

Game Engines Task 3

Vivid Jump



Graphical Assets

Backgrounds made with photoshop

Font (menu, paused menu and credits): LiberationSans SDF (TMP_FontAsset), normal

Font (Score): Arial,bold



With the backgrounds we wanted everything to be different from each other but that they would be consistent as well because of the style. We wanted that the backgrounds would give a theme to each and every section of the game.

The main colours for the game where mainly purples, blues and magenta. For the fonts we wanted that the main and credits where different from the main game to make it pop out more to people.

Scripts Overview

The Mainmenu script codes for the play button to go to the next level.

The OptionsMenu script codes the Quit button for the player to quit the game.

The CreditsMenu script loads the Menu screen when the player clicks on the main menu button.

The PauseMenu has the coding of which button the player has to press to activate the pause menu, to load the menu and to quit the game as well.

```
# PauseMenu

Assembly Information
Filename Assembly-CSharp.dll

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

public class PauseMenu : MonoBehaviour
{
    public static bool GamelsPaused = false;

    public GameObject pauseMenuUI;
    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GamelsPaused)
            {
                Resume();
            } else
            {
                Pause();
            }
        }
    }

    public void Resume ()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GamelsPaused = false;
    }

    void Pause ()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GamelsPaused = true;
    }

    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene("Menu");
    }

    public void QuitGame()
    {
        Debug.Log ("Quitting game...");
        Application.Quit();
    }
}
```

```
# OptionsMenu

Assembly Information
Filename Assembly-CSharp.dll

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

public class OptionsMenu : MonoBehaviour
{
    public void QuitGame ()
    {
        Debug.Log("Quit!");
        Application.Quit();
    }
}
```

```
# Mainmenu

Assembly Information
Filename Assembly-CSharp.dll

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

public class Mainmenu : MonoBehaviour
{
    public void PlayGame ()
    {
        SceneManager.LoadScene("Level 1");
    }
}
```

Movement: The keys (left arrow key and right arrow key) for the player to press for the platforms to spin along with the helix in order for the ball to descend down and continue the game.

PlatformEnd: The script for Level 2 which programs that when the player is on the last platform they go to the credits scene.

Passcheck: The trigger to increase the score by one each time it passes the platforms.

KillPlayer: If the player goes on an enemy platform which are the specific platforms that are coloured red the player will die.

Goal: This allows the player to move on to the next level when the player reaches the last platform.

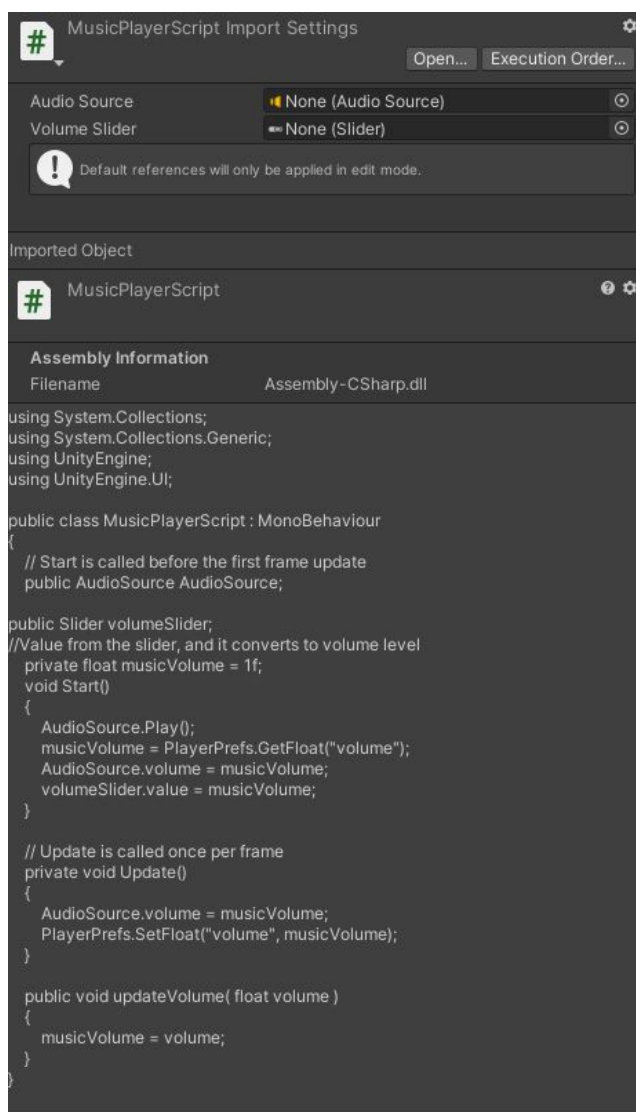
End: The script is for the end credits and it's the same as the OptionMenu script

BallControl: Was about a test on how the ball will jump and how it will jump with its rigidbody

CameraController: Programed with the camera to follow the player when the ball is moving.

