Unity Backend

“랭킹” 기능을 이용한 유저 랭킹 관리 (랭킹 등록)

Created in 2023-06-07

Last Updated 2023-06-09

Unity Version 2022.2.2f1

Index

- 인게임 점수 구현
- 게임오버 UI 구현
- 랭킹으로 사용할 유저 정보 데이터 추가
- 랭킹 데이터 등록

인게임 점수 구현

- 인게임 점수 관리
- 인게임 점수 출력
- 적, 운석 폭발 효과

인게임 점수 구현

인게임 점수 관리

- 점수를 관리하는 변수와 프로퍼티 정의
  - GameController Script 수정

```
1   using UnityEngine;
2
3   public class GameController : MonoBehaviour
4   {
5       private int     score = 0;
6
7       public  bool    IsGameOver { set; get; } = false;
8       public  int     Score
9       {
10          set => score = Mathf.Max(0, value);
11          get => score;
12      }
13
14      public void GameOver()...
34
35      public void AfterGameOver()...
40  }
```

인게임 점수 구현

- 적을 처치했을 때 호출하는 OnDie() 메소드 정의
  - Enemy Script 수정

```
1   using UnityEngine;
2
3   public class Enemy : MonoBehaviour
4   {
5       [SerializeField]
6       private int             scorePoint = 100;       // 적을 처치했을 때 획득하는 점수
7       private GameController  gameController;
8
9       public void Setup(GameController gameController)...
13
14      public void OnDie()
15      {
16          // 플레이어의 점수를 scorePoint만큼 증가
17          gameController.Score += scorePoint;
18          // 적 캐릭터 삭제
19          Destroy(gameObject);
20      }
21
22      private void OnTriggerEnter2D(Collider2D collision)...
29  }
```

인게임 점수 구현

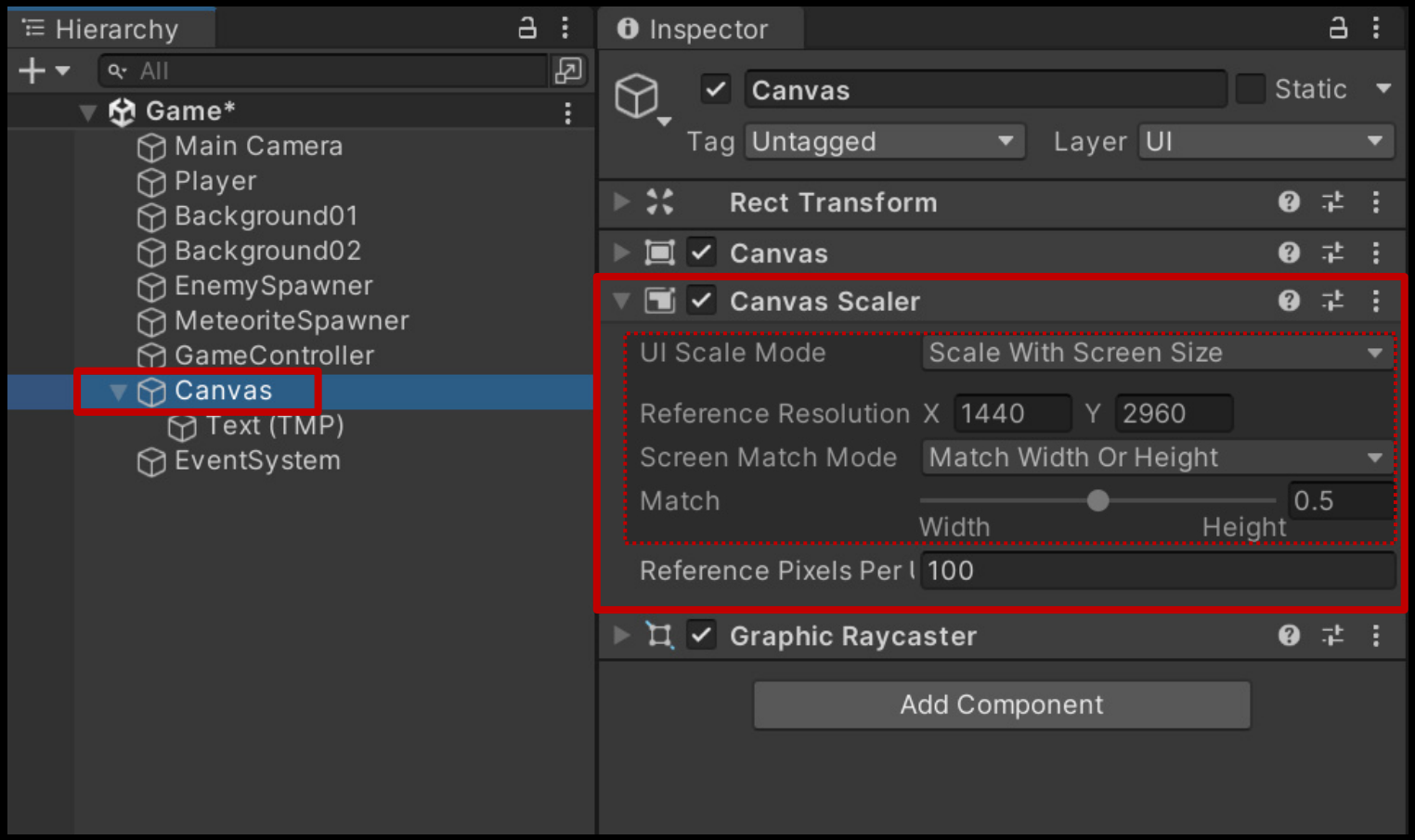
- 플레이어 발사체와 충돌한 적을 바로 삭제하지 않고 OnDie() 메소드 호출
  - PlayerProjectile Script 수정

```
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              collision.GetComponent<Enemy>().OnDie();
11
12              Destroy(gameObject);
13          }
14      }
15  }
```

인게임 점수 구현

인게임 점수 출력

- 인게임 획득 점수를 출력하는 "Text - TextMeshPro" UI 생성 및 설정
  - GameObject - UI - "Text - TextMeshPro"



인게임 점수 구현

- 인게임 획득 점수를 출력하는 "Text - TextMeshPro" UI 생성 및 설정 (계속)

Hierarchy
All
Game
Main Camera
Player
Background01
Background02
EnemySpawner
MeteoriteSpawner
GameController
Canvas
PlayerScore
EventSystem

Inspector
PlayerScore
Static
Tag Untagged
Layer UI
Rect Transform
stretch
top
Left 25
Pos Y -20
Pos Z 0
Right 0
Height 50
Anchors
Min X 0 Y 1
Max X 1 Y 1
Pivot X 0.5 Y 1
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Canvas Renderer
TextMeshPro - Text (UI)
NotoSansKR-Bold SDF Material (Material)
Shader TextMeshPro/Distance Fi
Edit...
Add Component

TextMeshPro - Text (UI)
Text Input
Enable RTL Editor
SCORE 00000
Text Style Normal
Main Settings
Font Asset NotoSansKR-Bold SDF (TMP_Fo
Material Preset NotoSansKR-Bold SDF Material
Font Style B I U S ab AB SC
Font Size 50
Auto Size
Vertex Color
Color Gradient
Override Tags
Spacing Options (em) Character 0 Word 0
Line 0 Paragraph 0
Alignment
Wrapping Enabled
Overflow Overflow
Horizontal Mapping Character
Vertical Mapping Character

