Author : Matthis Le Texier
Date : 12/10/2019

CS 485 - MidTerm Project Write Up

Team members :

- Harshit Lamba
- Aymeric Geoffrey Figuie
- Cedric Cevag Karaoglanian
- Matthis Pierre Jean Le Texier

1) Unity knowledge

The first thing I had to do is coming back to Unity and remind myself the architecture of a game and how to work with Unity.

I started reading some guide (that I cannot find anymore) about Unity's way of working. That's to say, how a game is builded in. A game is composed of one or more **scenes** that can be linked together and represents different parts of the game.

So the first thing is to create a new scene where the major part of the game will be played. We can add objects to it, such as lights, 3D models, 3D sounds etc. Each of them are considered like **GameObject** and we can attach to them some **components** that have a specific effect on it.
For instance, let's say we add a simple 3D plan to the scene, we can add a **rigidbody** to it to handle physics like collisions with others GameObject in the scene.

Another interesting thing and important one is the possibility to create a parent-child link between GameObjects. Once an object is the child of another, when we move the parent in the scene along X, Y or Z axes, the child will also be move accordingly. That’s one of severals characteristics of parent-child relationship.

I learned plenty of others things on Unity like :

- UI system with viewports,
- add GameObject access in a script attached to another GameObject,
- add parameters to a script and change this values directly in the Unity view
- updating a GameObject position on user input
- etc

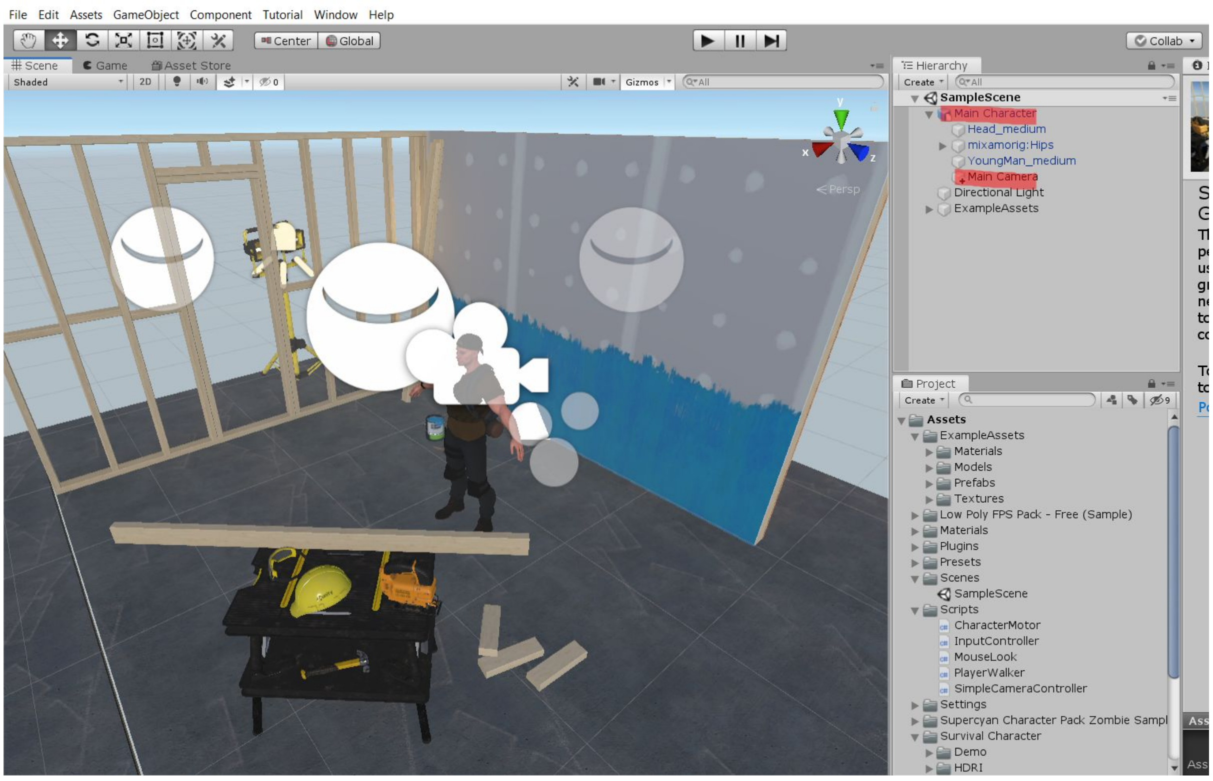
2) Game proposal contribution

So we decided finally to go for a survival zombi game because we had to make a choice between all of our ideas we got after a brainstorming.
I participated in the decision of making a first person shooter, and one of the reason was that is simpler because we only have to find 3D model of arms since the player will only see that, instead of finding a full animated character.

