UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

BACHELOR’S THESIS

Portal_Tiles

proposed by

Tudor-Andrei Frățeanu

Session: july, 2024

Scientific Coordinator

Lect. dr. Moruz Alex

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Portal_Tiles

Tudor-Andrei Frățeanu

Session: july, 2024

Scientific Coordinator

Lect. dr. Moruz Alex

Avizat,

Îndrumător lucrare de licență,

Lect. dr. Moruz Alex.

Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Frățeanu Tudor-Andrei** domiciliat în **România, jud. Suceava, mun. Suceava, str. prof. Leca Morariu, nr. 20, bl. A, et. 1, ap. 2**, născut la data de **04 Februarie 2001**, identificat prin CNP **5010204330255**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2024, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Portal_Tiles** elaborată sub îndrumarea domnului **Lect. dr. Moruz Alex**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Portal_Tiles**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Tudor-Andrei Frățeanu**

Data: ..... Semnătura: .....

Contents

Motivation **2**

Introduction **3**

1 Game description **5**

1.1 Game Engine 6

1.2 Assets 6

2 Implemented mechanics **7**

2.1 Character control 7

2.1.1 Encountered problems and solutions 7

2.2 Portals 8

2.2.1 Encountered problems and solutions 8

2.3 Cube -> Pressure plate -> Door 8

3 Implementation details **9**

3.1 StartMenu 9

3.1.1 Structure in Unity 10

3.1.2 Implementation details 11

3.2 Select Level 13

3.2.1 Structure in Unity 13

3.2.2 Implementation details 14

3.3 Player 16

3.3.1 Structure in Unity 16

3.3.2 Implementation details 16

3.4 TEST_LVL 19

3.4.1 Structure in Unity 20

3.5 Portals 20

3.6 Traps 24
3.7 Cube ->PressurePlate ->Door 26

4 Animations and sound **29**
4.1 Animator 29
4.2 Sounds 31

5 Levels **32**
5.1 Level 1 32
5.2 Level 2 33
5.3 Level 3 33
5.4 Level 4 34
5.5 Level 5 34

Conclusion **36**

Bibliography **37**

Motivation

I have decided to address this topic because, since I discovered the games from Valve, I realized that I want to pursue a career in Game Design. One of these games was **Portal**, released on October 10th, 2007, which revolutionized the video game industry by introducing innovative mechanics and redefining the way puzzles and narration are designed in a game.

Not only the fact that **Portal** redefined industry standards, it inspired a new generation of game designers to explore and innovate, in order to create new gaming experiences that surprise and delight players from all over the world.

Introduction

Video games have come a very long way from their basic beginnings in the '70s and '80s, becoming an integral part of global culture. Over the decades, video games have become much more than a form of entertainment. They have influenced various aspects of society, such as art, education, communication, and mental health.

Regarding video games are still incredibly popular. A recent assessment of the Game Design field showed that this industry is worth more than the film, television, and music industries combined. Video games help people escape problems in life and immerse themselves in fictional worlds.

In education, video games have demonstrated incredible potential as they can be used as teaching tools that help both young children and teenagers learn various concepts in an interactive way. Video games can help in the development of cognitive skills, such as problem-solving, critical thinking, and hand-eye coordination.

Regarding communication and mental health, it is enough to look back a few years ago, during the pandemic, when we had to isolate ourselves at home. Video games played a very important role for many young people, who were able to continue interacting with classmates and friends in a more meaningful way than through messaging or video calls.

Puzzle-Platformer is one of the most popular subgenres in the video game industry. These games consist of **platforming**, which involves jumping and avoiding obstacles, and puzzles, which are solved with logical thinking and applying strategies. The first popular series like **Super Mario Bros**, **Donkey Kong**, and **Sonic** set the stage for more technologically advanced games like **Portal**. For this game, the innovative mechanic of "portals" was developed, which brought a new level of complexity to both **platforming** and **puzzle-solving**.

The purpose of **Portal_Tiles** is to adapt the mechanics found in **Portal**, a **3D Puzzle-Platformer**, **First-Person Shooter (FPS)** game, into a **2D Puzzle-Platformer**

one. This adaptation could offer a different experience from the norm provided by games of this type.

Chapter 1

Game description

Portal_Tiles is a **Puzzle-Platformer 2D** game made with **Unity**. Upon opening the game, the player is greeted by a start screen where they can make the following selections:

- **START** - player can choose to start the game. It will start with the first level.
- **SELECTLEVEL** - player can choose which level to play (the selected level must be unlocked by completing previous level).
- **QUIT** - player can close the game.
- **RESET GAME** - player can reset the progress made in the game.

Additionally, on the right side of the screen, a list of instructions is provided for the player.

For amusement purposes, the player can control their character, which is located on the '_' character in the title.

The player has access to 5 game levels plus one where they can test the implemented mechanics. The first 2 levels focus on the use of "portals" in the platforming

part, while the next 3 introduce **puzzle** elements that can be solved using the previously mentioned mechanic as well as using a system of cube -> pressure plate -> door, commonly found in **Portal**. The player's goal is to navigate the level from the starting position to the finish, avoiding obstacles and solving puzzles.

1.1 Game Engine

Unity is a game engine known for its versatility and ease of use. It is used to create games and interactive applications on both 2D and 3D platforms. As a programming language, **Unity** uses **C#** for scripts that handle logic, interactions, and behaviors.

1.2 Assets

Assets (resources) are the components that make up the visual, audio, and functional content of a project. They can be downloaded from the **Unity Store**.

For the visual part, textures, animations, and fonts, I used the following assets packages:

- Free Pixel Space Platform Pack
- karsiori Pixel Art Padlock Pack - Animated
- Masalimov Ilnur
- Pixel Adventure 1
- Warped Shooting Fx
- Thaleah_PixelFont

Regarding the audio part, backround music and other effects, I've used:

- CasualGameSounds
- Sc-Fi Music
- Warped Shooting Fx

Chapter 2

Implemented mechanics

2.1 Character control

Horizontal movement is achieved by modifying the property **velocity.x** of the object **Player's** rigidbody in each frame. For **Rigidbody2D**, the **velocity** property is of type **Vector2** (two-dimensional vector), so, to update the character's speed on the **O**x axis, the following formula is used:

$$dirX * v\ where, \tag{2.1}$$

- $dirX \in \{-1, 1\}$ (-1 represents left direction - key 'A'; and 1 represents right direction - key 'D').
- v represents the speed which we want the player to move.

On the **Oy** axis, we need to maintain the object's current speed to ensure that movements on the **Ox** axis do not interfere with other mechanisms such as gravity or jumping.

In a similar way, for vertical movement, a force is applied on the **Oy** axis, while the current speed is maintained on the **Ox** axis. This allows the player to jump when pressing the 'SPACE' key.

2.1.1 Encountered problems and solutions

The first problem encountered was the case where the player could jump multiple times in the air. The solution was to allow the character to jump only if it is on a surface from which it can jump.

The second problem was that the player could ”stick” to walls when in the air. The solution was to add a material to the **Rigidbody 2D** component with the friction coefficient property set to 0.

2.2 Portals

To place a portal, the player has a weapon that can shoot either a blue portal (left click) or an orange portal (right click). After the portals are placed, the player can traverse from the orange portal to the blue portal and vice versa.

2.2.1 Encountered problems and solutions

The first problem encountered was the case where the player might try to place a portal in a corner or abuse this mechanic to skip parts of the level. The solution was to ensure that the player can place portals only on certain **surfaces**, thus ensuring that the player cannot cheat and cannot introduce a situation that would disrupt the normal flow of the game.

The second problem was the case where, for example, the player might try to place two blue portals and one orange portal. In this case, traversing from the orange portal to the blue portals would cause unpredictable game behavior. The solution was to ensure that at any point in the game, there cannot be two portals of the same type; each time the player tries to place a second portal of the same color, the first portal is deleted.

The third problem was the teleportation itself. Because entering a portal is done at the collision detection level, when the player was teleported from one portal to another, they remained stuck in a state of teleportation between portals. The solution was to use a cooldown between teleportations and to increase the distance between the portal and the location where the player is teleported.

2.3 Cube -> Pressure plate -> Door

To solve some sections, the player needs to use this mechanic. They can pick up a **cube**, carry it with them through portals, and place it on a **pressure plate** to open a **door** that will allow them to progress in the level.

Chapter 3

Implementation details

3.1 StartMenu

StartMenu is the first window with which the player can interact. At the top is the game title, where the character controlled by the player is also located. In the middle, the player has the following buttons available:

- **START** - player can choose to start the game. It will start with the first level.
- **SELECTLEVEL** - player can choose which level to play (the selected level must be unlocked by completing previous level).
- **QUIT** - player can close the game.

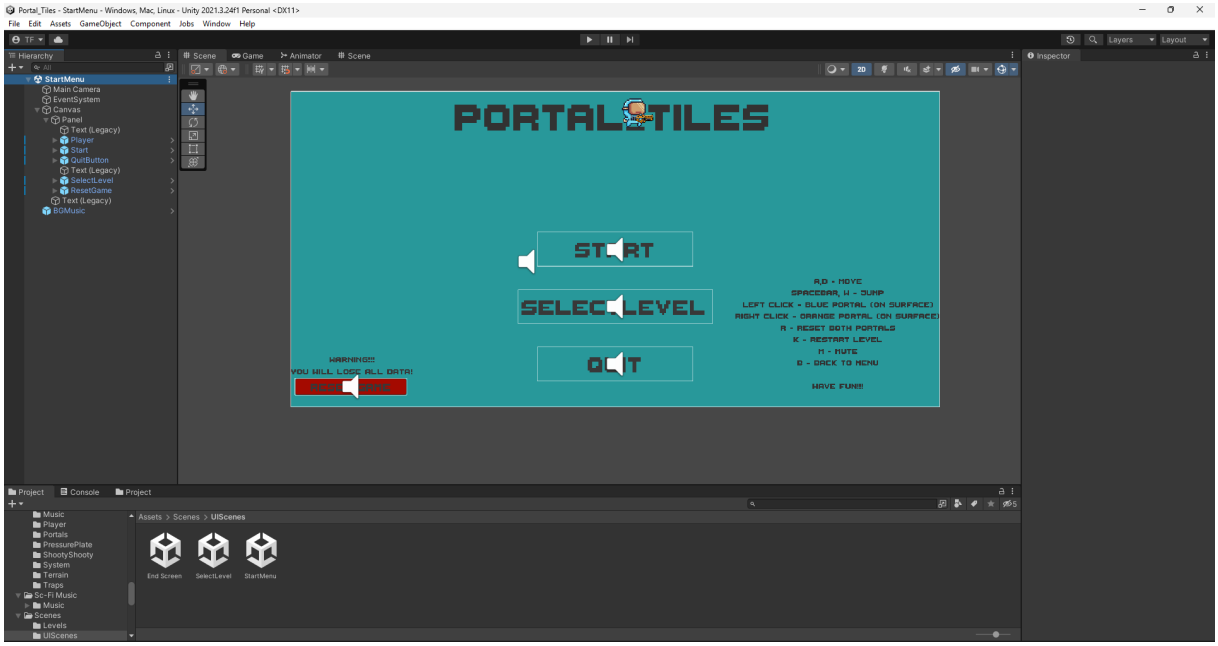
In the right side an instruction set is displayed:

- **A,D - MOVE**

- **SPACEBAR - JUMP**
- **LEFT CLICK - BLUE PORTAL (ON SURFACE)**
- **RIGHT CLICK - ORANGE PORTAL (ON SURFACE)**
- **R - RESET BOTH PORTALS**
- **M - MUTE**
- **B - BACK TO MENU**

And in the bottom left side is a button, **RESET GAME**, which can reset the progress made in the game.

