A. D. Patel Institute of Technology

(A Constituent college of CVM University)

New V.V. Nagar

Computer Science and Design Department

Project Report

On

Platformer Game

Submitted by

Name: Drashti Modi

Enrolment Number: 12102130601023

Guided By:

Prof. Siddharth Shah

Certification

This is to certify that the Mini Project Report submitted entitled **"Platformer Game"** has been carried out by **Dwiti Bajaj** (121021306010) under guidance in partial fulfilment for the Degree of Bachelor of Technology in Computer Science and Design, 6th Semester of A D Patel Institute of Technology, CVM University, New Vallabh Vidyanagar during the academic year 2023-24.

Prof. Siddharth Shah

Internal Guide

Acknowledgement

We would like to express our sincere gratitude to the following individuals and organizations for their invaluable contributions and support throughout the development of our Platformer Game project:

We would like to thank our mentor Prof. Siddharth Shah for her guidance, encouragement, and valuable feedback that helped shape our project and enhance its quality.

We extend our heartfelt thanks to our team members for their collaboration, hard work, and dedication. Each member's unique contributions played a crucial role in the success of this project.

We are grateful to A.D. Patel Institute of Technology for providing us with the necessary resources, facilities, and opportunities to undertake this project.

Special thanks to Project in charge Prof. Siddharth Shah for sharing their expertise and insights, as well as for their assistance and feedback.

We appreciate the support and encouragement from all individuals who contributed to our project in any way, whether through motivation, inspiration, or practical assistance

Abstract

Platformer games have been a cornerstone of the video game industry since its inception, offering players dynamic experiences characterized by precise controls, engaging level design, and immersive gameplay. This abstract provides an overview of our research, which delves into the mechanics of platformer games, examining their evolution, key components, and design principles.

Our study begins by analysing the historical roots of platformer games, tracing their origins from early arcade classics to modern iterations across various gaming platforms. We explore how technological advancements have shaped the evolution of platformers, from the pixelated worlds of the past to the visually stunning landscapes of contemporary titles.

Furthermore, we conduct a comparative analysis of popular platformer games, dissecting their gameplay mechanics, level design philosophies, and player engagement strategies. Through this analysis, we identify common elements that contribute to the success of platformer games, such as tight controls, responsive feedback, and a balance between challenge and reward.

In addition to examining existing titles, we propose a framework for designing compelling platformer experiences. Drawing upon insights from our analysis and incorporating principles of game design theory, we outline guidelines for creating immersive worlds, crafting meaningful challenges, and fostering player agency within the game environment.

Overall, our research sheds light on the enduring appeal of platformer games and provides valuable insights for developers seeking to innovate within this genre. By understanding the mechanics and design principles that underpin successful platformer experiences, developers can create games that captivate players and stand the test of time in an ever-evolving gaming landscape.

Table of Contents

1. Introduction

1.1 Background:

Platformer games, also known as platform games or jump-and-runs, form a genre deeply entrenched in the history of video gaming. They trace their roots back to the early days of arcade gaming in the 1980s when developers sought to create experiences that capitalized on the emerging capabilities of home gaming consoles and personal computers.

One of the earliest and most iconic examples of the platformer genre is Nintendo's "Donkey Kong" released in 1981, which introduced players to the concept of navigating a character through a series of platforms while avoiding obstacles and enemies. This pioneering title laid the foundation for what would become a staple genre in the gaming industry.

Throughout the 1980s and 1990s, platformer games experienced a golden age with the rise of iconic franchises such as "Super Mario," "Sonic the Hedgehog," and "Mega Man." These games showcased innovative level design, tight controls, and imaginative worlds, captivating players, and establishing enduring legacies in gaming culture.

The platformer genre continued to evolve with the advent of three-dimensional graphics and advanced hardware capabilities in the late 1990s and early 2000s. Games like "Super Mario 64" and "Banjo-Kazooie" pushed the boundaries of what was possible in terms of exploration, immersion, and gameplay mechanics, ushering in a new era of 3D platforming experiences.

