Corona-Themed Zombie FPS Shooter Game

City, University of London BSc Computer Science

Jin Kato

[Jin.kato@city.ac.uk](mailto:Jin.kato@city.ac.uk)

07900157621

Consultant: Macfarlane Andrew

Project Proposal: Jin Kato

No arrangement for proprietary interests in the project.

992 words

Chapter 1: Introduction

**Problem To Be Solved**

According to Gov.uk, the official website for the British government, there have been around 3.7 million people who tested positive (1). The actual figure would be higher as 3.7 million is only for the people who were tested. Many feel stressed because of the deadly virus and the restriction caused by it. They also feel powerless since Corona is a virus, which cannot be reasoned with or have many ways to deal with except for social isolation, washing hands, and wearing masks. Playing video games will help the user relax and at the same time a Corona-themed game may help release the frustration towards the virus by destroying it in-game. Furthermore, there are more opportunity to play video games due to covid restrictions and social isolation causing most people to stay home.

**Project Objectives**

The main object of the project is to create a futuristic-themed fps game that could potentially help the user deal with the covid pandemic by destroying virus-looking enemies in game.

**Beneficiaries**

The beneficiaries of this software will be for the users who enjoys first-person-shooter video games and potentially people who are affected by Corona-virus pandemic. Destroying virtual viruses may help people who are affected release their anger and frustration, which may help them feel more in control of this uncertain time and refreshed.

**Motivation**

There are many ways programming can deal with the pandemic, such as using data science to track/predict the spread or creating an app/website that find recent updates about the virus. I decided to create a video game since I enjoy playing first person shooters and have long wanted to create a video game. Since I did not have too many opportunities to make games, except in the first year for the introductory Java module, the individual project module was a great opportunity to learn this new skill-set. The ability to learn a new skill-set is an important skill for a programmer due to the constantly changing and evolving nature of technology. This skill would help me become a better programmer and give me confidence that I will be adaptable to changes.

**Work Performed**

The project roughly consisted of the three phases, learning/planning, basic implementation (phase one), and refinement (phase two). In the learning/planning stage, I researched the most basic requirements for the game such as what platform and language I would use. Then I watched tutorials on Youtube and other online resources to learn and experiment on game development on Unity and C#. Based on this, I planned the most basic requirement for the game. In the basic implementation stage, the basic requirements were implemented and the requirements for the next stage was decided. Then in the refinement stage those requirements were implemented. In another word, the three stages I have mentioned could be thought as Pre-Alpha, Alpha, and Beta/Release Candiate.

Bellow is the rough list of the work performed for phase one, the basic implementation phase, and phase two, the refinement phase.

Phase One

Player World

-Creating a Plane

Implement Player In the Center With Shooting

-Creating Player

-Camera Movement For the Player

-Creating Bullet

-Shooting

Enemy

-Getting a Virus Like Sphere

-Making It Move Towards Player

Player and Enemy Interaction

-Player Loses Health When Hit By Virus

-Bullet Destroys Virus

-Create Multiple Enemies from the side and making them approach

UI

-Show Player Health

-Show Player Score

-Start Menu

-Pause Menu

-Game Over Menu

Audio

-Shooting Audio

-Sound When Enemy Hit With Bullet

-Sound When Enemy Hits Player

-Sound When Player Dies

-Background Music at Start Screen and in Game

Refinement Stage (Phase 2)

-Adding 3d models of the virus and gun

-Adding a cursor

-Refinement of the game world

-making the plane and the skybox more futuristic and interesting

-Refinement of the menu screen

-Make virus spawn faster with time

**Assumption**

An assumption was made that there are enough tutorials and resources online that a beginner game-developer could create a decent fps-game. Furthermore, that there are enough free assets, such as sound effect and 3d models, online that I would not have to create all of them from scratch using different software. Making every part of the game from scratch could potentially be too time-consuming and I wanted to focus on creating the whole game to get an overview of each steps required to create a game and how they relate to each other.

Chapter 2: Output

2.1 Unity Editor Project

The main output is the project folder used in the Unity application to create the project, which includes all the assets, source code, and dependencies used and submitted as a zip package.

The main output is a Unity project and its asset used to create the game and the export of the project for windows that includes files to run the exported game. The project contains 710 lines of code of which 50 were reused.

File Type: .zip

Intended Recipient: Players and examiner

Results: Result Section

2.2 Unity Project Build Output for Windows

This is the 32-bit and 64-bit build version of the Unity editor project mentioned before.

File Type: .exe

Intended Recipients: Players and examiners

Results: To Be Added

Chapter 3: Literature Review

3.1 Platform

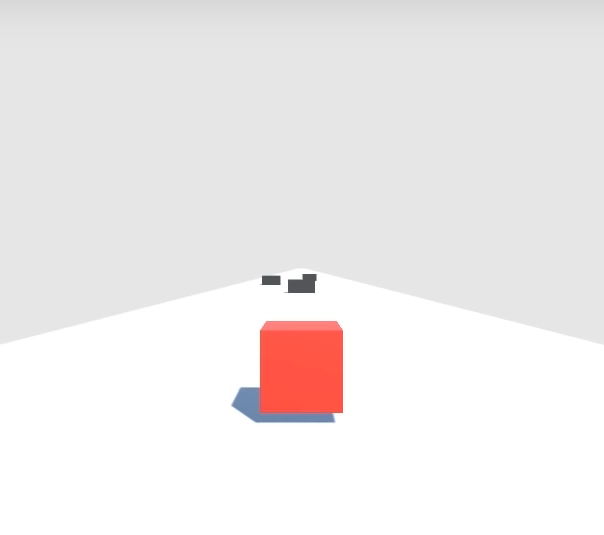
One of the very first thing that was considered was the platform used create the game. There was an option to create a game from scratch, similar to how I made a game rudimentary 2d platformer for the introductory Java module taken in year one. However, since I wanted an overview of the game development process, creating everything from scratch was too time-consuming.

For the development platform Unity and Unreal Engine were considered. They are two of the most popular development platforms to create videos games. For example, popular titles like Fortnite, Street Fighter V, and Borderlands 3 were made using Unreal Engine (Drake, 2021). Assassins Creed, Temple Run, Hearthstone, and Cuphead were made using Unity (Drake, 2021). The main difference between Unity and Unreal is that Unreal has higher fidelity visuals out of the box, requires higher processing power, and higher-learning curve (Creative Bloq, 2021). Since I am a beginner at game development and my focus was not about detailed 3D visuals (which the Unreal Engine offered) but development, I decided to use Unity.