3.1.1 Structure in Unity



The scene for the StartMenu screen is located in "Assets/Scenes/UIScenes". It consists of a camera pointed towards a **Canvas** where the **UI** elements are found. Attached to this **Canvas** is a **Panel** with a set background color, on which elements such as **Text**, **Buttons**, and the **Player** object are placed. Additionally, in the **Canvas** we have the **BGMusic** object, which serves as the audio source for the background music.

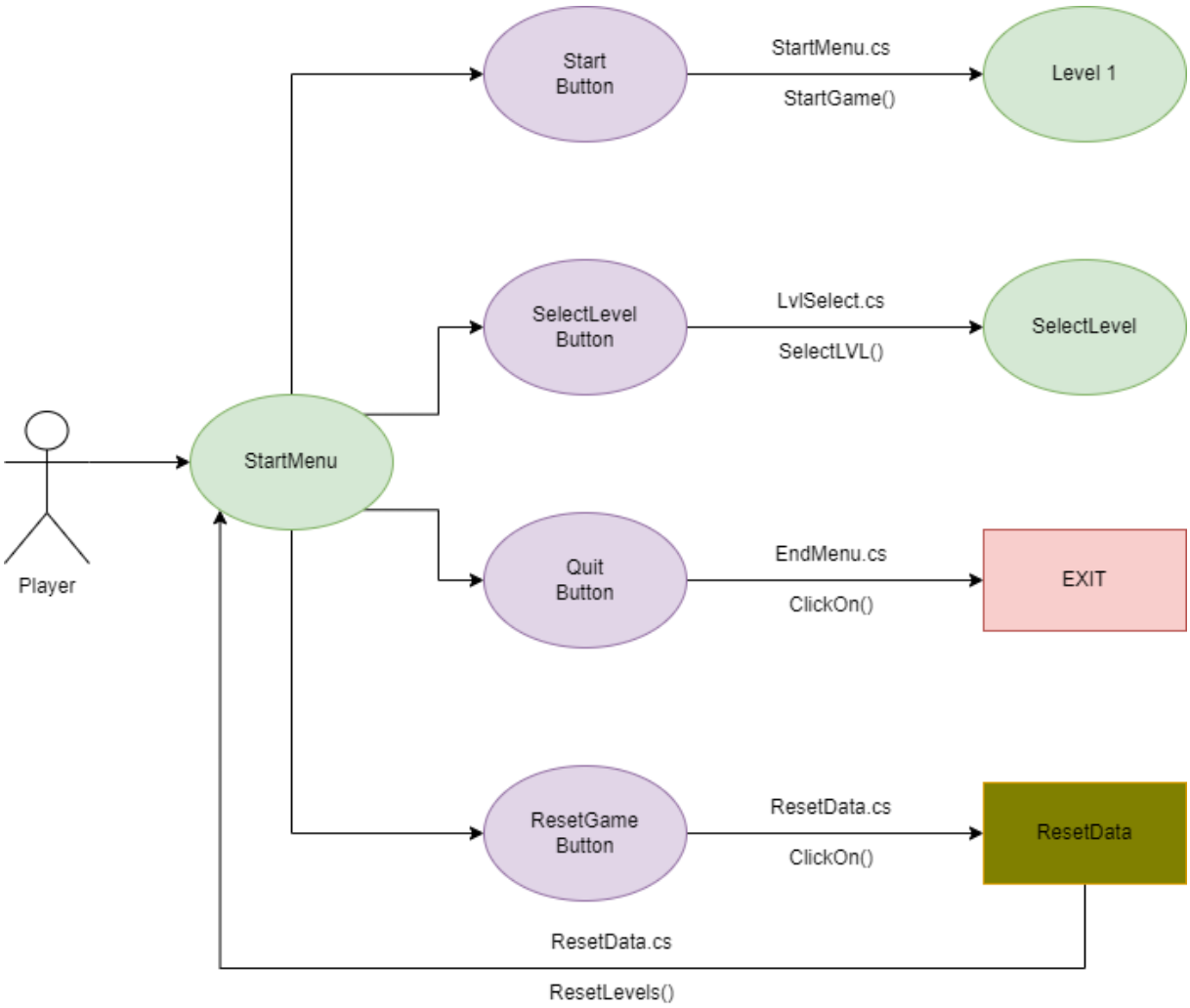
3.1.2 Implementation details

- **START** - When the button is clicked, the method **StartGame()** from the script **StartMenu.cs** is called. This method invokes **Game()** after a delay. The method **Game** uses **SceneManager.LoadScene()** to load the Scene for level 1.

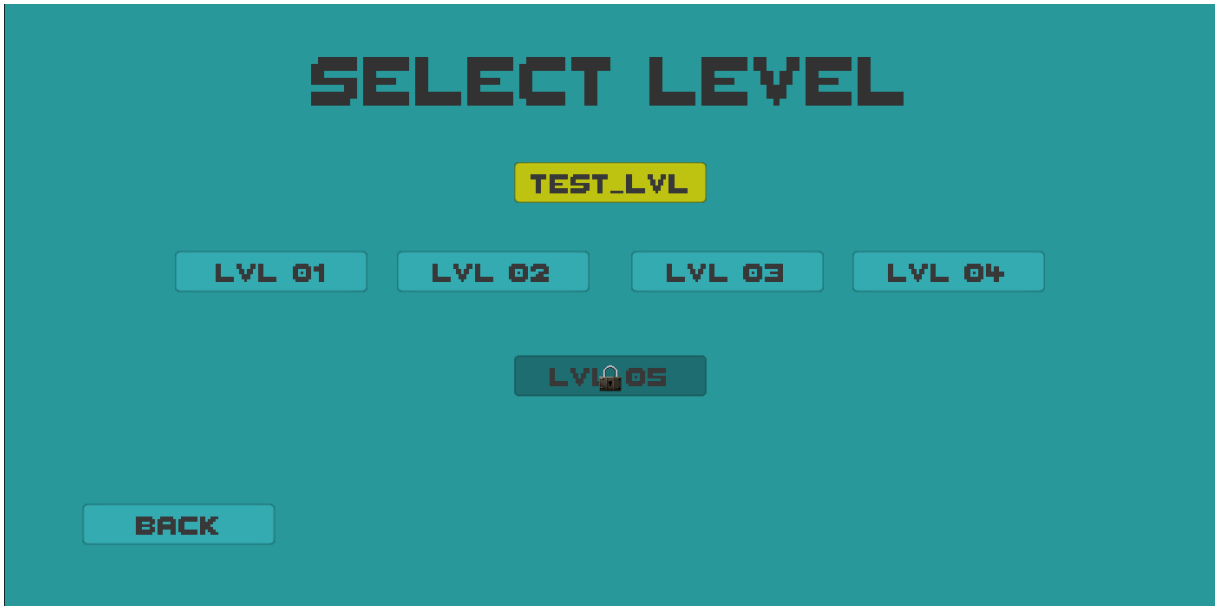
- **SELECTLEVEL** - When the button is clicked, the method **SelectLVL** from the script **LvlSelect.cs** is called. This method calculates the index of the **SelectLevel** Scene in the **lvlNum** variable, from the Build Settings and invokes the Game method.

- **QUIT** - Similar to the **StartMenu.cs** script, the **ClickOn()** method from the **EndMenu.cs** script is called. This invokes the **Quit()** method which will make the application exit.

- **RESETGAME** - When this button is clicked, the **ClickOn()** method from the **ResetData.cs** script is called. This method invokes the **ResetLevels()** method after a delay, which resets the saved data in **PlayerPrefs** (completed levels, unlocked levels) and reloads the current scene so that these changes take effect.

