3rd Defense Report
Aperture

SCOTT TALLEC"α"

ABHISHEK BOSE"β"

RAJAT JOHN"Ω"

SOFIANE BEKHAT "γ"

Contents

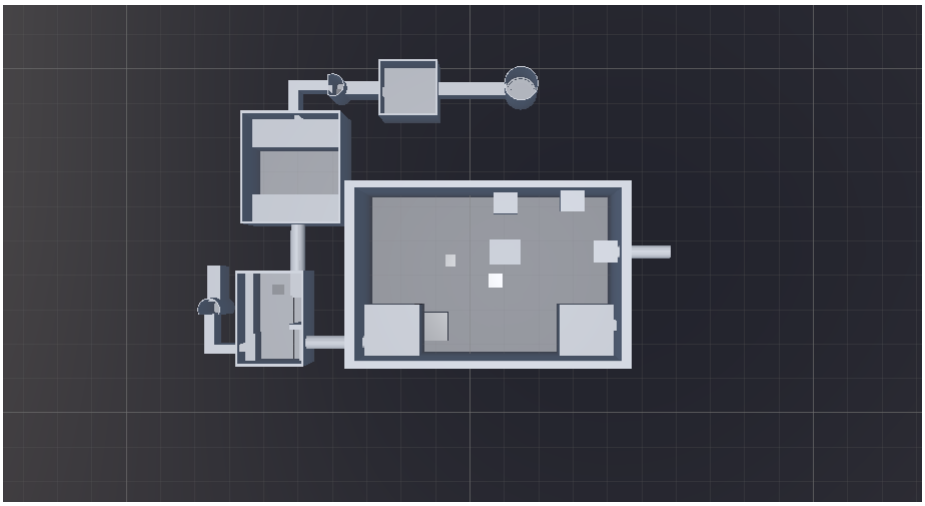
1 Introduction

This project has sometimes been a joy sometimes not so much, but ultimately this project has taught us so much about Game development and programming as a whole. We have learned a lot and have hopefully been able to put our vision into the game as we are very passionate about this project.
Here are the details of our finals updates to the project and any issue we may have encountered during our time with it.

2 Advancement in the project

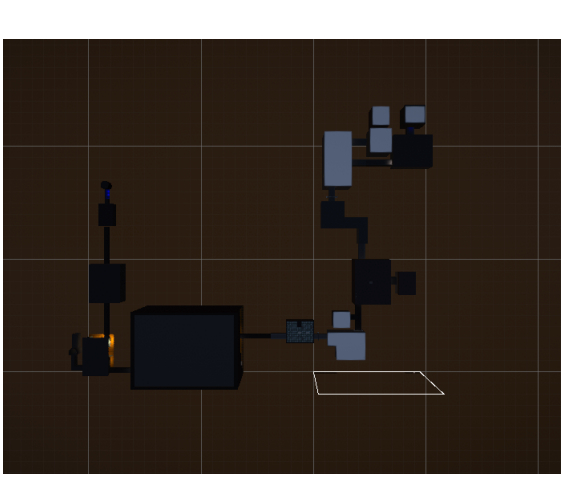
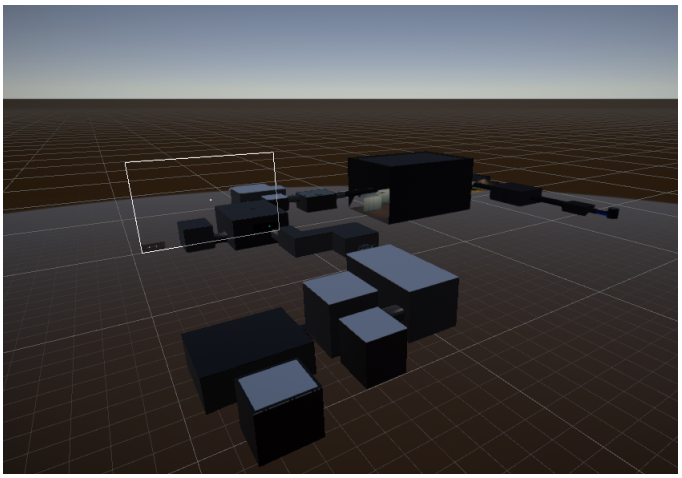
2.1 Level Design

Since the beginning of the project, We was really exited about it and scared as well, because we didn't have much knowledge on Unity and Blender. During the early days of the project, we spent a lot of time on watching videos and reading tutorials about 3D modelling and principles of game design. we preferred "baby steps", which means learning the concepts slowly and make gradual progress. Through this method of working, we managed to build the first prototype of the game map (1st defence) , Which is this one :



We were proud of the result that we have achieved, But still, somehow it wasn't enough. There were some problems. For instance the rooms were not perfectly connected, the rendering of the rooms was not great (the rooms were clinking). We immediately started to look for a solution. After a long research, we found the solution of the problems.
The map

The map of the game is finally complete. The map is basically composed of two levels, level 1, which is an easy level. Then we have level 2, which is harder than the previous one. The map is is furnished with some sci-fi assets that have been downloaded from the Unity Asset Store. In addition, lights and particle effects have been placed in different parts of the map.

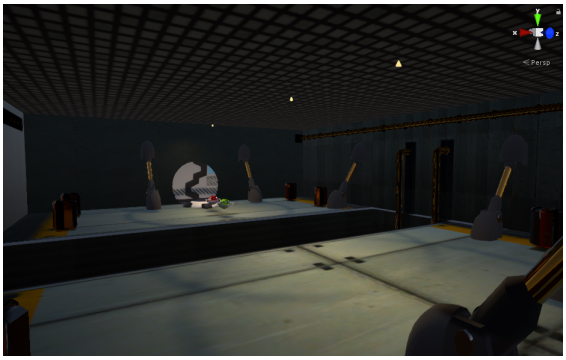
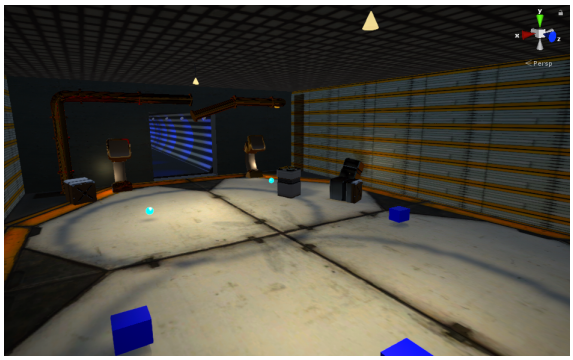



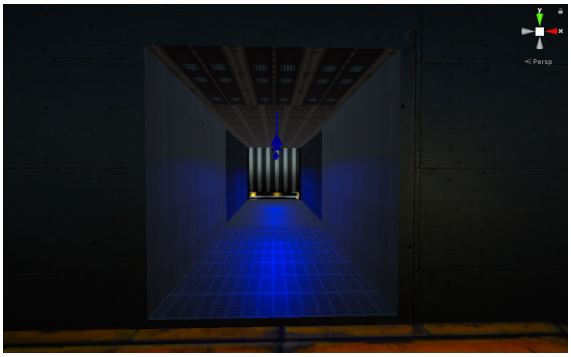
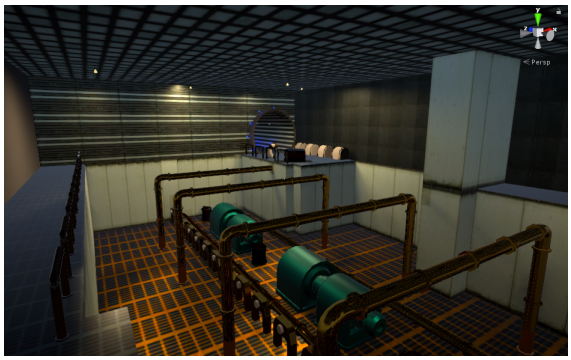
Now we will walk-through the map and talk about every modifications that have been made.

LEVEL 1

Few modifications have been made to the first level. First of all, we have added roofs, finally!! Plus, some assets have been placed in each room of the level, their purpose is to furnish and add more details to the level. Finally, lights have been placed in all rooms and smoke particle effect was implemented in the third room of the level (we will talk about it in details later).

Here are some pictures of the level 1 :

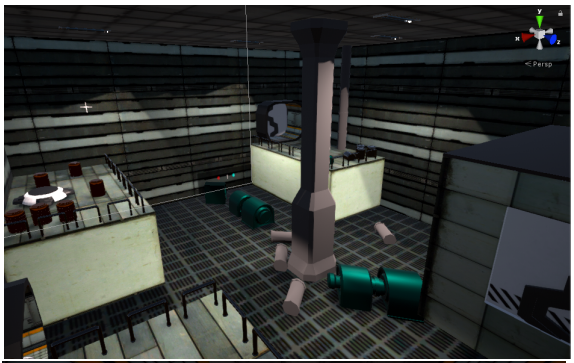
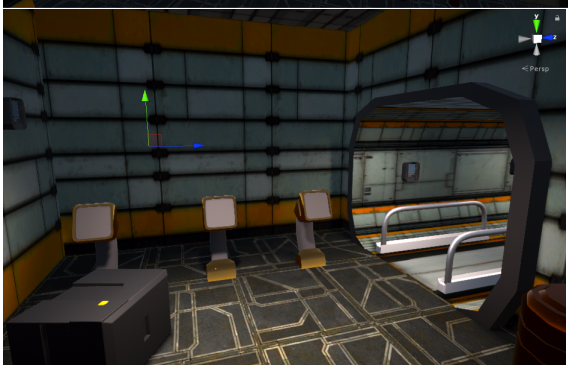



LEVEL 2

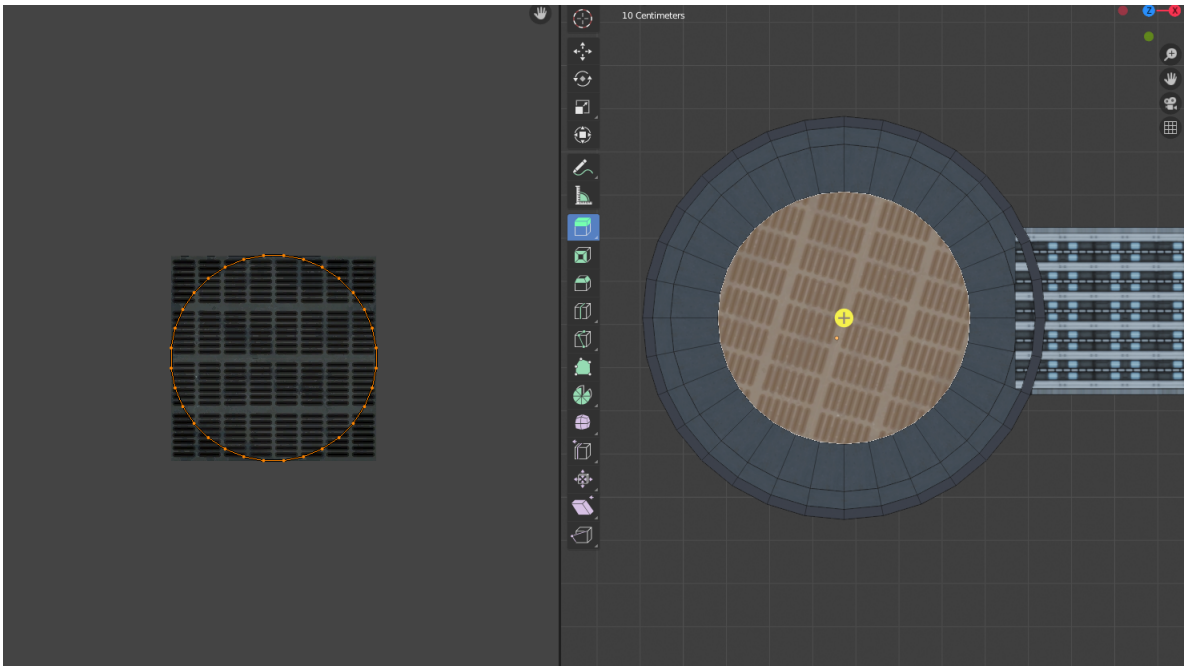
The level two is as well implemented, by using the powerful software Blender 3D. The level was built the same way as the previous one : Different rooms connected through corridors. In order to differentiate the two levels, different textures have been applied and a different corridor has been created and different lighting colors as well.
Like the previous level, some assets have been placed int order to furnish the map and add details onto it.

