

**LAPORAN
PEMROGRAMAN VISUAL**



Dosen Pengampu :

Evianita Dewi Fajrianti, S.Tr.T., M.Tr.T., Ph.D

Disusun oleh :

1. Anaya Rizka Putri Novianti (5124521002)
2. Muhammad Irsadul Ibad (5124521007)
3. Muhammad Afif Habiburrahman (5124521010)
4. Mayva Sofiana (5124521018)
5. Devi Hidayah (5124521022)

**TEKNOLOGI MULTIMEDIA BROADCASTING
DEPARTEMEN TEKNOLOGI MULTIMEDIA KREATIF
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA
KAMPUS LAMONGAN
TAHUN 2025**

LAPORAN UNITY PATHWAY
JUNIOR PROGRAMMING AS SOON AS POSSIBLE

GAMBARAN UMUM UNITY PATHWAY JUNIOR PROGRAMER.

Pathway Junior Programming dibagi menjadi beberapa unit / mission. Di halaman pathway, unit yang terlihat adalah :

1. Player Control
2. Basic Gameplay
3. Sound and Effects
4. Gameplay Mechanics
5. User Interface
6. Manage Scene Flow and Data
7. Apply Object-Oriented Principles

Pada laporan ini, pembahasan difokuskan pada unit ketiga, yaitu Sound and Effects. Bagian ini menekankan pentingnya penggunaan audio dan efek visual untuk meningkatkan pengalaman bermain serta memberikan umpan balik yang sesuai terhadap tindakan pemain. Melalui tahap ini, peserta belajar bagaimana menambahkan dan mengatur suara latar, efek suara, serta efek visual yang muncul ketika terjadi interaksi dalam permainan, seperti tabrakan, lompatan, atau pencapaian tujuan tertentu.

Tahap ini juga mencakup pemahaman tentang cara mengontrol volume, menyesuaikan waktu pemutaran suara agar selaras dengan peristiwa di dalam game, serta mengoptimalkan performa agar efek yang ditambahkan tidak mengganggu jalannya permainan. Materi ini menjadi dasar penting sebelum melanjutkan ke tahap pengembangan mekanik gameplay yang lebih kompleks pada misi berikutnya.

DAFTAR ISI

GAMBARAN UMUM UNITY PATHWAY	i
DAFTAR ISI	ii
BAB I PENDAHULUAN	1
1.1 Latar Belakang.....	1
1.2 Tujuan.....	1
1.3 Peralatan	2
BAB II PEMBAHASAN	3
2.1 Basic Gameplay.....	3
2.1.1 Setting Up The Project and Initial View.....	3
2.1.2 Adding and Configuration The Player Character	3
2.1.3 Creating The Player Jump Mechanic.....	3
2.1.4 Addjusting Jump Force And Gravity.....	4
2.1.5 Adding Obstacles and Spawn System	4
2.1.6 Creating a Moving and Looping Background	5
2.1.7 Adding a Game Over System and Object Management	5
2.1.8 Setting Up and Synchronizing Player Animation.....	6
2.1.9 Refining Animation Transition and Logic.....	7
2.1.10 Adding Particle and Sound Effect	7
2.2 Challenge 3 – Ballons, Bombs, & Booleans	8
2.2.1 Understanding and Analizing The Challenge	8
2.2.2 Fixing Ballon Control and Background Movement	9
2.2.3 Fixing Object Spawning and Firework Position.....	9
2.2.4 Smoothing Background Loop and Adding Movement Limits	9
2.2.5 Debugging With Boolean Logic	9
2.3 Lab 3 – Player Control	10
2.3.1 Create Player Controller and Plan Your Code.....	10
2.3.2 Basic Movement From User Input	10
2.3.3 Constrain The Player's Movement	10
BAB III PENUTUP	11
3.1 Kesimpulan.....	11

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam pengembangan sebuah permainan digital, audio dan efek visual memiliki peran yang sangat penting dalam menciptakan suasana serta pengalaman bermain yang menarik dan mendalam. Unsur suara tidak hanya berfungsi sebagai pelengkap, tetapi juga sebagai elemen yang mampu memberikan umpan balik langsung kepada pemain, menegaskan aksi yang dilakukan, dan memperkuat emosi yang ingin disampaikan dalam permainan. Misalnya, suara loncatan, ledakan, atau langkah kaki dapat membantu pemain memahami konteks situasi tanpa harus melihat seluruh elemen visual di layar.

