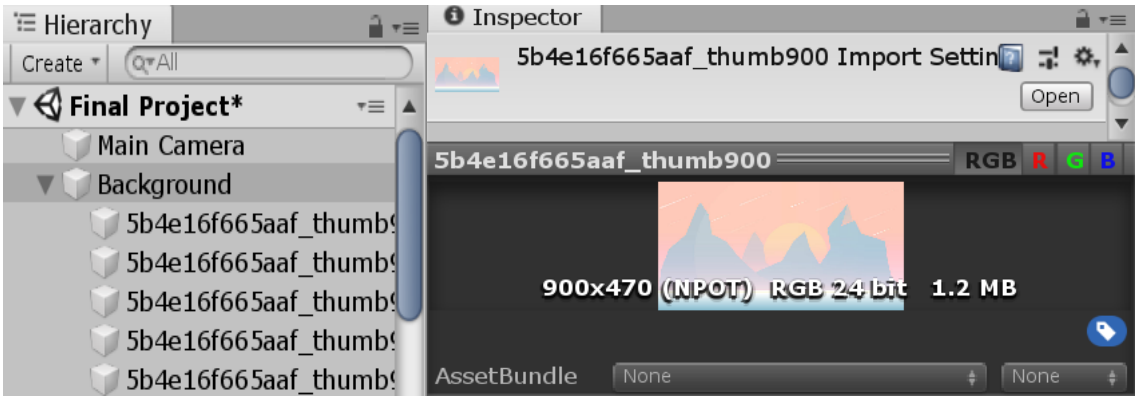
Specification Document

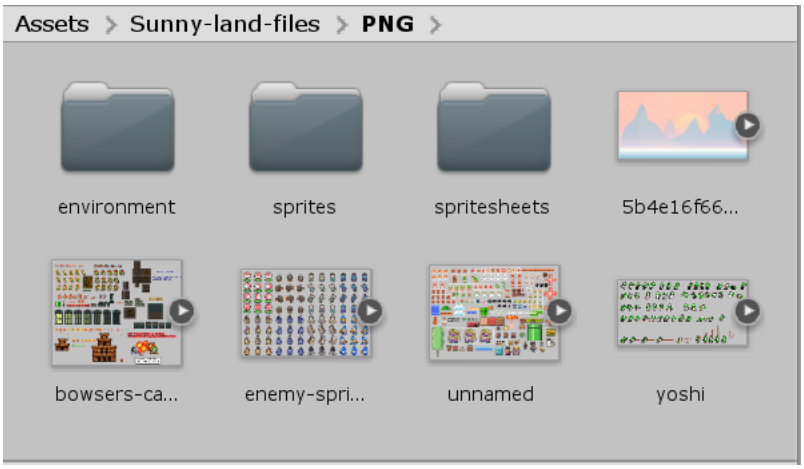
For the Project in Unity I have used Unity, which is a great tool for the development of videogames as it is really intuitive and easy to use. In my case, I have created an easy 2D platform game with a player who has to pass through some platforms and enemies to reach the flag. I am going to explain the main aspects of the project:

First of all, I created a proper scenario for the character and enemies. For that I had to adjust the Main Camera, a component that is already in the project to manage what is seen in the Gameplay.