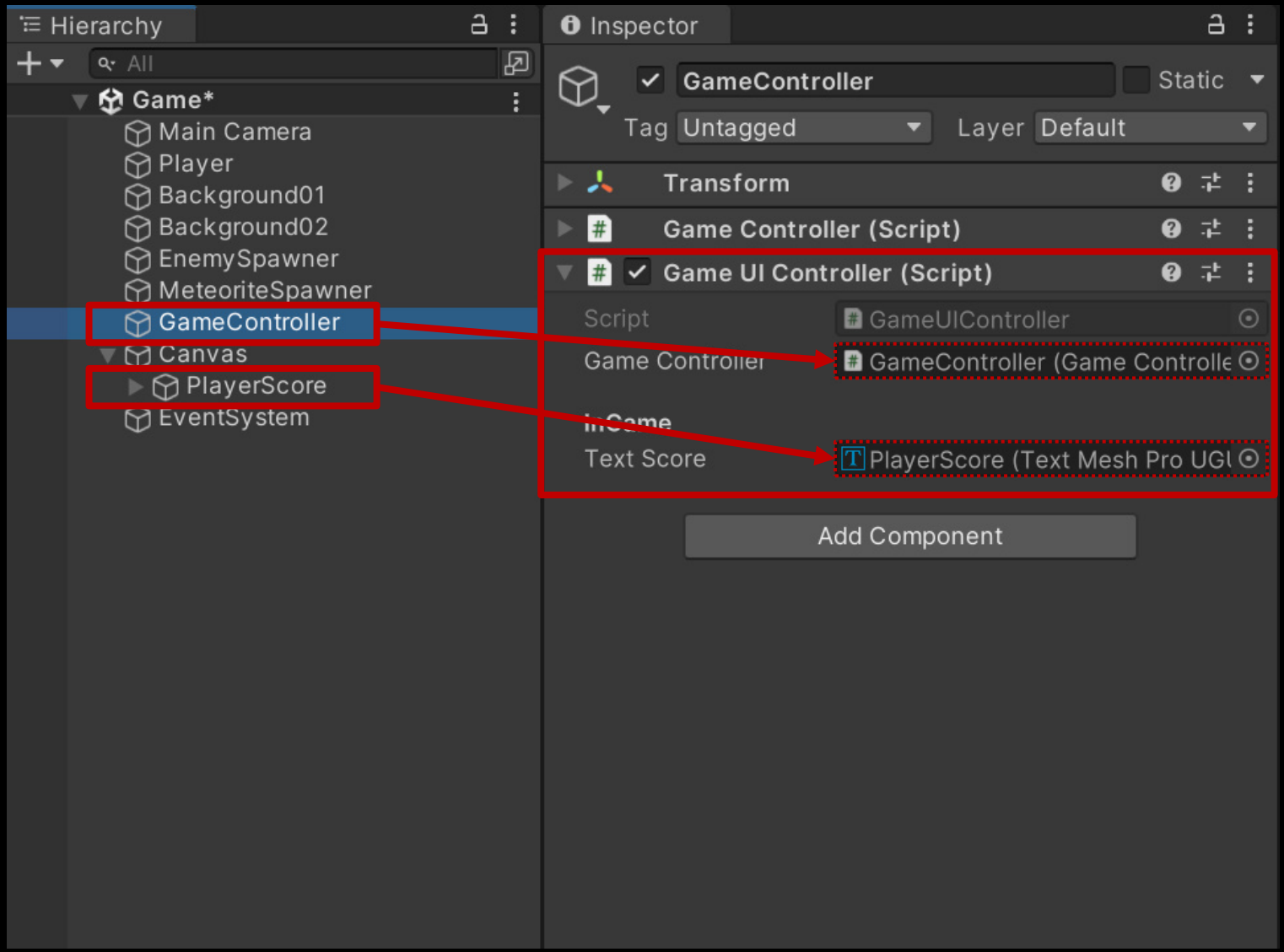
인게임 점수 구현

- 인게임, 게임오버 UI들을 제어하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "GameUIController"로 변경

```
1  using UnityEngine;
2  using TMPro;
3
4  public class GameUIController : MonoBehaviour
5  {
6      [SerializeField]
7      private GameController  gameController;
8
9      [Header("InGame")]
10     [SerializeField]
11     private TextMeshProUGUI textScore;
12
13     private void Update()
14     {
15         textScore.text = $"SCORE {gameController.Score}";
16     }
17 }
```

