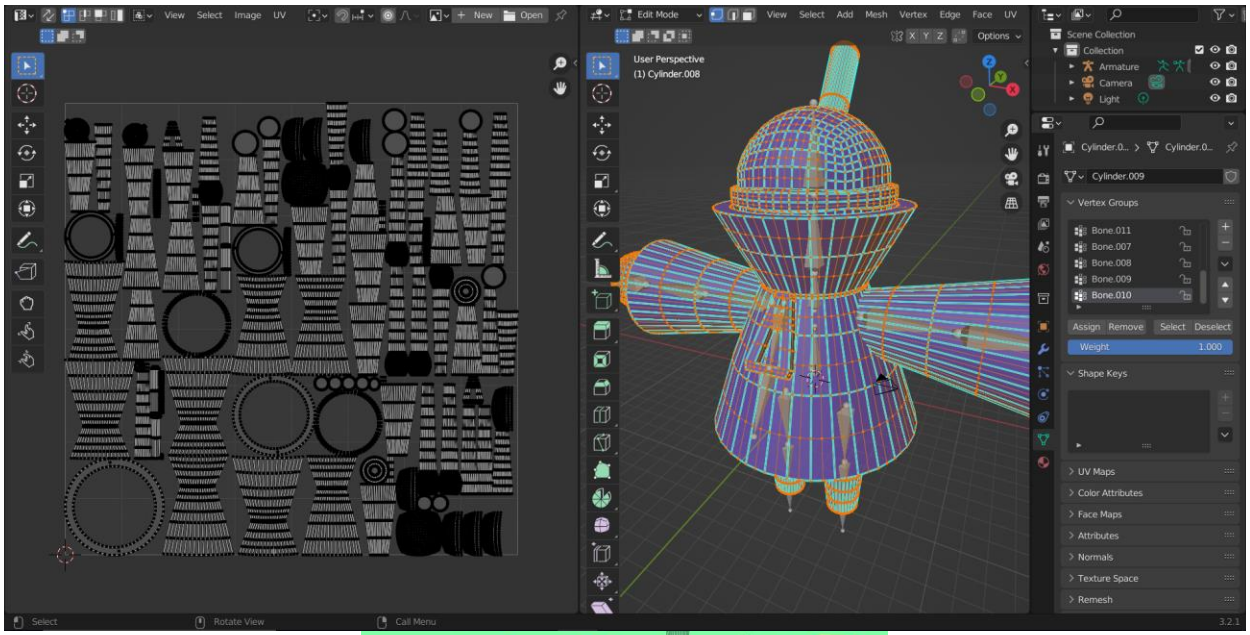


Figure 4.4: The character model's finished rig, with weights shown for the left upper arm bone (red indicates a weight of 1, whereas dark blue indicates a weight of 0).

The final step that needs to be completed before the model is ready to use in the development environment is texturing. Texture can be applied to any model using its UV map. UV mapping is the process of projecting a model's surface onto a 2D image so that individual areas of the image may be programmatically assigned onto the mesh at runtime to give the model texture data.[14] To get this image and start creating a texture, the model needs to go through a process called UV Unwrapping. Blender allows the user to automatically unwrap a model's UVs, however, to make an efficient UV map as seen in industry, it is important to mark seams onto the model. Seams are used in UVs for any section of the map which should be split to ensure the 3D mesh can be properly converted into a 2D UV map.



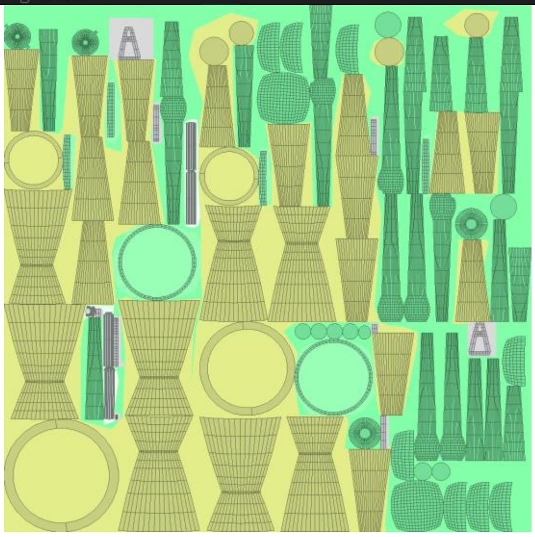


Figure 4.5.1 and 4.5.2: A screenshot of the NPC character's model and its associated UV map in blender as well as the final UV map.

4.2.3 Animation

With the model completed, it can now be imported into Unity for animation. For an object to be animated in Unity, it needs to have an animator component and an animator controller component. The animator component serves as the central point for unity to handle animation on an object. When controlling an object and its animations via script, Unity references this component first.

The animator controller handles the logic behind animation in Unity. This component contains information about the different animations (animation clips) given to an object, when and how to transition between them and any relevant parameters and variables that may be used to inform said transitions. Below you will see the animation controller for the NPC in my project.

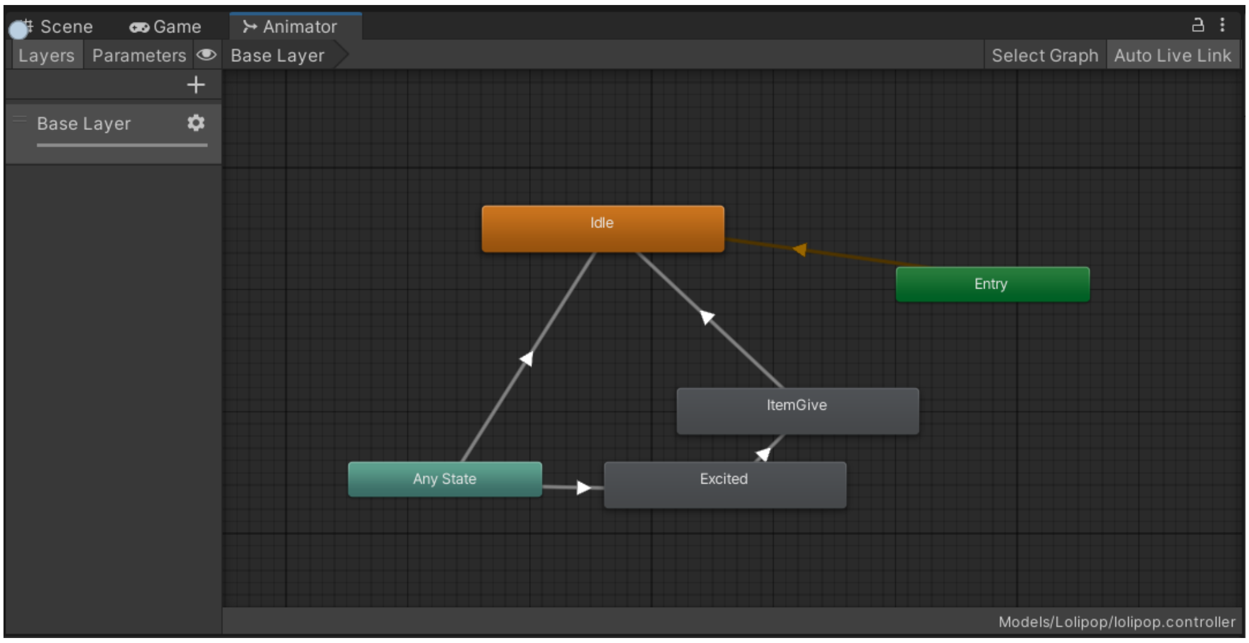
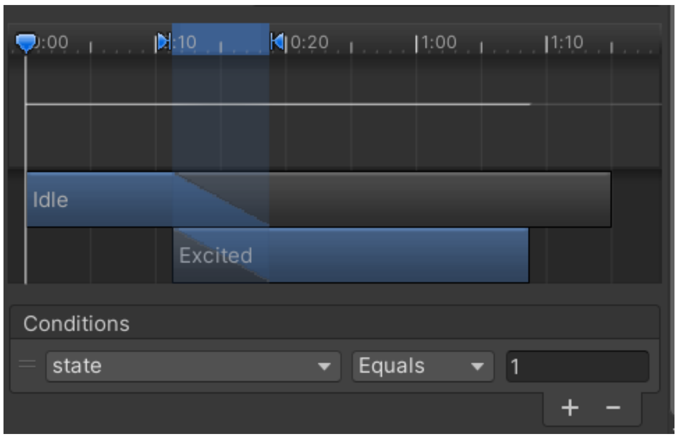


Figure 4.6: The animator controller for the "lollipop" gameobject.

Each user defined state (in this scenario Idle, Excited and ItemGive) each contain data about an animation clip to play when that state is reached. Every animator controller starts off in the "entry" state once activated. From here, a transition is made to the default state, defined by the developer, in this case the Idle state. Since there is no pathway directly out of the idle state, the transition from idle to excited is made once the player performs the action of painting the wall beside the NPC, which will change the animator variable "state" to be equal to 1, initiating the transition from the "any state" state into the "excited" state and play the corresponding animation.