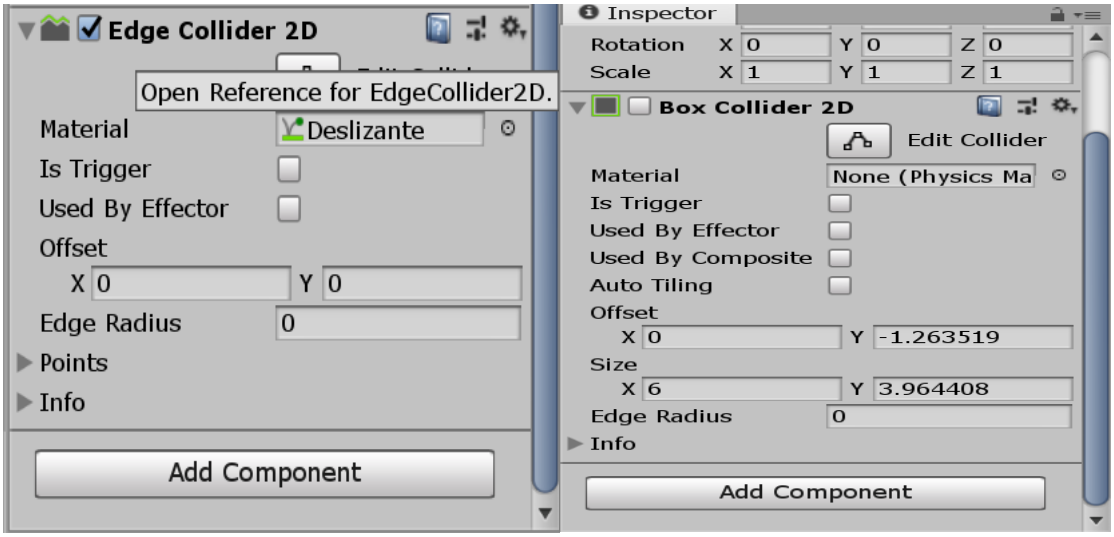
As you can see, there are X and Y components to delimitate the distance the camera has to cover and, as I decided that my videogame would be linear, I only touched the X axis so that the camera cover from the Min Position(Start of the game) to the Max Position(End of the game).



Next step was to create the background of the game. For that I created an object in which you can store different pictures so that I cover all the extension of the level.

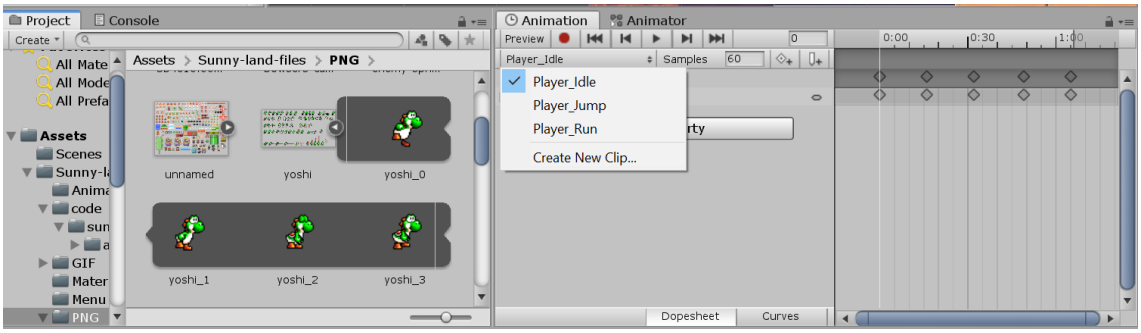


After that I followed the same procedure and I created the rest of components for my level, for that I download some sprite sheets of objects in 2D to help me (also for the characters I am going to talk about later).



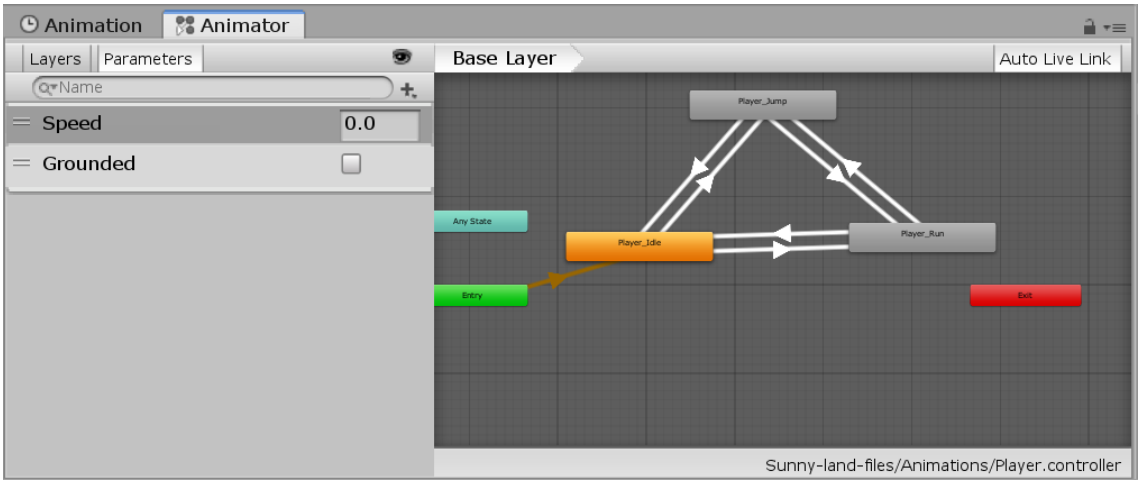
Once I had all the components of the level placed I had to put both Box Collider and Edge Collider to later avoid the characters fall down of the ground and different platforms, and I added a material called “Deslizante” to the Edge collider that helps the characters to avoid getting stucked in the ground.