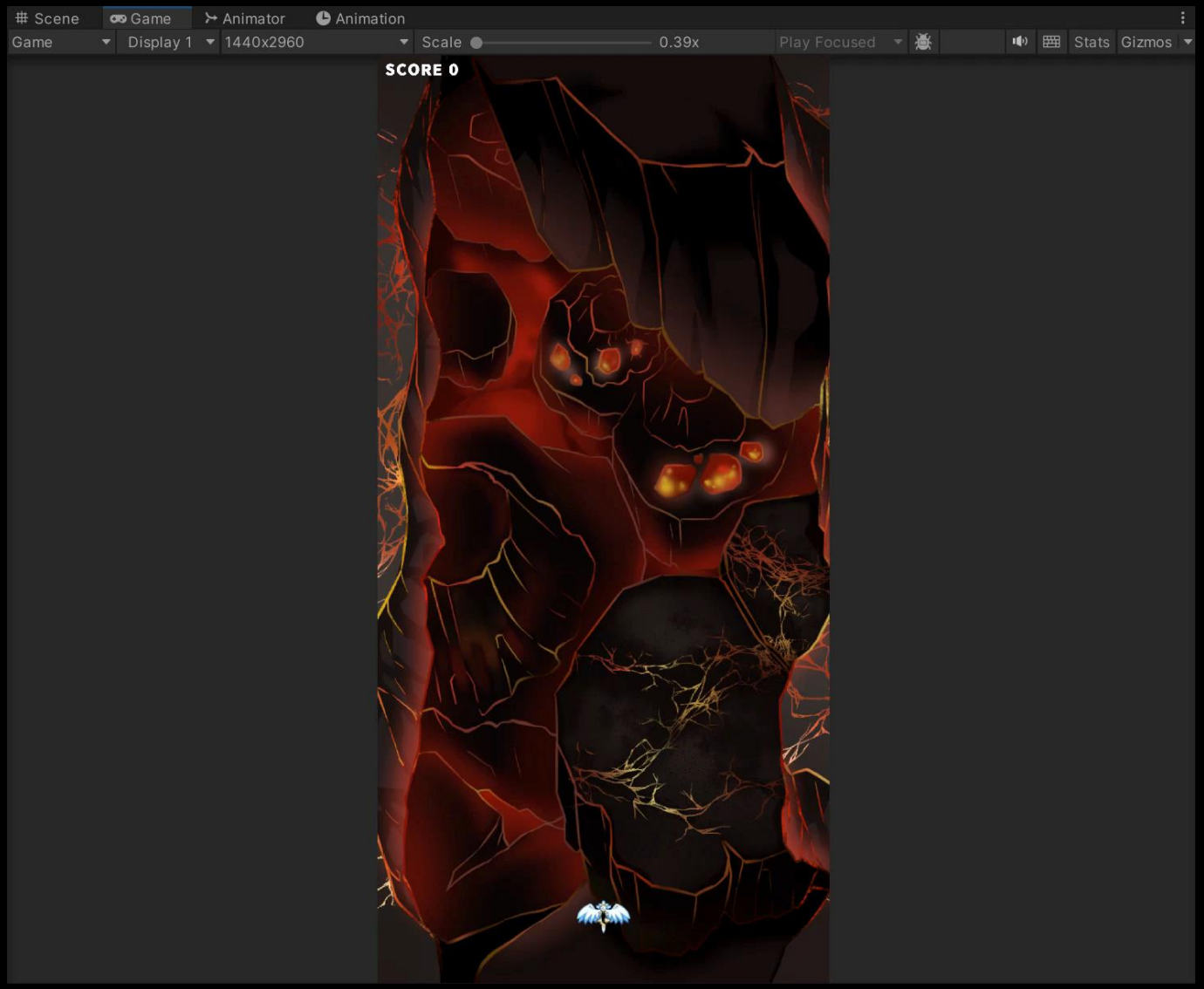
인게임 점수 구현

- GameController 오브젝트에 "GameUIController" 컴포넌트 추가 및 설정



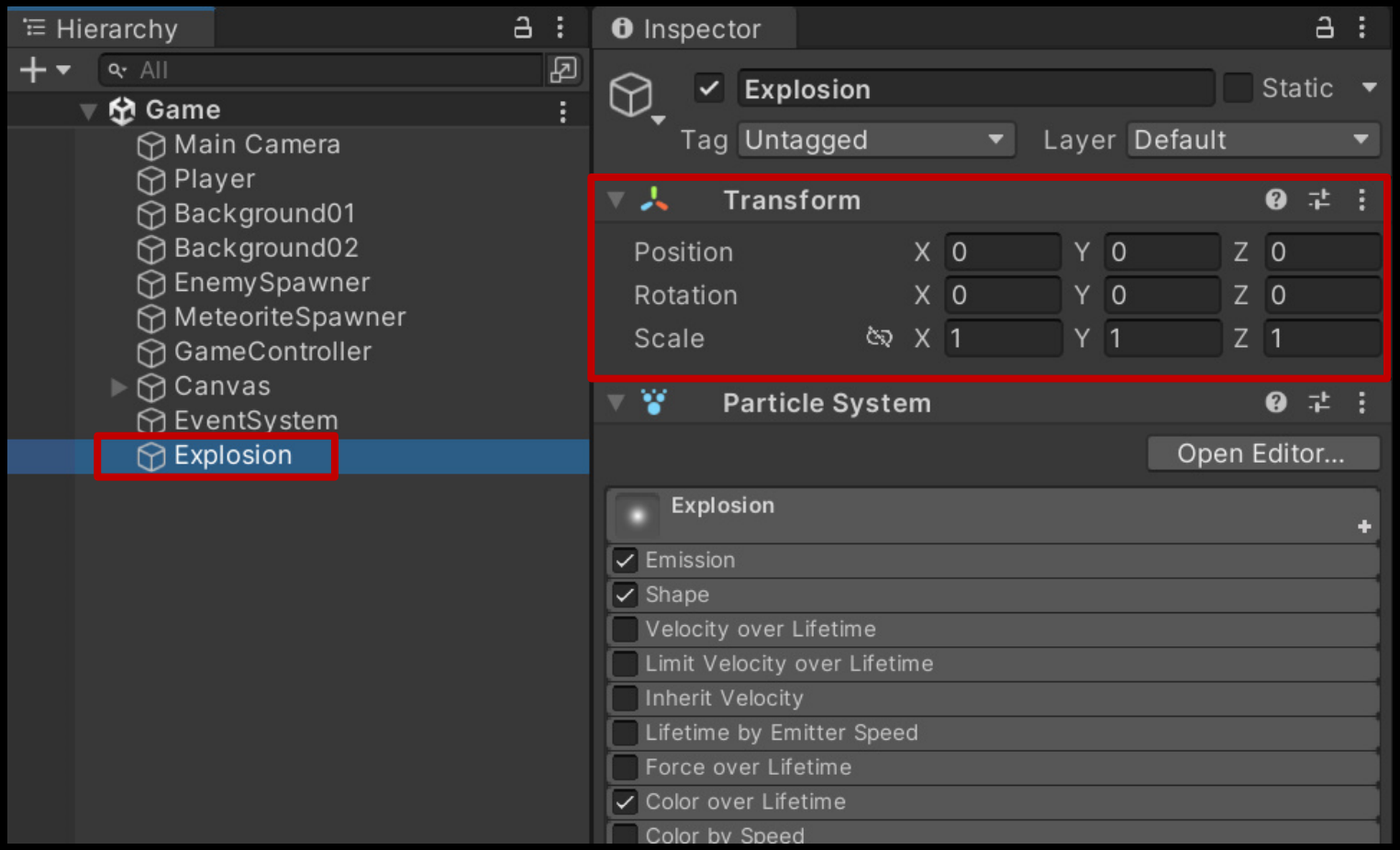
인게임 점수 구현

- 결과 화면



인게임 점수 구현

- 적, 운석 폭발 효과
  - 폭발 이펙트 생성
    - GameObject - Effects - Particle System



인게임 점수 구현

- 폭발 이펙트 설정

Explosion

Duration	0.5
Looping	☐
Prewarm	☐
Start Delay	0
Start Lifetime	0.2
Start Speed	1
3D Start Size	☐
Start Size	0.2 ~ 0.5
3D Start Rotation	
Start Rotation	
Flip Rotation	0
Start Color	
Gravity Source	3D Physics
Gravity Modifier	0
Simulation Space	Local
Simulation Speed	1
Delta Time	Scaled
Scaling Mode	Local
Play On Awake*	☑
Emitter Velocity Mode	Rigidbody
Max Particles	20
Auto Random Seed	☑
Stop Action	None
Culling Mode	Automatic
Ring Buffer Mode	Disabled