Melalui *mission 3: Sound and Effects* pada Unity Pathway Junior Programmer, peserta diajak untuk memahami secara menyeluruh bagaimana mengintegrasikan suara latar (background music), efek suara (sound effects), dan efek visual (visual effects) yang relevan dengan jalannya permainan. Materi ini tidak hanya berfokus pada aspek teknis penerapan, tetapi juga mengajarkan pentingnya penyesuaian nada, volume, dan waktu pemutaran agar selaras dengan interaksi pemain.

Selain itu, penggunaan efek visual seperti partikel, cahaya, atau animasi singkat juga memperkuat kesan dinamis dan realisme dalam game. Dengan mempelajari tahap ini, peserta memperoleh pemahaman tentang bagaimana audiovisual dapat menciptakan imersi, meningkatkan kepuasan bermain, dan menjadikan permainan lebih hidup serta profesional. Tahap ini juga menjadi dasar penting sebelum melanjutkan ke pengembangan mekanik gameplay yang lebih kompleks di misi-misi berikutnya.

1.2 Tujuan

- 1 Memahami peran audio dan efek visual dalam meningkatkan pengalaman bermain.
- 2 Mempelajari cara menambahkan dan mengatur suara di dalam Unity, termasuk musik latar dan efek suara interaktif.

- 3 Mengimplementasikan efek visual sederhana seperti partikel, cahaya, atau animasi singkat untuk memperkuat suasana permainan.
- 4 Mengatur sinkronisasi antara suara, efek, dan peristiwa dalam game agar respons terasa lebih natural.
- 5 Meningkatkan imersi dan kenyamanan pemain melalui penggunaan kombinasi audio-visual yang tepat dan seimbang.

1.3 Peralatan

- Laptop.
- Unity hub
- Unity Editor (Versi 2022)
- Notepad ++ / Visual Studio (Code Editor)

BAB II

PEMBAHASAN

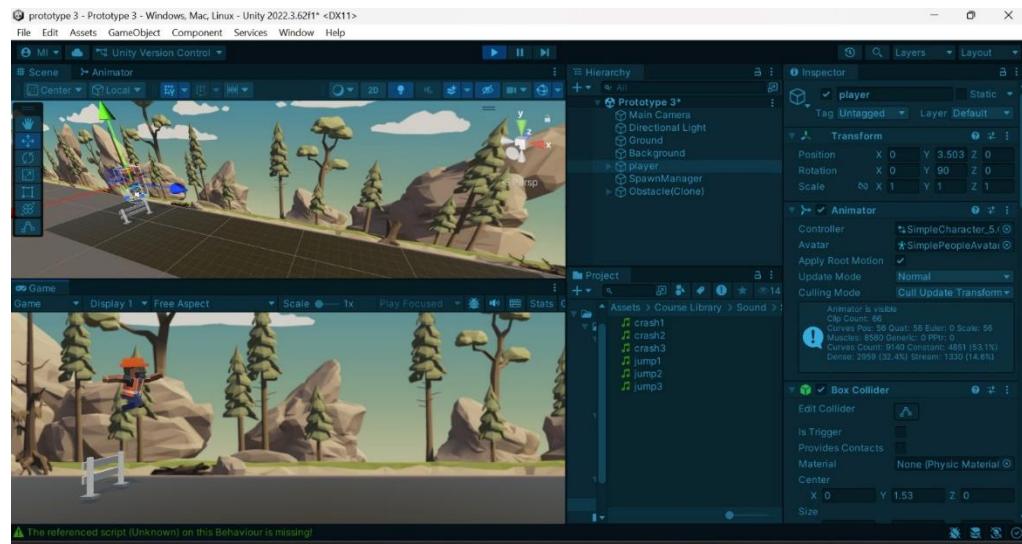
2.1 Sound and Effect

2.1.1 Setting Up the Project and Initial View

Pada langkah pertama, kami membuka proyek prototype yang sudah disiapkan untuk pelajaran ini. Kami mulai dengan mengganti warna background pada kamera utama agar tampilan permainan terlihat lebih menarik dan tidak monoton. Langkah ini juga membantu kami mengenal lebih jauh tentang bagaimana kamera di Unity bekerja sebagai pandangan utama pemain terhadap dunia game.