Here are some pictures of the level 2 :


Mapping of textures

The textures have been applied in each room. The process of applying textures (Texture mapping) was done in Blender, by using the UV map Editor.



The picture above represents the UV map editor. With this tool, one can bake different textures in different parts of a mesh. One can scale, rotate, move, and adjust the settings till he gets a good combination between the mesh face and the texture itself.

We got the texture pack online. This texture pack includes more than 30 sci-fi textures.

2.2 Game Graphics

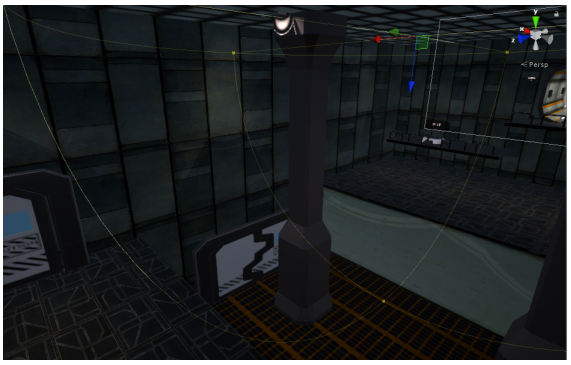
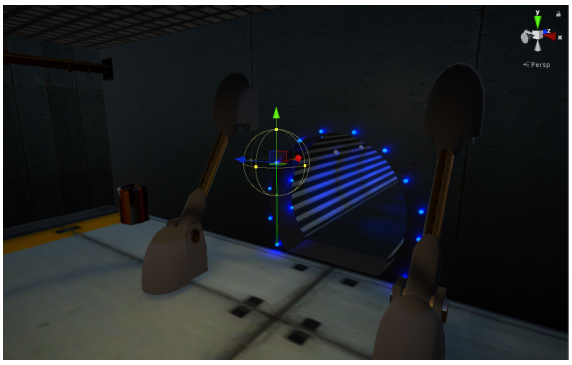
In this part we will illustrate the the modifications that have been made in order to enhance the graphics of the game.

Lighting

Lighting is an important element for graphics.In fact it is the one which improves drastically the look of the game and as well its graphics.

The lights that we used are predefined lighting systems that have already been implemented by unity. We mainly used two different lighting systems: The spotlight and the point light. The spotlight is used to light the rooms and corridors. The point light was used to light specific area of a room. In our game, it corresponds to the blue lights around the entrance of the first corridor of the first level.

Here are there two lighting systems used in our game : the first one is the spotlight, the second one is the point light

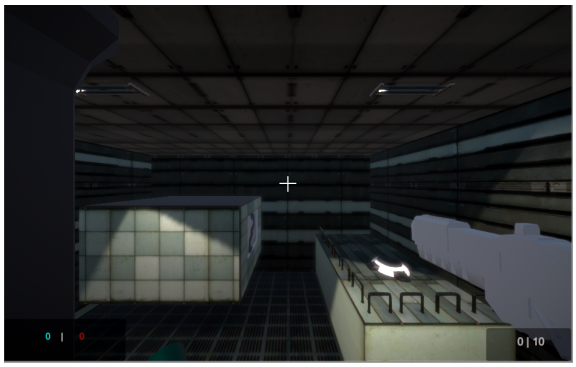
Post Processing effects

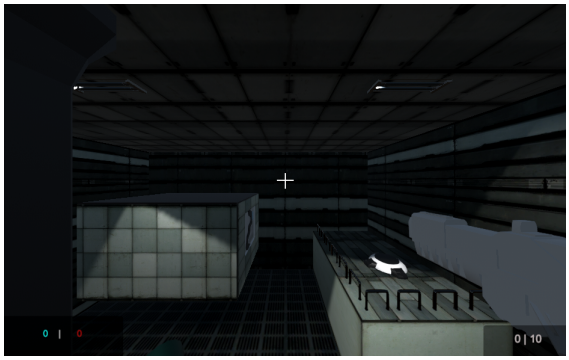
The term post-processing is used in the video/film business for quality-improvement image processing methods used in video playback devices, and video players software and transcoding software. It is also commonly used in real-time 3D rendering to add additional effects

We used the post processing effects in order to improve the visual look of the game.Thanks to these effects, the game looks much better than before. We used the unity post processing stack in order to access and apply the visual effects.

There are a lot of visual effects that we could apply, but we just chose the ones that satisfies our expectations. The effects that we used are : Anti-aliasing, Ambient occlusion, Depth of Field, Motion Blur, Color grading.

Here are the images that show the difference between the game with visual effect, and without them :





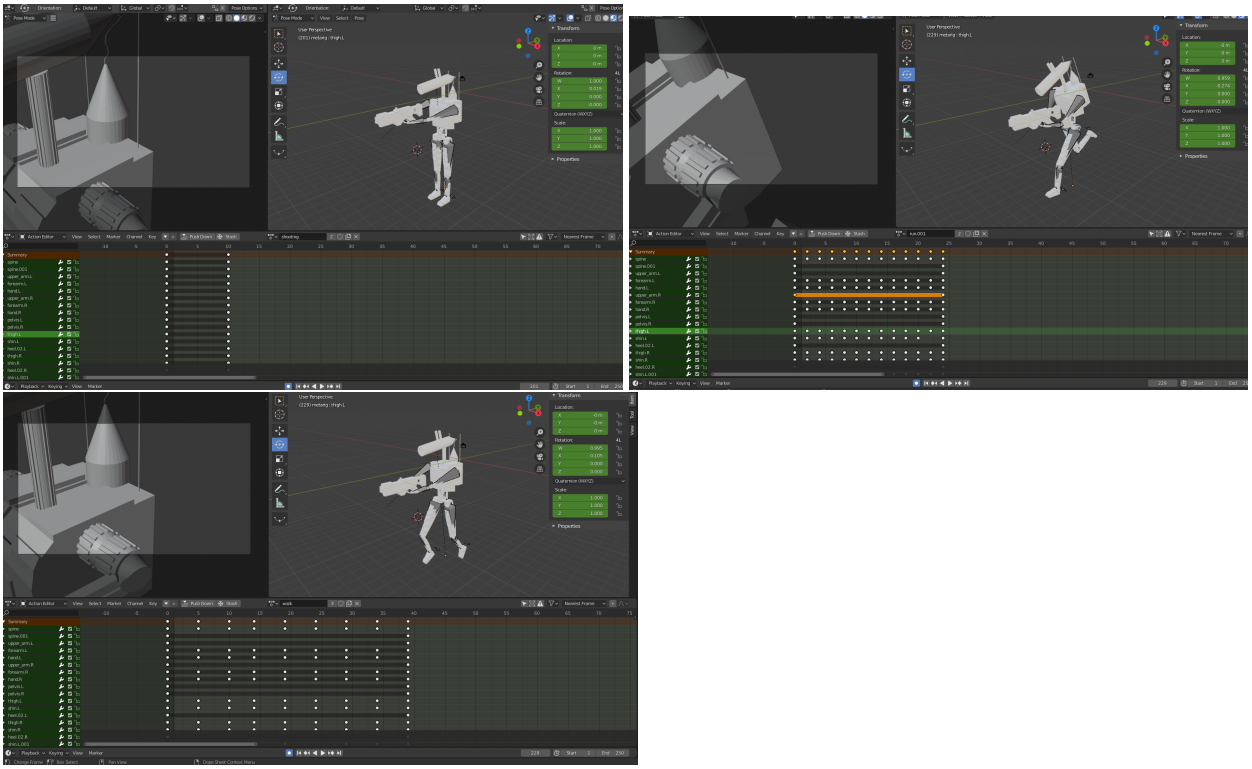
2.3 Main Character, Bot and Animations

Main Character

previously we have showed for the first time the main character of the game. The main character is a robot and it is inspired from the battle droids from Star Wars franchise.
We have not modified the model of the character, but we have modified its animations.

we have attached a gun to the character and as a consequence , we needed to modify the walking and running animations and add another animation as well , which is the standing animation, in which the player is just standing and holding a gun.