Random Between Two Constants 선택

Emission

Rate over Time	0
Rate over Distance	0

Bursts

Time	Count	Cycles	Interval	Probability
0.000	20	Cy 1	0.010	1.00

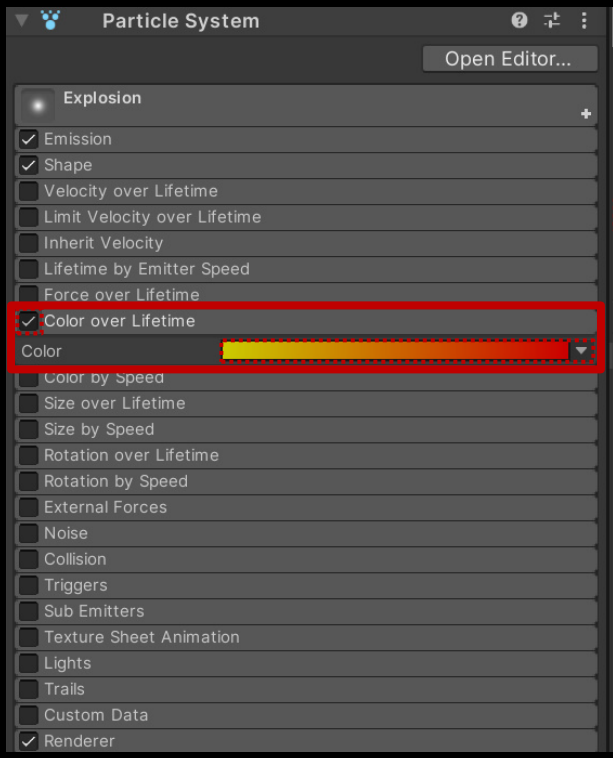
Shape

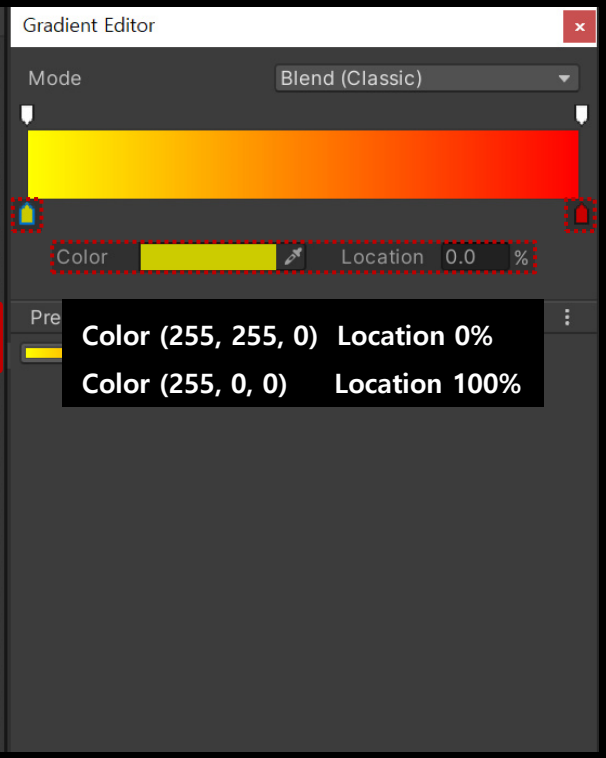
Shape	Circle
Radius	0.1
Radius Thickness	0.1
Arc	360
Mode	Random
Spread	0
Texture	None (Texture 2D)
Position	X 0 Y 0 Z 0
Rotation	X 0 Y 0 Z 0
Scale	X 1 Y 1 Z 1
Align To Direction	☐
Randomize Direction	0
Spherize Direction	0
Randomize Position	0

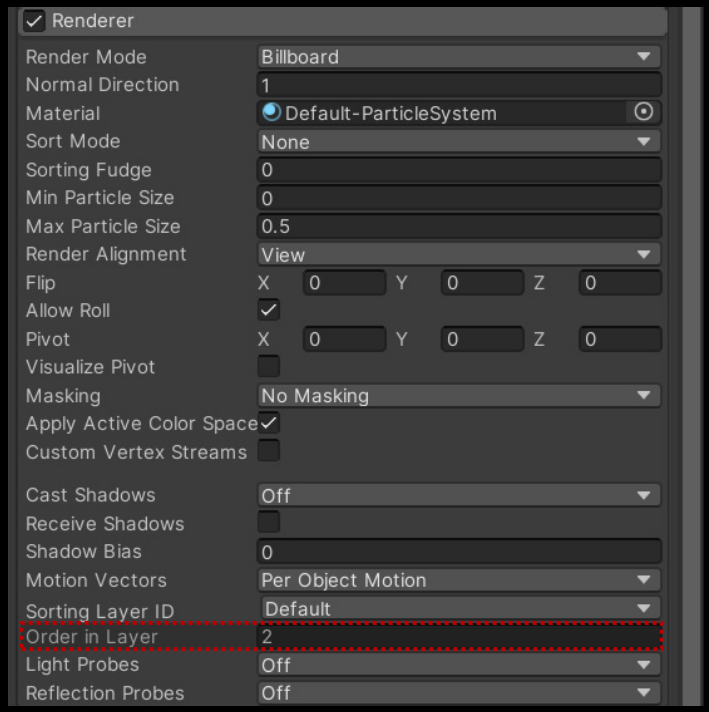
Scene Tools

인게임 점수 구현

- 폭발 이펙트 설정 (계속)