In the contemporary gaming landscape, platformer games remain a vibrant and diverse genre, encompassing a wide range of styles and subgenres. From indie darlings like "Celeste" and "Hollow Knight" to blockbuster titles such as "Super Mario Odyssey" and "Rayman Legends," platformer games continue to captivate players with their timeless appeal, inventive gameplay mechanics, and immersive worlds.

With the advent of digital distribution platforms and accessible game development tools, the barrier to entry for creating platformer games has lowered, leading to an explosion of creativity and innovation within the genre. Today, developers of all sizes could contribute to the rich tapestry of platformer gaming, pushing the boundaries of

what is possible and delighting players with fresh experiences that harken back to the genre's storied history.

1.2 Objectives:

Through this project, we seek to address the following objectives:

- To provide players with an engaging and enjoyable gameplay experience.
- Encourage exploration and discovery within their game worlds.
- To provide players with a sense of mastery and accomplishment as they overcome challenges and progress through the game.
- To create platformer games with high replay value, offering players incentives to revisit levels, discover new pathways, or attempt speedruns and challenges.
- Offer customizable difficulty settings, accessible control options, and inclusive representation in character design and storytelling.

1.3 Scope

The scope of this project encompasses:

- The scope includes designing gameplay mechanics, art, audio, and user interfaces, ensuring technical implementation, testing, localization, and engaging community support.
- Post-launch plans may involve updates and monetization strategies. The goal is to create an immersive, polished platformer game that delights players and sustains commercial success.

2. Assets Phase

2.1 Finding Assets:

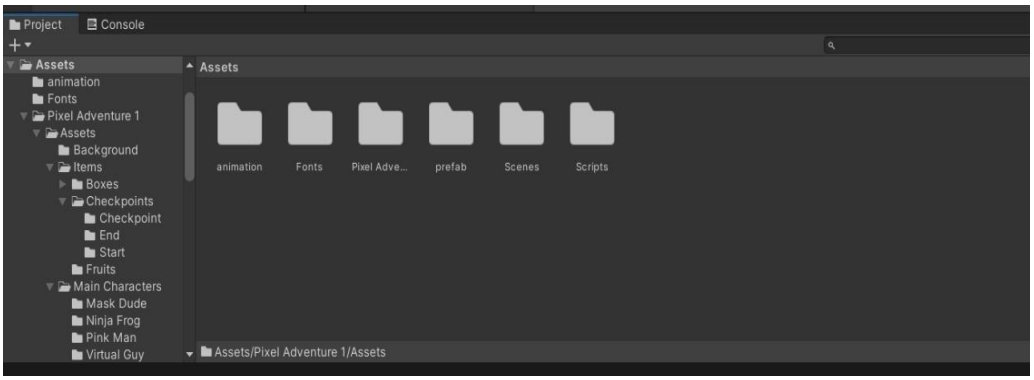
In the realm of Unity3D platformer game development, acquiring assets is paramount to creating an immersive and engaging gaming experience. The process of asset acquisition spans across diverse channels, each offering unique resources tailored to the specific needs of developers. Firstly, the Unity Asset Store stands as a cornerstone resource, providing an extensive array of assets ranging from character controllers to visual effects. Additionally, community forums, websites, and social media groups within the Unity3D ecosystem serve as vibrant hubs where developers share and recommend assets, both free and paid, facilitating collaborative resource sharing. Moreover, third-party asset marketplaces offer an alternative avenue for asset acquisition, offering a plethora of options beyond the official Unity store. Curated asset packs and bundles are another valuable resource, providing developers with collections of assets designed explicitly for platformer game development, streamlining the asset acquisition process. Open-source projects on platforms like GitHub further contribute to the asset pool, offering scripts, tools, and assets shared by the developer community. Furthermore, tutorials and templates available online furnish developers with step-by-step guidance and downloadable assets, expediting the development process. Lastly, asset creation tools, whether integrated within Unity3D or external software, empower developers to craft custom assets tailored to their platformer game's unique requirements. Through a comprehensive exploration of these channels, developers can assemble a diverse array of assets to enrich their Unity3D platformer game projects, enhancing gameplay, aesthetics, and overall player experience.