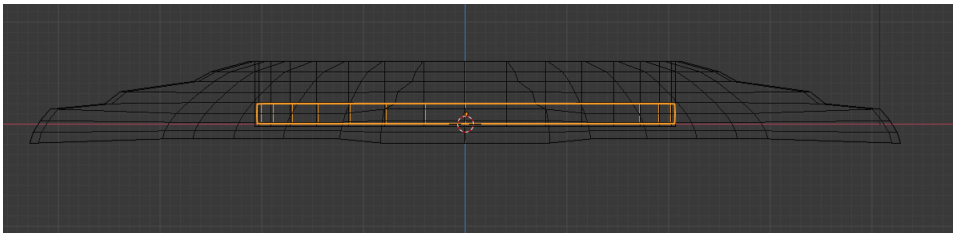
This is the result :

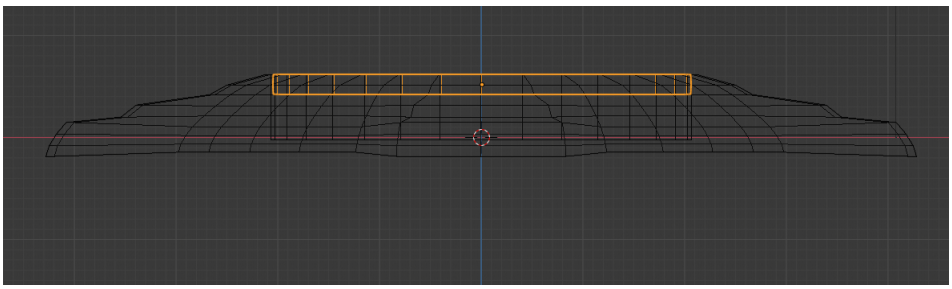


The first picture is the standing animation. The second one is the running animation. The third one is the walking animation.
Animations

We made few animations that we have used in the game: The pressure point animation and the door animation The pressure point animations were made on blender. The door animations were made on Unity

The pressure point is a platform that allows the player to open a door by standing on that pressure point or by placing an object onto it.
We have two different animations for the pressure point : the pressing animation (first picture) and the release animation (second picture)

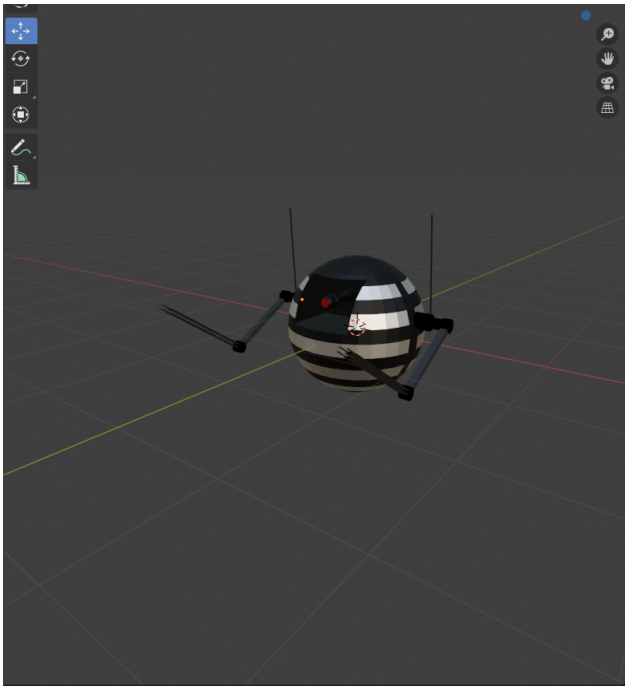




Regarding the door, the model was downloaded from unity asset store (with textures) and we just added the "open door" animation and "closed door" animation.

Bot

We also designed the model of the bot, and here it is :

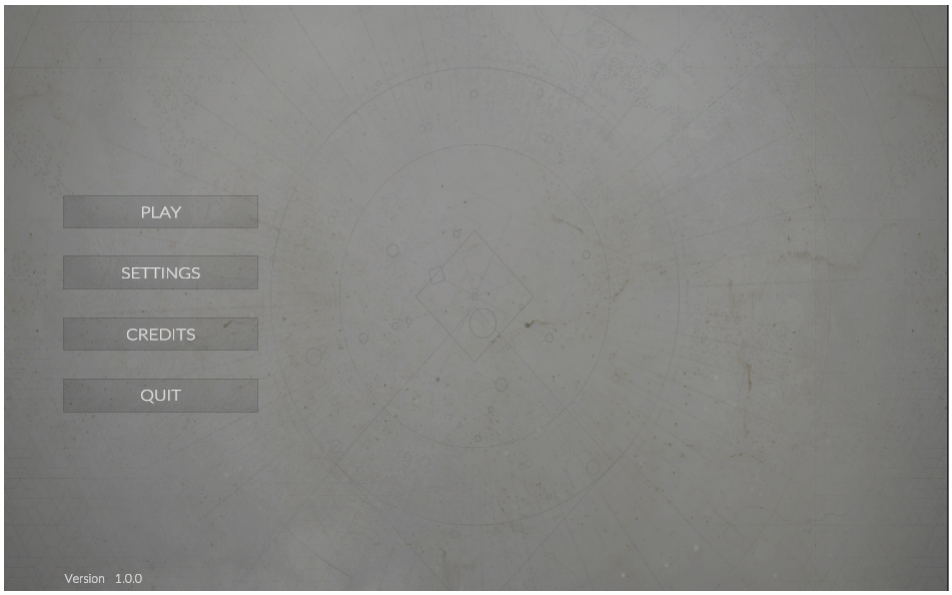


2.4 UI

Main Menu

Previously we have implemented basic functionalities of the main menu. Now we have changed drastically the menu from the previous one, and here it is :

The image above is the "welcome" panel, where the player need to "PRESS ANY KEY" in order to go to the main menu :

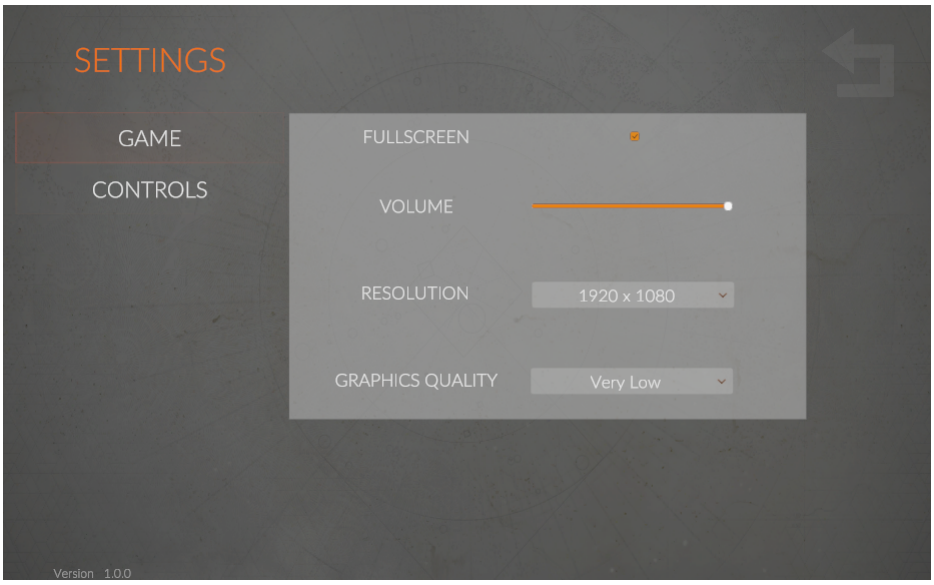


In this menu, the player has a choice to click on 4 different buttons :

1 - **PLAY BUTTON** : It allows the player to play the game

2 - **Settings Button** : It allows the player to access the settings and change the settings

Here is the settings panel :



In the settings panel, the player can change the game settings (fullscreen,resolution,game quality,volume)

3 - **CREDITS BUTTON** :Here the player can see who are the authors of the game

Last thing about the main menu: the transition between different sub-menus (settings menu, credits ..) are made by using animations.

Pause menu

The Pause Menu is really simple, whenever the player presses **P**, the pause menu, through an animation, it appears or disappears (it depends whether the pause menu is open or not). In the pause menu, the player can either resume or quit (to main menu or desktop)

Here is the pause menu :

other menus

Finally, we have two different menus that appear in the game, depending on the circumstances. The menus are "Die Menu"(that appears whenever the player dies) and "Win menu" (that appears when the player finishes the game).

Remarks : all the menus have background music (except the "win menu") and all the buttons of the menus play a "click sound" whenever they are clicked (except the "die menu" and "win menu")

2.5 Sound and particle effects

we have implemented shooting sound effect, button-click sound effect(just on the main menu and pause menu). We also implemented three particle

effects, the smoke particle effect, gun shoot particle effect, and enemy hit particle effect (Whenever the player shoots to a bot, the sparks comes out of the bot).

2.6 User Interface and HUD

A pause menu was added this time around as a part of the UI with fully functional buttons.
Code

```
public class PauseMenuScript : MonoBehaviour
{
    public GameObject PauseMenuUI;

    private bool isPaused = false;
    // Update is called once per frame
    void FixedUpdate()
    {
        if (Input.GetButton("Cancel"))
        {
            isPaused = !isPaused;
            if (isPaused)
            {
                Pause();
            }
            else
            {
                UnPause();
            }
        }
    }

    public void Pause()
    {
        isPaused = true;
        Time.timeScale = 0;
        PauseMenuUI.SetActive(true);
        AudioListener.pause = true;
    }
    public void UnPause()
    {
        isPaused = false;
        Time.timeScale = 1;
        PauseMenuUI.SetActive(false);
        AudioListener.pause = false;
    }

    public void Quit()
    {
        Application.Quit();
    }
}
```

2.7 Gameplay

If you remember, we talked about a surprise which binds the whole project. In this report, we are going to talk about that surprise in detail.
THE PORTALS.

Before jumping on that, we wanna make it clear, the Power Up, we talked about in the earlier defence, that was needed to be scrapped out because it did not have a utility in the whole game.

Instead of that, our game includes two characters. Player1: The one who has the Portal gun and Player2: The one who has the gun for killing bots. Together the players have to work together, to solve the puzzle and get out of the map safely and securely. Now this was the multiplayer version.

Coming on to the single player version, the map is similar. But animations of certain things are tweaked. You can look it for yourself in the project.

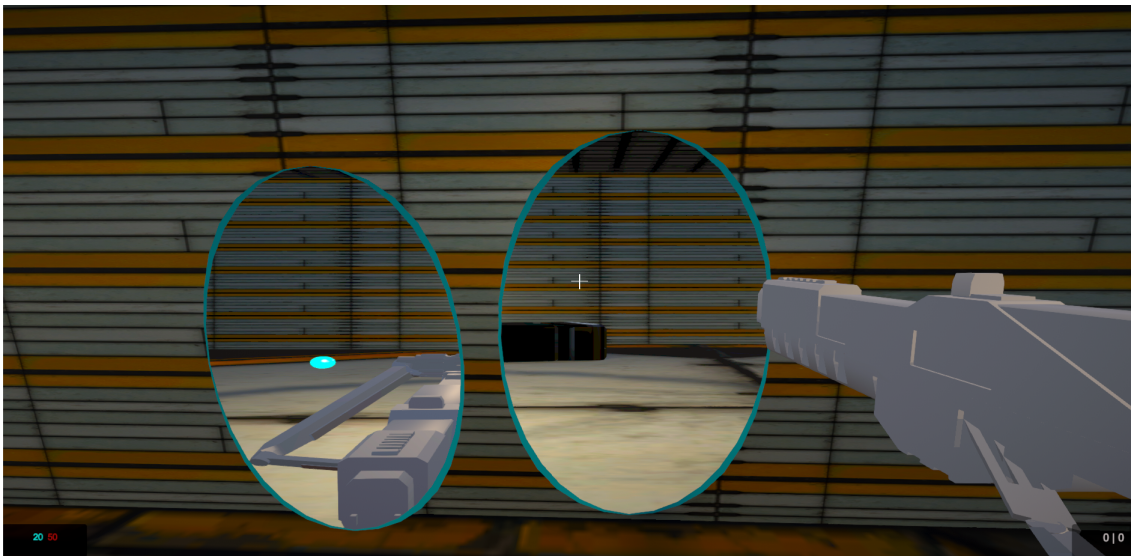
Along with all this, we have implemented a mechanism where we can pick up certain objects and place it on Pressure Points to open a door.

