

# Unity Backend

게임 시스템 구현과 “게임 정보 관리” 기능을 이용한 유저 정보 갱신

Created in 2023-03-23  
Last Updated 2023-03-24  
Unity Version 2022.2.2f1

## *Index*

- ◆ 게임 시스템 구현 I
- ◆ 게임 정보 업데이트

# 게임 시스템 구현 I

- Lobby -> Game 씬 전환
- 플레이어 이동
- 배경화면 스크롤
- 플레이어 공격
- 적 자동 생성
- 운석 자동 생성
- 오브젝트 충돌 처리



# 게임 시스템 구현 I

## ■ Lobby -> Game 씬 전환

### ■ Game 씬 생성

#### □ File - New Scene

The screenshot shows the Unity interface. On the left, the 'File' menu is open, with 'New Scene' highlighted. The 'New Scene' dialog is open, showing 'Scene Templates in Project'. The 'Basic 2D (Built-in)' template is selected and highlighted with a red box. The 'Basic 3D (Built-in)' template is also visible. The 'Create' button at the bottom right of the dialog is also highlighted with a red box.

File Edit Assets GameObject Component Services

New Scene Ctrl+N

Open Scene Ctrl+O

Open Recent Scene >

Save Ctrl+S

Save As... Ctrl+Shift+S

Save As Scene Template...

New Project...

Open Project...

Save Project

Build Settings... Ctrl+Shift+B

Build And Run Ctrl+B

Exit

New Scene

Scene Templates in Project

2D Basic 2D (Built-in)

3D Basic 3D (Built-in)

Empty

Basic 2D (Built-in)

Description  
Contains an orthographic camera setup for 2D games. Works with built-in renderer.

To begin using a template, create a template from an existing scene in your project.  
Click to see Scene template documentation.

Load additively

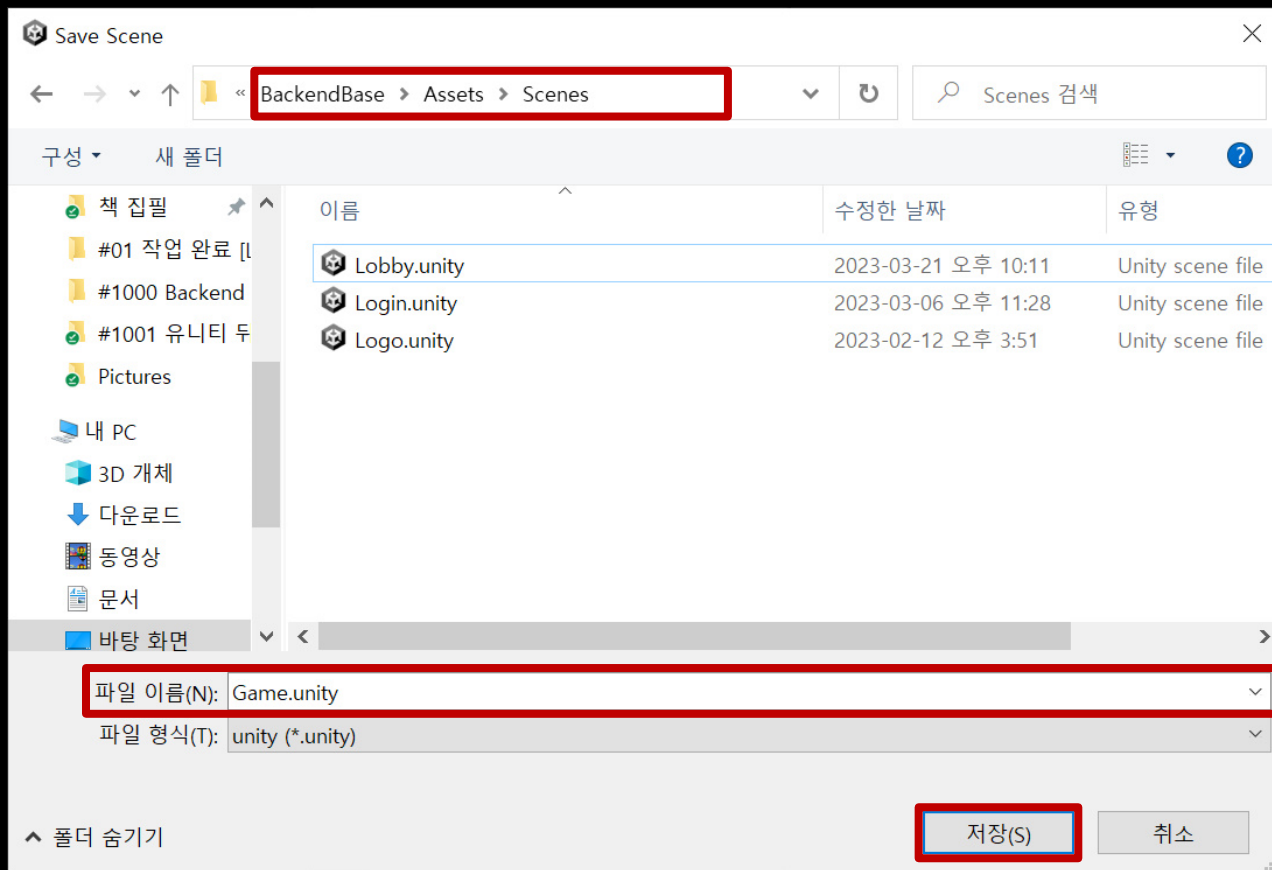
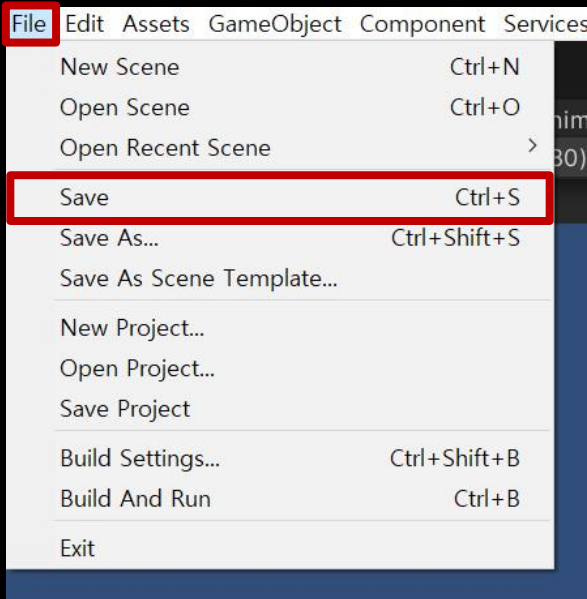
Create Cancel



# 게임 시스템 구현 I

## ■ Game 실행 저장

### □ File - Save





# 게임 시스템 구현 I

- Game 실행 등록
  - File - Build Settings

The screenshot displays the Unity Build Settings interface. On the left, the 'Scenes In Build' list contains four entries: 'Scenes/Logo' (0), 'Scenes/Login' (1), 'Scenes/Lobby' (2), and 'Scenes/Game' (3). The 'Scenes/Game' entry is checked and highlighted with a red box. A red arrow points from this entry to the 'Game' scene in the 'Assets > Scenes' panel on the right. Below the list is an 'Add Open Scenes' button. The 'Platform' section shows 'Windows, Mac, Linux' selected, with various configuration options for the target platform.

Scenes In Build	Count
✓ Scenes/Logo	0
✓ Scenes/Login	1
✓ Scenes/Lobby	2
✓ Scenes/Game	3

Assets > Scenes

- Game
- Lobby
- Login
- Logo

Platform: Windows, Mac, Linux

Target Platform: Windows

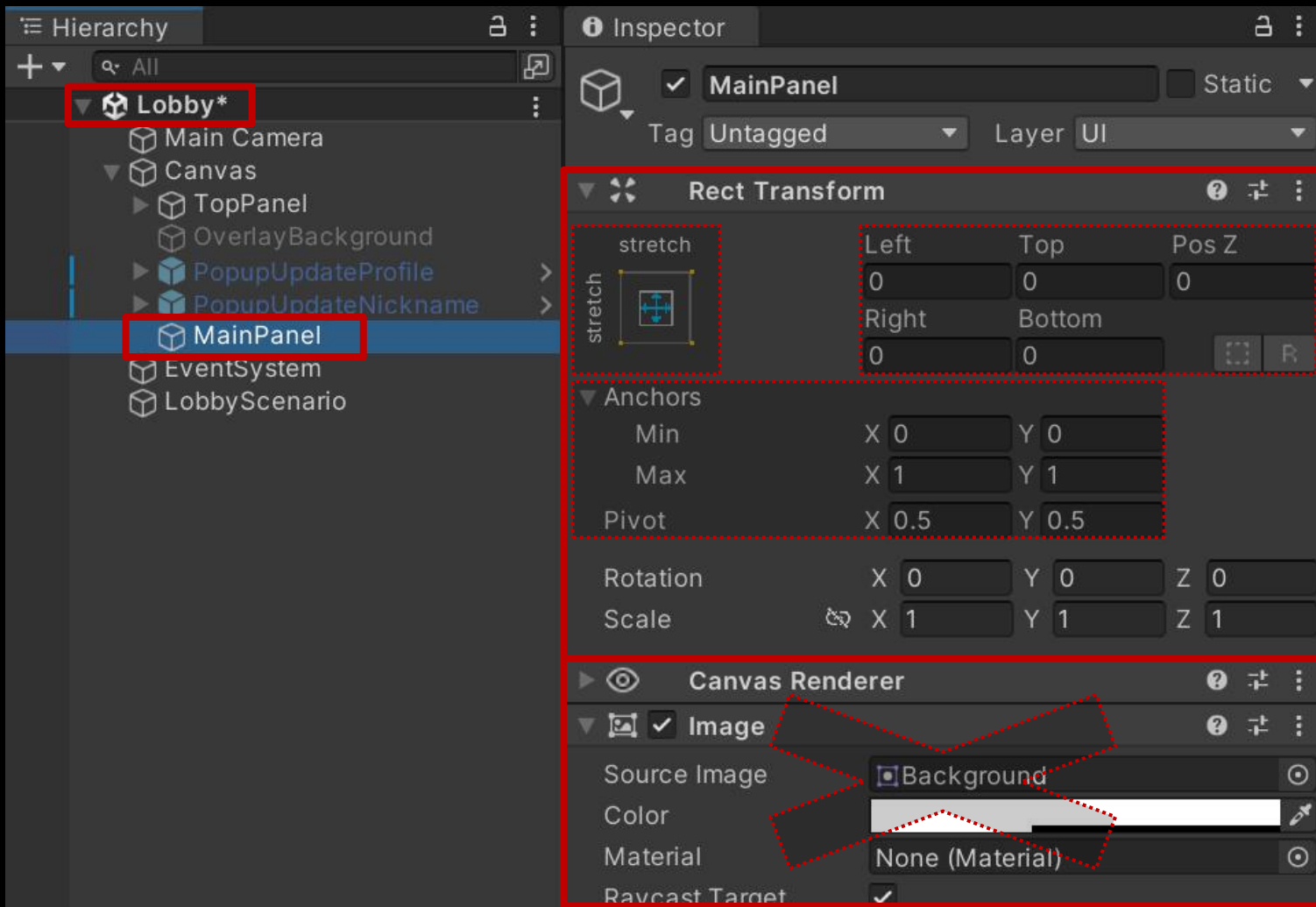
Architecture: Intel 64-bit

Compression Method: Default



# 게임 시스템 구현 I

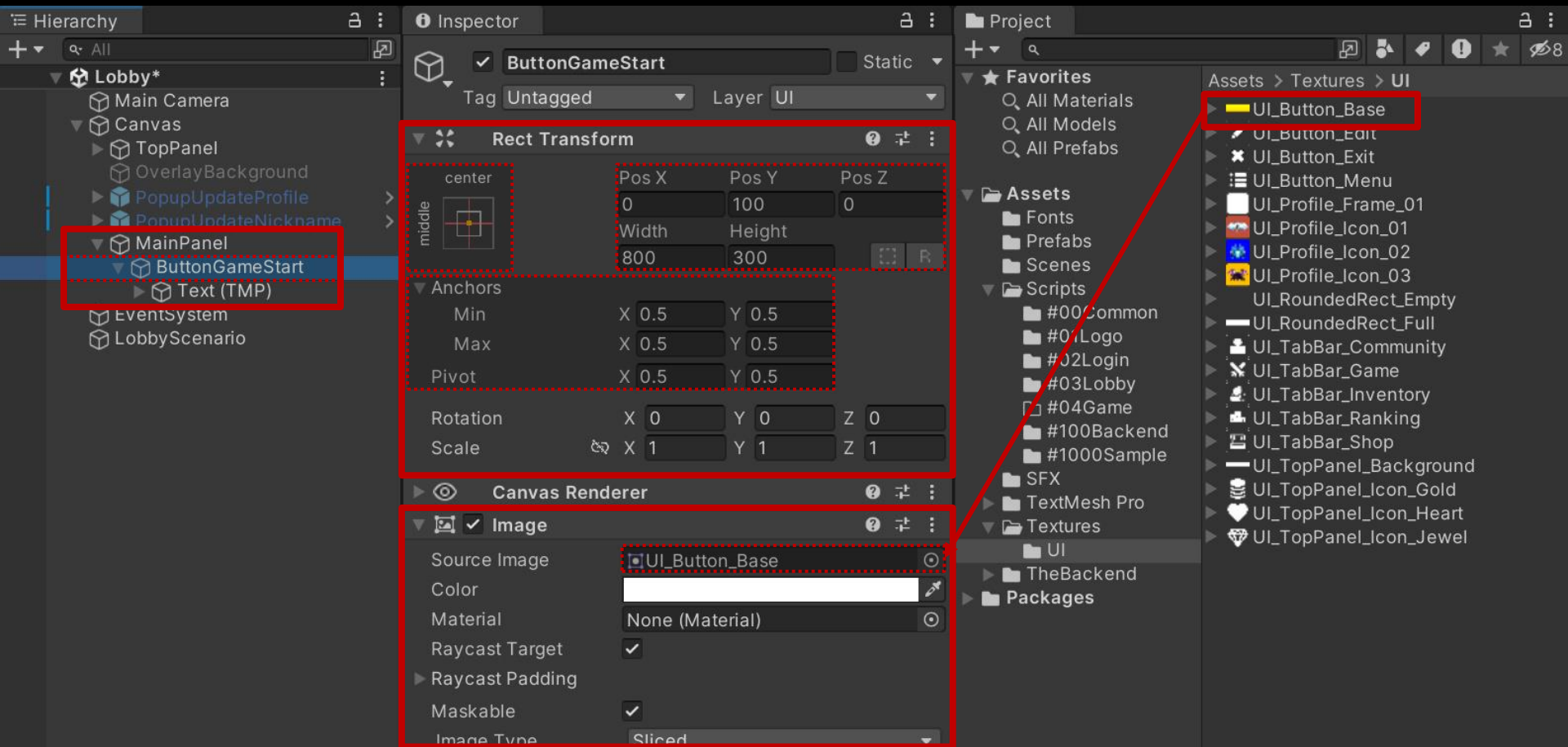
- 로비 메인에 출력하는 UI를 관리하는 Panel UI 생성 및 설정
  - GameObject - UI - Panel





# 게임 시스템 구현 I

- 게임 시작 Button UI 생성 및 설정
  - GameObject - UI - "Button - TextMeshPro"







# 게임 시스템 구현 I

## ■ 게임 시작 Button UI 생성 및 설정 (계속)

The screenshot displays the Unity Inspector window for a TextMeshPro component. The Hierarchy panel on the left shows the object path: Lobby\* > Canvas > ButtonGameStart > Text (TMP). The Inspector panel shows the following settings:

- Text (TMP)**: Tag: Untagged, Layer: UI
- Rect Transform**
- Canvas Renderer**
- TextMeshPro - Text (UI)**:
  - Text Input: Enable RTL Editor (disabled)
  - Text: 게임시작
  - Text Style: Normal
  - Main Settings**:
    - Font Asset: NotoSansKR-Bold SDF (TMP\_Fc)
    - Material Preset: NotoSansKR-Bold SDF Material
    - Font Style: B I U S ab AB SC
    - Font Size: 100
    - Auto Size: (disabled)
    - Vertex Color: (color picker)
    - Color Gradient: (disabled)
    - Override Tags: (disabled)
    - Spacing Options (em): Character 0, Word 0, Line 0, Paragraph 0
    - Alignment: (left-aligned)
    - Wrapping: Enabled



# 게임 시스템 구현 I

- SceneNames 열거형에 "Game" 상수 추가
  - Utils Script 수정

```
1 using UnityEngine.SceneManagement;
2
3 public enum SceneNames { Logo=0, Login, Lobby, Game, }
4
5 public static class Utils
6 {
7     public static string GetActiveScene()...
11
12     public static void LoadScene(string sceneName="")...
23
24     public static void LoadScene(SceneNames sceneName)...
29 }
```



# 게임 시스템 구현 I

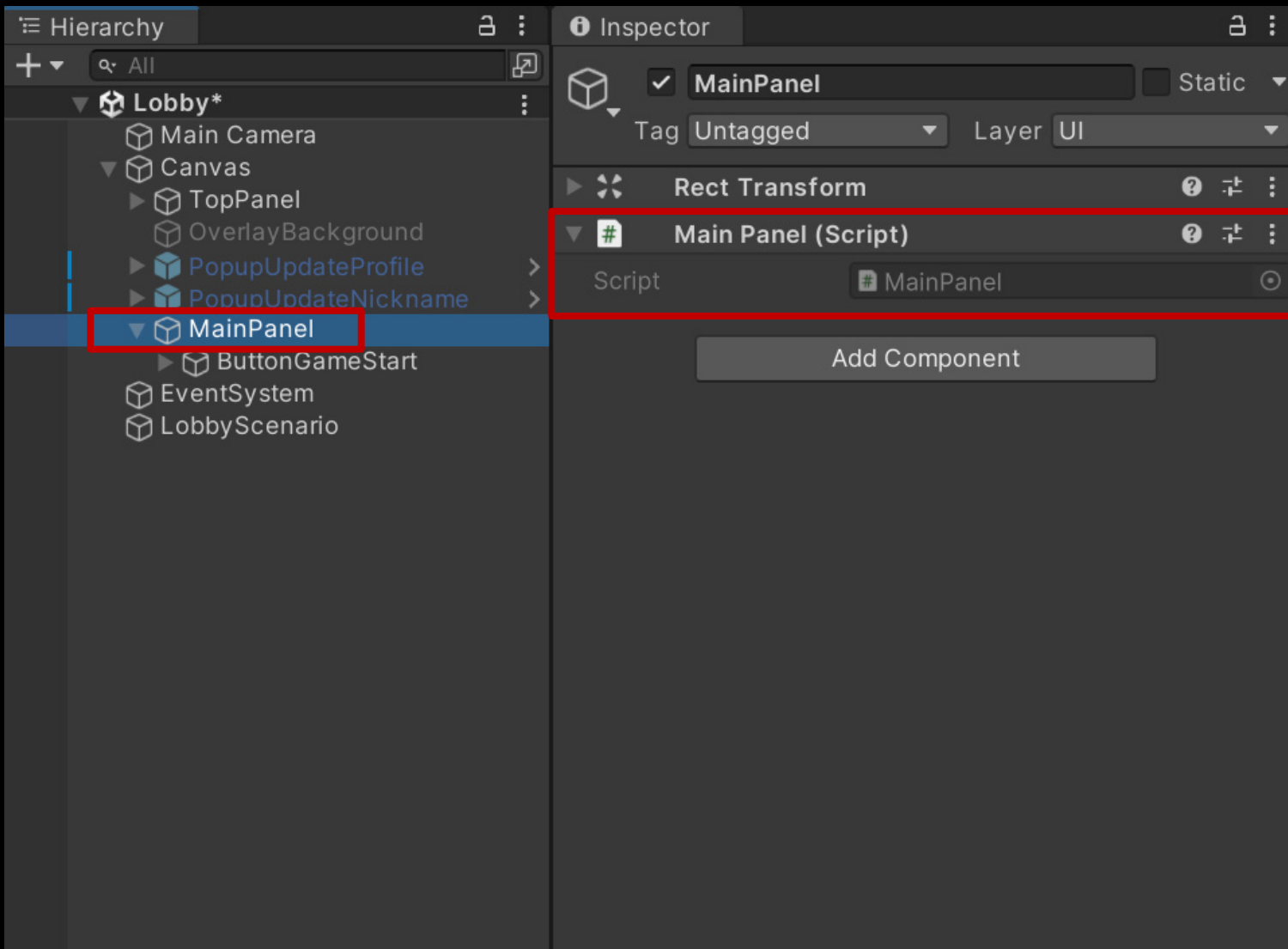
- MainPanel UI를 제어하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "MainPanel"로 변경

```
1 using UnityEngine;
2
3 public class MainPanel : MonoBehaviour
4 {
5     public void BtnClickGameStart()
6     {
7         Utils.LoadScene(SceneNames.Game);
8     }
9 }
```



