Aren Chen (qc525), Barkin Simsek (bs3528), Junior Garcia (jfg388)

Professor Mohamad Eid

ENGR-UH 1000

9 December 2017

Computer Programming for Engineers Final Project Report

1. Introduction

In a world bursting with boring entertainment applications, our project aims to obliterate feelings of boringness and sadness among users through a highly interactive game that immerse users in the world of exciting space combat. The aforementioned will be achieved through the use of an Oculus Rift Head Mounted Display, a highly immersive set of virtual reality goggles. It will provide users with the opportunity of escaping the bounds of reality by entering the game itself and using the natural body movements to control the game. This will engage users in an unprecedented manner, as they will feel the adrenaline running down their veins when they experience "first-hand" how the enemies are quickly approaching.

The inspiration behind this game was Cat vs. Dog, a two-player fighting game. We were entertained by its features as we found it amusing how the cat and the dog engaged in a ceaseless battle to demonstrate their superiority by throwing things at each other. However, we wanted our game not to be merely a copy of this game. We wanted to expand its scope of possibilities by making it into a highly interactive space combat game where the user is teletransported into the scary vastness of space. Hence, the inception of VRATTACK.

The following are the objectives VRATTACK will accomplish:

- Utilize the Oculus Rift Head Mounted Display as the medium through which users will experience the digital world we create
- Merge users with the game character through the use of natural body movements
- Proving a single player, first person shooter experience
- Immerse users in the world of space combat
- Arouse happiness among users due to the highly entertaining characteristics of the game
- Create a sense of self-accomplishment in the users who will try and beat their own previous performances

- Increase the spirit of camaraderie among the users and group members through the use of this highly creative gaming application

VRATTACK

Image 1: VRATTACK Logo

2. Project Development

Device(s)

At first, we were planning to utilize the Novint Falcon as our primary tool to engage the users. We wanted to use its haptic modality and its force feedback features to make shooting more difficult for the user. However, we had a lot of problems with the installation of the drivers, and those problems went beyond our realm of understanding and even that of some experts on the matter. That, combined with a lack of time, forced us to use the Oculus Rift Head Mounted Display as a backup device.

Despite this quandary, we acquired a vast range of knowledge on how windows drivers work and how to install/remove them, how operating systems handle I/O devices and even the basics of marshalling with the C# programming language to modify memory representations during the troubled periods. Such knowledge inspires us to not look at this unfortunate time as merely a mishap we must forget, but as a learning opportunity for us and as a chance for VRATTACK to pivot to technology we didn't primarily consider. The Oculus Rift Head Mounted Display proved to be a more suitable technology for our intended purposes behind the game we conceptualized in our minds. We take pride in the decision of switching into the and hope to use the Novint Falcon in future refurbishments of VRATTACK.

In addition to the Oculus Rift Head Mounted Display headset, we used the Xbox remote controller to give additional functionalities such as shooting and walking around the game environment. The Xbox remote controller proved to be a good complement to the Oculus Rift Head Mounted Display, as the user can use the later just to better enjoy the features of the environment by moving his head freely while the Xbox remote controller is used for the necessary action moves of the game, like shooting and moving.

Image 2 and 3: Novint Falcon Game Controller and Oculus Rift Head Mounted Display

Image 4: Xbox Remote Controller

Software

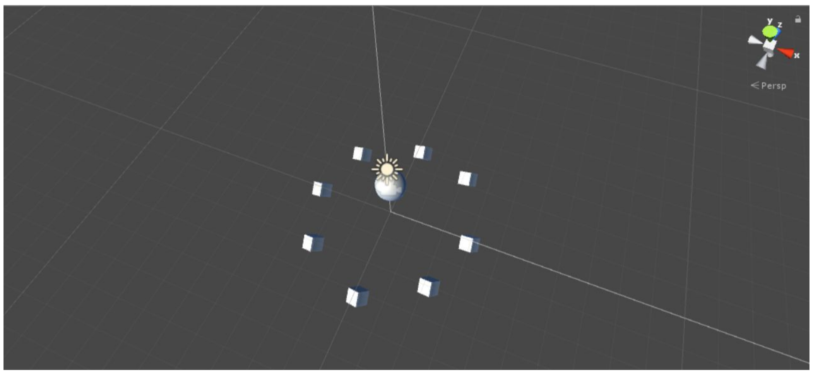
As the main avenue for development, we chose to use the latest version (2017.2) of the Unity game engine and the latest version (Community 2017) of Visual Studio IDE to code the scripts for the game. We decided to choose Unity because it is reliable and it has a huge community behind it that enabled us to solve our problems relatively faster. We could ask for help in the online forums and receive it in expedited amounts of time. In addition, the Unity tutorial we received in one of the laboratory sessions made us familiar with the software before embarking on this feat. Similarly, we decided to use Visual Studio IDE as our text editor and compiler because it is reliable and we used it for the other assignments for the class.

After obtaining the Oculus Rift Head Mounted Display from the NYUAD library, we downloaded the Oculus software from (https://www.oculus.com/setup/) and then installed the necessary drivers to our computers. We then completed the setup of the Oculus Rift Head Mounted Display and Xbox remote controller by following the adjacent instructions.

Game

Once the devices were ready, we delved into the Unity website in search of tutorials on shooting games that would give us an insight on how to develop our own version of the shooting game.

At first, we utilized cubes as a way of representing the enemies. We did this during the primitive stages of the game, and it helped us come up with more innovative ways of creating enemies.



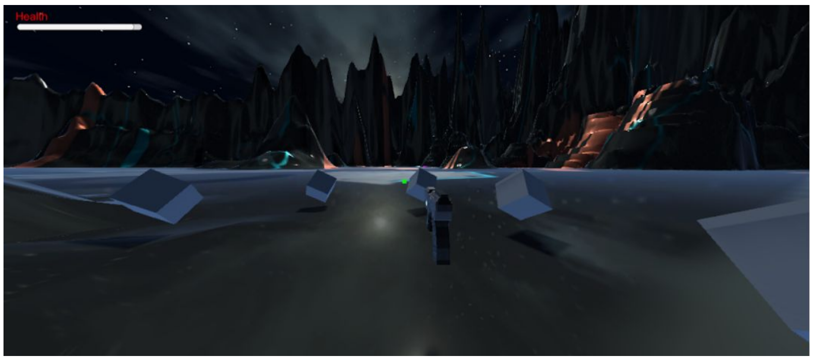


Image 5 and 6: Development of the game

After we developed the cubes as a our mock enemies, we decided to create actual enemies that have more autonomy and determination to kill the user.