We also implemented a gun change manager for single player version. Although we didn't get the time to animate, it still works quite well. From last defence, we have also improved Mouse Look mechanism of the player. Now the players cannot move left and right. It can only go forward and backwards, but this is relative to the camera direction.

All this will be talked about in the following few pages.

2.7.1 The Portals

The Portals. We took an inspiration from the game called the Portals., where the transform of the player is changed whenever it collides with a portal. The animation will be shown in the video. Portals have Cameras attached to it. Something like this.



So when the the player who has the Portal gun shoots, these Portals appear and both the Player1 and Player2, can go through it.

The concept is pretty simple, when a player collides, the position of the player is being changed to the portals position, therefore giving an illusion that, the Player is being teleport-ed.

Left mouse button throws a Left Portal, and Right Mouse Button throws Right Portal.

Throw Portal Code

```
 void Update()
{
    if (Input.GetMouseButtonDown(1))
        throwPortalLeft(leftPortal);


    if (Input.GetMouseButtonDown(0))
        throwPortalRight(rightPortal);

    if (turnOnLeft && turneOnRight)
    {
        mesh1.enabled = true;
        mesh2.enabled = true;
        leftPortal.GetComponent<StepThroughPortal>().enabled = true;
        rightPortal.GetComponent<StepThroughPortal>().enabled = true;
    }

}

void throwPortalLeft(GameObject portal)
{
    int x = Screen.width / 2;
    int y = Screen.height / 2;

    GunSound.Play();
    muzzleFlash.Play();
```

```
Ray ray = GetComponent<Camera>().ScreenPointToRay(new Vector3(x, y));
RaycastHit hit;
if (Physics.Raycast(ray, out hit))
{
    Collider = hit.collider;

    if (!turnOnLeft)
    {
        if (Collider.CompareTag("Ground"))
        {
            if (CanmakeOnGroundLeft)
            {
                Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                portal.transform.position = hit.point;
                portal.transform.rotation = hitObjectRotation;
                CanmakeOnGroundRight = false;
                turnOnLeft = true;
            }
        }

        else if (Collider.CompareTag("Roof"))
        {
            if (CanmakeOnRoofLeft)
            {
                Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                portal.transform.position = hit.point;
                portal.transform.rotation = hitObjectRotation;
                CanmakeOnRoofRight = false;
                turnOnLeft = true;
            }

        }
        else if (Collider.CompareTag("Wall"))
        {
            Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
            portal.transform.position = hit.point;
            portal.transform.rotation = hitObjectRotation;
            CanmakeOnGroundRight = true;
            CanmakeOnRoofRight = true;
            turnOnLeft = true;
        }
        else
        {
            Debug.Log("Cannot Create Portal on that Object");
        }

    }
    else
    {
        if (Collider.CompareTag("Ground"))
        {
            if (CanmakeOnGroundLeft)
            {
                Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                portal.transform.position = hit.point;
                portal.transform.rotation = hitObjectRotation;
                CanmakeOnGroundRight = false;
            }
        }

        else if (Collider.CompareTag("Roof"))
        {
            if (CanmakeOnRoofLeft)
            {
                Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
```

```
                    portal.transform.position = hit.point;
                    portal.transform.rotation = hitObjectRotation;
                    CanmakeOnRoofRight = false;
                }

            }
            else if (Collider.CompareTag("Wall"))
            {

                Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                portal.transform.position = hit.point;
                portal.transform.rotation = hitObjectRotation;
                CanmakeOnGroundRight = true;
                CanmakeOnRoofRight = true;
            }
            else
            {
                Debug.Log("Cannot Create Portal on that Object");
            }
        }
    }
}


void throwPortalRight(GameObject portal)
{
    int x = Screen.width / 2;
    int y = Screen.height / 2;

    GunSound.Play();
    muzzleFlash.Play();

    Ray ray = GetComponent<Camera>().ScreenPointToRay(new Vector3(x, y));
    RaycastHit hit;
    if (Physics.Raycast(ray, out hit))
    {
        Collider = hit.collider;
        if (!turneOnRight)
        {
            if (Collider.CompareTag("Ground"))
            {
                if (CanmakeOnGroundRight)
                {
                    Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                    portal.transform.position = hit.point;
                    portal.transform.rotation = hitObjectRotation;
                    CanmakeOnGroundLeft = false;
                    turneOnRight = true;
                }
            }

            else if (Collider.CompareTag("Roof"))
            {
                if (CanmakeOnRoofRight)
                {
                    Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                    portal.transform.position = hit.point;
                    portal.transform.rotation = hitObjectRotation;
                    CanmakeOnRoofLeft = false;
                    turneOnRight = true;
                }

            }
            else if (Collider.CompareTag("Wall"))
            {
```

```
                Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                portal.transform.position = hit.point;
                portal.transform.rotation = hitObjectRotation;
                CanmakeOnGroundLeft = true;
                CanmakeOnRoofLeft = true;
                turneOnRight = true;
            }
            else
            {
                Debug.Log("Cannot Create Portal on that Object");
            }


        }
        else
        {
            if (Collider.CompareTag("Ground"))
            {
                if (CanmakeOnGroundRight)
                {
                    Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                    portal.transform.position = hit.point;
                    portal.transform.rotation = hitObjectRotation;
                    CanmakeOnGroundLeft = false;
                }
            }

            else if (Collider.CompareTag("Roof"))
            {
                if (CanmakeOnRoofRight)
                {
                    Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                    portal.transform.position = hit.point;
                    portal.transform.rotation = hitObjectRotation;
                    CanmakeOnRoofLeft = false;
                }
            }
            else if (Collider.CompareTag("Wall"))
            {
                Quaternion hitObjectRotation = Quaternion.LookRotation(hit.normal);
                portal.transform.position = hit.point;
                portal.transform.rotation = hitObjectRotation;
                CanmakeOnGroundLeft = true;
                CanmakeOnRoofLeft = true;
            }
            else
            {
                Debug.Log("Cannot Create Portal on that Object");
            }
        }

    }
```

We know the code looks big. But there are few cases which needed to be handled. First of all, we all know almost everything has to be a collider to detect collisions. So, to make portals on specific grounds and Walls, we had to use Tags.

NOTE: You cannot create two Portals on the Ground or on the Roof, because of the Rotation problems. We faced incredible problem in fixing the

rotation of the player. That's why we had to create some boundation. To be honest, we really don't know the exact solution for this.

Now lets talk about how the the Players step through the Portals. The concept is simple. It checks the the angles b/w the Portals, if they are in 180 degrees apart. The Player rotation is not changed. If they are not, the rotation of the player is changed.

Step Through Portals Code

```
public GameObject otherPortal;

private float angle;
private float angle1;
private bool _overLappingPlayer = false;
private bool _overlappingObject = false;
private Collider Player;
private Collider pickUpObject;


[SerializeField] private float throwForce;

void FixedUpdate()
{
    angle = Vector3.Angle(transform.forward, otherPortal.transform.forward);

    if (_overLappingPlayer)
    {
        var transform1 = Player.transform;
        angle1 = Vector3.Angle(otherPortal.transform.eulerAngles, transform1.eulerAngles);


        if (Mathf.Round(angle) == 90f)
        {
            if (70f <= angle1 && angle1<= 95f)
            {
                if (Mathf.Round(Vector3.Angle(transform1.up, otherPortal.transform.forward)) == 0f)
                {

                    Debug.Log("Illegal Movement");
                }
                else
                {
                    if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
                    {
                        var forward = otherPortal.transform.forward;
                        transform1.SetPositionAndRotation(otherPortal.transform.position + forward * 5, Quaternion.Normalize(transform1.rotation));

                    }
                    else if (Input.GetKey(KeyCode.S)||Input.GetKey(KeyCode.DownArrow))
                    {
                        var back = -otherPortal.transform.forward;
                        transform1.SetPositionAndRotation(otherPortal.transform.position  - back * 7, Quaternion.Normalize(transform1.rotation));

                    }
                }
            }
            else
            {
                if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
                {
                    var forward = otherPortal.transform.forward;
                    transform1.SetPositionAndRotation(otherPortal.transform.position + forward * 5, Quaternion.LookRotation(forward));

                }
                else if (Input.GetKey(KeyCode.S)||Input.GetKey(KeyCode.DownArrow))
                {
                    var back = -otherPortal.transform.forward;
                    transform1.SetPositionAndRotation( otherPortal.transform.position - back * 7, Quaternion.LookRotation(back));
                }
            }

        }

        else if (Mathf.Round(angle) == 180f)
        {
            if (70f<=angle1 && angle1<=92f)
            {
                if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
                {
                    var forward = otherPortal.transform.forward;
                    transform1.SetPositionAndRotation(otherPortal.transform.position + forward * 5, Quaternion.Normalize(transform1.rotation));
                }
```

```
                else if (Input.GetKey(KeyCode.S)||Input.GetKey(KeyCode.DownArrow))
                {
                    var back = -otherPortal.transform.forward;
                    transform1.SetPositionAndRotation(otherPortal.transform.position - back * 7, Quaternion.Normalize(transform1.rotation));
                }
            }
            else
            {
                if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
                {
                    var forward = otherPortal.transform.forward;
                    transform1.SetPositionAndRotation(otherPortal.transform.position + forward * 5,Quaternion.LookRotation(forward));
                }
                else if (Input.GetKey(KeyCode.S)||Input.GetKey(KeyCode.DownArrow))
                {

                    var back = -otherPortal.transform.forward;
                    transform1.SetPositionAndRotation(otherPortal.transform.position - back * 5, Quaternion.LookRotation(back));
                }
            }
        }
        else
        {
            if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow))
            {
                var forward = otherPortal.transform.forward;
                transform1.SetPositionAndRotation(otherPortal.transform.position + forward * 5, Quaternion.LookRotation(forward));
            }
            else if (Input.GetKey(KeyCode.S)||Input.GetKey(KeyCode.DownArrow))
            {
                var back = -otherPortal.transform.forward;
                transform1.SetPositionAndRotation(otherPortal.transform.position - back * 5, Quaternion.LookRotation(back));
            }
        }

        _overLappingPlayer = false;

    }

    if (_overlappingObject)
    {
        var transformObj = pickUpObject.transform;
        var forward = otherPortal.transform.forward;
        transformObj.position = otherPortal.transform.position + forward * 5;
        transformObj.GetComponent<Rigidbody>().AddForce(forward * throwForce);
        _overlappingObject = false;
    }
}

void OnTriggerEnter(Collider other)
{

    if (other.CompareTag("Player1"))
    {
        _overLappingPlayer = true;
        Player = other;
    }

    if (other.CompareTag("Player2"))
    {
        _overLappingPlayer = true;
        Player = other;
    }

    if (other.CompareTag("PickUpObject"))
    {
        _overlappingObject = true;
        pickUpObject = other;

    }
}

void OnTriggerExit(Collider other)
{
    if (other.CompareTag("Player1"))
    {
        _overLappingPlayer = false;
    }

    if (other.CompareTag("Player2"))
    {
        _overLappingPlayer = false;
    }

    if (other.CompareTag("PickUpObject"))
    {
        _overlappingObject = false;
    }

}
```

2.7.2 Animation Controllers

There are quite a few animations involved in our game. From small particle effect to door animations, everything has been implemented which we thought would make our game more immersive.