# 게임 시스템 구현 I

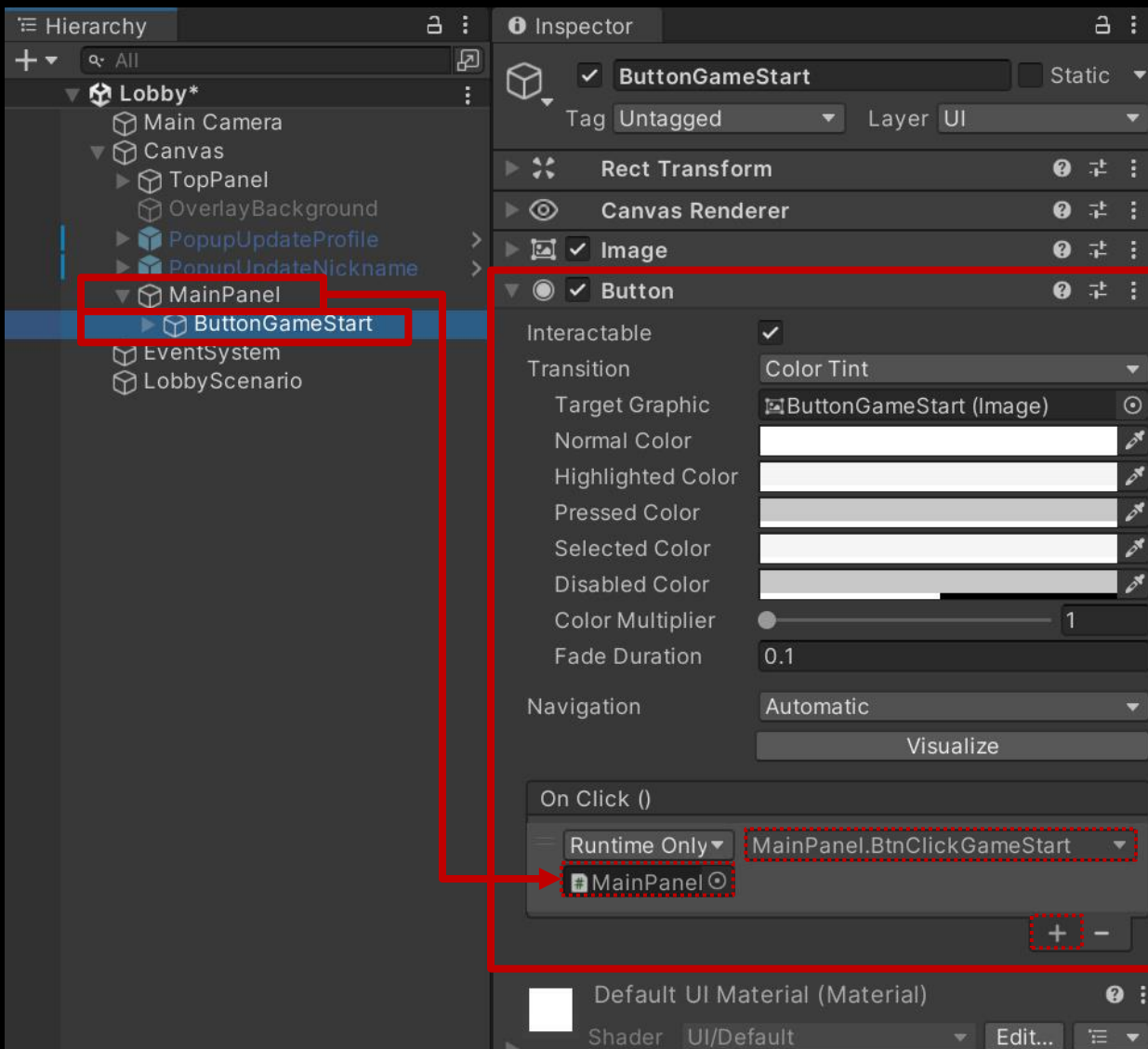
- MainPanel 오브젝트에 "MainPanel" 컴포넌트 추가





# 게임 시스템 구현 I

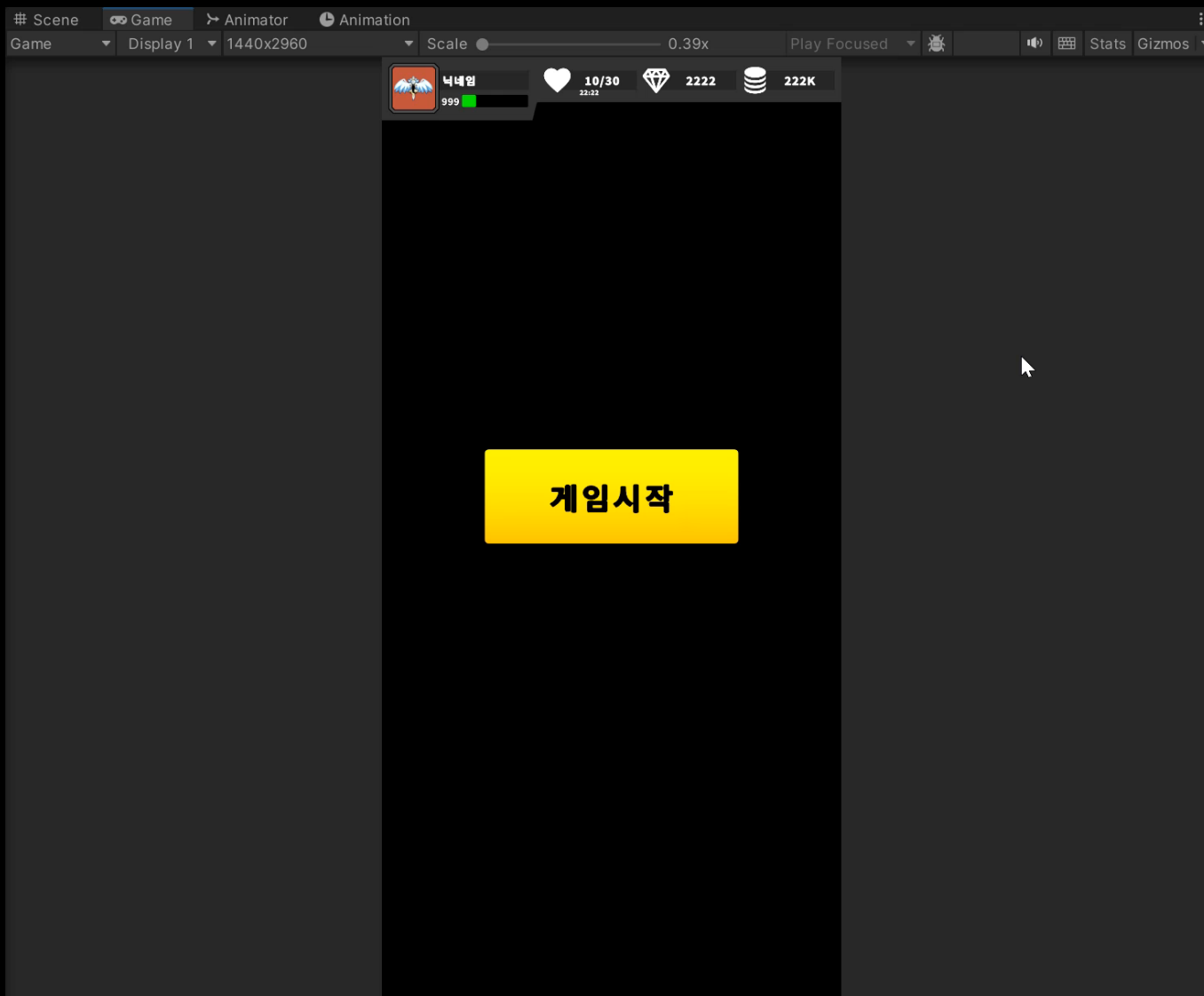
- ButtonGameStart 오브젝트의 "Button" 컴포넌트 onClick() 이벤트 설정





# 게임 시스템 구현 I

## ■ 결과 화면

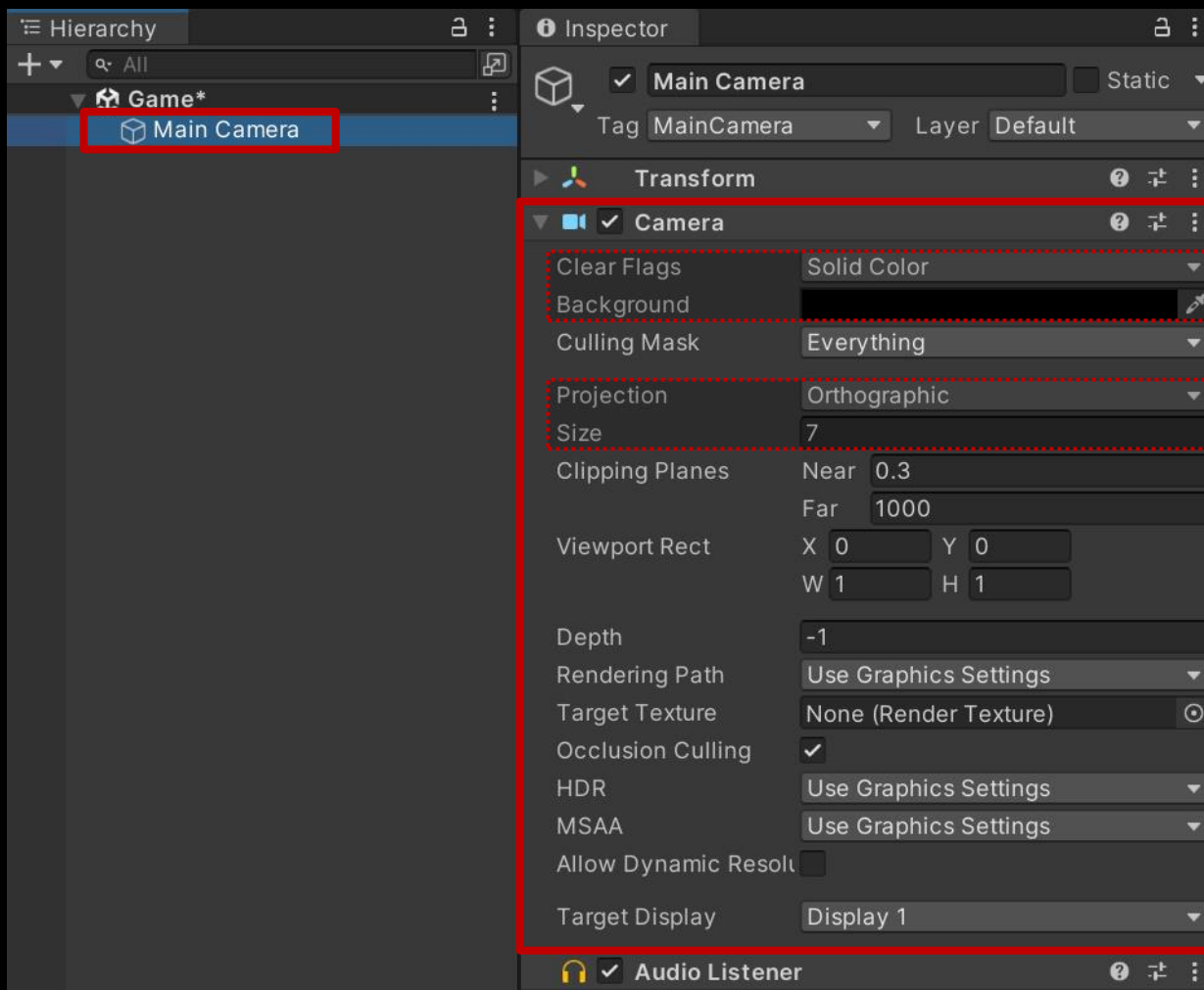




# 게임 시스템 구현 I

## ■ 플레이어 이동

### ■ 카메라 설정





# 게임 시스템 구현 I

## ■ Player\_Idle 텍스처 분할

- 이미지 분할이 가능하도록 Sprite Mode를 Multiple로 변경

The screenshot displays the Unity development environment. On the left, the Inspector panel shows the 'Player\_Idle (Texture 2D) Import Settings'. The 'Sprite Mode' dropdown is set to 'Multiple', and the 'Sprite Editor' button is highlighted with a red box. Below the Inspector, the 'Apply' button is also highlighted. On the right, the Hierarchy panel shows the 'Assets > Textures' folder structure, with 'Player\_Idle' selected under the 'UI' folder. A text box in the bottom right corner provides instructions for the steps shown in the screenshot.

1. Sprite Mode를 "Multiple"로 설정
2. "Apply" 버튼을 눌러서 설정 저장
3. "Sprite Editor" 버튼 선택





# 게임 시스템 구현 I

- Sprite Editor View에서 텍스처를 셀 개수만큼 분할 (Type : Grid By Cell Count)

The screenshot shows the Unity Sprite Editor window. The 'Slice' menu is open, showing the following settings:

- Type: Grid By Cell Count
- Column & Row: C 3, R 1
- Offset: X 0, Y 0
- Padding: X 0, Y 0
- Keep Empty Rects:
- Pivot: Center
- Custom Pivot: X 0, Y 0
- Method: Delete Existing

The 'Apply' button in the top right corner is highlighted with a red box. Below the settings, a character sprite is shown divided into three equal-width sections by vertical dashed yellow lines. The 'Slice' button at the bottom of the settings panel is also highlighted with a red box.

설정이 완료되면 "Apply"를 눌러 분할한 텍스처 저장

Player\_Idle\_0 : Position(0, 0) / Size(100, 50)  
Player\_Idle\_1 : Position(100, 0) / Size(100, 50)  
Player\_Idle\_2 : Position(200, 0) / Size(100, 50)



# 게임 시스템 구현 I

※ 이 페이지는 설명을 위한 페이지로 따로 실습하지 않습니다.

- Player\_Idle 텍스처 아래에 분할된 텍스처들 확인 가능

Inspector - Unsaved Changes Detected

Unapplied import settings for  
'Assets/Textures/Enemy01\_Idle.png'

Save Discard Cancel

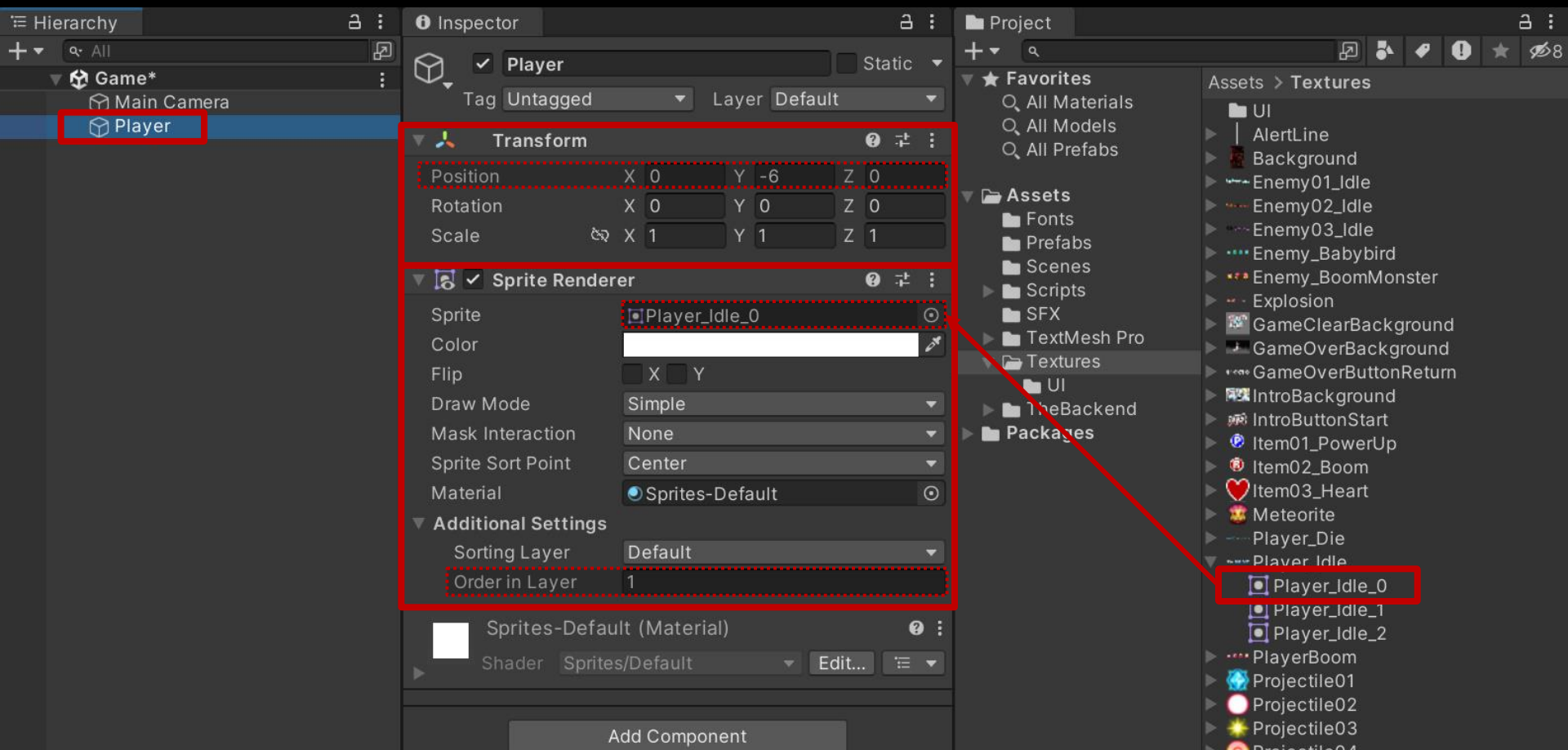
Sprite Editor View를 닫고 유니티 에디터로 돌아왔을 때  
변경 사항이 저장되지 않았다고 뜬다면 "Save" 버튼 클릭

Tip. 원본을 훼손하여 텍스처를 분할하는 것이 아니기 때문에  
분할된 텍스처 뿐만 아니라 원본 텍스처도 사용 가능하다



# 게임 시스템 구현 I

- 플레이어 오브젝트 생성 및 설정
  - GameObject - 2D Object - Sprites - Square



Tip. 화면에 그려지는 순서 설정

“Sprite Renderer” 컴포넌트에 있는 Order in Layer 변수 (숫자가 클수록 앞에 그려진다)



# 게임 시스템 구현 I

- 오브젝트 이동을 제어하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "Movement2D"로 변경

```
1 using UnityEngine;
2
3 public class Movement2D : MonoBehaviour
4 {
5     [SerializeField]
6     private float moveSpeed = 0;           // 이동 속도
7     [SerializeField]
8     private Vector3 moveDirection = Vector3.zero; // 이동 방향
9
10    public Vector3 MoveDirection { set => moveDirection = value; }
11
12    private void Update()
13    {
14        transform.position += moveDirection * moveSpeed * Time.deltaTime;
15    }
16 }
```

기본 이동에 대한 내용만 작성하여 이동이 가능한 모든 객체들이 사용

실행 이후 동일한 방향으로만 이동하는 오브젝트는 Inspector View에서 미리 moveDirection을 설정하고, 실행 이후 방향이 바뀌는 오브젝트는 코드에서 MoveDirection 프로퍼티에 접근해 방향 설정



