Game Programming

Lecture IX
User Interface

Izzet Fatih Senturk

Score Test UI Setup

- Get a certain amount of points when you kill an enemy
- Display the points you have on the screen

- Create a UI Canvas and Text
  - Right click the Hierarchy window select UI -> Text
  - UI Canvas is automatically added and the Text becomes the child of the Canvas
  - Event system is automatically added
- Canvas is the white element on the screen and it hosts UI objects
- Event system is used to interact with the UI

Score Test UI Setup

- Change the name of the Text object: “Score_text”
- Change the color to white
- Change the font size to 20
- Change the text to “Score: 0”
- Position the UI element to the top right corner
  - Set x = -100, y =-50
  - Width = 100, Height = 30
  - For proper scaling use Rect Transform component and select top right anchor preset
- Scale the size of the object with the screen size
  - Select Canvas. On the Canvas Scaler component change Scale Mode from Constant Pixel Size to Scale with Screen Size

Score Implementation

- Create a new C# script named UIManager_sc
- Add the new script to the Canvas object as a component
- Create a private variable for the Player to keep track of the score

```
[SerializeField]
0 references
private int _score = 0;
```

- We need a public method to increase score in the Player script
  - Increase the score by the given points

```
public void AddScore(int points)
{
    _score += points;
}
```

Score Implementation

- When do we increase score?
  - We check if the Enemy is hit by Laser in the Enemy script
- Enemy script must be able to communicate with the Player script
  - Create a private handler variable for the Player script
  - Find the Player component first and then get the corresponding script component of the Player during start

```
private Player_sc _player_sc;

0 references
void Start()
{
    _player_sc = GameObject.Find("Player").GetComponent<Player_sc>();
}
```

Score Implementation

- Now we are ready to increase the score
  - First check we could obtain the Player script component successfully
  - Increase the Player score before destroying the Enemy object

```
else if (other.tag == "Laser")
{
    Destroy(other.gameObject);

    if (_player_sc != null)
    {
        _player_sc.AddScore(10);
    }

    Destroy(this.gameObject);
}
```

Score Implementation

- Update score on the screen
  - Player script should be able to access UI Manager script
  - Create a private handler variable for the UI Manager script in the Player script

```
private UIManager_sc _uiManager_sc;
```

- During start, find the UI Manager script component and check if null

```
void Start()
{
    transform.position = new Vector3(-2, 0, 0);
    _spawnManager_sc = GameObject.Find("Spawn_Manager").GetComponent<SpawnManager_sc>();
    _uiManager_sc = GameObject.Find("Canvas").GetComponent<UIManager_sc>();

    if (_spawnManager_sc == null)
    {
        Debug.LogError("The Spawn Manager is NULL");
    }

    if (_uiManager_sc == null)
    {
        Debug.LogError("UI Manager is NULL");
    }
}
```

Score Implementation

- Update AddScore function
  - UI Manager handler calls the UpdateScore function to update the score display on the screen
  - Currently, there is no UpdateScore function in the UI Manager script. We will implement it next

```
public void AddScore(int points)
{
    _score += points;
    _uiManager_sc.UpdateScore(_score);
}
```

Score Implementation

- Update UI Manager script
- Access Text UI element on the Canvas
  - First, we need to add UnityEngine.UI library to access available UI elements
  - Create a private Text handler variable and save
  - In the Hierarchy, drag and drop the Score_text to the Score Text field of the UI_Manager script

```
using UnityEngine.UI;

0 references
public class UIManager_sc : MonoBehaviour
{
    [SerializeField]
    0 references
    private Text _scoreText;
```

Score Implementation

- During start, set the initial score value on the screen
- Define the public UpdateScore function so that Player script can update the score display when needed

```
void Start()
{
    _scoreText.text = "Score: " + 0;
}

1 reference
public void UpdateScore(int playerScore)
{
    _scoreText.text = "Score: " + playerScore;
}
```

Lives Display

- Start the game by default with three lives
- Display the corresponding lives sprite on the screen
- Add an Image component to display lives sprites
  - Right click on the Canvas and create an Image component
  - Rename the Image component as "Lives_Display"
  - Set the Source Image: "Three" lives sprite as default
  - Set Preserve Aspect on
  - Anchor top left and set positions x = 90 and y = -40
- How to change the lives sprite in an efficient way during the game?