Image 7 : Sample alien

After creating the enemies of the game, we then worked on the more aesthetic details of the game, which are mentioned in the next subsection.

Different Features of the Game

Menu

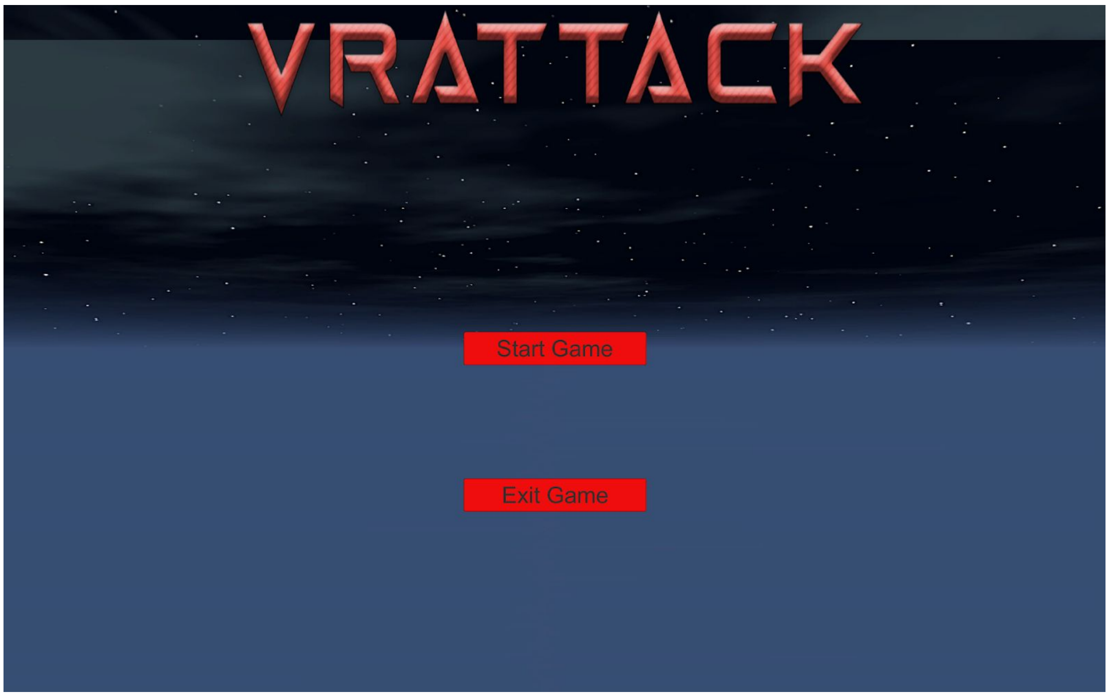


Image 8 : Game menu

VR integration

Head movements
Xbox controller

Shooting

First person experience
Move with Oculus Rift

Auto Spawning Aliens

Moving aliens
Two types with different speed, damage, and health

Main Components of the Game

Gun

https://www.assetstore.unity3d.com/en/#!/content/90249

Image 8 : Gun used in the game

Aliens (Two Types)

https://www.assetstore.unity3d.com/en/#!/content/36365

Image 9 and 10 : Alien types used in the game

Arena

Arena is created in the Unity

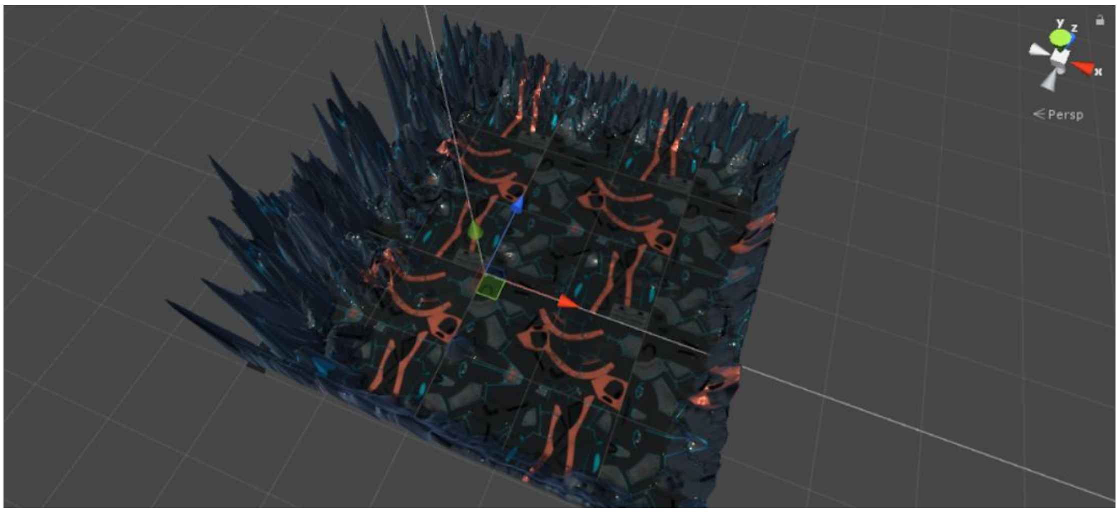


Image 11 : Arena used in the game

Logo

Created in Adobe Photoshop

VRATTACK

Image 12 : Logo of the game

Audio

https://freesound.org/ and edited with Audacity Software

"You are dead" Screen

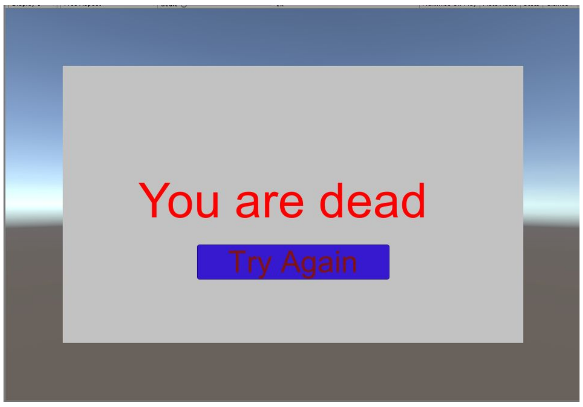


Image 13 : You are dead screen

Scripts

Enemies Attacking Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//attacks of enemies
public class Attack : MonoBehaviour {
    public float attackSpeed ;//how many seconds the enemies are going to
attack once
    public float damage ;//amount of damage enemies do to Ali
    public GameObject user;//Ali(the player)
    PlayerHealth playerhealth;//health

    float CountTime;//timer

    void Start () {
        user = GameObject.FindGameObjectWithTag("Player");//find Ali
        playerhealth = user.GetComponent<PlayerHealth>();
    }