3.2 Programming Language

For the programming language, C# was decided since “Unity uses Mono, which is a cross-platform implementation of Microsoft’s .NET framework”, which is written in C#, and Unity’s libraries are built using C# (Buckley, 2021). Unity considers C# to be the main language for use with the engine. To confirm, I searched online tutorials and resources regarding game development using Unity and they were almost always using C#. Furthermore, C# is a object-oriented programming language, which I am comfortable with (Microsoft, 2021).

3.3 Tutorials

At the beginning of the project during the start of the semester, since I had minimal experience for game development, I followed tutorials to better understand Unity so I can plan better and get a clearer idea of the game that I wanted to create. Having a better understanding of Unity and creating things by following tutorials gave me a general sense of the scope of the game that was feasible within the time-frame and a better idea of the game that I wanted to create. One of the most helpful tutorials was “Unity Beginner Tutorials” series on Youtube by Brackeys. Brackeys has 1.37M subscribers as of April 24th and is one of the largest channel for game development tutorials. The 10-video series covers the basic of Unity, such as installation, creating object/game world, camera movement, UI, and export, by creating a game. The game was a simple game where you control a red box and the player tries to dodge the on-coming black tiles. I really liked the simplistic and futuristic aesthetic of the game. I was also surprised by how though the game was simple the minimalistic and futuristic style makes the game look polished. Thus, I decided to tentatively make minimalism and futuristic/sci-fi style the aesthetic theme of the game that I will be creating. The series gave a good introduction to Unity and overview of the game development process on Unity, which helped me get a better picture of what I wanted to create. 

Game Created in Brackeys “Unity Beginner Tutorial”

3.3 Other Similar Games and Inspiration

I searched on Steam, one of the largest video game distribution service, to find other Corona-themed-FPS games. While there were more 2D games compared to 3D, most likely because 2D games are easier to create, there were some 3D games. One of titles was “Call of Coronga” developed by Luiz Felipe Da Silva Marian published by MushRoom Angels games.

The game is a first-person-shooter and the player shoots down people who are infected with Covid. This game was similar in concept to the game that I wanted to make but looking at the screenshot of the game, I realized that I did not want to create a game with dark and gritty atmosphere since I wanted to players to not feel a sense of dread, which they could already be feeling. This cemented my choice of aesthetic, minimalistic and futuristic/sci-fi, mentioned before in the last chapter (chapter 3.2), since it has a lighter and less stressful aesthetic.

Chapter 4: Method

As mentioned before, the project roughly consisted of the three phases, learning/planning, basic implementation (phase one), and refinement (phase two). In the learning/planning stage, I fleshed out the basic idea of the game that I wanted to create by following tutorials, reading resources, and experimenting on Unity. Based on this, a list of features and sub features were created for the first build of the game. In the basic implementation stage, the basic features were implemented and the features for the next stage was decided. Then in the refinement stage those requirements were implemented. In another word, the three stages I have mentioned could be thought as Pre-Alpha, Alpha, and Beta/Release Candiate.

4.1 Learning/Experimentation Stage

4.1.1 Software Development Methodology

For the development of the project, because I was not familiar with software development methodologies Waterfall method was first considered due to its straightforward structure. Waterfall method follows a strict well-defined step of requirement gathering and documentation, design, implementation, testing, deployment, and maintenance (Lucidchart, 2021). I thought that since the project has a time limit and I am inexperienced with game development, having a detailed plan would benefit me not waste time. However, it quickly became apparent that the Waterfall method would not work as I was trying to build a class diagram. I noticed there are too many holes in my knowledge for game development to build a detailed plan/proper test and I was not sure how long a functionality of a game in the design would take.

I decided to use the Agile method. The Agile method was fitting since it was much more adaptive than the Waterfall method because Agile’s focus on iterative process. In that way, Agile can handle more unknowns, whether it may be in terms of the actual project or outside of the project such as my ADHD symptoms, and allow me to implement as many functionality for the game as possible before the deadline.

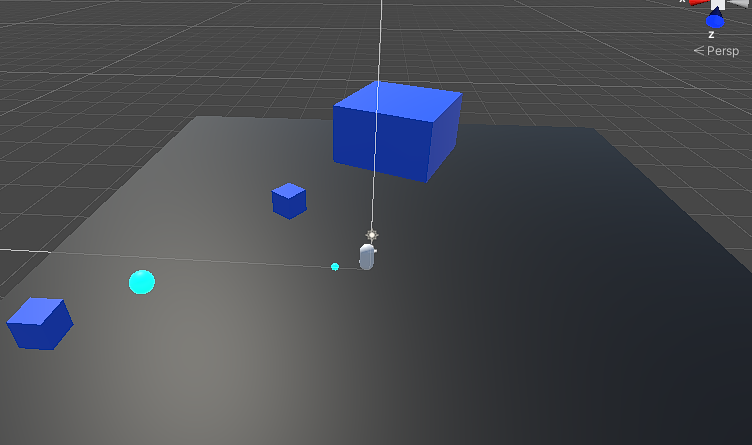
4.1.2 Version Control System

For the version controls system, git was used, since it is a version control system that is widely used and I have experience with it. Additionally, GitHub was also used to store the source code externally.

I noticed that there are quite a lot of files when first committing my project to git. Many were project files and temp files used for the Unity editor. I knew that not all files needed to be committed so I watched a Youtube video “How to use GitHub with Unity” by Brackeys. In the video a .gitignore file for 3D unity project made by github user FullStackForger was used to not commit unnecessary files to git (FullStackForger, 2021). The .gitignore file was used for the project.

4.1.3 Experimentation

The first stage was focused on understanding Unity as fast as possible to set the scope and details of the project. I experimented with the basics such as cameras, changing colors, adding object to the world, creating short scripts to adding movements to the blue spheres etc.



4.1.4 Planning Features

Gaining better understanding of Unity a list of features was created in the order of implementation. The plan was created so that a minimum viable product can be created.

|  |  |
| --- | --- |
| Game World | -Creating a Plane |
| Player | -Creating Player  -Camera Movement For the Player  -Creating Bullet  -Shooting |
| Enemy | -Getting a Virus Like Sphere  -Making It Move Towards Player |
| Player and Enemy Interaction | -Player Loses Health When Hit By Virus  -Bullet Destroys Virus  -Create Multiple Enemies from the side and making them approach |
| UI | -Show Player Health  -Show Player Score  -Start Menu  -Pause Menu  -Game Over Menu |
| Audio | -Shooting Audio  -Sound When Enemy Hit With Bullet  -Sound When Enemy Hits Player  -Sound When Player Dies  -Background Music at Start Screen and in Game |

4.2 Basic Implementation Stage (Phase One)

In phase one, the planned features were implemented. The focus of the stage was the functionality of the game instead of the aesthetic, since aesthetic can always be improved later on.