Websites:

- Unity Asset Store
- CGTrader
- Itch.io

2.2 Importing Assets:

Once assets are sourced from various channels, importing them into Unity3D is a pivotal step in the game development process. Unity's intuitive interface facilitates seamless asset integration, allowing developers to easily incorporate diverse elements into their platformer games. To begin, assets obtained from the Unity Asset Store or third-party marketplaces can be downloaded directly to a local directory. Within the Unity Editor, developers navigate to the Project panel and select the designated folder where assets will be stored. Dragging and dropping assets from the local directory into the Project panel initiates the import process. Unity automatically parses the assets, generating appropriate meta files and organizing them within the project hierarchy. For assets obtained from open-source platforms or created using external tools, developers may import them similarly by dragging and dropping into the Unity Editor. Once imported, assets are readily accessible for use within the game environment. Developers can manipulate imported assets by adjusting settings, applying textures, configuring scripts, and integrating them into scenes using Unity's robust toolset. By streamlining the import process, Unity empowers developers to efficiently leverage a diverse array of assets, facilitating the creation of captivating and immersive platformer games.



3. Designing Phase

3.1 Creating Project

Creating a project in Unity is a fundamental step towards developing your platformer game. Follow these steps to set up your project:

1. Launch Unity:
   - Open the Unity Hub application on your computer. If you haven't installed Unity yet, download and install it from the official Unity website.

2. Create a New Project:
   - In the Unity Hub, click on the "New" button to create a new project.

3. Project Configuration:
   - Template:
     - Choose the appropriate template for your project. For a platformer game, you can start with the 2D template.
   - Project Name:
     - Give your project a descriptive name.
   - Location:
     - Specify the location on your computer where you want to save the project files.
   - Unity Version:
     - Select the Unity version you want to use for the project. It's recommended to use the latest stable version available unless you have specific reasons for choosing an older version.

4. Create Project:

   - Once you've configured the project settings, click on the "Create" button to create the project. Unity will initialize the project files and open the Unity Editor.

5. Unity Editor:

- The Unity Editor is where you'll build and design your game. Familiarize yourself with the different panels and windows, such as the Scene view, Game view, Hierarchy, Project, and Inspector panels.

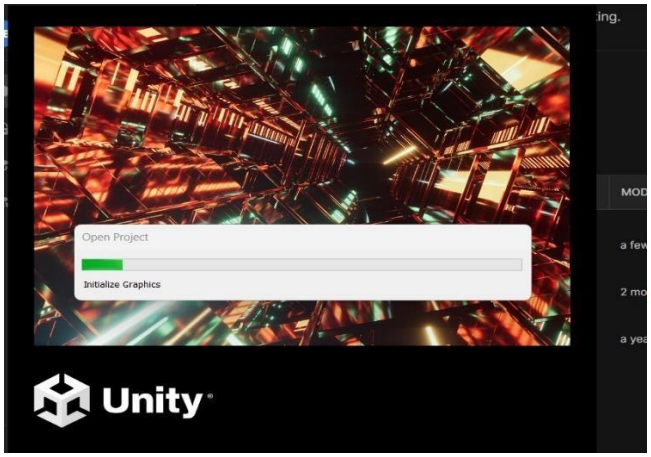
6. Import Assets:
   - If you have assets such as sprites, animations, or scripts for your platformer game, you can import them into the project by dragging and dropping them into the Project panel in Unity. Alternatively, you can use the "Import" button in the Project panel to import assets from your computer.

7. Scene Setup:
   - Start by setting up your game scene. Create a new scene by going to File > New Scene. In the scene, you can design the layout, place platforms, add characters, and set up the environment.

8. Scripting:
   - If you need custom functionality for your platformer game, you'll likely need to write scripts. Create C# scripts by right-clicking in the Project panel and selecting "Create > C# Script." Double-click the script to open it in your preferred code editor and start coding.