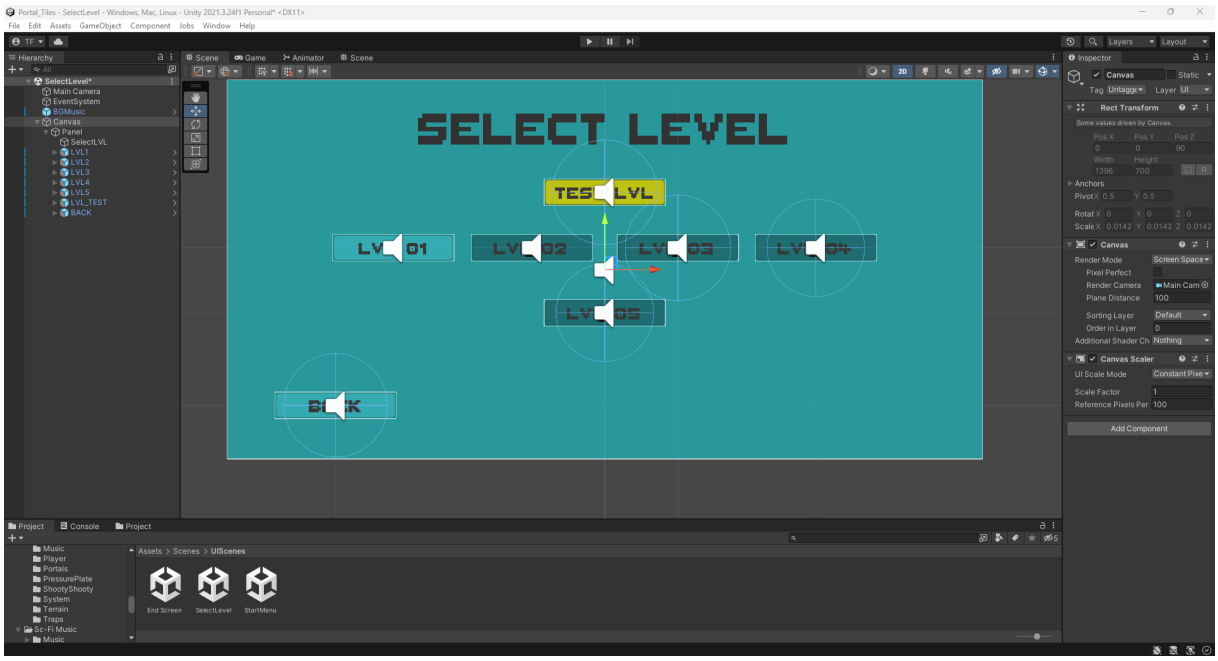

Start
Button
StartMenu.cs
StartGame()
Level 1
SelectLevel
Button
LvlSelect.cs
SelectLVL()
SelectLevel
StartMenu
Player
Quit
Button
EndMenu.cs
ClickOn()
EXIT
ResetGame
Button
ResetData.cs
ClickOn()
ResetData
ResetData.cs
ResetLevels()

3.2 Select Level



In this scene, the player can choose to play one of the five levels (as long as they have been unlocked), try the **TEST_LVL** if they want to test certain mechanics, or return to the start screen by pressing **BACK**. If a level has not been unlocked, it will appear like the **LVL 05** button; with a different color and a lock over it.

3.2.1 Structure in Unity



This **Scene** is located in "Assets/Scenes/UIScenes". In terms of structure, it does not differ much from the **StartMenu** scene.

3.2.2 Implementation details

For **TEST_LVL**, **BACK**, **LVL 01**, no different functionality is implemented compared to the buttons described above. However, for levels 2-5, a method has been implemented to remember which levels have been unlocked between game sessions.

For example, for the **LVL 02** button, we first need to check if the corresponding level has been unlocked and update its appearance. To achieve this, a **Lock** object containing the lock sprite and the **UpdateCompletion.cs** script was attached to this button:

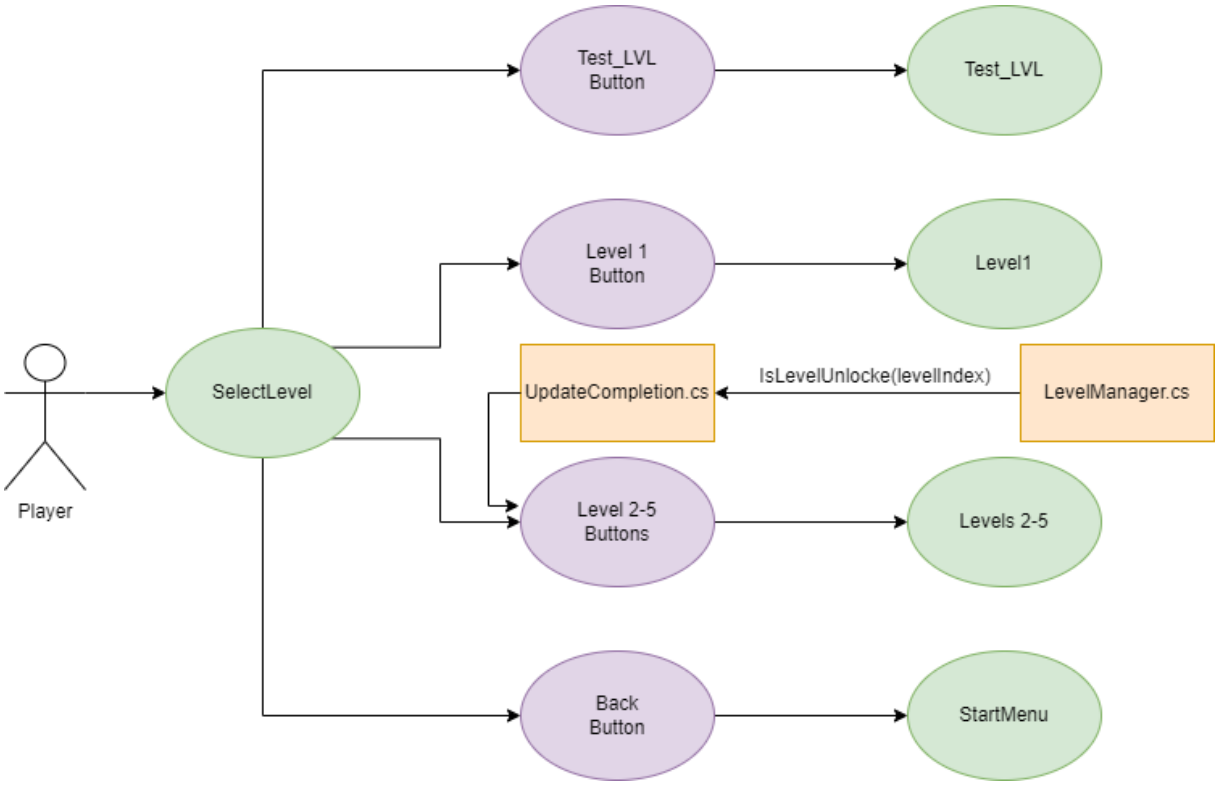
```
public class UpdateCompletion : MonoBehaviour
{
    public Color LockUnlockColor;
    public Color ButtonUnlockColor;
    public GameObject GameObject;
    private SpriteRenderer spriteRenderer;
    public Image image;
    public int index;
    Unity Message | 0 references
    void Start()
    {
        // Update the sprites of the buttons if the corresponfing levels are unlocked
        spriteRenderer = GetComponent<SpriteRenderer>();
        if (LevelManager.IsLevelUnlocked(index))
        {
            spriteRenderer.color = LockUnlockColor;
            image.color = ButtonUnlockColor;

        }
    }

}
```

This script updates the color of the lock (making it transparent if the level is unlocked) and the color of the button.

The actual ”unlock check” is performed by the **IsLevelUnlocked(int levelIndex)** method from the script **LevelManager.cs**:

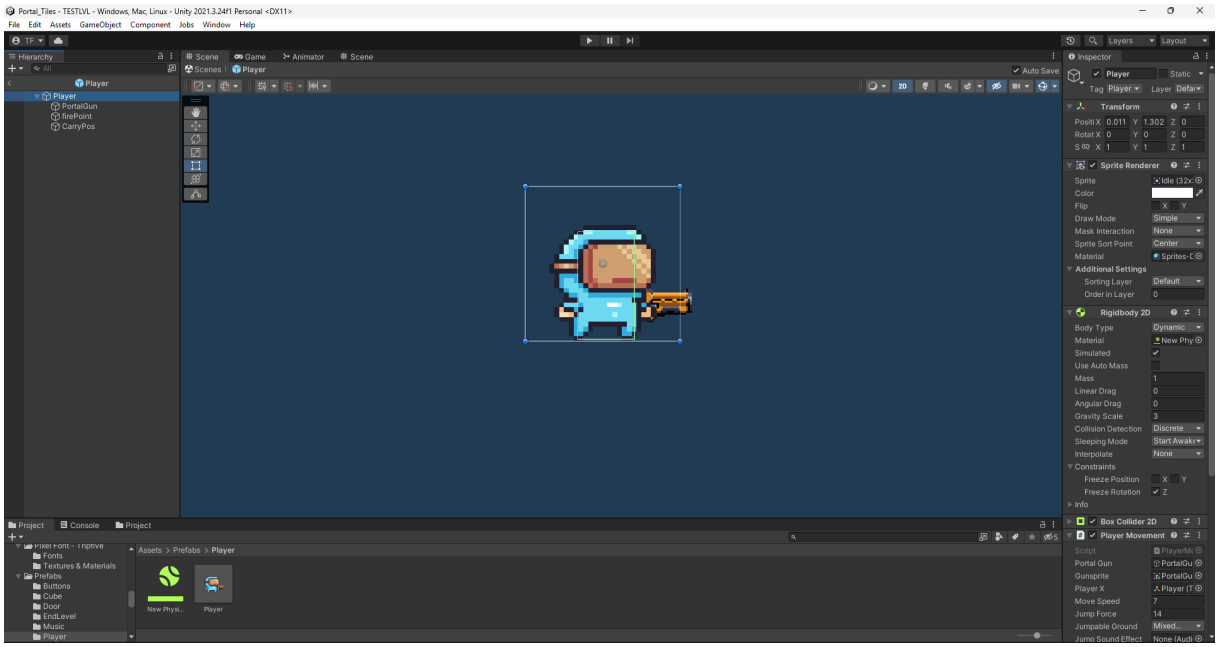
```
// Check if a level is unlocked
4 references
public static bool IsLevelUnlocked(int levelIndex)
{
    string unlockedLevels = PlayerPrefs.GetString(UNLOCKED_LEVELS_KEY, "");
    return unlockedLevels.Contains(levelIndex.ToString());
}
```

This method retrieves the saved data about unlocked levels from PlayerPrefs and checks if the index of the level given as a parameter is found in this data.



3.3 Player

3.3.1 Structure in Unity



This prefab is located in "Assets/Prefabs". The **Player** object was created with the following components:

- **Transform** - Standard component for any Unity object, from here you can set the position, rotation, and scale of the object.
- **Sprite Renderer** - Component used to attach an image to the object.
- **Rigidbody 2D** - Component that provides physical properties to the object
- **Box Collider 2D** - Component related to **Rigidbody 2D** that handles collisions with other objects.
- **Animator** - Component that handles the animations of the object.

The following objects have been attached to this object: **PortalGun**, **firePoint**, **CarryPos**.