4.2.1 Game World

For the game world, a simple plane was used. The plane is 3D object included in Unity. Different sizes of the plane were tested to decrease aliasing of the edges from the players perspective. A circular plane was also tested.

The default skybox was used to represent the skybox.

4.2.2 Player

A player 3D object needed to be placed in the center with the ability to look around according to mouse input and shoot bullets. A capsule object was used to represent the player and was placed in the center. The camera was also place at the center to achieve a first-person-view. The code for camera movement was taken from a Youtube video (Learn Everything Fast, 2017). A 3D unity rectangle object was used as a gun with an empty object as a fire point to shoot the bullets from. A square prehab object was used as the bullet. The prehab has a trail render which creates a trail when shot from the fire point of the gun according to user’s mouse click.

4.2.3 Enemy

Enemy viruses will spawn from the edge of the plane and move closer to the player in the center. A floating spherical 3D object was used to represent the enemy virus. The gravity effect is turned off for the sphere so it would float. The sphere was chosen since it fit the 3D virus object that would be added later. The sphere were set so that they do not collide with one another and accidently delete themselves. The sphere moves to the center of the plane.

4.2.4 Player and Enemy Interaction

The player will have a certain amount of health and will take damage if the enemy hits the player. Once the health becomes zero, it is game over. The virus will also have health, which is decrease every time the player hits it with a bullet. Whenever a player destroys a virus, the player will gain a point to its score.

In this stage, a health pool represented by integer variable and a function that decreases the health were created for the virus and the player. Collision detection system were implemented for the bullet and the player object. On collision to the virus by the bullet would decrease the health of the virus, destroy it, and add a point to the player. On collision to the player by the virus, the player’s health will decrease and destroyed once it reaches zero.

4.2.4 UI

During the UI component for showing the score and health There were two main types of UI required for the game. One is the in-game UI, which showed the score and the health of the player. The other is a menu, pause, and game over screen.

The score UI was showed as a number at the center-top of the middle of the screen and the health bar was located at the top left, which shortened as the player took damage. On the slider was also a number showing exactly how much health the player had left. The score and health were connected and was updated according to the player object.

A separate scene was created for the main starting menu with a simple purple background with the title of the game, Corona Destroyer, and a start button that changes the scene to the game scene and starts the game and a quit button that quits the application.

The game over scene was set up in a similar manner with a simple purple background that shows the final score of user. The restart button loads the game and the quit button quits the application.

The pause screen was not a different scene but a UI component in the game scene, which shows whenever the user presses the escape button. Upon pressing escape, the game stops by not allowing the player to shoot or look around and stops the viruses from moving. The pause screen has a continue button which un-pauses the game by hiding the pause screen, enabling player camera movement/shooting, and unfreezes the viruses. The restart button reloads the main game scene and restarts the game. The main menu button changes the scene to the main menu scene mentioned earlier. The quit button quits the application.

4.2.4 Audio

Background music and sound effect are crucial to keep the player engaged. Three different background music for the three different scenes were added. Each background music has a different tone. The bgm for the menu scene has a sense of anticipation, the main game scene has an upbeat tone, and the game over scene has a relative sad tone.

The sound effect for when the player takes damage, shooting the bullet, enemy being destroyed, and clicking one of the menu buttons were added.

4.2.4 Planning

After the basic implementation was done. I tested the game by letting others play with the game. I also played numerous times to find improvements to make. Based on the input from the tester and my experience from the playing the game, I created a list of problems with the current game in order of importance and its solution.

|  |  |
| --- | --- |
| Problems (in order of importance) | Solutions |
| The sphere, representing the virus, and the rectangle object, representing the gun, looks unprofessional and too simple. | Adding 3d models of the virus and gun |
| The game world, the plane and the default skybox, seems too bland. | Refinement of the game world  -making the plane and the skybox more futuristic and interesting by making the plane have glowing grid patters and change the color of skybox |
| The player does not know where the bullet will go because of the lack of cursor. | Adding cursor UI and making the bullet move towards the specific point of the cursor |
| The UI and menu screen looks unprofessional and uninteresting | Refinement of the UI/menu screen |
| The game feel too predictable | -Make virus spawn faster with time  -add virus with different color, health, damage, and speed |

4.3 Refinement Stage (Phase Two)

In phase two, the solutions were implemented.

4.3.1 3D Models

Originally, simple 3D shapes were used since I wanted to focus on the basic functionality and had no experience with 3D models. In order to make the game more professional, I decided to add 3D models for the virus and gun. Creating a 3D model from scratch using Blender, a free 3D modeling software, but since there were more features that needed to be implemented and learning another software would be time-consuming, I decided to find free 3D assets online. Virus and gun 3D models in “Creator Kit: FPS” on Unity Learn, a tutorial platform that Unity offers, was used.

4.3.2 Improving Game World

Originally, a simple plane and the default skybox was used for the game world. In order to make the game feel more futuristic and aesthetic, a grid like pattern was added to the plane by adding material found in “NeonSphere – Starter Pack”, a free asset pack from the unity asset store. The intensity of the material was adjusted so that the plane would glow to give a sci-fi look.

The default skybox was changed to a fading neon purple to give a futuristic atmosphere to the game.

Due to the changes in the color of the plane and the skybox, the colors of other objects, such as the bullet needed to be adjusted so that they are visible after the changes to the game world.

4.3.3 Cursor

In order to, make it easier and less frustrating to hit targets, cursor was added to the center of the screen. While the details of how to make the bullet move towards the cursor when shot will be described later in the result section of this report, a ray-cast was used to detect if the ray cast hit an object and if the ray-cast hit an object the bullet will move towards that location. If the ray cast did not hit anything, the bullet would be shot off into a far off point.

4.3.4 Improving UI/Menu

The text in UI and the menu used default font and the back of the menu was a solid color, which made the game look unprofessional. In order to improve them, the font was changed to a more aesthetic, sci-fi looking one.

In order to make the background more interesting, we used an image of the background of the game without the gun or the UI. Due to the changes made to the game world described earlier, the game world looked great. Furthermore, having the game world in the starting menu showed what kind of game the player would be playing, which potentially gives a sense of anticipation to the player.