# 게임 시스템 구현 I

- 플레이어 캐릭터를 제어하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "PlayerController"로 변경

```
1  using UnityEngine;
2
3  public class PlayerController : MonoBehaviour
4  {
5      private Movement2D movement2D;
6
7      private void Awake()
8      {
9          movement2D = GetComponent<Movement2D>();
10     }
11
12     private void Update()
13     {
14         // 방향 키 or wasd키를 눌러 이동방향 설정
15         float x = Input.GetAxisRaw("Horizontal");
16         float y = Input.GetAxisRaw("Vertical");
17
18         movement2D.MoveDirection = new Vector3(x, y, 0);
19     }
20 }
```



# 게임 시스템 구현 I

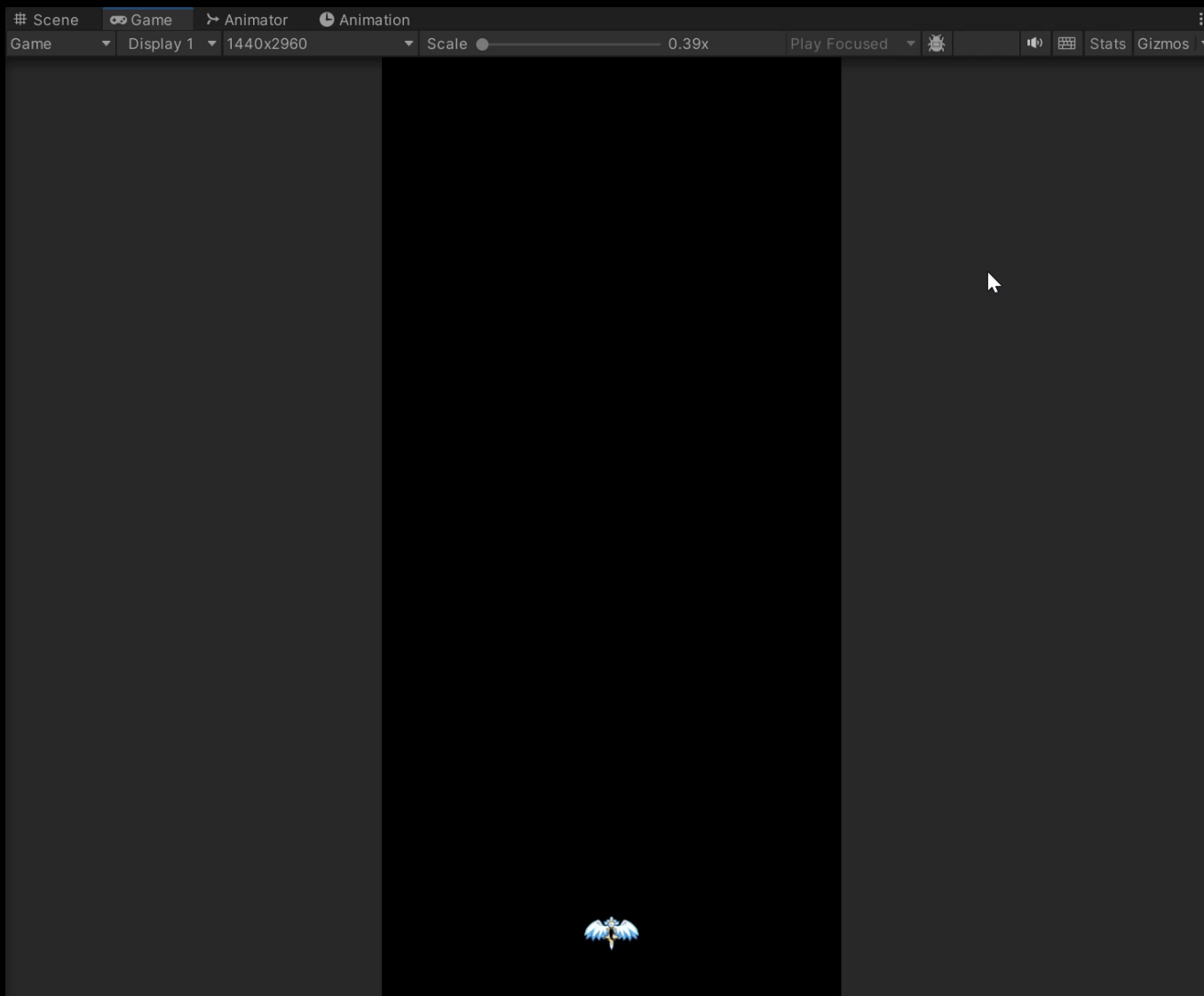
- Player 오브젝트에 "PlayerController", "Movement2D" 컴포넌트 추가

The screenshot displays the Unity development environment. On the left, the Hierarchy panel shows a tree structure under 'Game\*' with 'Main Camera' and 'Player' objects. The 'Player' object is selected and highlighted with a red box. On the right, the Inspector panel shows the properties of the selected 'Player' object. The 'Player' component is checked, and its 'Tag' is set to 'Untagged' and 'Layer' to 'Default'. Below it, the 'Transform' component is visible. The 'Sprite Renderer' component is checked. The 'Player Controller (Script)' component is checked, and its 'Script' is set to 'PlayerController'. The 'Movement 2D (Script)' component is checked, and its 'Script' is set to 'Movement2D'. The 'Move Speed' property is set to 5, and the 'Move Direction' is set to X: 0, Y: 0, Z: 0. The 'Sprites-Default (Material)' component is visible below, with its 'Shader' set to 'Sprites/Default'. An 'Add Component' button is visible at the bottom of the Inspector panel.



# 게임 시스템 구현 I

## ■ 결과 화면





# 게임 시스템 구현 I

- 스테이지의 데이터를 저장하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "StageData"로 변경

```
1 using UnityEngine;
```

```
2
```

```
3 [CreateAssetMenu]
```

```
4 public class StageData : ScriptableObject
```

```
5 {
```

```
6     // 스테이지의 크기(상하좌우 범위)를 limitMin, limitMax 변수에 저장
```

```
7     [SerializeField]
```

```
8     private Vector2 limitMin;
```

```
9     [SerializeField]
```

```
10    private Vector2 limitMax;
```

```
11
```

```
12    // 다른 클래스에서는 LimitMin, LimitMax 프로퍼티를 통해
```

```
13    // 스테이지의 크기를 확인할 수 있다.
```

```
14    public Vector2 LimitMin => limitMin;
```

```
15    public Vector2 LimitMax => limitMax;
```

```
16 }
```

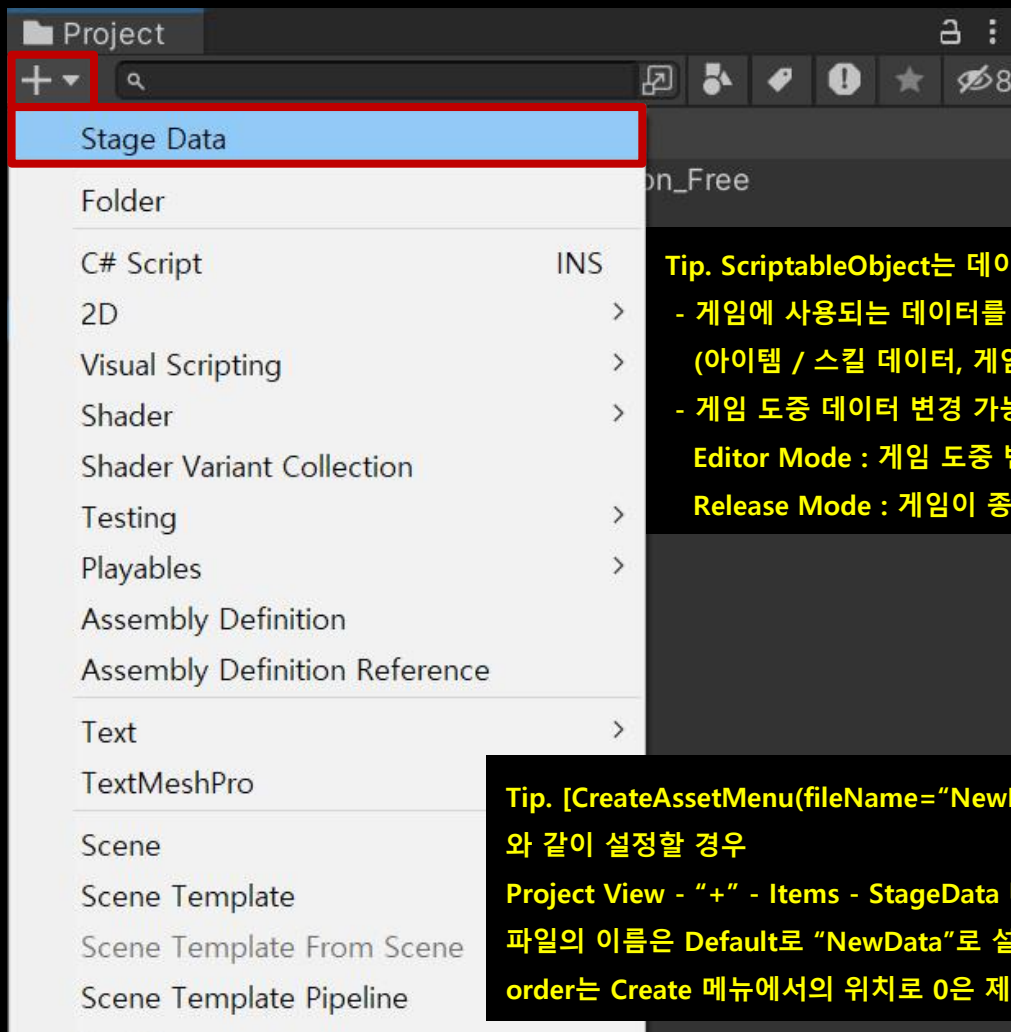
부모 클래스로 ScriptableObject를 사용하면 해당 클래스를 에셋(파일)의 형태로 저장할 수 있고, 3번째 줄과 같이 클래스 상단에 [CreateAssetMenu] 어트리뷰트를 작성하면 Project View의 생성("+") 메뉴에 메뉴로 등록된다





# 게임 시스템 구현 I

- StageData 타입의 Asset 파일 생성
  - Project View - "+" - Stage Data



**Tip. ScriptableObject**는 데이터를 파일로 저장할 수 있는 클래스  
- 게임에 사용되는 데이터를 저장해두고 게임 중간에 불러오기 가능  
(아이템 / 스킬 데이터, 게임 내 데이터, 씬 사이에 유지되는 데이터)  
- 게임 도중 데이터 변경 가능  
**Editor Mode** : 게임 도중 변경된 데이터가 저장  
**Release Mode** : 게임이 종료되면 게임 시작 전 데이터로 리셋

**Tip. [CreateAssetMenu(fileName="NewData", menuName="Items", order=999)]**  
와 같이 설정할 경우  
**Project View - "+" - Items - StageData** 메뉴가 생성되며,  
파일의 이름은 Default로 "NewData"로 설정  
**order**는 Create 메뉴에서의 위치로 0은 제일 위, 999면 일반적으로 제일 아래에 배치



# 게임 시스템 구현 I

## ■ StageData 타입의 Asset 파일 설정

The screenshot shows the Unity Inspector window for a StageData script and the Hierarchy window. The Inspector window displays the StageData script with the following configuration:

Property	X	Y
Limit Min	-3	-6.6
Limit Max	3	6.6

The Hierarchy window shows the following structure:

- Assets
  - Scripts
  - SFX
  - TextMesh Pro
  - Textures
  - TheBackend
  - Stage01

Annotations in the image include:

- A red box around the Inspector window's configuration fields.
- A red box around the Stage01 asset in the Hierarchy window.
- A red box around the Stage01 asset in the Assets panel.
- Text boxes indicating the values: "Limit Max : (3, 6.6)" and "Limit Min : (-3, -6.6)".
- A text box explaining: "StageData 클래스에 선언한 모든 public or [SerializeField] 타입의 변수가 표시되며, 데이터로 저장할 수 있다".



# 게임 시스템 구현 I

- StageData 정보를 바탕으로 플레이어 캐릭터 이동 범위 제어
  - PlayerController Script 수정

```
1 using UnityEngine;
2
3 public class PlayerController : MonoBehaviour
4 {
5     [SerializeField]
6     private StageData stageData;
7     private Movement2D movement2D;
8
9     private void Awake()...
10
11
12
13
14     private void Update()...
15
16
17
18
19     private void LateUpdate()
20     {
21         // 플레이어 캐릭터가 스테이지 범위 바깥으로 나가지 못하도록 제어
22         transform.position = new Vector3(Mathf.Clamp(transform.position.x, stageData.LimitMin.x, stageData.LimitMax.x),
23             Mathf.Clamp(transform.position.y, stageData.LimitMin.y, stageData.LimitMax.y));
24     }
25 }
```

ScriptableObject 기반으로 생성된 Asset은 일반적인 클래스 객체에 접근하는 것과 동일하게 접근하여 변수, 메소드 사용 가능

플레이어의 좌표(x, y)가 StageData 클래스에 설정된 LimitMin, LimitMax의 범위를 벗어나지 않도록 연산

== built-in method ==

```
float result = Mathf.Clamp(float value, float min, float max);
```

value 값이 min보다 작으면 result = min

value 값이 max보다 크면 result = max

value 값이 min과 max 사이의 값이면 result = value



# 게임 시스템 구현 I

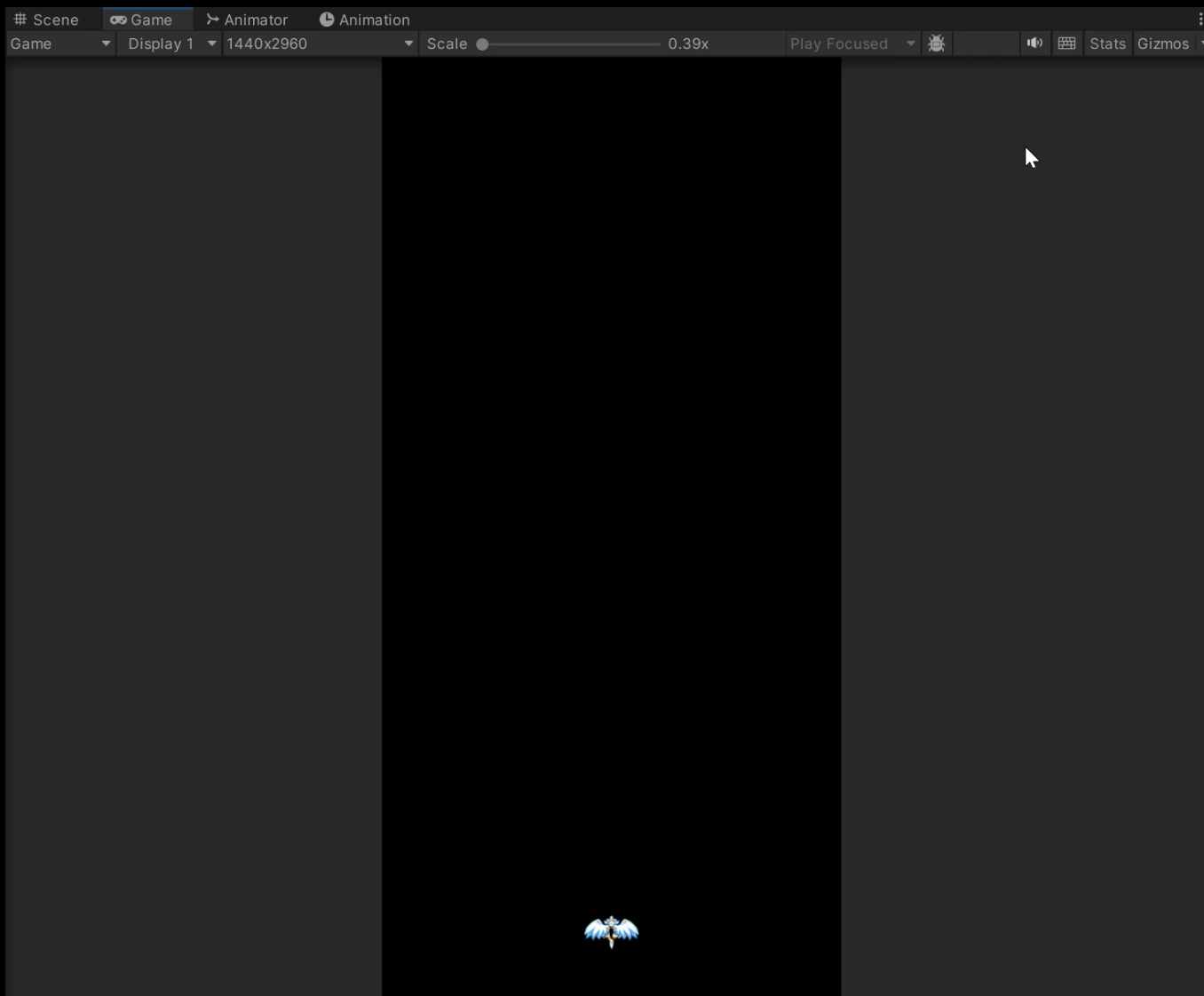
- Player 오브젝트의 "PlayerController" 컴포넌트 변수 설정

The screenshot displays the Unity Inspector window for a 'Player' object. The 'Player Controller (Script)' component is selected, and its 'Stage Data' property is set to 'Stage01 (Stage Data)'. A red box highlights the 'Player Controller (Script)' component and its 'Stage Data' property. A red arrow points from the 'Stage01 (Stage Data)' property in the Inspector to the 'Stage01' asset in the Project Hierarchy window. The Project Hierarchy window shows the 'Assets' folder expanded, with 'Stage01' highlighted. The Hierarchy window on the left shows the 'Player' object selected under the 'Game\*' object.