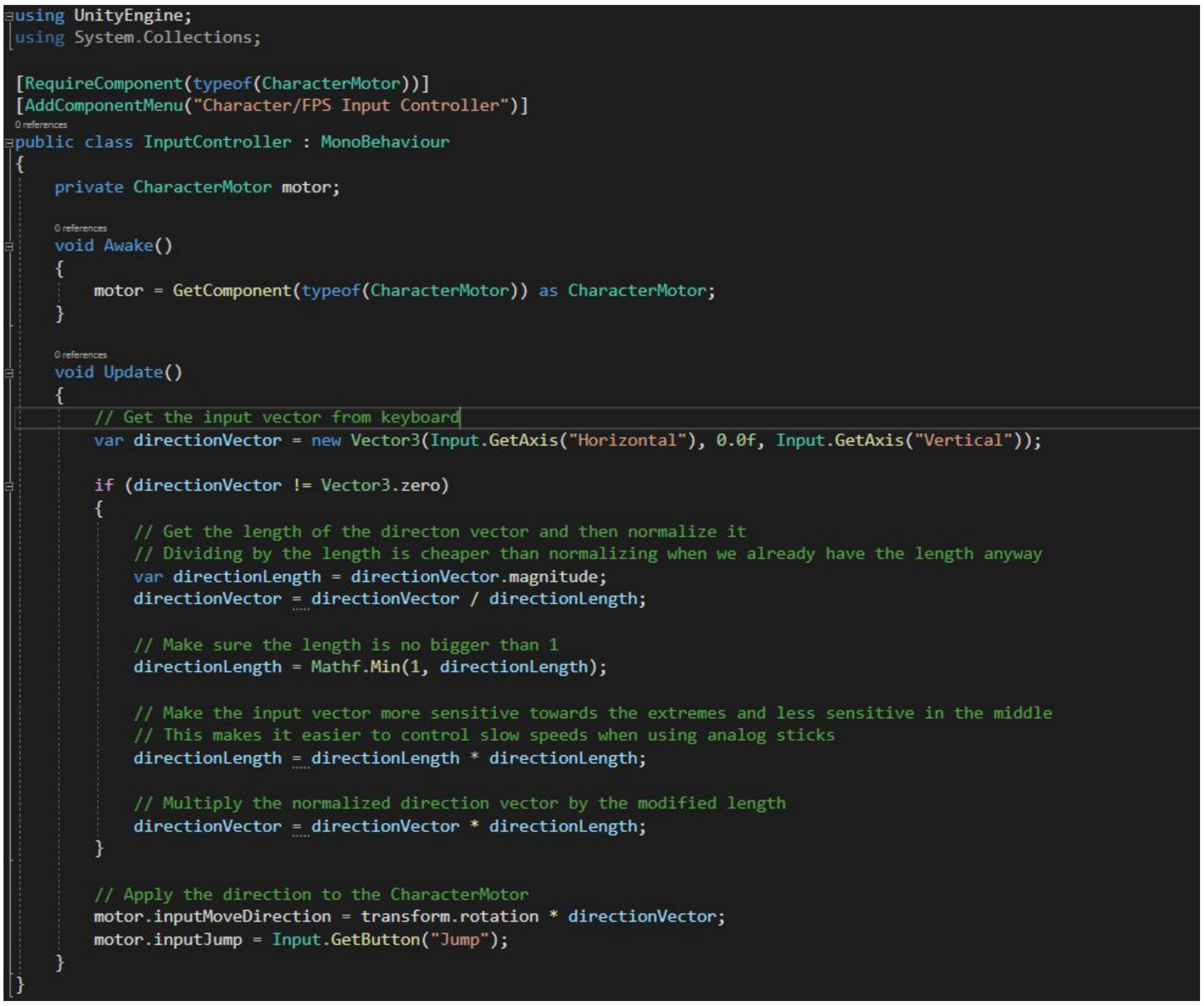
3) Implementation

Form now, I have been working exclusively on player movements scripting.
So, I did a simple scene with a player character GameObject where I put the camera as a child of it and positioned it relatively to make a realistic person view.

