Game Programming

Lecture X
Effects (FX)

Izzet Fatih Senturk

Enemy Explosion Setup

- Play the enemy explosion animation when the enemy is destroyed
- First, create the animation
  - Open Enemy prefab
  - When the Enemy is selected, go to animation window (if it is not open, Window -> Animation -> Animation)
  - Click create and save as Enemy_Destroyed_anim under Animations folder
  - Enable recording
  - Select all sprites under Enemy_Explodes_Sequence and drag to the animation window
  - Disable recording and play the animation for testing
  - Go back to the Scene
  - Select Enemy_Destroyed_anim and unselect Loop Time
- Play the game for testing
  - Enemy destroy animation plays by default

Enemy Explosion Setup

- Open the Enemy_Destroyed_anim animation controller
  - Named as Enemy, rename it as Enemy_Destroyed
  - Open Enemy_Destroyed
  - Orange color is the default state and automatically played
  - Tell Unity to not to play the destroy animation until its time
- Right click and create an empty state
  - Select the new state and rename it as Empty
  - Right click the Empty state and make it the default state
- Make a transition between Empty and Enemy_Destroyed_anim
  - Right click Empty and click Transition and then select target
  - Open Parameters on the left, add Trigger, rename it OnEnemyDeath
  - Select the transition and go to settings, add a new Condition
- If the trigger is activated, the logic will transition from Empty to Destroy animation
  - Test it while playing

Enemy Explosion Implementation

- Open Enemy script
- Create a handler to the Animator component

```
private Animator _anim;
```

- In start, assign the component

```
void Start()
{
    _player_sc = GameObject.Find("Player").GetComponent<Player_sc>();
    if (_player_sc == null)
    {
        Debug.LogError("The Player is NULL");
    }
    _anim = GetComponent<Animator>();
    if (_anim == null)
    {
        Debug.LogError("The Animator is NULL");
    }
}
```

Enemy Explosion Implementation

- Call the setTrigger method when the Enemy is destroyed

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Player")
    {
        //Damage the player
        Player_sc player = other.transform.GetComponent<Player_sc>();
        if (player != null)
        {
            player.Damage();
        }
        _anim.SetTrigger("OnEnemyDeath");
        Destroy(this.gameObject);
    }
    else if (other.tag == "Laser")
    {
        Destroy(other.gameObject);

        if (_player_sc != null)
        {
            _player_sc.AddScore(10);
        }
        _anim.SetTrigger("OnEnemyDeath");
        Destroy(this.gameObject);
    }
```

Enemy Explosion Implementation

- Test the game
  - Animation doesn't play
  - We destroy the object too soon before we can run the animation
- Add delay before destroying the Enemy object
  - Check the animation length (~ 2.3 seconds)
  - Add sufficient delay to Destroy method

```
_anim.SetTrigger("OnEnemyDeath");
Destroy(this.gameObject, 2.8f);
```

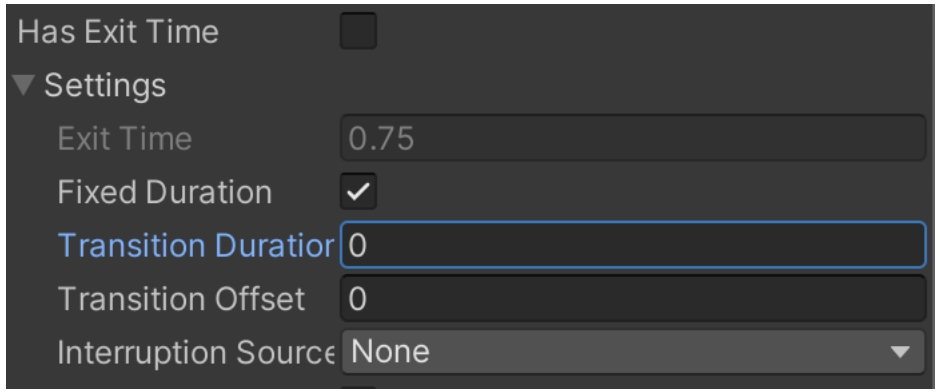
- A bug in the game
  - After we destroy the Enemy, we can still get hurt during 2.8 seconds!
  - How to fix the bug?

Enemy Explosion Implementation

- Make the Enemy stop to fix the bug
  - Set Enemy speed o so that it will not move further
  - If the Player moves towards the explosion, the damage will make sense

```
_anim.SetTrigger("OnEnemyDeath");
_speed = 0;
Destroy(this.gameObject, 2.8f);
```

- There is a delay between we shot the Enemy and the animation starts
  - Turn off the Has Exit Time for the transition between Empty and Destroy states
  - Also set Transition Duration to o in the settings



Asteroid Behaviour

- Setup the asteroid
  - Drag the asteroid sprite to the Hierarchy Window
  - Position it above the Player
  - Set sorting layer: Foreground
  - Add CircleCollider2D and adjust its size and set isTrigger true
  - Add RigidBody2D and set GravityScale 0
  - Add a new C# script named Asteroid_sc
    - Add the new script to the Asteroid game object
    - In the script, we will rotate the game object from the Z axis continously
  - Make the asteroid game object a prefab

Asteroid Behaviour

- Asteroid script
  - Rotate the game object with the given rotation speed continously

```
[SerializeField]
1 reference
private float _rorateSpeed = 20.0f;