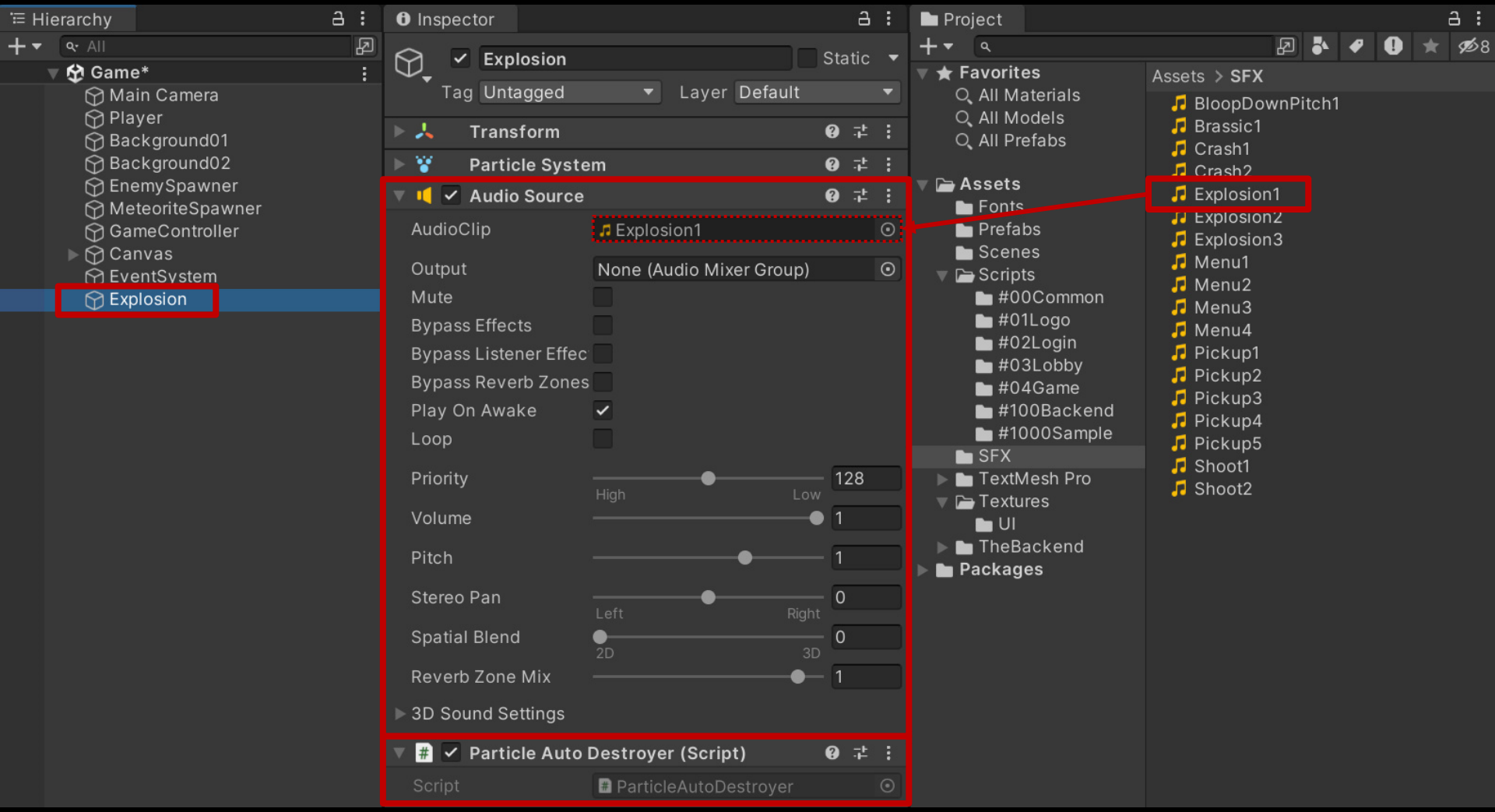
인게임 점수 구현

- 파티클의 재생이 완료되면 오브젝트를 삭제하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "ParticleAutoDestroyer"로 변경

```
1  using UnityEngine;
2
3  public class ParticleAutoDestroyer : MonoBehaviour
4  {
5      private ParticleSystem  particle;
6
7      private void Awake()
8      {
9          particle = GetComponent<ParticleSystem>();
10     }
11
12     private void Update()
13     {
14         // 파티클 재생이 완료되면 삭제
15         if ( particle.isPlaying == false )
16         {
17             Destroy(gameObject);
18         }
19     }
20 }
```