Getting muzzle flash and particle effect wasn't difficult. It was available online. And wasn't hard to code either. Just a code line like this did the trick.

```
gameObject.Play();
```

Door Animations, Pressure Points and Character animations were something which needed coding. Mr.Bekhat(The Game Designer), provided us with the animations. But coding them to get triggered at the right time is another thing.

Door Animation Code

```
private Animator animator;
private bool IsTriggered;
private bool IsDoor1;

void Start()
{
   if (this.gameObject.CompareTag("Door1"))
   {
      IsDoor1 = true;
   }
   animator = GetComponent<Animator>();
   IsTriggered = false;
}

void OnTriggerEnter(Collider other)
{
   if (other.CompareTag("PickUpObject"))
   {
      IsTriggered = true;
      DoorControl("Open");
   }

   if (other.CompareTag("Player1"))
   {
      IsTriggered = true;
      DoorControl("Open");
   }
    if (other.CompareTag("Player2"))
   {
      IsTriggered = true;
      DoorControl("Open");
   }

}

void OnTriggerExit(Collider other)
{
   if (!IsDoor1)
   {
      if (IsTriggered)
      {
         IsTriggered = false;
         DoorControl("Close");
      }
   }

}


void DoorControl(string dir)
{
   animator.SetTrigger(dir);
}
```

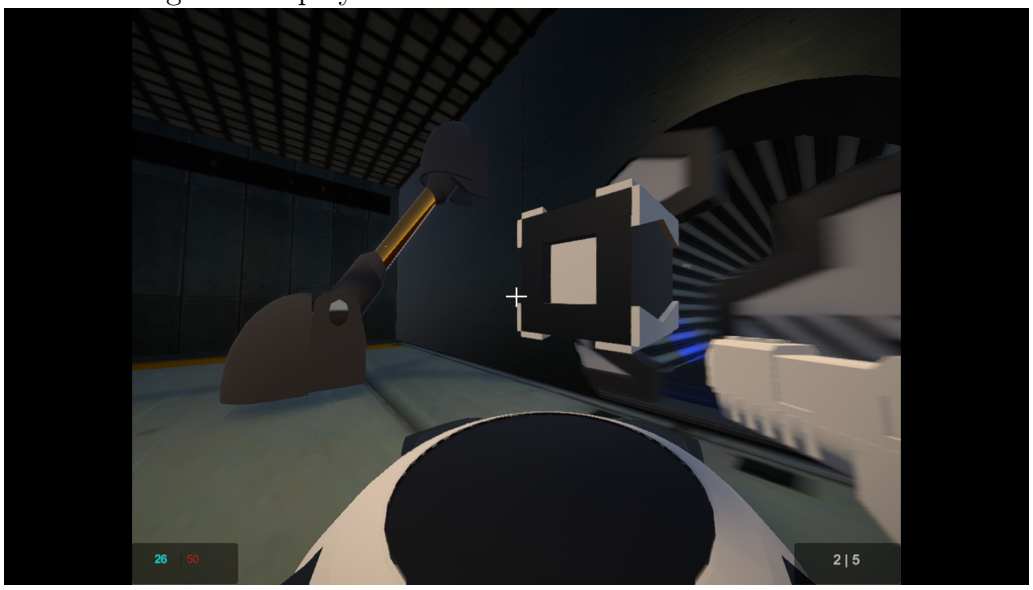
Pressure Point animation Code

```
 Animator animator;

private bool IsPressed;
// Start is called before the first frame update
void Start()
{
    IsPressed = false;
    animator = GetComponentInParent<Animator>();
}

// Update is called once per frame
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("PickUpObject"))
    {
        IsPressed = true;
        ButtonPressed("PressDown");
    }

    if (other.CompareTag("Player1"))
    {
        IsPressed = true;
        ButtonPressed("PressDown");
    }
    if (other.CompareTag("Player2"))
    {
        IsPressed = true;
        ButtonPressed("PressDown");
    }
}

void OnTriggerExit(Collider other)
{
    if (other.CompareTag("PickUpObject"))
    {
        IsPressed = false;
        ButtonPressed("NoPress");
    }
    if (other.CompareTag("Player1"))
    {
        IsPressed = true;
        ButtonPressed("NoPress");
    }
    if (other.CompareTag("Player2"))
    {
        IsPressed = true;
        ButtonPressed("NoPress");
    }
}



void ButtonPressed(string dir)
{
    animator.SetTrigger(dir);
}
```

As you can see, Colliders are being used to detect the collisions and acting on it. A glimpse of the animation will be shown in the video presentation.

Same is done for every other object which needs to be animated.

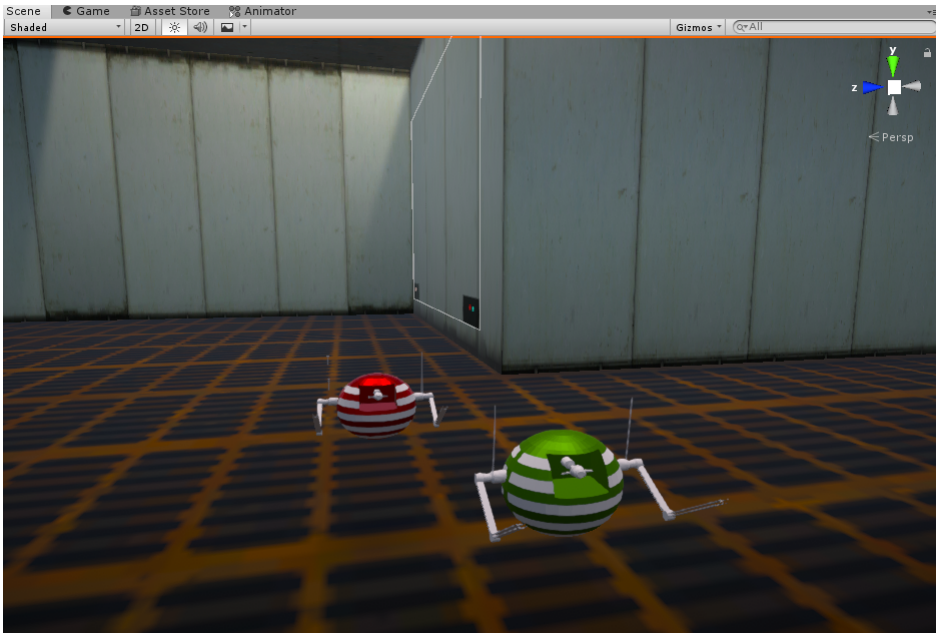
We also modified few things from the initial AI provided by Mr.John. We assign Bots to specific players to have better Bot Control. We have 3 kinds of Enemy Bots. THE STANDARDs and the ALPHA version. These bots are implemented in such a way and animated in such a way that they pose optimal challenge to the player.



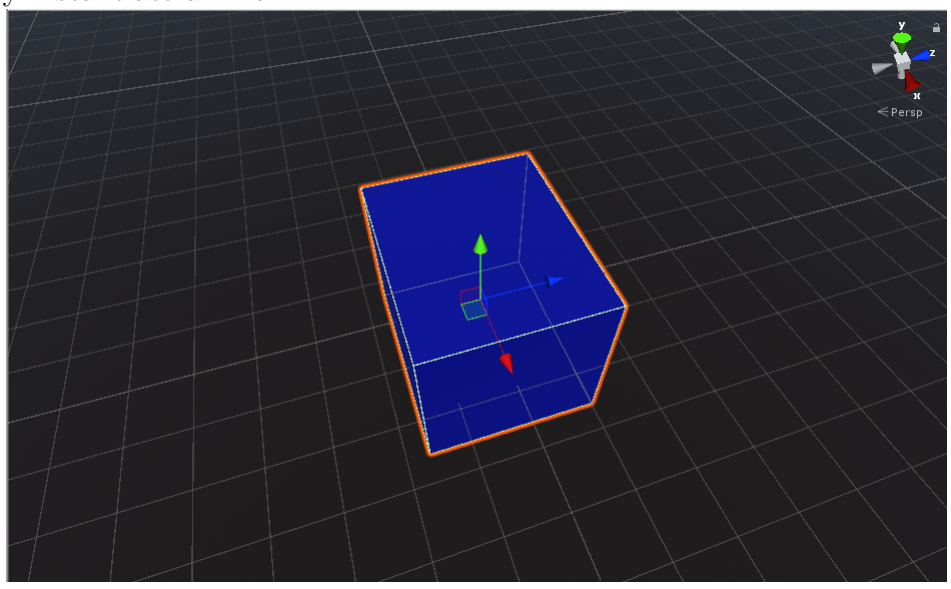
As you can see in the picture, there is pick up object mechanism, when you drop the cube on that pressure point, Door opens and closes up. A toggling affect.

2.7.3 How The Game Works

Multiplayer Version: Two players are spawned into the map. As you go through the map, you will face obstacles in terms of Enemy Bots.

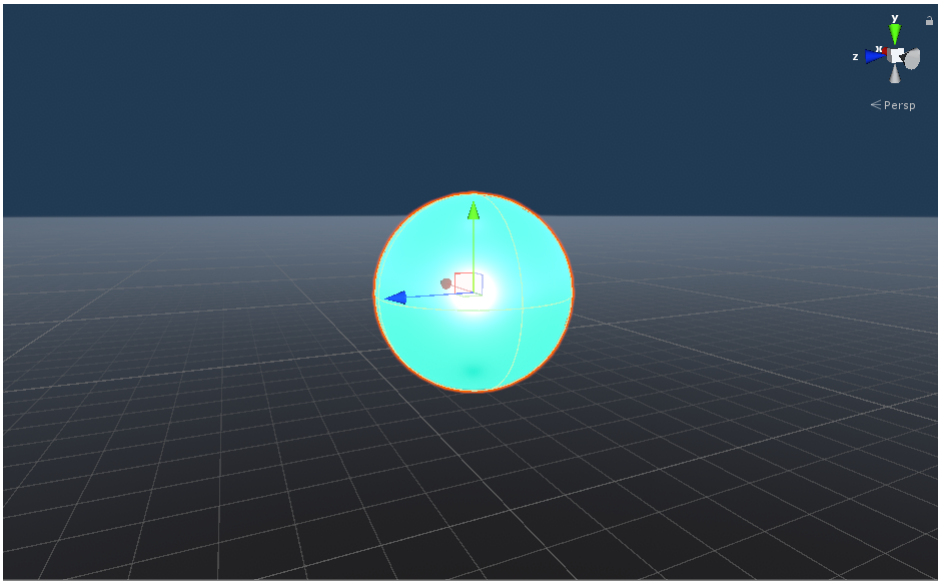


These two bots are standard version as said before. Both the player should work as a tag team, to kill these bots, and escape them into oblivion. There is a reward in terms of Loot attached to these bots. When you kill red Bots, they instantiate ammo.

