3D Endless Runner Project

Team Members:

Yasmine Zeineldin 320230084

Amnah ElKheshen 320230082

Yousef Warshana 320230192

Introduction

This project represents the development of a 3D Infinite Runner game created using Unity Engine and programmed in C#. The core idea of the game is to allow the player to survive for as long as possible while avoiding dynamically spawned obstacles. The game follows the classic infinite runner concept, where the environment continues endlessly and the player's score increases over time rather than through predefined levels.

The game is designed to be simple yet challenging. The player controls a character that can jump to avoid incoming spikes. As time progresses, the difficulty naturally increases due to random obstacle spawning, which keeps the gameplay engaging and unpredictable. A scoring system tracks the player's survival time, and a high score system saves the best performance using Unity's persistent storage.

Tools & Frameworks Used:

- Game Engine: Unity 6 (6000.2.14f1)
- Scripting Language: C# (C-Sharp)
- IDE: Visual Studio / VS Code
- Version Control: https://github.com/YasmineZein/3D-Runner.git

Project Structure and Team Distribution

The project was divided into three main parts:

Part 1 – Game Management & Scoring System

Responsible Member: Yasmine Zein El-Din

This part focuses on overall game control, obstacle spawning, score calculation, and high score persistence.

Part 2 – Player Mechanics & Physics

Responsible Member: Youssef Warshana

This part handles player movement, jumping logic, collision with the ground, and physics-based behavior.

Part 3 – UI, Scene Control & Obstacles

Responsible Member: Amnah El-Kheshen

This part manages scene reloading, level navigation, UI interactions, obstacle movement, and game-over detection.

Part 1: Game Management & Scoring System

The GameManager script is the backbone of the game. It controls the game flow, obstacle spawning, score increment, and high score logic.

Code Snippet:

GameManager Declaration

```
public class GameManager : MonoBehaviour
{
    public GameObject spike;
    public Transform spawnPoint;
    public float score = 0;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI highScoreText;
}
```

Explanation:

This section defines the main variables used in the game. The spike object is the obstacle prefab that will be spawned continuously. The spawnPoint determines where obstacles appear. The score variable tracks survival time, while TextMeshProUGUI elements are used to display the current score and best score on the screen.

Game Initialization

```
void Start()
{
    GameStart();
    int savedHighScore = PlayerPrefs.GetInt("HighScore", 0);
    highScoreText.text = "Best: " + savedHighScore.ToString();
}
```

Explanation:

When the game starts, the obstacle spawning coroutine is launched. The high score is retrieved from PlayerPrefs, which allows the score to persist even after closing the game. If no score exists, it defaults to zero.

Score Update Logic

```
void Update()
{
    if (GameObject.FindWithTag("Player") != null)
    {
        score += Time.deltaTime;
        scoreText.text = ((int)score).ToString();
    }
}
```

Explanation:

The score increases every frame as long as the player exists in the scene. Time.deltaTime ensures the score increases based on real time rather than frame rate, making scoring fair across different devices.

Obstacle Spawning System

```
IEnumerator SpawnSpikes()
{
    while(true)
    {
        float waitTime = Random.Range(0.5f, 2f);
        yield return new WaitForSeconds(waitTime);
        float pos = Random.Range(0f, 1f);
        Vector3 newpos;
        if(pos > 0.5f)
            newpos = new Vector3(spawnPoint.position.x,
spawnPoint.position.y + 1.5f, spawnPoint.position.z);
        else
            newpos = spawnPoint.position;
        Instantiate(spike, newpos, Quaternion.identity);
    }
}
```

Explanation:

This coroutine spawns spikes at random intervals and at two different heights. The randomness ensures the game does not become predictable, increasing replay value and difficulty over time.

High Score Validation

```
public void CheckHighScore()
{
    int currentScore = (int)score;
    int savedHighScore = PlayerPrefs.GetInt("HighScore",
0);



    if (currentScore > savedHighScore)
    {
        PlayerPrefs.SetInt("HighScore", currentScore);
        PlayerPrefs.Save();
        highScoreText.text = "New Best Score: " +
currentScore;
    }
}
```

Explanation:

This method compares the player's score with the saved high score and updates it if a new record is achieved. This adds motivation for the player to keep improving.

Part 2: Player Mechanics & Physics

This script manages player movement and jumping using Unity's physics system.

Player Initialization

```
void Awake()
{
    rb = GetComponent<Rigidbody>();
}
```

Explanation:

The Rigidbody component is retrieved to apply physics-based forces such as jumping.

Jump Logic

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.Space) &&
canJump)
    {
        rb.AddForce(Vector3.up * jumpForce,
ForceMode.Impulse);
    }
}
```

Explanation:

The player can jump only when grounded. Using ForceMode.Impulse creates a realistic jump effect.

Ground Detection

```
private void OnCollisionEnter(Collision
collision)
{
    if(collision.gameObject.CompareTag("ground"))
        canJump = true;
}
```

Explanation:

This ensures the player cannot jump infinitely in the air, maintaining fair gameplay.

Part 3: UI, Scene Control & Obstacles

Scene Control Script

The reload script manages restarting the game, loading new levels, toggling modes, and quitting.

```
public void RestartGame()
{

SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
```

Explanation:

Reloads the current scene to restart the game after losing.

Spike Movement Script

```
void Update()
{
    transform.Translate(Vector3.left * speed * Time.deltaTime);
}
```

Explanation:

Spikes move toward the player continuously, simulating forward player movement while keeping the player stationary.

Game Over Detection

```
private void OnTriggerEnter(Collider collision)
{
    if(collision.gameObject.CompareTag("Player"))
    {
        collision.gameObject.SetActive(false);

GameObject.FindAnyObjectByType<GameManager>().Che
ckHighScore();
    }
}
```

Explanation:

When the player hits a spike, the character is disabled and the high score is checked. This cleanly ends the run.

Conclusion

This project successfully demonstrates the development of a 3D Infinite Runner game using Unity and C#. The game combines physics-based player control, randomized obstacle spawning, real-time scoring, and persistent data storage. The modular structure allowed the team to work efficiently while integrating all systems seamlessly.

Overall, the project reflects a solid understanding of Unity fundamentals, C# scripting, and basic game design principles, while leaving room for future improvements such as animations, sound effects, and difficulty scaling.

Screenshots of the game