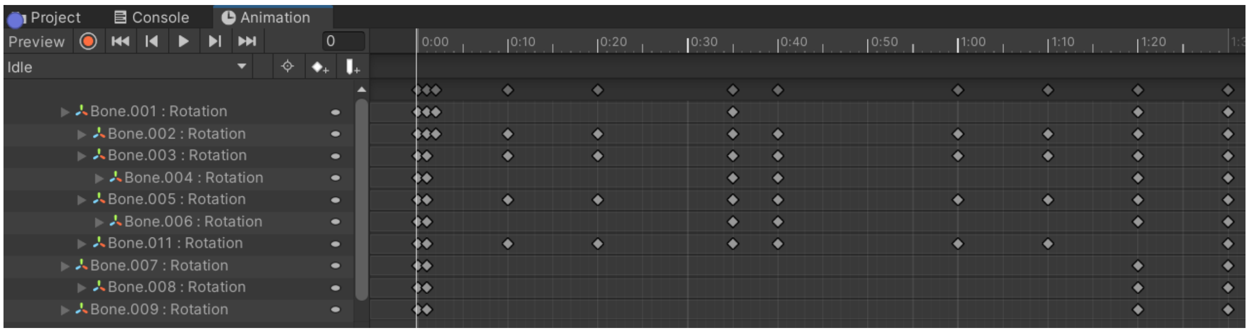


Conditions

state Equals 1

Figure 4.7: The transition information between the Idle and Excited state.

The animations themselves are made with unity's inbuilt animation window. The animation window provides the developer with a keyframe based animation workflow that allows the developer to select individual aspects of the associated game object, ranging from individual bones to associated scripts as well as scale and rotation of bones and meshes.



Bone.001 : Rotation
Bone.002 : Rotation
Bone.003 : Rotation
Bone.004 : Rotation
Bone.005 : Rotation
Bone.006 : Rotation
Bone.011 : Rotation
Bone.007 : Rotation
Bone.008 : Rotation
Bone.009 : Rotation

Figure 4.8: The animation window in Unity.

4.3 Shader Programming

To achieve the desired look for the project, custom shader code would have to be written. Unity handles graphics processing primarily with DirectX and as such, shaders are written in HLSL (High Level Shader Language), a shader language which shares similarities to C. [15] The two shaders made for the project were a cel shader, which replicates the shading of a cartoon, and an outline shader, which works to draw an outline around on-screen objects at runtime.

The cel shader works by requesting lighting data from the scene in the Pass section of the shader code, a section of code in a shader that allows the developer to add post-processing to the scene. The shader makes use of the Blinn-Phong lighting model to calculate the light on the applied surface. The Blinn-Phong model makes use of continuous calculations of the dot product between the viewer and the light source. If we decide instead to calculate the lighting data using the surface normal N and a halfway vector between the surface normal and the light source, H – we can calculate more accurate lighting data than that of just the Blinn model alone.

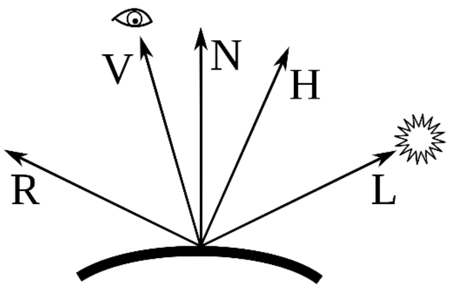
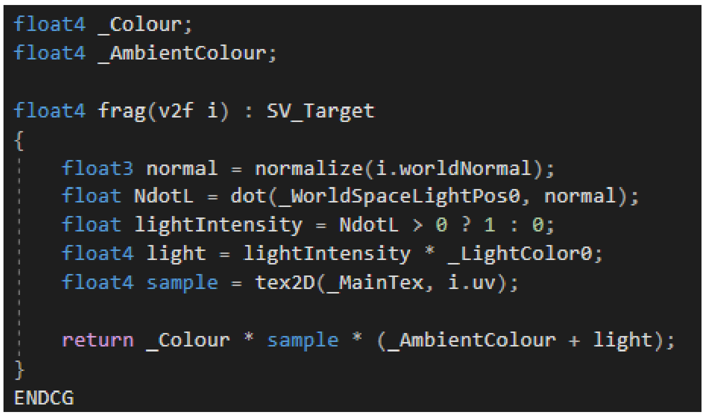


Figure 4.9: The vectors used to calculate lighting data in Blinn-Phong shading [16]
(In this situation, we primarily make use of dot product of vectors N and H).

We can calculate the dot product between these two vectors N and H (N·H) to provide us with information about the intensity of lighting for the viewer. The dot product takes these two vectors and returns a single value – in this scenario when the two vectors are unit vectors, that is to say they are parallel in the same direction, they will return the value of 1 and when they are perpendicular, they will return a value of 0. For the use in our lighting model, a value of 1 is an intense white highlight and a value of 0 is a darkened shadow.



```
float4 _Colour;
float4 _AmbientColour;

float4 frag(v2f i) : SV_Target
{
    float3 normal = normalize(i.worldNormal);
    float NdotL = dot(_WorldSpaceLightPos0, normal);
    float lightIntensity = NdotL > 0 ? 1 : 0;
    float4 light = lightIntensity * _LightColor0;
    float4 sample = tex2D(_MainTex, i.uv);

    return _Colour * sample * (_AmbientColour + light);
}
ENDCG
```

Figure 4.10: The fragment shader used in the final product.

Here you can see the fragment shader used to create the toon shading effect seen in the game. The first step that the shader takes is to calculate the surface normal using information the world normal which is simply calculated with the following line:

```
float3 worldNormal : NORMAL;
```

The shader then proceeds to calculate the dot product between the surface normal and the light source which is used to create lighting data in the following line that declares the variable *lightIntensity*. The *lightIntensity* variable makes use of a ternary operator which helps create the toon-like appearance of the shader. Essentially it takes this dot product and returns a value of 1 if it is greater than 0, otherwise it returns a value of 0, which creates this distinctive cel shaded appearance.

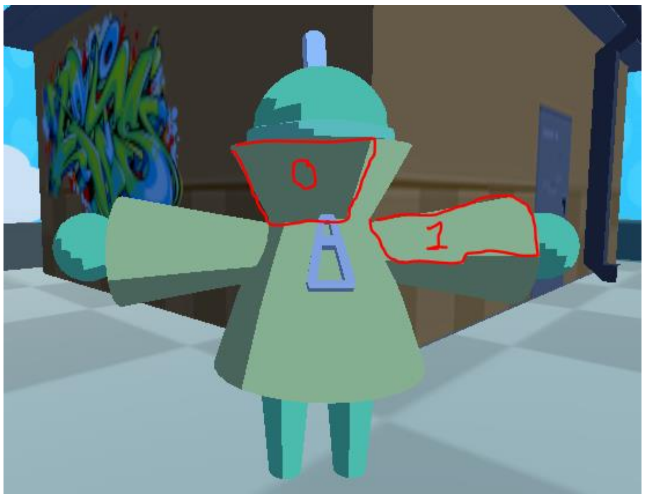


Figure 4.11: A labelled screenshot showing the cel shader working in game with the dark shaded areas being calculated with the dot product of 0 and the light areas being calculated by rounding up any dot product value greater than 0.

The outline shader makes use of post processing methods to sample the image taken from the game's camera and use it for data processing. This shader is fundamentally different to the cel shader that we just covered as it is a *Screen-Space Shader,* which means that rather than working on the material itself, the shader works on a render texture taken from the camera and is altered every frame, in real time. The outline shader works by sampling the depth texture taken from the camera and comparing adjacent pixels in said depth texture. Depth textures work by giving each pixel on the screen a numerical value between 0 and 1 inclusive, with a 1 meaning that the pixel data is close to the camera and 0 meaning it is far from the camera. In this set of encoded pixels, if the values of two adjacent pixels vary drastically, we will draw an edge, creating the desired outline effect.

This does, however, create some undesired artefacts in certain situations. On lower poly models, you may see that the outline shader draws a line on the interior faces of a model. This is because on either side of the outline there are two pixels with differing values in the depth texture. In this sense, the outline shader is technically working – it has detected a hard edge and drawn a line along it, however, as mentioned before it can lead to undesired outcomes as seen in the figure below.

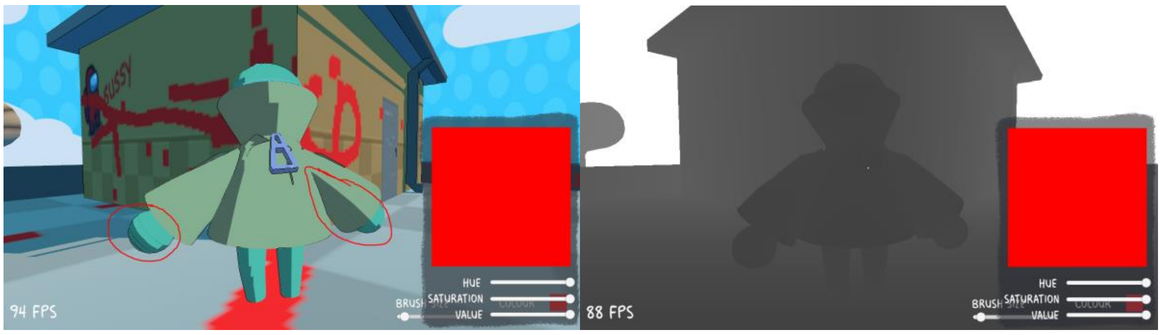
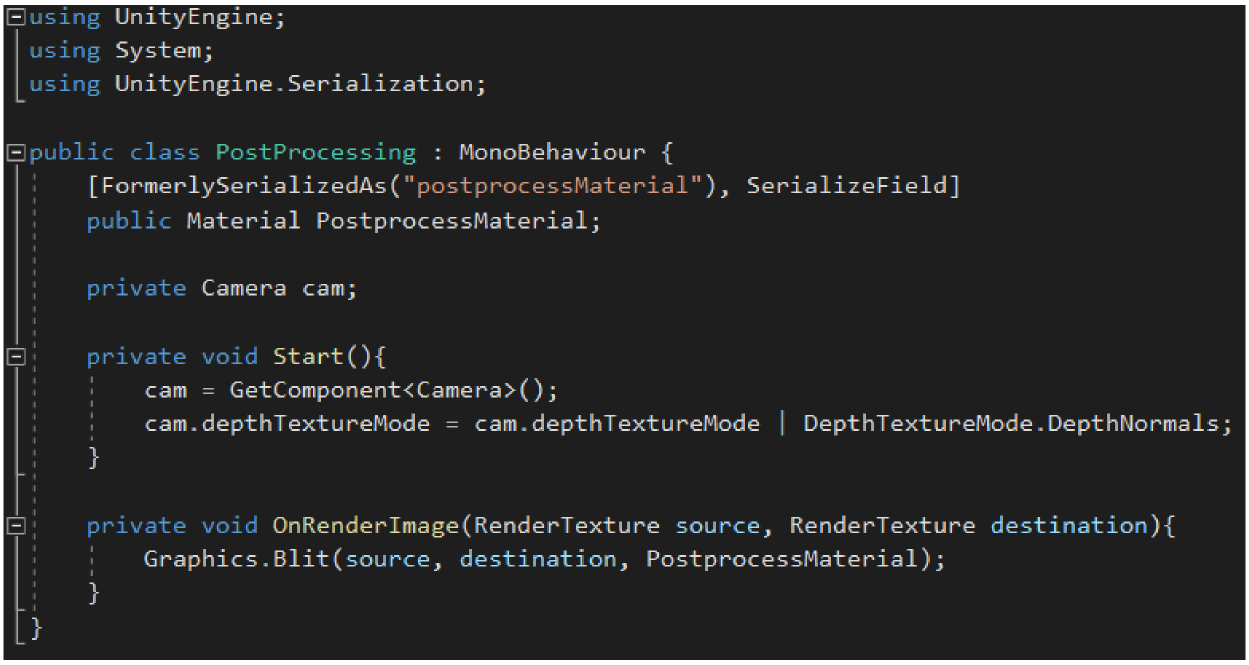


