Faculty of Computer Science & Engineering
Advanced Web Design

Ѓорѓи Галевски, 196041

Спасе Костадински, 196116

Faculty of Computer Science & Engineering 1

Project Report: Unity WebGL Snake Game 3
Abstract 3
1. Introduction 3
Project Goals 3
Objectives 3
2. History of the Game Snake 4
Origin and Evolution 4
Influence on the Gaming Industry 4
3. Game Overview 4
Snake Game Features 4
Design Choices 5
Game Mechanics 5
Technical Details 5
4.Technical Implementation 5
Code Overview 5
Start Screen class 4
Game Over Screen (GameOverScreen.cs) 5
Snake Class 6
ScoreManager Class 8
Food Class 9
Enhanced Audio 10
Collaborative Development 11
7. Development Process 13
Unity and WebGL 13
Development Phases 13
Hosting and Deployment 13
itch.io: The Online Hosting Solution 13
Version Control 14
8. Challenges and Solutions 14
Compatibility Issues 14
Online Hosting Solution 14
Version Control Collaboration 14
Technical Optimization 14

9. Testing and Quality Assurance 14
Player Testing 14
Issue Identification and Resolution 14
User Feedback 15
10. Progress 15
Player Engagement Metrics 15
Community Feedback 15
11. Future Development 15
Scoreboard Leaderboard 15
Difficulty Levels 15
Level Mode 15
Additional Features 15
Community Involvement 15
Roadmap and Updates 16
12. Conclusion 16

Project Report: Unity WebGL Snake Game

Abstract

The Unity WebGL Snake Game project aimed to introduce developers to web-based game development using Unity. In this report, we discuss the process of developing a classic Snake game with added features such as scoring and high scores, as well as the challenges faced, design choices, and the journey of making this game.

1. Introduction

Project Goals

The main goal of our project was the development of the Unity WebGL Snake Game. Through this endeavor, our aim was to gain a solid understanding of the intricacies involved in creating web-based games using the powerful Unity engine. Our primary focus was to design a game that would not only captivate players but could also be easily accessed and enjoyed through web browsers. By embarking on this project, we looked to set up a foundation for learning and innovation, encouraging experimentation and exploration within the realm of web-based game development.

Objectives

In pursuit of our goal, we set several goals:

- Develop a functional Snake game with classic gameplay.

- Implement scoring mechanics and high score tracking.
- Explore the Unity WebGL platform for web deployment.

2. History of the Game Snake

Origin and Evolution

To appreciate the significance of our Snake game project, it is essential to explore the history of the game Snake itself. Snake, in its simplest form, originated in the early 1970s as a classic arcade game known as "Blockade." In this game, players controlled a line that grew longer with each movement, aiming to outmaneuver their opponent.

The game Snake as we know it today gained widespread popularity in the late 1970s and early 1980s with the advent of personal computers and early mobile phones. Nokia's iconic 1997 mobile phone, the Nokia 6110, included a pre-installed Snake game that became a sensation. The game's premise was straightforward: players controlled a growing snake, guiding it to eat food items while avoiding collisions with walls and the snake's own tail.

The success of Snake paved the way for countless adaptations and variations across different platforms. The game's addictive nature and simplicity made it a staple on mobile devices, leading to its enduring legacy in the world of casual gaming.

Influence on the Gaming Industry

Snake's influence on the gaming industry extends beyond nostalgia. It played a crucial role in popularizing mobile gaming and the concept of "casual gaming." The game's mechanics, which are easy to grasp but challenging to master, set a precedent for many successful mobile games that followed.

Additionally, Snake's grid-based movement and emphasis on strategy and reflexes have inspired game developers in various genres. Concepts like score-based gameplay, power-ups, and leaderboards owe some of their roots to the timeless appeal of Snake.

By creating our Unity WebGL Snake Game, we pay tribute to this iconic title while adding modern twists and features to make it accessible to a new generation of players.

3. Game Overview

Snake Game Features

Our Snake game encompasses the following features:

- **Classic Gameplay**: Players control a snake to collect food items while avoiding collisions with walls and the snake's own body.
- **Scoring System**: Points are awarded for each food item collected, increasing the player's score.

- **High Score Tracking**: The game keeps track of the highest score achieved, motivating players to surpass their previous records.

Design Choices

In designing the game, we opted for a minimalist aesthetic with grid-based movement and colorful visuals. The decision to add scoring and high scores aimed to enhance player engagement.

Game Mechanics

Beyond the core features, our game introduced additional mechanics:

- **Grid-Based Movement**: The snake moves in a grid, allowing precise control.
- **Collision Handling**: The game detects collisions with walls, the snake's body, and food items, triggering proper actions.
- **Restart Mechanism**: When the snake collides with itself or obstacles, the game restarts, challenging players to improve their skills.

Technical Details

The game was developed in Unity, using C# scripts for gameplay mechanics. WebGL deployment ensured that players could access the game directly through web browsers, eliminating the need for other installations or downloads.