The game over scene was deleted since it had a uninteresting monotone background. Instead, a UI component was added to the actual game scene and showed whenever the player’s health reached zero. At that moment, the game would pause, which gave an interesting background of the moment the player died, making the game-over scene more exciting and interesting.

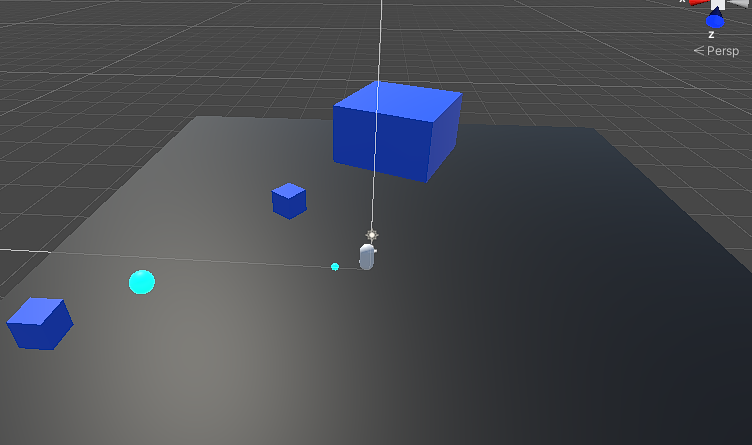
4.3.5 Adding Variation To the Virus

In order to add more variation to the game play, a virus of red color was added with different stats. The red virus moves slower, but has more health and deals more damage.

Furthermore, a timer was created to spawn the virus gradually faster as the game progresses. The green virus will be spawned more compared to the red virus. The spawn-rate ratio between the red and green virus was decided on trial-and-error.

Chapter 5: Result

5.1 Learning/Planning Stage

Initially, I experimented with Unity to learn as much as possible. At this stage, I was not focused on creating the game but trying things out. A cylindrical player in the center with the main camera attached was created. Rectangular 3D objects and a black plane was created as a game world. Scripts were created to move the spherical balls to learn about movement. 

5.2 Basic Implementation (Stage One)

5.2.1 Game World

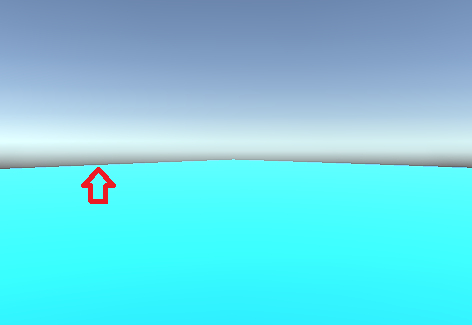
In Unity, a scene is where the contents of the game, the assets, are stored in Unity. In my project the scene was named “main” and it contains the assets created.

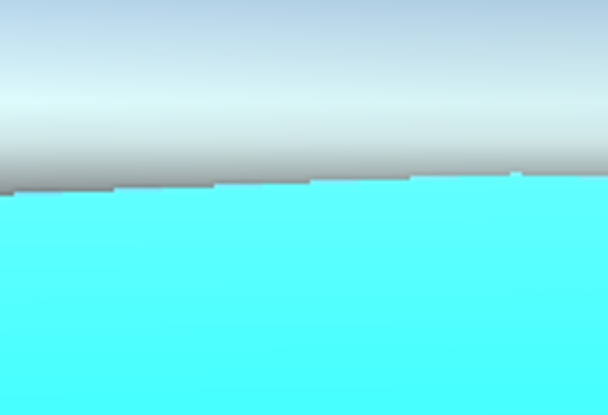
In Unity, in the Lighting panel, the lighting and sky-box can be adjusted. I used the Default-Skybox with Directional Light. The skybox made the game look expansive and the directional light created shadows, which made the game look more realistic. In order to make the color more vibrant, I used “Auto Generate” under “Lightmapping Setting”, instead of manually setting the lighting setting.

The game world is a large square plane. The player’s point of view (the main camera) is set to the center. The color is set light blue using “Materials”. In unity, materials are scripts used to define the looks, such as texture of the object. Light blue was chosen since it gave the game a futuristic look and was relatively easier on the eye compared to other neon colors. Reddish neon colors were not considered since the virus is red and the virus could blend-in with the ground making it more difficult to see from the player.

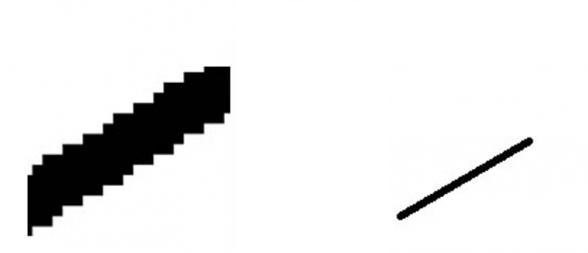
In Unity, “Main Camera” object is used to decide what is shown when the game is run. The camera was set above the center of the plane, facing one side of the plane.

Originally, the plane was scaled by 5. The plane was made small as possible to reduce the size of the game and make the game run smoother by making the game render less. The plane could be small since the player is immobile in the center. However, from the player’s point of view, the edge of the square looked jagged.

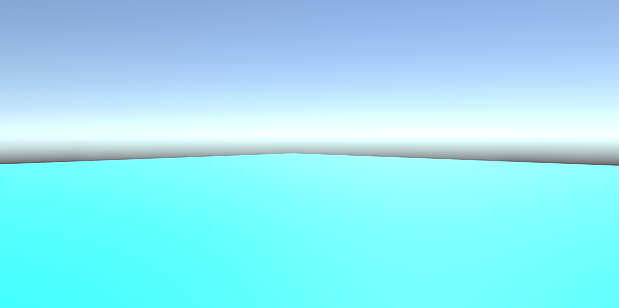




The edges are jagged because images on screen are created using tiny squares called pixels. For example, seen bellow, the close-up of the diagonal line (right) is on the left.

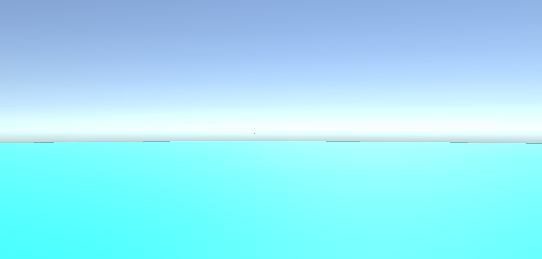


Anti-aliasing was applied in an attempt to make the edges look more straight. Anti-aliasing allows the edge of each pixels to blend in with one another, blurring them. This technique creates a smoother-looking edges. In the project setting under “Quality”, we can change the degree of anti-aliasing. There are four degrees, Disabled, 2x Multi Sampling (the default setting), 4x Multi Sampling, and 8x Multi Sampling. The 8x Multi Sampling did improve the look of the edges.