Figure 4.12.1 and 4.12.2: A screenshot of the outline shader in effect – note how there are faint outlines present in the circled areas due to the changes in the depth texture shown in figure 4.12.2.

This information is given to the shader with a render texture. Attached to the camera is a material which contains the following script:



```
using UnityEngine;
using System;
using UnityEngine.Serialization;

public class PostProcessing : MonoBehaviour {
    [FormerlySerializedAs("postprocessMaterial"), SerializeField]
    public Material PostprocessMaterial;

    private Camera cam;

    private void Start(){
        cam = GetComponent<Camera>();
        cam.depthTextureMode = cam.depthTextureMode | DepthTextureMode.DepthNormals;
    }

    private void OnRenderImage(RenderTexture source, RenderTexture destination){
        Graphics.Blit(source, destination, PostprocessMaterial);
    }
}
```

Figure 4.13: The postprocessing script applied to the camera gameobject.

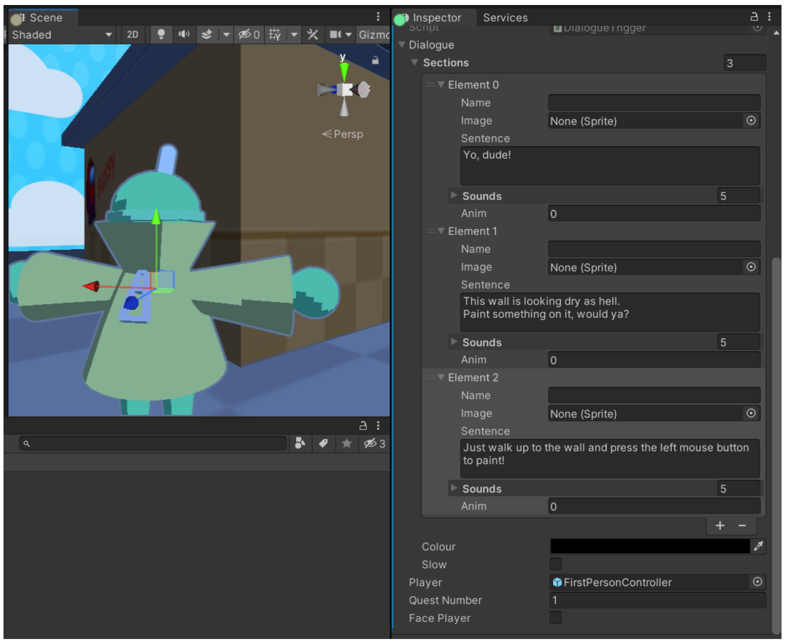
In the start function, the script gets the attached camera object and requests that it renders both a *DepthNormals* texture – this is the information we are using to create our outline as seen in figure 5.10. Then, using the *OnRenderImage* method, a method that is called after the camera has finished rendering a frame, we use the *Graphics.Blit* function which will take the depth texture taken from the camera and draw the pixels onto the post processing material that we have attached. This material then has our outline shader applied and the resulting render texture is output onto the camera.

It is worth noting that the code for the outline shader was influenced by independent research into resources such as "*The Unity Shaders Bible*" [17] and created by adapting code from online tutorials[18] and as such will share some similarities.

4.4 Dialogue System

To give the characters in my game some more life, it was decided to create a dialogue system to display text and dialogue onto the screen, since Unity does not provide one natively. The dialogue system works by having a main *DialogueManager* class – the class that controls the output of text onto the screen. This DialogueManager class contains information for all the associated elements to control – the text element to edit in real time, the animator of the associated textbox, the audio component to play the character's voice from and so on.

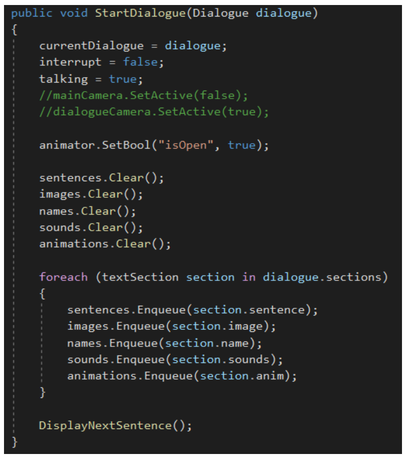
Each gameobject in the scene with associated dialogue contains a script *DialogueTrigger* which contains the dialogue to be displayed onto the screen. You can see in the figure below that the lollipop character has three sections of dialogue to feed into the DialogueManager class.



```
Script            DialogueTrigger
Dialogue
  Sections                     3
    Element 0
      Name
      Image       None (Sprite)
      Sentence
        Yo, dude!
      Sounds                   5
      Anim        0
    Element 1
      Name
      Image       None (Sprite)
      Sentence
        This wall is looking dry as hell.
        Paint something on it, would ya?
      Sounds                   5
      Anim        0
    Element 2
      Name
      Image       None (Sprite)
      Sentence
        Just walk up to the wall and press the left mouse button
        to paint!
      Sounds                   5
      Anim        0
  Colour
  Slow
Player            FirstPersonController
Quest Number      1
Face Player
```

Figure 4.14: The DialogueTrigger script for the lollipop character shown in the inspector.

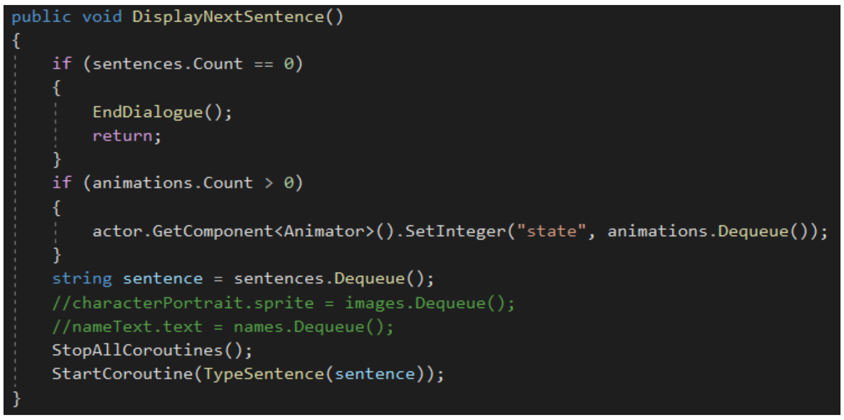
In this array of dialogue sections, you will notice that they each contain information such as a name, image, text, sounds and animation. Each of these sections are part of their own struct called *TextSection*, which is in turn fed into a class called *Dialogue.* This is done as it allows us to create easily editable dialogue sections as seen in figure 5.14.
When the player is within an interactable range of the character, going up to the character and pressing Q will trigger the DialogueManager's *StartDialogue()* function. This function sets up the dialogue to be read and takes each element of the TextSection struct and enqueues it into an array.



```
public void StartDialogue(Dialogue dialogue)
{
    currentDialogue = dialogue;
    interrupt = false;
    talking = true;
    //mainCamera.SetActive(false);
    //dialogueCamera.SetActive(true);

    animator.SetBool("isOpen", true);

    sentences.Clear();
    images.Clear();
    names.Clear();
    sounds.Clear();
    animations.Clear();

    foreach (textSection section in dialogue.sections)
    {
        sentences.Enqueue(section.sentence);
        images.Enqueue(section.image);
        names.Enqueue(section.name);
        sounds.Enqueue(section.sounds);
        animations.Enqueue(section.anim);
    }

    DisplayNextSentence();
}
```

Figure 4.15: The StartDialogue() function.