4.Technical Implementation

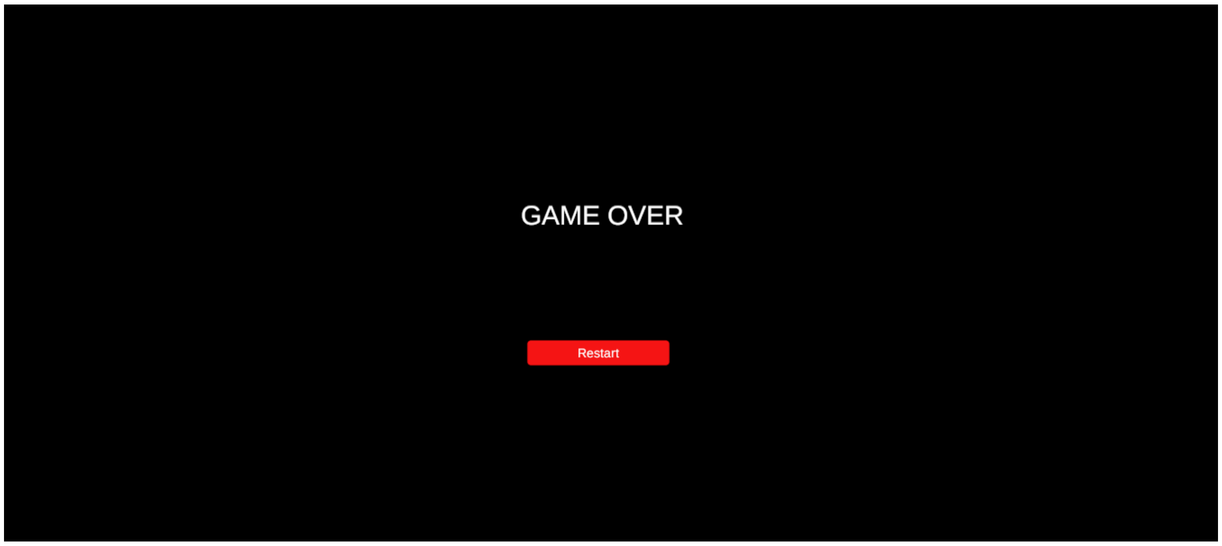
Code Overview

Our Unity WebGL Snake Game is powered by a well-structured and efficient codebase. Let us delve into the technical details of the game's implementation, which is essential for developers and enthusiasts who want to understand the underlying mechanics.

Game Over Screen (GameOverScreen.cs)

The Game Over Screen appears when the player's snake collides with obstacles or itself, signaling the end of the current game session. The screen provides options for restarting the game. Here's a breakdown of its functionality:

- **Set Up**: The **SetUp** method in the **GameOverScreen.cs** script ensures that the game over screen is displayed when needed. This method is called when the game ends.