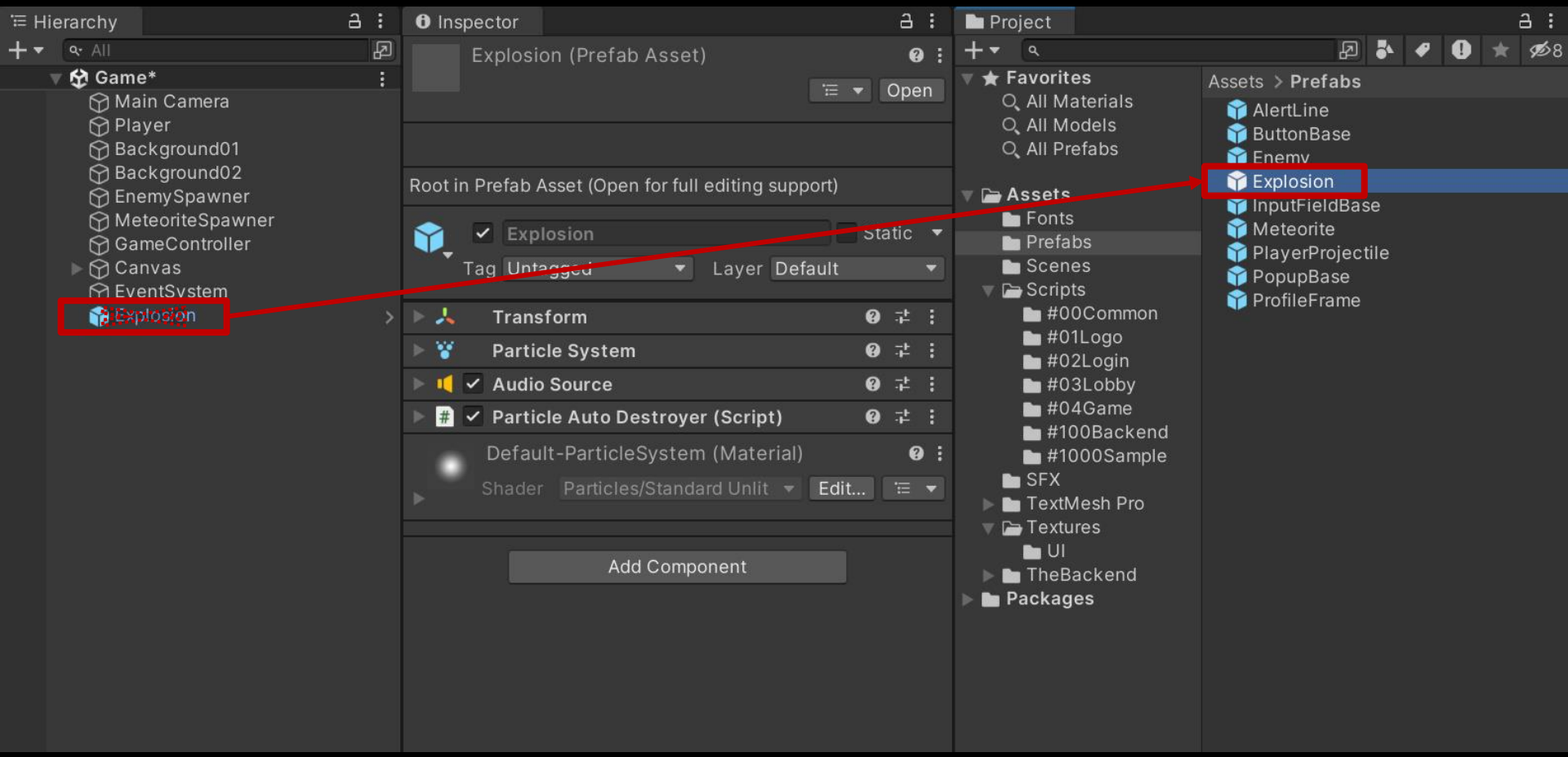
인게임 점수 구현

- Explosion 오브젝트에 컴포넌트 추가 및 설정



인게임 점수 구현

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



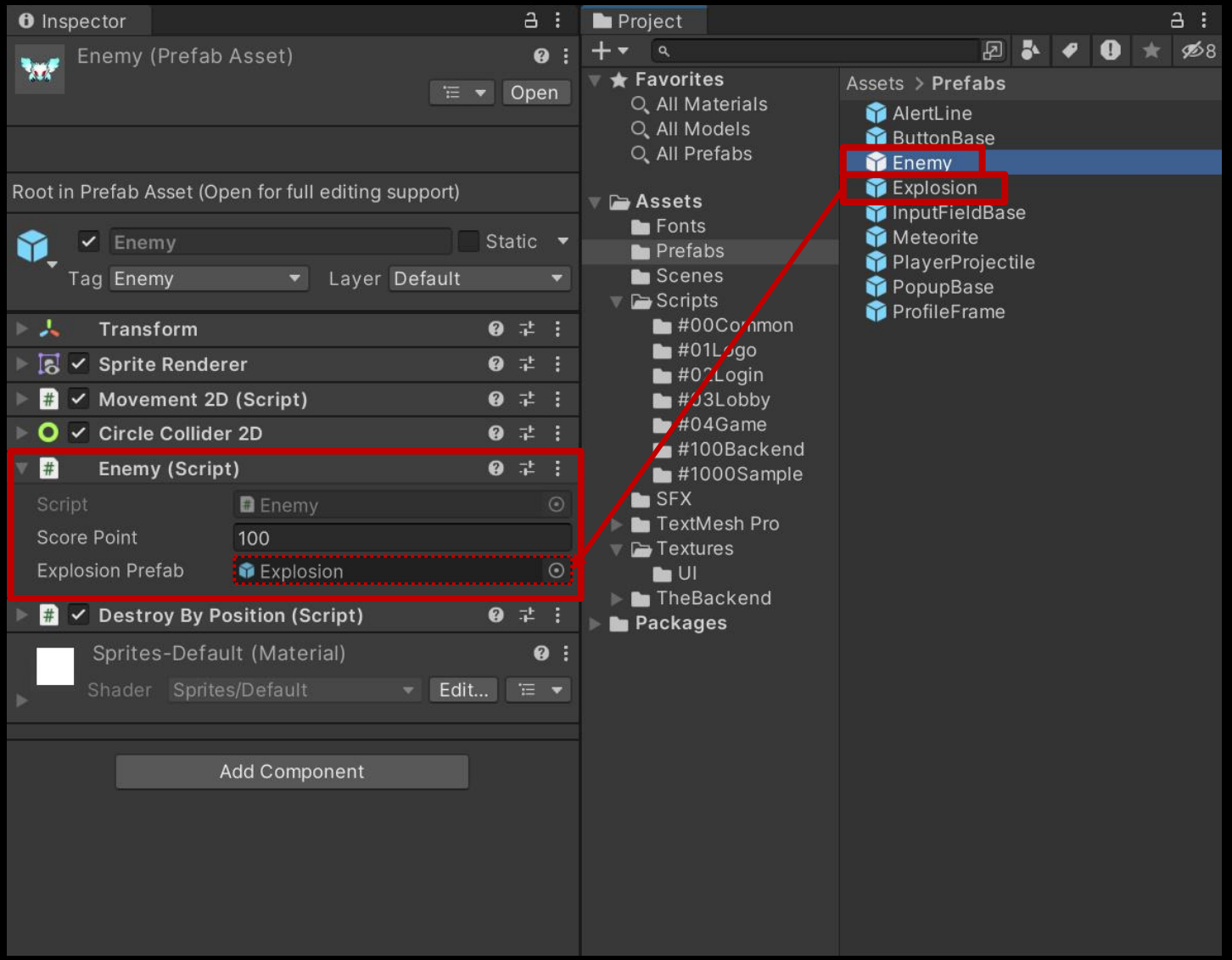
인게임 점수 구현

- 적 캐릭터가 사망할 때 폭발 이펙트 생성
  - Enemy Script 수정

```
1   using UnityEngine;
2
3   public class Enemy : MonoBehaviour
4   {
5       [SerializeField]
6       private int             scorePoint = 100;       // 적을 처치했을 때 획득하는 점수
7       [SerializeField]
8       private GameObject      explosionPrefab;        // 폭발 이펙트 프리팹
9       private GameController  gameController;
10
11      public void Setup(GameController gameController)...
15
16      public void OnDie()
17      {
18          // 폭발 이펙트 생성
19          Instantiate(explosionPrefab, transform.position, Quaternion.identity);
20          // 플레이어의 점수를 scorePoint만큼 증가
21          gameController.Score += scorePoint;
22          // 적 캐릭터 삭제
23          Destroy(gameObject);
24      }
25
26      private void OnTriggerEnter2D(Collider2D collision)
27      {
28          if ( collision.CompareTag("Player") )
29          {
30              OnDie();
31              gameController.GameOver();
32          }
33      }
34  }
```

인게임 점수 구현

- Enemy 프리팹의 "Enemy" 컴포넌트 변수 설정



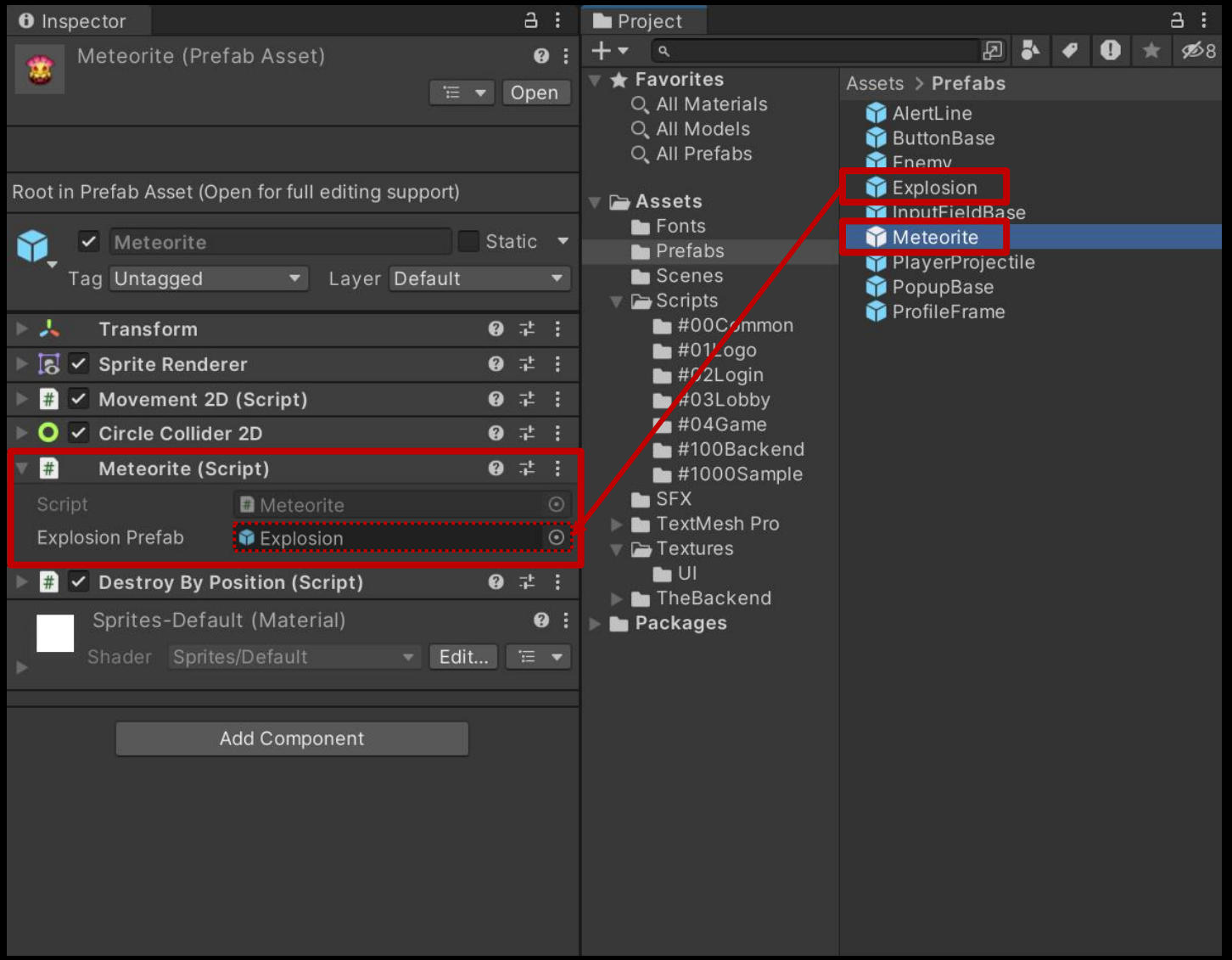
인게임 점수 구현

- 운석이 사망할 때 폭발 이펙트 생성
  - Meteorite Script 수정

```
1   using UnityEngine;
2
3   public class Meteorite : MonoBehaviour
4   {
5       [SerializeField]
6       private GameObject      explosionPrefab;        // 폭발 이펙트 프리팹
7       private GameController  gameController;
8
9       public void Setup(GameController gameController)...
13
14      private void OnTriggerEnter2D(Collider2D collision)
15      {
16          if ( collision.CompareTag("Player") )
17          {
18              // 폭발 이펙트 생성
19              Instantiate(explosionPrefab, transform.position, Quaternion.identity);
20              // 운석 삭제
21              Destroy(gameObject);
22
23              gameController.GameOver();
24          }
25      }
26  }
```