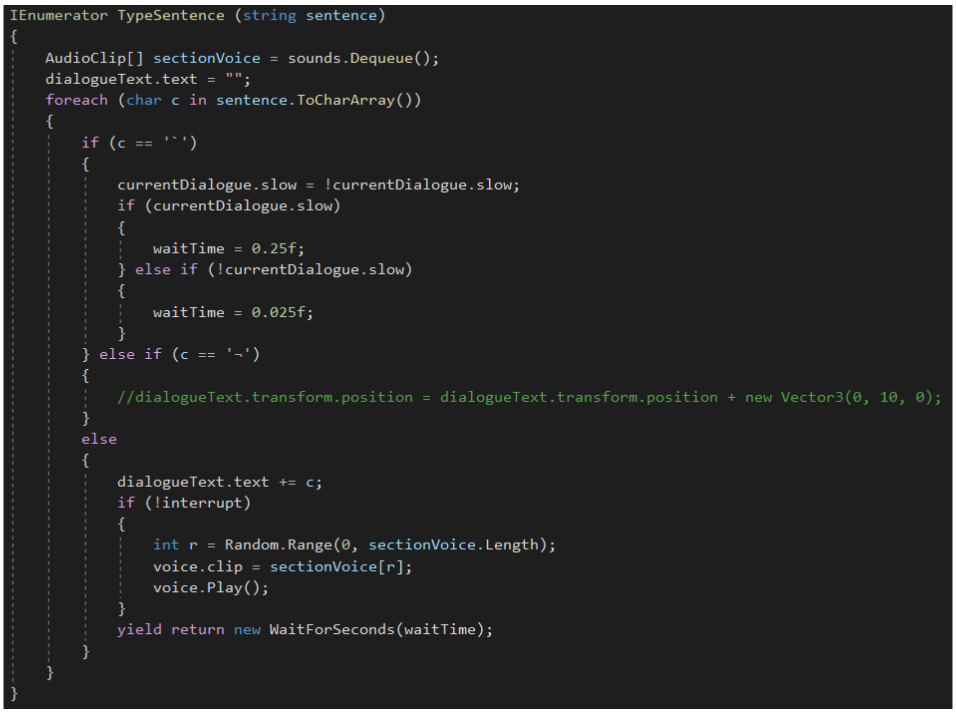
Once this is done, the manager will loop through the *DisplayNextSentence()* function – a function that dequeues the next sentence to be read or ends the dialogue if the queue is empty.



```
public void DisplayNextSentence()
{
    if (sentences.Count == 0)
    {
        EndDialogue();
        return;
    }
    if (animations.Count > 0)
    {
        actor.GetComponent<Animator>().SetInteger("state", animations.Dequeue());
    }
    string sentence = sentences.Dequeue();
    //characterPortrait.sprite = images.Dequeue();
    //nameText.text = names.Dequeue();
    StopAllCoroutines();
    StartCoroutine(TypeSentence(sentence));
}
```

Figure 4.16: The DisplayNextSentence() function.

Once the function has a sentence to display, it will call the *TypeSentence* coroutine. This function is where all the heavy lifting is done, the function now dequeues the next audio clip array (the associated voice with the current text section) and the current text section into a character array. Then for each character in said array, the current character will be added to the string displayed onto the screen and one of the given audio clips will be played at random as the text is displayed onto the screen, creating the illusion of speech. Once the character has been written to the screen, the coroutine waits for the designated amount of time and repeats for the next character in the array. This process is repeated until the dialogue queue is empty and DisplayNextSentence() calls the EndDialogue() function instead.

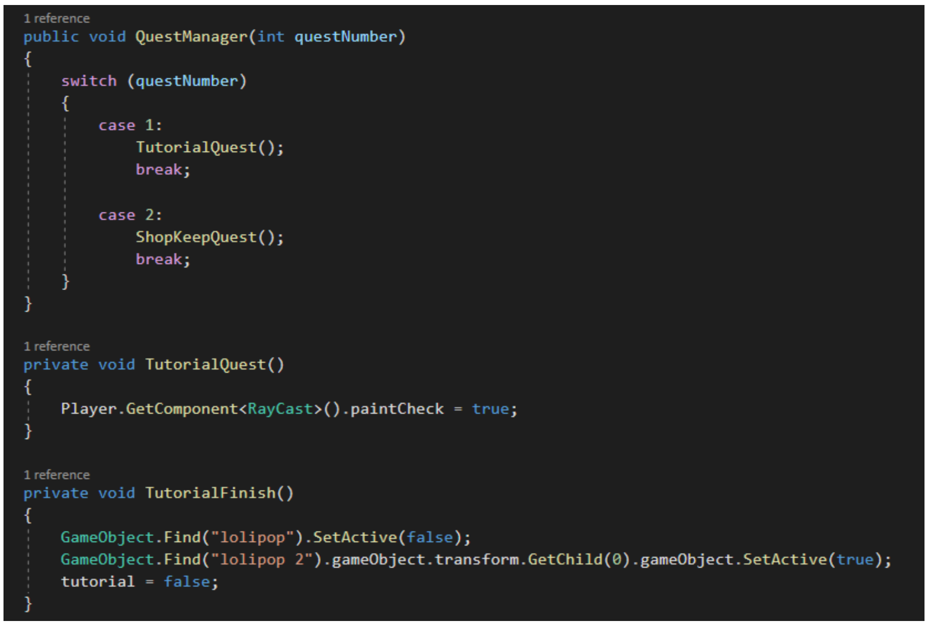


```
IEnumerator TypeSentence (string sentence)
{
    AudioClip[] sectionVoice = sounds.Dequeue();
    dialogueText.text = "";
    foreach (char c in sentence.ToCharArray())
    {
        if (c == '`')
        {
            currentDialogue.slow = !currentDialogue.slow;
            if (currentDialogue.slow)
            {
                waitTime = 0.25f;
            } else if (!currentDialogue.slow)
            {
                waitTime = 0.025f;
            }
        } else if (c == '¬')
        {
            //dialogueText.transform.position = dialogueText.transform.position + new Vector3(0, 10, 0);
        }
        else
        {
            dialogueText.text += c;
            if (!interrupt)
            {
                int r = Random.Range(0, sectionVoice.Length);
                voice.clip = sectionVoice[r];
                voice.Play();
            }
            yield return new WaitForSeconds(waitTime);
        }
    }
}
```

Figure 4.17: The TypeSentence() coroutine.

4.5 Gameplay

The project’s gameplay consists of primarily of quests which utilise a flagging system to track the player’s progress. This is done by using an external quest handler object, stored in the world space, and marked with a do not destroy on load tag so that it persists between scenes. This quest handler consists of Boolean data for each quest which is used to run a corresponding function that manages different aspects of the scene. Each quest is given an associated integer value starting from one as zero is used for NPCs that have dialogue but do not have a linked quest. This integer value is set in the DialogueManager for each NPC under the “Quest Number” value.

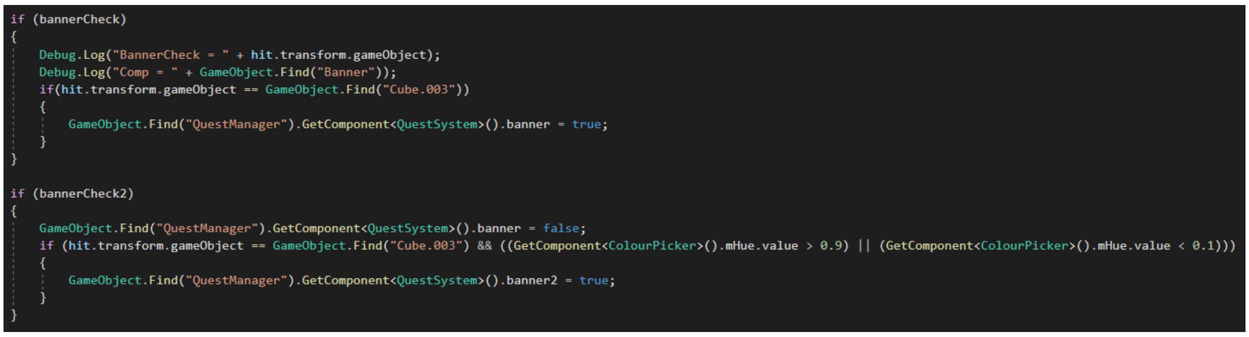


```
1 reference
public void QuestManager(int questNumber)
{
    switch (questNumber)
    {
        case 1:
            TutorialQuest();
            break;

        case 2:
            ShopKeepQuest();
            break;
    }
}

1 reference
private void TutorialQuest()
{
    Player.GetComponent<RayCast>().paintCheck = true;
}

1 reference
private void TutorialFinish()
{
    GameObject.Find("lolipop").SetActive(false);
    GameObject.Find("lolipop 2").gameObject.transform.GetChild(0).gameObject.SetActive(true);
    tutorial = false;
}
```

Figure 4.18: A section from the QuestManager script, showcasing how quests are managed.

Quests work primarily by checking for changes in world data. Once a specific flag has been activated, it will check to see if an associated action has been completed. For example, in the second quest, given by the red NPC, he will ask the player to paint something onto his shop’s banner. Once this dialogue has been triggered, the quest manager will set a Boolean value for the associated quest (in this case “bannerCheck”) to be true, enabling code in the painting script to check if it is painting to a specific object.

There are other elements which can be checked in order to run this quest structure, such as in the second part of the quest, where the NPC asks the player to add some red to their banner design. In this instance, the check code compares to see if it is currently editing the UV of the banner object and then checks to see if the player’s selected colour is within a specific range. In this example, the player’s selected hue value must be either greater than 0.9 or less than 0.1.



```
if (bannerCheck)
{
    Debug.Log("BannerCheck = " + hit.transform.gameObject);
    Debug.Log("Comp = " + GameObject.Find("Banner"));
    if(hit.transform.gameObject == GameObject.Find("Cube.003"))
    {
        GameObject.Find("QuestManager").GetComponent<QuestSystem>().banner = true;
    }
}

if (bannerCheck2)
{
    GameObject.Find("QuestManager").GetComponent<QuestSystem>().banner = false;
    if (hit.transform.gameObject == GameObject.Find("Cube.003") && ((GetComponent<ColourPicker>().mHue.value > 0.9) || (GetComponent<ColourPicker>().mHue.value < 0.1)))
    {
        GameObject.Find("QuestManager").GetComponent<QuestSystem>().banner2 = true;
    }
}
```

Figure 4.19: *The logic behind the banner quest.*

To handle providing different dialogue to the player depending on what section of the quest the player is on, once the current objective has been met, the NPC is set to be inactive and a duplicate NPC with a different set of dialogue is activated instead.