GameManager: It manages how the score will work and how the next level will be called, this script was used as a reference in order for us instead of having to do a lot of scripts which could have made the game unorganized.

MusicPlayerScript: A script to make the volume increase and decrease using the slider which saves the setting on each different scene.



The screenshot shows the Unity Inspector for the MusicPlayerScript. The top section is the 'Import Settings' panel, which includes 'Audio Source' set to 'None (Audio Source)' and 'Volume Slider' set to 'None (Slider)'. Below this is the 'Imported Object' section, showing the script is attached to an object. The main section displays the script code, which includes using statements for System.Collections, System.Collections.Generic, UnityEngine, and UnityEngine.UI. The script defines a MusicPlayerScript class that inherits from MonoBehaviour. It has a public AudioSource field, a public Slider volumeSlider, and a private float musicVolume. The Start method calls AudioSource.Play() and sets the volume. The Update method calls AudioSource.volume = musicVolume and PlayerPrefs.SetFloat("volume", musicVolume). There is also an updateVolume method.

```
# MusicPlayerScript Import Settings
Open... Execution Order...

Audio Source: None (Audio Source)
Volume Slider: None (Slider)
! Default references will only be applied in edit mode.

Imported Object
# MusicPlayerScript

Assembly Information
Filename: Assembly-CSharp.dll

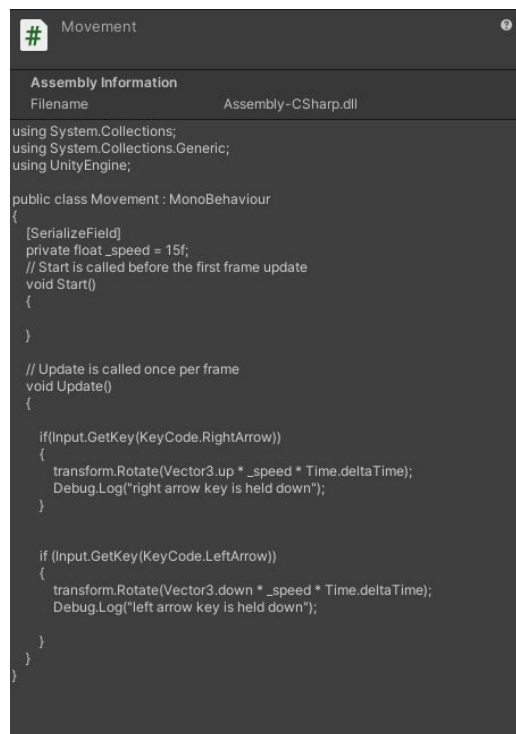
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class MusicPlayerScript : MonoBehaviour
{
    // Start is called before the first frame update
    public AudioSource AudioSource;

    public Slider volumeSlider;
    //Value from the slider, and it converts to volume level
    private float musicVolume = 1f;
    void Start()
    {
        AudioSource.Play();
        musicVolume = PlayerPrefs.GetFloat("volume");
        AudioSource.volume = musicVolume;
        volumeSlider.value = musicVolume;
    }

    // Update is called once per frame
    private void Update()
    {
        AudioSource.volume = musicVolume;
        PlayerPrefs.SetFloat("volume", musicVolume);
    }

    public void updateVolume( float volume )
    {
        musicVolume = volume;
    }
}
```



The screenshot shows the Unity Inspector for the Movement script. The top section is the 'Assembly Information' panel, showing the filename as 'Assembly-CSharp.dll'. The main section displays the script code, which includes using statements for System.Collections, System.Collections.Generic, and UnityEngine. The script defines a Movement class that inherits from MonoBehaviour. It has a private float _speed set to 15f. The Start method is called before the first frame update. The Update method is called once per frame. It checks for the RightArrow key and rotates the transform around the y-axis. It also checks for the LeftArrow key and rotates the transform around the x-axis.

