Game Development

Assignment: Individual reflections

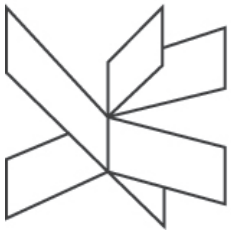
Jordi Rafael Lazo Florensa

17th May 2022

Software Technology Engineering

Teacher: Jakob Knop Rasmussen

IT-GMD1-S22



VIA University College

1 Individual reflections

For this assignment of the IT-GMD1-S22 course I have decided to create a video game that is strongly inspired by two classic video games that I played during my childhood. These are:

- *Super Mario Bros.* (1985) : Mario and Lugi traverse side-scrolling stages while avoiding hazards such as enemies and pits with the aid of power-ups such as the Super Mushroom, Fire Flower, and Starman.
- *Sonic the Hedgehog* (1991): the gameplay involves collecting rings as a form of health, and a simple control scheme, with jumping and attacking controlled by a single button.

The game consists of controlling a character who has to pass all the available levels and at the same time get as many coins as possible. To do this, the character will encounter different fixed and movable platforms and enemies that will try to kill him. Although if you don't want to ignore the enemies you can always jump on them to kill them. Of course, there is no need to worry since the character has unlimited lives.

The inclusion of coins has been inspired by the rings in the *Sonic the Hedgehog* and the coins of *Super Mario Bros.* and the ability to kill enemies by jumping over them has been inspired by this too.

In order to create this video game, all the knowledge that has been taught during the course classes has been used. The implementation of scripts has been essential to be able to control the player. For the movement of floating platforms, the application of vectors has been essential. In order for the player to fall off the platforms and not get stuck in their corners, it has been necessary to use predefined physics in Unity. Background music and specific audios have also been added depending on the actions made by the player. Finally, a small user interface has been added that allows you to start and end the game, as well as see the coins obtained.

Finally, I will proceed to comment on how the project is structured and what the longest script in the game consists of. As you can see in the image below, the project has been organized into the following folders.

- *CasualGamesSounds*: the sounds of the player's interactions with the other objects have been stored in this folder.
- *Fonts*: in this folder the font used in the user interface has been stored.
- *Materials*: this folder contains the different materials applied to the different objects.
- *Music Tracks*: this folder contains some background songs.
- *Physics Material*: this folder contains the physics applied to the platforms.
- *Prefabs*: this folder contains the game objects created.
- *Scenes*: this folder contains the different scenes (levels) created.
- *Scripts*: this folder contains all the scripts that are used in the game.

- *Textures*: this folder contains the textures added in the video game (downloaded from the internet).

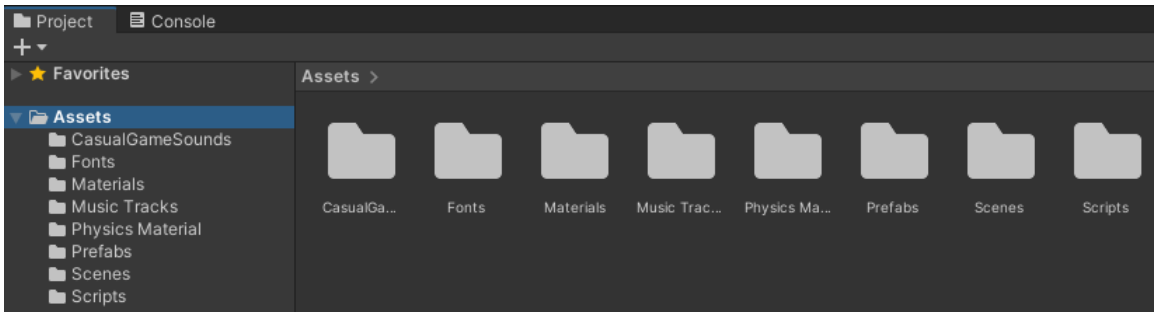


Figure 1: *Project folder structure.*

```
1 reference
public class PlayerMovement : MonoBehaviour
{
    6 references
    Rigidbody rb;
    2 references
    [SerializeField] float movementSpeed = 6f;
    1 reference
    [SerializeField] float jumpForce = 5f;
    1 reference
    [SerializeField] Transform groundCheck;
    1 reference
    [SerializeField] LayerMask ground;
    1 reference
    [SerializeField] AudioSource jumpSound;
    // Start is called before the first frame update
    0 references
    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        float horizontalInput = Input.GetAxis("Horizontal");
        float verticalInput = Input.GetAxis("Vertical");

        rb.velocity = new Vector3(horizontalInput *movementSpeed,rb.velocity.y, verticalInput * movementSpeed);

        if(Input.GetButtonDown("Jump") && IsGrounded()){
            Jump();
        }
    }
    2 references
    private void Jump(){
        rb.velocity = new Vector3(rb.velocity.x,jumpForce,rb.velocity.z);
        jumpSound.Play();
    }

    0 references
    private void OnCollisionEnter(Collision collision){
        if(collision.gameObject.CompareTag("Enemy Head")){
            Destroy(collision.transform.parent.gameObject);
            Jump();
        }
    }

    1 reference
    bool IsGrounded(){

        return Physics.CheckSphere(groundCheck.position,.1f,ground);
    }
}
```

Figure 2: *PlayerMovement class.*

The *PlayerMovement* class is the most complex of all since it is in charge of moving and interacting with the environment. In this class, in the update function, the player's movements and the jump are specified. There are also two extra functions *OnCollisionEnter* which destroys the enemy object, as long as it is the head, and the *IsGrounded* function which checks at all times if the player is in contact with the ground, otherwise he cannot jump.