The other main gameplay areas are allowing the player to paint freely on any surface and explore an open environment with hidden puzzles. One such example of this is an invisible platform that the player must cover in paint to be able to see and traverse over. This effect is achieved by creating a transparent UV map for the object and setting the material to have a “cutout” rendering mode. This is a type of rendering included in Unity’s default shader which renders textures’ transparency accurately. It takes the texture’s alpha channel and uses it to inform the output of the texture, where an alpha value of 0 indicates a pixel on the material should be rendered completely transparent and a value of 255 indicates that the pixel should be rendered completely opaque.

We can use this rendering technique to create textures with mixed levels of transparency, which helps us to make textures for objects such as windows, however, in this scenario it is used to create an invisible platform which can be revealed to the player by changing the pixel values of the texture to be opaque.

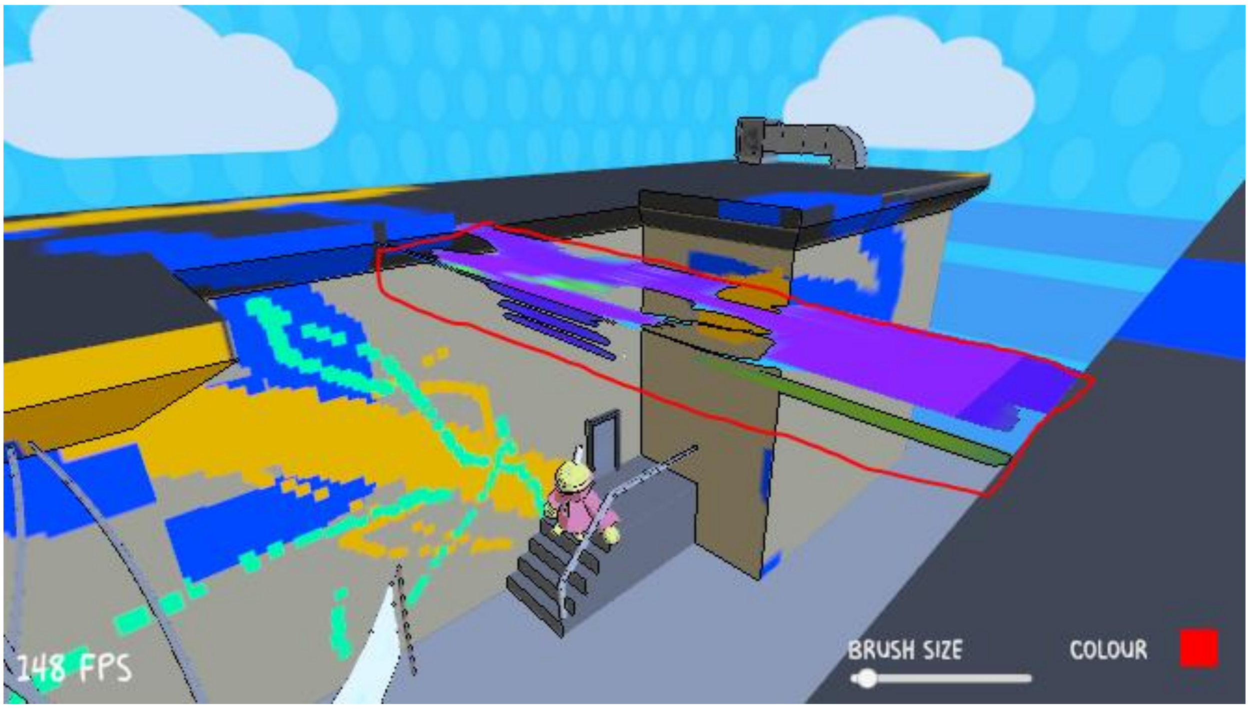


Figure 4.20: *An invisible platform in the game, making use of the cutout rendering mode.*

In terms of the player being able to freely paint any surface, it was important to allow them to use any colours they wanted for this, and the decision was made to provide this functionality through an HSV (Hue, Saturation and Value) slider.

The following chapter will talk about the testing and iterative design process used in this project.

5.0 Testing

5.1 User testing

Feedback was conducted through a mixture of user testing and iterative development. The project was shown to potential users as it neared completion to get feedback on how the project could be improved. This form of development is called "Iterative Game Design"[19], a design methodology that allows the developer to ensure that they are creating a product which the player will enjoy, as to not have diminishing returns on profit output.

While this project is not one that is making a profit and therefore does not need to ensure that the game is reflective of the player's wants for financial reasons, this methodology was implemented as it helps to create an overall better product.

5.2 Outcomes of the testing process

After playing the game during testing, players were asked to answer the following questions:

- What did you enjoy about the game?
- Did you dislike anything specific about the game?
- Were there any aspects of the game you would like to see added or improved?

Feedback was taken for each of these questions and used to shape the project into its current state. Specific ways in which the feedback from these questions was implemented is as follows:

5.2.1 What did you enjoy about the game?

When asked about what they enjoyed about the game, players main response was to do with the novelty of the core painting mechanic itself. Players reported that the mechanic was a lot of fun to play with and that "*it wasn't something I've really seen before*", which indicates that there was good interest in the mechanic, coupled with comments that players thought there could be a lot of mileage in the mechanic and started suggesting mechanics of their own, such as different types of paint that allow the user to take on different properties, similar to that of the gels in *Portal 2*. This input was taken on board and used to help drive the creation of the invisible platform that can be painted to be seen, as well as luminous paint which was planned to be implemented through the use of texture emission [20] but did not make it into the current version.

Other feedback consisted of comments that the players enjoyed the art style a lot and that the cel shading gave the game a unique look that they hadn't seen a lot before. One person commented that it made the game look like a comic book and suggested that characters should have an outline, which was later implemented into the game with the outline shader.

5.2.2 Did you dislike anything about the game?

Players' main issue with the game was to do with the fact that certain aspects of the game and its mechanics were not immediately obvious to the player. For example, many players tried to walk up to the NPC and interact with them but did not manage to work out how to do so as it was not shown anywhere in the game. To combat this, a change was made to have an indicator over the top of the NPC telling the player what key they had to press to interact.

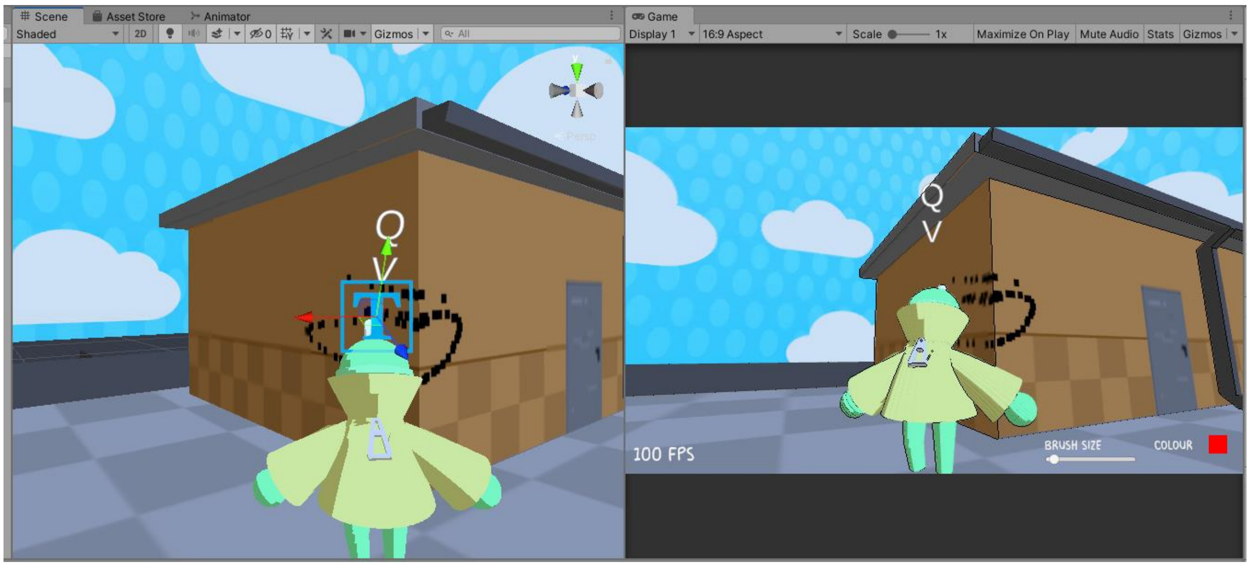


Figure 5.1: One of the NPCs with a button prompt hovering over their head, removing ambiguity on how to interact.

5.2.3 Were there any aspects about the game you would like to see added or improved?

The most common response to this question was simply that players wished that there was more content in the game - more quests and areas to explore as the current environment felt small and lacking in things to do. Players stated that they were interested in the world and the potential story that could be told with more characters and a bigger environment, as well as the potential for unique scenarios which could be executed should the project have more time and resources.

One such idea that arose for a unique way of interacting with the world was to have a character whose UVs are completely blank, and they offer the player a quest to paint them so that they look good to go on a date with another NPC. Once again, however, this is something that could not be added within the timeframe of the project.