# 게임 시스템 구현 I

## ■ 결과 화면





# 게임 시스템 구현 I

## ■ 배경화면 스크롤

### ■ 배경 오브젝트 생성 및 설정

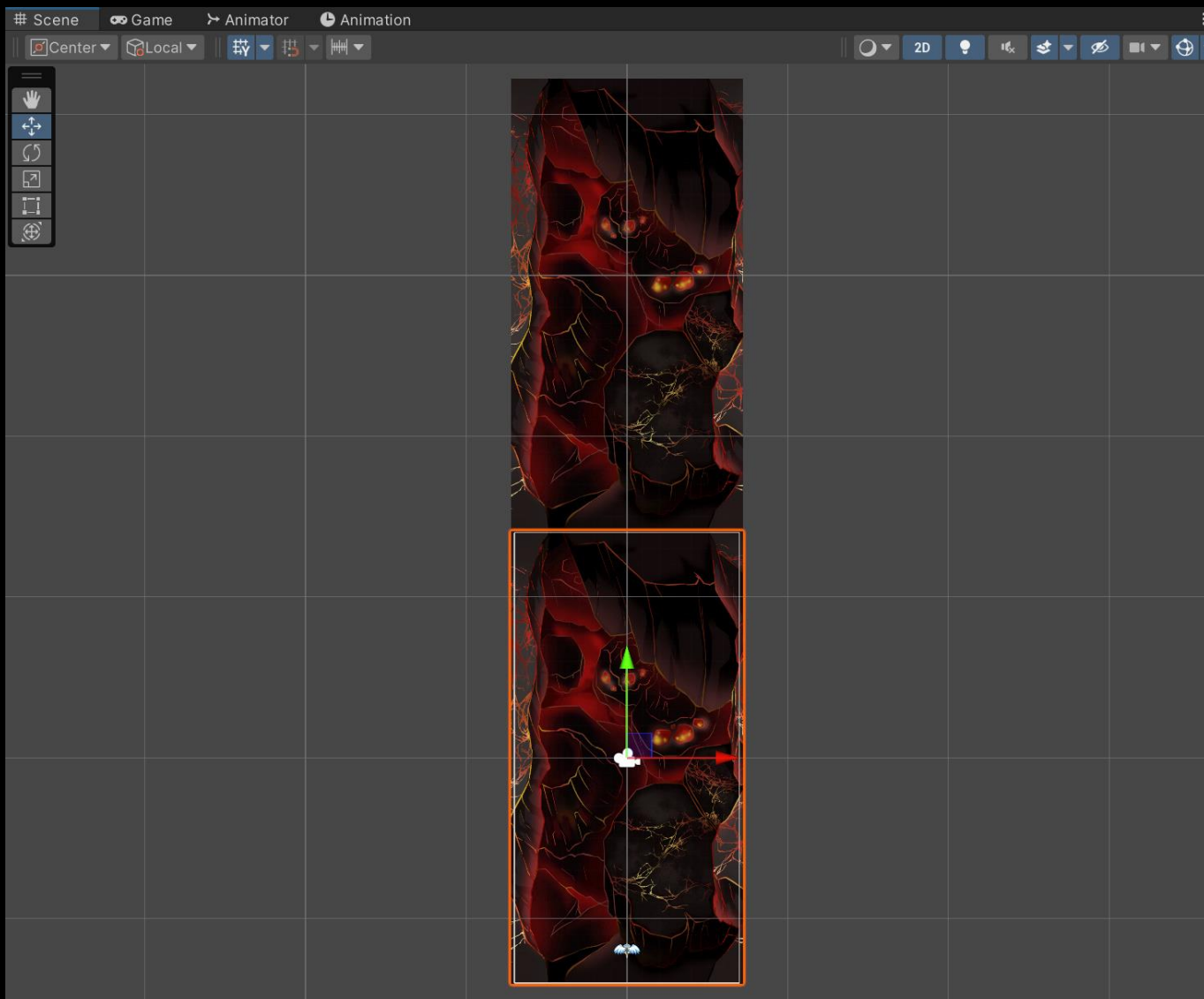
- GameObject - 2D Object - Sprites - Square

**Background01** 위치 : Position (0, 0, 0) / 크기 : Scale (1, 1.1, 1)  
**Background02** 위치 : Position (0, 14.08, 0) / 크기 : Scale (1, 1.1, 1)



# 게임 시스템 구현 I

## ■ 결과 화면





# 게임 시스템 구현 I

- 배경 오브젝트를 제어하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "ParallaxBackground"로 변경

```
1 using UnityEngine;
2
3 public class ParallaxBackground : MonoBehaviour
4 {
5     [SerializeField]
6     private Transform target; // 현재 배경과 이어지는 배경
7     [SerializeField]
8     private float scrollAmount = 14.08f; // 두 배경 사이의 y 거리
9     [SerializeField]
10    private float moveSpeed = 3; // 배경의 이동 속도
11    [SerializeField]
12    private Vector3 moveDirection = Vector3.down; // 배경의 이동 방향
13
14    private void Update()
15    {
16        // 배경이 moveDirection 방향으로 moveSpeed의 속도로 이동
17        transform.position += moveDirection * moveSpeed * Time.deltaTime;
18
19        // 배경이 설정된 범위를 벗어나면 위치 재설정
20        if ( transform.position.y <= -scrollAmount )
21        {
22            transform.position = target.position - moveDirection * scrollAmount;
23        }
24    }
25 }
```





# 게임 시스템 구현 I

- Background01 오브젝트에 "ParallaxBackground" 컴포넌트 추가 및 설정

The screenshot shows the Unity Inspector window for the 'Background01' object. The 'Parallax Background (Script)' component is selected, and its 'Target' property is set to 'Background02 (Transform)'. The 'Scroll Amount' is 14.08, 'Move Speed' is 3, and 'Move Direction' is X: 0, Y: -1, Z: 0. A red box highlights the 'Parallax Background (Script)' component and its 'Target' property. A red arrow points from the 'Background01' object in the Hierarchy panel to the 'Parallax Background (Script)' component in the Inspector. A text box at the bottom left states: 'Background01 오브젝트의 Target은 Background02'.



# 게임 시스템 구현 I

- Background02 오브젝트에 "ParallaxBackground" 컴포넌트 추가 및 설정

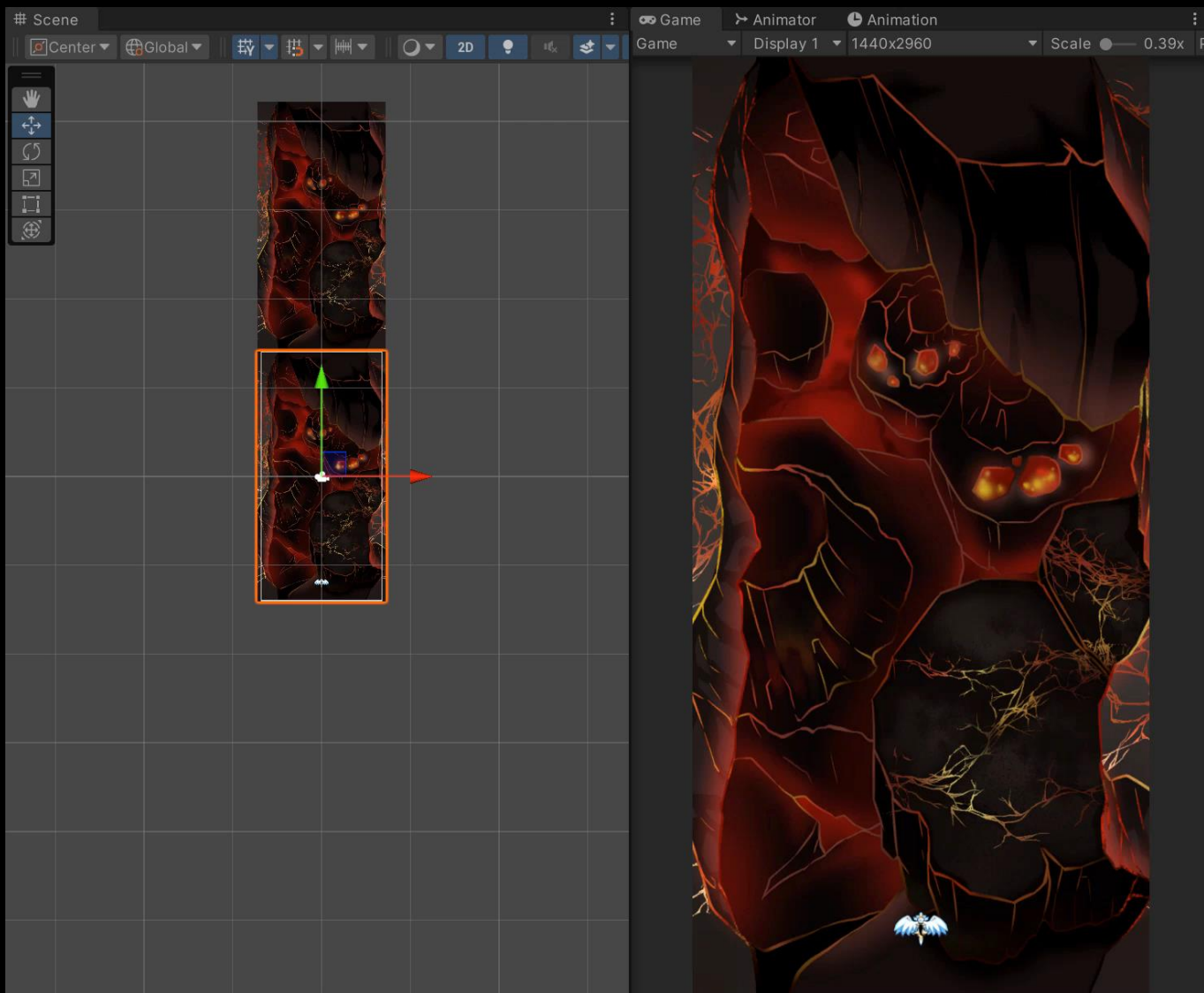
The screenshot shows the Unity Inspector window with the Hierarchy panel on the left and the Inspector panel on the right. In the Hierarchy panel, the 'Background02' object is selected and highlighted with a red box. In the Inspector panel, the 'Parallax Background (Script)' component is selected and highlighted with a red box. The 'Target' property of the script is set to 'Background01 (Transform)', which is also highlighted with a red dashed box. A red arrow points from the 'Background01' object in the Hierarchy panel to the 'Target' property in the Inspector panel. Below the Inspector panel, there is a text box with the text 'Background02 오브젝트의 Target은 Background01'.

Background02 오브젝트의 Target은 Background01



# 게임 시스템 구현 I

## ■ 결과 화면





# 게임 시스템 구현 I

## ■ 플레이어 공격

### ■ 발사체 오브젝트 생성 및 설정

- GameObject - 2D Object - Sprites - Square

The screenshot displays the Unity Inspector for a **PlayerProjectile** object. The **Sprite Renderer** component is highlighted with a red dashed box, showing the **Sprite** set to **Projectile01** and **Order in Layer** set to **1**. The **Movement 2D (Script)** component is also highlighted with a red dashed box, showing **Move Speed** set to **8** and **Move Direction** set to **(0, 1, 0)**. A red arrow points from the **Projectile01** texture in the **Assets > Textures** panel to the **Sprite** field in the **Sprite Renderer** component.

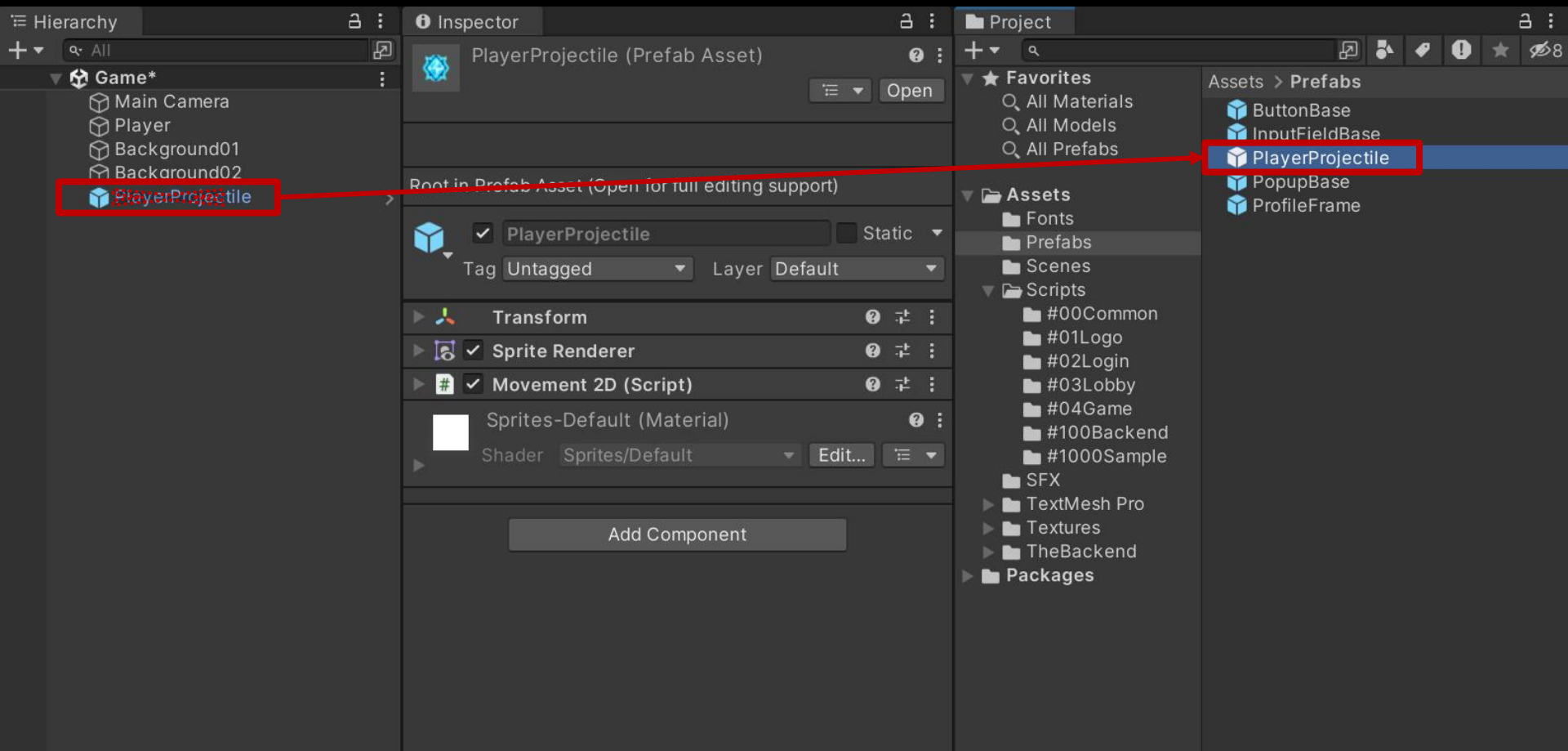
**Movement2D의 MoveSpeed와 MoveDirection 변수로 인해 (0, 8, 0) 속력으로 이동**



# 게임 시스템 구현 I

## ■ 발사체 오브젝트 Prefab 생성

- Hierarchy View의 "PlayerProjectile" 오브젝트를 Project View로 Drag & Drop
- Hierarchy View에 있는 "PlayerProjectile" 오브젝트 삭제





# 게임 시스템 구현 I

- 플레이어의 공격을 제어하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "Weapon"으로 변경

```
1 using UnityEngine;
2
3 public class Weapon : MonoBehaviour
4 {
5     [SerializeField]
6     private GameObject projectilePrefab; // 공격할 때 생성하는 발사체 프리팹
7     [SerializeField]
8     private float attackRate = 0.1f; // 공격 속도
9
10    private float lastAttackTime = 0; // 마지막 공격시간 체크
11
```



# 게임 시스템 구현 I

- 플레이어의 공격을 제어하는 스크립트 생성 및 작성 (계속)

```
12 public void WeaponAction()
13 {
14     // 마지막 공격으로부터 attackRate 시간만큼 지나야 공격 가능
15     if ( Time.time - lastAttackTime > attackRate )
16     {
17         // 현재 플레이어 위치(transform.position)에 발사체 오브젝트를 생성
18         Instantiate(projectilePrefab, transform.position, Quaternion.identity);
19
20         // 공격주기가 되어야 공격할 수 있도록 하기 위해 현재 공격시간 저장
21         lastAttackTime = Time.time;
22     }
23 }
24 }
```

== built-in property ==

Time.time

게임이 시작된 직후부터 현재까지 흐른 시간

== built-in method ==

Instantiate(GameObject object, Vector3 position, Quaternion angle);

오브젝트 생성 메소드

매개변수로 받은 object를 복제하여 position 위치, angle 회전을 적용해 게임 내에 배치한다



# 게임 시스템 구현 I

- Player 오브젝트에 "Weapon" 컴포넌트 추가 및 변수 설정

The screenshot displays the Unity Inspector window for a 'Player' object. The 'Weapon (Script)' component is selected, and its properties are visible:

- Script: Weapon
- Projectile Prefab: PlayerProjectile
- Attack Rate: 0.1

The 'PlayerProjectile' prefab is also highlighted in the 'Assets > Prefabs' panel on the right. A red box highlights the 'Weapon (Script)' component and its properties in the Inspector, and another red box highlights the 'PlayerProjectile' prefab in the Prefabs panel. A red arrow points from the 'PlayerProjectile' prefab to the 'Projectile Prefab' field in the 'Weapon (Script)' component.





# 게임 시스템 구현 I

## ■ 공격주기마다 자동 공격

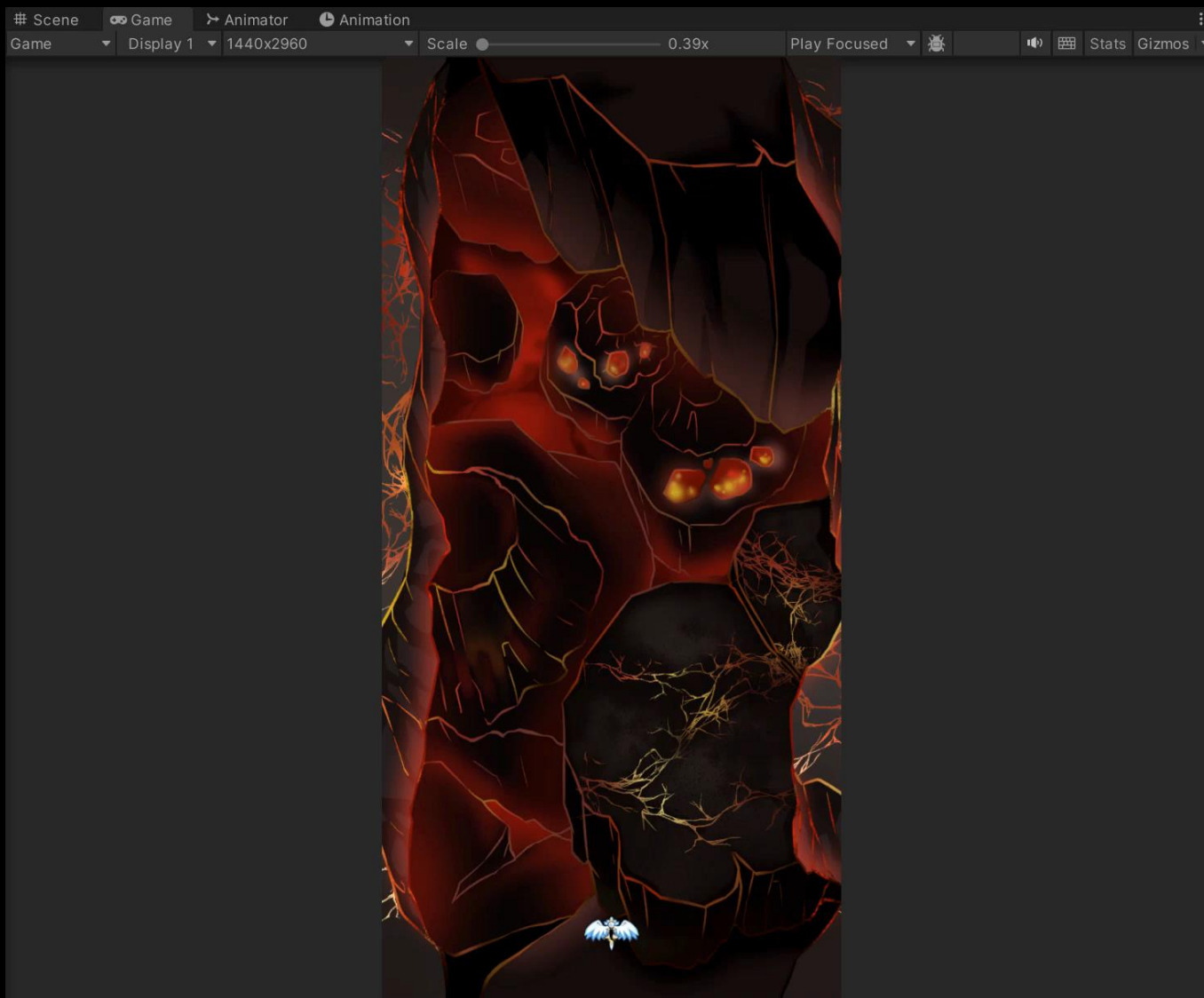
### □ PlayerController Script 수정

```
1 using UnityEngine;
2
3 public class PlayerController : MonoBehaviour
4 {
5     [SerializeField]
6     private StageData stageData;
7
8     private Movement2D movement2D;
9     private Weapon weapon;
10
11     private void Awake()
12     {
13         movement2D = GetComponent<Movement2D>();
14         weapon = GetComponent<Weapon>();
15     }
16
17     private void Update()
18     {
19         // 방향 키 or wasd키를 눌러 이동방향 설정
20         float x = Input.GetAxisRaw("Horizontal");
21         float y = Input.GetAxisRaw("Vertical");
22
23         movement2D.MoveDirection = new Vector3(x, y, 0);
24
25         // 공격 주기를 계산하고, 공격이 가능하면 공격
26         weapon.WeaponAction();
27     }
28
29     private void LateUpdate() { ... }
30
31 }
32
33
34
35 }
```