Lives Display

- Changing the lives sprite during the game
  - Define an array for lives sprites in the UI Manager script

```
[SerializeField]
0 references
private Sprite[] _liveSprites;
```

- Go to Canvas on the Hierarchy and set the size of the Live Sprites as 4
  - Set elements accordingly: element 3 refers to 3 lives, etc.

- Define an Image variable to access the Image component displayed on the screen
  - Set the Image from the Inspector Window

```
[SerializeField]
0 references
private Image _livesImg;
```

Lives Display

- Define a function to update the lives sprite on the screen
  - Get current lives as the argument
  - Use the argument as the index of the lives sprite to be selected

```
public void UpdateLives(int currentLives)
{
    _livesImg.sprite = _liveSprites[currentLives];
}
```

- When do we call UpdateLives function?
  - When the lives is updated in the Player script

```
_lives--;
_uiManager_sc.UpdateLives(_lives);
```

Game Over Text

- Display a "Game Over" text when the player runs out of lives
- Create a Text element under Canvas
  - Right click Canvas and select Text
  - Rename it as "Game_Over_text"
  - Rename the actual message as "GAME OVER"
  - Set the color white
  - Set font size to 50
  - Set horizontal and vertical overflows to overflow
  - Change the location to the center of the screen
- By default, "Game Over" text will be turned off
  - Disable object from the Inspector
  - It should be turned on when the Player is out of lives

Game Over Text

- Define a private variable to reference the "Game Over" text in the UI Manager script
  - Set the value of the variable in the Inspector Window
- Make sure that the text game object is disabled at start

```
private Text _gameOverText;

0 references
void Start()
{
    _scoreText.text = "Score: " + 0;
    _gameOverText.gameObject.SetActive(false);
}
```

Game Over Text

- Activate the game object when the player is out of lives
  - Check current lives value in the UpdateLives function
  - When the current lives is less than 1, set the text game object active

```
public void UpdateLives(int currentLives)
{
    _livesImg.sprite = _liveSprites[currentLives];

    if (currentLives == 0)
    {
        _gameOverText.gameObject.SetActive(true);
    }
}
```

Game Over Text Flicker

- Define a coroutine in the UI Mamanger script to turn on/off the Game Over text object (or just update the text content)
  - Set text as Game Over
  - Wait for half a second
  - Set the text nothing
  - Wait for half a second

```
IEnumerator GameOverFlickerRoutine()
{
    while (true)
    {
        _gameOverText.text = "GAME OVER";
        yield return new WaitForSeconds(0.5f);
        _gameOverText.text = "";
        yield return new WaitForSeconds(0.5f);
    }
}
```

Game Over Text Flicker

- Call the flicker coroutine function when the game is over

```
public void UpdateLives(int currentLives)
{
    _livesImg.sprite = _liveSprites[currentLives];

    if (currentLives == 0)
    {
        _gameOverText.gameObject.SetActive(true);
        StartCoroutine(GameOverFlickerRoutine());
    }
}
```

R Key to Restart Level

- Create a new Text object under Canvas to give instructions on how restart the game
  - Create a Text object under Canvas and rename it “Restart_text”
  - Set the text as “Press the 'R' key to restart the level”
  - Set the color white
  - Set font size 28
  - Set vertical overflow: Overflow
  - Stretch the text so that it fits one line and then center it
- Disable the text game object so that it starts in the off position
  - We will enable it when the player is out of lives

R Key to Restart Level

- Define a private variable to store the text object
  - Serialize the variable
- Set the variable value of Canvas from the Inspector window

```
[SerializeField]
0 references
private Text _restartText;
```

R Key to Restart Level

- Enable restart text object when the game is over
  - Set active true
- Define a new function for the Game Over sequence

```
public void UpdateLives(int currentLives)
{
    _livesImg.sprite = _liveSprites[currentLives];

    if (currentLives == 0)
    {
        GameOverSequence();
    }
}

1 reference
void GameOverSequence()
{
    _gameOverText.gameObject.SetActive(true);
    _restartText.gameObject.SetActive(true);
    StartCoroutine(GameOverFlickerRoutine());
}
```