3.3.2 Implementation details

Additionally, the Player object has the PlayerMovement.cs script attached, where in the Update() method, called every frame, the logic for character movement is defined:

```
// Update is called once per frame
Unity Message | 0 references
private void Update()
{
    // Horizontal movement logic
    playerX = GetComponent<Transform>();
    dirX = Input.GetAxisRaw("Horizontal");
    mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);



    // Check if the player is grounded so that it cannot jump multiple times while in the air
    if (Input.GetButtonDown("Jump") && IsGrounded())
    {
        jumpSoundEffect.Play();
        rb.velocity = new Vector2(rb.velocity.x, jumpForce);

    }

    // Update the animation state
    UpdateAnimationState();

}
```

he variable dirX receives input from the keyboard with values from the set $\{-1, 1\}$: pressing 'A' assigns -1, and pressing 'D' assigns 1. This variable indicates the direction on the Ox axis and updates the rb.velocity component of the Player object. For jumps, it checks if 'W' or 'SPACEBAR' was pressed and if the Player object is on a surface from which it can jump. Then, it updates the rb.velocity component for the Oy axis.

For the functionality of the PortalGun object, the following scripts were used:

1. **LookAtMouse.cs**

```
1 reference
private void LAMouse()
{
    // Updates the "PortalGun" objects so that it always points at cursor

    Vector2 direction = Camera.main.ScreenToWorldPoint(Input.mousePosition) - m_transform.position;
    float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
    Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.forward);
    m_transform.rotation = rotation;


}
Unity Message | 0 references
void Update()
{
    LAMouse();
}
```

In this script, the distance is calculated using the formula:

$$directon = (mousePosX - objPosX, mousePosY - objPosY) \tag{3.1}$$

where **mousePosX**, **mousePosY** are the coordinates of the cursor in world space, and **objPosX**, **objPosY** are the coordinates of the object's position. Then, the angle in degrees between the **Ox** axis and the direction vector is calculated using the formula:

$$angle = arctan(\frac{directionY}{directionX}) \times \frac{180}{\pi} \tag{3.2}$$

where $\frac{180}{\pi}$ is the conversion factor from radians to degrees.

Finally, to rotate the object around the **Oz** axis based on the angle calculated in radians Θ, a quaternion is calculated using the formula:

$$rotation = (cos(\frac{\Theta}{2}), 0, 0, sin(\frac{\Theta}{2})) \tag{3.3}$$

where, $cos(\frac{\Theta}{2})$ represents the real component of the quaternion, and $sin(\frac{\Theta}{2})$ represents the imaginary component. The other elements remain 0 because rotation is only around the **Oz** axis.

2. **shootBL.cs; ShootOR.cs**

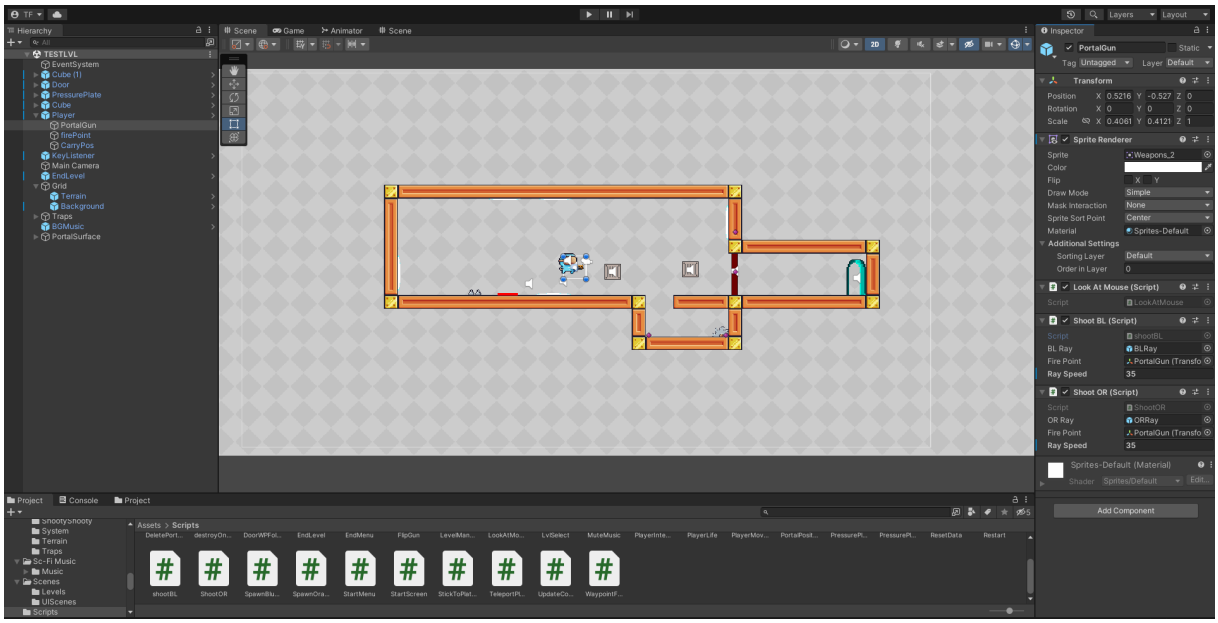
   Both scripts function in the same way to launch 2 projectiles named BLRay and RRay.

```
void Update()
{

    // Aim logic -> Aim "FirePoint" based on mouse position
    lookDirection = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    lookDirection = new Vector2(lookDirection.x - transform.position.x, lookDirection.y - transform.position.y);
    lookAngle = Mathf.Atan2(lookDirection.y, lookDirection.x) * Mathf.Rad2Deg;

    firePoint.rotation = Quaternion.Euler(0, 0, lookAngle);

    if(Input.GetMouseButtonDown(0))
    {
      //Shoot logic

        GameObject rayClone = Instantiate(BLRay);
        rayClone.transform.position = firePoint.position;
        rayClone.transform.rotation = Quaternion.Euler(0, 0, lookAngle);
        new WaitForSeconds(shootCooldown);
        rayClone.GetComponent<Rigidbody2D>().velocity =  firePoint.right * raySpeed;


    }

}
```

The logic behind aiming is similar to that in LAMouse().

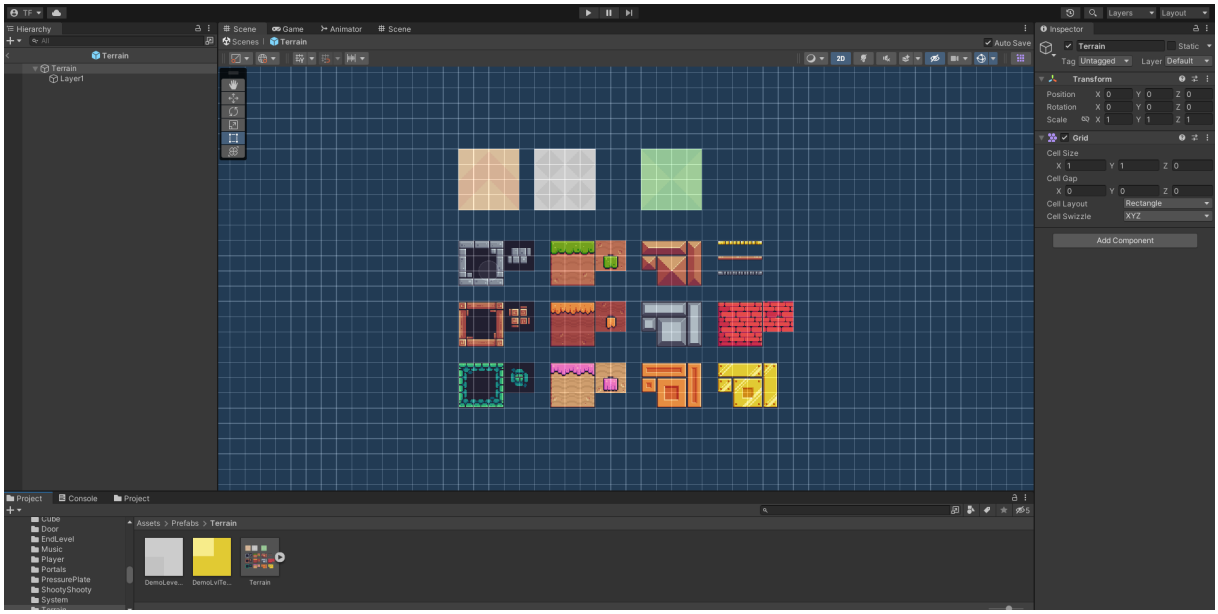
3.4 TEST_LVL

This level was used during the development of the game to test various aspects and implemented mechanics. I decided to include this level in the final version to allow players to familiarize themselves with and test these features as well.



