Unity Case Assignment

ROULETTE MINI GAME

CASE REPORT

Author:

Yağız AKALIN

June 27, 2024

Contents

1 Introduction

1.1 Project Overview

In the realm of the gaming industry, a "mini-game" is an additional gameplay experience that complements the main game, often referred to as a "game within a game." This project focused on creating a modular Roulette Game with one variation, designed to easily add a second variation with different animations and assets in the future. The main goal was to show proficiency in using the Unity Game Engine, simply called "Unity" here.

To help with the implementation, we used a reference video, "Barbeque-Party-Ref", along with the necessary assets. These provided a solid foundation for building a game similar to the Barbeque Party variation, ensuring that the game mechanics, animations, and overall user experience were of high quality and engaging for players.

1.2 Objectives

The project has several key objectives that we aim to achieve. By meeting these objectives, we are able to demonstrate not only our technical skills in Unity but also our ability to design engaging gameplay, solve problems creatively, and apply advanced game development concepts effectively.

1.2.1 Scene Implementation

First, we aim to create an initial scene with a start button that takes the player to the Barbeque Party scene, which is our actual game scene. We must also add a button in the actual game scene to spin the roulette, making the interface easy to use.

1.2.2 Abstract Roulette Mechanic

We aim to implement the Roulette Game mechanic in an abstract way, different from the polished example provided. This will allow us to showcase our problem-solving and design skills.

1.2.3 Reward Mechanism

We must develop a reward system using food icons that correspond to the roulette's stopping slice. This added an element of fun and surprise as players looked forward to the rewards.

1.2.4 Randomness Implementation

We try to ensure the roulette stopped on a random slice using robust algorithms to maintain fairness and excitement in the game.

1.2.5 Player's Wallet Update

We aim to implement a simple player wallet system that included only the provided food items. The wallet must be updated automatically when players won a reward, and it will be displayed clearly in the game.

1.2.6 Win Sign and Scene Reset

We must add a basic tick shaped win sign that appeared after each reward is collected. Once all slices are collected, the game reset to the initial scene. The game will be designed with an endless loop in mind and the player can continue playing the game until he clicks the close button on the top right.

2 Design and Architecture

In this Unity project, our aim is to create a user-friendly and visually appealing interface while ensuring a modular and extendable design. The project consists of two main scenes: the Start Menu and the Game Scene.

2.1 Start Menu Design

2.1.1 Canvas Creation

To start, I created a Canvas in Unity's Hierarchy to serve as the base for all UI elements.

2.1.2 Background Setup

I added a Panel to the Canvas and assigned a custom background image to it, providing an engaging backdrop for the start menu.

2.1.3 Start Button

I added a Button element to the Canvas for starting the game. I used Unity's SceneManagement to implement functionality that transitions to the main game scene when the button is clicked, allowing the user to begin playing.

2.2 Game Scene Design

2.2.1 UI Elements

Like the start menu, I created a Canvas and added a background image. Additionally, I included a title image labeled "Barbecue Party" using TextMesh Pro for clear text rendering.

2.2.2 Close Button

I added a button to close the game, linked to a script (MainMenuCloseButton) that handles the quit functionality.

2.2.3 Free Button

Another button, styled with a green button image and labeled "Free," was added to start the game.

2.2.4 Reward Boxes

The scene features 14 different reward boxes arranged around a roulette. These boxes are represented by GameObjects, each containing multiple UI elements (e.g., background colors, icons) to indicate the current state and reward. Since the color of the boxes will change every time the roulette mechanism rotates, I added the yellow, blue and dark gray boxes together with the rewards given in the assets and named them as BGYellow, BGBlue, BGGrey, Icon and Reward under each object, respectively. Here, I accepted the index value of the box at the top left as 0 because this box is considered as the starting point. Then, continuing from the box below, the index number was increased until the last box, assigning index values from 0 to 13 to all boxes in total.

2.3 Wallet Panel

2.3.1 Wallet Button

A button representing the player's wallet was added to the Canvas, linked to a script (Wallet.cs) that toggles the wallet panel's visibility.

2.3.2 Wallet Content

The wallet panel contains 14 slots, seven at the top and seven at the bottom, each corresponding to a possible reward. These slots are represented by GameObjects and were named Loc1, Loc2, etc., each holding an image of the respective reward. This design allows the player to easily see which rewards they have collected.

3 Core Mechanisms

The core gameplay mechanisms revolve around the roulette system and reward management, implemented with C# scripting and Unity's coroutine system for smooth animations and timed events.

3.1 Roulette Mechanism

3.1.1 Free Button Functionality

A script (Free.cs) manages the logic for the Free button, which starts the roulette spinning. The script uses lists to track the state of each reward box (e.g., yellow, blue, grey backgrounds, icons).