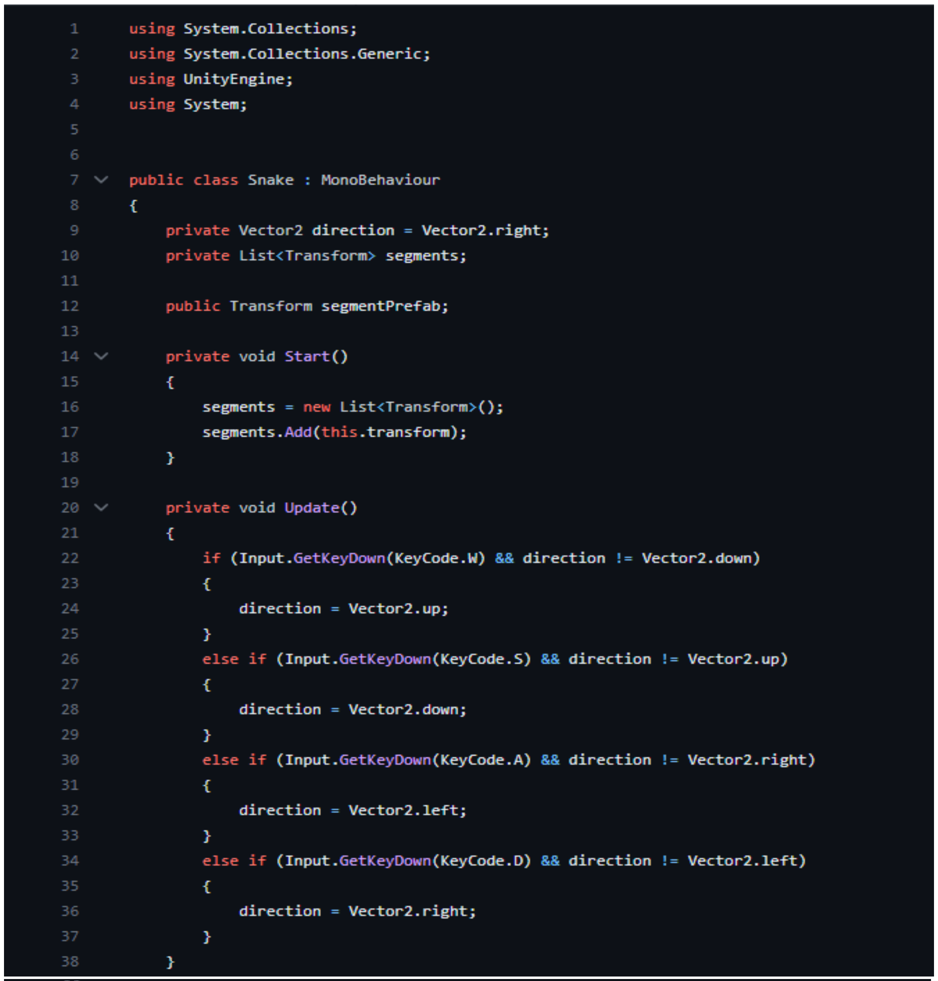


Restart Button: The "Restart" button allows players to quickly restart the game without returning to the start screen. It retrieves the current scene's index and reloads it, effectively resetting the game.

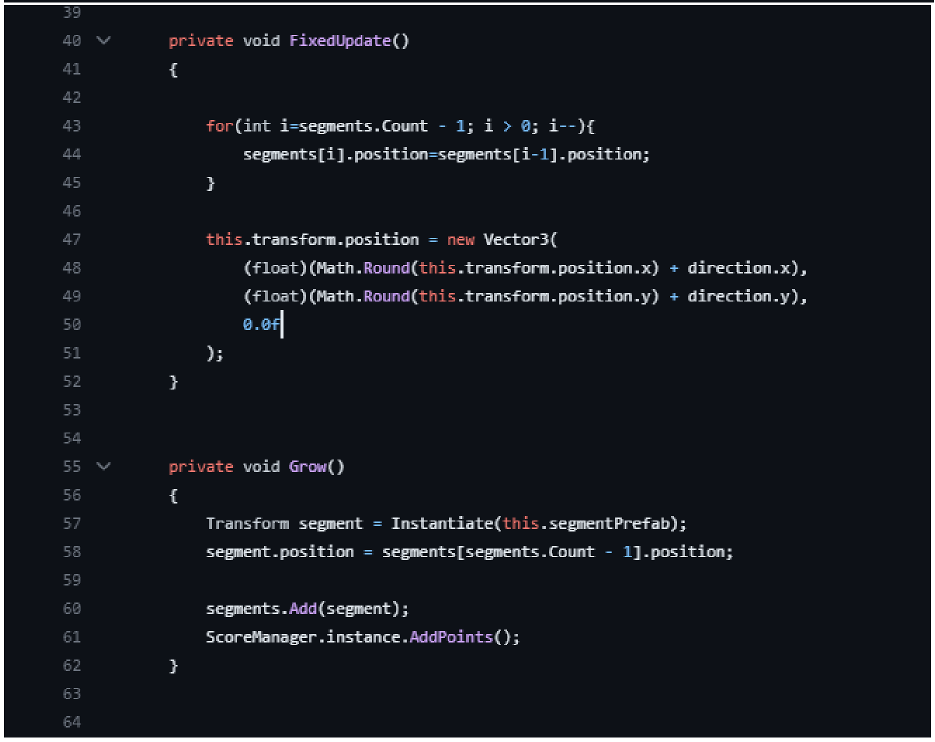
Snake Class

The **Snake** class is at the core of our game's functionality. It controls the behavior of the snake, its movement, growth, and collision detection.