    //attack the player every "attackspeed" time and do damage with amoung
of "damage"
    void Update () {
        CountTime += Time.deltaTime;
        if((CountTime > attackSpeed )&&
(user.transform.position-transform.position).magnitude<5)
        {
            CountTime = 0;
            playerhealth.takeDamage(damage);
        }
    }
}
```

Button Manage Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;

//used to load scenes in UI
public class ButtonManage : MonoBehaviour {
 public void NewGameBtn(string name)
    {
        SceneManager.LoadScene(name);
    }
```

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

    void Update()
    {
        if (Input.GetButton("A"))
        {
            SceneManager.LoadScene(name);

        }
        else if(Input.GetButton("X"))
            Application.Quit();
    }
}
```

Destroy When Out Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//the invisible cube destroy things that goes out of map
public class DestroyWhenOut : MonoBehaviour {


      void Start () {

      }


      void Update () {

      }

    void OnTriggerExit(Collider other)
    {
        Destroy(other.gameObject);
    }
}
```

Enemy Coming Towards You Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//make enemies move and face forward to the player
public class EnemyToward : MonoBehaviour {
    public GameObject target;//the target enemies are going to
    public float speed;//the speed of enemies

      void Start () {

      }


      void Update () {
        Vector3 path
=target.transform.position-transform.position;//calculate the path to target
        path = path / path.magnitude;//calculate only the direction of the
path
        path.Set(path.x, 0.0f, path.z);//set the path on the ground
        transform.forward = path;//make enemies look forward to the target

        transform.position = transform.position + speed * path;//move
enemies with speed "speed"

      }
}
```

Killed Enemy Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;

//to test if an enemy is been killed
public class Killed : MonoBehaviour {
    public int num;//number of bullet an enemy can handle

    public AudioSource mu;//the sound effect of an enemy's death

    private Rigidbody rb;


      void Start () {
        rb = GetComponent<Rigidbody>();

      }


      void Update () {

      }

    void OnTriggerEnter(Collider other)
    {
        //destroy the enemy if it has no life anymore, minus onoe life if
it still has after collide with a bullet
```

```
        if (other.gameObject.CompareTag("bullet"))
        {

            if (num <= 1)
            {
                Destroy(gameObject);
                scoreManager.score += 1;//add one score
                mu.Play();

            }
            else
            {
                num -= 1;

            }
        }
    }

}
```

Player Health Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using UnityEngine;

// the health of player
public class PlayerHealth : MonoBehaviour {
    public float fullHealth = 500;//maximum health
    public float currentHealth;//current health
    public Slider healthslider;//the health bar in UI
    public AudioSource music;//the sound when being attacked
    public string name;//the name of the next scene



    private bool isDead=false;//test if Ali (the character) is dead


    void Awake()
    {
        currentHealth = fullHealth;

    }

      void Start () {

      }
      void Update () {
        if (isDead)
        {
            SceneManager.LoadScene(name);//load the next scene if Ali is 
dead
            gameObject.SetActive(false);



        }
      }

    public void takeDamage(float amount)
    {
        music.Play();//play the sound of hurting when Ali is being attacked
        currentHealth -= amount;//reduce health

        healthslider.value = currentHealth;//show the health on UI

        //actions when Ali is dead
        if (currentHealth <= 0)
        {
            music.Play();
            isDead = true;
        }
    }
}
```

Score Panel Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using UnityEngine;

//this is the score panel in UI
public class scoreManager : MonoBehaviour {

    public static int score;//the score
    Text txt;

      void Start () {

        score = 0;
        txt = GetComponent<Text>();

}


      void Update () {
        //show the score on the screen
        txt.text = "Score: " + score;
      }

    public void addScore()
    {
        //add one score
        score += 1;
    }
}
```

Shooting Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Shoot : MonoBehaviour {
    //Beam is the bullet, music is the sound of shooting
    public GameObject Beam;
    public AudioSource music;
      //detect if user tries to shoot five times a second
      void Start () {
        InvokeRepeating("SH", 0, 0.2f);
    }


      void Update () {

      }

    //play the shooting sound and generate a new bullet in front of the gun
    void SH()
    {
        if (Input.GetButton("B")||Input.GetMouseButtonDown(0))
        {
            music.Play();
            Instantiate(Beam, transform.position, Quaternion.identity);
        }
    }
}
```

Shoot Away Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//behavior of the bullets after been shot
public class ShootAway : MonoBehaviour {
        public GameObject gun;//the gun
        public int num=0;
        public float speed;//speed of the bullets
        private Vector3 direction;//direction of the bullets

        void Start()
        {
           num += 1;
           direction = gun.transform.forward;//set direction as the
direction the gun points to
        }
    void Update()
        {
        if (num >1)
        {
            transform.position = transform.position + speed * direction;//the
bullets fly with speed "speed" and direction "direction"
            transform.Rotate(new Vector3( 0.0f, Time.time, Time.time) *
50.0f);//self rotation of bullets
        }

        }
}
```

Spawning Enemies Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//generating more and more enemies with faster rate by the map
public class SpawnEnemies : MonoBehaviour {

    public GameObject A1;//enemy type one
    public GameObject A2;//enemy type two
    public float spawnSpeed;//time of generating one enemies
    public AudioSource music;//the background music of the game
    private float timer;

    void Awake()
    {
        //play the background music repeatedly
        music.playOnAwake=true;
        music.loop = true;
    }
      void Start () {
      }

      void Update () {
        //call "spawn" function every "spawnSpeed" time, which is becoming
smaller until to 1
```