3.4.1 Structure in Unity

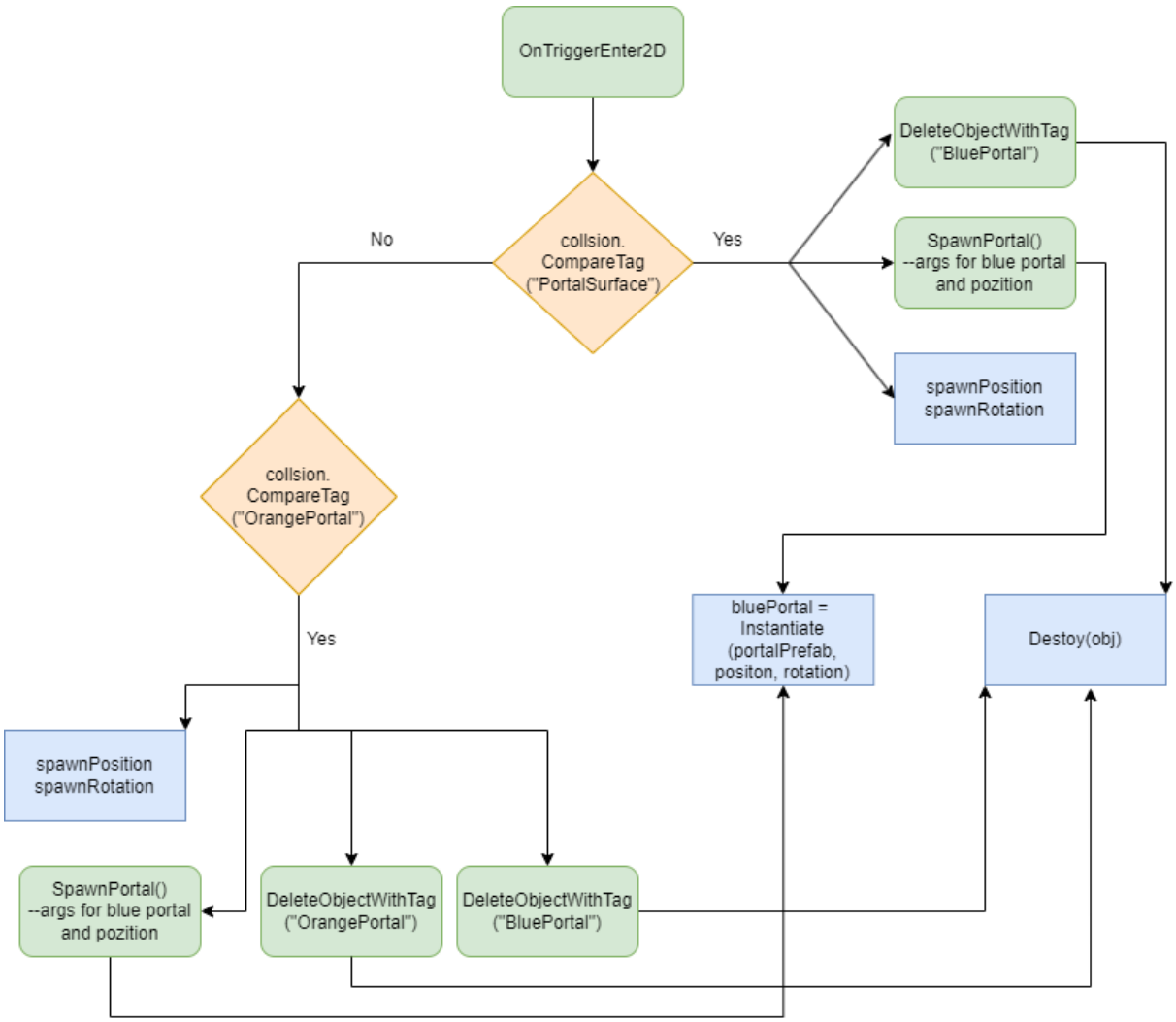
This scene is located in "Assets/Scenes/Levels". The background and the ground are objects of type **Tilemap**. To draw them, I used the **Tile Palette** tool of the editor.



The background and the ground are placed on two separate layers. To make the ground a solid object that the character can stand on, I attached a **Rigidbody** and a **Tilemap Collider 2D** to it. To provide the ground with a collider that appears only at the contour level and not at the grid cell level, I attached a **Composite Collider 2D**. This ensures smoother gameplay performance.

3.5 Portals

If a collision is detected between an object with tag **BLRay** or **ORRay** with an object with tag **PortalSurface**, one of the scripts **SpawnBluePortal.cs** or **SpawnOrangePortal.cs** is activated.



The corresponding portal will be placed on the surface based on its rotation, and the old portal of the same type or the one on the surface will be deleted.

```
Unity Script | 0 references
public class destroyOnCollsion : MonoBehaviour
{
    public GameObject Bray;
    public GameObject Oray;
    [SerializeField] private AudioSource spawnPortalSoundEffect;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        Bray = GameObject.FindWithTag("BLRay");
        Oray = GameObject.FindWithTag("ORRay");
    }

    // Update is called once per frame
    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision)
    {

        if (collision.CompareTag("BLRay"))

        {
            spawnPortalSoundEffect.Play();
            Destroy(GameObject.FindGameObjectWithTag("BLRay"));
        }
        if (collision.CompareTag("ORRay"))

        {
            spawnPortalSoundEffect.Play();
            Destroy(GameObject.FindGameObjectWithTag("ORRay"));
        }
    }


}
```

This script is attached to all other surfaces in the game to delete objects **BLRay** and **ORRay** upon collision detection.

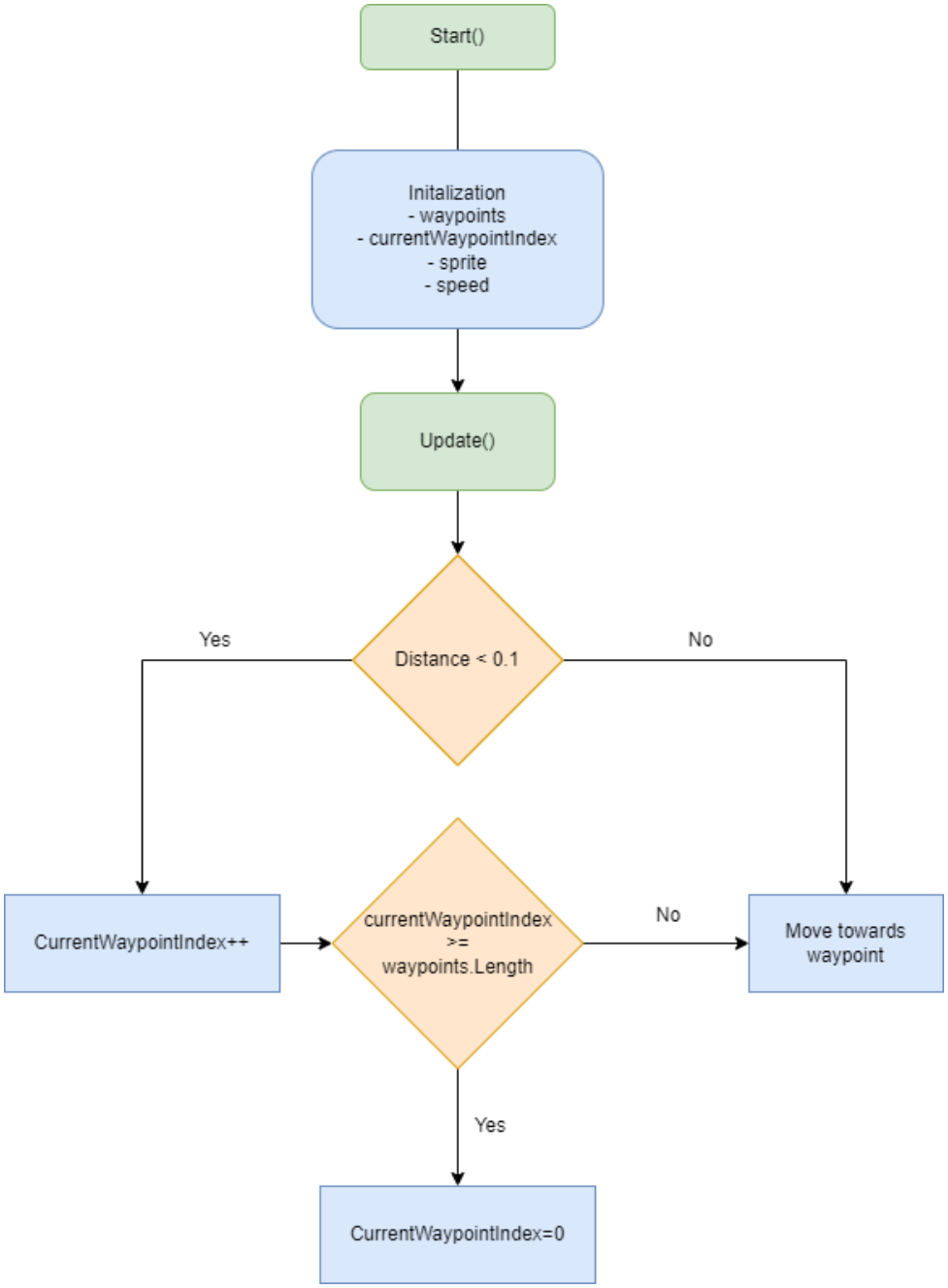
The script **TeleportPlayer.cs** is attached to the **Player** object, and it activates when the player comes into contact with one of the two portals.

```
private IEnumerator TeleportCoroutine(Transform destination)
{
    canTeleport = false;
    teleportSoundEffect.Play();
    // Check if the destination portal is rotated
    float rotationZ = destination.eulerAngles.z;

    // Determine the spawn offset based on the rotation
    Vector3 spawnOffset = Vector3.zero;
    if (rotationZ == -90f)
    {
        spawnOffset = Vector3.down + teleportOffset;
    }
    else if (rotationZ == 90f)
    {

        spawnOffset = Vector3.up + teleportOffset;
    }
    else if (rotationZ == 180f || rotationZ == -180f)
    {
        spawnOffset = Vector3.left + teleportOffset;
    }
    else if (rotationZ == 0f)
    {
        spawnOffset = Vector3.right + teleportOffset;
    }
    else
    {
        spawnOffset = teleportOffset; // If rotation is different, just use the teleportOffset
    }
    // Teleport the player to the destination
    transform.position = destination.position + spawnOffset;

    // Cooldown to avoid having the player looping between portals
    yield return new WaitForSeconds(teleportCooldown);

    canTeleport = true;
}
```

This script modifies the position of the player based on the position of the portal to which they will teleport, taking into account its orientation on the game world.

3.6 Traps

In platforming games, trap elements have always been introduced to make the player's progress through a level more challenging. In this game, I decided to introduce two types of traps: **Spikes** and **Saw**. **Spikes** are static objects placed on the floor of the level, while **Saw** is a moving object that travels between two empty game objects named **Waypoint1** and **Waypoint2**. To implement this movement, I used the script **WaypointFollower.cs**. This script iterates through an array that stores the two waypoints and moves the **Saw** object alternately towards each of them.



Additionally, this script is used by objects called **MovingPlatform**, also used as elements in platforming.

To determine if the character has touched a trap and ”died,” the script **PlayerLife.cs** has been attached to the **Player** object. If a collision is detected with an object tagged as **trap**, the portals are disabled and the level is restarted.