However, we could still see the aliasing and anti-aliasing requires more resource to render objects, I decided to set back the anti-aliasing to the default, 2x Multi Sampling.

A spherical plane was also tested. Since unity does not provide a spherical plane, a sphere was flattened by scaling the y-axis by 0.01 and x/z-axis by a 100. But black line appeared on the edges. Scaling the sphere-plane by 200 produced similar aliasing issue.



Instead, the plane was scaled by 20, eliminating the jagged edges. The plane is large enough that from the player’s point of view, the plane seems to go beyond the horizon. Since the player cannot move, this large square suffices as the game world.

5.2.2 Player

5.2.2.1 Player Model

A capsule was used to represent the player. Capsule seemed the most appropriate as it had the most humanoid-shape compared to other 3d objects, such as sphere, square, and cube, provided by Unity. There is a “ragdoll” 3d object option that allows users to specify detailed parts of a body, such as the legs, torso, head, and total mass etc. But since the player will be immobile in the center, it was not considered. An empty object Player was created and made the parent object of Capsule for organization.

5.2.2.2 Camera

The camera moves according to the mouse movement. If the mouse moves right, the camera turns right and if it moves up the camera turns up etc.

For the camera movement, I used code that I found from a Youtube video (Learning Everything Fast, 2017).

A C# script, Camera.cs was created and was attached to the Main Camera.

The mouse can move in two directions, horizontally (along the x-axis) or vertically (along the y-axis). In order to turn the camera horizontally, the camera is rotated on the y-axis according to the mouse’s horizontal movement, according the x-coordinate. In order to turn the camera vertically (up and down), the camera is rotated on the x-axis according to the y-coordinate of the mouse. Transform.eulerAngles represents three-dimensional rotation around the z, x, and y-axis in that order.

When running the game on unity, we could not turn the camera when the mouse cursor moved outside of the “Game” window, where you run the game. Furthermore, the mouse cursor was visible. This was a huge annoyance as the mouse cursor would need to be dragged to the center or inside the “Game” window continuously. An empty object Game Manager was added to the game world. GameManager.cs was attached to it. Cursor.lockState = CursorLockMode.Locked; was added to the Start() so that the cursor icon disappeared and the cursor was locked in the center, when the game is run.

5.2.2.3 Bullet

The bullet asset was created using a Prefab. In Unity, prehabs are reusable assets that allows for creating and storing game object with all the components and property values. This prevents making the same edit to every copy as they are all kept in sync when using Prefabs. I created a cube with red material and added a Trail Renderer component to add a trail to make the bullet more interesting than just a cube.

A c# script Bullet.cs was created and attached to the prehab. In Start() the velocity is set with

rb.velocity = transform.forward \* speed;

Transform.forward returns a normalized vector representing the z-axis. This direction is multiplied by the speed to set the velocity of the Rigidbody of the bullet, which moves the bullet in a forward direction. A Rigidbody component applies Unity’s physics engine to the object attached and also allows the user to tune its affect on the object. While the velocity could be set in Update(), I called it in Start() as the bullet only goes in one direction and setting the velocity every update for multiple bullets use unnecessary resource.

In order to shoot the bullet with mouse click, a long cuboid with green material was created to represent a gun and was positioned around the middle of the capsule. An empty object fire position was also created to set the position in which the bullet will be fired. The gun and the fire position was made to be the child of Main Camera so that the gun will move as the player moved the mouse to look around.

Player.cs was added to the Player. The Update() function will constantly check for a mouse click with if (Input.GetMousebuttonDown(0)). Once the mouse is clicked, Instantiate is called to create a bullet object from the Fire Position that we created before.

One of the problems encountered was how the bullet was never destroyed once they were shot. They flew outwards indefinitely until the game was ended. A countdown mechanics was implemented to destroy the bullet after a set time.

Games are presented on screen according to the frame rate, the frequency at which images (frames) are displayed on screen. In Unity, the Update() function is called at every frame. However, frame rate is affected by the specs of the computer and the refresh of the display, so it varies according to the user’s machine. Say in Update() an object was moved by 1cm, a user with 60 frames/sec would move 60 cm every second but a user with 30 frames/sec would move by 30cm. Even on the same machine, framerate fluctuates as it is affected by cpu and gpu usages. To be independent from framerate, Time.deltaTime was used. Time.deltaTimes is the amount of the time the last frame took to complete.

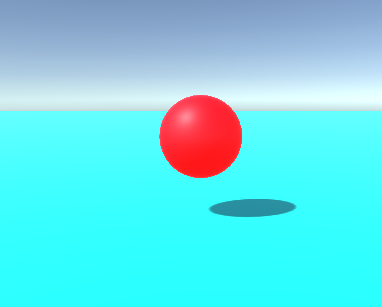
bullet\_duration = bullet\_duration - Time.deltaTime;

Subtracting Time.deltaTime from a duration, would ensure that a countdown will be consistent amongst different machines. When the bullet\_duration reaches zero, the bullet is destroyed by calling the Destory method.

Another problem encountered was how when turning the camera and shooting. The gun and the bullets would flicker when moving. This was because the code that moved the camera in Camera.cs was inside Update(). In Unity, the Update() method is called for all the object in unknown order. The reason for the flickering was because the camera moved slightly earlier than the bullet before the bullet location was updated so the bullet appeared in space. A better approach would be for the bullet to be first updated to the new location and the camera turns to capture the bullet. Instead of using Update(), LateUpdate() was used to in Camera.cs to move the camera. LateUpdate() is called after all Update() functions are called.

5.2.3 Enemy

Initially, a sphere 3d-object was used to represent the enemy. RigidBody component was added to apply movement. Virus.cs was attached to this object, where in Start() the virus is faced to the center and the velocity is set to move towards the center. Instead of using more resource by constantly setting the same velocity in Update(), it was set in Start() since the virus is moving in one direction towards the center with the same speed.



When running the game, the sphere jittered as it approached the player. This was fixed by changing the interpolation of the RigidBody from “none” to “Extrapolate”. According to the Unity documentation about Rigidbody.interpolation, physcis is running at discrete timesteps, while graphics is rendered at variable frame rates. “This can lead to jittery looking objects, because physics and graphics are not completely in sync” (Unity, 2021).