```
        timer += Time.deltaTime;
        if (timer > spawnSpeed)
        {
            timer = 0;
            Invoke("spawn",0);
            if (spawnSpeed > 1)
            {
                spawnSpeed -= 0.01f;
            }
        }
      }
     void spawn()
    {
        Vector3 spawnPoint = new Vector3(Random.value * 200 - 50, -1.0f,
Random.value * 200 - 100);//randomly select the place an enemy spawns
        float type = Random.value;//randomly select which enemy to spawn
        if (type>0.5)
        {
            Instantiate(A1, spawnPoint, Quaternion.identity);//put a new
enemy type one at "spawnPoint"
        }
        else
        {
            Instantiate(A2, spawnPoint, Quaternion.identity);//put a new
enemy type two at "spawnPoint"
        }
    }
}
```

3. Results and Evaluation

We were able to not only design and code VRATTACK, but also merge this game with Oculus Rift Head Mount Display technology to provide the immersive experience we promised to our users. However, in order to reach the summit of the mountain of success, we needed to climb its rough path of challenges. Firstly, the Novint Falcon was the biggest challenge we overcame as it did not function with our computer despite consultation sessions with experts on the matter. We tried different operating system and different computers, but results were always same. After switching to Oculus Rift Head Mounted Display, everything went smoother because of the support of the both Unity and Oculus community. Later on, in developing the game itself, we encountered many challenges.

Challenges

Camera

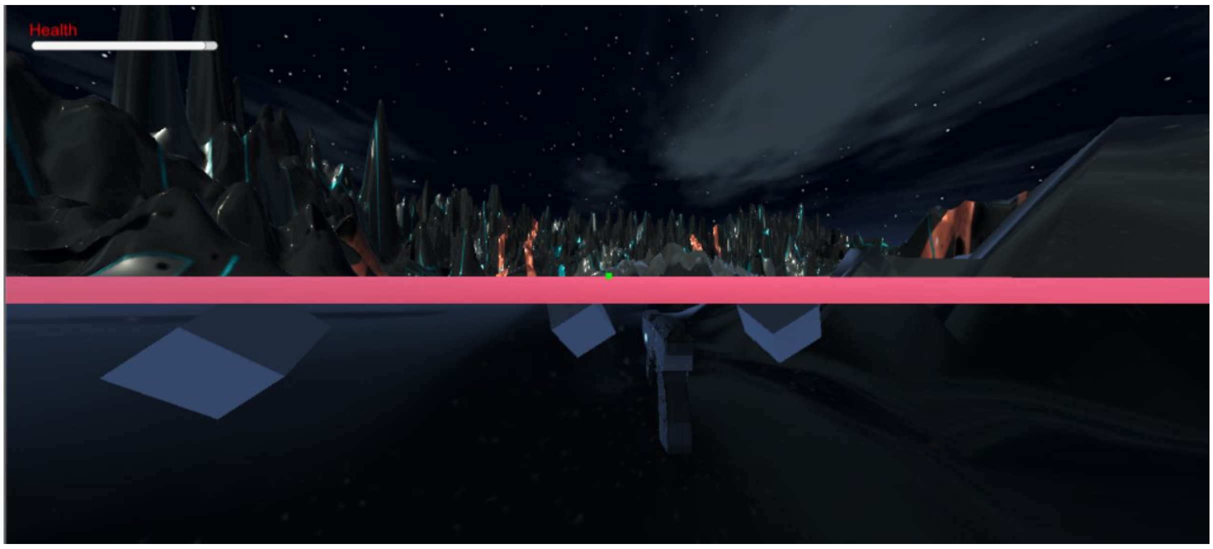


Image 14: Red plane is the camera plane

When setting up the VR camera in Unity, we encountered the first challenge: by default its view, the camera was looking horizontally to the ground. So in the game view, the player was seeing both under and top of the ground (Image 14). Our solution was creating two planes, one on top of the other and making the top plane transparent. The lower plane served as the floor for the environment (Image 15). By placing the camera on the top plane, we were able to fix the camera and manipulate its position to our desire.

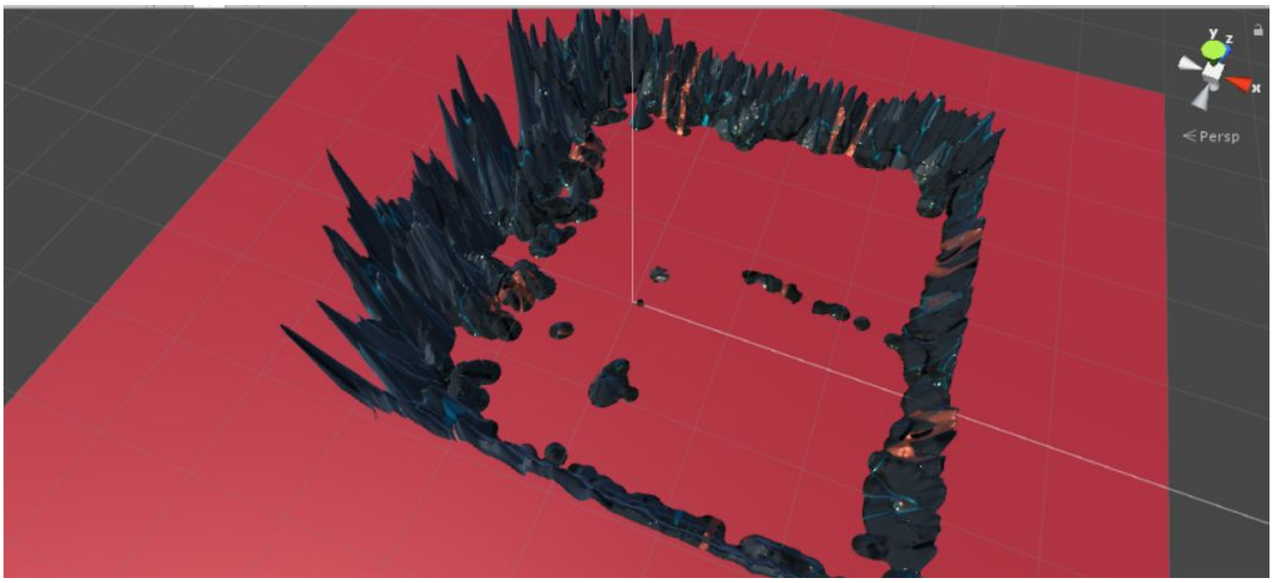


Image 15: Red plane is the actual transparent layer in the game

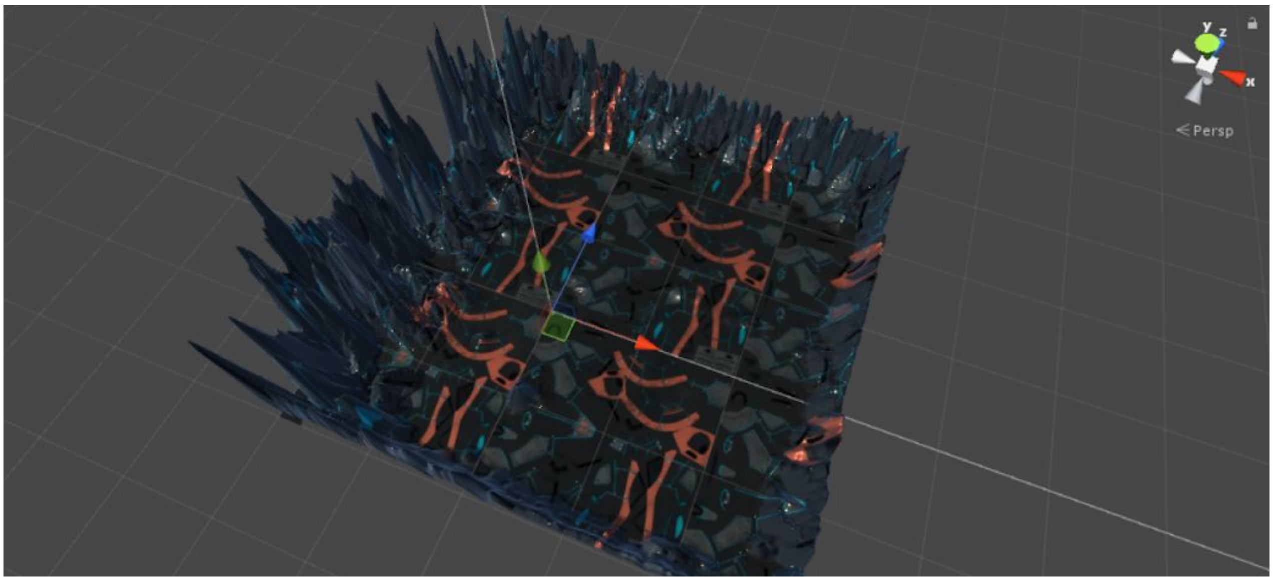


Image 16: The lower plane is used as the arena in the game