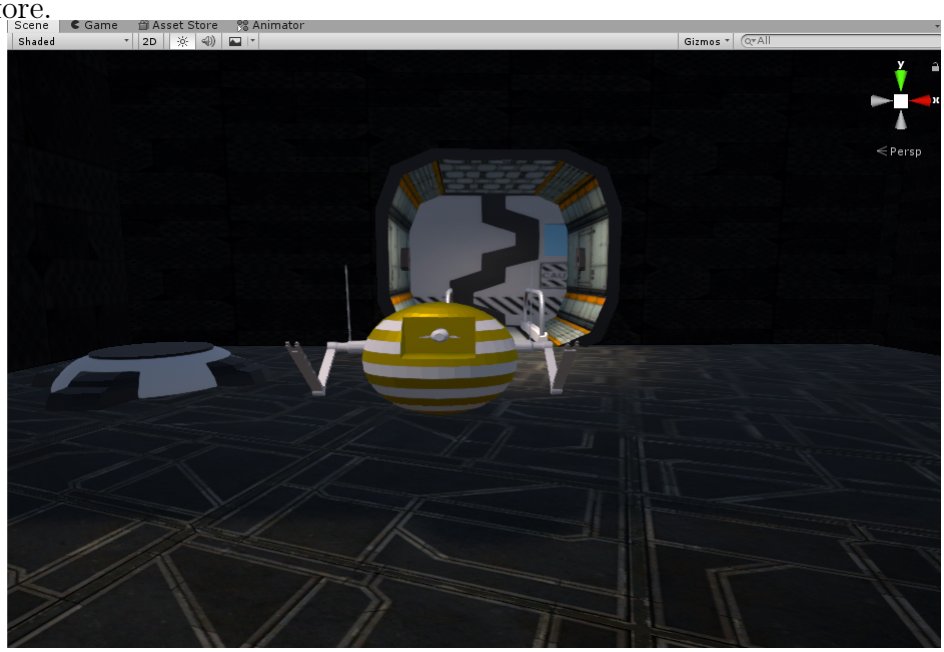


Players will also find ammos spread across in the map, to have extra help. But the catch is, only Player2 is allowed to collect ammo in Multiplayer mode.

When you kill green Bots, they instantiate Shields



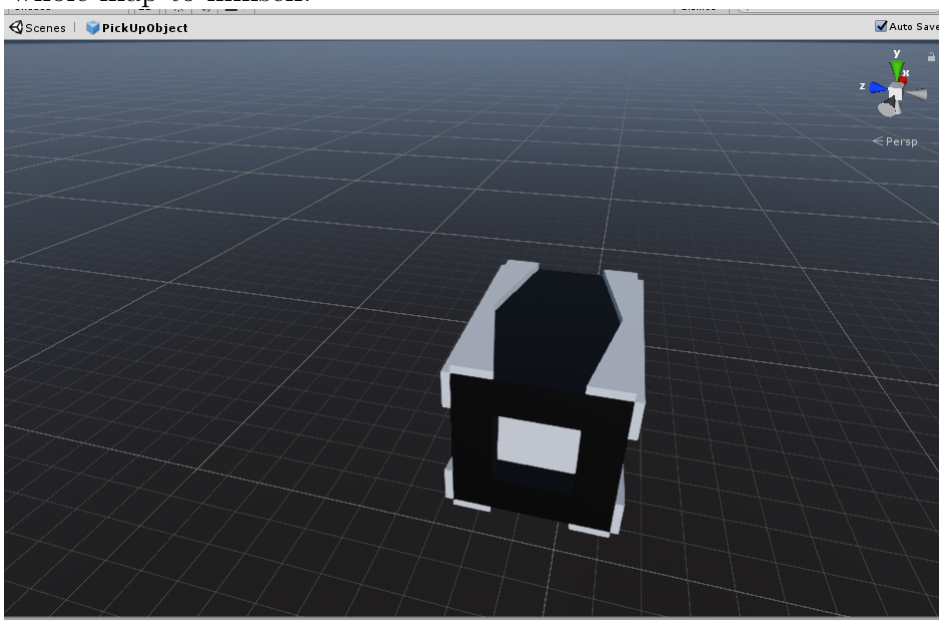
These Loots are there to help the players, to face the wrath which is dying to stand upon them.(Kidding) Game is not that hard, if you have played FPS before.



This Bot is Alpha version, more health, more holding power. Much harder

to kill. But there is a sweet reward for killing this bot. The Player will get extra ammos and the shields.

Single Player Version: It's same as before, just the number of Enemy Bots are reduced and more Objects are added to help the player. In this version, the player will have both Portal Gun, and the killer Gun. You can toggle it using Right Shift. The Player can collect both the shield and object. He has the whole map to himself.



This is the Object the Player1 can pick up. Player2 doesn't have the freedom to pick up objects in Multiplayer Mode.

2.7.4 Game mechanics

This time around, the mechanics have been changed but not to a great extent as to not let unnecessary content to affect the quality of our work.

A loot system as been added to finish puzzles in the game as now enemies will drop anything ranging from some ammo or a crucial piece to finish the puzzle

Small adjustments were to the 'bothealth' script to make sure that as soon as an enemy died, loot was instantiated at that specific point.

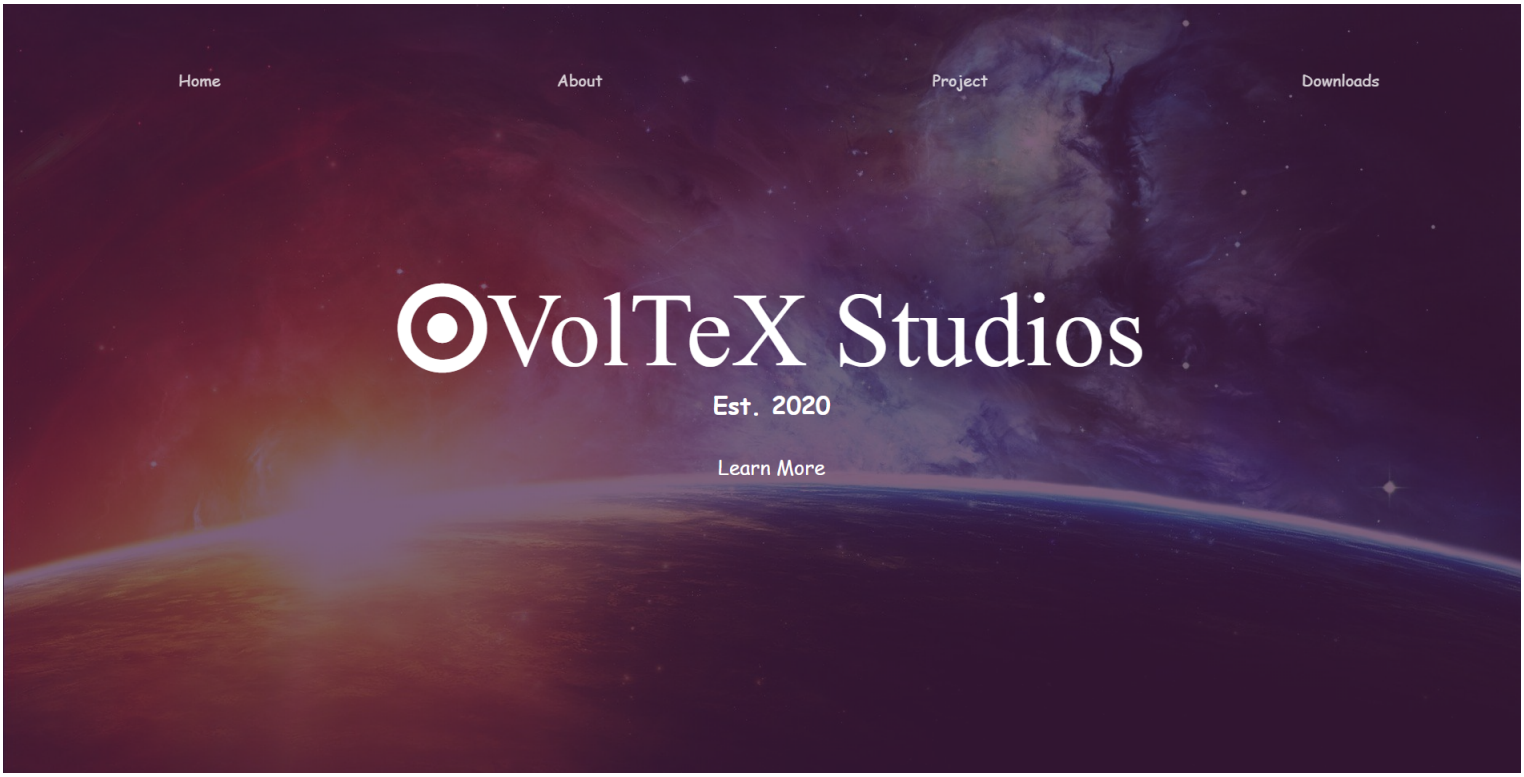
Code

```
public int MaxHealth;
   private int health;
   public GameObject Loot;
   public GameObject Shield;
   public GameObject HealthCanvas;
   public Slider slider;
   public ParticleSystem damage_effect;
   void Start()
   {
       health = MaxHealth;
       slider.value = CalculateHealth();
   }

   void dmginflict(int dmg)
   {
       health -= dmg;
       damage_effect.Play();
   }

   void Update()
   {
       if (health < MaxHealth)
       {
           HealthCanvas.SetActive(true);
       }
       slider.value = CalculateHealth();

       if (health <= 0)
       {
          dead();
       }

   }

   void dead()
   {

       if (this.CompareTag("BotType1"))
       {
           Instantiate(Loot, transform.position, Quaternion.identity);
           Destroy(this.gameObject);
       }

       if (this.CompareTag("BotType2"))
```

```
        {
            Instantiate(Shield, transform.position, Quaternion.identity);
            Destroy(this.gameObject);
        }

        if (this.CompareTag("BotType3"))
        {
            Instantiate(Loot, transform.position + Vector3.right * 5, Quaternion.identity);
            Instantiate(Shield, transform.position - Vector3.right * 5, Quaternion.identity);
            Destroy(this.gameObject);
        }

    }

    float CalculateHealth()
    {
        return (float)health / MaxHealth;
    }
```

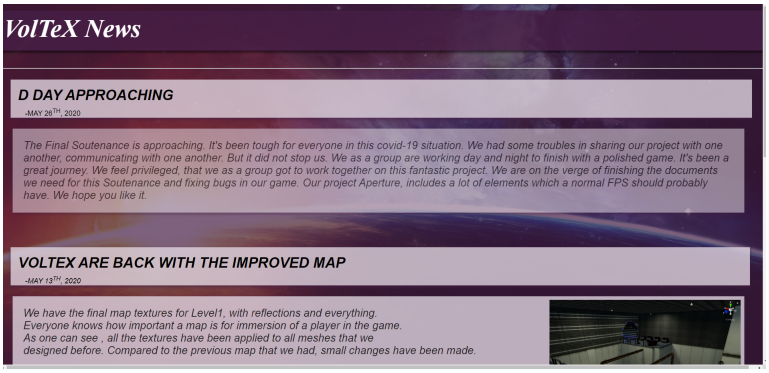
As one can see, a slider property has been added. This has been explained in the UI section.