3.1.2 Coroutine and Randomness

The roulette spinning is handled using a coroutine (IEnumerator Play()), allowing for timed delays and smooth transitions. The first thing done in the IEnumerator Play() method is to check whether the roulette has reached the last round. If it reaches the last round, the game will be reset, otherwise it will continue to spin. Random numbers determine how many initial spins the roulette makes and which reward it stops at. To do this, a randomly generated number between 1 and 3 is assigned to a variable called randomNumber and the roulette is initially made to roll as many times as this random number. Afterwards, in order to randomly choose from 14 different rewards, a random number between 0 and 14 is created and this is assigned to the variable named randomRewards. Roulette continues to trace rewards until it reaches the reward, whose index number corresponds to this randomly generated value.

3.1.3 Reward Selection

The roulette checks if the selected reward has been previously chosen. If not, it adds the reward to the wallet, updating the corresponding slot in the wallet panel.

3.2 Wallet Management

3.2.1 Displaying Rewards

The wallet panel updates to show collected rewards by activating the relevant image in the appropriate slot (e.g., Loc1, Loc2, etc).

3.2.2 Interaction Handling

Scripts manage the interaction states of buttons, ensuring they are only clickable at the correct times.

3.3 Animations

The script controls the visual feedback of the roulette and reward boxes, including color changes and icon appearances. The coroutine ensures that these changes happen in a timed, visually appealing manner. The architecture and mechanisms ensure a clean and extendable project structure. By following these

design principles, I created a user-friendly interface and an engaging gameplay experience, demonstrating proficiency in Unity and efficient use of its features.

4 Challenges and Solutions

During the development of this Unity project, I encountered several challenges that required careful consideration and problem-solving. One of the initial hurdles was implementing smooth transitions between the start menu and the actual game scene. Leveraging Unity's SceneManagement capabilities, I managed to handle scene transitions efficiently. This involved writing scripts to manage loading screens and ensuring that the game state persisted correctly between scenes. Additionally, designing a fair and unbiased roulette mechanism that provides a truly random outcome was crucial. By incorporating Unity's Random.Range function to generate random numbers for both the initial spins and the final reward selection, and conducting extensive testing, I ensured a balanced gameplay experience.

Managing the timing and sequence of animations, particularly for the roulette spins and reward display, posed another challenge. I implemented coroutines using the IEnumerator interface, allowing for precise control over the timing of animations and delays, resulting in smooth transitions and a polished visual experience. Keeping track of the state of each reward box and ensuring the correct display of collected rewards in the wallet was also complex. I used lists to manage the state of each UI element, updating them dynamically based on the player's interactions, which required careful planning and debugging to ensure all states were accurately represented.

Ensuring that the UI elements were intuitive and responsive to user inputs was another critical aspect. By designing clear and easily recognizable buttons, using consistent visual cues and feedback mechanisms, and implementing interactive scripts to handle button states, I ensured that buttons were only clickable at appropriate times. These challenges, while demanding, ultimately enhanced my understanding of Unity and improved my problem-solving skills.

5 Conclusion

This Unity project provided a comprehensive opportunity to showcase and enhance my skills in game development, UI design, and scripting. Through the creation of a modular roulette game, I was able to delve deep into the intricacies of Unity, from scene management and UI layout to coroutine handling and randomness implementation. The process of overcoming various challenges not only strengthened my problem-solving abilities but also highlighted the importance of meticulous planning and testing in game development. By integrating user-friendly interfaces and engaging gameplay mechanics, I aimed to create a seamless and enjoyable experience for players.

In conclusion, this project underscored the significance of flexibility and adaptability in design, allowing for future expansions and variations with minimal rework. The lessons learned and skills acquired during this development journey will undoubtedly contribute to my future endeavors in the gaming industry.

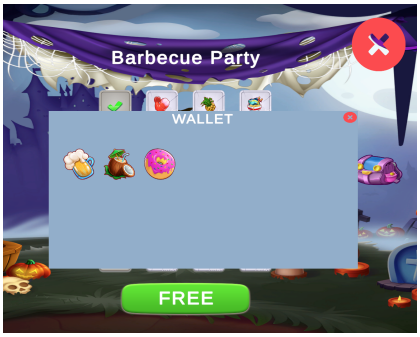
A Appendix

A.1 Start Menu and Screenshots of the Gameplay

(a) Start Menu

(b) Actual Game Scene

(c) Game Play Scene



(d) Wallet Panel

A.2 Code Snippets

A.2.1 StartMenu.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class StartMenu : MonoBehaviour
{
    public void OnStartButtonClicked()
    {
        SceneManager.LoadScene("SampleScene");
    }
}
```

A.2.2 MainMenuCloseButton.cs

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

public class MainMenuCloseButton : MonoBehaviour
{
    public void CloseGame()
    {
        Application.Quit();
    }

}
```

A.2.3 Wallet.cs

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

public class Wallet : MonoBehaviour
{
    [SerializeField]
    GameObject WalletPanel;

    public void walletOpen()
    {
        WalletPanel.SetActive(true);
    }
```

```
    public void walletClose()
    {
        WalletPanel.SetActive(false);
    }