By that way, the actual camera stays on the top transparent plane and all other aliens are moving according to the lower plane. We tried to solve the problem by not allowing camera to drop below the top plane. However, that solution didn’t work and we decided to choose the more practical solution.

Arena

The combat arena also posed a problem. Initially, it was two small. Also, it didn't have any mountains and/or obstacles for the user. As a result, we increased its size and we implemented mountains and valleys by using a gun texture, which unexpectedly gave the appearance of an incredible space arena.

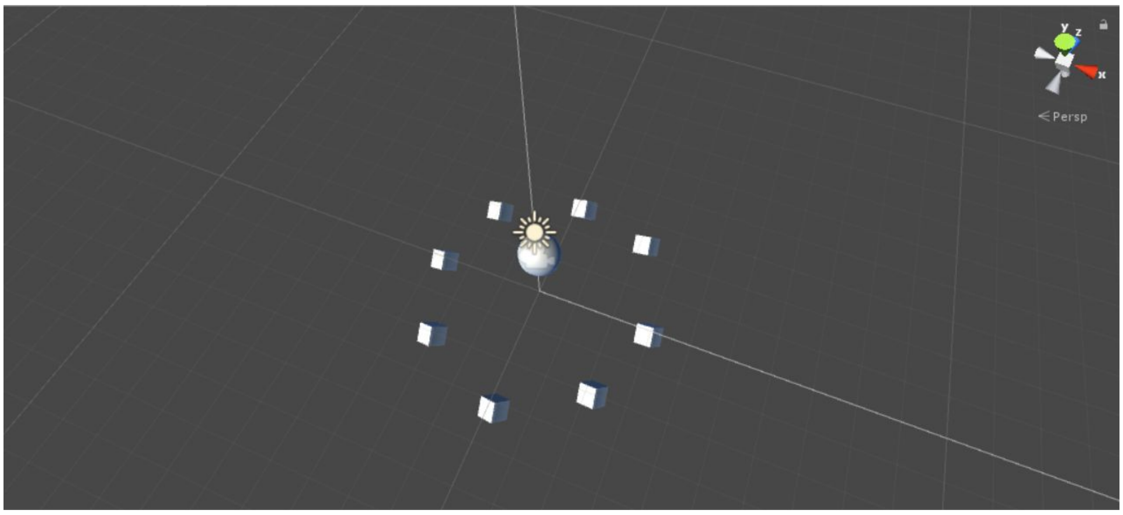


Image 17: The first arena demonstration with gun on the center and aliens on the outside

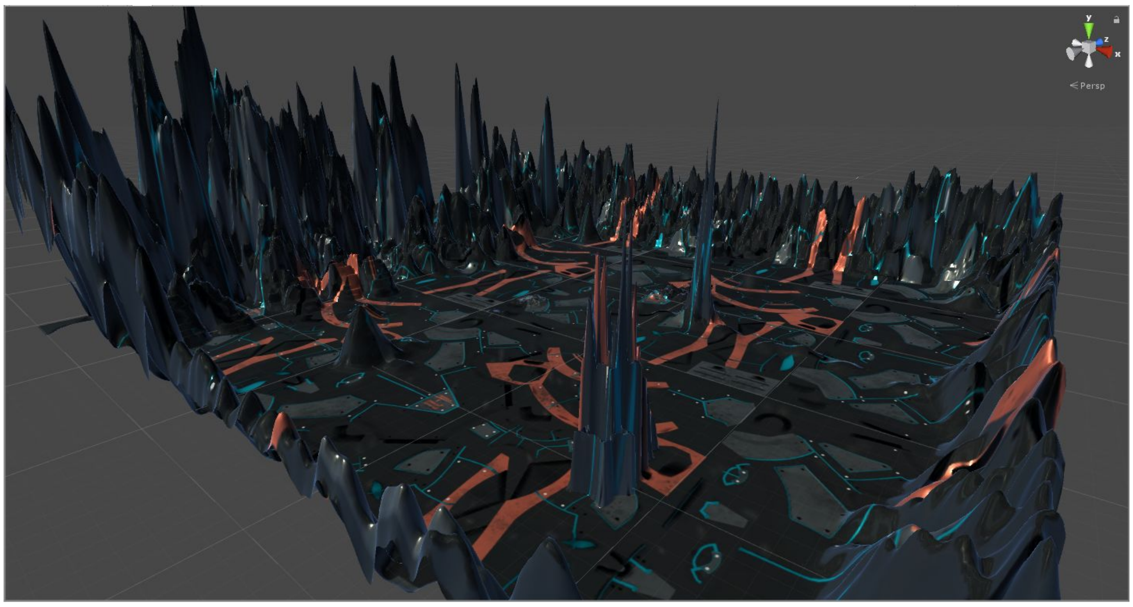


Image 18: The bigger arena

Enemies

In the game we started with cubes as the aliens. Later, we replaced them with the alien figures. Moving those figures was a big challenge for us, especially in moving in the direction of the game character. We solved this problem by, drawing vectors between the game character and the aliens and calculate its directional vector and then multiply it with preset speed. Than, we make the aliens move across this vector. This vector changed once per frame; thus the aliens are always moving toward the player.

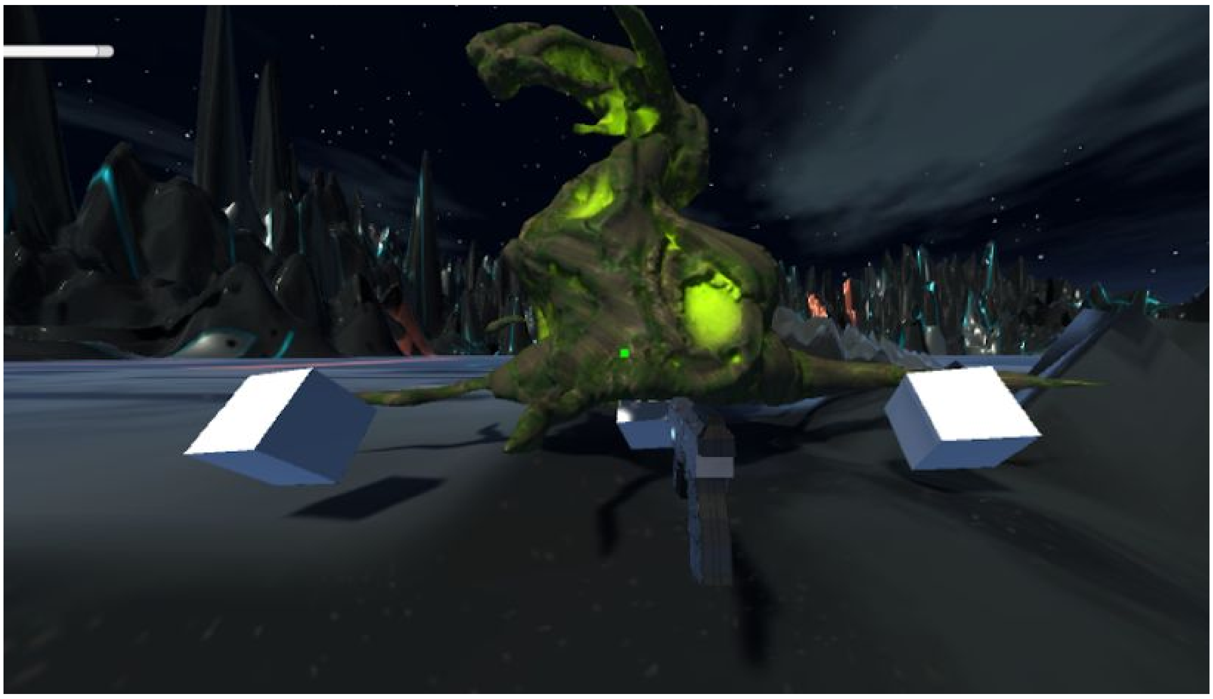


Image 19: An alien moving towards the game character

Killing the aliens with the gun was another challenge. When we fired the gun, the beams were not moving directly towards the aliens. This problem was solved by obtaining the forward direction of the camera and fire the laser beams in that direction. The generation and destruction of the laser beams was also challenging since there should always be a “reference” laser beam in order to create more, which should not be destroyed. The solution was to create indexes of the laser beams: the first one, which is the original laser beam, was not moving with index=1 while other clones of it were. It was also buried under the ground so it cannot be seen.

Spawning new aliens automatically was also difficult. Since we needed to increase the difficulty level of the game as it progress, which required faster generation speed. This problem was solved by using scripts that shortened the spawn time for each spawn of the alien.

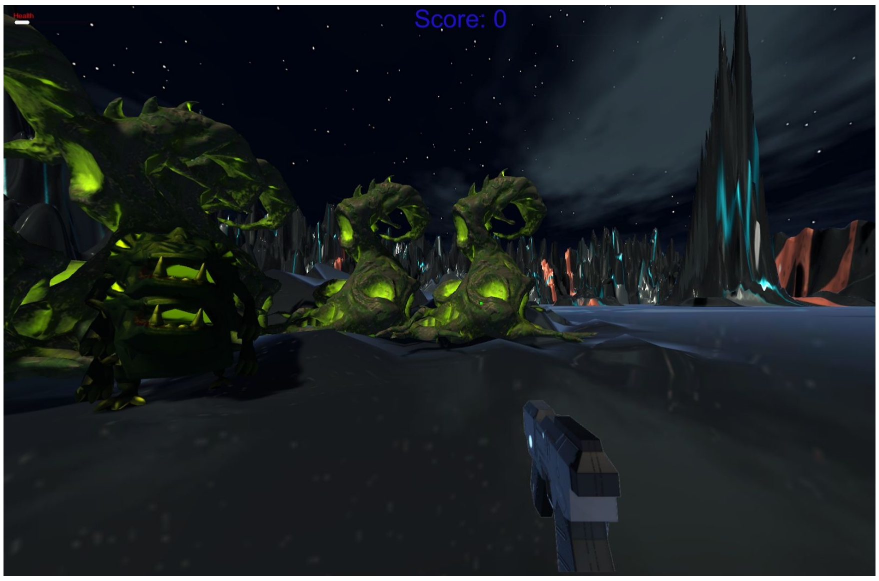


Image 20: Aliens moving towards the game character