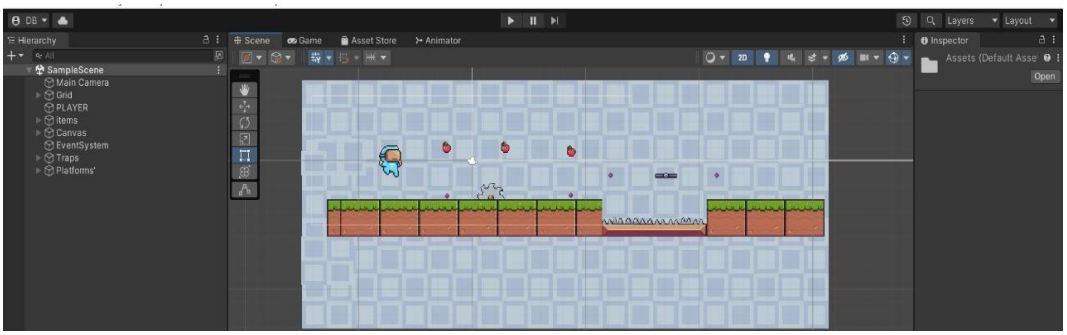
9. Testing:
   - Continuously test your game as you develop it. Use the Play button in the Unity Editor to playtest your game scene and make adjustments as needed.

10. Save and Build:
    - Remember to save your project regularly using File > Save Project. When you're ready to build your game for distribution, go to File > Build Settings, select your target platform, and click on the "Build" button.



3.2 Creating GameObjects

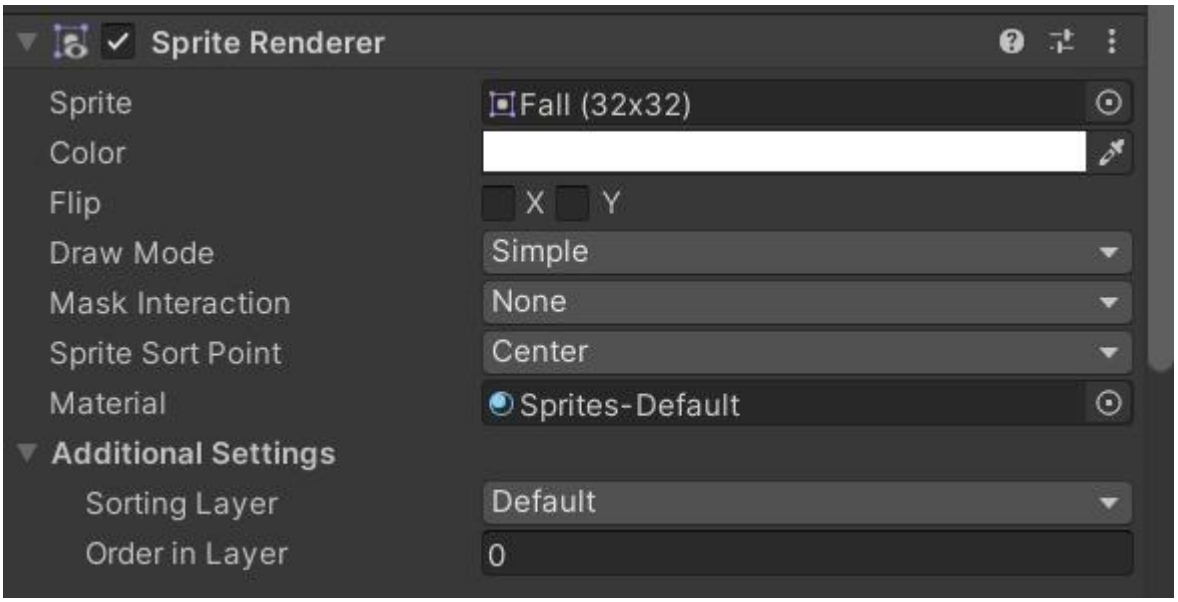
In Unity, creating GameObjects is foundational to constructing the environment and entities within your platformer game. Utilizing the Unity Editor, developers navigate to the Hierarchy panel, where they can initiate the creation of a new GameObject by right-clicking or through the GameObject menu. Once generated, developers typically assign descriptive names to these GameObjects, facilitating organization within the scene. GameObjects derive functionality and appearance through components attached to them. Developers can easily add components via the Inspector panel, tailoring GameObject behavior to suit specific needs, whether it involves physics simulations, collisions, or rendering. Unity offers built-in primitive shapes for creating 3D objects, streamlining the process of designing levels and environmental elements. Moreover, importing assets such as 3D models or spritesheets enables the creation of more intricate GameObjects, including characters, enemies, and props. GameObjects can be arranged hierarchically by establishing parent-child relationships, facilitating scene organization and management. Additionally, GameObject instantiation via scripting empowers developers to dynamically generate entities during runtime, enabling dynamic and responsive gameplay experiences. Through these methods, developers can systematically create and manipulate GameObjects to construct immersive worlds for their platformer games within the Unity environment.



