Report for Defence 2
Aperture

SCOTT TALLEC"α"

ABHISHEK BOSE"β"

RAJAT JOHN"Ω"

SOFIANE BEKHAT "γ"

Contents

1 Introduction

Now that we have at least some experience in making a game. We have decided to implement some more of the game-play aspects which makes the game come closer to he vision we had in mind for a game of this type.

Here are the details of the work accomplished so far, with the progress done, and the work to be done until the next defense.

2 Advancement in the project

2.1 Level Design and Graphics

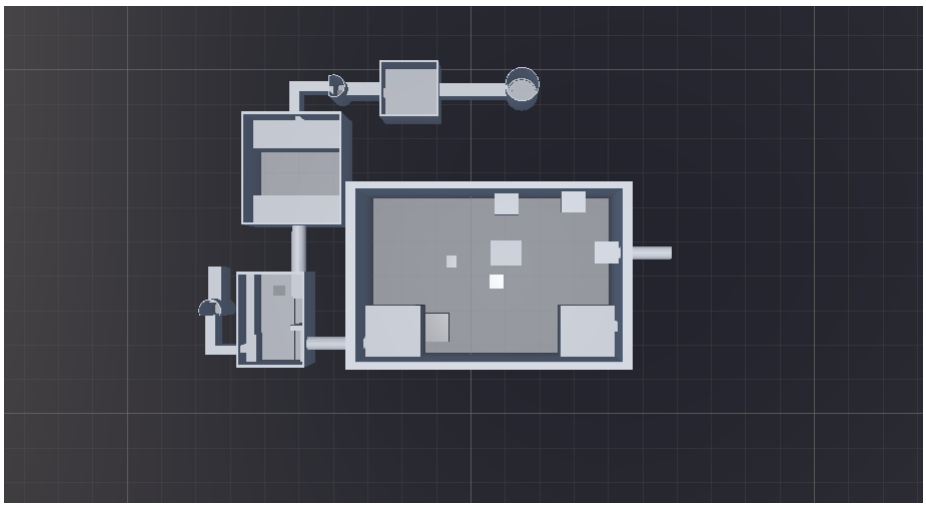
In this part, we will illustrate the main idea of the level design of the game and how it is structured. We will talk about the graphics of the game and explain the code behind the user interface.

Main concept

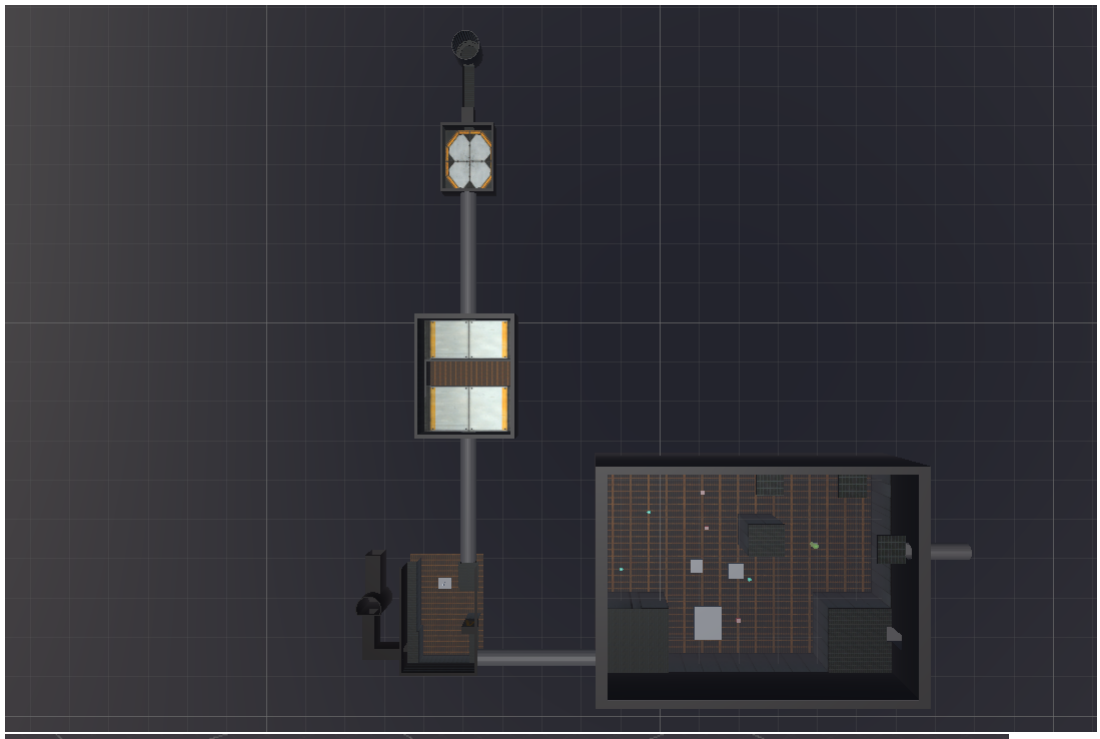
The level design and the graphics of a game are very important, In fact it is the first thing that one sees in a game and especially in our case , a portal type game where the design of a level is a crucial element for the game in order to make solvable puzzles.