2.1.2 Adding and Configuring the Player Character

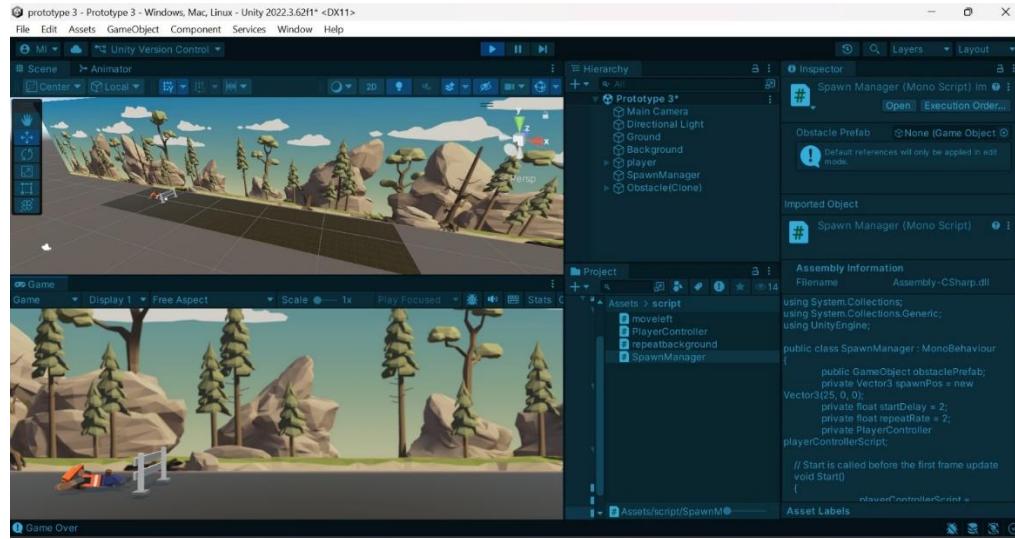
Setelah itu, kami menambahkan karakter utama permainan dengan membuat objek Player di scene. Agar karakter dapat berinteraksi dengan dunia fisik di dalam game, kami menambahkan komponen Rigidbody supaya karakter terpengaruh oleh gravitasi dan gaya dorong, serta Collider agar dapat menyentuh atau bertabrakan dengan permukaan lain seperti lantai atau rintangan.



2.1.3 Creating the Player Jump Mechanic

Kami membuat fitur lompat dengan memberikan gaya dorong ke atas. Awalnya karakter melompat otomatis, lalu kami ubah agar lompatan terjadi hanya saat

tombol spasi ditekan. Dengan begitu, pemain bisa mengatur waktu lompatan sendiri, seperti pada game platformer pada umumnya.



```
Assembly Information
Filename: Assembly-CSharp.dll
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class SpawnManager : MonoBehaviour
{
    public GameObject obstaclePrefab;
    private Vector3 spawnPos = new Vector3(0, 0, 0);
    private float startDelay = 2;
    private float repeatRate = 2;
    private PlayerController playerControllerScript;
    // Start is called before the first frame update
    void Start()
    {
        playerControllerScript =
    }
```

2.1.4 Adjusting Jump Force and Gravity

Untuk membuat gerakan terasa alami, kami menyesuaikan nilai jumpForce agar tinggi lompatan sesuai keinginan, serta gravityModifier untuk mengatur kecepatan jatuh karakter. Kami melakukan beberapa percobaan hingga mendapatkan keseimbangan yang pas antara ketinggian lompatan dan respons jatuh. Selain itu, kami menambahkan kondisi agar karakter hanya dapat melompat ketika berada di tanah mencegah terjadinya double jump yang tidak diinginkan. Penyesuaian ini penting untuk menjaga pengalaman bermain yang adil dan realistik.