# 게임 시스템 구현 I

## ■ 결과 화면



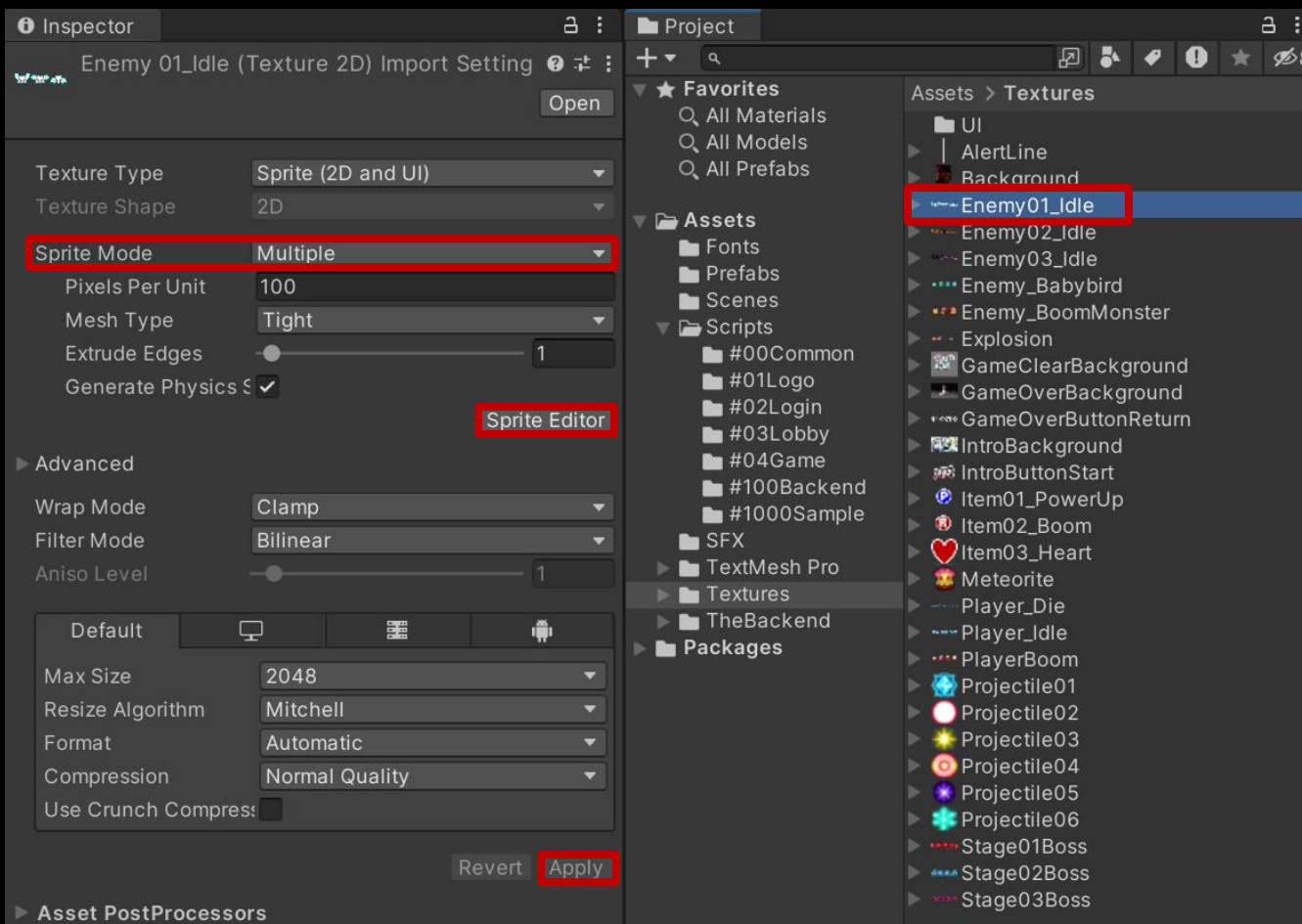


# 게임 시스템 구현 I

## ■ 적 자동 생성

### ■ Enemy01\_Idle 텍스처 분할

- 이미지 분할이 가능하도록 Sprite Mode를 Multiple로 변경





# 게임 시스템 구현 I

- 빈 공간을 마우스로 드래그해 수동 텍스처 분할

Sprite Editor

Sprite Editor ▾ Slice ▾ Trim

Revert Apply

설정이 완료되면 "Apply"를 눌러 분할한 텍스처 저장

날개 부분이 겹쳐 자동 분할 불가능

Enemy01\_Idle\_0 : Position XY(0, 0) / WH(93, 60)  
Enemy01\_Idle\_1 : Position XY(93, 0) / WH(108, 60)  
Enemy01\_Idle\_2 : Position XY(201, 0) / WH(99, 60)

Sprite

Name	Enemy01_Idle_0			
Position	X	0		
	Y	0		
Border	W	93		
	H	60		
Pivot	L	0	T	0
	R	0	B	0
	Center			
	Normalized			
Custom Pivot	X	0.5	Y	0.5



# 게임 시스템 구현 I

- 적 오브젝트 생성 및 설정
  - GameObject - 2D Object - Sprites - Square

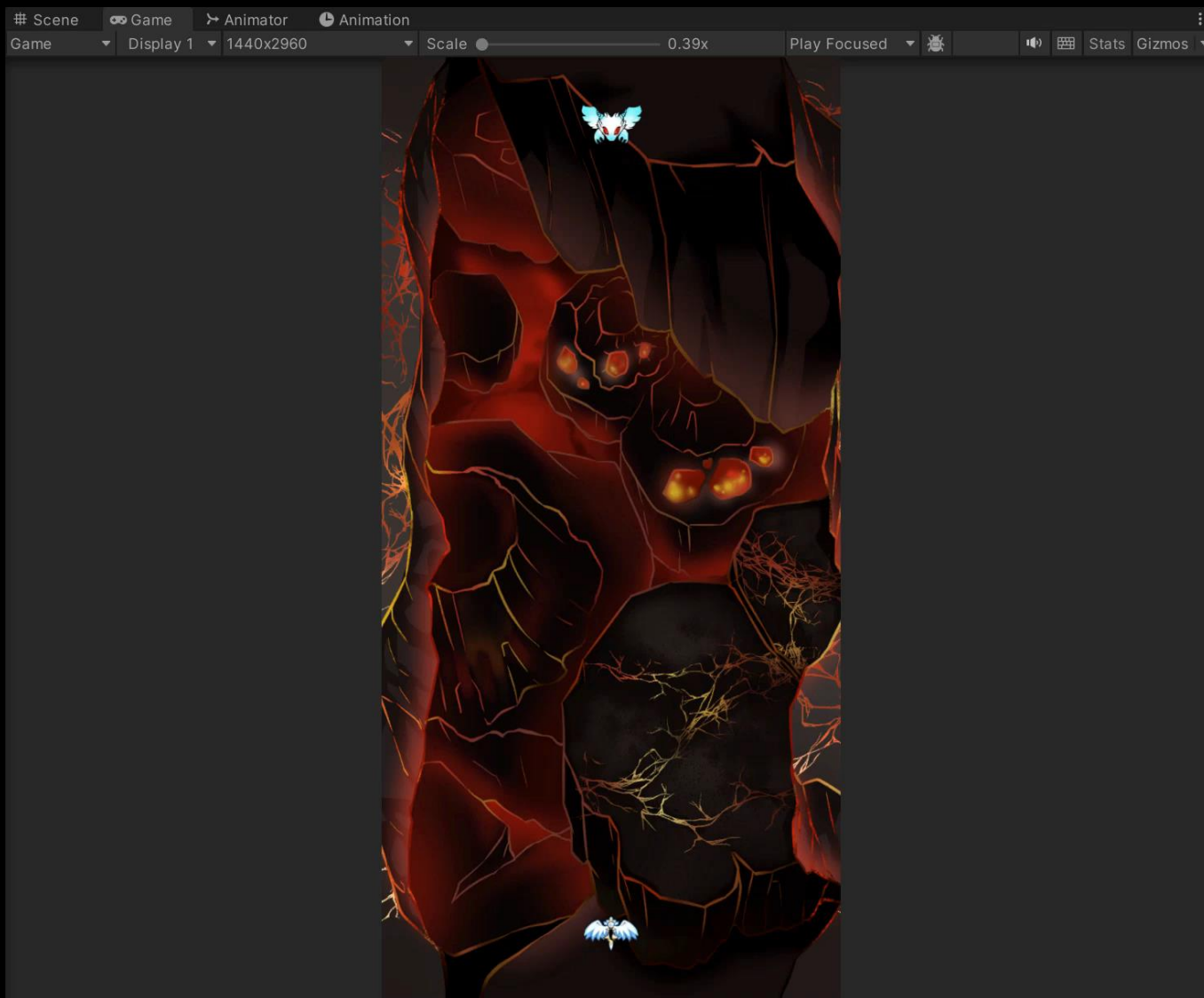
The screenshot displays the Unity engine interface with the following components:

- Hierarchy Panel:** Shows the scene hierarchy with 'Enemy' selected under the 'Game\*' root.
- Inspector Panel:** Shows the properties of the selected 'Enemy' object, which is a 2D Sprite Renderer. The 'Transform' component is set to Position (0, 6, 0), Rotation (0, 0, 0), and Scale (1, 1, 1). The 'Sprite Renderer' component has 'Sprite' set to 'Enemy01\_Idle\_0', 'Color' is white, 'Flip' is unchecked, 'Draw Mode' is 'Simple', 'Mask Interaction' is 'None', 'Sprite Sort Point' is 'Center', and 'Material' is 'Sprites-Default'. Under 'Additional Settings', 'Sorting Layer' is 'Default' and 'Order in Layer' is '1'. The 'Movement 2D (Script)' component has 'Script' set to 'Movement2D', 'Move Speed' is '5', and 'Move Direction' is (0, -1, 0).
- Project Panel:** Shows the 'Assets' folder structure, including 'Textures' and 'Sprites'.
- Assets > Textures Panel:** Shows the 'Enemy01\_Idle\_0' texture asset selected, which is a square sprite.



# 게임 시스템 구현 I

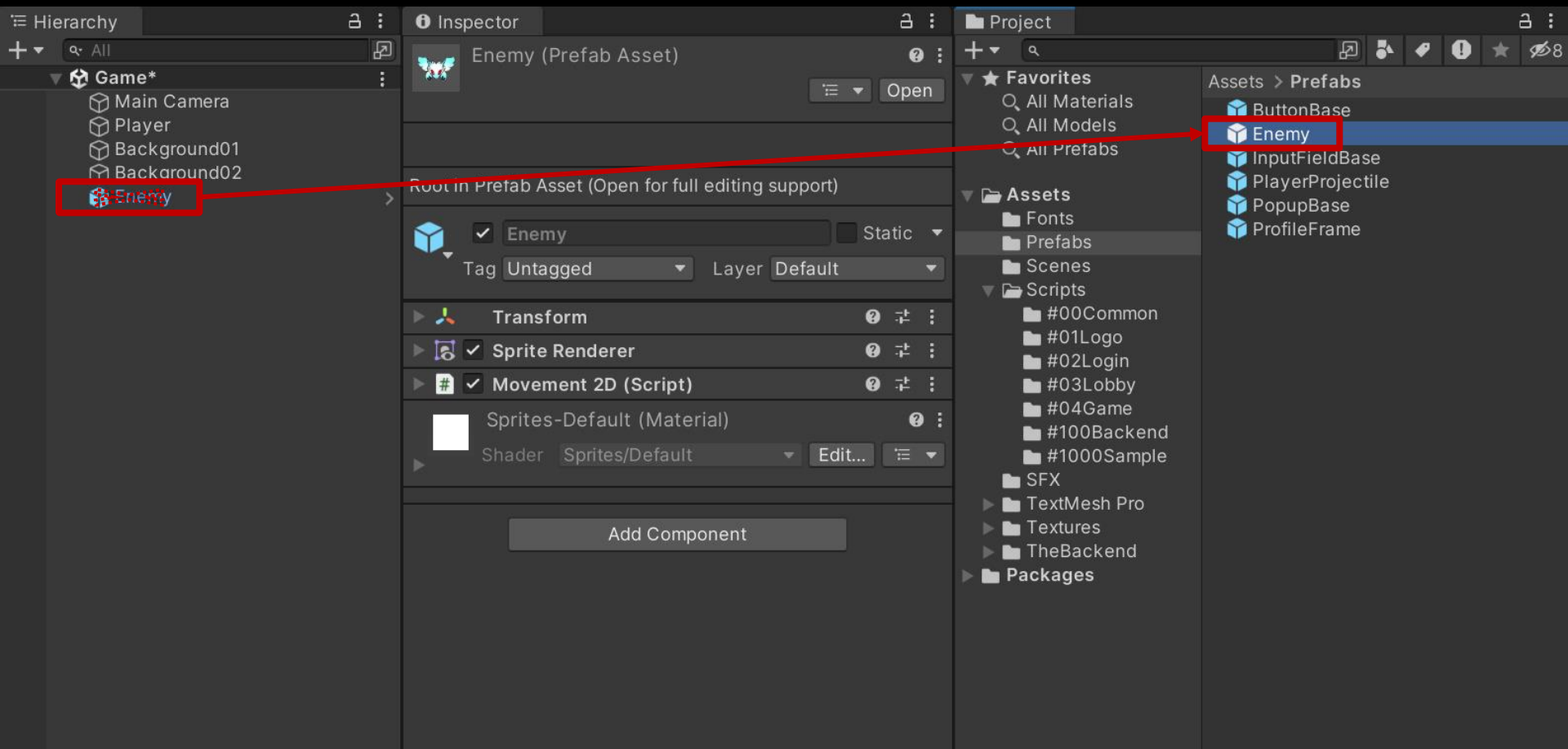
## ■ 결과 화면





# 게임 시스템 구현 I

- 적 오브젝트 Prefab 생성
  - Hierarchy View의 "Enemy" 오브젝트를 Project View로 Drag & Drop
  - Hierarchy View에 있는 "Enemy" 오브젝트 삭제





# 게임 시스템 구현 I

- 적 캐릭터 생성 제어를 위한 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "EnemySpawner"로 변경

```
1 using System.Collections;
2 using UnityEngine;
3
4 public class EnemySpawner : MonoBehaviour
5 {
6     [SerializeField]
7     private StageData    stageData;           // 적 생성을 위한 스테이지 크기 정보
8     [SerializeField]
9     private GameObject   enemyPrefab;        // 복제해서 생성할 적 캐릭터 프리팹
10    [SerializeField]
11    private float         spawnCycleTime;     // 생성 주기
12
13    private void Awake()
14    {
15        StartCoroutine(nameof(Process));
16    }
17
```





# 게임 시스템 구현 I

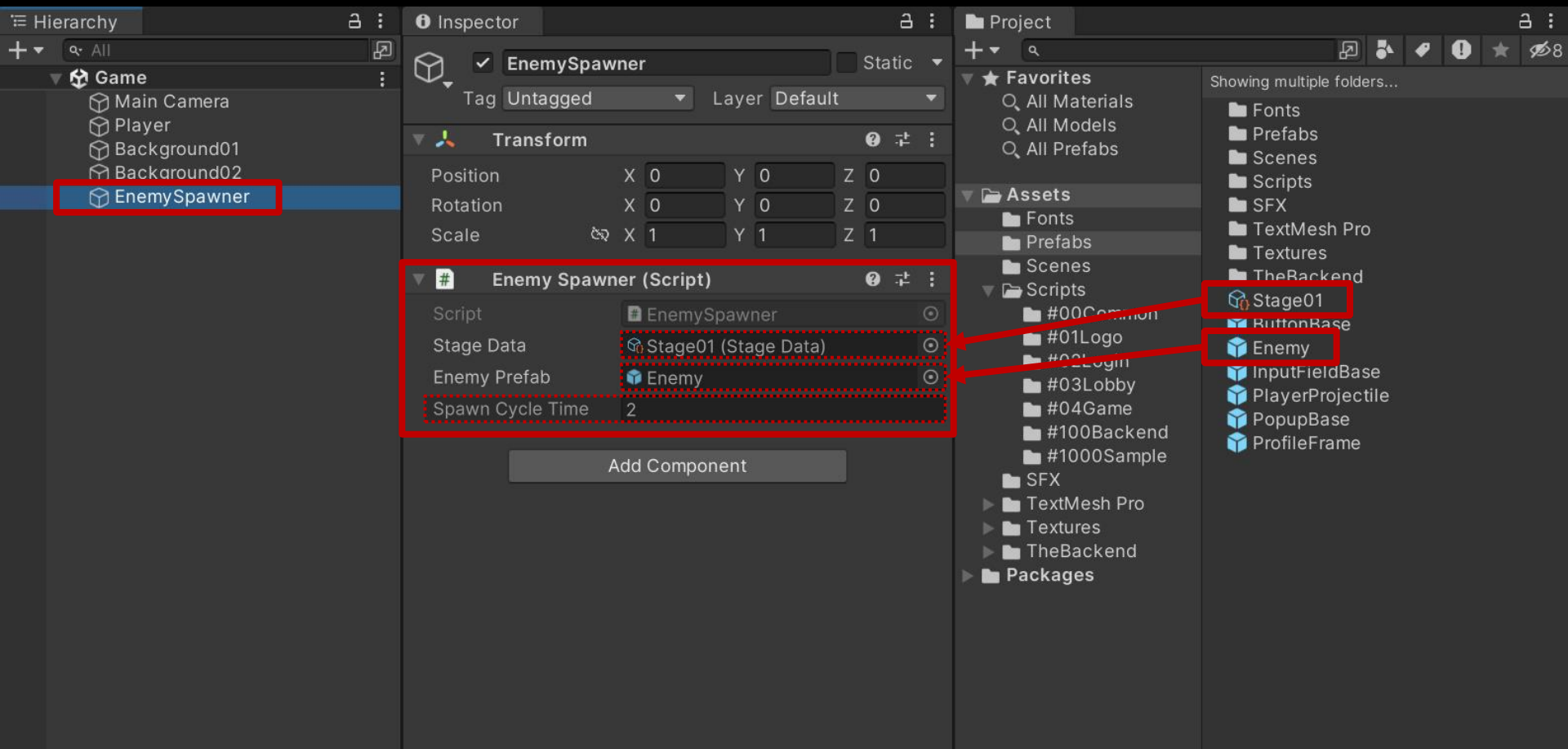
- 적 캐릭터 생성 제어를 위한 스크립트 생성 및 작성 (계속)

```
18 private IEnumerator Process()  
19 {  
20     int     enemyCount = 5;      // 한번에 생성하는 적 숫자  
21     float   distance = 1.2f;    // 생성되는 적 사이의 거리  
22     float   firstX = -2.4f;     // 첫 번째 적의 생성 위치 (왼쪽 끝)  
23  
24     while ( true )  
25     {  
26         for ( int i = 0; i < enemyCount; ++ i )  
27         {  
28             Vector3 position = new Vector3(firstX + distance * i, stageData.LimitMax.y + 1, 0);  
29             Instantiate(enemyPrefab, position, Quaternion.identity);  
30         }  
31  
32         // spawnCycleTime 시간동안 대기  
33         yield return new WaitForSeconds(spawnCycleTime);  
34     }  
35 }  
36 }
```



# 게임 시스템 구현 I

- 빈 오브젝트를 생성하고, "EnemySpawner" 컴포넌트 추가 및 설정
  - GameObject - Create Empty (단축키 : Ctrl+Shift+'N')





# 게임 시스템 구현 I

## ■ 결과 화면





# 게임 시스템 구현 I

## ■ 운석 자동 생성

- 오브젝트(SpriteRenderer)를 깜빡이게 하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "FadeEffect"로 변경

```
1  using System.Collections;
2  using UnityEngine;
3
4  public class FadeEffect : MonoBehaviour
5  {
6      private SpriteRenderer  spriteRenderer;
7      private float          fadeTime = 0.1f;
8
9      private void Awake()
10     {
11         spriteRenderer = GetComponent<SpriteRenderer>();
12
13         StartCoroutine(nameof(TwinkleLoop));
14     }
15
```