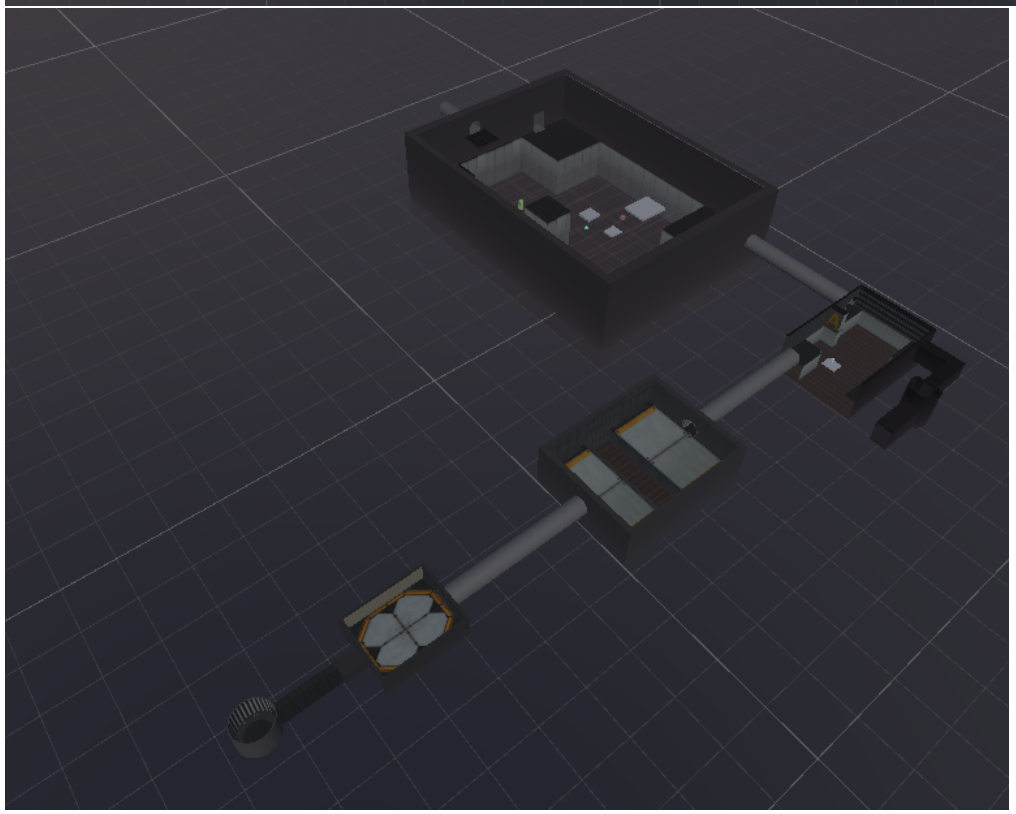
The main idea is to have different levels with a lot of variety and different level of difficulty, that will increase throughout the game.Therefore, the first level is a tutorial level, where the player can learn the controls of the game and also its mechanics. The next levels will be designed according to the puzzles that will be implemented for the game.

The Tutorial level, Textures and shading

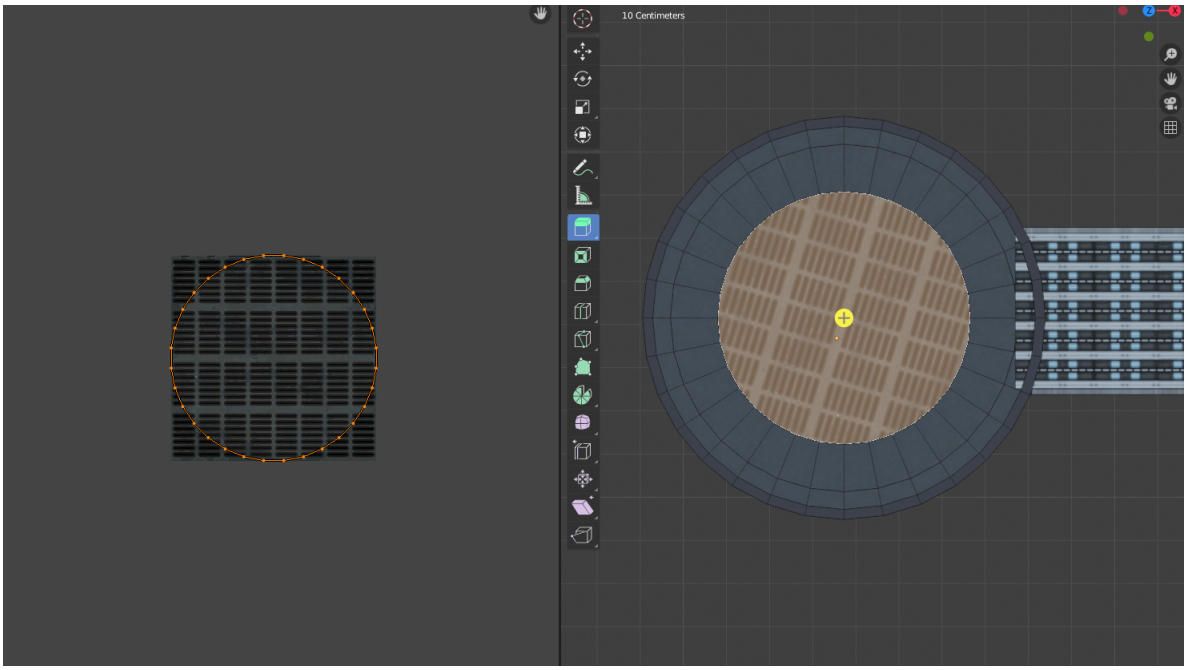


The Picture above is the tutorial level , before adding all necessary textures an shading. After a long study of how to bake models with good textures, This is the result :