2.1.5 Adding Obstacles and Spawn System

Kami membuat objek Obstacle yang bergerak ke arah kiri secara konstan, mensimulasikan kesan bahwa pemain sedang berlari maju. Kemudian, kami menambahkan Spawn Manager untuk memunculkan rintangan secara berkala di posisi tertentu. Sistem ini diatur agar rintangan muncul setiap beberapa detik.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class SpawnManager : MonoBehaviour
6  {
7      public GameObject obstaclePrefab;
8      private Vector3 spawnPos = new Vector3(25, 0, 0);
9      private float startDelay = 2;
10     private float repeatRate = 2;
11     private PlayerController playerControllerScript;
12
13     // Start is called before the first frame update
14     void Start()
15     {
16         playerControllerScript = GameObject.Find("player").GetComponent<PlayerController>();
17         InvokeRepeating("SpawnObstacle", startDelay, repeatRate);
18     }
19
20     // Update is called once per frame
21     void Update()
22     {
23     }
24
25     void SpawnObstacle()
26     {
27         if(playerControllerScript.gameOver == false)
28         {
29             Instantiate(obstaclePrefab, spawnPos, obstaclePrefab.transform.rotation);
30         }
31     }
32 }
33
34

```

C# source file length: 836 lines: 34 Ln: 31 Col: 10 Pos: 828 Windows (CR LF) UTF-8 INS

2.1.6 Creating a Moving and Looping Background

Untuk menampilkan latar yang terus bergerak, kami membuat script RepeatBackground.cs. Saat latar mencapai batas tertentu, script otomatis mengembalikannya ke posisi awal. Efek ini menciptakan ilusi dunia yang terus bergerak tanpa henti dan membuat tampilan lebih hidup.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class repeatbackground : MonoBehaviour
6  {
7      private Vector3 startPos;
8      private float repeatWidth;
9      // Start is called before the first frame update
10     void Start()
11     {
12         startPos = transform.position;
13         repeatWidth = GetComponent<BoxCollider>().size.x / 2;
14     }
15
16     // Update is called once per frame
17     void Update()
18     {
19         if (transform.position.x < startPos - repeatWidth)
20         {
21             transform.position = startPos;
22         }
23     }
24 }
25

```

C# source file length: 559 lines: 25 Ln: 19 Col: 60 Pos: 502 Windows (CR LF) UTF-8 INS

2.1.7 Adding a Game Over System and Object Management

Untuk membuat permainan terasa lengkap, kami menambahkan sistem Game Over yang aktif ketika karakter pemain menabrak rintangan. Begitu kondisi ini terjadi, semua objek yang bergerak ke kiri seperti latar belakang dan rintangan akan berhenti dengan menghentikan skrip MoveLeft. Selain itu, Spawner Manager