When I had all components set and placed, I started with the player and its programming. First of all, I created the character as any other object from the level, but then I had to create all its movements and synchronize them through the animator.



As you can see in the picture I had different pictures of the player that I had to put in a sequence so that applying the animation seems like it is moving. I did that for the three basic movements that you can see in the picture (Idle:

When you are not pressing any button; Jump: When you press the Up button; Run: When you press the Right or Left button).



With the animator I joined all the animations of the player and I set two parameters, Ground and Speed. These parameters will help to define the state of the character and for its following programming.

In each of the arrows of above I defined the parameters in this way, depending on the change of state. The case of the picture would be if the character changes from jumping to running, so the parameter grounded is true as it says that the character is touching the floor and the speed is greater than 0.1 so the player is moving. All these values will be set in the scripts of both player and platforms of the level.

When I set all the main functions of our character I started with the script of the character:

PlayerController.cs: Bloc de notas

Archivo Edición Formato Ver Ayuda

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

public class PlayerController : MonoBehaviour
{
public float maxSpeed = 10f;
public float speed = 2f;
public bool grounded;
public float jumpPower = 6.5f;
public bool hit;
public bool death;

private Rigidbody2D rb2d;
private Animator anim;
private SpriteRenderer spr;
private bool jump;
private bool doubleJump;
private bool movement = true;
private int hitCount = 0;
```

I first defined all the variables I needed for the functions that I will explain later. These variables are for the speed, the jumping function and for the damage of the character (With the function will be easier to see their functionality).I also defined some objects of Unity in the script, such as Rigid Body(a component that characters need to be affected by gravity and other forces), Animator and Sprite Renderer(to manage the overlapping of the layers).

PlayerController.cs: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
    // Start is called before the first frame update
    void Start()
    {
rb2d = GetComponent<Rigidbody2D>();
anim = GetComponent<Animator>();
 spr = GetComponent<SpriteRenderer>();
    }

    // Update is called once per frame
    void Update()
    {
anim.SetFloat("Speed", Mathf.Abs(rb2d.velocity.x));
anim.SetBool("Grounded", grounded);

if(grounded){
doubleJump = true;
}

if(Input.GetKeyDown(KeyCode.UpArrow)){
if(grounded){
jump = true;
doubleJump = true;
}else if (doubleJump){
jump = true;
doubleJump = false;
}
}
  }
```

In the Start function, I have initialized the objects I mentioned before. Then, in the Update function I have defined both Speed and Grounded so the program relate them with their animations and apply it on the character. After that I have set with if functions the jumping function of the player, so that it jumps when the up button is pressed.

For the FixedUpdate function, I basically set the movement of the character, in order to apply more or less speed to it and to limit this speed. Also with the if functions containing h </> +- 0.1f I set the player so that it can go to the right and to the left. Finally, I apply the force to the jump so that the player can jump higher or lower depending on the jump power I set for it.

PlayerController.cs: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
}
  }
        void FixedUpdate(){

Vector3 fixedVelocity = rb2d.velocity;
fixedVelocity.x *= 0.75f;

if(grounded){
rb2d.velocity = fixedVelocity;
}
float h = Input.GetAxis("Horizontal");
if(!movement) h = 0;

rb2d.AddForce(Vector2.right * speed * h);

float limitedSpeed = Mathf.Clamp(rb2d.velocity.x, -maxSpeed, maxSpeed);
rb2d.velocity = new Vector2(limitedSpeed, rb2d.velocity.y);

if(h > 0.1f){
transform.localScale = new Vector3(1f, 1f, 1f);
}

if(h < -0.1f){
transform.localScale = new Vector3(-1f, 1f, 1f);
}

if (jump){
rb2d.velocity = new Vector2(rb2d.velocity.x, 0);
rb2d.AddForce(Vector2.up * jumpPower, ForceMode2D.Impulse);
jump = false;
}