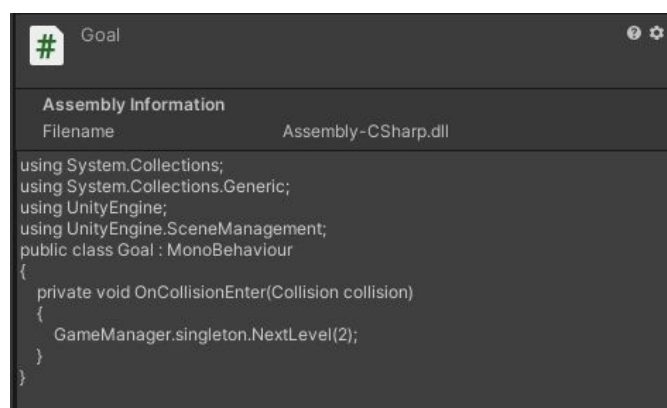
```
# Movement
Assembly Information
Filename: Assembly-CSharp.dll

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

public class Movement : MonoBehaviour
{
    [SerializeField]
    private float _speed = 15f;
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        if(Input.GetKey(KeyCode.RightArrow))
        {
            transform.Rotate(Vector3.up * _speed * Time.deltaTime);
            Debug.Log("right arrow key is held down");
        }

        if (Input.GetKey(KeyCode.LeftArrow))
        {
            transform.Rotate(Vector3.down * _speed * Time.deltaTime);
            Debug.Log("left arrow key is held down");
        }
    }
}
```



The screenshot shows the Unity Inspector for the Goal script. The top section is the 'Assembly Information' panel, showing the filename as 'Assembly-CSharp.dll'. The main section displays the script code, which includes using statements for System.Collections, System.Collections.Generic, UnityEngine, and UnityEngine.SceneManagement. The script defines a Goal class that inherits from MonoBehaviour. It has a private void OnCollisionEnter(Collision collision) method that calls GameManager.singleton.NextLevel(2).

```
# Goal
Assembly Information
Filename: Assembly-CSharp.dll

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

public class Goal : MonoBehaviour
{
    private void OnCollisionEnter(Collision collision)
    {
        GameManager.singleton.NextLevel(2);
    }
}
```

Process

The game started by us first doing research and seeing what we might need, this started by us designing on how the game would be. From there we started by seeing how we can make the shapes and the basic code necessities in order to start the game. After making the helix and the cubes where the ball would bounce and a basic rigid body for testing, we started adding basic code which made the test build work.

After that we got to work on more research on how we can make it better and more efficient, we first redesigned the entire layout of the first level and quickly got into what we were gonna add. First we arranged the shapes, prefabs and placed them into folders which made our work more simpler and organized. After that we got to work on the layout and after that we made the coding for the controls, camera and score which were improved even for the ball instead of having it bounce higher and higher it would just bounce on the same level.

With all this done our process would be easier since all we would have to do is just copy and retouch some things and add extra features that would make the game look better. We got to work on level 2 which wasn't hard since it's the same layout with just a different setup and backgrounds from the first level. Then we added the main menu which was already designed and planned, all we had to do was to make the buttons work. After all the scenes were ready we made sure that the menu, pause, options and credits worked.

The final touches were adding the music which wasn't hard to find but we wanted the music to match the theme of the game. After that we made code for the volume slider which allows the player to control the music and last but not least we organized everything on level 2.

When making all the backgrounds they were all planned from the beginning and were placed mid-way when making the game. We did all the background on Photoshop and they were easy to add on Unity even when designing for the scenes.

Testing

| Functionality | Expected Output | Actual Output | Status |
|---|---|--|--------|
| When player wins the game the credits pop up | When player goes on the final platform in level 2 the credits pop up | When the player is off the platforms in level 2 the credits pop up | |
| When the player passes the platforms they get a point | When the player goes between the platform while falling the player gets one point | The player can get more points if they go between the platforms and quickly go on the same platform to stay in the same section | |
| If the player presses the left or right the helix moves | The helix moves left and right | The helix moves left and right | |
| The music plays in it's own scene. | The music would change when the player goes to another scene of the game | The music works how it's supposed to however when the player dies or goes to the next level it makes the game a bit slow to load because of it | |
| If the player drops to the last platform he will go to the next level | The player will be sent to the next level | The player moves to the next level after beating the first level | |