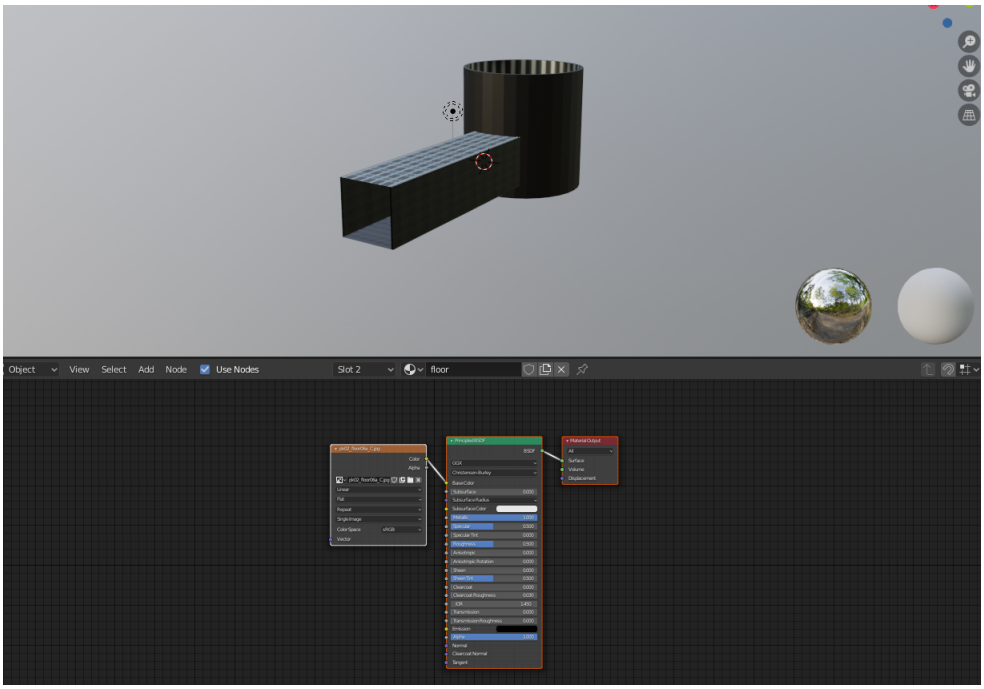




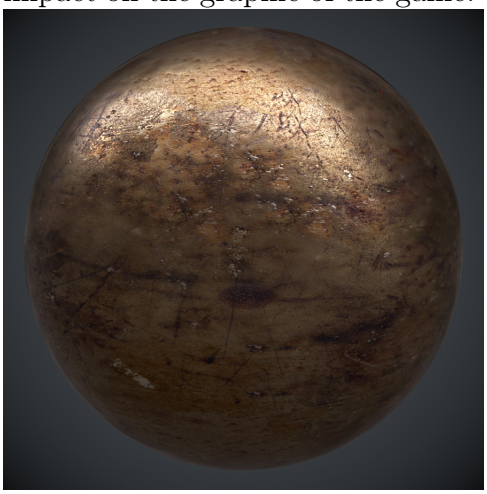
As one can see , all the textures have been applied to all meshes that we designed before.Compared to the previous map that we had, small changes have been made : now there are instead just corridors that link the rooms, no elevators (for now). The process of texturing is made possible thanks to a powerful tool from Blender, which is called UV Mapping.



The picture above rapresents 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.



The picture above shows the shading editor. Shading stands for a process than allows one to modify the look of a material. Through shading, one could be able to obtain the so called materials, which stands for Physically based rendering.The quality of PBR materials are really high, and it makes a huge impact on the graphic of the game. Here is an example of one PBR material:

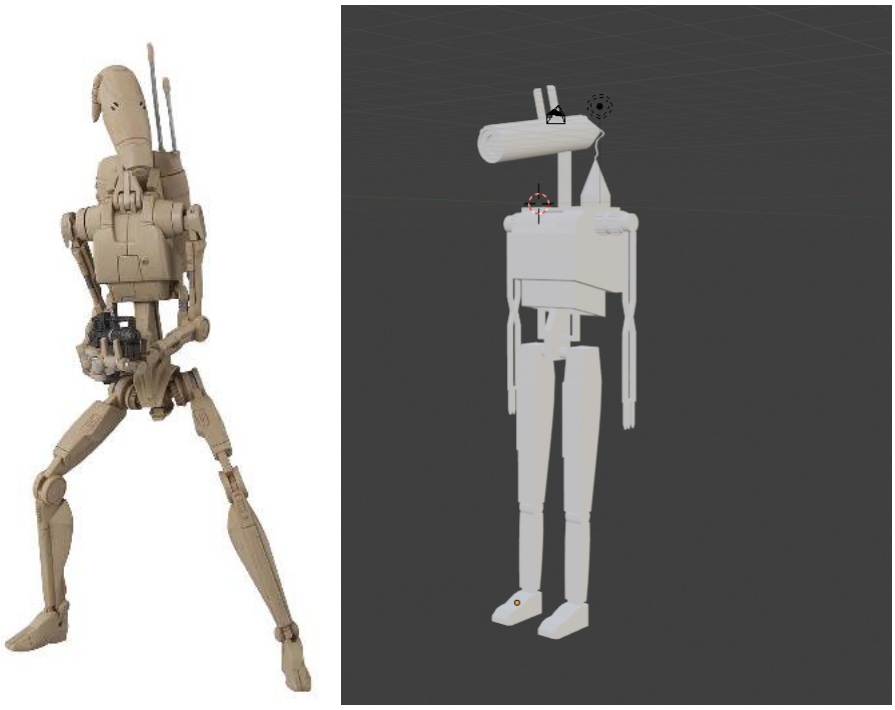


the shading process of the game is not finished yet. In fact, this will be one of our main objectives for the future. For now, we can say that our first level is complete.