Debug.Log(rb2d.velocity.x);
}
```

Later I have defined OnBecameInvisible function to respawn the player in case it goes out from the camera vision, for example if it falls from a platform. The function EnemyJump is to say that the player is jumping and the EnemyKnockBack function is for jumping backwards if the player is hit by an enemy. There is a count to 3 and every enemy hit increases the count, so the player can receive 3 hits before it dies. When it dies , it will respawn in the starting position. I also added a color for the player when it is hit by an enemy.

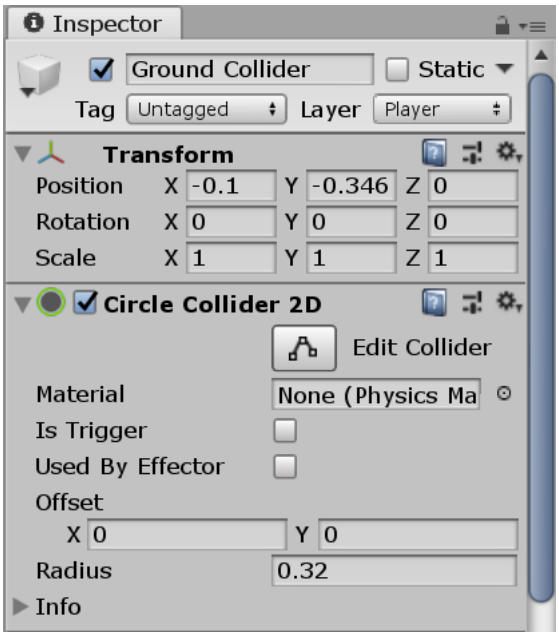
PlayerController.cs: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
}
  void OnBecameInvisible(){
transform.position = new Vector3(-6,0,0);

}
public void EnemyJump(){
jump = true;
}
public void EnemyKnockBack(float enemyPosX){
hitCount++;
        if(hitCount == 3){
                rb2d.velocity = new Vector3(0f, 0f,0f); //speed null
                death = true;
                movement = false;
                Invoke("DeathDelay", 0.5f);
                Invoke("Respawn", 0.7f);
                Invoke("EnableMovement", 0.8f);
                spr.color = Color.white;
                hitCount = 0;}
else{
jump = true;
}
float side = Mathf.Sign(enemyPosX - transform.position.x);
rb2d.AddForce(Vector2.left * side * jumpPower, ForceMode2D.Impulse);

movement = false;
Invoke("EnableMovement", 0.7f);

Color color = new Color(255/255f, 106/255f, 0/255f);
spr.color = color;
}
```

I created EnableMovement function to avoid the player to move while it is jumping backwards by an enemy´s hit. The DeathDelay function is the one which sets the death of the character, so when it gets true is means that the character has received 3 hits. Respawn function, as its name indicates, is for the respawn of the player and finally the instant Death function is implemented for the final boss of the game, in this case Bowser, so that if Bowser touches the player, it will instantly dies no matter how many hits it has.

```
void EnableMovement(){
movement = true;
spr.color = Color.white;
}

 void DeathDelay(){
        death = false;

    }

    void Respawn(){
        rb2d.velocity = new Vector3(0f, 0f,0f);         //speed null
        transform.position = new Vector3(-12.389f,-2.34f,0);    //respawn coordinates
    }
 void InstantDeath(){
hitCount = 2;
}

}
```



After the setting of the player script, I add it a new object with a component called Circle collider. This collider will help the player to detect when it is touching the ground, as it will collide with the other colliders.

When I had this component in the player, I had to add also a new script to program the player with this circle collider so it correctly detects when is in contact with the ground.