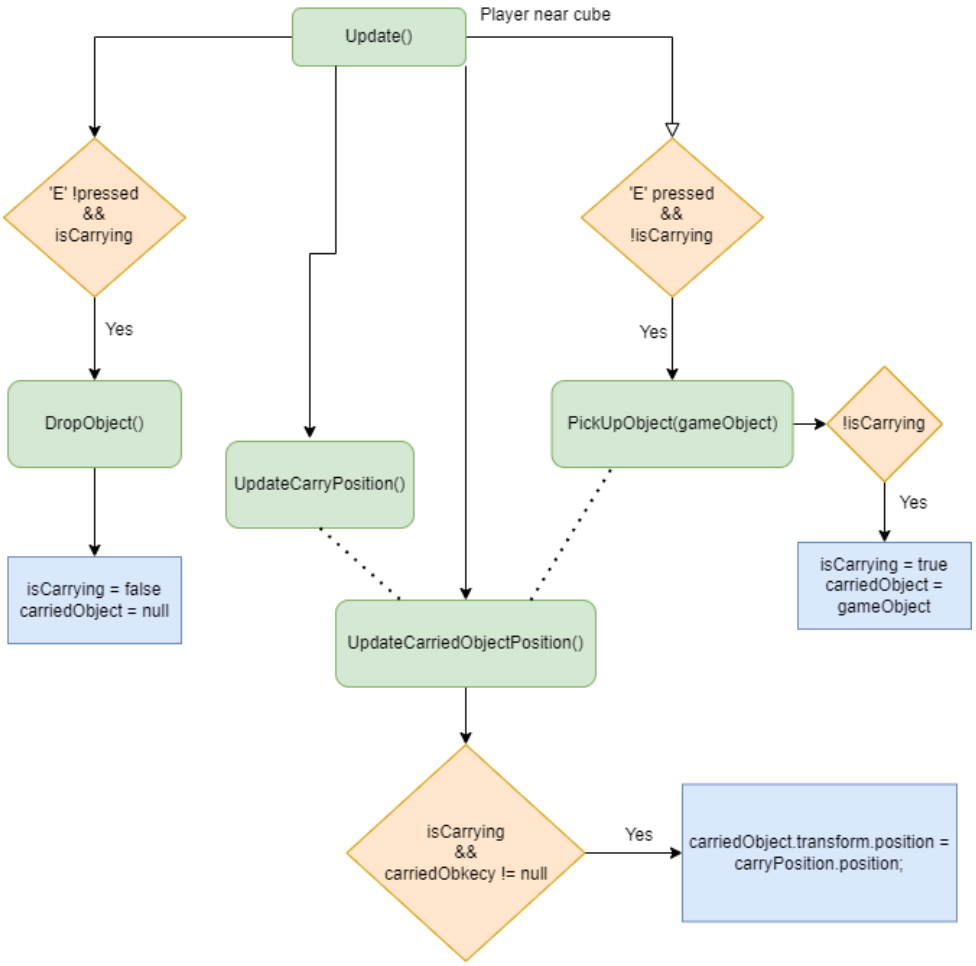
```
public class PlayerLife : MonoBehaviour
{
    private Animator anim;
    private Rigidbody2D rb;
    [SerializeField] private AudioSource dieSoundEffect;
    [SerializeField] private AudioSource spawnSoundEffect;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
    }

    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if(collision.gameObject.CompareTag("Trap"))
        {
            Die();
        }

    }
    1 reference
    private void Die() // Die logic
    {
        rb.bodyType = RigidbodyType2D.Static;
        dieSoundEffect.Play();
        anim.SetTrigger("death"); // Set trigger for Animator
        DeleteObjectsWithTag("BluePortal");
        DeleteObjectsWithTag("OrangePortal");
    }
    2 references
    private void DeleteObjectsWithTag(string tag)
    {
        GameObject[] objectsWithTag = GameObject.FindGameObjectsWithTag(tag);
        foreach (GameObject obj in objectsWithTag)
        {
            Destroy(obj);
        }
    }
    0 references
    private void RestartLevel() // Called in animation event Player_Death
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        spawnSoundEffect.Play();
    }
}
```

3.7 Cube ->PressurePlate ->Door

This mechanic is divided into three parts:

1. Player interaction with the cube.

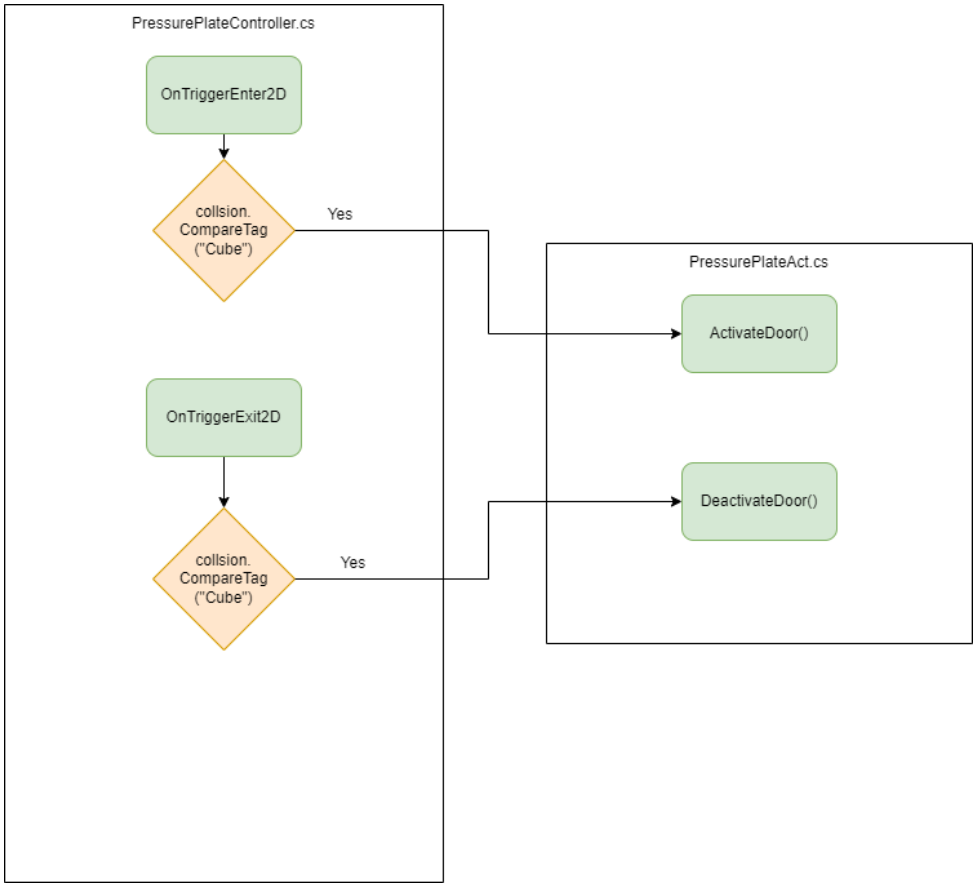
   The cubes are on their own separate layer, ensuring that the player can only interact with them. The **Player** object has the **PlayerInteraction.cs** script attached to handle this interaction. This script checks if the Player is near a cube in order to pick it up. It also verifies if the player is already carrying another cube. Once picked up, as the player moves, the script updates the cube's position every frame.



   Additionally, the script **TeleportPlayer.cs** has been added to this object to allow interaction with portals.

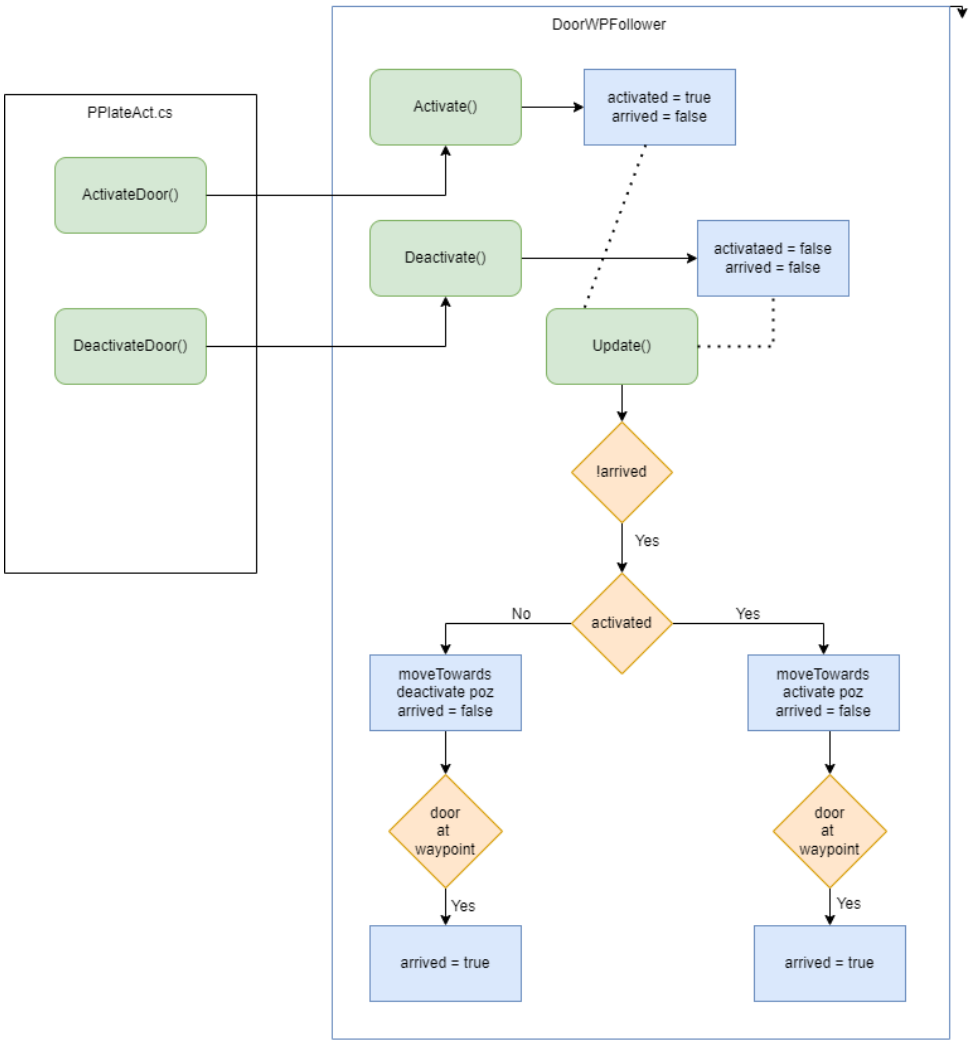
2. Cube interaction with the pressure plate

   When the cube is placed on a pressure plate, this collision is detected, and the script **PressurePlateController.cs** is called. This script checks for collision events, updates the sprite for the pressure plate, and calls the methods **ActivateDoor()** and **DeactivateDoor()** from the script **PressurePlateAct.cs** accordingly.