juga dinonaktifkan agar tidak ada rintangan baru yang muncul. Kami juga menambahkan logika untuk menghancurkan objek rintangan yang keluar dari batas layar agar tidak menumpuk di dalam scene, menjaga performa permainan tetap stabil dan efisien.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Space) && isOnGround && !gameOver)
    {
        playerRb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
        isOnGround = false;
    }
}
```

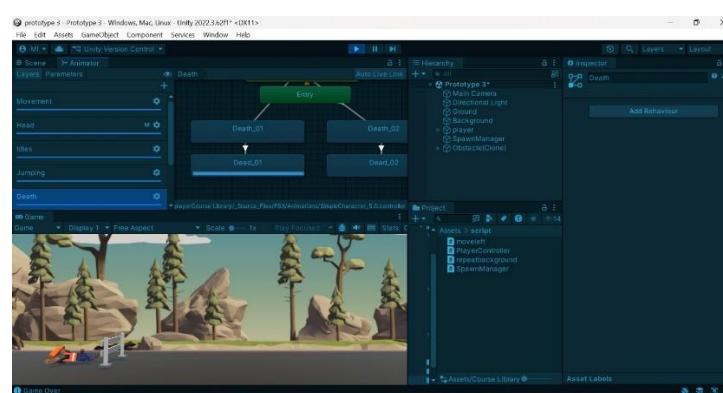
```
C:\Users\arsach\prototype3\Assets\script\moveleft.cs - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
moveleft.cs SpawnManager.cs repeatbackground.cs PlayerController.cs
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class moveleft : MonoBehaviour
6  {
7      private float speed = 30;
8      private PlayerController playerControllerScript;
9      private float leftBound = -15;
10
11     // Start is called before the first frame update
12     void Start()
13     {
14         PlayerControllerScript = GameObject.Find("player").GetComponent<PlayerController>();
15     }
16
17     // Update is called once per frame
18     void Update()
19     {
20         if (playerControllerScript.gameOver == false)
21         {
22             transform.Translate(Vector3.left * Time.deltaTime * speed);
23         }
24
25         if (transform.position.x < leftBound && gameObject.CompareTag("Obstacle"))
26         {
27             Destroy(gameObject);
28         }
29     }
30 }
31
```

length: 745 lines: 31 Ln: 25 Col: 83 Pos: 699 Windows (CR LF) UTF-8 INS

2.1.8 Setting Up and Synchronizing Player Animations

Kami menggunakan Animator Controller untuk mengatur animasi seperti berlari, melompat, dan jatuh. Run State dijadikan animasi utama, lalu kami menambahkan parameter agar animasi bisa berganti berdasarkan input pemain atau kondisi karakter. Ini membuat gerakan karakter tampak lebih halus dan alami.

```
private Animator playerAnim;
```

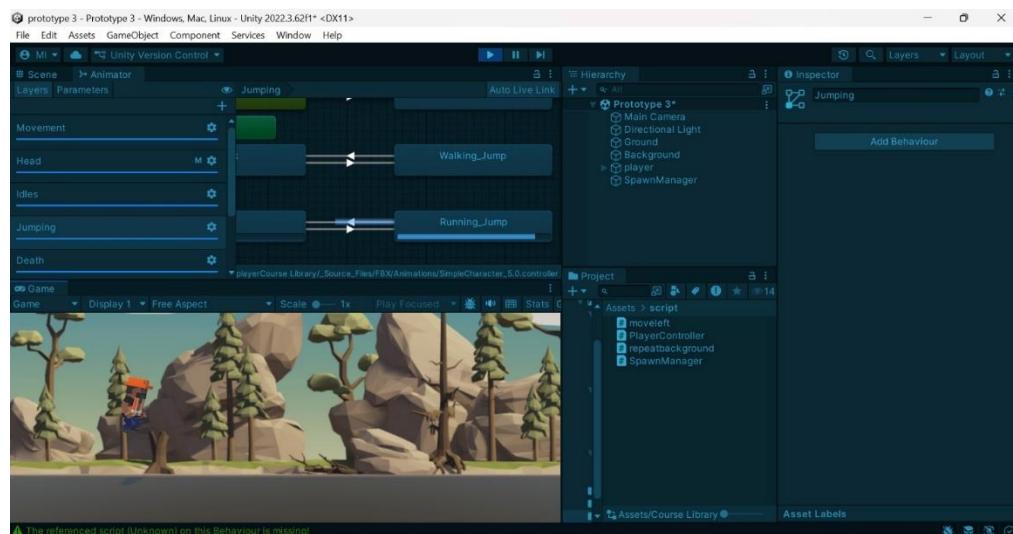


2.1.9 Refining Animation Transitions and Logic

Agar animasi terlihat halus, kami menyesuaikan transition duration dan menghapus exit time yang tidak diperlukan. Kami juga menambahkan kondisi yang menghubungkan kecepatan vertikal karakter dengan animasi misalnya, saat kecepatan vertikal negatif, maka animasi Falling akan diputar. Kami memastikan animasi lompatan hanya berjalan ketika karakter benar-benar berada di tanah, sehingga tidak terjadi bug seperti melompat di udara.