# 게임 시스템 구현 I

- 오브젝트(SpriteRenderer)를 깜빡이게 하는 스크립트 생성 및 작성 (계속)

```
16 private IEnumerator TwinkleLoop()
17 {
18     while ( true )
19     {
20         // Alpha 값을 1에서 0으로 : Fade Out
21         yield return StartCoroutine(OnFade(1, 0));
22
23         // Alpha 값을 0에서 1로 : Fade In
24         yield return StartCoroutine(OnFade(0, 1));
25     }
26 }
27
28 private IEnumerator OnFade(float start, float end)
29 {
30     float current = 0;
31     float percent = 0;
32
33     // fadeTime 시간동안 while() 반복문 실행
34     while ( percent < 1 )
35     {
36         current += Time.deltaTime;
37         percent = current / fadeTime;
38
39         Color color = spriteRenderer.color;
40         color.a = Mathf.Lerp(start, end, percent);
41         spriteRenderer.color = color;
42
43         yield return null;
44     }
45 }
46 }
```

== built-in method ==

float result = Mathf.Lerp(start, end, percent);

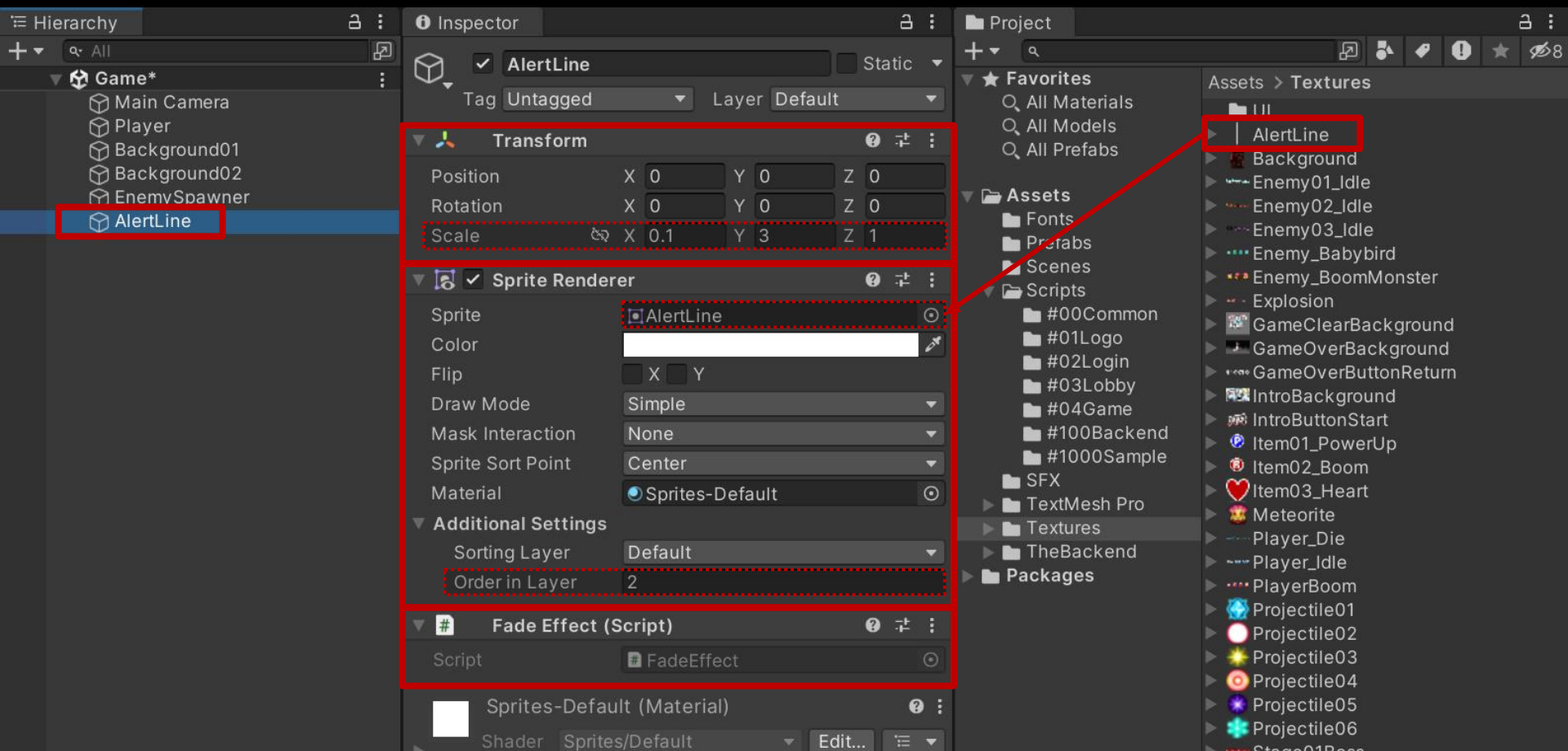
start와 end 사이의 값 중 percent 위치에 있는 값을 반환

예를 들어 start가 0, end가 100일 때 percent가 0.3이면 30을 반환



# 게임 시스템 구현 I

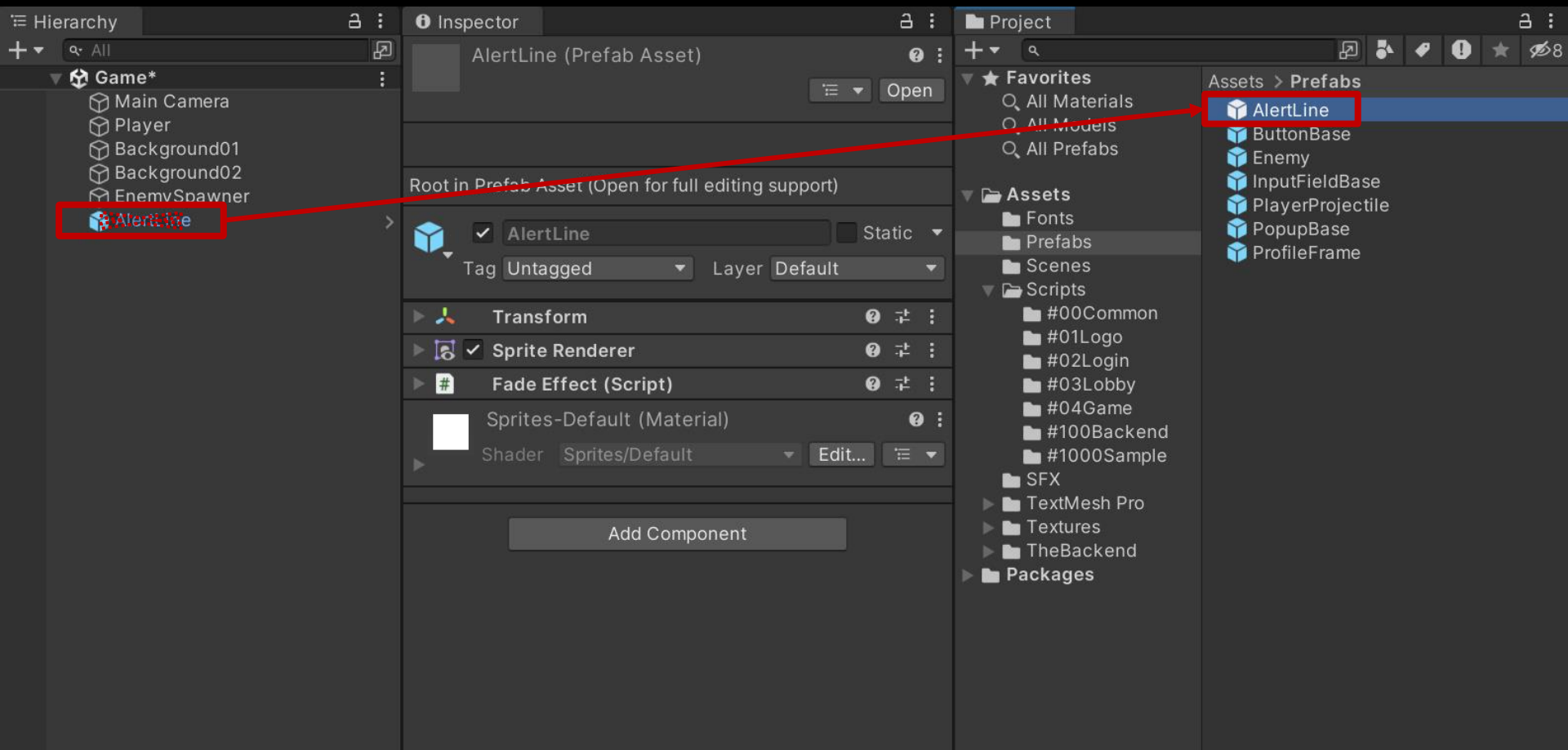
- 경고 선 오브젝트 생성 및 설정
  - GameObject - 2D Object - Sprites - Square





# 게임 시스템 구현 I

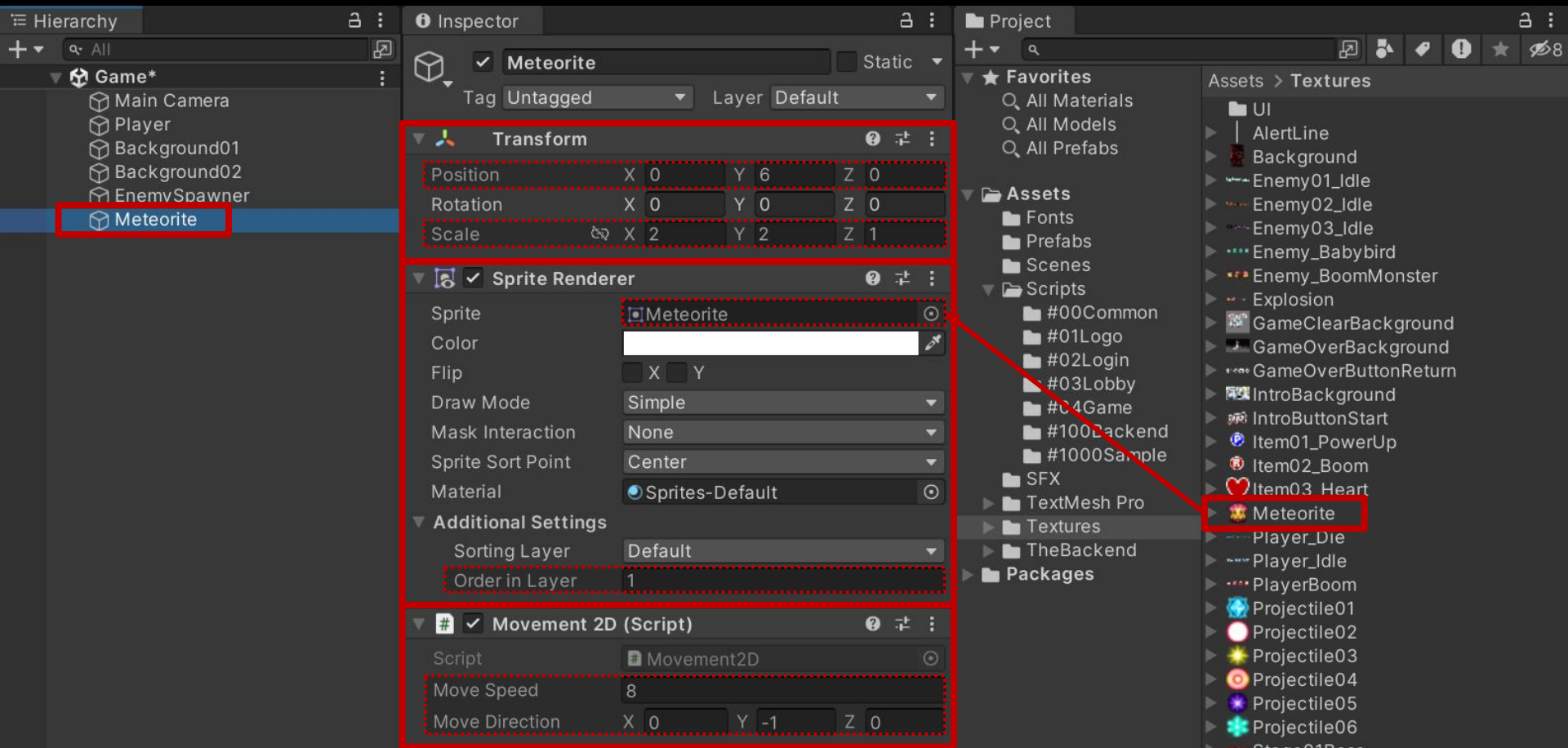
- 경고 선 오브젝트 Prefab 생성
  - Hierarchy View의 "AlertLine" 오브젝트를 Project View로 Drag & Drop
  - Hierarchy View에 있는 "AlertLine" 오브젝트 삭제





# 게임 시스템 구현 I

- 운석 오브젝트 생성 및 설정
  - GameObject - 2D Object - Sprites - Square

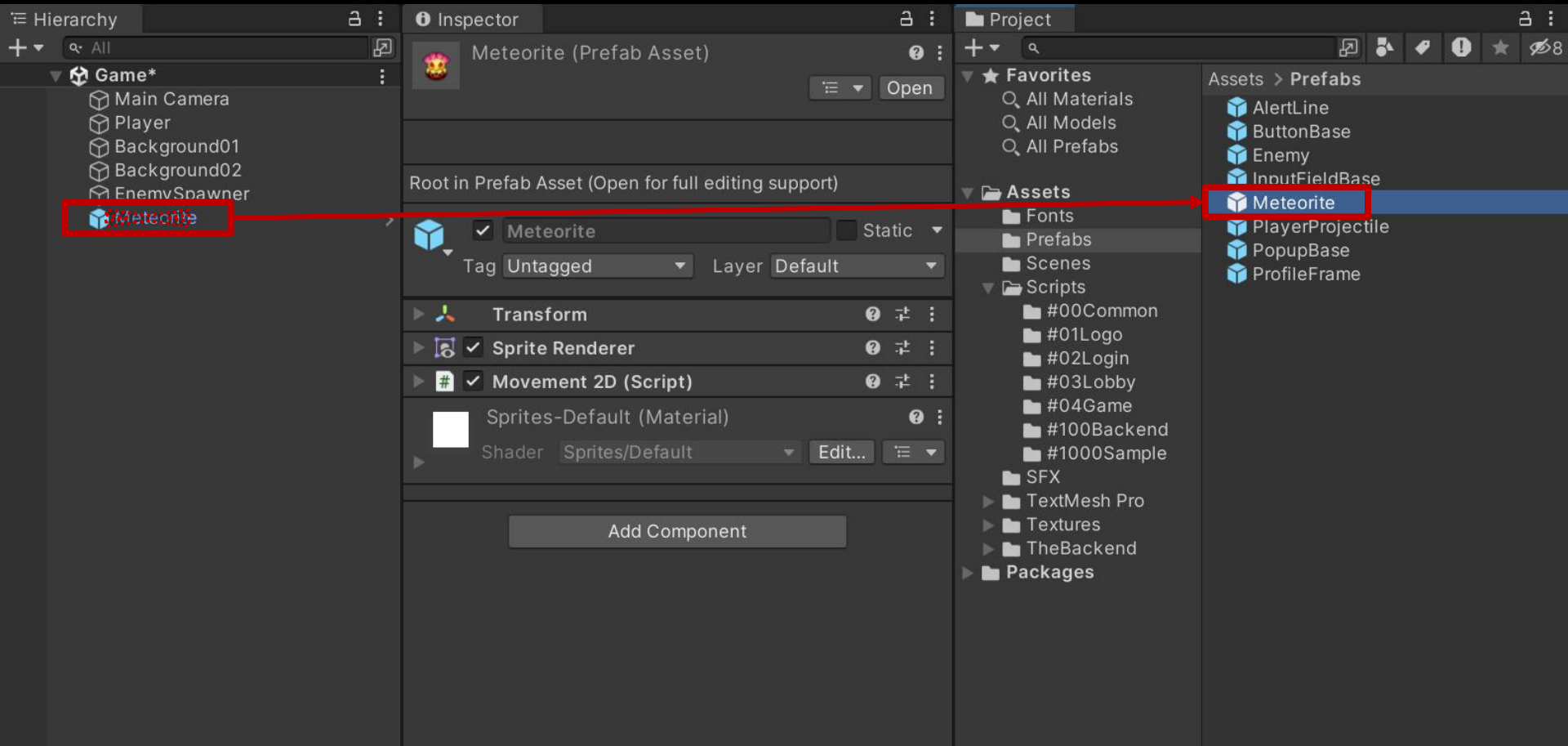






# 게임 시스템 구현 I

- 운석 오브젝트 Prefab 생성
  - Hierarchy View의 "Meteorite" 오브젝트를 Project View로 Drag & Drop
  - Hierarchy View에 있는 "Meteorite" 오브젝트 삭제





# 게임 시스템 구현 I

- 경고 선, 운석 생성을 제어하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "MeteoriteSpawner"로 변경

```
1  using System.Collections;
2  using UnityEngine;
3
4  public class MeteoriteSpawner : MonoBehaviour
5  {
6      [SerializeField]
7      private StageData    stageData;           // 경고선/운석 생성을 위한 스테이지 크기 정보
8      [SerializeField]
9      private GameObject   alertLinePrefab;    // 복제해서 생성할 경고선 프리팹
10     [SerializeField]
11     private GameObject   meteoritePrefab;    // 복제해서 생성할 운석 프리팹
12     [SerializeField]
13     private float        minSpawnCycleTime = 1; // 최소 생성 주기
14     [SerializeField]
15     private float        maxSpawnCycleTime = 4; // 최대 생성 주기
16
17     private void Awake()
18     {
19         StartCoroutine(nameof(Process));
20     }
21
```



# 게임 시스템 구현 I

## ■ 경고 선, 운석 생성을 제어하는 스크립트 생성 및 작성 (계속)

```
22 private IEnumerator Process()
23 {
24     while ( true )
25     {
26         // 대기 시간 설정 (minSpawnCycleTime ~ maxSpawnCycleTime)
27         float spawnCycleTime = Random.Range(minSpawnCycleTime, maxSpawnCycleTime);
28         // spawnCycleTime 시간동안 대기
29         yield return new WaitForSeconds(spawnCycleTime);
30
31         // 경고선/운석이 생성되는 x 위치는 스테이지 크기 범위 내에서 임의의 값을 선택
32         float x = Random.Range(stageData.LimitMin.x, stageData.LimitMax.x);
33
34         // 경고선 오브젝트 생성
35         GameObject alertLineClone = Instantiate(alertLinePrefab, new Vector3(x, 0, 0), Quaternion.identity);
36
37         // 1초 대기 후
38         yield return new WaitForSeconds(1);
39
40         // 경고선 오브젝트 삭제
41         Destroy(alertLineClone);
42
43         // 운석 오브젝트 생성 (y 위치는 스테이지 상단 위치 + 1)
44         Instantiate(meteoritePrefab, new Vector3(x, stageData.LimitMax.y + 1, 0), Quaternion.identity);
45     }
46 }
47 }
```

1. 1~4 초 동안 대기
2. 경고 선, 운석 x 위치 설정
3. 경고 선 생성 및 위치 설정
4. 1초 대기
5. 경고 선 삭제
6. 운석 생성 및 위치 설정