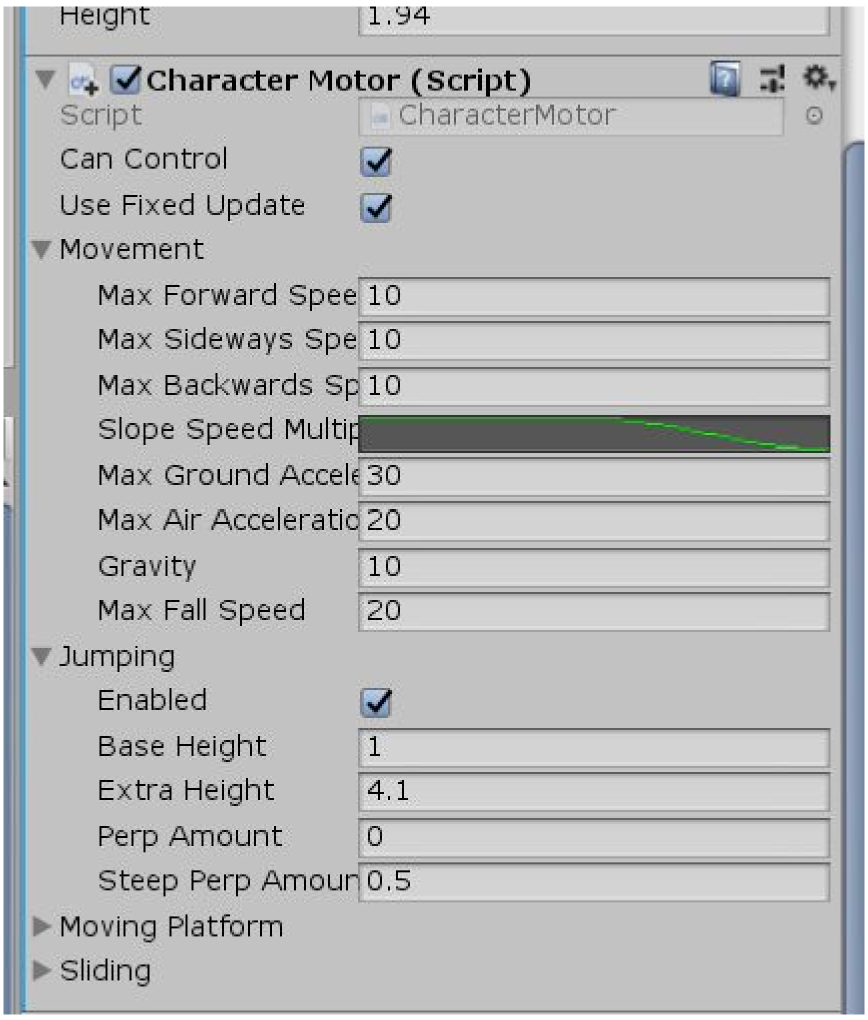
Then I had to add a script to the character and get the user inputs to move the character accordingly.



I first worked on the player movements on the map, the simplest way possible and ended with this script :