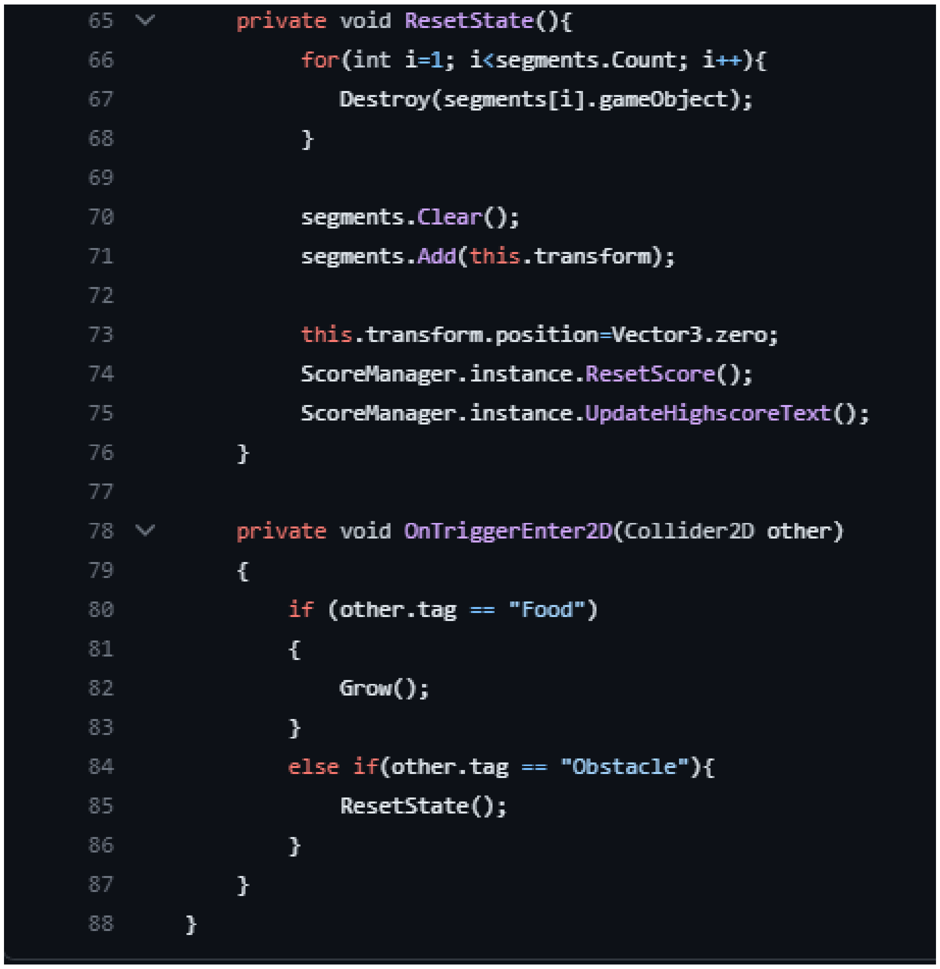
- **Direction Control**: The snake's movement is determined by player input using the **Update** method. It checks for key presses (W, A, S, D) and updates the **direction** vector accordingly. Importantly, it also prevents the snake from reversing its direction by checking the current direction.
- **Movement and Growth**: In the **FixedUpdate** method, the snake's position is updated based on the current direction. It employs a grid-based movement system, ensuring precise control. Additionally, the snake can grow when it collides with food items, which is handled in the **Grow** method.
- **Collision Handling**: The **OnTriggerEnter2D** method is responsible for detecting collisions. If the snake collides with food, it calls the **Grow** method to increase the snake's length. In case of a collision with obstacles (defined by the "Obstacle" tag), the **ResetState** method resets the game, clearing the snake and resetting the score.



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


public class Snake : MonoBehaviour
{
    private Vector2 direction = Vector2.right;
    private List<Transform> segments;

    public Transform segmentPrefab;

    private void Start()
    {
        segments = new List<Transform>();
        segments.Add(this.transform);
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.W) && direction != Vector2.down)
        {
            direction = Vector2.up;
        }
        else if (Input.GetKeyDown(KeyCode.S) && direction != Vector2.up)
        {
            direction = Vector2.down;
        }
        else if (Input.GetKeyDown(KeyCode.A) && direction != Vector2.right)
        {
            direction = Vector2.left;
        }
        else if (Input.GetKeyDown(KeyCode.D) && direction != Vector2.left)
        {
            direction = Vector2.right;
        }
    }
```



```csharp

    private void FixedUpdate()
    {

        for(int i=segments.Count - 1; i > 0; i--){
            segments[i].position=segments[i-1].position;
        }

        this.transform.position = new Vector3(
            (float)(Math.Round(this.transform.position.x) + direction.x),
            (float)(Math.Round(this.transform.position.y) + direction.y),
            0.0f
        );
    }


    private void Grow()
    {
        Transform segment = Instantiate(this.segmentPrefab);
        segment.position = segments[segments.Count - 1].position;