2.2 The Main Character

2.2.1 Model

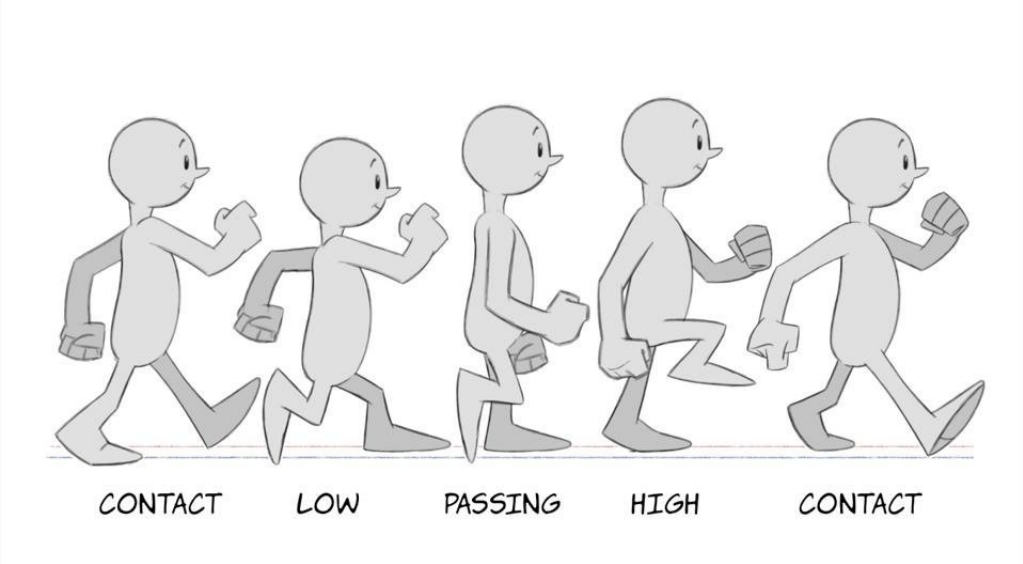
We have finally implemented our main character for the game. It was a long process of study and trials. In fact, it was one of the hardest things that we did so far. As said previously, It is a portal game set in a sci-fi environment.Therefore, our character has to be either a droid, or a robot. At the end we chose robot as our main character. The robot that we made is inspired from battle droids from star wars. After a long hard work, this is what we came up with :

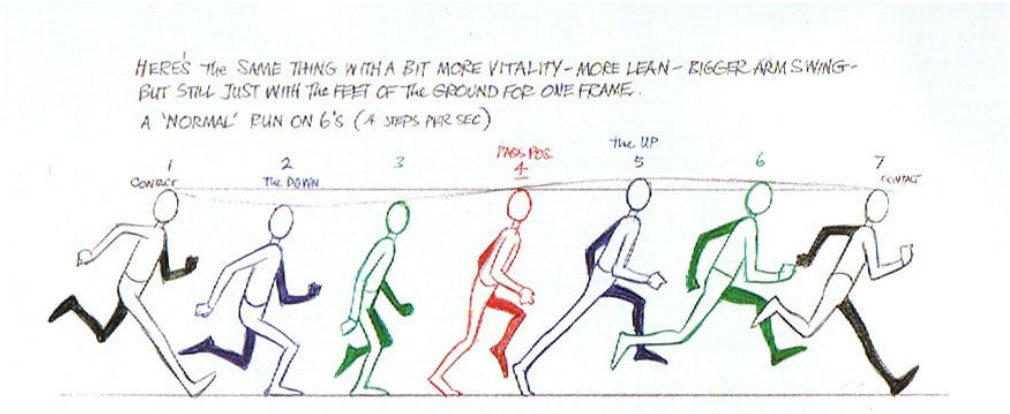


The process of modelling this character was a bit tricky. In order to solve problems that we might had faced during the modelling, we chose to model each part of the body separately (Head, Upper body, hands lower body ..). At the end , we put everything together in one single objects, obtaining the character as a result. Remark: the model was made on Blender.

2.2.2 Model animations :

So far, we have implemented only two animations, the walking animation, and running animation.Obviously more animations will be implemented. The process of animating an object is not hard , and not easy at the same time. It requires precision , creativity, and patience. We used the so called paper dope sheets, which are basically papers in which are drawn the sequences of an animation. Through the dope sheets, we were able to replicate the animations on blender more accurately. Here are the paper dope sheets that we used for the animations:





In addition, The animations were made possible thanks to tools offered by Blender.

2.3 Gameplay

2.3.1 Movement

This time around, we implemented the movement in a way that synchronises with the animation of the character. It was not hard to understand the blend tree and the whole animator. But it might get tricky when we start to implement a thing which is hard BUT A SURPRISE.....

There were some tweaks done to the movement of our character, like the leftshift is now a toggle. You click once, and then if you press the up arrow button, or any arrow button for instance, the player will move around with a greater speed, if we click the leftshift again, it toggles off. It was a optimization which we thought could be important for controlling the player.

For the Level One, we added one thing to our movement script and Camera Control script as the levels after tutorial level are co-op. We are checking if the PhotonView is ours or not. If yes, you execute the basic function of the player.
Implementing the CameraControl for Multiplayer was a problem, because at first we were facing an issue of camera being inverted, meaning, the camera of our player is actually the view of the 2nd player. So this is how we fixed it. Just one line of code, in our PhotonPlayer script

```
myAvatar.transform.Find("PlayerCamera").gameObject.SetActive(true);
```

myAvatar is the GameObject Player, and we try to look for child camera, and switch it on if the PhotonView is ours.

2.3.2 Animator

Now let's talk about the Animator. Understanding blendtree was a bit hard at first, but as progression, one gets a hang of it.