3. Pressure plate interaction with the door.

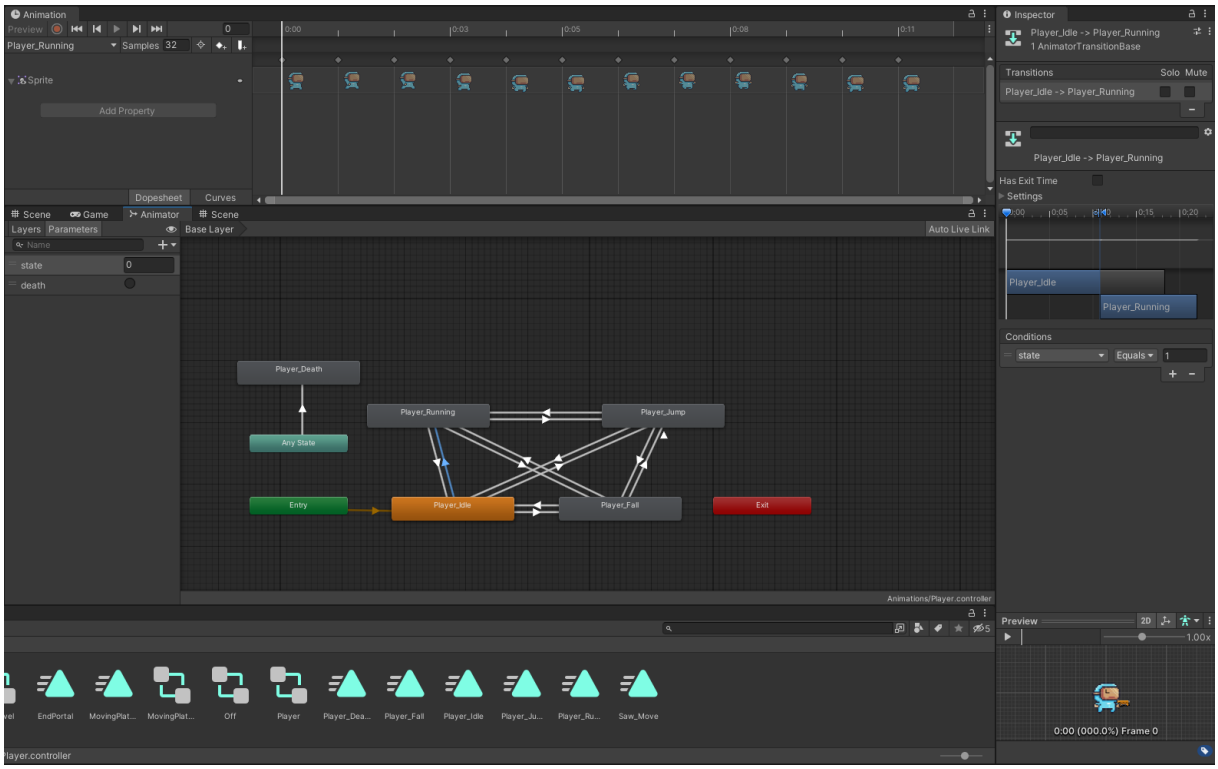
   The script **PressurePlateAct.cs** has been attached to the **Door** object. This script sets the sprite of the door based on which method is called and invokes one of the methods **Activate()** or **Deactivate()** from the script **DoorWPFollower**. This script functions similarly to the script **WaypointFollower.cs**, but this time the door moves only when it has been activated or deactivated instead of moving in a loop.



Chapter 4

Animations and sound

4.1 Animator



To animate the **Player** object, I added the **Animator** component, which is connected to an **Animation Controller**.

In the image above, the **Animation Controller** is a state machine that controls how and when transitions occur between animations. Each state in this graph represents an animation. States are connected to each other through transitions that can be triggered by parameters or specific conditions (for example, transitioning from **Idle** to **Running** and back).

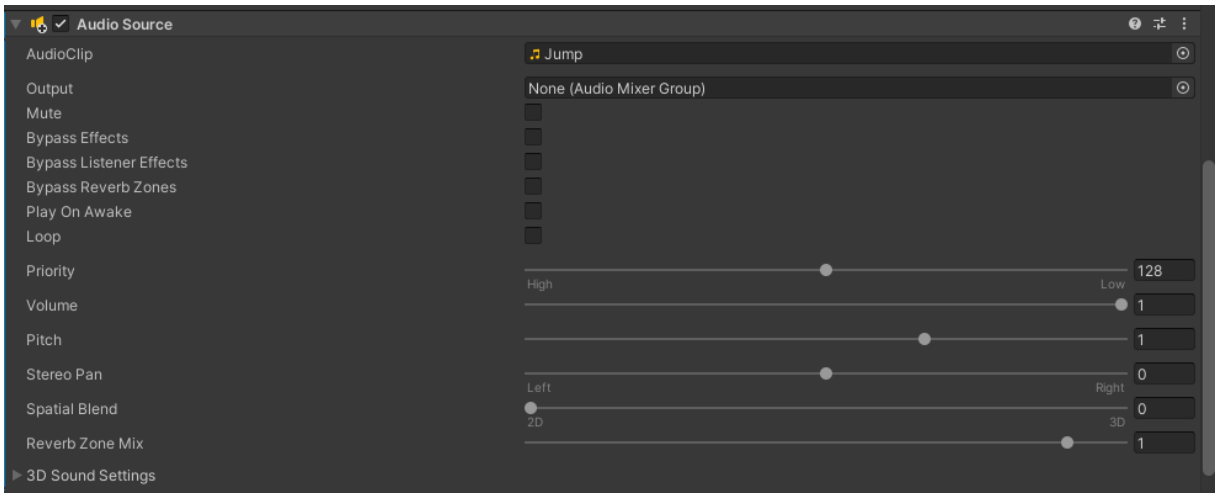
In **PlayerMovement.cs**, we use an **enum** to represent different movement states of the character (idle, running, jumping, falling), and these states are set in the animator using an **Integer** parameter.

```
if (rb.velocity.y > .1f)
{
    state = MovementState.jumping;
    if (dirX > 0f)
    {
        sprite.flipX = false;
    }
    else if (dirX < 0f)
    {
        sprite.flipX = true;
    }
}
else if (rb.velocity.y < -.1f)
{
    state = MovementState.falling;
    if (dirX > 0f)
    {
        sprite.flipX = false;
    }
    else if (dirX < 0f)
    {
        sprite.flipX = true;
    }

}
if (dirX > 0f)
{
    sprite.flipX = false;
}
else if (dirX < 0f)
{
    sprite.flipX = true;
}
anim.SetInteger("state", (int)state );
```

For example, in this part of the script, we determine whether the **Player** is jumping or falling, and update the animation state:

4.2 Sounds

For various objects, I have attached the **Audio Source** component. This component enables the object to play an audio clip. For example, on the **Player**, multiple **Audio Source** components have been attached. One of these handles the sound for when the player jumps.




```
if (Input.GetButtonDown("Jump") && IsGrounded())
{
    jumpSoundEffect.Play();
    rb.velocity = new Vector2(rb.velocity.x, jumpForce);

}
```

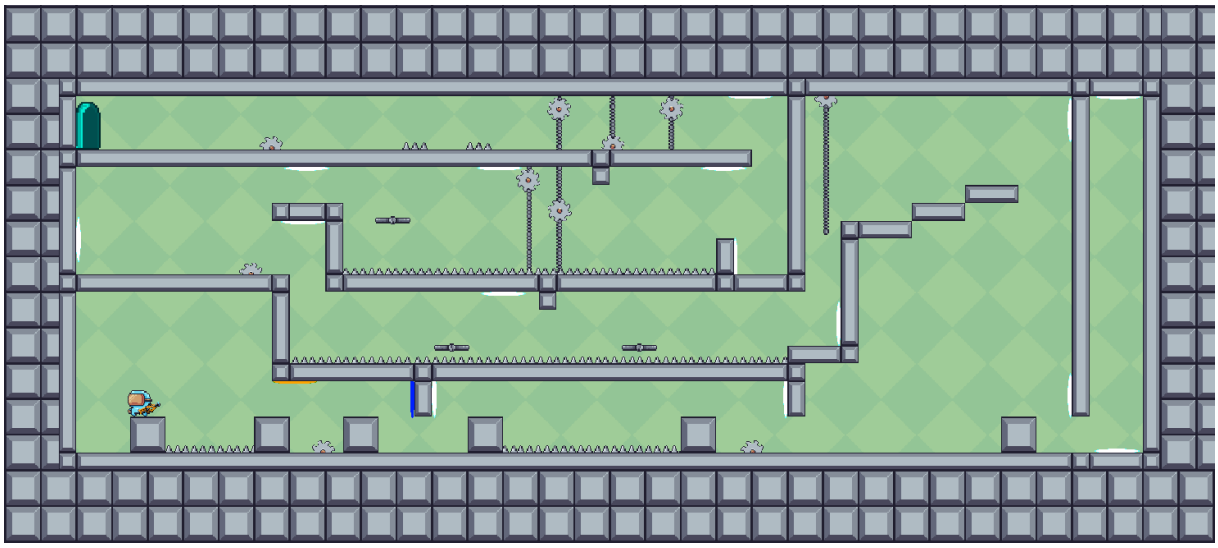
In **PlayerMovement.cs** this sound is being played any time the player jumps.

Chapter 5

Levels

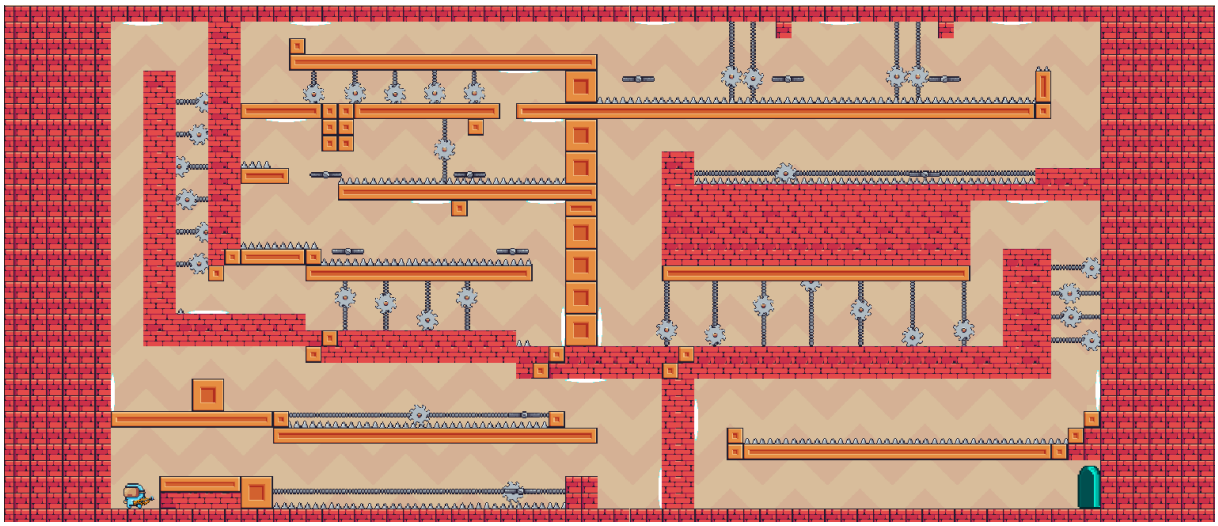
5.1 Level 1

In this level, the player needs to use portals and platforms to navigate through obstacles. When they reach the end of the level, the scene for the next level will be loaded.