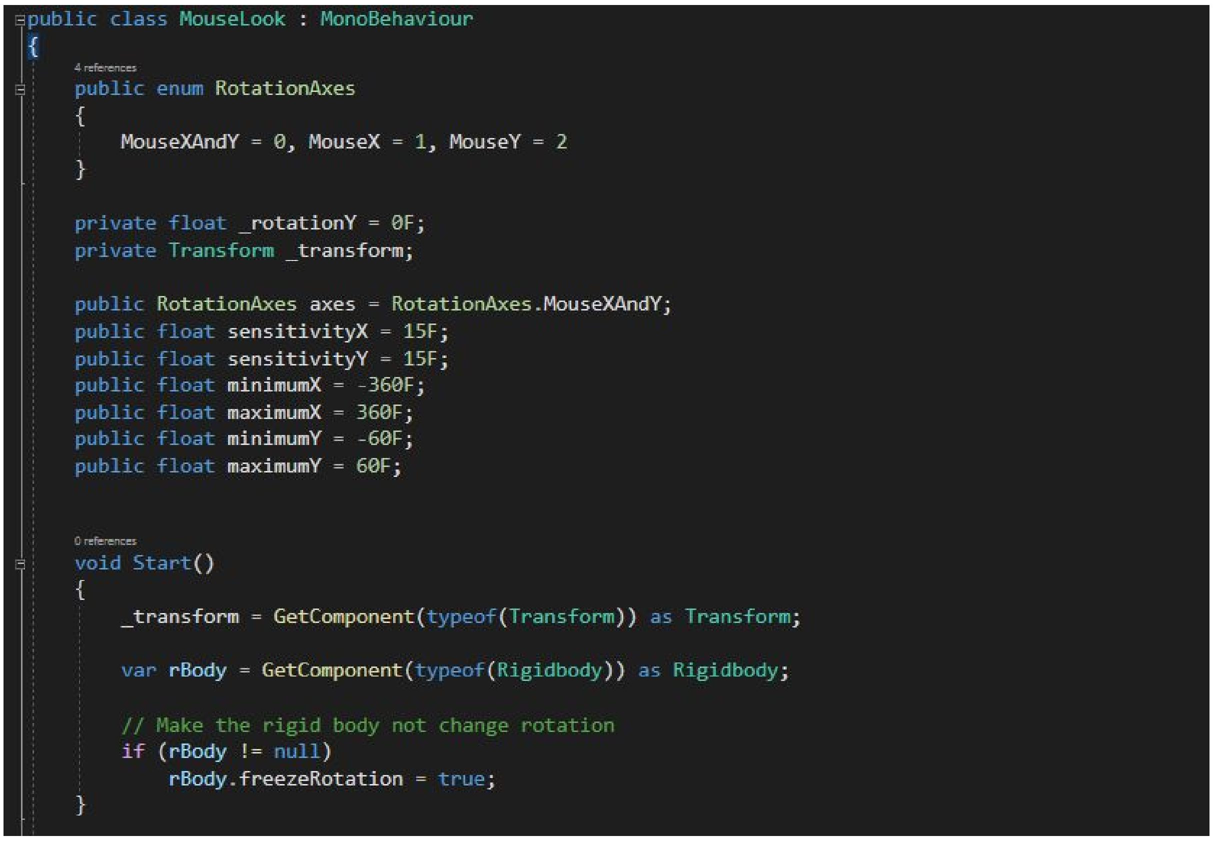

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(CharacterMotor))]
[AddComponentMenu("Character/FPS Input Controller")]
0 references
public class InputController : MonoBehaviour
{
    private CharacterMotor motor;

    0 references
    void Awake()
    {
        motor = GetComponent(typeof(CharacterMotor)) as CharacterMotor;
    }

    0 references
    void Update()
    {
        // Get the input vector from keyboard
        var directionVector = new Vector3(Input.GetAxis("Horizontal"), 0.0f, Input.GetAxis("Vertical"));

        if (directionVector != Vector3.zero)
        {
            // Get the length of the directon vector and then normalize it
            // Dividing by the length is cheaper than normalizing when we already have the length anyway
            var directionLength = directionVector.magnitude;
            directionVector = directionVector / directionLength;

            // Make sure the length is no bigger than 1
            directionLength = Mathf.Min(1, directionLength);

            // Make the input vector more sensitive towards the extremes and less sensitive in the middle
            // This makes it easier to control slow speeds when using analog sticks
            directionLength = directionLength * directionLength;

            // Multiply the normalized direction vector by the modified length
            directionVector = directionVector * directionLength;
        }