}
```

A.2.4 Free.cs

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

public class Free : MonoBehaviour
{
    // A list to store 14 yellow boxes while the roulette
        mechanism is working
    [SerializeField]
    GameObject[] objectsYellow;

    // A list to store 14 blue boxes while the roulette
        mechanism chooses certain reward
    [SerializeField]
    GameObject[] objectsBlue;

    // A list to store 14 grey boxes while the roulette
        mechanism stops after choosing certain reward
    [SerializeField]
    GameObject[] objectsGrey;

    // A list to store tick shaped win sign
    [SerializeField]
    GameObject[] objectsIcon;

    // A list to store 14 different rewards, i.e. foods
    [SerializeField]
    GameObject[] objectsRewards;

    //A button object to indicate Free Button to start the
        roulette
    [SerializeField]
```

```
Button freeButton;

//A button to indicate our Wallet
[SerializeField]
Button walletButton;

//A list of 14 different locations which indicate each
    reward's location in our wallet
[SerializeField]
GameObject[] Loc1, Loc2, Loc3, Loc4, Loc5, Loc6, Loc7,
    Loc8, Loc9, Loc10, Loc11, Loc12, Loc13, Loc14;

//A variable to keep indexes of images. We determine which
    reward the relevant point corresponds to in our wallet
public List<int> numbers;

//A counter variable used to indicate which round of
    roulette is in
public int counter;

public IEnumerator Play()
{

    //The game resets when the roulette reaches the last
        round.
    if (numbers.Count == 14)
    {
        SceneManager.LoadScene(0);
    }
    else
    {
        freeButton.interactable = false; //Free Button is
            inactive while roulette keeps spinning
        walletButton.interactable = false; //Wallet Button
            is inactive while roulette keeps spinning

        //Random numbers from 1 to 3 are generated and the
            roulette is initially turned by this random
            number
        int randomNumber = Random.Range(1, 4);
        // print(randomNumber);
        Debug.Log(randomNumber);

        for (int i = 0; i < randomNumber; i++)
        {
            foreach (var item in objectsYellow)
```

```
    {
        item.SetActive(true);
        yield return new WaitForSeconds(.2f);
        item.SetActive(false);
    }
}

//A random reward will be chosen according to this
    random number.
int randomRewards = Random.Range(0, 14);

// print("RANDOM REWARDS = " + randomRewards);
Debug.Log("RANDOM REWARDS = " + randomRewards);

if (numbers.Contains(randomRewards))
{
    print("There is a same number");
    freeButton.interactable = true;
}
else
{
    counter++;
    print("Counter" + counter);
    numbers.Add(randomRewards);
    //In this switch case, the chosen reward will
        be sent to the wallet.
    switch (counter)
    {
        case 1:
            Loc1[randomRewards].SetActive(true);
            break;
        case 2:
            Loc2[randomRewards].SetActive(true);
            break;
        case 3:
            Loc3[randomRewards].SetActive(true);
            break;
        case 4:
            Loc4[randomRewards].SetActive(true);
            break;
        case 5:
            Loc5[randomRewards].SetActive(true);
            break;
        case 6:
            Loc6[randomRewards].SetActive(true);
            break;
```

```
        case 7:
            Loc7[randomRewards].SetActive(true);
            break;
        case 8:
            Loc8[randomRewards].SetActive(true);
            break;
        case 9:
            Loc9[randomRewards].SetActive(true);
            break;
        case 10:
            Loc10[randomRewards].SetActive(true);
            break;
        case 11:
            Loc11[randomRewards].SetActive(true);
            break;
        case 12:
            Loc12[randomRewards].SetActive(true);
            break;
        case 13:
            Loc13[randomRewards].SetActive(true);
            break;
        case 14:
            Loc14[randomRewards].SetActive(true);
            break;
    }
    //The roulette keeps spinning until we reach
        the index of the chosen reward
    for (int i = 0; i < randomRewards; i++)
    {
        objectsYellow[i].SetActive(true);
        yield return new WaitForSeconds(.2f);
        objectsYellow[i].SetActive(false);
    }
    //The yellow box glows twice after we reach
        the certain reward
    for (int i = 0; i < 2; i++)
    {
        objectsYellow[randomRewards].SetActive(true);
        yield return new WaitForSeconds(.2f);
        objectsYellow[randomRewards].SetActive(false);
        yield return new WaitForSeconds(.2f);
        objectsYellow[randomRewards].SetActive(true);
        yield return new WaitForSeconds(.2f);
        objectsYellow[randomRewards].SetActive(false);
        yield return new WaitForSeconds(.2f);
    }
```

```
                //The yellow box becomes blue, then the tick
                    sign appears.
                objectsBlue[randomRewards].SetActive(true);
                objectsIcon[randomRewards].SetActive(true);
                yield return new WaitForSeconds(2f);
                objectsBlue[randomRewards].SetActive(false);
                objectsRewards[randomRewards].SetActive(false);
                //Finally the box become dark grey and the
                    reward has been chosen
                objectsGrey[randomRewards].SetActive(true);
                freeButton.interactable = true;
                walletButton.interactable = true;
            }
        }

    }

    public void EnumPlay()
    {
        StartCoroutine(Play());
    }

}
```