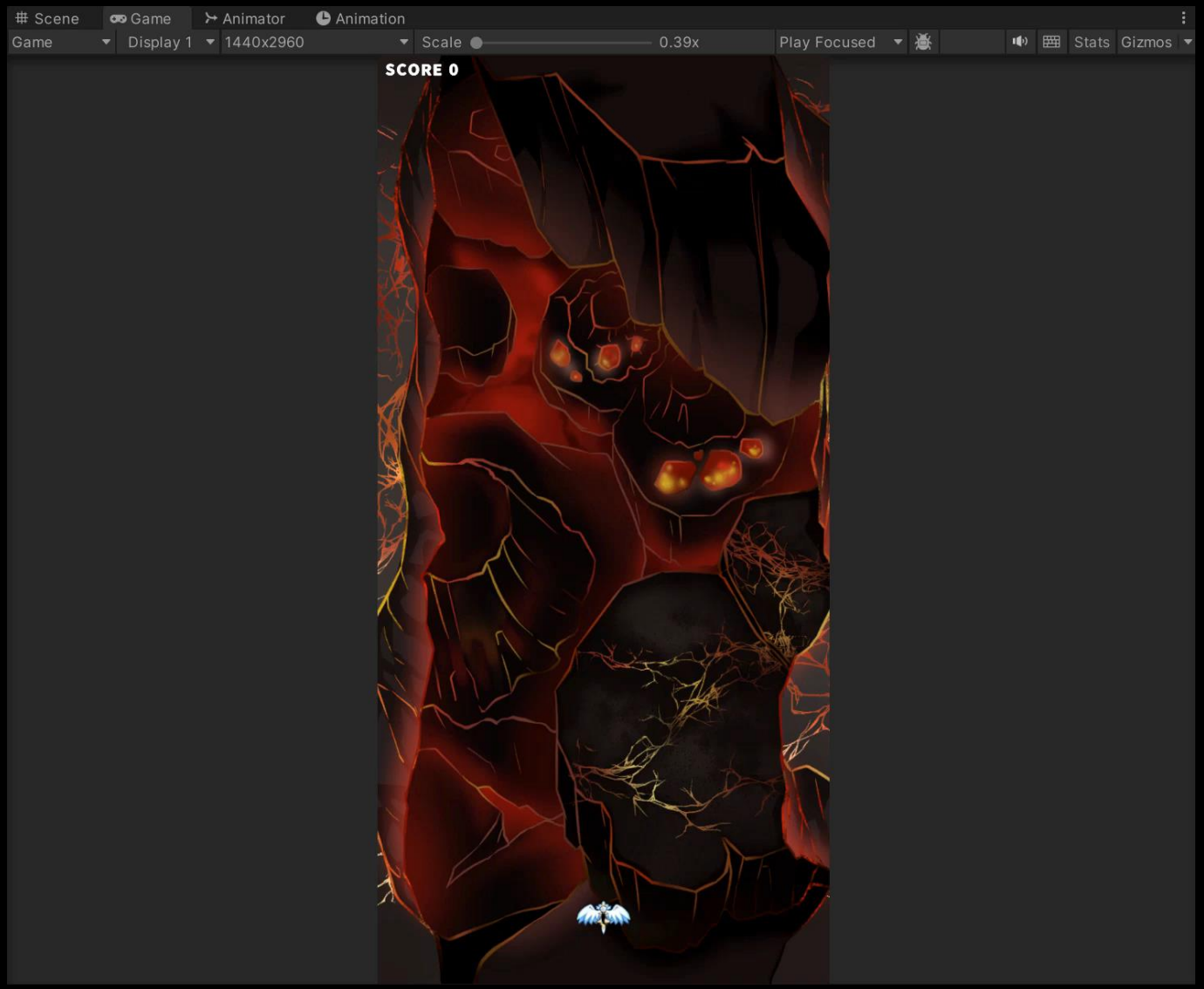
인게임 점수 구현

- Meteorite 프리팹의 "Meteorite" 컴포넌트 변수 설정



인게임 점수 구현

- 결과 화면



게임오버 UI 구현

- 게임오버 되었을 때 인게임 오브젝트 비활성화
- 게임오버 UI 출력
- Scale Effect
- Counting Effect

게임오버 UI 구현

- **게임오버 되었을 때 인게임 오브젝트 비활성화**
  - 게임오버 되었을 때 호출하고 싶은 메소드 등록 및 호출
    - GameController Script 수정

```
1   using UnityEngine;
2   using UnityEngine.Events;
3
4   public class GameController : MonoBehaviour
5   {
6       [SerializeField]
7       private UnityEvent  onGameOver;     // 게임오버 되었을 때 호출할 메소드 등록 및 실행
8
9       private int     score = 0;
10
11      public  bool    IsGameOver { set; get; } = false;
12      public  int     Score...
17
```