Code:

```
public class WalkingAnimation : MonoBehaviour
{
    public Animator animator;
    private PhotonView PV;
    private float InputX;
    private float Run;
    private bool Sprint;

    // Start is called before the first frame update
    void Start()
```

```
    {
        animator = gameObject.GetComponent<Animator>();
        PV = GetComponent<PhotonView>();
    }

    // Update is called once per frame
    void Update()
    {
        if (PV.IsMine)
        {
            InputX = Input.GetAxisRaw("Vertical");
            animator.SetFloat("InputX", InputX);

            if (Input.GetKeyDown(KeyCode.LeftShift))
            {
                Sprint = !Sprint;
            }

            if (Sprint)
            {
                if (Input.GetButton("Vertical"))
                {
                    animator.SetBool("Run", true);
                    InputX = Input.GetAxisRaw("Vertical");
                    animator.SetFloat("InputX", InputX);
                }
                else
                {
                    animator.SetBool("Run", false);
                }
            }
        }

    }
}
```

Here as well you can see we are doing the same as before, asking the program, if it is my photonView or not. If yes, we animate along with move.

2.3.3 Jumping

Optimization and Jump for Lvl1:

For lvl 1, we thought of adding a power up which needs to be earned in the game. The power up involves a mid air jump just once.

```
        if (isGRounded && PowerUp == 0 &&  Input.GetButton("Jump"))
        {
            _rb.AddForce(Vector3.up + thrusterForce * Time.fixedDeltaTime, ForceMode.Impulse);
            PowerUp++;
        }
        if (Input.GetButtonUp("Jump"))
        {
            isGRounded = false;
        }
        if (!isGRounded && PowerUp<2 && Input.GetButtonDown("Jump"))
        {
            _rb.AddForce(Vector3.up + (thrusterForce-gravitypull) * Time.fixedDeltaTime, ForceMode.Impuls
            PowerUp++;
```

```
        }
    }
private void OnCollisionEnter(Collision collision)   //To check if the player is colliding with collider
{
    if (collision.gameObject.CompareTag("Ground"))
    {
            isGRounded = true;
            PowerUp = 0;
    }
}
```

As usual, We are using a ColliderTag to see if the player is on the ground. Also, We are using a variable called PowerUp, which limits the player jumping more than once in the air.

2.3.4 Game mechanics

The mechanics of the game have been improved upon this time around to give the game something stronger to stand on than your average aim and shoot.

Since the shooting has already been implemented, we decided to add a reload to the gun to give it a little more depth.

This required a new input called "Reload" mapped to the 'r' button. After this, simple code was implemented to utilise this new input.

Code

```
public class GunReload : MonoBehaviour
    {
        public GameObject CrossObject;
        public GameObject MechanicsObject;
        public int ClipCount;
        public static int ReserveCount;
        public int ReloadAvailable;
        void Update ()
        {
            ClipCount = Ammo.currentammo;
            ReserveCount = Ammo.internalammo;

            if (ReserveCount == 0)
            {
                ReloadAvailable = 0;
            }
            else
            {
                ReloadAvailable = 10 - ClipCount;
            }

            if(Input.GetButtonDown("Reload"))
            {
                if (ReloadAvailable >= 1)
                {
                    if (ReserveCount <= ReloadAvailable)
                    {
                        Ammo.internalammo -= ReserveCount;
                        Ammo.currentammo += ReserveCount;
                    }
                    else
                    {
                        Ammo.internalammo -= ReloadAvailable;
                        Ammo.currentammo += ReloadAvailable;
                    }
                }
            }
        }
```

As mentioned in the HUD section, a shield system has been implemented by updating the "PlayerHealth" script as shown below.

Code

```
public class PlayerHealth : MonoBehaviour
    {
        public static int health = 50;

        public static int shield = 0;

        void PlayerDmg(int dmg)
        {
            if (shield == 0 && dmg < health)
                health -= dmg;
            else
            {
                if (shield == 0 && dmg > health)
                    health = 0;
                else
                {
                    if (dmg < shield)
                        shield -= dmg;
                    else
                        shield = 0;
                }
            }
        }

        void ShieldDes()
        {
            if (shield > 0)
                shield = 0;
        }
    }
```

Here the "ShieldDes" method will be called upon by one of our brand new bot which aims to destroy your shields. But more on that later.
Now that we have shields, we need a away to increase them once you take some damage. This is where our shield pickups come in. Since we have an "ammodrop" script, all we had to do is modify it a little bit and recolour the object.

Code

```
public class ShieldPickup : MonoBehaviour
{
    private void OnTriggerEnter(Collider collider)
    {
        int res = PlayerHealth.shield + 20;
        if (res > 50)
            PlayerHealth.shield = 50;
        else
            PlayerHealth.shield = res;
        if(PlayerHealth.shield < 50)
            this.gameObject.SetActive(false);
    }
}
```

Immense progress was made with the AI as now we have a patrol system which allows the enemy bot to patrol an area and if the player comes too close, the bot chases and starts shooting the player. If the player gets too close for the bot's comfort, the bot starts retreating. If the player runs away and is too far to chase, the bot goes back to patrolling. Below is the code for when the player is close and the patrol code.