**== built-in method ==**

`int result = Random.Range(int min, int max);`

min ~ max-1 사이의 정수 중 임의의 정수를 result에 반환

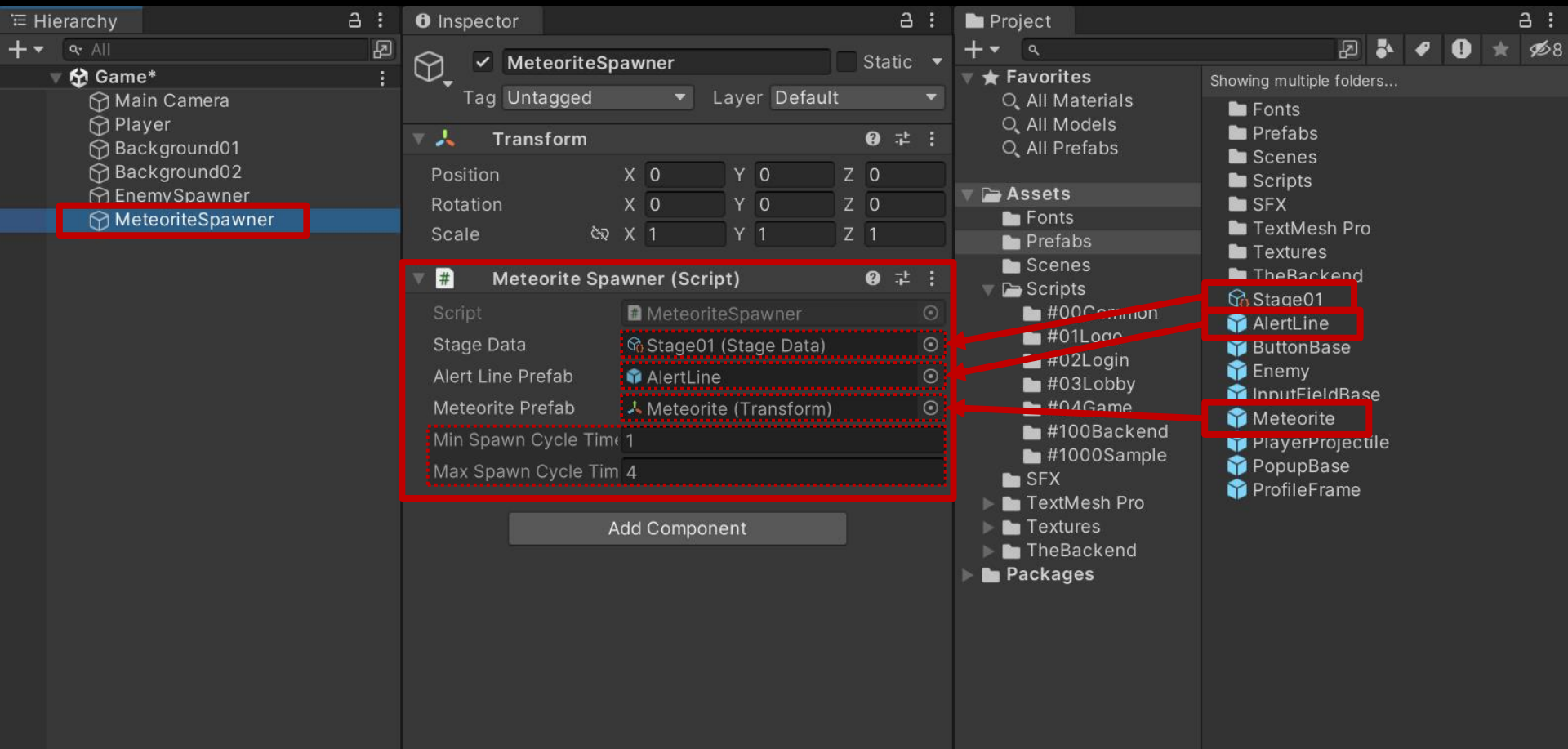
`float result = Random.Range(float min, float max);`

min ~ max 사이의 실수 중 임의의 실수를 result에 반환



# 게임 시스템 구현 I

- 빈 오브젝트를 생성하고, "MeteoriteSpawner" 컴포넌트 추가 및 설정
  - GameObject - Create Empty (단축키 : Ctrl+Shift+'N')





# 게임 시스템 구현 I

## ■ 결과 화면





# 게임 시스템 구현 I

## ■ 오브젝트 충돌 처리

### ■ 게임을 제어하는 스크립트 생성 및 작성

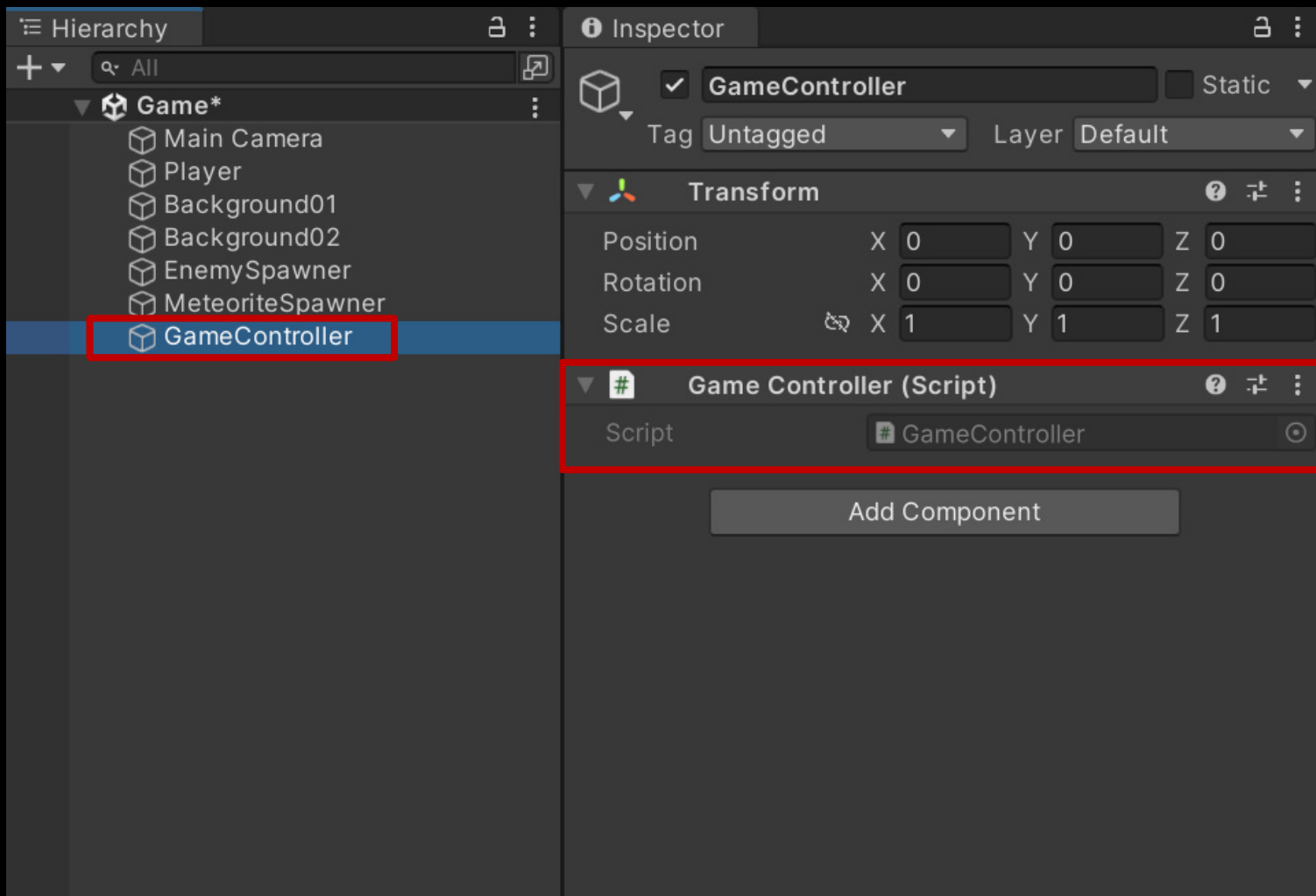
- C# Script 생성 후 스크립트의 이름을 "GameController"로 변경

```
1  using UnityEngine;
2
3  public class GameController : MonoBehaviour
4  {
5      public void GameOver()
6      {
7          // 로비 씬으로 이동
8          Utils.LoadScene(SceneNames.Lobby);
9      }
10 }
```



# 게임 시스템 구현 I

- 빈 오브젝트를 생성하고, "GameController" 컴포넌트 추가 및 설정
  - GameObject - Create Empty (단축키 : Ctrl+Shift+'N')





# 게임 시스템 구현 I

- Player 오브젝트에 "CircleCollider2D", "Rigidbody2D" 컴포넌트 추가 및 설정

The screenshot shows the Unity Inspector for the 'Player' object. The 'Circle Collider 2D' component is selected, and its settings are as follows:

- Material: None (Physics Material 2D)
- Is Trigger:
- Used By Effector:
- Offset: X 0, Y 0
- Radius: 0.25

The 'Rigidbody 2D' component is also selected, and its settings are as follows:

- Body Type: Dynamic
- Material: None (Physics Material 2D)
- Simulated:
- Use Auto Mass:
- Mass: 1
- Linear Drag: 0
- Angular Drag: 0.05
- Gravity Scale: 0
- Collision Detection: Discrete
- Sleeping Mode: Start Awake
- Interpolate: None
- Constraints: Freeze Position (X  Y ), Freeze Rotation (Z )



중력의 영향을 받아 아래로  
추락하지 않도록 Gravity Scale = 0





# 게임 시스템 구현 I

## ■ Player 오브젝트 태그 설정

The screenshot shows the Unity Hierarchy and Inspector panels. In the Hierarchy panel, the 'Player' object is selected. In the Inspector panel, the 'Tag' dropdown menu is open, and 'Player' is selected. A red box highlights the 'Tag' dropdown and the 'Player' option in the menu. A black text box with white text is overlaid on the Hierarchy panel.

Player 오브젝트의 Tag를  
기본으로 제공되는 "Player"로 설정



# 게임 시스템 구현 I

- 적의 충돌 처리 및 적 제어를 위한 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "Enemy"로 변경

```
1  using UnityEngine;
2
3  public class Enemy : MonoBehaviour
4  {
5      private GameController  gameController;
6
7      public void Setup(GameController gameController)
8      {
9          this.gameController = gameController;
10     }
11
12     private void OnTriggerEnter2D(Collider2D collision)
13     {
14         if ( collision.CompareTag("Player") )
15         {
16             gameController.GameOver();
17         }
18     }
19 }
```



# 게임 시스템 구현 I

- Enemy 프리팹에 "CircleCollider2D", "Enemy" 컴포넌트 추가 및 설정

**OnTriggerEnter2D() 메소드 호출을 위해 is Trigger = true**

**Inspector Panel:**

- Component: Enemy
- Tag: Untagged
- Layer: Default
- Transform
- Sprite Renderer
- Movement 2D (Script)
- Circle Collider 2D** (Highlighted):
  - Material: None (Physics Material 2D)
  - Is Trigger:**  (Highlighted with red dashed box)
  - Used By Effector:
  - Offset: X 0, Y 0
  - Radius: 0.35** (Highlighted with red dashed box)
  - Layer Overrides
  - Info
- Enemy (Script)** (Highlighted):
  - Script: Enemy
- Sprites-Default (Material)
- Shader: Sprites/Default

**Assets > Prefabs:**

- AlertLine
- ButtonBase
- Enemy** (Highlighted)
- Input-IdBase
- Meteorite
- PlayerProjectile
- PopupBase
- ProfileFrame

**Assets > Scripts:**

- #00Common
- #01Logo
- #02Login
- #03Lobby
- #04Game

**Assets > Sprites:**

- Te: [Image of a butterfly character with a red circle collider]



# 게임 시스템 구현 I

- 적을 생성할 때 Enemy.Setup() 메소드 호출
  - EnemySpawner Script 수정

```
1  using System.Collections;
2  using UnityEngine;
3
4  public class EnemySpawner : MonoBehaviour
5  {
6      [SerializeField]
7      private GameController  gameController;
8      [SerializeField]
9      private StageData      stageData;           // 적 생성을 위한 스테이지 크기 정보
10     [SerializeField]
11     private GameObject      enemyPrefab;        // 복제해서 생성할 적 캐릭터 프리팹
12     [SerializeField]
13     private float           spawnCycleTime;    // 생성 주기
14
15     private void Awake()...
19
```



# 게임 시스템 구현 I

## □ EnemySpawner Script 수정 (계속)

```
20 private IEnumerator Process()
21 {
22     int    enemyCount = 5;    // 한번에 생성하는 적 숫자
23     float  distance = 1.2f;  // 생성되는 적 사이의 거리
24     float  firstX = -2.4f;   // 첫 번째 적의 생성 위치 (왼쪽 끝)
25
26     while ( true )
27     {
28         for ( int i = 0; i < enemyCount; ++ i )
29         {
30             Vector3 position = new Vector3(firstX + distance * i, stageData.LimitMax.y + 1, 0);
31             GameObject enemy = Instantiate(enemyPrefab, position, Quaternion.identity);
32             enemy.GetComponent<Enemy>().Setup(gameController);
33         }
34
35         // spawnCycleTime 시간동안 대기
36         yield return new WaitForSeconds(spawnCycleTime);
37     }
38 }
39 }
```



# 게임 시스템 구현 I

- EnemySpawner 오브젝트의 "EnemySpawner" 컴포넌트 변수 설정

The screenshot displays the Unity Inspector interface. On the left, the Hierarchy panel shows a tree view under 'Game\*' with objects: Main Camera, Player, Background01, Background02, EnemySpawner (highlighted with a red box), MeteoriteSpawner, and GameController (highlighted with a red box). On the right, the Inspector panel shows the 'Enemy Spawner (Script)' component. Its variables are: Script (EnemySpawner), Game Controller (GameController, highlighted with a red dashed box and a red arrow from the Hierarchy panel), Stage Data (Stage01), Enemy Prefab (Enemy), and Spawn Cycle Time (2). An 'Add Component' button is visible at the bottom.



# 게임 시스템 구현 I

- 운석의 충돌 처리 및 운석 제어를 위한 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "Meteorite"로 변경

```
1  using UnityEngine;
2
3  public class Meteorite : MonoBehaviour
4  {
5      private GameController  gameController;
6
7      public void Setup(GameController gameController)
8      {
9          this.gameController = gameController;
10     }
11
12     private void OnTriggerEnter2D(Collider2D collision)
13     {
14         if ( collision.CompareTag("Player") )
15         {
16             gameController.GameOver();
17         }
18     }
19 }
```



# 게임 시스템 구현 I

- Meteorite 프리팹에 "CircleCollider2D", "Meteorite" 컴포넌트 추가 및 설정

**OnTriggerEnter2D() 메소드 호출을 위해 is Trigger = true**

**Inspector Panel:**

- Meteorite** (Static)
- Tag: Untagged, Layer: Default
- Transform
- Sprite Renderer
- Movement 2D (Script)
- Circle Collider 2D** (Highlighted with red box)
  - Edit Collider
  - Material: None (Physics Material 2D)
  - Is Trigger:  (Highlighted with red dashed box)
  - Used By Effector:
  - Offset: X 0, Y 0
  - Radius: 0.12 (Highlighted with red dashed box)
  - Layer Overrides
  - Info
- Meteorite (Script)** (Highlighted with red box)
  - Script: Meteorite
- Sprites-Default (Material)
- Shader: Sprites/Default

**Project Panel:**

- Assets > Prefabs
  - Meteorite (Highlighted with red box)
  - AlertLine
  - ButtonBase
  - Enemy
  - InputFieldBase
  - PlayerProjectile
  - PopupBase
  - ProfileFrame

**Assets > Scripts:**

- #00Common
- #01Logo
- #02Login
- #03Lobby
- #04Game
- #100E
- #100C
- SFX
- TextMes
- Textures
- TheBack
- Packages

**Assets > Textures:**

- Meteorite (Image with a red circle outline)





# 게임 시스템 구현 I

- 운석을 생성할 때 Meteorite.Setup() 메소드 호출
  - MeteoriteSpawner Script 수정

```
1 using System.Collections;
2 using UnityEngine;
3
4 public class MeteoriteSpawner : MonoBehaviour
5 {
6     [SerializeField]
7     private GameController gameController;
8     [SerializeField]
9     private StageData stageData; // 경고선/운석 생성을 위한 스테이지 크기 정보
10    [SerializeField]
11    private GameObject alertLinePrefab; // 복제해서 생성할 경고선 프리팹
12    [SerializeField]
13    private GameObject meteoritePrefab; // 복제해서 생성할 운석 프리팹
14    [SerializeField]
15    private float minSpawnCycleTime = 1; // 최소 생성 주기
16    [SerializeField]
17    private float maxSpawnCycleTime = 4; // 최대 생성 주기
18
19    private void Awake()...
23
```



# 게임 시스템 구현 I

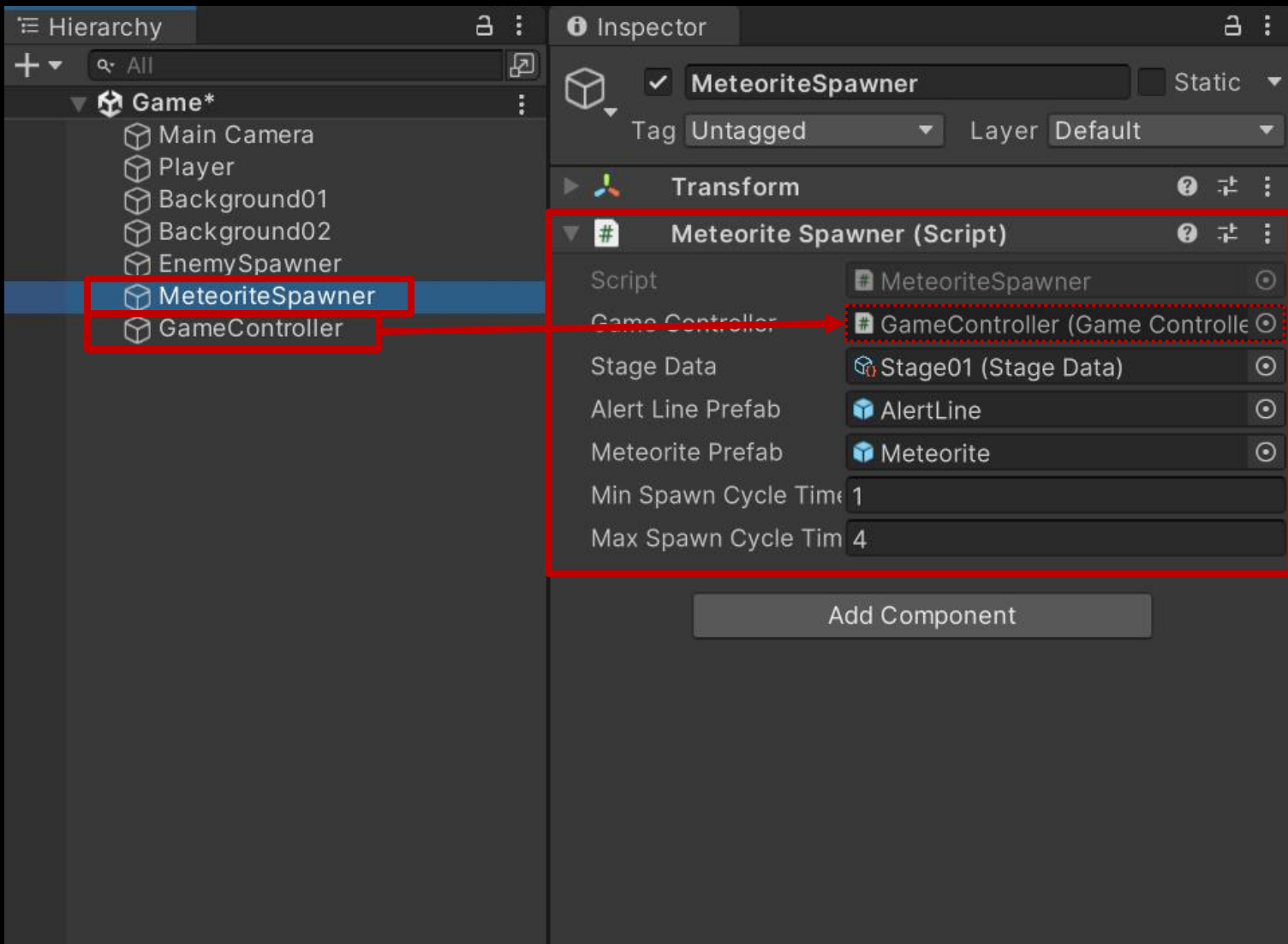
## □ MeteoriteSpawner Script 수정 (계속)

```
24 private IEnumerator Process()  
25 {  
26     while ( true )  
27     {  
28         // 대기 시간 설정 (minSpawnCycleTime ~ maxSpawnCycleTime)  
29         float spawnCycleTime = Random.Range(minSpawnCycleTime, maxSpawnCycleTime);  
30         // spawnCycleTime 시간동안 대기  
31         yield return new WaitForSeconds(spawnCycleTime);  
32  
33         // 경고선/운석이 생성되는 x 위치는 스테이지 크기 범위 내에서 임의의 값을 선택  
34         float x = Random.Range(stageData.LimitMin.x, stageData.LimitMax.x);  
35  
36         // 경고선 오브젝트 생성  
37         GameObject alertLineClone = Instantiate(alertLinePrefab, new Vector3(x, 0, 0), Quaternion.identity);  
38  
39         // 1초 대기 후  
40         yield return new WaitForSeconds(1);  
41  
42         // 경고선 오브젝트 삭제  
43         Destroy(alertLineClone);  
44  
45         // 운석 오브젝트 생성 (y 위치는 스테이지 상단 위치 + 1)  
46         GameObject meteorite = Instantiate(meteoritePrefab, new Vector3(x, stageData.LimitMax.y + 1, 0), Quaternion.identity);  
47         meteorite.GetComponent<Meteorite>().Setup(gameController);  
48     }  
49 }  
50 }
```



# 게임 시스템 구현 I

- MeteoriteSpawner 오브젝트의 "MeteoriteSpawner" 컴포넌트 변수 설정





# 게임 시스템 구현 I

- 플레이어 발사체 제어를 위한 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "PlayerProjectile"로 변경

```
1  using UnityEngine;
2
3  public class PlayerProjectile : MonoBehaviour
4  {
5      private void OnTriggerEnter2D(Collider2D collision)
6      {
7          if ( collision.CompareTag("Enemy") )
8          {
9              Destroy(collision.gameObject);
10             Destroy(gameObject);
11         }
12     }
13 }
```



# 게임 시스템 구현 I

## ■ PlayerProjectile 프리팹에 컴포넌트 추가 및 설정

The screenshot shows the Unity Inspector for a **PlayerProjectile** prefab. The components and their settings are as follows:

- Player Projectile (Script)**: Script assigned to **PlayerProjectile**.
- Circle Collider 2D**:
  - Material: None (Physics Material 2D)
  - Is Trigger:  (highlighted with a red dashed box)
  - Used By Effector:
  - Offset: X 0, Y 0
  - Radius: 0.1 (highlighted with a red dashed box)
- Rigidbody 2D**:
  - Body Type: Dynamic
  - Material: None (Physics Material 2D)
  - Simulated:
  - Use Auto Mass:
  - Mass: 1
  - Linear Drag: 0
  - Angular Drag: 0.05
  - Gravity Scale: 0 (highlighted with a red dashed box)

On the right, the **Assets > Prefabs** panel shows the **PlayerProjectile** prefab selected (highlighted with a red box).