R Key to Restart Level

- In which script we should accept the 'R' key to restart the level?
- Typically, a Game Manager script is used to keep the current status of the game
- Create an empty object and rename it as "Game_Manager"
- Create a new C# script named "GameManager_sc"
  - Add the new script to the "Game_Manager" as a component
- Define a Boolean variable in the Game Manager script to check if the game is over or not
- Define a public function to set the Boolean variable true

```
[SerializeField]
0 references
private bool _isGameOver;
```

```
public void GameOver()
{
    _isGameOver = true;
}
```

R Key to Restart Level

- Define a handler variable in the UIManager to call the GameOver function of the GameManager
  - Initialize it during start
- Call GameOver function of the Game Manager script in the Game Over sequence

```
private GameManager_sc _gameManager_sc;

0 references
void Start()
{
    _scoreText.text = "Score: " + 0;
    _gameOverText.gameObject.SetActive(false);
    _gameManager_sc = GameObject.Find("Game_Manager").GetComponent<GameManager_sc>();
}
```

```
void GameOverSequence()
{
    _gameManager_sc.GameOver();
    _gameOverText.gameObject.SetActive(true);
    _restartText.gameObject.SetActive(true);
    StartCoroutine(GameOverFlickerRoutine());
}
```

R Key to Restart Level

- Include SceneManagement library to be able to access Scene management functions

```
using UnityEngine.SceneManagement;
```

- Check key input and restart the scene if conditions are met
  - Scene can be loaded by index (faster) or name

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.R) && _isGameOver == true)
    {
        SceneManager.LoadScene("Game");
    }
}
```

- We will get an error when we run the game. We need to add scenes to build
  - File -> Build Settings -> Add Open Scenes

Creating the Main Menu

- Create a new Scene
  - File -> New Scene
  - Save it as "Main_Menu" under Assets/Scenes
- Set background
  - Create UI->Image and save as "Title_Screen_img"
  - Set the source image as "MainMenu" sprite from Assets/UI
  - Set x and y positions as 0
  - Preserve the Aspect Ratio
  - Change height and width as 512
- Darken the background
  - Select the Main Camera
  - Set Background color Black
  - Change Clear Flags to Solid Color
- Add the nebula background
  - Pick "SpaceBG_Overlay" and drag it to the scene
  - Focus on the object and then stretch it to the sides so that it fits to the screen

Creating the Main Menu

- Create a new button
  - Right click Canvas -> UI -> Button
  - Rename it as "New_Game_button"
  - Move it to right side of the screen
  - Change the text as "New Game"
  - Change Highlited Color to light blue
- Add a new script to the Canvas
  - Create a new folder under Scripts named "Main_Menu"
  - Create a new C# script named "MainMenu_sc" in the new folder
  - Add the new script to the Canvas
- Change canvas scaler
  - Select Canvas and change Canvas Scaler to Scale with Screen Size
- Anchor the button to the top right screen

Creating the Main Menu

- Add the new scene to the build settings
  - Make sure that the Main Menu is the first scene
  - Open build settings and delete the Game scene
  - Add the Main Menu by clicking Add Open Scenes
  - Drag the Game scene to the build settings next
  - Also update the Game Manager script if index is used for the scene
- In the Main Menu script, include SceneManagement library and define a public function to load the Game scene

```
using UnityEngine.SceneManagement;

0 references
public class MainMenu_sc : MonoBehaviour
{
    0 references
    public void LoadGame()
    {
        SceneManager.LoadScene(1);
    }
}
```

Creating the Main Menu

- How to call the LoadGame function in the MainMenu script?
  - Select New Game button and click + in “On Click”
  - Drag the Canvas to the click event and select MainMenuc_sc -> LoadGame function

```
using UnityEngine.SceneManagement;

0 references
public class MainMenu_sc : MonoBehaviour
{
    0 references
    public void LoadGame()
    {
        SceneManager.LoadScene(1);
    }
}
```

Creating the Main Menu

- Create a credits text
  - Right click Canvas and select UI -> Text
  - Rename it as Credits_text
  - Change the text field as you wish
  - Adjust the color
  - Position it to the center