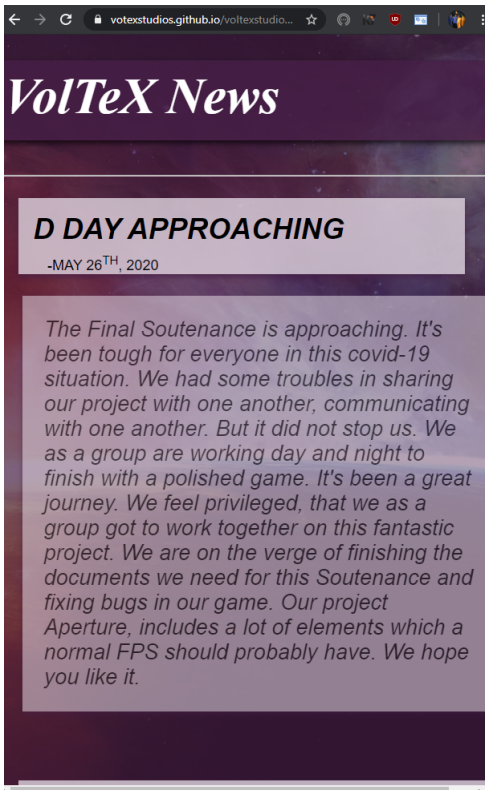
2.8 Website


Home
About
Project
Downloads
VolTeX Studios
Est. 2020
Learn More

Website has been updated drastically. Earlier, it was only Home Page and few other pages were updated, but now, all the pages are updated. And has been worked in such a way that it is responsive.

Some Snippets from the Website





As you can see the news section of our website is working quite well with the change of pixels.

Same can be said about Project section. We have also added "Good Luck" button where you can download the build of the project.

We hope you like it.

2.8.1 Link

We are using GitHub pages for it to be accessible to our jury. The link : https://votexstudios.github.io/voltexstudios/

2.8.2 Components of the Website

The Home Page has 4 sections:

- Home: Takes one to the Home Page
- About:
  - Our Team: One can gather information of our team members. There is a button on the page, which leads to our respective emailID. One can contact us if they want.
  - News: One can get the information about our progress.
- Project: Aperture, one can gather information about our Game.
- Downloads: One can download the BookofSpecs, Report1, Report2, and Report3

2.9 Network

For this defense we focused on stabilizing the network since the network, fixing a lot of bugs and despite the endless hours we spent doing this, there are still many things we wished we could have fixed.

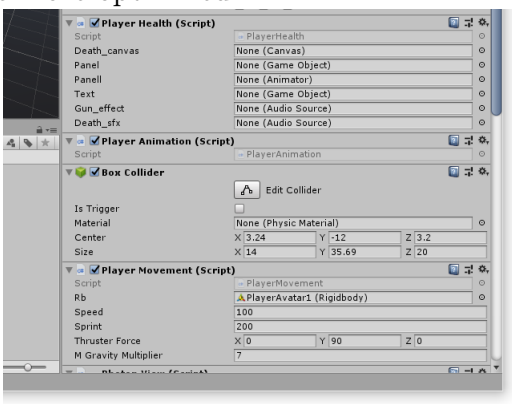
The network part was mostly finished. Most of time was spent converting a Single Player version of the game into a Multiplayer experience but doing so proved extremely tedious. We essentially had to merge manually the network version of the game with an incompatible single version and lost a tremendous amount of time doing so. Most of our work went into fixing these issues, dealing with merge conflicts and doing so for multiple iterations. Not to mention it was difficult to find good documentation or tutorials for the Photon Engine, so our researching skills had to be sharp.

2.9.1 Animation Synchronization

We initially sought to solve this problem by building an animate function. One that would trigger not with movement but with the animation itself, unfortunately the movement had already been written and the animation

aswell in terms of the movement so we couldn't implement this concept.

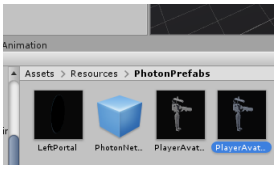
The solution we ended up implementing was the PhotonAnimatorView component where you get fine control over which parts of the animation you want to synchronize. And then placing the component within the PhotonView component. Ultimately this method proved useful though most likely the former method was more optimized.



2.9.2 Different Players

Knowing there would be two very different players in the game, we decided to duplicate the current prefab and and modify the different children so that it would fit the player. That's why we had two Prefabs Player Avatar1 and Player Avatar2, this would also prove useful for distinguishing the players (each player was a different objects and had different tags, "Player1" and "Player2" respectively)

The first player, or MasterClient will always take the place of PlayerAvatar1 which has the portal. The second player, (not MasterClient) will always be instantiated as PlayerAvatar2. This allows for unique customization of both characters.



There was a more optimal solution which would have been to use prefab variations. But considering we were very inexperienced when it came to Unity and Photon, we opted for the Two separate prefab solution.

2.9.3 Bots

Considering everything had been implemented in Single Player we were very worried if the bots would even function in a scene where multiple players subsided. Ultimately the solution we found was to give every individual Player a Tag. Have two separate scripts to choose from which would focus on the object with tag "Player1" or for the other script "Player2".

There was also PhotonView component to the Bots so that they synchronize on both screens, this wasn't necessary to some extent since the bots would react to the presence of the object regardless, but there would be some deviation between the two.

Finally a problem with the health script arose which uses the method Destroy, which unfortunately is only local, a solution we tried to implement was PhotonView.Destroy(). But in order to use this we would need to be MasterClient, and unfortunately the gunner is not the MasterClient. Between the creation of the health script and the deadline there were far more important things to stabilize/fix such as portals and it's somewhat regretful.

Again a better solution would have been to use Prefab variations, but even so the AI would need to be able to to react to multiple players.

2.9.4 Portals

The Portals presented a serious problem. Fortunately we were able to make them function completely. There was an issue with accessing the portals from the prefab. Since we could not assign them to the player form within the inspector, and since the portals were being traced using PhotonView, we opted for finding the player with PhotonView.Find(PhotonID) (The PhotonID is manually set if it's in the scene) something we found out much later.

```
{
 if (gameObject.transform.CompareTag("LeftPortal"))
        {
            otherPortal = PhotonView.Find(997).gameObject.transform;
        }

        if  (gameObject.transform.CompareTag("RightPortal"))
            otherPortal = PhotonView.Find(998).gameObject.transform;
}
```

and

```
{
void Start()
{
    leftPortal = PhotonView.Find(998).gameObject;
    rightPortal = PhotonView.Find(997).gameObject;
    mesh1 = leftPortal.GetComponent<MeshRenderer>();
    mesh2 = rightPortal.GetComponent<MeshRenderer>();
    mesh1.enabled = false;
    mesh2.enabled = false;
    leftPortal.GetComponent<StepThroughPortal>().enabled = false;
    rightPortal.GetComponent<StepThroughPortal>().enabled = false;
}
}
```

On top of the two portals are being tracked using photon view which allows for both players to see and traverse the portals. No portals are actually instantiated,instead two prexisting portals within the scene are being moved from wall to wall.

2.9.5 PickUpObject

Similar to before, since the player was not already present in the scene, we needed to find a way to apply the TempParent to every object. TempParent is an object attached to our prefab PlayerAvatar1, here could not use Photon View to solve issue instead we opted for

```
tempParent =  GameObject.Find("TempParent");
```

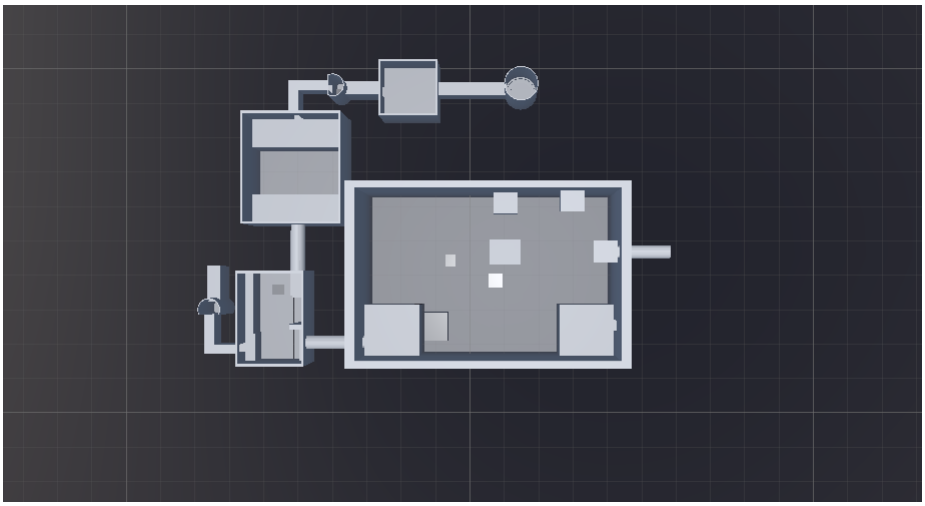
Photon View has also been applied to every single cube on the map so that if it's moved by player 1, player 2 can follow it's movement and more importantly the pressure pads on the floor get activated when the cube gets dropped on them. This causes each client to localy open the door, therefore no need to use PhotonViewAnimator for these two.

3 Project Summary

We have found this project to be a really interesting one and we have learned so many skills that will be useful for us later on in life. Below is a summary of each part of our project showcasing the progress we have made start to finish.

3.0.1 Level Design

Since the beginning of the project, we were really exited about it and scared as well, because we didn't have much knowledge on Unity and Blender. During the early days of the project, we spent a lot of time on watching videos and reading tutorials about 3D modelling and principles of game design. we preferred "baby steps", which means learning the concepts slowly and make gradual progress. We were not sure on how to design the map, we had many prototypes. After some thinking, we built the first prototype of the map



we were proud of the result that we have achieved, But still, somehow it wasn't enough. There were some problems. For instance the rooms were not perfectly connected, the rendering of the rooms was not great (the rooms were clinking). we immediately started to look for a solution. After a long research, we found the solution of the problems.