        segments.Add(segment);
        ScoreManager.instance.AddPoints();
    }


```

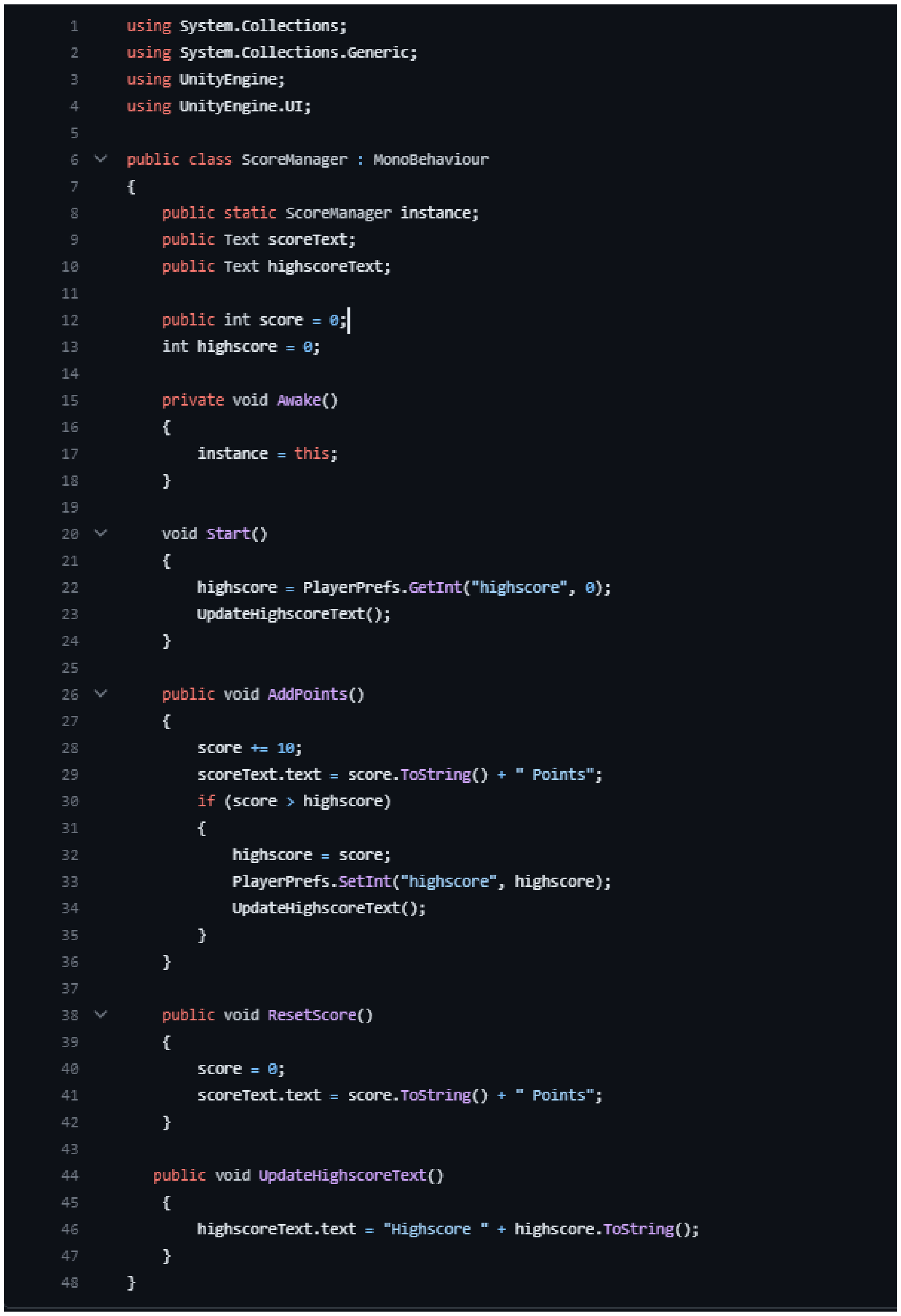


```
    private void ResetState(){
        for(int i=1; i<segments.Count; i++){
            Destroy(segments[i].gameObject);
        }

        segments.Clear();
        segments.Add(this.transform);