CheckGround.cs: Bloc de notas

Archivo Edición Formato Ver Ayuda

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

public class CheckGround : MonoBehaviour
{

private PlayerController player;
private Rigidbody2D rb2d;

    // Start is called before the first frame update
    void Start()
    {
      player = GetComponentInParent<PlayerController>();
      rb2d = GetComponentInParent<Rigidbody2D>();
    }

void OnCollisionEnter2D(Collision2D col){
if(col.gameObject.tag == "Platform"){
rb2d.velocity = new Vector3(0f, 0f, 0f);
player.transform.parent = col.transform;
player.grounded = true;
}
}
```

In this script I first created an object for the player and another for its rigid body. Then I initialize them in the Start function. In the OnCollisionEnter2D function we set the player so that when it touches a platform, it detects it as it is touching ground and change its state.

The OnCollisionStay2D function is designed for hold the player in moving platforms (of which I am going to show the script later).Finally, OnCollisionExit2D is set for when the player jumps or is not touching the ground, so that it notices that is not touching ground and changes its state.

```
    void OnCollisionStay2D(Collision2D col){
if (col.gameObject.tag == "Ground"){
player.grounded = true;
}
if (col.gameObject.tag == "Platform"){
player.transform.parent = col.transform;
player.grounded = true;
}
}
   void OnCollisionExit2D(Collision2D col){
if (col.gameObject.tag == "Ground"){
player.grounded = false;
}
if (col.gameObject.tag == "Platform"){
player.transform.parent = null;
player.grounded = false;
}
}
}
```

Once we have the player all set, we will proceed with the enemies setting and programming. The procedure is almost the same as the one for the player, but enemies only have animation for idle and for walking.

In terms of the script, enemies have a similar one to the player but with some differences. In the script there are also variables for the speed and maximum speed, as an object for the Rigid Body that is initialized in Start function. The Fixed Update function is quite similar to the one in Player.

With respect to the OnTriggerEnter2D function, it is designed to destroy the enemy in case that the player jumps in it. If the player touches the enemy in any other part, it will receive damage. Some enemies have variations of this script.

```
        void OnTriggerEnter2D(Collider2D col){
        if(col.gameObject.tag == "Player"){
        float yOffset = 0.4f;
        if(transform.position.y + yOffset < col.transform.position.y){
        col.SendMessage("EnemyJump");
        Destroy(gameObject);
        }else{
        col.SendMessage("EnemyKnockBack", transform.position.x);

        }

        }
        }
        }
```

```
void OnTriggerEnter2D(Collider2D col){
if(col.gameObject.tag == "Player"){
Destroy(gameObject);
col.SendMessage("EnemyKnockBack", transform.position.x);

}
```

In the case of Bomb enemy, the OnTriggerEnter2D function is changed so that no matter where the player touches the enemy, it will be destroyed and will cause damage to our player.

Archivo Edición Formato Ver Ayuda

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

public class EnemyController : MonoBehaviour
{
public float maxSpeed = 1f;
public float speed = 1f;

private Rigidbody2D rb2d;

    // Start is called before the first frame update
    void Start()
    {
     rb2d = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    void FixedUpdate()
    {
      rb2d.AddForce(Vector2.right * speed);
  float limitedSpeed = Mathf.Clamp(rb2d.velocity.x, -maxSpeed, maxSpeed);
rb2d.velocity = new Vector2(limitedSpeed, rb2d.velocity.y);

    if(rb2d.velocity.x > -0.01f && rb2d.velocity.x < 0.01f){
speed = -speed;
rb2d.velocity = new Vector2(speed, rb2d.velocity.y);
}
if(speed < 0){
transform.localScale = new Vector3(1f, 1f, 1f);
}

else if(speed > 0){
transform.localScale = new Vector3(-1f, 1f, 1f);
}
    }
```

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

public class EnemyControllerPlant : MonoBehaviour
{
   // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void FixedUpdate()
    {

    }
void OnTriggerEnter2D(Collider2D col){
if(col.gameObject.tag == "Player"){
col.SendMessage("EnemyKnockBack", transform.position.x);

}


}
}
```

In the Plant enemy, as you can see, I have deleted all the movement code from FixedUpdate function as I want this enemy to be static. Also this enemy is invulnerable, so the Destroy command is also deleted.

Finally in the Bowser script, I have added the calling to the function InstantDeath that I created in the Player script. This function makes that if the player touches Bowser, it will die no matter how much damage does it has. Also I have made Bowser invulnerable, so there is no Destroy function.