One key piece of feedback from a user was that they found it difficult to know where they were aiming their paint in the game as it was in the centre of the screen but without a marker it can be difficult to pinpoint the exact centre of the screen. To rectify this issue, a minimalistic reticule was added to the screen, helping players aim.

This is a scenario which highlights the importance of user testing and iterative design as important quality of life features such as the aforementioned reticule may not have been added without input from the player as often times the developer can become overly familiar with their own project and may not notice when seemingly obvious issues arise.

6.0 Conclusion and Evaluation

This project was undertaken with the intent of creating a game that makes use of a novel idea and allows the player to experiment with drawing technologies in an attempt to create a game that has an interesting and unique gameplay loop. It is my belief that this intention was met to an appropriate standard.

All of the core requirements were met and implemented into the final solution, however, the majority of the optional requirements ended up not implemented due to time constraints. There were also certain requirements outlined in the early stages of development which ended up changing towards the end of development due to the iterative nature of the project's design. One such example was the ability to switch between a first and third-person perspective. The design of the project ended up different to how it was at the start and ended up being played entirely from the first person. Furthermore, the aspect of a modelled player character with animations was shifted over to a non-playable character as the model and animation work would rarely be seen, if at all in an entirely first-person game.

Certain elements of the project did not come to fruition due to complexity and time constraints. One such element was the implementation of a marching squares algorithm to create more realistic looking paint. There was an attempt to implement such functionality towards the start of the project, however, as the project progressed and the implementation of the painting mechanic became solidified to use the setpixels() method, it became clear that implementing such a feature would not be possible due to the limitations of this methodology. What exists in the project now works well and serves its purpose, but should development continue outside of the bounds of the university submission, more research into methodologies to implement such a feature would be conducted.

Development of the project overall ran smoothly and encountered few obstacles, mainly due to the fact that development had gotten a head start in the month of October. There was, however, a large amount of time spent attempting to figure out and optimise the painting code as the setpixels() method wasn't something that ended up implemented into the project until its final month of development, however, this turned out to be vital to the overall cleanliness of the code as implementing the mechanic in this way simplified the core functionality from $O(n^2)$ down to $O(n)$, drastically increasing the game's performance and fixing issues and complaints discovered in user testing.

It is my intention to continue to work on this project once I graduate and see it completed with more content and complexity. In terms of further work, I believe the game will need to have more content, with a larger overworld and a larger cast of non-playable characters for the user to interact with as well as more varied gameplay outside of the main painting mechanic. There has been a good start to exploration of the environment in a creative manner through the example of the invisible platform that the player has to reveal, however, I intend to implement mechanics such as luminous paint that the player can put down to light up dark and hard to see areas. There is a lot that can be built upon from the core painting mechanic, but it is essential to implement different mechanics and mini games to keep the gameplay loop fresh and exciting, so the player does not grow bored.

In section 1.3.2, the design problems with the project were discussed and intentions were stated. Specifically, the intention of achieving the *flow state*, popularised by the work of psychologist Mihaly Csikszentmihalyi. From user feedback and testing, I would like to believe that this was in some way or form achieved. It was observed in testing that players would often find themselves getting briefly distracted from the objectives given to them in order to explore the environment and paint onto the environment, particularly when previous players had left drawings of their own. Due to the nature of the project, the player is more likely to find themself in the area of boredom, as the game is not particularly challenging in its current state. Going forward with development, it will be important to implement more challenge into the game that the player can seek out on their own terms in order to keep them in this flow state.

7.0 Bibliography

[1] - *Chicory: A Colorful Tale on Steam* (2021) *Store.steampowered.com*. Available at: https://store.steampowered.com/app/1123450/Chicory_A_Colorful_Tale/ (Accessed: October 20, 2022).

[2] - Schell, J. (2020) "What Skills Does a Game Designer Need? - Chapter 1, Page 3," in *The art of game design: a book of lenses*. CRC Press.

[3] - Swink, S. (2017) "'The Three Building Blocks of Game Feel' - Chapter One: Defining Game Feel," in *Game feel: A game designer's Guide to Virtual Sensation*. CRC Press, Taylor & Francis Group.

[4] - Csikszentmihalyi, M. (2009) *Flow: The psychology of optimal experience*. Harper and Row.

[5] – Greg Lobanov 30, 2019 (2019) *How painting works in Chicory: A Colorful Tale*, *Game Developer*. Available at: https://www.gamedeveloper.com/design/how-painting-works-in-chicory-a-colorful-tale (Accessed: October 18, 2022).

[6] - *Marching squares* (2022) *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Marching_squares (Accessed: October 24, 2022).

[7] - Technologies, U. *Texture2D.SetPixels*, *Unity Documentation*. Unity Technologies. Available at: https://docs.unity3d.com/ScriptReference/Texture2D.SetPixels.html (Accessed: October 5, 2022).

[8] - Technologies, U. *Compute shaders, Unity Documentation*. Unity. Available at: https://docs.unity3d.com/Manual/class-ComputeShader.html (Accessed: November 2, 2022).

[9] - *Suchart: Genius artist simulator on steam* (2022) *SuchArt: Genius Artist Simulator on Steam*. Available at: https://store.steampowered.com/app/1293180/SuchArt_Genius_Artist_Simulator/ (Accessed: October 20, 2022).

[10] - *Jet Set Radio on Steam* (2022) *Store.steampowered.com*. Available at: https://store.steampowered.com/app/205950/Jet_Set_Radio/ (Accessed: October 23, 2022).

[11] - Foundation, B. *Home of the blender project - free and open 3D creation software*, *blender.org*. Available at: https://www.blender.org/ (Accessed: March 8, 2023).

[12] - Foundation, B. *Animation & Rigging*, *blender.org*. Available at: https://www.blender.org/features/animation/ (Accessed: March 8, 2023).

[13] – Foundation, B. *Weight Painting - Blender Manual*. Available at: https://docs.blender.org/manual/en/latest/sculpt_paint/weight_paint/introduction.html (Accessed: March 8, 2023).

[14] - *UV mapping* (2022) *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/UV_mapping (Accessed: March 8, 2023).

[15] - Technologies, U. *Custom Shader Fundamentals*, *Unity*. Available at: https://docs.unity3d.com/Manual/SL-VertexFragmentShaderExamples.html (Accessed: February 22, 2023).

[16] - *Blinn–Phong reflection model* (2023) *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_reflection_model (Accessed: February 22, 2023).

[17] - Espindola, F. (2022) *The unity shaders bible: A linear explanation of shaders from beginner to advanced*. Santiago, Chile: Jettelly.

[18] - *Outlines via postprocessing* (2018) *Ronja's tutorials*. Available at: https://www.ronja-tutorials.com/post/019-postprocessing-outlines/ (Accessed: February 25, 2023).

[19] - Will Luton Blogger October 15, 2009 (2009) *Making Better Games through iteration*, *Game Developer*. Available at: https://www.gamedeveloper.com/design/making-better-games-through-iteration#close-modal (Accessed: March 1, 2023).

[20] - Technologies, U. *Emission*, *Unity*. Available at: https://docs.unity3d.com/Manual/StandardShaderMaterialParameterEmission.html

8.0 Appendices

8.1 BCS Code of Conduct

BCS CODE OF CONDUCT

This Code of Conduct:

• sets out the professional standards required by BCS as a condition of membership.

• applies to all members, irrespective of their membership grade, the role they fulfil, or the jurisdiction where they are employed or discharge their contractual obligations.

• governs the conduct of the individual, not the nature of the business or ethics of any Relevant Authority*.

1. Public Interest

You shall:

a. have due regard for public health, privacy, security and wellbeing of others and the environment.

b. have due regard for the legitimate rights of Third Parties.

c. conduct your professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement.

d. promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise.

2. Professional Competence and Integrity

You shall:

a. only undertake to do work or provide a service that is within your professional competence.

b. **NOT** claim any level of competence that you do not possess.

c. develop your professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to your field.

d. ensure that you have the knowledge and understanding of Legislation* and that you comply with such Legislation, in carrying out your professional responsibilities.

e. respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work.

f. avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction.

g. reject and will not make any offer of bribery or unethical inducement.

3. Duty to Relevant Authority

You shall:

a. carry out your professional responsibilities with due care and diligence in accordance with the Relevant Authority's requirements whilst exercising your professional judgement at all times.

b. seek to avoid any situation that may give rise to a conflict of interest between you and your Relevant Authority.

c. accept professional responsibility for your work and for the work of colleagues who are defined in a given context as working under your supervision.

d. NOT disclose or authorise to be disclosed, or use for personal gain, or to benefit a third party, confidential information except with the permission of your Relevant

Confidential to BCS Members

Trustee Board Regulations Schedule 3 v8 Code of Conduct for BCS Members Page 3 of 5

Reviewed by Trustee Board 8 June 2022

Authority, or as required by Legislation.

e. NOT misrepresent or withhold information on the performance of products, systems or services (unless lawfully bound by a duty of confidentiality not to disclose such information), or take advantage of the lack of relevant knowledge or inexperience of others.

4. Duty to the Profession

You shall:

a. accept your personal duty to uphold the reputation of the profession and not take any action which could bring the profession into disrepute.

b. seek to improve professional standards through participation in their development, use and enforcement.

c. uphold the reputation and good standing of BCS, the Chartered Institute for IT.

d. act with integrity and respect in your professional relationships with all members of BCS and with members of other professions with whom you work in a professional capacity.

e. encourage and support fellow members in their professional development.