Game Menu

Creating a game menu was not easy because there was no easy option for creating game menus, we had to figure it out. We came with the solution of using different scenes and creating one of the scenes as the game menu. During the build of the game, we were able to choose what scenes should come first. By that way, we linked the scenes each other and created the game menu.

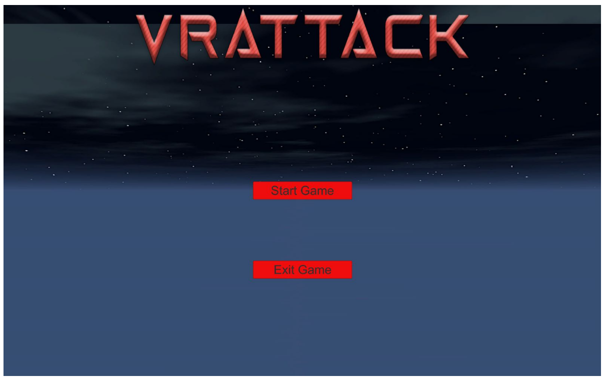


Image 21 : Game Menu

Control

In this game, the actions of the player were achieved by using Xbox controller, whose inputs were not integrated in the official utilities for Unity of Oculus Rift. We had to manually setup the inputs in Unity and integrate the inputs from the controller with the scripts in the official utilities. For every movement of the player, the game could then take inputs from either the Xbox controller or the keyboard. Since the official utilities' scripts were complicated, the process took us a long time to perfectly combining them together. In addition, the user can also use one of the joysticks on the controller to change the direction he/she is facing, like the rotation of the head in the Oculus Rift. These approaches made VRATTACK not only a pure VR game, but also a multi-device compatible FPS game.

Errors & Debugging

Syntax Errors

During the programming we faced with many many syntax errors. Sometimes it was because of the missing parenthesis, semicolon or mismatching characters. Even, we faced with public and private variable problems. We eliminated those syntax errors by collaboratively checking and debugging the code visually.

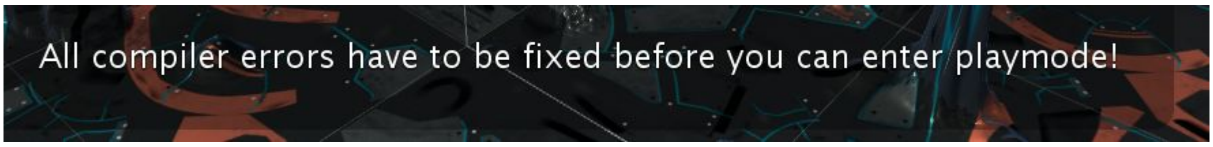

All compiler errors have to be fixed before you can enter playmode!

Image 22: Unity syntax error warning

Thanks to Unity's syntax error warnings, debugging was easier because sometimes it was telling what was missing. So, it was very helpful for us and saved a lot of time.

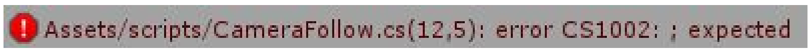
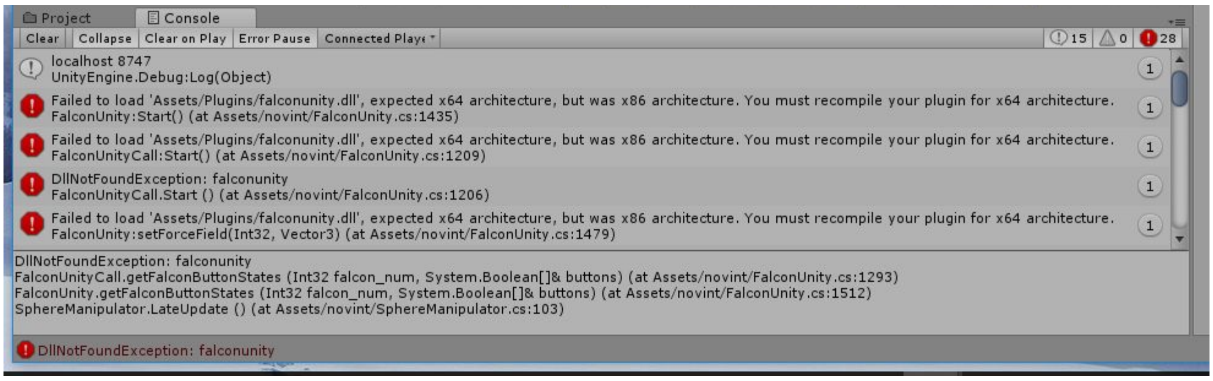

Assets/scripts/CameraFollow.cs(12,5): error CS1002: ; expected

Image 23: Unity syntax error warning with suggestion

falconunity.dll

Unfortunately, we spent a lot of time on trying to fix this error for the Novint Falcon gaming controller. This error was the biggest error that we faced. Actually the file was there, in the folder. We tried to fix the drivers and we couldn't solve the problem.