Code

```
public class EnemyAI : MonoBehaviour
{
    private Transform playerTransform;

    public float speed = 8f;

    public static float StoppingDis = 7f;

    public float RetreatDis = 5f;

    private Vector3 direc;

    private Quaternion rot;

    void Start()
    {
        playerTransform = GameObject.FindGameObjectWithTag("Player").transform;
    }

    void Update()
    {
        direc = playerTransform.position - transform.position;
        rot = Quaternion.LookRotation(direc);
        transform.rotation = rot;
        if (Vector3.Distance(transform.position, playerTransform.position)> StoppingDis)
        {
            transform.position = Vector3.MoveTowards(transform.position, playerTransform.position, speed * Time.deltaTime);
        }
        else if(Vector3.Distance(transform.position, playerTransform.position) < StoppingDis && Vector3.Distance(transform.position,
        {
            transform.position = this.transform.position;
        }
        else if(Vector3.Distance(transform.position, playerTransform.position) < RetreatDis)
        {
            transform.position = Vector3.MoveTowards(transform.position, playerTransform.position, -speed * Time.deltaTime);
        }
    }

}


 public class PatrolAI : MonoBehaviour
    {
        public float Speed;
        public Transform[] MoveSpots;
        private int RandomSpot;
        private float waitTime;
        public float startWaitTime;

        // Start is called before the first frame update
        void Start()
        {
            waitTime = startWaitTime;
            RandomSpot = Random.Range(0,MoveSpots.Length);
        }

        // Update is called once per frame
        void Update()
        {
            transform.Rotate(0,0,0);
            Rigidbody m_Rigidbody = GetComponent<Rigidbody>();
            transform.position = Vector3.MoveTowards(transform.position, MoveSpots[RandomSpot].position, Speed * Time.deltaTime);
            if(Vector3.Distance(transform.position,MoveSpots[RandomSpot].position) < 0.2f)
            {
                if (waitTime <= 0)
                {
                    m_Rigidbody.constraints = RigidbodyConstraints.None;
```

```
                RandomSpot = Random.Range(0,MoveSpots.Length);
                waitTime = startWaitTime;
            }
            else
            {
                waitTime -= Time.deltaTime;
                m_Rigidbody.constraints = RigidbodyConstraints.FreezePosition;
            }
        }
    }
}
```

Here we can see that the bot finds out the position of the game object tagged as player and moves towards it once the player is within a certain distance. If the player is too far, the bot patrols between the array of spots in the scene called "MoveSpots" in the "PatrolAI" script. To toggle between the two scripts and to check the player's distance, the following script was implemented. Code

```
public class PlayerDisChecker : MonoBehaviour
{
    private Transform playerTransform;

    public float AwareDis = 8f;
    // Start is called before the first frame update
    void Start()
    {
        playerTransform = GameObject.FindGameObjectWithTag("Player").transform;
    }

    // Update is called once per frame
    void Update()
    {
        if (Vector3.Distance(transform.position, playerTransform.position) > AwareDis)
        {
            this.gameObject.GetComponent<PatrolAI>().enabled = true;
            this.gameObject.GetComponent<EnemyAI>().enabled = false;
        }
        else
        {
            this.gameObject.GetComponent<PatrolAI>().enabled = false;
            this.gameObject.GetComponent<EnemyAI>().enabled = true;
        }
    }
}
```

Now, we have to implement the shooting mechanic for the bot as to provide a challenge to the player. We have implemented two types of guns for the bot which essentially means we have two different bots. One of them deals 5 damage to shields then health every 1 second and the other destroys the player's shield completely but cannot reduce the player's health, this happens once every 5 seconds. Individually they are pretty easy to deal with but together they can cause major issues for the player.