```
playerAnim.SetTrigger("Jump_trig");
dirtParticle.Stop();

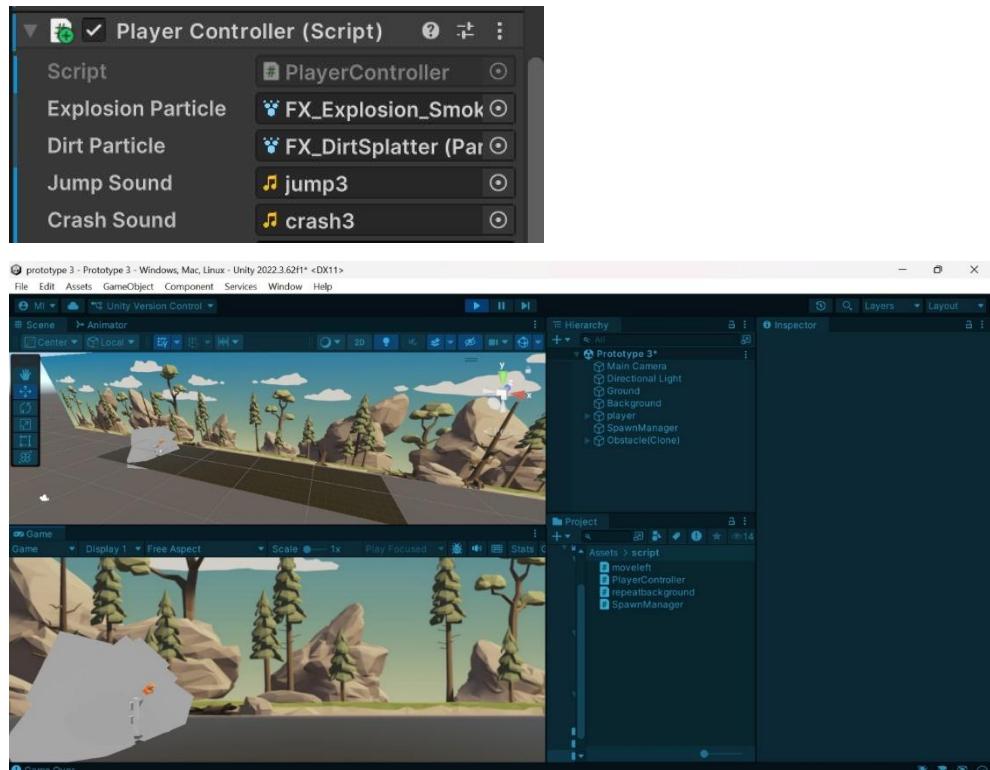
playerAnim.SetBool("Death_b", true);
playerAnim.SetInteger("DeathType_int", 1);
```



2.1.10 Adding Particle and Sound Effects

Untuk memperkaya tampilan visual dan audio, kami menambahkan berbagai efek partikel seperti ledakan, debu, dan percikan tanah yang muncul saat tabrakan atau ketika karakter melompat. Selain itu, kami menambahkan musik latar (background music) ke kamera agar terdengar di seluruh scene, dan juga menambahkan efek suara seperti suara lompatan dan benturan yang dipicu oleh aksi tertentu.

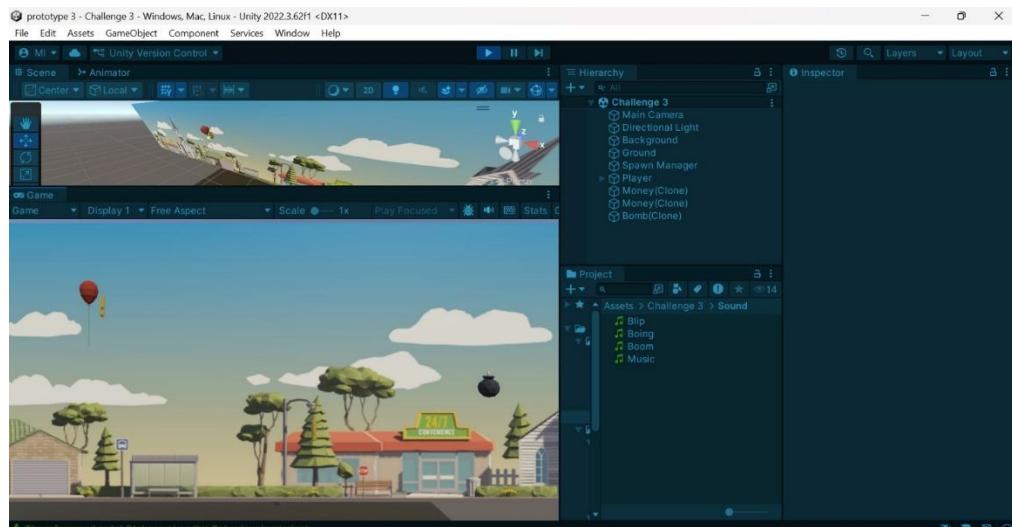
```
private AudioSource playerAudio;
public ParticleSystem explosionParticle;
public ParticleSystem dirtParticle;
public AudioClip jumpSound;
public AudioClip crashSound;
```



2.2 Challenge 3 - Balloons, Bombs, & Booleans

2.2.1 Understanding and Analyzing the Challenge

Pada tantangan ini, kami diminta menganalisis logika dalam script PlayerControllerX.cs dan SpawnManagerX.cs. Kode yang disediakan tidak sepenuhnya benar, sehingga kami harus memahami cara kerja variabel dan kondisi boolean agar game berjalan sesuai fungsinya.



2.2.2 Fixing Balloon Control and Background Movement