8.2 Project Proposal

Use of drawing technology in gaming to create an immersive and creative experience

By Jamie Darbon

Candidate Number 218721

Supervised by Paul Newbury

Aims

Drawing technology in games has been used to create some of the most creative and inventive experiences currently available. Due to the nature of the medium, there is a lot of versatility in the concept as well – in games such as “Drawn to Life”, the player gets to draw their own character and world from scratch that can be interacted with in the game space. In “Chicory: A Colourful Tale”, the player can use the drawing technology to interact with the pre-existing world and use their paint to customise it to their desire as well as traverse the world in interesting and unique ways, only achievable with the use of the drawing technology.

The best use of this type of technology, I believe, is to use it in order to create an interesting and engaging gameplay experience with rich mechanics and depth first and then secondarily (but no less importantly) focus on the drawing technology itself and the customisation which this creates. It is important to make this distinction early on so that by the end of the development cycle you have an immersive and engaging game rather than an in-depth piece of drawing software. This can be seen in games which explore this concept and their review scores, for example, “Chicory: A Colourful Tale”, a game which explores the concept of using drawing technology to interact with the game world and whose gameplay is built around the concept, is currently sitting on a 97% positive review score on Steam and a score of 90/100 on Metacritic. *Jet Set Radio* explores a similar idea with more limited technology from the year 2000, applying decals onto game objects with alpha transparency to create the illusion of graffiti. In this game, the drawing is achieved by motion inputs and is a secondary mechanic, complementing the main gameplay loop.

I believe the reason this is the case is because Chicory uses the drawing mechanic to complement both its game and its story – the paints used on the game world can be used to light up dark caves and swim through water and in a story context, the painting theme is used to explore deep emotional areas that are expressed through the characters’ art. This creates a rich and complex experience that is relevant to the game’s mechanics and uses them to enhance its storytelling. This project will attempt to replicate the aforementioned mechanical depth and creativity to create an immersive experience.

Objectives

- Conduct research into the existing solutions and market space for games that use digital drawing technology to enhance their gameplay.

- Develop a prototype for the main gameplay functionality that allows the player to draw freely onto the in-game environment.
    - This should allow the player to change the size and colour of the brush used, as well as the shape and brush texture.
- Use the prototype to further develop gameplay mechanics that branch off the painting mechanic, allowing for versatility in level design and puzzles using different types of paints.
- Develop a test plan and perform user research to inform design decisions while getting feedback about what does and doesn't work with the game.
- Create and rig a fully modelled 3D character that the player can control and use to interact with the game world.

Relevance

My project is relevant to my degree by its nature – as I am working to complete a game development degree, it makes sense that I use the skills which I have learned over the last couple of years to make a game for my final year project. It will take elements from a large amount of the modules which I have studied in my time as an undergraduate, such as programming (from various modules), 3D modelling, user testing and research, development within a game engine among others. I will be using an agile development strategy, as I believe it best fits the nature of this type of software and there will be research into how a human interacts with a computer system in the context of digital drawing technologies.

Personally, I have intended to go into the games industry for as long as I can remember, so having a game project that could showcase mechanical complexity to put into my portfolio is something that is very relevant to me. Further, I have been interested in art for many years, doing a lot of drawing and even making it my main source of income for a few years – the subject matter is something I am very interested in and the concept that I am trying to achieve combines two of my passions into one larger product, providing me with a perfect mix of a project that brings together my personal interests with my technological capabilities, asking me to explore areas such as marching squares to create something that could really help me out in the future as I prepare to enter the working world.

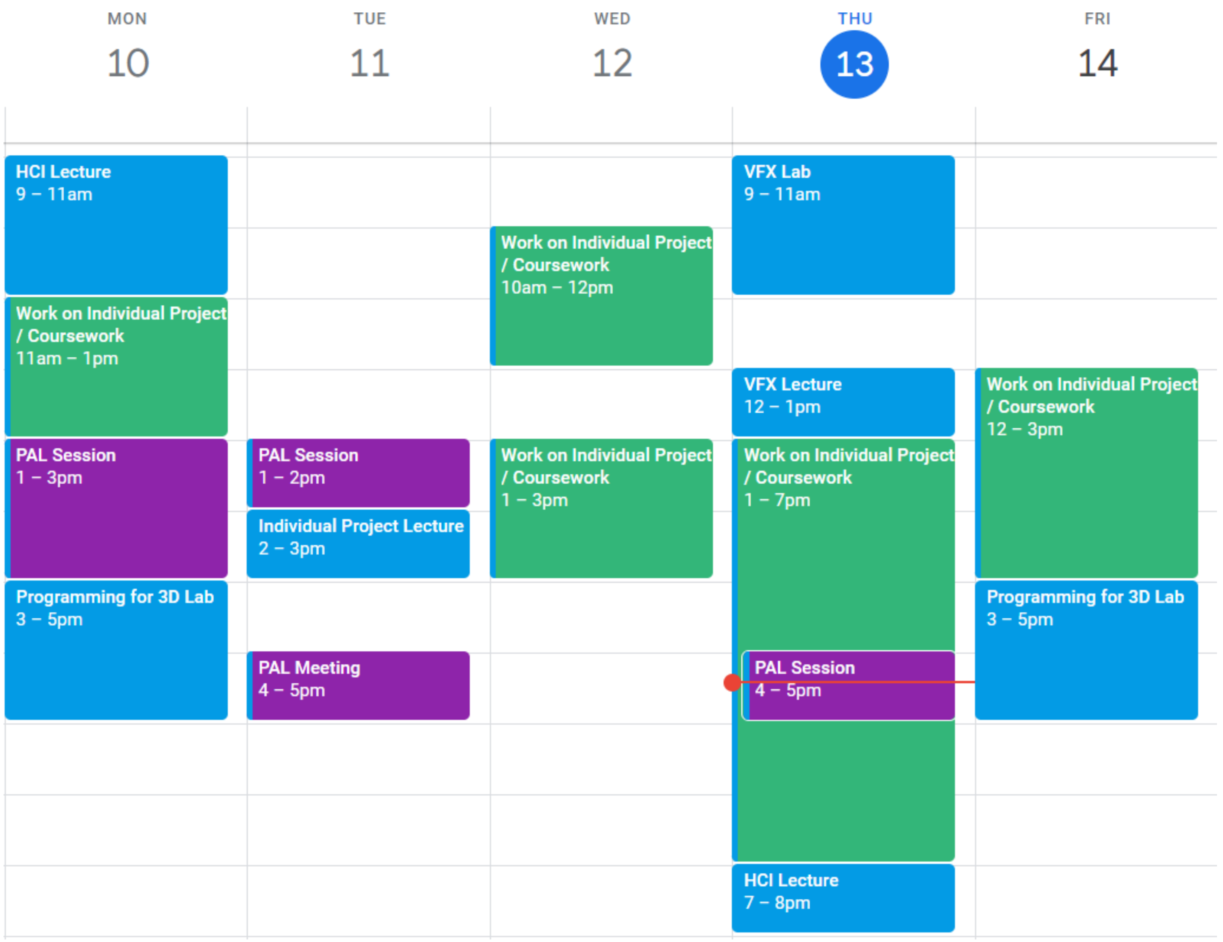
Resources Required

I will be making this game in the Unity engine. I have chosen to do so after doing research game engines and come to that conclusion for a few reasons. First off, the game will be in 3D – *Chicory: A Colourful Tale*, the main inspiration behind the game was made in Game Maker Studio 2, which I initially looked into to try and see if there was any reason for me to follow suite, however, after making the decision that my project would be in 3D, which Chicory is *not*, I decided that either Unity or Unreal would be appropriate choices for it. I chose not to use Unreal mainly since the effect I can achieve is facilitated very well in Unity using *render textures*, something that Unreal does not have (at least in my research). A render texture is a type of texture that Unity updates and renders at run time, this makes it very easy to create textures that can be edited through player interaction and hence informed my choice of engine.

The software I will be using is either free or licensed to the university (Unity, Blender, Photoshop, etc), so there will be no need to purchase specialist software.

Weekly Schedule

Below you will find the weekly timetable that I will be using to complete this project, considering I am on campus a lot this year, I have set aside time to work on the project in the labs, although as the term progresses and coursework starts to get released, I will likely have to juggle this time with other responsibilities, in which case I have weekends free to take time from.


MON 10
TUE 11
WED 12
THU 13
FRI 14
HCI Lecture 9 – 11am
Work on Individual Project / Coursework 11am – 1pm
PAL Session 1 – 3pm
Programming for 3D Lab 3 – 5pm
PAL Session 1 – 2pm
Individual Project Lecture 2 – 3pm
PAL Meeting 4 – 5pm
Work on Individual Project / Coursework 10am – 12pm
Work on Individual Project / Coursework 1 – 3pm
VFX Lab 9 – 11am
VFX Lecture 12 – 1pm
Work on Individual Project / Coursework 1 – 7pm
PAL Session 4 – 5pm
HCI Lecture 7 – 8pm
Work on Individual Project / Coursework 12 – 3pm
Programming for 3D Lab 3 – 5pm

Bibliography

- Game Developer. 2019. *How Painting Works In Chicory: A Colorful Tale*. [online] Available at: <https://www.gamedeveloper.com/design/how-painting-works-in-chicory-a-colorful-tale> [Accessed 12 October 2022].
- Docs.unity3d.com. 2022. *Unity - Scripting API: Texture2D.SetPixels*. [online] Available at: <https://docs.unity3d.com/ScriptReference/Texture2D.SetPixels.html> [Accessed 7 October 2022].
- Store.steampowered.com. 2022. *Chicory: A Colorful Tale on Steam*. [online] Available at: https://store.steampowered.com/app/1123450/Chicory_A_Colorful_Tale/
- Store.steampowered.com. 2022. *Kingspray Graffiti VR on Steam*. [online] Available at: https://store.steampowered.com/app/471660/Kingspray_Graffiti_VR/
- Store.steampowered.com. 2022. *Jet Set Radio on Steam*. [online] Available at: https://store.steampowered.com/app/205950/Jet_Set_Radio/
- En.wikipedia.org. 2022. *Marching squares - Wikipedia*. [online] Available at: https://en.wikipedia.org/wiki/Marching_squares