3.3 Editing Sprites

Editing sprites in Unity allows developers to customize and refine the visual elements of their platformer games. After importing sprites into the Unity project, developers select the desired sprite in the Assets panel, revealing its properties in

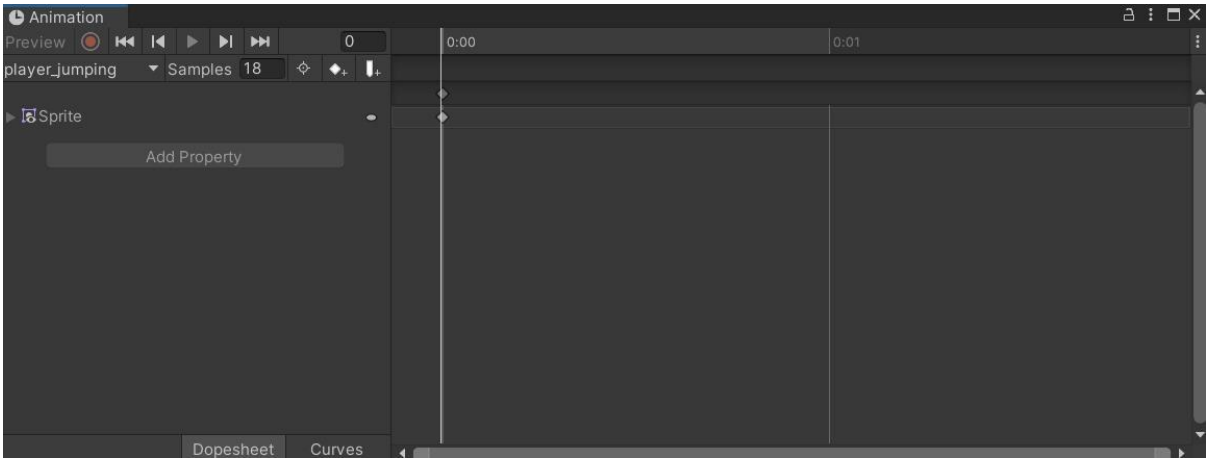
the Inspector panel. Here, adjustments can be made to attributes such as texture, pivot point, and rendering mode. For more advanced edits, developers can utilize the Sprite Editor by double-clicking on the sprite in the Assets panel. Within the Sprite Editor, a range of editing tools is available, including cropping, resizing, and rotating functionalities. Cropping enables the removal of unwanted portions of the sprite, enhancing optimization and performance. Resizing options facilitate adjustments to the sprite's dimensions, while rotation tools allow for orientation modifications. Colour and transparency adjustments can be made through material properties, enabling developers to tweak the visual appearance of sprites. Once edits are complete, changes can be saved using the Apply button in the Sprite Editor. Testing the sprite in the game scene ensures that edits have been successfully applied, providing developers with the opportunity to fine-tune visuals to their desired specifications. Through these editing processes, developers can effectively tailor sprites to suit the aesthetic and functional requirements of their platformer games within the Unity environment.