```
localhost 8747
UnityEngine.Debug:Log(Object)
Failed to load 'Assets/Plugins/falconunity.dll', expected x64 architecture, but was x86 architecture. You must recompile your plugin for x64 architecture.
FalconUnity:Start() (at Assets/novint/FalconUnity.cs:1435)
Failed to load 'Assets/Plugins/falconunity.dll', expected x64 architecture, but was x86 architecture. You must recompile your plugin for x64 architecture.
FalconUnityCall:Start() (at Assets/novint/FalconUnity.cs:1209)
DllNotFoundException: falconunity
FalconUnityCall.Start () (at Assets/novint/FalconUnity.cs:1206)
Failed to load 'Assets/Plugins/falconunity.dll', expected x64 architecture, but was x86 architecture. You must recompile your plugin for x64 architecture.
FalconUnity:setForceField(Int32, Vector3) (at Assets/novint/FalconUnity.cs:1479)

DllNotFoundException: falconunity
FalconUnityCall.getFalconButtonStates (Int32 falcon_num, System.Boolean[]& buttons) (at Assets/novint/FalconUnity.cs:1293)
FalconUnity.getFalconButtonStates (Int32 falcon_num, System.Boolean[]& buttons) (at Assets/novint/FalconUnity.cs:1512)
SphereManipulator.LateUpdate () (at Assets/novint/SphereManipulator.cs:103)