```
void OnTriggerEnter2D(Collider2D col){
if(col.gameObject.tag == "Player"){

col.SendMessage("InstantDeath");
col.SendMessage("EnemyKnockBack", transform.position.x);

}
}
```

I also added some improvements to the level to make it funnier, so I programmed moving and falling platforms and also background music.

PlataformaMovil.cs: Bloc de notas

Archivo Edición Formato Ver Ayuda

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

public class PlataformaMovil : MonoBehaviour
{
public Transform target;
public float speed;

private Vector3 start,end;

    // Start is called before the first frame update
    void Start()
    {
     if(target != null){
target.parent = null;
start = transform.position;
end = target.position;
}
    }

    // Update is called once per frame
    void Update()
    {

    }

void FixedUpdate()
    {
     if(target != null){
float fixedSpeed = speed * Time.deltaTime;
transform.position = Vector3.MoveTowards(transform.position, target
}
if(transform.position == target.position){
target.position = (target.position == start) ? end : start;

}
    }
}
```

PlataformaFalling.cs: Bloc de notas

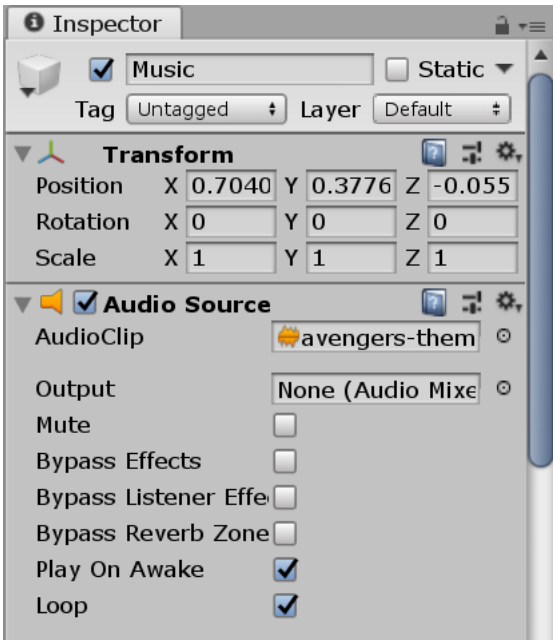
Archivo Edición Formato Ver Ayuda

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

public class PlataformaFalling : MonoBehaviour
{
public float fallDelay = 1f;
public float respawnDelay = 5f;

private Rigidbody2D rb2d;
private PolygonCollider2D pc2d;
private Vector3 start;

    // Start is called before the first frame update
    void Start()
    {
     rb2d = GetComponent<Rigidbody2D>();
     pc2d = GetComponent<PolygonCollider2D>();
     start = transform.position;
    }

    // Update is called once per frame
    void Update()
    {

    }
void OnCollisionEnter2D(Collision2D col){
if (col.gameObject.CompareTag("Player")){
Invoke("Fall",fallDelay);
Invoke("Respawn",fallDelay + respawnDelay);
}
}
void Fall(){
rb2d.isKinematic = false;
pc2d.isTrigger = true;

}

void Respawn(){
transform.position = start;
rb2d.isKinematic = true;
rb2d.velocity = Vector3.zero;
pc2d.isTrigger = false;
}
}
```

This is the script for the moving platform. First of all I had to define variables for speed of the platform, the target it has to reach and the start and end of the movement. In the Start function I assigned start position to the position of the platform and the end position to the one in which is the target. In FixedUpdate I have applied movement to the platform and then I have set the change of the target position to the start position when the platform reaches the target position, so that is always moving from one side to the other.

In the script of the Falling Platform, I defined variables for the delay it takes the platform to fall after the player is in it and the time it takes to respawn. I have also defined some objects for the Rigid Body, Polygon Collider and for the start position of the platform and I initialized them in Start function. The OnCollisionEnter2D function activates the countdown for the falling of the platform when the player touches it. Fall function as its name says is for making the falling movement and Respawn for the reappearance of the platform after the respawn Delay.

Setting the music of the level was easy, I just downloaded a MP3 song from Youtube and then I created an Object with an Audio Source, which reproduces the file you put on it, so I put the MP3 song into this component.



So this was all the procedure for the game, I tried to make it similar to Nintendo Game Yoshi´s island but in a much simpler version.