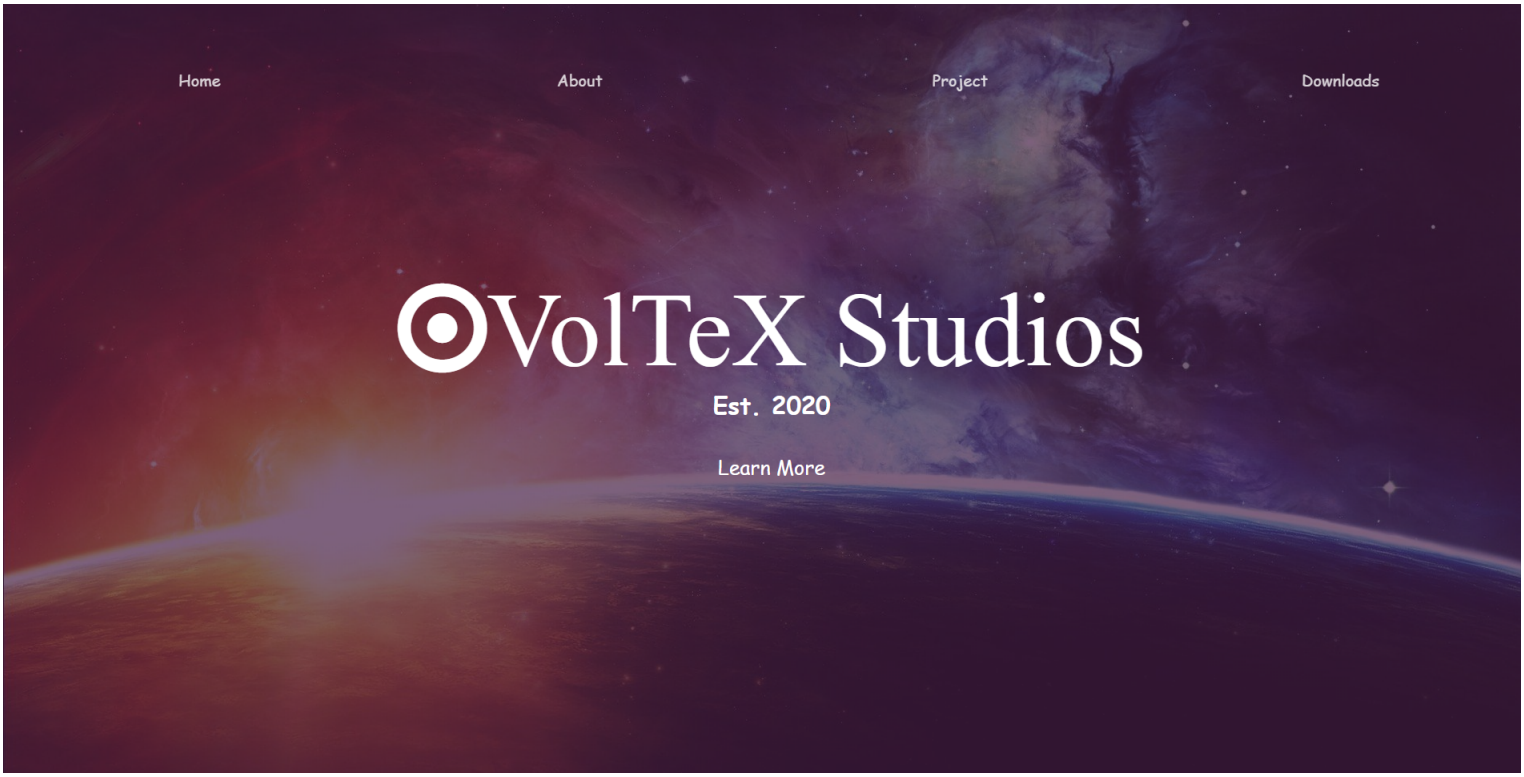
Code

```
public class EnemyGun : MonoBehaviour
{
    private float TimeBtwShot;

    public float StartTimeBtwShot;

    private float targetdis;

    private float range;

    public int dmg = 5;

    void Start()
    {
        TimeBtwShot = StartTimeBtwShot;
        range = EnemyAI.StoppingDis;
    }

    void Update()
    {
        if (TimeBtwShot <= 0)
        {
            RaycastHit shot;
            if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out shot))
            {
                targetdis = shot.distance;
            }

            if (targetdis < range)
            {
                shot.transform.SendMessage("PlayerDmg", dmg);
            }

            TimeBtwShot = StartTimeBtwShot;
        }
        else
        {
            TimeBtwShot -= Time.deltaTime;
        }
    }
}
```

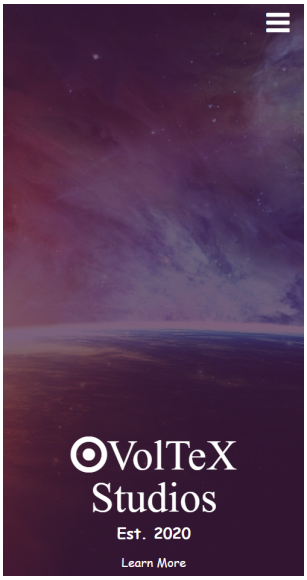
```
public class ElecGun : MonoBehaviour
    {
        private float TimeBtwShot;

        public float StartTimeBtwShot;

        private float targetdis;

        private float range;

        // Start is called before the first frame update
        void Start()
        {
            TimeBtwShot = StartTimeBtwShot;
            range = EnemyAI.StoppingDis;
        }

        // Update is called once per frame
        void Update()
        {
            if (TimeBtwShot <= 0)
            {
                RaycastHit shot;
                if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out shot))
                {
                    targetdis = shot.distance;
                }

                if (targetdis < range)
                {
                    shot.transform.SendMessage("ShieldDes");
                }

                TimeBtwShot = StartTimeBtwShot;
            }
            else
            {
                TimeBtwShot -= Time.deltaTime;
            }
        }
    }
```

The scripts will be attached to specific bots that will be differentiated by their colour.

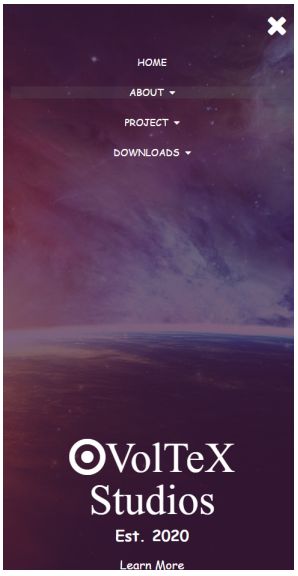
2.4 Website


Home
About
Project
Downloads
VolTeX Studios
Est. 2020
Learn More

Earlier, we just mentioned about the HomePage of our website. But this time around, we worked on it for it to be responsive website which means, it is optimized for all screenwidth.



As one can see, it's responding well to a different screen width. That is what we wanted. If you click on the bars at top right. The following happens:



One can see a drop down menu option. And yes, it involves a bit of Javascript, for the options to be clickable.

2.4.1 Link

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

2.4.2 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(which is not updated yet)
- Project: Aperture, one can gather information about our Game(which is not updated yet)
- Downloads: One can download the BookofSpecs, Report1, Report2, and Report3

2.5 Network

In network a lot of the work was put into implementing making it fucntion on it's own aswell as fully integrating it within the singleplayer version of the game. As such all the features added where to assist in accomplishing that goal.