DllNotFoundException: falconunity
```

Image 24: The missing dll file error

4. Conclusion and Future Work

After days of coding and discussing, we were able to bring to life VRATTACK. VRATTACK is a single-player game where users take on the role of Ali, an interplanetary space fighter that must destroy all the vicious, brain-eating aliens that are in his way and prevent him from carrying on his planetary duties as the Chief Police Officer of the Milky Way Galaxy.

The game starts with a menu display that shows two options: Play Game and Exit Game, all under the captivating logo of VRATTACK. By selecting Play Game, users will then jump onto Ali's body on a space arena where all brain-eating aliens will be approaching. To survive, the user must shoot at the aliens that are approaching. As the user kills them, the Score counter above the screen will increase as the number of aliens also killed increases. The user must try and survive as long as he can, and the longer he survives by killing more aliens, the more the number of aliens that will spawn in the map, making it more difficult for the user to survive.

We believe we achieved our objectives with VRATTACK. Those objectives were:

- Utilize the Oculus Rift Head Mounted Display as the medium through which users will experience the digital world we create
- Merge users with the game character through the use of natural body movements
- Proving a single player, first person shooter experience
- Immerse users in the world of space combat
- Arouse happiness among users due to the highly entertaining characteristics of the game
- Create a sense of self-accomplishment in the users who will try and beat their own previous performances
- Increase the spirit of camaraderie among the users and group members through the use of this highly creative gaming application

This project could later be utilized as a model for DIY, which stands for Do It Yourself. In addition, Novint Falcon can be revised and added to the project by further working on the project without time constraints. We hope this project can inspire future aspiring game developers not to be afraid about the daunting task of creating their own game but rather, embrace the challenge fiercely and jump on it without necessarily having a Computer Science or Engineering Degree. In addition, this game can also serve as a foundation for further iterations of Space Combat games. We hope that this game revolutionizes the Gaming Industry with its captivating interface and its jaw-dropping features and with the help of other game developers, we hope to create further iterations of VRATTACK that are more marketable and more powerful in their gaming potential.

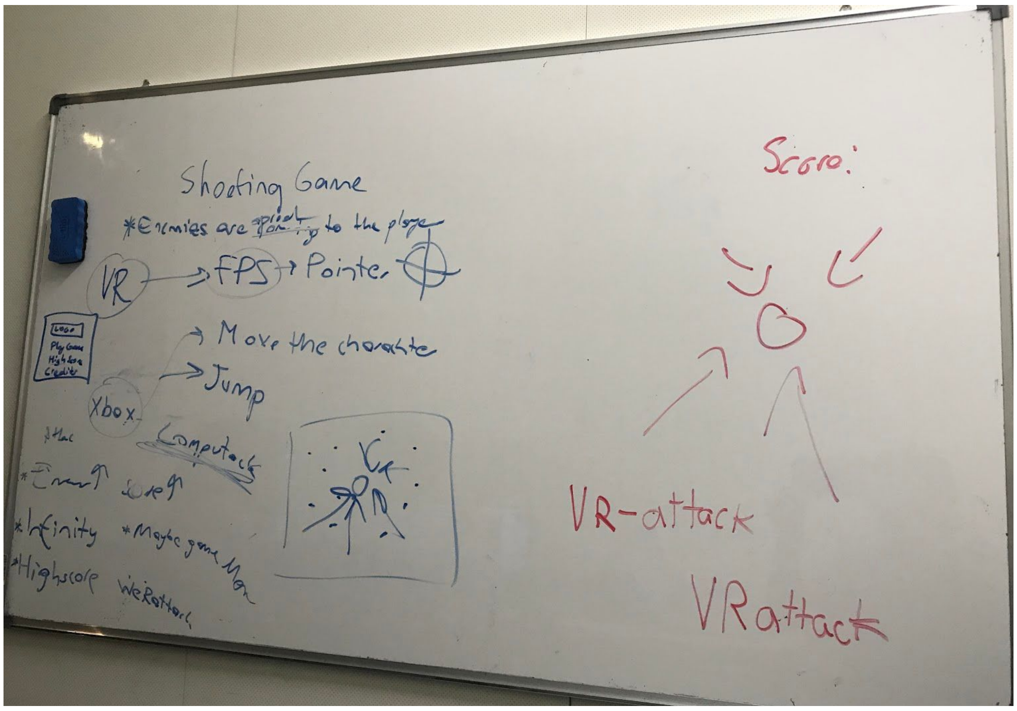


Image 24: This picture shows the result of our brainstorm session for game development

After this game come to fruition, we were beyond ecstatic to see how the users we chose to try out the game where satisfied by its interface. They told us that this game was similar to those created by companies who spent years and millions of dollars in delivering similar products, which further heighten their astonishments as it was unfathomable for them to believe that a trio of freshman students developed this product. Therefore, this project delivered on its promises in the introduction section, which was to immerse users in the world of spacecraft in a highly interactive and entertaining way.

5. Reflection on Learning

This project opened up a vast plethora of learning opportunities for us, the creators. First and foremost, we implemented the programming skills we acquired throughout the semester, something some of us had never experienced before in a project of this magnitude which forces us to create a real project with real-life implications as opposed to our previous assignments that at their can best be described as merely hypothetical.

We learned the intrinsic details of Novint Falcon when our project was based on it and all of the technicalities that come with utilizing this highly interesting technology, included but not limited to, its haptic modalities, human-computer interaction through haptic technology, and installation of drivers about the device. We learned how to utilize the platform Unity for game developing, a platform we will definitely be using in the future. We learned how to borrow different APIs from different sources to improve the features of our game. We learned how to use our transferable programming skills as we were capable of programming in C#, a language that is different from C++, the one we have been used to use throughout the semester. We learned how to utilize the online forums such as stackoverflow.com and online communities in order to clear the bugs in our coding and any other misconception we had about the Unity Platform itself. Most importantly, we are now capable of launching and implementing our own project to the market because we simulated such process with VRATTACK by creating brainstorm stations, deciding on the name of the product, defining its characteristics, and presenting to random customers that can pinpoint its mistakes for us so we can repair them in an ever-involving process of product improvement.

In addition, we also learned the pivotal skill of power-pitching a product in a limited amount of time to a sea of new faces who will determine the validity of the product as VRATTACK will be presented to Professor Mohamad Eid and the entire class. In preparation for the aforementioned, we sharpened our skills in various presentation tools like PowerPoint and Adobe Premiere Pro to deliver an efficient pitch that can excite our “possible customers” on VRATTACK. But most importantly, we learn how to be passionate and work as a team on a product that we truly believe in, skills that are imminent as most real-life engineering projects require teamwork and coordination.