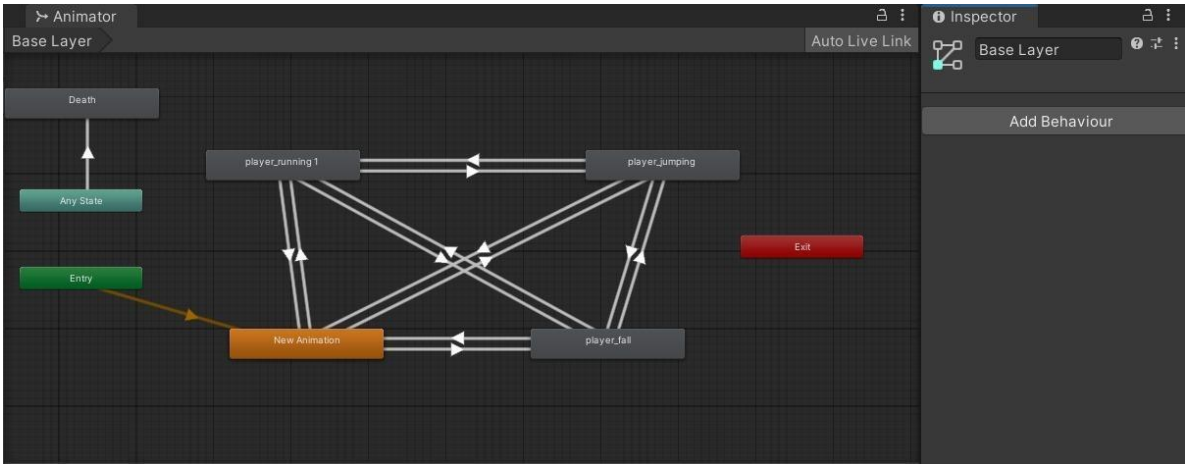


3.4 Creating Animations

Creating animations in Unity enables developers to breathe life into characters, objects, and environmental elements within their platformer games. The process typically begins with importing sprite sheets or individual image frames representing different animation states, such as walking, jumping, or attacking. Once imported, these assets can be organized into animation clips using Unity's Animation window. Within the Animation window, developers can define

keyframes for each animation state, specifying the position, rotation, and scale of sprites at different points in time. By interpolating between keyframes, Unity automatically generates smooth transitions between animation states, creating fluid and dynamic motion. Additionally, developers can utilize the Animator component to control the playback of animations in response to game events or user input. By configuring animation parameters and transitions within the Animator Controller, developers can create complex animation behaviors, such as blending between multiple animation states or triggering animations based on specific conditions. Through these tools and workflows, developers can craft engaging and immersive animations that enhance the gameplay experience and bring their platformer games to life.




Animator
Base Layer
Auto Live Link
Death
Any State
Entry
player_running 1
player_jumping
Exit
New Animation
player_fall
Inspector
Base Layer
Add Behaviour

4. Coding Phase

4.1 Giving Movements

Giving movements to objects, characters, and elements within a platformer game in Unity involves animating them to create dynamic and interactive experiences.

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

public class Babla : MonoBehaviour
{
    public Rigidbody2D rb;

    public float speed=7f;
    public Animator anim;
    public SpriteRenderer sprite;
    private BoxCollider2D box;
    [SerializeField]private LayerMask Jumpable;
    // Start is called before the first frame update
    void Start()
    {

        anim = GetComponent<Animator>();
        sprite = GetComponent<SpriteRenderer>();
        rb = GetComponent<Rigidbody2D>();
        box = GetComponent<BoxCollider2D>();
    }

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

        float h = Input.GetAxisRaw("Horizontal");
        rb.velocity = new Vector2( (h*speed), rb.velocity.y);

        if(Input.GetButtonDown("Jump") && isGrounded())
```

```
        {
            rb.velocity=new Vector2(rb.velocity.x, 9f);