2.5.1 Menu Implementation

The menu had to function hand in hand with the multiplayer since we previously had a non-working menu. We coded each button to map to a script with loaded a new scene in.

```
    public class PhotonRoom : MonoBehaviourPunCallbacks, IInRoomCallbacks
{
    // Room Info
    public static PhotonRoom room;
    private PhotonView PV;
    public int currentScene;
```

```
private void Awake()
{
    if (PhotonRoom.room == null)
        PhotonRoom.room = this;
    else
    {
        if (PhotonRoom.room != this)
        {
            Destroy(PhotonRoom.room.gameObject);
            PhotonRoom.room = this;
        }
    }

    DontDestroyOnLoad(this.gameObject);
}

public override void OnEnable()
{
    base.OnEnable();
    PhotonNetwork.AddCallbackTarget(this);
    SceneManager.sceneLoaded += OnSceneFinishedLoading;
}

public override void OnDisable()
{
    base.OnDisable();
    PhotonNetwork.RemoveCallbackTarget(this);
    SceneManager.sceneLoaded -= OnSceneFinishedLoading;
}

void Start()
{
    PV = GetComponent<PhotonView>();
}


public override void OnJoinedRoom()
{
    base.OnJoinedRoom();
    Debug.Log("We are now in a room");
    StartGame();
}


void StartGame()
    {

        if (!PhotonNetwork.IsMasterClient)
            return;
        PhotonNetwork.LoadLevel(MultiplayerSettings.multiplayerSetting.multiplayerScene);
    }

    void OnSceneFinishedLoading(Scene scene, LoadSceneMode mode)
    {
        currentScene = scene.buildIndex;
        if (currentScene == MultiplayerSettings.multiplayerSetting.multiplayerScene)
        {
          RPC_CreatePlayer();

        }
    }

    [PunRPC]
    private void RPC_LoadedGameScene()
    {
        {
            PV.RPC("RPC_CreatePlayer", RpcTarget.All);
        }
    }

    [PunRPC]
    private void RPC_CreatePlayer()
    {
```

```
            PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "PhotonNetworkPlayer"), transform.position,
                Quaternion.identity, 0);
        }


        // Update is called once per frame
}
```

Multiple things happen here, first whenever we join a Photon room, StartGame() is called which will load a scene and when the scene is finished loading, the PhotonNetworkPlayer will spawn the Avatar in. The Scene that loads is based on on the build index which is determined in the Inspector when we build. Using this we can load whatever scene we want in whichever order and bind a button to load the next scene. This is called whenever we join a room which is triggered when we click multiplayer or singleplayer.

Eventually we could add in the menu options that load a specific level (or scene), allowing to skip the initial level.

2.5.2 Disconnect Button

```
public void DisconnectPlayer()
{
    StartCoroutine(DisconnectAndLoad());
}

IEnumerator DisconnectAndLoad()
{
    PhotonNetwork.LeaveRoom();
    while (PhotonNetwork.InRoom)
        yield return null;
    SceneManager.LoadScene(MultiplayerSettings.multiplayerSetting.menuScene);
}
```

This is the script that loads whenever the disconnect button is pressed (toggled on and off by escape), it disconnects from the room and promptly loads a new scene that's the menu. So we no longer have to restart the game everytime we want to access the menu.

2.5.3 SinglePlayer

In this game we decided to have the SinglePlayer. In order to do this, whenever the single player button is activated instead of simply joining a room we also set the maximum number of players to 1 instead of the default 2. It's a naive solution to implementing a SinglePlayer but it works adequately. A possible optimization would be to deactivate all the PhotonView inspectors (all the multiplayer elements), as in our SinglePlayer the PhotonEngine is still actively updating every coordinate of the Player every frame.

```
public void OnSinglePlayerButtonClicked()
{
```

```
        MultiplayerSettings.multiplayerSetting.maxPlayers = 1;
        battleButton.SetActive(false);
        CancelButton.SetActive(true);
        PhotonNetwork.JoinRandomRoom();
    }
```

2.5.4 Mouse Movement

The last time Player Movement was implemented, but the camera remained fixed in the world. We therefore needed to add a camera to the player prefab so that we he is spawned in (instantiated) and corresponding camera appears. But a problem arose when adding a camera as the child of said prefab, the first and second players would have inverted cameras. A fix to this was only activating the camera that was spawned in only if it belonged to the client (the player).

```
    void Start()
{

        PV = GetComponent<PhotonView>();

        if ((PV.IsMine) && PhotonNetwork.IsMasterClient)
        {
            myAvatar = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "PlayerAvatar"),
                GameSetup.GS.spawnPoints[0].position, GameSetup.GS.spawnPoints[0].rotation, 0);
            myAvatar.transform.Find("PlayerCamera").gameObject.SetActive(true);
        }
        else if (PV.IsMine)
        {
            myAvatar = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "PlayerAvatar"),
                GameSetup.GS.spawnPoints[1].position, GameSetup.GS.spawnPoints[1].rotation, 0);
            myAvatar.transform.Find("PlayerCamera").gameObject.SetActive(true);
        }

    }
```

The spawn points are manually set in the inspector and decide where the player will be loaded in a specific level.

3 And After?

We have therefore made good progress with this defense, but we do not stop here! Even if what remains to be done here is attributed to some people in the group, we will develop them, for the most part, together, in order to allow each of us to be able to code different things. Here are the forecasts for the 2nd defense.

3.0.1 Level Design

For the level Design, new levels will be added, with high quality textures, different size and different puzzles to solve.

3.0.2 Gameplay

New power ups will be implemented and the AI will be improved, to make the game more challenging. Hopefully implementing navmesh to make the AI "smarter", our aim is to make the bots not a nuisance but a good set of extra obstacles to make the game more engaging.

3.0.3 User Interface

Regarding the main menu, new buttons and animations will be implemented, to make the main menu look cooler , and modern. And for the HUD , we will implement the health bar and a pause menu.

3.0.4 Network

Animations will load for when a non-client player moves. A chat system will also be added so that the players can communicate with one another in-game.