5.2.4 Player and Enemy Interaction

One of the problems encountered was when the virus collided with another virus, they disappeared. The objects were destroyed upon contact. Three different solutions were considered. The first was turning the virus’s Sphere Collider off. Colliders are used to define the area of an object where if another object’s collider intersected, a collision event is created. However, since the virus need to interact with other objects such as bullets and player, this options was not optimal.

Another option was to create a layer for the virus and change the physics setting so that the object in the virus layer will not interact with other viruses since they are in the same layer. In Unity, we can create layers and assign them to objects. For example, the ground plane and rocks can be put in a “World” layer, the enemies could be put in the “Enemy” layer etc. Virus layer was created and in Edit -> Project Settings -> Physics the layer collision matrix can be changed to set whether each layer collides with the other layers or itself. Once the duplicated virus objects were assigned to the “Virus” layer, in the layer collision matrix, the setting was changed so that the “Virus” layer did not collide with itself.

Ultimately, I made the virus object into a trigger. In a Collider component, we can check “Is Trigger”. Triggers are ignored by the physics engine, so an object would not physically collide with another object.

5.2.4.1 Score System

Whenever the bullets collides with the virus and destroys it, the player will receive a point. In order to do so, an int variable, current\_score was created in Player.cs. AddPoints(int points) adds the points to the current\_score.

The bullet object needed to be able to update the current\_score and the game over scene that will be shown after the game needs access to the current\_score since it will show the player’s final score. Since two objects needed to have access to Player.cs and there is only a need for one instance of Player.cs since this is a one-player game, instead of assigning instances of Player.cs to those two objects, Player.cs was made into a static instance.

public static Player instance;

The above line was added a variable in Player.cs. “Static” indicates that every instance of a class will contain the exact same pointer to a variable.

private void Awake()

{

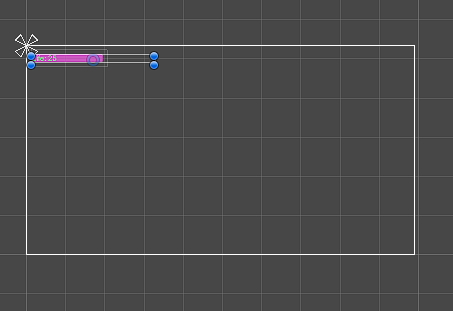
instance = this;

}

Awake() function, as seen above, was added to Player.cs. What Awake() functions do is that they are called before Start() functions. “this” is the instance of the Player.cs. The specific instance of the Player.cs when the game is run is set to the static variable “instance”. This allows other classes to access the functions and variables of Player.cs.

5.2.5 UI

In order to display the amount of life that the player has left, a UI that shows how much life the player has was created on the top left corner. In Unity, a “Canvas” is an rectangular area that includes all UI element. The first child element of the canvas in the hierarchy is drawn first, then the second child is drawn next and so on. A slider was used to show how much life the player has left and another text component was drawn on top by making the text component the second child of canvas after the slider.



A canvas



Canvas in the hierarchy

In order to set the value of the slider, two new UI variables health\_bar of class Slider and life\_text of class Text were added to Player.cs. The Slider and Text class are included in a native library called UnityEngine.UI, which was imported in the beginning. The Slider and Text component from the hierarchy were assigned to those variables. In Start() the life\_text and health\_bar slider’s values are set. In order to update the value whenever the player is hit by the virus, in the TakeDamage() function those two components are updated.

Another text-UI component was added to show the player the score and was placed in the top center of the screen. The score is updated in AddPoints() function in Player.cs, showing the current\_score.

5.2.5.1 Menu Screen

In order to create a start menu, a new Scene called menu was created. In the new scene, a Canvas component contains a text component that shows the game title and two button components, which starts the game and quits the game.

A new script menu.cs was created. This script contains two public functions StartGame() and QuitGame(). StartGame() using the native SceneManager class’s LoadScene() function to load the “main” scene, which contains that main game. Initially, call the LoadScene() method returned an error but that is because I was missing a crucial. In order to load scenes, we need to go to File->Build Settings and add all the scenes used in the game.

The QuitGame() function calls the Quit() function in the native Application class to shut down the game.

**Pause Screen**

The pause screen was created so that the player can stop playing for a moment, quit the game, or restart the game. The pause screen is activated by pressing the escape key in game. This action is registered in the Update() section in GameManager.cs if the pause screen is not already active. The pause screen was made in a similar fashion to the menu screen, but instead of making a new scene it was made inside the canvas of the “main” game scene. A new public GameObject variable pause\_screen that contains the pause screen component and Pause(), Unpause(), MainMenu(), Restart(), and QuitGame() were added to the GameManager.cs. The Pause() function activates and makes the pause screen visible and unlocks the cursor so it can freely move. Then sets Time.timeScale to 0f, which basically freezes the since Time.timeScale controls how fast objects move in the game. The Unpause() function deactivates the pause screen, locks the cursor to the center and sets Time.timeScale back to 1f. However, since the shooting mechanic and camera movement is based on mouse movement and not on time, the player can move the camera and shoot a stationary bullet that doesn’t move in pause screen. So a guards were set so that the player can shoot or move the camera only if the pause screen is not active.

The resume button was made larger and placed in the center than the other buttons so that the player does not accidently click other options and lose the progress of the game.

**Game Over Screen**

The game over screen shows the score of the user.

**-Can potentially add about how you can make other menu scripts use menu.cs**

**Audio**

-can talk about the master mixer etc and how the audio section works

**BGM**

In order to keep a player engaged to the game, the player needs to be stimulated in different ways as possible. Background music adds another layer of stimulation. I looked for different free background music. There were many great songs on Sound Cloud, a prominent online audio distribution platform. Songs that are too complicated or lyrics were avoided since it could distract the player from the game and could potentially mask the sound effects in game. BGM needs to be simple enough that different kinds of players will not have any problem, but needed to be engaging enough to entice them.

Ultimately, 8-bit music were used as the BGM. 8-bit music, also known as “chiptunes”, are a genre of electronic and synth-music that sounds as though they were made using computer chips. The 8-bit music fit the futuristic color scheme that was used in game. Parts of 8-bit-styled-song tend to have simple and repetitive musical notes so it did not distract the player. There were many great 8-bit BGM so I decided to use different songs for each scenes, main menu, the actual game, and the game over scene. For the main menu scene, “Underclocked” by Eric Skiff was used since the song conjured a feeling of anticipation, which fit the anticipation that player would feel when the player is about the start the game. During the game, “Powerup!” by Jeremy Blake was used since it has a relatively upbeat tone. An upbeat-tone is better than a sadder-tone since the player will be listening to it repeatedly and the player will less likely to be frustrated. For the game over scene “A night of Dizzy Spells” by Eric Skiff was used since it has a more toned-down, sadder feeling, which was suiting since the player failed.