- Greg Lobanov (Lead Developer of Chicory: A Colorful Tale). 2022, *personal email enquiry* [online] [Accessed 13 October 22]

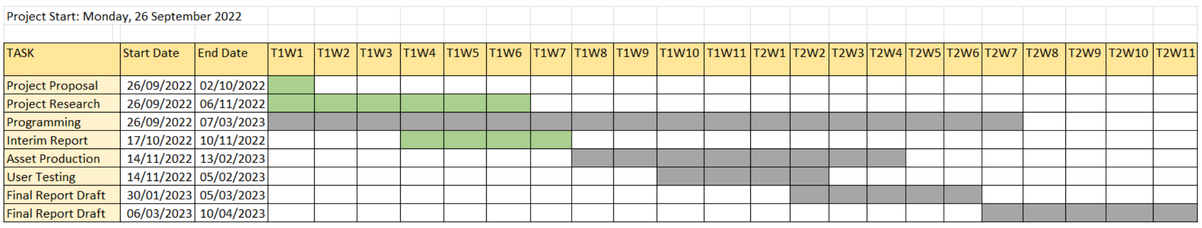
8.3 Project Plan

8.4 Gantt Chart

I have created a Gantt Chart that will be followed through development as closely as possible. Game development is not consistent and as such, it can be hard to accurately predict what will happen and when, however, this combined with the outline of the desired functionality showcased in chapter 2.2.2 will be followed as closely as possible. There are multiple elements of development which will be worked on in tandem, rather than one at a time and as such, may cause an impact on the delivery date of other sections and tasks.

In some cases, the elements in the Gantt Chart are generally vaguer and can be seen as blanket terms. This is the case with programming and asset creation – these elements consist of many smaller elements (as outlined in chapter 2.2.2) which can be tackled in any order.

The Gantt Chart also does not include the Christmas break, over which the project will continue to be worked on. I intend to complete the project itself by week 8 of the second semester, leaving ample time and resources to write the final report as this is a critical aspect of the final deliverable.



Project Start: Monday, 26 September 2022

TASK	Start Date	End Date	T1W1	T1W2	T1W3	T1W4	T1W5	T1W6	T1W7	T1W8	T1W9	T1W10	T1W11	T2W1	T2W2	T2W3	T2W4	T2W5	T2W6	T2W7	T2W8	T2W9	T2W10	T2W11
Project Proposal	26/09/2022	02/10/2022	🟩																					
Project Research	26/09/2022	06/11/2022	🟩	🟩	🟩	🟩	🟩	🟩																
Programming	26/09/2022	07/03/2023	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
Interim Report	17/10/2022	10/11/2022				🟩	🟩	🟩	🟩															
Asset Production	14/11/2022	13/02/2023								▨	▨	▨	▨	▨	▨	▨	▨							
User Testing	14/11/2022	05/02/2023										▨	▨	▨	▨									
Final Report Draft	30/01/2023	05/03/2023													▨	▨	▨	▨	▨					
Final Report Draft	06/03/2023	10/04/2023																		▨	▨	▨	▨	▨

Figure 8.1: Project Gannt Chart.

8.5 Project Tasks

8.5.1 Project Proposal – A document describing the project to be sent off to a supervisor for review

8.5.2 Project Research – Background research on existing solutions and examples to the project, outlined in chapter 2.1

8.5.3 Programming – Core programming for the project to be done in Unity with C#, highly important and critical to the project's completion

8.5.4 Interim Report – Report detailing the status of the project as of week 7, semester 1

8.5.5 Asset Production – Creation of assets to be used in the game project, ranging from UI to 3D models to audio and sound design

8.5.6 User Testing – Testing to be performed during development in order to gather feedback and use it to inform the game's production

8.5.7 Final Report Draft – First draft of the final report

8.5.8 Final Report – Included in the final deliverable, a report detailing the development of the finished product from week 1 of the first semester to the end of the submission period

As seen in the Gantt Chart, these tasks will run in parallel for the sake of efficiency, with the aim of completing them all for week 8 of the second semester so that all efforts can be focussed on the final report thereon. This has its benefits, especially during the user testing period as it will allow for feedback to be gathered and implemented within a short time frame.

The programming is seen to be the most important and critical task of the project's development itself, outside of the final report. For this reason, it has the highest priority and will run for the longest duration. Within the final weeks of programming, there will be a focus on bug fixing and troubleshooting, refraining from adding new features so that the game may be left in a finished and completed state.

There are certain aspects of the game that cannot be completed without other prior aspects being completed first. One such aspect is asset creation, hence why it is due to start further into the project's lifespan than other tasks. For example, the character controller and movement should be completed before modelling and rigging a 3D character or creating a 3D environment for the player to explore, else they will not be able to be properly implemented. User testing should be completed before the final report draft is written as this will be included in the report for reference as an appendix.

8.6 Completed Work

Programming for the game has been started and has been making progress since week 1 of the first semester. Currently, the painting mechanic of the game is near completion, with some alterations needed to make it more efficient with larger brush sizes and further development needed to allow the player to change the paint colour and brush texture at runtime (these can currently be changed in the Unity editor; however, the player will not have access to this so they must be implemented into the game either through UI or through mappable controls). An example of the current functionality can be found in figure 8.2, shown below.

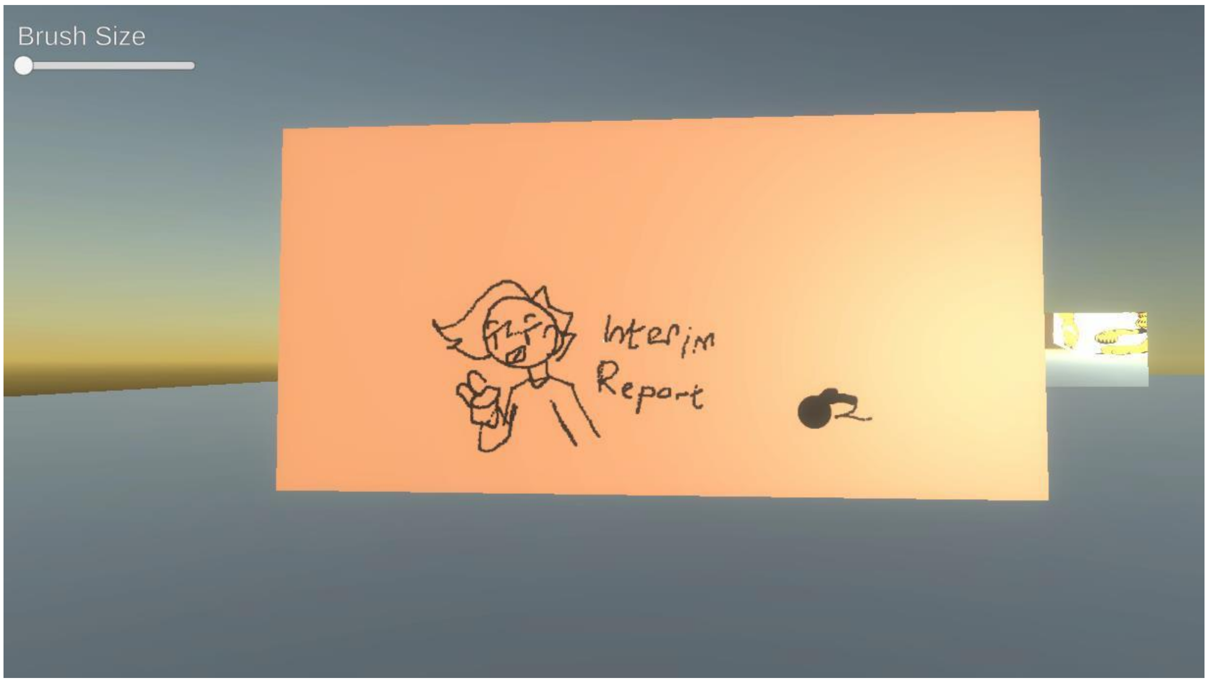


Figure 8.2: A screenshot from the project in its current state depicting a player-made drawing drawn onto the environment.

Basic player movement has also been created, allowing the player to walk, run and jump in the game environment. Progress has started on providing the player with UI feedback for their options, currently the brush size is displayed on the HUD, as shown in figure 8.2 and work has started on creating a colour picker and UI elements for displaying which brush type is currently in use.

There is work on implementing a marching squares based solution to the project in an effort to make the paint look more realistic rather than creating a brush that shows raw pixels on a texture.

Project research has been largely completed and I have had an email conversation with Greg Lobanov, creator of *Chicory: A Colorful Tale*, in which he advised me on how he went around making his painting mechanic and points me in the direction of further reading, however, there may be room for further research into specific game design for this type of project.

The project proposal can be found attached in the appendix and was completed in the first weeks of the first semester. The next chapter will discuss the design and implementation of the solutions in the final project.

8.7 List of software used:

- Unity was chosen as the game's main development engine, as well as the software used to create animations for the models created in this project.
- Blender was used to create models, their rigs, and UV maps.
- Photoshop and Paint Tool SAI were used to create UI elements, as well as the creation of assets such as the skybox, any materials seen in game and texturing UV maps.
- Visual Studio Code was used to create all code assets, including C# scripts and HLSL shader code.
- GitHub was used for version control.
    - Fork was used as a desktop client for GitHub.