On the bottom right, a preview of the **PlayerProjectile** prefab is shown, which is a blue, glowing, circular object with a complex, multi-layered design.

OnTriggerEnter2D() 메소드 호출을 위해 is Trigger = true

중력의 영향을 받아 아래로 추락하지 않도록 Gravity Scale = 0



# 게임 시스템 구현 I

## ■ 적 캐릭터 오브젝트 태그 생성 및 설정

The screenshot displays the Unity development environment. On the left, the Inspector panel shows the 'Enemy (Prefab Asset)' selected. Under the 'Tag' dropdown menu, 'Enemy' is selected and highlighted with a red box. The right panel shows the Project Hierarchy with 'Assets > Prefabs' expanded, and the 'Enemy' prefab highlighted with a blue selection bar and a red box. A black text box with white text is overlaid on the Hierarchy panel, containing the following instructions:

1. 아무 오브젝트나 선택하고 - Tag 메뉴 - Add Tag
2. Tags - "+" - "Enemy" 태그 생성
3. Enemy 프리팹의 태그를 "Enemy"로 설정



# 게임 시스템 구현 I

- 화면 밖으로 나가는 오브젝트를 삭제하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "DestroyByPosition"으로 변경

```
1  using UnityEngine;
2
3  public class DestroyByPosition : MonoBehaviour
4  {
5      [SerializeField]
6      private StageData    stageData;
7      private float        destroyWeight = 2;
8
9      private void LateUpdate()
10     {
11         if ( transform.position.y < stageData.LimitMin.y - destroyWeight ||
12             transform.position.y > stageData.LimitMax.y + destroyWeight ||
13             transform.position.x < stageData.LimitMin.x - destroyWeight ||
14             transform.position.x > stageData.LimitMax.x + destroyWeight )
15         {
16             Destroy(gameObject);
17         }
18     }
19 }
```



# 게임 시스템 구현 I

- Enemy, Meteorite, PlayerProjectile 프리팹에 컴포넌트 추가 및 설정

The screenshot displays the Unity Inspector and Hierarchy panels. In the Inspector, the 'Destroy By Position (Script)' component is selected, and its 'Stage Data' field is set to 'Stage01 (Stage Data)'. A red box highlights this field, with a red arrow pointing to the 'Stage01' entry in the Hierarchy panel. The Hierarchy panel shows a tree structure of assets, with 'Stage01', 'Enemy', 'Meteorite', and 'PlayerProjectile' highlighted in blue. A red box also highlights the 'Stage01' entry in the Hierarchy panel.

Inspector Panel:

- 3 Prefab Assets
- Root in Prefab Asset (Open for full editing support)
- Tag: — Layer: Default
- Transform
- Sprite Renderer
- Movement 2D (Script)
- Circle Collider 2D
- Destroy By Position (Script)**
  - Script: DestroyByPosition
  - Stage Data: **Stage01 (Stage Data)**
- Sprites-Default (Material)

Hierarchy Panel:

- Project
  - Favorites
    - All Materials
    - All Models
    - All Prefabs
  - Assets
    - Fonts
    - Prefabs
    - Scenes
    - Scripts
      - #00Common
      - #01Logo
      - #02Login
      - #03Lobby
      - #04Game
      - #100Backend
      - #1000Sample
      - SFX
      - TextMesh Pro
      - Textures
      - TheBackend
    - Packages

Showing multiple folders...

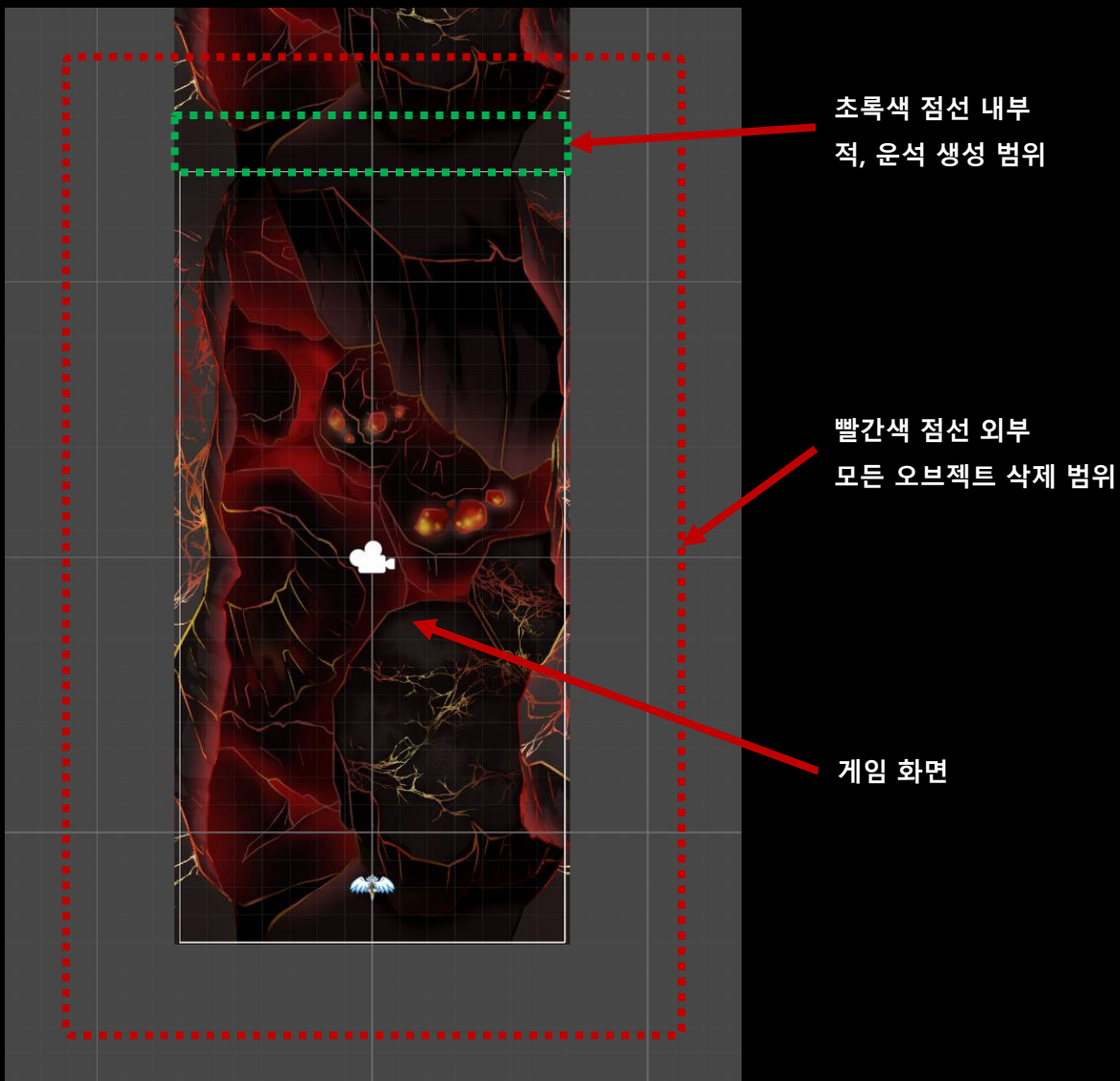
- Fonts
- Prefabs
- Scenes
- Scripts
- SFX
- TextMesh Pro
- Textures
- TheBackend
- Stage01**
- AlertLine
- ButtonBase
- Enemy**
- InputFieldBase
- Meteorite**
- PlayerProjectile**
- PopupBase
- ProfileFrame





# 게임 시스템 구현 I

- 적, 운석 생성 위치와 오브젝트 삭제 위치





# 게임 시스템 구현 I

## ■ 결과 화면



# 게임 정보 업데이트

- 게임오버 되었을 때 유저 경험치, 레벨 정보 갱신



# 게임 정보 업데이트

## ■ 게임오버 되었을 때 유저 경험치, 레벨 정보 갱신

UpdateV2() : 테이블에 저장되어 있는 값 중 inDate 컬럼의 값과 소유하는 유저의 owner\_inDate가 일치하는 row를 검색하여 수정

스키마 정의/미정의 여부 관계없이 데이터를 수정할 수 있다.

자기 자신의 public/private 데이터를 수정할 수 있다.

타인의 public 데이터를 수정할 수 있다.

타인의 private 데이터를 수정할 수 없다.

숫자형 데이터의 경우, 최대 9007199254740991(약 9000조)까지 안전하게 값을 넣을 수 있다. 그 이상의 데이터를 삽입할 경우, 일의 자리, 십의 자리 수는 0으로 내림처리가 된다. 숫자가 증가하여 자리 수가 많아질수록 작은 수는 내림처리가 되며 큰 수는 왼쪽 기준으로 16~18자리의 수까지 값이 유지된다.

param에 row 내 일부 컬럼만 존재할 경우 해당 컬럼만 수정된다.

param에 존재하지 않는 컬럼이 삭제되지 않는다.

param에 row 내 존재하지 않는 컬럼이 존재할 경우

스키마 미정의 테이블의 경우 row에 존재하지 않는 컬럼을 update 한 경우 새로운 컬럼이 추가

스키마 정의 테이블의 경우 스키마를 선언하지 않은 컬럼을 update 한 경우 에러 발생



# 게임 정보 업데이트

- 게임 정보를 업데이트하는 GameDataUpdate() 메소드 정의
  - BackendGameData Script 수정

```
1  using UnityEngine;
2  using Backend;
3  using UnityEngine.Events;
4
5  public class BackendGameData
6  {
7      [System.Serializable]
8      public class GameDataLoadEvent : /*UnityEngine.Events.* /UnityEvent { }
9      public GameDataLoadEvent onGameDataLoadEvent = new GameDataLoadEvent();
10
11     private static BackendGameData instance = null;
12     public static BackendGameData Instance...
13
14
15
16
17
18
19     private UserData userData = new UserData();
20     public UserData UserData => userData;
21
22
23
24
25     private string gameDataRowInDate = string.Empty;
26
27
28     /// <summary> 뒤끝 콘솔 테이블에 새로운 유저 정보 추가
29     public void GameDataInsert()...
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67     /// <summary> 뒤끝 콘솔 테이블에서 유저 정보를 불러올 때 호출
68     public void GameDataLoad()...
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
```



# 게임 정보 업데이트

- BackendGameData Script 수정 (계속)

```
120  /// <summary>
121  /// 뒤끝 콘솔 테이블에 있는 유저 데이터 갱신
122  /// </summary>
123  public void GameDataUpdate(UnityAction action=null)
124  {
125      if ( userGameData == null )
126      {
127          Debug.LogError("서버에서 다운받거나 새로 삽입한 데이터가 존재하지 않습니다." +
128                          "Insert 혹은 Load를 통해 데이터를 생성해주세요.");
129          return;
130      }
131
132      Param param = new Param()
133      {
134          { "level",      userGameData.level },
135          { "experience", userGameData.experience },
136          { "gold",      userGameData.gold },
137          { "jewel",    userGameData.jewel },
138          { "heart",    userGameData.heart }
139      };
140
```



# 게임 정보 업데이트

## BackendGameData Script 수정 (계속)

```
141 // 게임 정보의 고유값(gameDataRowInDate)이 없으면 에러 메시지 출력
142 if ( string.IsNullOrEmpty(gameDataRowInDate) )
143 {
144     Debug.LogError($"유저의 inDate 정보가 없어 게임 정보 데이터 수정에 실패했습니다.");
145 }
146 // 게임 정보의 고유값이 있으면 테이블에 저장되어 있는 값 중 inDate 컬럼의 값과
147 // 소유하는 유저의 owner_inDate가 일치하는 row를 검색하여 수정하는 UpdateV2() 호출
148 else
149 {
150     Debug.Log($"{gameDataRowInDate}의 게임 정보 데이터 수정을 요청합니다.");
151
152     Backend.GameData.UpdateV2("USER_DATA", gameDataRowInDate, Backend.UserInDate, param, callback =>
153     {
154         if ( callback.IsSuccess() )
155         {
156             Debug.Log($"게임 정보 데이터 수정에 성공했습니다. : {callback}");
157
158             action?.Invoke();
159         }
160         else
161         {
162             Debug.LogError($"게임 정보 데이터 수정에 실패했습니다. : {callback}");
163         }
164     });
165 }
166 }
167 }
```



# 게임 정보 업데이트

- 게임오버 되었을 때 유저의 경험치/레벨 정보를 갱신하고 씬 전환
  - GameController Script 수정

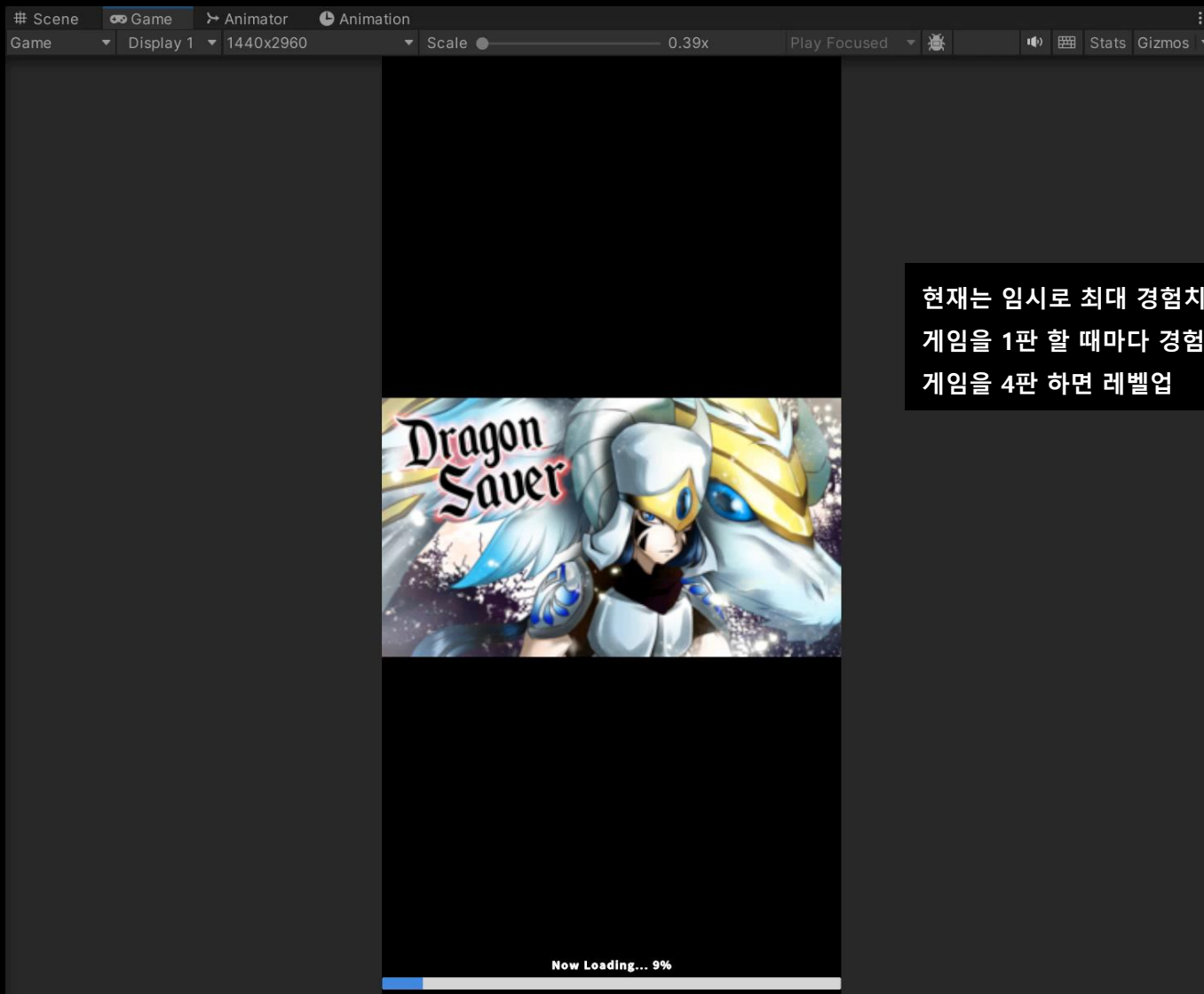
```
1 using UnityEngine;
2
3 public class GameController : MonoBehaviour
4 {
5     public bool IsGameOver { set; get; } = false;
6
7     public void GameOver()
8     {
9         // 중복 처리 되지 않도록 bool 변수로 제어
10        if ( IsGameOver == true ) return;
11
12        IsGameOver = true;
13
14        // 경험치 증가 및 레벨업 여부 검사
15        // (현재 레벨 시스템에 대한 설정이 없기 때문에 경험치의 최대치를 100으로 가정)
16        // (게임을 한번 플레이할 때마다 경험치는 25씩 증가)
17        BackendGameData.Instance.UserGameData.experience += 25;
18        if ( BackendGameData.Instance.UserGameData.experience >= 100 )
19        {
20            BackendGameData.Instance.UserGameData.experience = 0;
21            BackendGameData.Instance.UserGameData.level ++;
22        }
23
24        // 게임 정보 업데이트
25        BackendGameData.Instance.GameDataUpdate(AfterGameOver);
26    }
27
28    public void AfterGameOver()
29    {
30        // 로비 씬으로 이동
31        Utils.LoadScene(SceneNames.Lobby);
32    }
33 }
```





# 게임 정보 업데이트

## ■ 결과 화면



현재는 임시로 최대 경험치는 100,  
게임을 1판 할 때마다 경험치 +25로  
게임을 4판 하면 레벨업