**Sound Effect**

**Menu Sound Effect**

Sound effects were added when clicking options on menu. When the player presses, “Restart” on the pause and game over menu or “Start” button on the main menu, a short and upbeat sound effect plays. When pressing “Quit Game” button or the “Main Menu” button on the pause menu, a more muted, in a way, sadder tone plays. I used to two different sound effect to add variation and in the hopes that a more upbeat sound effect may slightly encourage players to play again and a sadder tone may slightly discourage players from quitting. The sound effect were found on freesound.org, a website with CC licensed audio samples.

In order to finish playing the sound effect before switching/loading scenes and quitting the game, the game needs to wait for a specific time. Coroutine was used. “A coroutine is a function that allows pausing its execution and resuming from the same point after a condition is met.”(1)

1: [**https://www.javatpoint.com/unity-coroutines**](https://www.javatpoint.com/unity-coroutines)

The sound effect is played and the WaitForSeconds() function is called to wait for a given amount of seconds. After those seconds, a scene is loaded or the game is quit.

WaitForSeconds() worked for the main menu and the game over scene but did not work for the pause menu. The function only started waiting after the screen was un-paused. This was because in order to stop the viruses from moving the Pause() function in GameManager.cs scales the time to 0f, stopping the flow of time in the game. WaitForSeconds() stops time based on scaled time, that is why it started to execute after the Time.timeScale was set back to default of 1f. WaitForSecondsRealtime() was used instead of WaitForSeconds() since the former uses unscaled time. The time scale was set to the default of one in the start of the main game, since the time scale was set to 0f in at the pause menu.

-can possibly elaborate on how coroutines was implemented

-can possible write about how I created SFXController.cs

**Game Sound Effect**

The sound effect for shooting, player taking damage, and virus exploding were implemented. The audio sources were found on freesound.org. The shooting sound effect sounds like a laser beam to fit with the futuristic theme. Instead of using an actual explosion sound, I used the sound of breaking glass. I wanted a crisp and lighter sound since the player will be constantly listening to it and the heavy sound of an explosion may not be comfortable.

**Audio Management**

Because of the growing number of the audio files used for the game and different scenes sharing the same audio files, I decided to make a static class that manages all the audio files for better code design. Instead of assigning the audio files directly to different files, each scene will only need an empty object with the AudioController.cs and can play the sound from any other scrips.

**Explosion Effect**

The explosion effect was one of the free effects in Shrebb’s Particle Collection, which was found on the unity asset store. The particle effect could be created in Unity but having access to a collection of effects made it fast and easy to compare and contrast different kinds of explosion, especially since I did not know what kind of explosion I wanted. I chose Shrebb’s package since it contained sci-fi themed explosion which matched the theme of the game.

**Export**

For the initial export, the menu screen was too small and the cursor was visible in the center during the game. The visibility of the cursor was set to false at the start of the game scene to hide the cursor by adding “Cursor.visible = true;” in Start() of GameOver.cs.



The Menu Screen During the Initial Export

In the canvas object, where the UI is located, the Canvas Scaler component sets the scale of the UI. For the “UI Scale Mode” section I choose “Scale With Screen Size” option instead of Constant Pixel Size. This made it so that the UI stretches according to the user’s screen size instead of staying at a constant size. The Reference Resolution, the resolution the UI layout is designed for, was set to 800 by 600. This valued was decided though testing out different aspect ratio and screen size and 800 by 600 looked the best.

The game is exported as a full screen game since the windowed border makes it seem less immersive to the player.

**Testing**

We tested phase 1 with my father and a friend of mine. My father said its hard to know where they are coming from and a radar-like system would help. A friend who has a not so optimized computer said that the lap has a difficult time running the game.

**Phase 2**

Now the basic game was done we got to phase 2 with new additional game components.

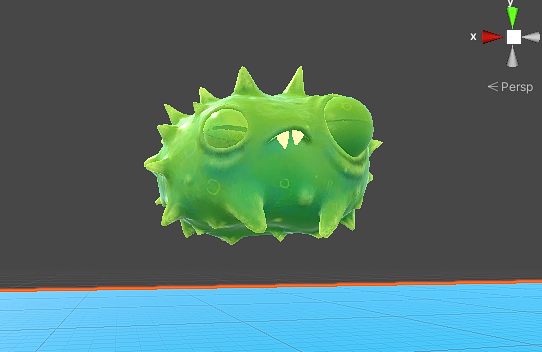
Testing

**Virus and Gun Models**

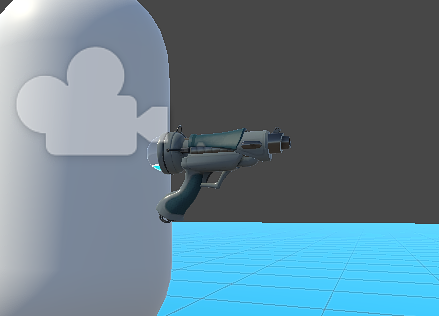
Initially, instead of using detailed 3D models, a sphere was used to represent the virus and an elongated-cube was used to represent the gun. This was done since I was inexperience with game developing, so I did not know how long it will take to implement basic mechanics such as shooting and camera movement. Working with simple shapes made it easier to create the basics of the game. Furthermore, I had not experience creating or working with 3d-models. Thus, the plan was to add detailed 3D models after the basics of the game were implemented and there was still time left.

A 3D model that looked like a virus was needed. Creating from scratch using Blender, a free and open-source 3D modeling software, was considered but the UI, audio, exporting, testing, and writing the report needed to be done, so learning another software from scratch, creating the asset, and then incorporating it to the game seemed too time-consuming. Thus, I looked for a free virus-model online but could not find any. There were a few models on Unity Asset Store, but they were not free. Ultimately, I used a 3D model in “Creator Kit: FPS” on Unity Learn, a tutorial platform that Unity offers.

The asset for the creator kit was downloaded using Unity’s package manager. The Germ Spike prehab and the gun, Healmatic500 were used. The Germ Spike has an animation where it bobs up and down and makes a continuous bumbling sound audible from a close-enough distance.