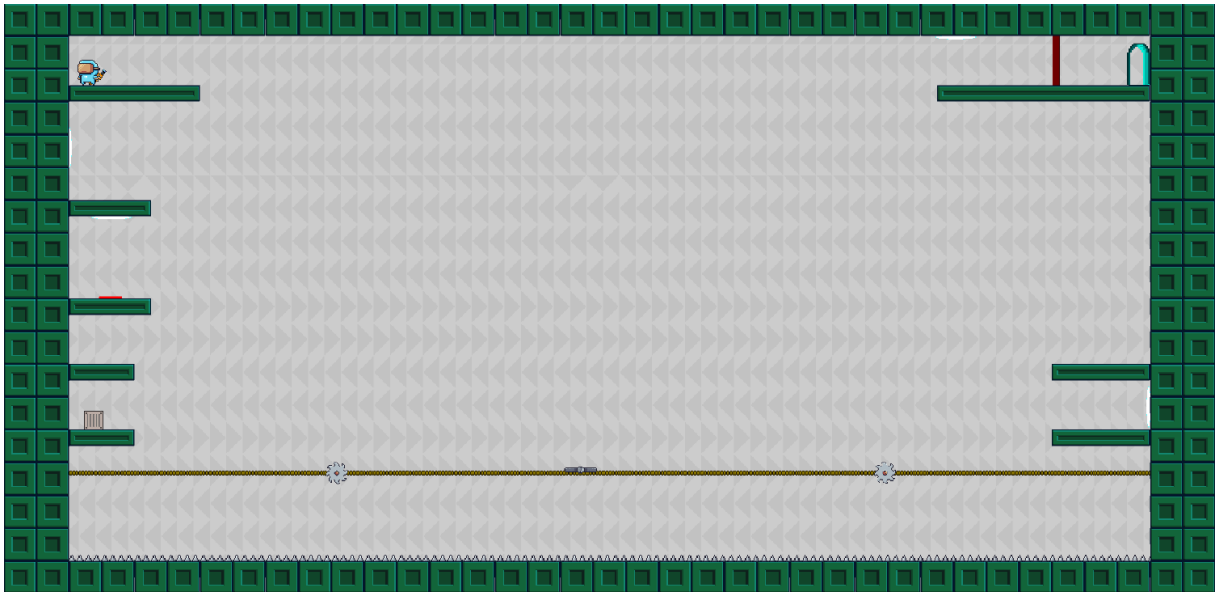
5.2 Level 2

Similar to the first level, this level contains only platforming elements, but the difficulty level is significantly higher.



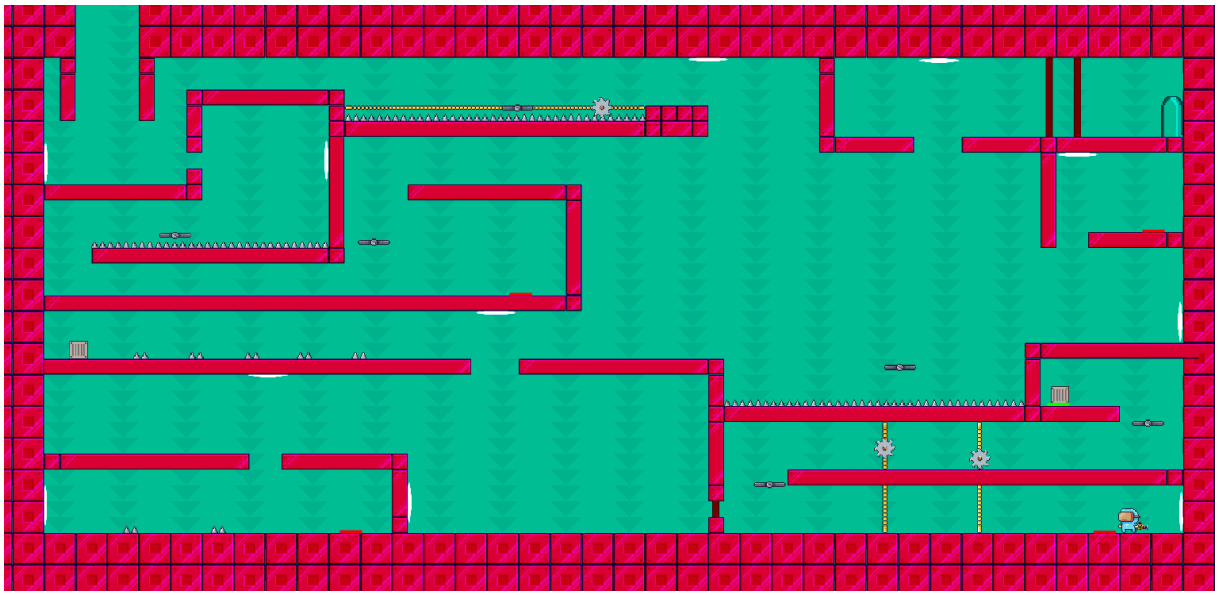
5.3 Level 3

This level is easier in terms of difficulty, but puzzle elements are also introduced.



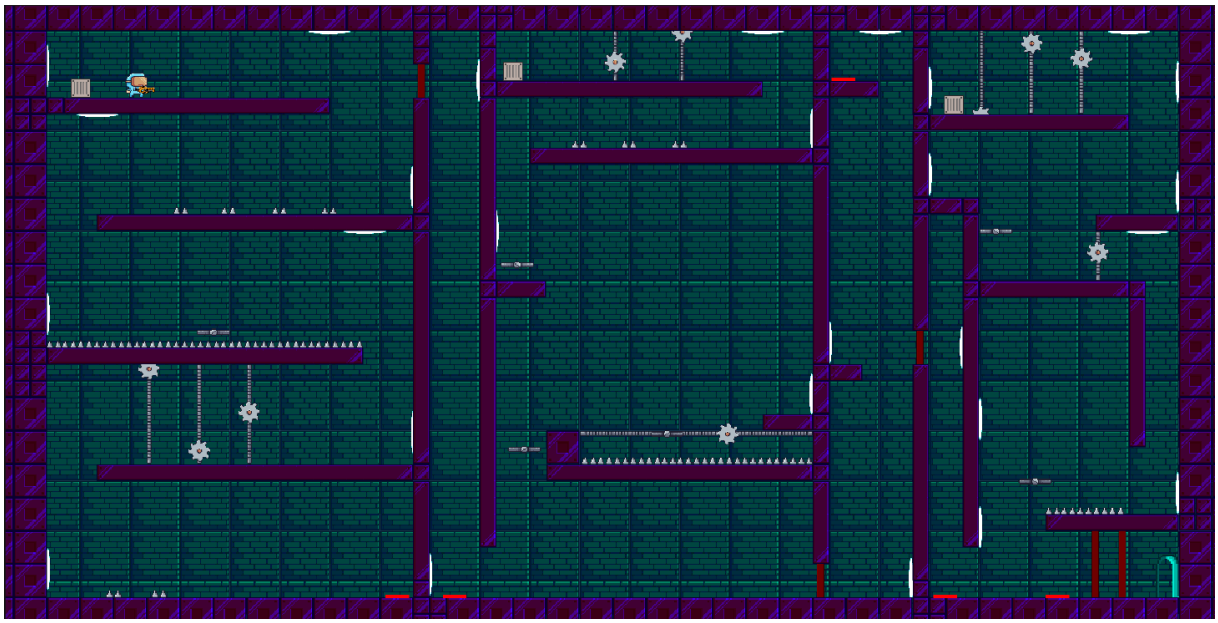
5.4 Level 4

Similar to level 3, this level contains both platforming and puzzle elements, but the difficulty level is higher.



5.5 Level 5

This is the last level of the game. It has a higher difficulty than the previous level.



After completing this level, the scene **End Screen** will be loaded. From here, the player can choose whether to exit the application or return to the main menu (**StartMenu**).

Conclusions

4 This game achieves the goal of implementing mechanics like **Portals** and **Cube -> PressurePlate -> Door** in a user-friendly manner for players, within the context of a **2D Puzzle-Platformer** game. The game is a prototype that demonstrates the feasibility of using these mechanics in a much more complex project, provided that the developer pays attention to the elements of **Level Design**.

This game could be further developed by adding levels that include enemies like **Turrets**, and by introducing a story to enhance player immersion. This story could be implemented through dialogues/monologues and through the environment itself (narrative through the environment).

In the future perspective, this game could be published on an online platform such as **Steam**, to gather feedback from players. Besides feedback, players can form a community and contribute to the future development of the game.

Bibliography

Sources that inspired and helped me during the development of this thesis:

- https://www.youtube.com/watch?v=TcranVQUQ5U&list=PLgOEwFbvGm5o8hayF index=2&ab_channel=Pandemonium
- https://www.youtube.com/watch?v=r5LtNI3Tm1M&ab_channel=MuddyWolf
- https://docs.unity3d.com/Manual/Unity2D.html
- https://assetstore.unity.com/
- https://www.gamepressure.com/newsroom/gaming-market-grows-beyond-ci z03265
- https://docs.unity3d.com/ScriptReference/Animator.html
- https://gametyrant.com/news/gaming-and-society-exploring-the-impact
- https://open.lib.umn.edu/mediaandculture/chapter/10-4-the-impact-of