After we have finished modelling the level 1, we immediately started working on the textures and it was not an easy task as we expected. it took us nearly two weeks in order to understand the so called texture mapping. In fact , now , the textures that are applied on the models are gorgeous and

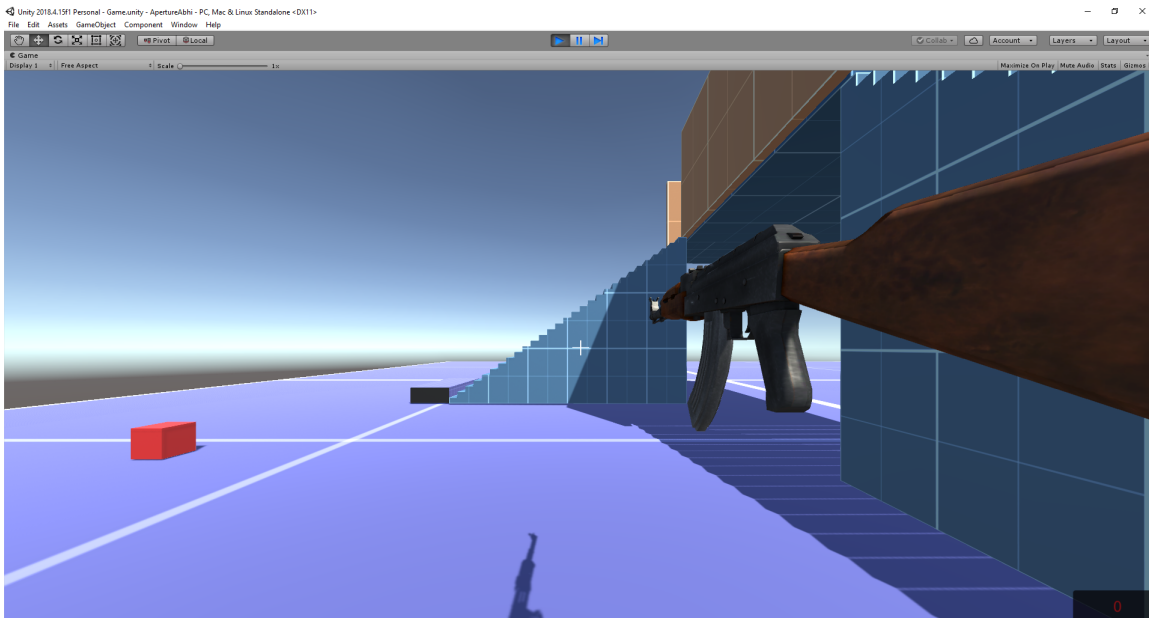
perfectly applied. After that we started thinking about how the player will interact with the map and he will go through it, because it is a portal game, and the level design of the map was crucial.Plus it was not only portal game, it was a coop portal game, where two players must solve the given puzzles in order to progress.It took us a lot of time to think about the puzzles, but at the end we managed to come up with an idea. This idea consists of using pressure points in order to open doors and progress. The two players must cooperate in order to solve these puzzles. We thought that the idea was great but not good enough, because just the puzzle itself won't enrich the gameplay of the game. So we thought to add some obstacles, these obstacles are the bots. These bots tend to kill the players. The game went from a normal portal game to a challenging game where the players need to fight against enemies and solve puzzles.

Obviously, the map has to satisfy the gameplay mechanics of the game. For this , we changed a little bit the first level. We removed the moving platforms, because they were useless at this point (because its a portal game as well). We noticed that it was too late to change the level 1, so we decided to use level 1 as a warm up level. Then we focused a lot on the second level.In fact the design of the second level is totally different from the first. In the second level, the player could use the portals in different situations. When we finished working on the map, we started working on the graphics. For the graphics it was not a problem, because we already knew what to implement.In fact, wed implemented the lights and post processing effects.

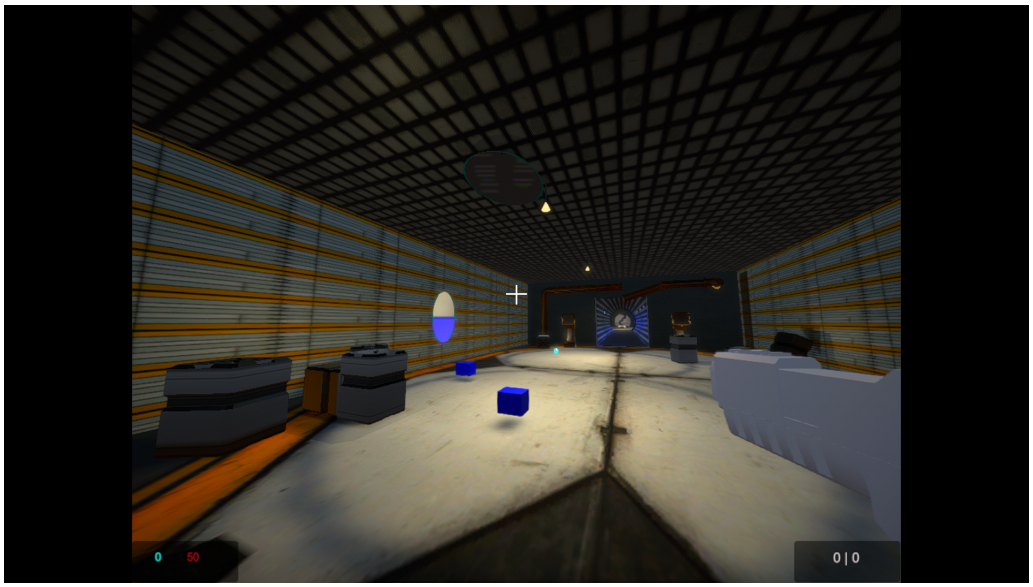
At the end, we were proud of the map that we created and we were really happy with it.

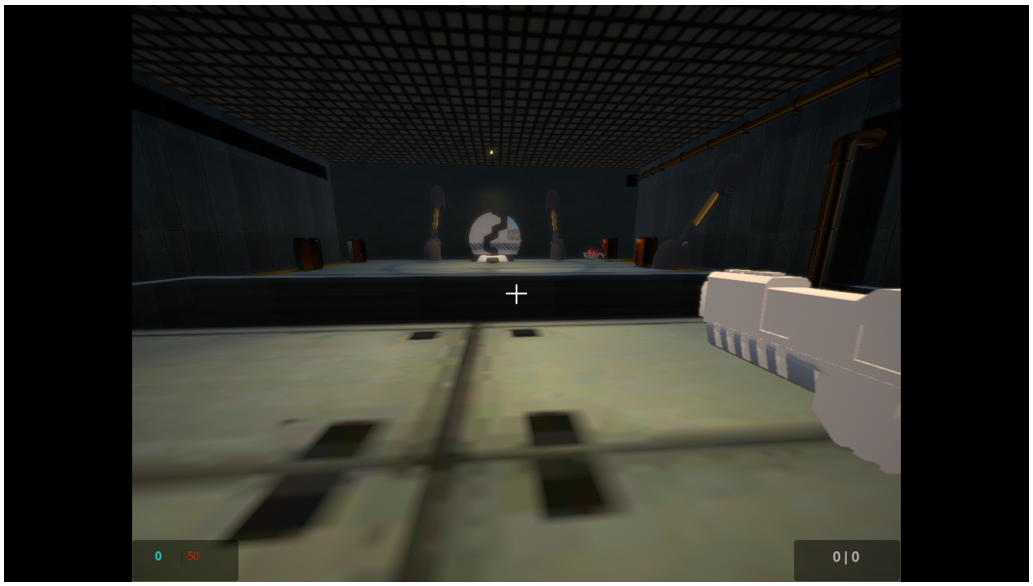
3.0.2 Game Mechanics and Game Play

At the very beginning before taking this huge journey of creating a game, we were nervous and scared as we were inexperienced and did not know which direction to look onto. So much information on the internet and such technical terms scared us initially, and honestly we doubted our abilities. In the first defence, we had character which was only a cylinder in a small map, jumping up and down and moving. NOTHING FANCY. From this



To this





Honestly, we are proud that after so much effort and work we have created this game. It's not perfect in any sense. It has some problems like the pick up objects might go out of the map when you're holding it. The problems faced during the creation of Portals. IT was all part of the learning curve.

But we can hold our head high, because we know one of Game Engines. Since childhood, everyone's fond of games(bar the exceptions), and it always raised a question. How the physics and the animations work? In few months, we have gathered that knowledge. And it feels overwhelming.

This project has caused us frustration a lot of times. We felt demotivated when things were not working our way. But it was all part of the learning curve. Thanks for this opportunity.

3.0.3 AI

With no previous knowledge in game development, the enemies were challenging to implement as they are moving parts in the game independent of the player. We needed bots to provide a challenge but not to make them too difficult. We mad them patrol an are rather than wander around an area as that is what we felt like a bot would be assigned to do. Making him chase the player was not a big issue but making them shoot was a different story. Implementing the shooting mechanic for the player was one thing, giving that ability to a uncontrollable entity is different. Thankfully we managed to make a pretty good enemy AI system with two bots with different abilities. One rapid shooting small projectiles that hurt the player's health and shield

and the other one shooting slowly but removing the shield completely.

3.0.4 User Interface

Starting out with the HUD, we needed something minimalist so as to fit our theme of sci-fi but also should be able to display all important information. This is where the idea for a number based health and shield system came in. While testing the game, we were constantly wondering if our gun was hitting the bots or not. When we decided to implement some checker, the idea struck that the player should have this ability as well. This is when we decided to add health bars for the enemy bots so as to make sure that the player knows if and when they hit the bot.

3.0.5 Network

My my my, did we not know what we were getting ourselves into. We had no prior experience developing a multiplayer let alone a singleplayer. It's our first time messing with any game Engine but as people who appreciates games we can say it was well worth it. We feel like understand the gaming industry and ~~hardships~~ joy of being a game developer and programmer. But in all seriousness, it was actually very satisfying being able to create a world we tailored to our taste.

We still felt like the multiplayer lagged a little behind the actual single player for example with a few bugs and missing particle effects. And at times it was extremely frustrating not knowing what little method could solve a problem and we would end up researching the solution to our problem only to find an outdated one form 2014 and would have implement or fix each little thing through tedious trial and error. One thing this project taught us is Patience. Having to go through other people's code to understand the problem was a first.

We recognize that the organizational structure was not the best, having every aspect compartmentalized (Covid certainly didn't help) leads to inefficient workflow. Maintaining a holistic, big picture mindset is very important especially for big projects. One must design all the facets of a game in consideration with one another.

Ultimately the network works and for people who have never made a game in their life and only recently started programming seriously, it's not

too bad. If the purpose was to learn and that purpose was definitely met.

4 Conclusion

And that's the end of your project, fruits of our continuous labor during the entire semester. We our proud of the result all things considered. We set out to create a 3D Platformer aswell as FPS elements. We consider that the book of specifications has been more or less with a few exception, perhaps we were too ambitious. We were glad to spend the time and effort to meet most of the objectives and realize a complete game.

We have acquired a good amount of experience in C but also in other fields such as WebDesign, Server side development, using software like Blender, the list goes. The experience was beneficial to all members of the group, teaching us good lessons about managing a group, the assigning of tasks and general organizational structure.

Our game Aperture turned out to be quite the adventure. We learned so much more than we have ever hoped for, it was well worth the time and effort. You don't have to see the whole staircase, just take the first step.