// Update is called once per frame
0 references
void Update()
{
    transform.Rotate(Vector3.forward * _rorateSpeed * Time.deltaTime);
}
```

Asteroid Explosion Animation

- Create the game object
  - We will use sprites in the Explosion folder under Sprites
  - Drag the first sprite from Explosion folder to the Hierarchy Window
  - Rename it as Explosion
- Animate the explosion
  - Select the Explosion object and open Animation Window
  - Click the Create button and save it as Explosion_anim under Animations folder
  - Enable recording mode, drag the sprites to the Animation Window and disable recording mode
  - Set Loop Time false for the animation
- Make the Explosion a prefab
  - Drag the Explosion object to the Prefabs folder
- Remove the Explosion game object from the Hierarchy Window

Asteroid Explosion Script

- In the Asteroid script
  - Check for the collision with the Laser
  - Instantiate Explosion object from prefab at the position of the asteroid
  - Destroy the Explosion object as well as the Asteroid
- Create a private variable for the explosion prefab to be instantiated later
  - Set the prefab value from the inspector window

```
[SerializeField]
0 references
private GameObject _explosionPrefab;
```

- If the Asteroid is collided with the laser
  - Instantiate Explosion object
  - Destroy the laser first and then the Asteroid

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Laser")
    {
        Instantiate(_explosionPrefab, transform.position, Quaternion.identity);
        Destroy(other.gameObject);
        Destroy(this.gameObject);
    }
}
```

Explosion Script

- Explosion (clone) object remains in the Hierarch Window after Asteroid is exploded
  - Create a new behavior script for the Explosion
- Create a new C# script named Explosion_sc
  - Attach the new script to the Explosion prefab
- The main idea of the script is destroying the game object after a certain amount of time when the game object is initialized
  - Wait for 3 seconds in the start and then destroy the game object

```
public class Explosion_sc : MonoBehaviour
{
    // Start is called before the first frame update
    0 references
    void Start()
    {
        Destroy(this.gameObject, 3f);
    }
}
```

Contolling the Spawn through the Asteroid

- Start spawning after the asteroid is destroyed
  - Asteroid script should communicate with the Spawn Manager
- Define a public method in the SpawnManager to start spawning

```
public void StartSpawning()
{
    StartCoroutine(SpawnEnemyRoutine());
    StartCoroutine(SpawnPowerupRoutine());
}
```

- Add some delay before spawning enemies and powerups

```
IEnumerator SpawnEnemyRoutine()
{
    yield return new WaitForSeconds(3.0f);
    while (_stopSpawning == false)
```

```
IEnumerator SpawnPowerupRoutine()
{
    yield return new WaitForSeconds(3.0f);
    while (_stopSpawning == false)
```

Contolling the Spawn through the Asteroid

- In the Asteroid script, define a private variable as the handler of the SpawnManager
  - Find the SpawnManager component during start

```
private SpawnManager_sc _spawnManager_sc;

0 references
private void Start()
{
    _spawnManager_sc = GameObject.Find("Spawn_Manager").GetComponent<SpawnManager_sc>();
    if (_spawnManager_sc == null)
    {
        Debug.LogError("Spawn Manager is NULL");
    }
}
```

- Update OnTriggerEnter2D and call start spawning method

```
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Laser")
    {
        Instantiate(_explosionPrefab, transform.p
        Destroy(other.gameObject);
        _spawnManager_sc.StartSpawning();
        Destroy(this.gameObject, 0.25f);
    }
```

Player Thrusters

- Add thruster to the Player object
  - Find the first thruster sprite in the Thruster folder and drag it to the Hierarchy Window
  - Rename it as Thruster
  - Set scale to 1,1,1
  - Set sorting layer: Foreground
- The Thruster object should move with the Player object
  - Make Thruster as the child of the Player
  - Position Thruster at the back of the Player accordingly
- Animate Thruster
  - Select Thruster and open Animation Window
  - Click Create and save it as Thruster_anim under Animations folder
  - Enable recording mode, drag and drop Thruster sprites to the Animation window and then disable recording mode

Player Damage Visualization

- Create the Player hurt object from sprite
  - Select the first fire sprite from Player Hurt folder and drag&drop to the Hierarchy Window
  - Rename it as Right_Engine
  - Make it a child of of the Player
  - Set scale 1,1,1
  - Position it to the right wing of the Player
  - Set the sorting layer: Foreground
  - Set order in layer: 1
- Duplicate the object
  - Rename it as Left_Engine
  - Position it to the left wing of the Player
- Turn off both objects initially
  - Enable them one by one when the Player is damaged

Player Damage Visualization

- Create two private variables as handlers of the right and left engine fires
  - Set them accordingly from the Inspector Window

```
[SerializeField]
0 references | 0 references
private GameObject _rightEngine, _leftEngine;
```

- Update Damage method
  - Check current lives and enable right and left engines accordingly

```
_lives--;

if (_lives == 2)
{
    _leftEngine.SetActive(true);
}
else if (_lives == 1)
{
    _rightEngine.SetActive(true);
}
```

Player Damage Visualization

- Animate the engine fires
  - Select the right engine and open Animation Window
  - Click create and save as Engine_Failure_anim under Animations
  - Enable record mode, select sprites in the Player Hurt folder and drag&drop to the Animation Window and then disable record mode
  - Select the left engine and add Animator component
  - Set the Animator Controller of the left engine: Engine_Failure