Masalah utama adalah balon tidak bisa dikendalikan dan latar belakang hanya bergerak setelah game selesai. Kami memperbaikinya dengan menambahkan gaya (Rigidbody.AddForce) hanya ketika *isGameOver == false*, sehingga balon bisa melayang saat permainan masih aktif. Kondisi logika pada animasi latar juga kami ubah agar hanya bergerak selama permainan berlangsung, bukan setelah *game over*.

2.2.3 Fixing Object Spawning and Firework Position

Objek seperti bom dan kembang api tidak muncul karena *SpawnManager* berhenti terlalu cepat. Kami memastikan pemunculan objek hanya berhenti ketika *isGameOver == true*. Selain itu, posisi kembang api kami sesuaikan dengan mengatur transformasi lokal agar efek muncul di dekat balon, bukan di samping jauh.

2.2.4 Smoothing Background Loop and Adding Movement Limits

Latar belakang tidak berulang dengan benar karena logika *repeatWidth* salah. Kami memperbaikinya dengan kondisi: if (*transform.position.x < startPos.x - repeatWidth*)

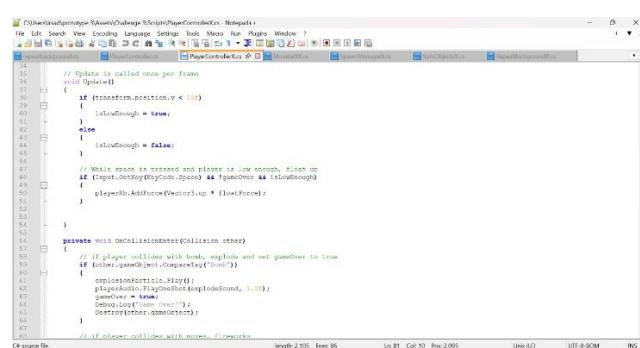
sehingga latar bergerak mulus tanpa putus. Kami juga menambahkan batas atas dan bawah agar balon tidak keluar layar, menggunakan logika boolean sederhana untuk membatasi posisi pada sumbu Y.

2.2.5 Debugging With Boolean Logic

Kami menggunakan *Debug.Log()* untuk melacak nilai boolean dan memastikan setiap kondisi bekerja sesuai tujuan. Cara ini memudahkan kami memahami alur logika dan memperbaiki kesalahan lebih cepat.