        // Apply the direction to the CharacterMotor
        motor.inputMoveDirection = transform.rotation * directionVector;
        motor.inputJump = Input.GetButton("Jump");
    }
}
```

And then I found a basic Unity script for player movements, integrating jump and some other useful variables you can modify directly on Unity to make the script works the way you want.

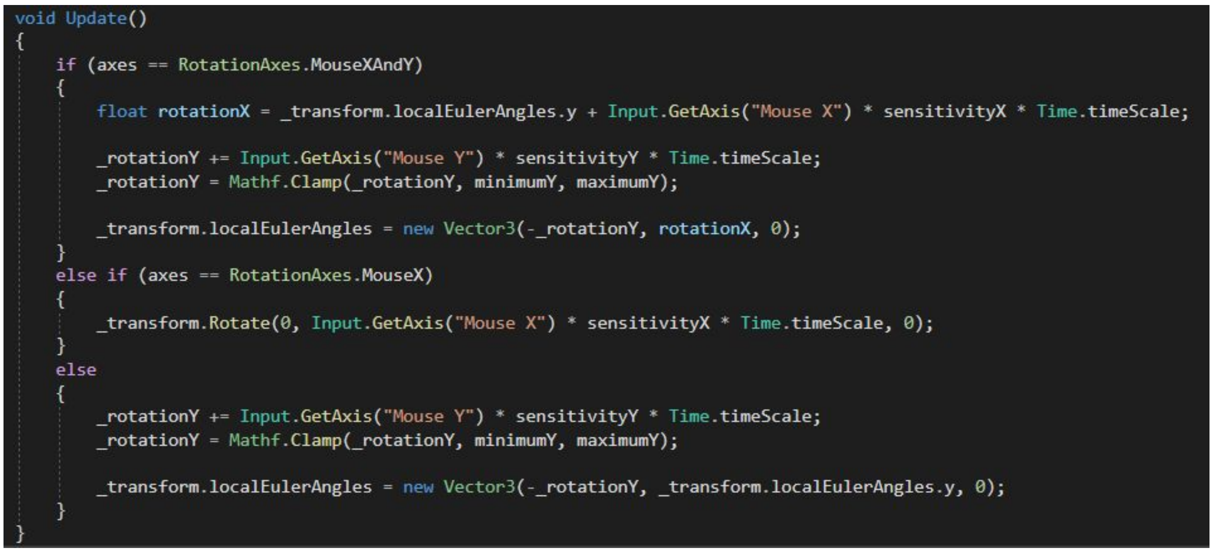

Height 1.94
Character Motor (Script)
Script CharacterMotor
Can Control ✓
Use Fixed Update ✓
Movement
Max Forward Spee 10
Max Sideways Spe 10
Max Backwards Sp 10
Slope Speed Multip
Max Ground Accele 30
Max Air Acceleratio 20
Gravity 10
Max Fall Speed 20
Jumping
Enabled ✓
Base Height 1
Extra Height 4.1
Perp Amount 0
Steep Perp Amoun 0.5
Moving Platform
Sliding

In this way, we can configure as we want for the game later, the possibilities of movement for our character (movement speed, jump height…).

The second thing I work on, is the view movements. It is an FPS so we have to be able to move the view with the mouse.
I had some problems with that since it's a lot of mathematical problematics and I tried a lot of things but it was always giving the nausea and that's absolutely what we don't want as a FPS game !
To fix this, I have to get information on how to handle this and I finally get something working very smooth and with the sensitivity configuration enabled :



```
public class MouseLook : MonoBehaviour
{
    4 references
    public enum RotationAxes
    {
        MouseXAndY = 0, MouseX = 1, MouseY = 2
    }

    private float _rotationY = 0F;
    private Transform _transform;

    public RotationAxes axes = RotationAxes.MouseXAndY;
    public float sensitivityX = 15F;
    public float sensitivityY = 15F;
    public float minimumX = -360F;
    public float maximumX = 360F;
    public float minimumY = -60F;
    public float maximumY = 60F;


    0 references
    void Start()
    {
        _transform = GetComponent(typeof(Transform)) as Transform;

        var rBody = GetComponent(typeof(Rigidbody)) as Rigidbody;

        // Make the rigid body not change rotation
        if (rBody != null)
            rBody.freezeRotation = true;
    }
```



```
void Update()
{
    if (axes == RotationAxes.MouseXAndY)
    {
        float rotationX = _transform.localEulerAngles.y + Input.GetAxis("Mouse X") * sensitivityX * Time.timeScale;

        _rotationY += Input.GetAxis("Mouse Y") * sensitivityY * Time.timeScale;
        _rotationY = Mathf.Clamp(_rotationY, minimumY, maximumY);

        _transform.localEulerAngles = new Vector3(-_rotationY, rotationX, 0);
    }
    else if (axes == RotationAxes.MouseX)
    {
        _transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityX * Time.timeScale, 0);
    }
    else
    {
        _rotationY += Input.GetAxis("Mouse Y") * sensitivityY * Time.timeScale;
        _rotationY = Mathf.Clamp(_rotationY, minimumY, maximumY);

        _transform.localEulerAngles = new Vector3(-_rotationY, _transform.localEulerAngles.y, 0);
    }
}
```

That’s it for the player controls for now.
I planned to work then on weapons handling (firing, aiming, ammunition system…) but someone else in the team want to work on it.

That’s why I took a break with scripting and started to think about the main gameplay.

I ended up with the idea that the player have obviously to survive the zombies waves. However, I did not want that much to make a infinite game with the only objective to survive the longest time possible.

That’s why I thought about a kind of escaping way. The player will have to find on the maps different things to repair something and escape with it, a spaceship for example.

So we give the player a different main objective, and if player succeed in and want to continue playing, he can challenge himself and try to escape in less amount of time, in other words, in less waves.

4) Next work

I schedule to continue my work on the game design since my teammates are already on the main tasks.
Nevertheless, I can be there for them as a support on different subjects and help them to make sure the project goes ahead.

I’m also interesting in working on the game UI including the start menu with “Play”, “Settings” etc and the HUD in-game with the player’s health, the weapon’s ammunition, the waves counter...