Germ Spike prefab



Healmatic500

A sphere collider, rigidbody, and the virus.cs script were added to the Germ Spike prehab. The Germ Spike prehab was tagged as “Virus”. The center point that the virus aim towards were lowered since the new virus with 3D model seemed to approach higher in the player’s view. In other words, at

transform.LookAt(new Vector3(0.0f, 1.6f, 0f));

in Start() function of Virus.cs, 1.6f was changed to 0.5f.

**Cursor**

The cursor is located in the center of the screen. In order to shoot the bullet to where the cursor is pointing ray cast was used. Ray cast is an imaginary line from a position to another. If the ray cast intersect with an object, the bullet will shoot to that position.

Initially, in Player.cs, a Ray object was created with

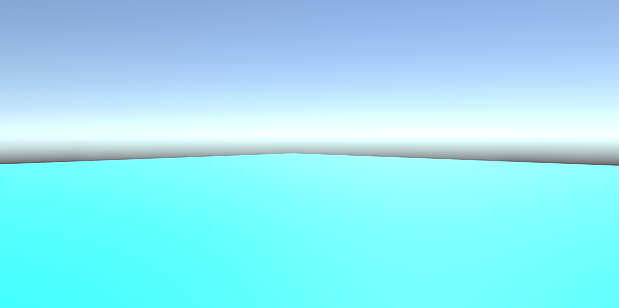
ray = main\_cam.ScreenPointToRay(new Vector3(x, y, 0));

The Unity native function ScreenPointToRay in Camera class returns a ray that starts from a specific position on screen, in our case, the center of the screen. However,

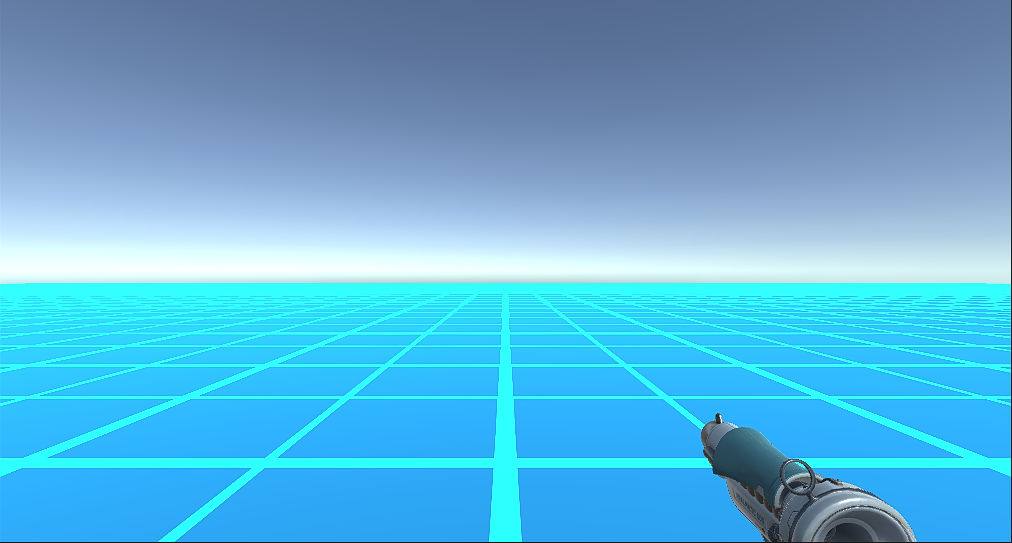
The Physics.RayCast(ray, out hit, distance) function returns a boolean if the “ray” parameter hits an object within the “distance” and outputs the information to “hit”, RaycastHit object. If the ray does not hit anything, the bullet is shot forward to the position into the distance.

**Skybox and Plane Material**

The aesthetic of the game world was bland since phase one was focused on the basic implementation of the game. To give a more futuristic look to the game, a material with tile like feature found in “NeonSphere – Starter Pack”, a free asset pack from the unity asset store, was added to the plane. The tiles were made smaller to create a grid-like structure and colors were changed to neon blue.

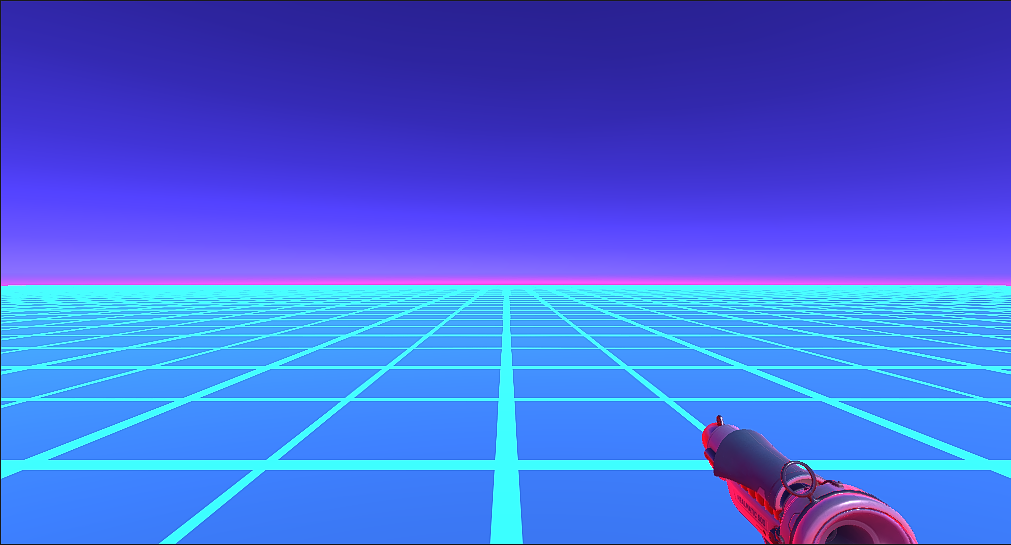


Before the Material



After Tile-Like Material Added

For the skybox instead of using the default sky box, a new neon purple sky box was created. The horizon was made more pinkish to add to the futuristic style.



With The New Neon Skybox

**Menu and UI**

I wanted to put the game in the background of the menu. I had a little photoshooting session in game. I implemented a pause function that doesn’t shot the pause menu screen so that I can screen shot parts of the game play as bellow.



However, the picture looked too cluttered in the menu. In order to be more simple and aesthetic, I used a picture with only the game world, which fit the futuristic and minimalistic feel of the game.

Changed the font and the background scene to the above. I got rid of the UI and the gun to make the picture less cluttered. I also deleted the game over scene and added a UI component to the main game scene, so that the game over ui will show at the moment of players death.

**Reflection**

It was crazy how changing small details like the font makes the game look so much more professional and better-looking.