        this.transform.position=Vector3.zero;
        ScoreManager.instance.ResetScore();
        ScoreManager.instance.UpdateHighscoreText();
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Food")
        {
            Grow();
        }
        else if(other.tag == "Obstacle"){
            ResetState();
        }
    }
}
```

ScoreManager Class

The **ScoreManager** class is in charge of tracking and displaying the player's score and high score.

- **Scoring**: When the snake consumes food, the **AddPoints** method increments the player's score by 10 points. The score is displayed in the UI, providing immediate feedback to the player.
- **High Score Tracking**: The high score is also managed by the **ScoreManager**. It retrieves the player's highest score from PlayerPrefs and updates it when a new high score is achieved. This persistent high score encourages players to improve their performance.
- **UI Integration**: The **scoreText** and **highscoreText** Text objects in the Unity scene are linked to the **ScoreManager** to display the current score and high score.



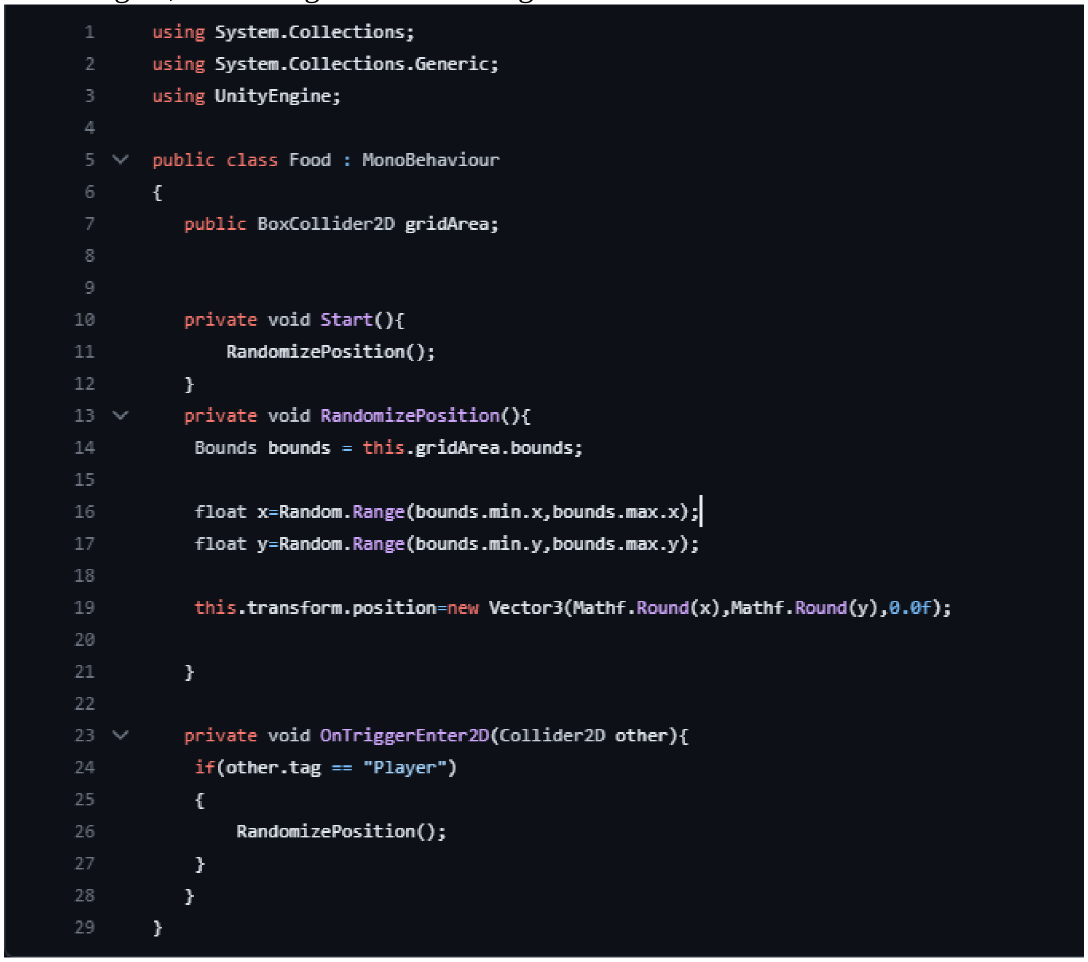
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ScoreManager : MonoBehaviour
{
    public static ScoreManager instance;
    public Text scoreText;
    public Text highscoreText;

    public int score = 0;
    int highscore = 0;

    private void Awake()
    {
        instance = this;
    }

    void Start()
    {
        highscore = PlayerPrefs.GetInt("highscore", 0);
        UpdateHighscoreText();
    }

    public void AddPoints()
    {
        score += 10;
        scoreText.text = score.ToString() + " Points";
        if (score > highscore)
        {
            highscore = score;
            PlayerPrefs.SetInt("highscore", highscore);
            UpdateHighscoreText();
        }
    }

    public void ResetScore()
    {
        score = 0;
        scoreText.text = score.ToString() + " Points";
    }

   public void UpdateHighscoreText()
    {
        highscoreText.text = "Highscore " + highscore.ToString();
    }
}
```

Food Class

The **Food** class handles the spawning and position of food items in the game.

- **Randomization**: When the game starts and when the snake consumes food, the **RandomizePosition** method generates random coordinates within the defined grid area (specified by the **gridArea** BoxCollider2D). This ensures that food appears at random locations, adding unpredictability to the game.

- **Collision Detection**: The **OnTriggerEnter2D** method in the **Food** class detects collisions with the player (the snake). Upon collision, the food's position is randomized again, simulating the snake eating the food.



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

public class Food : MonoBehaviour
{
   public BoxCollider2D gridArea;


   private void Start(){
       RandomizePosition();
   }
   private void RandomizePosition(){
    Bounds bounds = this.gridArea.bounds;

    float x=Random.Range(bounds.min.x,bounds.max.x);
    float y=Random.Range(bounds.min.y,bounds.max.y);

    this.transform.position=new Vector3(Mathf.Round(x),Mathf.Round(y),0.0f);