```
else if (other.gameObject.CompareTag("Ground"))
{
    playerRb.AddForce(Vector3.up * 10, ForceMode.Impulse);

    playerAudio.PlayOneShot(bounceSound, 1.0f);
}
```



```
else if (other.gameObject.CompareTag("Ground"))
{
    playerRb.AddForce(Vector3.up * 10, ForceMode.Impulse);

    playerAudio.PlayOneShot(bounceSound, 1.0f);
}

// Update is called once per frame
void Update()
{
    if (transform.position.y < 1.0)
        isGrounded = true;
    else
        isGrounded = false;

    // While space is pressed and player is low enough, float up
    if (Input.GetMouseButton(0))
        playerRb.AddForce(Vector3.up * (float)Force);
}

private void OnCollisionEnter(Collision other)
{
    // If player collides with bomb, explode and set gameOver to true
    if (other.gameObject.CompareTag("Bomb"))
    {
        explosionForce = 5000;
        playerRb.AddExplosionForce(explosionForce, Vector3.up, 1.0f);
        gameOver = true;
        Debug.Log("Game Over!");
        Destroy(gameObject);
    }
}

// If player collides with balloon, fireworks
if (other.gameObject.CompareTag("Balloon"))
{
    // If balloon is alive enough
    if (isAliveEnough)
    {
        // If balloon is not yet exploded
        if (!balloon.isExploded)
        {
            balloon.isExploded = true;
            balloon.exploded();
        }
    }
}
```

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlayerControllerX : MonoBehaviour
6  {
7      public bool gameOver;
8
9      public float floatForce;
10     private float gravityModifier = 1.5f;
11     private Rigidbody playerRb;
12
13     public ParticleSystem explosionParticle;
14     public ParticleSystem fireworksParticle;
15
16     private AudioSource playerAudio;
17     public AudioClip moneySound;
18     public AudioClip explodeSound;
19     public AudioClip bounceSound;
20
21     private bool isLowEnough = true;
22
23     // Start is called before the first frame update
24     void Start()
25     {
26         playerRb = GetComponent<Rigidbody>();
27         playerAudio = GetComponent<AudioSource>();
28
29         // Apply a small upward force at the start of the game
30         Physics.gravity *= gravityModifier;
31         playerRb.AddForce(Vector3.up * 5, ForceMode.Impulse);
32     }
33
34     // Update is called once per frame
35 }

```

C# source file length: 2,105 lines: 86 Lx: 81 Col: 10 Pos: 2,095 Unix (LF) UTF-8 BOM INS

2.3 Lab 3 - Player Control

2.3.1 Create Player Controller and Plan Your Code

Tahap ini menekankan pentingnya perencanaan struktur kode sebelum implementasi. Pengembang membuat script PlayerController untuk mengatur logika utama gerakan pemain.

2.3.2 Basic Movement From User Input

Di sini memahami bagaimana Unity membaca input dari pemain (seperti tombol panah atau WASD) dan mengubahnya menjadi gerakan objek. Ini merupakan dasar dari interaksi antara pemain dan lingkungan game.

2.3.3 Constrain The Player's Movement

Pada bagian ini, diterapkan logika pembatasan agar pemain tidak keluar dari area permianan. Konsepnya menggunakan batas koordinat tertentu atau collider agar ruang permianan tetap terkendali. Setelah sistem kontrol berjalan, tahap ini berfokus pada optimalisasi. Kode dibersihkan dari elemen yang tidak perlu dan proyek diekspor sebagai cadangan (backup), memastikan hasil kerja aman dan mudah digunakan ulang.

BAB III

PENUTUP

3.1 Kesimpulan

Pada Unity Junior Programmer Pathway unit Sound and Effects, kami belajar bagaimana menggabungkan elemen audio, efek visual, dan logika pemrograman untuk menciptakan pengalaman bermain yang menarik. Melalui tahap ini, kami memahami cara menambahkan musik latar, efek suara, serta partikel visual yang selaras dengan aksi dalam game agar suasana permainan terasa lebih hidup dan responsif.

Dalam Challenge 3 – Balloons, Bombs, & Booleans, kami mendalami penerapan boolean logic untuk mengatur kondisi permainan seperti pergerakan latar, pemunculan objek, dan pembatasan ruang gerak karakter. Kami juga belajar menganalisis serta memperbaiki kesalahan logika melalui proses debugging.

Secara keseluruhan, materi ini memperkuat kemampuan kami dalam menggunakan Unity dan C#, melatih cara berpikir logis, serta meningkatkan kreativitas dalam mengelola elemen audio-visual agar game menjadi lebih interaktif dan profesional.