        }

        if(h>0f)
        {
            anim.SetInteger("state", 1);
            sprite.flipX = false;
        }

        else if(h<0f)
        {
            anim.SetInteger("state", 1);
            sprite.flipX = true;
        }

        else
        {
            anim.SetInteger("state", 0);
        }

        if(rb.velocity.y>0.1f)
        {
            anim.SetInteger("state", 2);
        }

        else if (rb.velocity.y < -0.1f)
        {
            anim.SetInteger("state", 3);
        }

    }
    private bool isGrounded()
    {
        return
Physics2D.BoxCast(box.bounds.center,box.bounds.size,0f,Vector2.down,0.1f,Jumpa
ble);

    }
}
```

4.2 Controlling Camera

Controlling the camera in a platformer game is crucial for providing players with a clear view of the game world and ensuring smooth navigation through levels.

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

public class CameraController : MonoBehaviour
{
    [SerializeField] private Transform Player;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        transform.position = new Vector3(Player.transform.position.x,
Player.transform.position.y, transform.position.z);
    }
}
```

4.3 Collecting Items

In a platformer game, collecting items adds depth and engagement to the gameplay by rewarding players for exploration and skilful navigation.

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

public class CollectItems : MonoBehaviour
```

```
{
    int apples = 0;
    [SerializeField] private Text appleText;


    // Update is called once per frame
    private void OnTriggerEnter2D(Collider2D collision)
    {

        if(collision.gameObject.CompareTag("apple"))
        {
            Destroy(collision.gameObject);
            apples++;
            appleText.text = "Apples - "+ apples;
            Debug.Log("apples : "+ apples);
        }
    }
}
```

4.4 Player Life & Scene Restart

Implementing player life and scene restart mechanics is essential for maintaining gameplay flow and providing players with a sense of challenge and progression in a platformer game.

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

public class PlayerLife : MonoBehaviour
{
    private Animator anim;
    private Rigidbody2D rb;
    // Start is called before the first frame update
    void Start()
    {
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
    }
```

```
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if(collision.gameObject.CompareTag("trap"))
        {
            anim.SetTrigger("death");
            rb.bodyType = RigidbodyType2D.Static;
            // wait for a second


        }
    }

    private void Restart()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

4.5 Creating Platform

Creating platforms in Unity for a platformer game involves setting up objects that the player can stand on, jump across, and interact with.

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

public class StickyPlatform : MonoBehaviour
{
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if(collision.gameObject.name == "PLAYER")
        {
            collision.gameObject.transform.SetParent(transform);
        }
    }

    private void OnCollisionExit2D(Collision2D collision)
    {
        if(collision.gameObject.name == "PLAYER")
```

```
        {
            collision.gameObject.transform.SetParent(null);
        }
    }

}
```

4.6 Creating and Following Waypoints

Creating and following waypoints is a common technique used in game development, especially for guiding non-player characters (NPCs) or controlling moving objects along predefined paths.

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

public class WayPointFollower : MonoBehaviour
{

    [SerializeField]private GameObject[] waypoints;
    int currentwaypointindex = 0;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if(Vector2.Distance(transform.position,
waypoints[currentwaypointindex].transform.position)<0.1f)
        {
            currentwaypointindex++;

            if(currentwaypointindex>=waypoints.Length)
                currentwaypointindex = 0;
        }
```

```
        transform.position = Vector2.MoveTowards(transform.position,
waypoints[currentwaypointindex].transform.position, 2f*Time.deltaTime);
    }
}
```

5. Conclusion

In conclusion, the implementation of waypoint systems in Unity offers a versatile and effective method for guiding object movement along predefined paths within a game environment. By carefully defining and arranging waypoints, developers can create dynamic and engaging movement patterns for NPCs, enemies, or other moving objects, enhancing gameplay depth and immersion. Using waypoint following scripts and careful management of waypoint transitions, objects can smoothly traverse complex paths, interact with obstacles, and respond to dynamic game events. Additionally, the flexibility of Unity's scripting and component-based architecture allows for the customization of waypoint systems to suit specific game mechanics and design requirements. Overall, the integration of waypoint systems in Unity empowers developers to create immersive and interactive experiences that captivate players and elevate the quality of their games.