   }

   private void OnTriggerEnter2D(Collider2D other){
    if(other.tag == "Player")
    {
        RandomizePosition();
    }
   }
}
```

Enhanced Audio

To immerse players in the gameplay experience, we've introduced new audio elements:

- **Game Start Sound**: This sound effect accompanies the transition from the start screen to the game, signaling the beginning of the Snake Game.
- **Game Over Sound**: When the player's snake meets its demise, the "Game Over" sound provides audible feedback, enhancing the sense of accomplishment and challenge.
- **Swallow Sound**: As the snake consumes food items, the "Swallow" sound effect adds satisfaction to each successful food intake.

Collaborative Development

Our development process involved collaborative work among team members. Version control using Git facilitated smooth collaboration by allowing multiple team members to work on various aspects of the project simultaneously. This cooperative approach streamlined development and ensured that changes could be tracked and integrated efficiently.

5. WebGL

WebGL, an acronym for Web Graphics Library, represents a transformative technology that has redefined the way we experience interactive content on the web. It's a JavaScript API (Application Programming Interface) that brings hardware-accelerated 2D and 3D graphics rendering capabilities to web browsers without the need for plugins or external applications. WebGL leverages the power of modern GPUs (Graphics Processing Units) to deliver stunning visuals and rich interactivity, making it a cornerstone of web-based gaming, simulations, data visualizations, and more.

At its core, WebGL is an open standard maintained by the Khronos Group, a consortium of industry leaders in graphics and multimedia. This standardization ensures cross-browser compatibility and makes it accessible to a broad audience, regardless of their preferred web browser or device. WebGL's compatibility extends across various platforms, including Windows, macOS, Linux, and mobile devices, making it a versatile choice for developers aiming to reach a diverse user base.

One of the key advantages of WebGL is its ability to tap into the immense power of the user's graphics hardware. By running code directly on the GPU, WebGL applications can achieve impressive performance levels, even for computationally demanding tasks. This feature is particularly crucial for graphics-intensive applications like 3D games and simulations, where smooth rendering and responsive interactivity are essential.

WebGL doesn't operate in isolation but rather as part of a larger technology stack, which includes HTML5, CSS, and JavaScript. This integration allows developers to create cohesive web experiences that seamlessly blend interactive graphics with standard web content. Additionally, WebGL's compatibility with WebGL Shading Language (GLSL) provides developers with extensive control over the rendering pipeline, enabling them to craft intricate visual effects and sophisticated simulations.

In recent years, WebGL has gained significant traction in various industries beyond gaming. It's employed in scientific and medical visualizations, architectural and engineering simulations, and data-driven visualizations. Its potential for real-time 3D visualization of complex datasets, such as climate models or molecular structures, has made it an invaluable tool for researchers and educators.

WebGL's contribution to web-based gaming cannot be overstated. It has facilitated the development of browser games that rival traditional desktop or console titles in terms of graphics and gameplay. These games can be instantly accessed by players through a web browser, eliminating the need for downloads or installations. This accessibility has been a driving force behind the rise of HTML5 and WebGL as a viable platform for game developers, particularly indie developers looking to reach a broad audience.

In conclusion, WebGL represents a significant milestone in web technology. Its ability to harness the potential of modern GPUs for rendering interactive graphics has revolutionized web-based content, enabling everything from visually stunning games to critical scientific simulations. With its cross-platform compatibility and growing support, WebGL continues to be a transformative force in web development, promising even more exciting possibilities for the future.

6. Unity

Unity, often referred to as the "game developer's Swiss Army knife," is a comprehensive and versatile game development engine that has played an instrumental role in shaping the gaming industry. Born in 2005, Unity has evolved into a powerhouse that empowers developers to create immersive and engaging experiences for a wide range of platforms, including desktop, mobile, console, and web.

One of Unity's standout features is its cross-platform capabilities. Developers can write code once and deploy their games to multiple platforms without the need for extensive platform-specific modifications. This capability significantly reduces development time and allows creators to reach a broader audience. Whether you're creating a mobile game for iOS and Android or a desktop game for Windows and macOS, Unity simplifies the process.

Unity's user-friendly interface is another hallmark feature. It provides an intuitive environment for designing, prototyping, and building games. The Asset Store, a marketplace integrated directly into Unity, offers a vast selection of assets, tools, and plugins that can be easily imported into projects. This accessibility accelerates development and fosters a vibrant community of developers and creators.

At the heart of Unity's scripting capabilities is C#, a versatile and widely-used programming language. C# offers a robust framework for developing complex gameplay mechanics, artificial intelligence, and user interfaces. This flexibility enables developers to create games that range from simple 2D puzzles to sprawling 3D open worlds. Additionally, Unity's component-based architecture encourages modularity, making it easier to manage and scale projects.

Unity's rendering engine ensures that games built with the engine are visually impressive. It supports cutting-edge graphics features such as physically-based rendering (PBR), dynamic lighting, and post-processing effects. These capabilities enable developers to craft visually stunning games that captivate players with realistic environments and eye-catching visual effects.

In recent years, Unity has expanded beyond traditional gaming. It's increasingly used for a variety of applications, including architectural visualization, automotive simulation, training simulations, and virtual reality experiences. Its real-time 3D capabilities have made it a favorite among industries seeking to create interactive and immersive content.

One of Unity's most significant contributions to the gaming world is its democratization of game development. The engine has enabled countless indie developers and small studios to bring their creative visions to life. This has led to the proliferation of unique and innovative games that might not have been possible otherwise.

Unity's commitment to accessibility extends to the web through WebGL deployment. Developers can export their Unity projects to WebGL, making them accessible via standard web browsers. This feature eliminates the need for players to download or install games, making it easier than ever for developers to reach a global audience.

In conclusion, Unity has left an indelible mark on the world of game development and beyond. Its cross-platform capabilities, user-friendly interface, and powerful scripting make it a top choice for developers worldwide. As Unity continues to evolve and adapt to emerging technologies, it remains a driving force behind innovation in interactive experiences.

7. Development Process

Unity and WebGL

Our choice of Unity as the development platform provided several advantages. Unity's user-friendly interface, extensive documentation, and support for WebGL export made it an ideal choice for our project.

Development Phases

Our development process consisted of several phases, including:

- **Conceptualization**: We brainstormed ideas and decided to recreate the classic Snake game.
- **Prototyping**: We created a basic prototype to test core mechanics.
- **Iterative Development**: We continually refined the game based on feedback and testing.
- **Polishing**: We focused on enhancing visuals, user experience, and performance.

Hosting and Deployment

One of the key milestones in our development journey was hosting the game online for wider accessibility. We chose to host the game on itch.io, a popular platform for indie game developers. This decision allowed us to make the game easily playable for anyone with an internet connection and a web browser.

itch.io: The Online Hosting Solution

By hosting our Unity WebGL Snake Game on itch.io, we provided players with a convenient and user-friendly way to access the game. Players no longer needed to download or install the game;

they could simply visit our itch.io page and start playing immediately. This approach eliminated compatibility issues and allowed players to enjoy our game on various devices and platforms.

Version Control

To maintain project integrity and collaborate effectively, we implemented version control using Git. This allowed team members to work concurrently on the project, track changes, and resolve conflicts.

8. Challenges and Solutions

Compatibility Issues

During development, we faced compatibility challenges. The game ran seamlessly on the host computer but encountered issues on other systems. This was a critical problem to address to ensure accessibility and a wider player base.

Online Hosting Solution

To resolve compatibility issues, we decided to host the game online. This solution enabled players to access the game via web browsers, eliminating system-specific constraints.

Version Control Collaboration

Effective collaboration among team members was crucial. Implementing version control allowed us to work together seamlessly, with clear version history and conflict resolution processes in place.

Technical Optimization

Optimizing the game for web deployment was a priority. We reviewed and improved the game's performance to ensure smooth gameplay experiences for players across different devices and browsers.

9. Testing and Quality Assurance

Player Testing

While we did not follow a formal testing process, we conducted player testing ourselves. This approach helped us identify various gameplay issues and refine mechanics for an enjoyable player experience.

Issue Identification and Resolution

One significant issue we identified was related to the snake's behavior when it became larger, and the player changed direction by 180 degrees. The snake would collide with itself, causing the game to restart. This issue was promptly resolved through iterative development and rigorous testing.

User Feedback

Player feedback played a crucial role in improving the game. We encouraged players to provide input on gameplay, controls, and overall experience, which informed subsequent updates and refinements.

10. Progress

Player Engagement Metrics

While the game is still in its initial stages, we have been actively monitoring player engagement and progress. Initial metrics indicate promising signs of player interest and growing playtime.

Community Feedback

Player feedback has been invaluable in identifying areas for improvement. We encourage players to share their thoughts and suggestions as we continue to develop and refine the game.

11. Future Development

Scoreboard Leaderboard

One of our main objectives for future development is implementing a scoreboard leaderboard. This feature will allow players to compete for the highest scores and achievements, fostering a sense of competition and recognition within the player community.

Difficulty Levels

To cater to a broader audience and provide varied gameplay experiences, we plan to introduce difficulty levels. Players will have the option to choose from three modes: easy, medium, and hard. Each mode will adjust the speed and complexity of the game, providing challenges suitable for players of different skill levels.

Level Mode

We aim to expand the game's content by introducing a level-based mode. In this mode, players will face progressively challenging levels, each with unique obstacles and objectives. Completing a level will unlock the next, offering a sense of achievement and progression.

Additional Features

In addition to the mentioned features, we are considering other enhancements, such as power-ups, customizable skins for the snake, and special challenges to keep the gameplay fresh and engaging.

Community Involvement

Player feedback and community engagement remain integral to our development process. We will continue to encourage players to share their thoughts, ideas, and suggestions, which will play a vital role in shaping the future of the game.

Roadmap and Updates

We will maintain transparency with our player base by sharing a development roadmap outlining our planned updates and features. Regular updates will bring new content and improvements, ensuring that players have a reason to return to the game and experience new challenges.

12. Conclusion

In conclusion, our Unity WebGL Snake Game project provided valuable insights into web-based game development using Unity. We successfully created an engaging game, overcame compatibility challenges, and learned from player feedback. This project served as a steppingstone for future game development endeavors.