게임오버 UI 구현

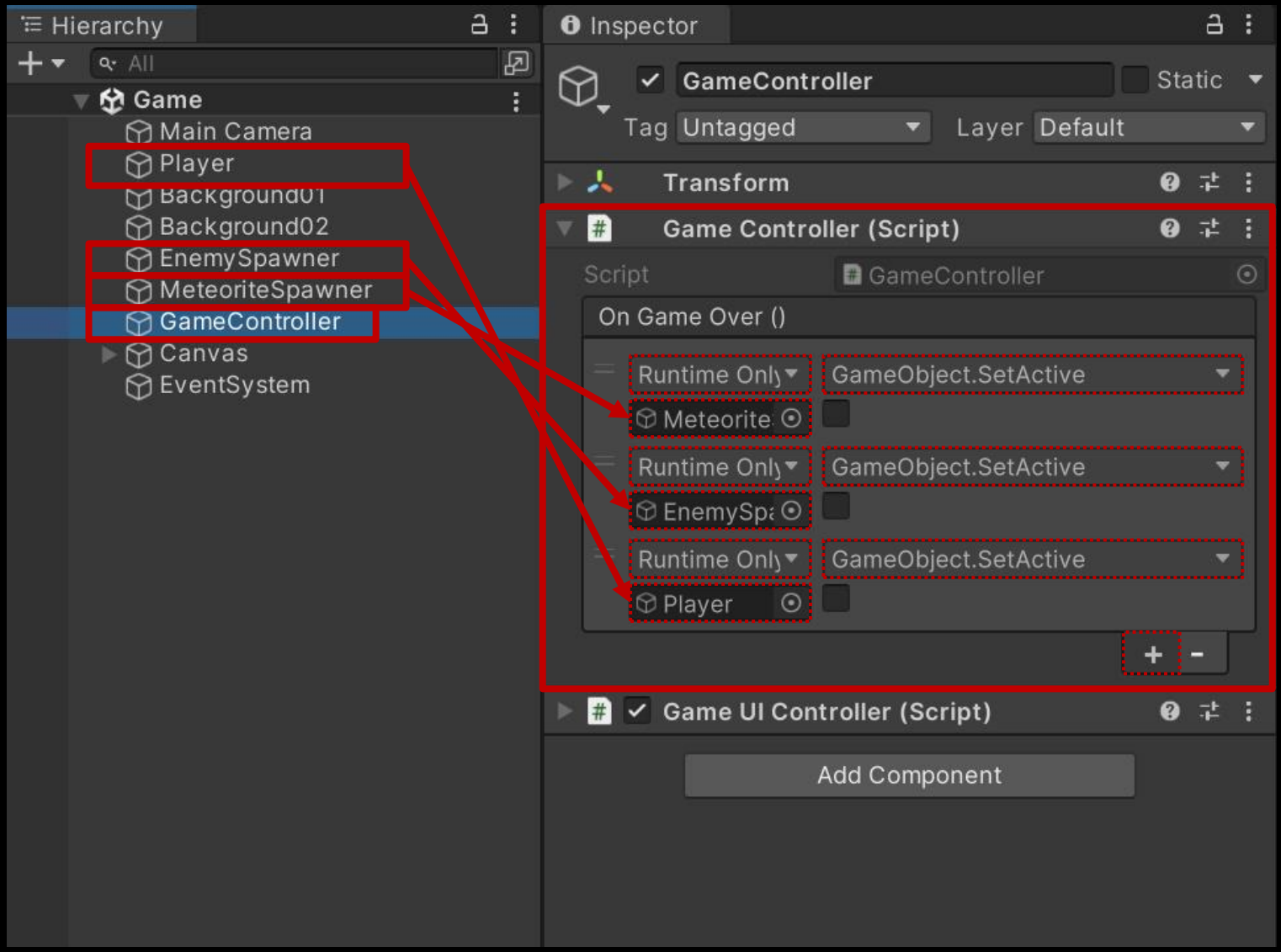
- GameController Script 수정 (계속)

```
18      public void GameOver()
19      {
20          // 중복 처리 되지 않도록 bool 변수로 제어
21          if ( IsGameOver == true ) return;
22
23          IsGameOver = true;
24
25          // 게임오버 되었을 때 호출할 메소드들을 실행
26          onGameOver.Invoke();
27
28          // 경험치 증가 및 레벨업 여부 검사
29          // (현재 레벨 시스템에 대한 설정이 없기 때문에 경험치의 최대치를 100으로 가정)
30          // (게임을 한번 플레이할 때마다 경험치는 25씩 증가)
31          BackendGameData.Instance.UserGameData.experience += 25;
32          if ( BackendGameData.Instance.UserGameData.experience >= 100 )...
37
38          // 게임 정보 업데이트
39          BackendGameData.Instance.GameDataUpdate(/*AfterGameOver*/);
40      }
41
42      /*public void AfterGameOver()
43      {
44          // 로비 씬으로 이동
45          Utils.LoadScene(SceneNames.Lobby);
46      }*/
47  }
```

게임오버 UI 출력 이후에 씬을 변경하기 때문에 AfterGameOver() 메소드 삭제

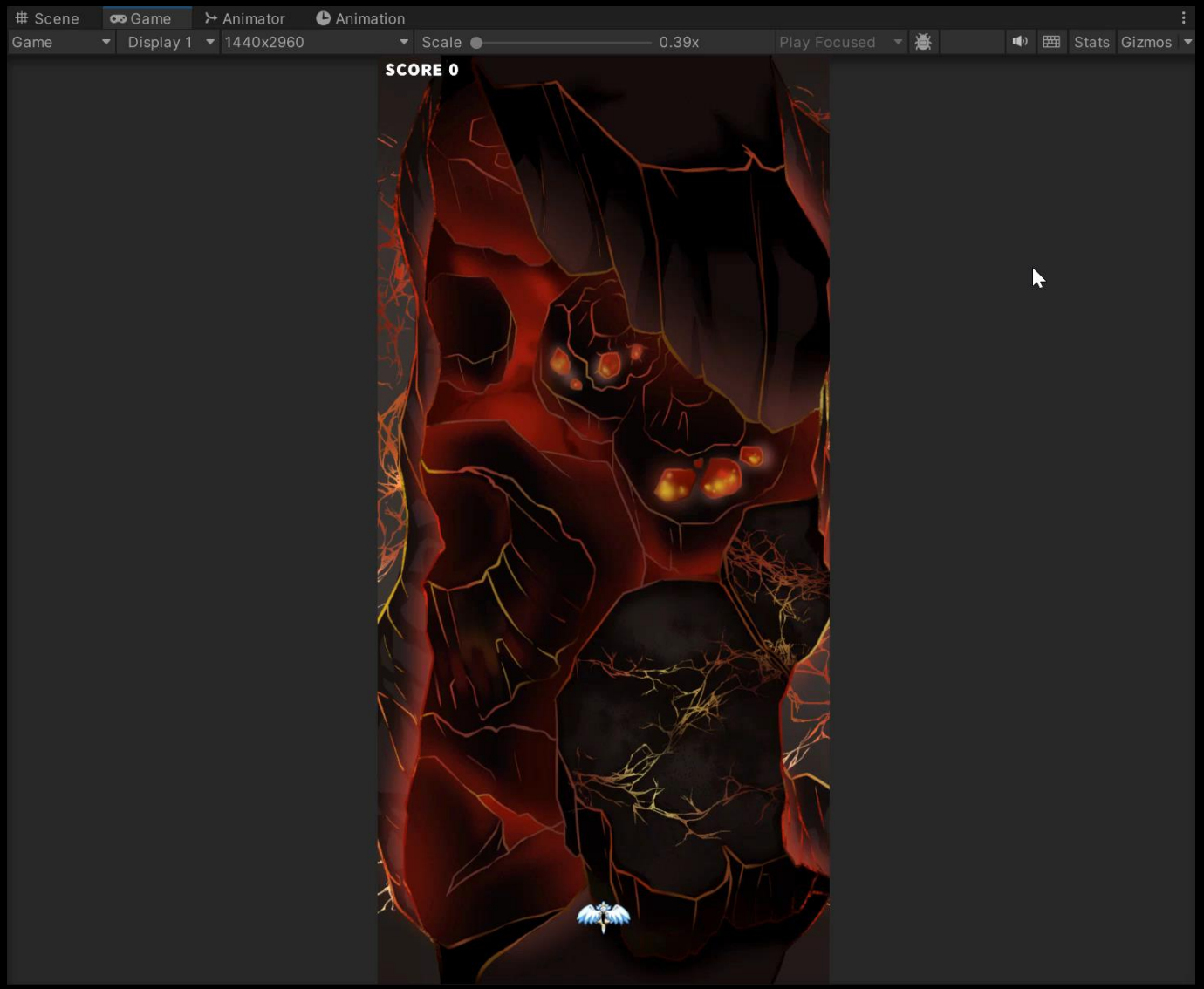
게임오버 UI 구현

- GameController 오브젝트의 "GameController" 컴포넌트 변수 설정



게임오버 UI 구현

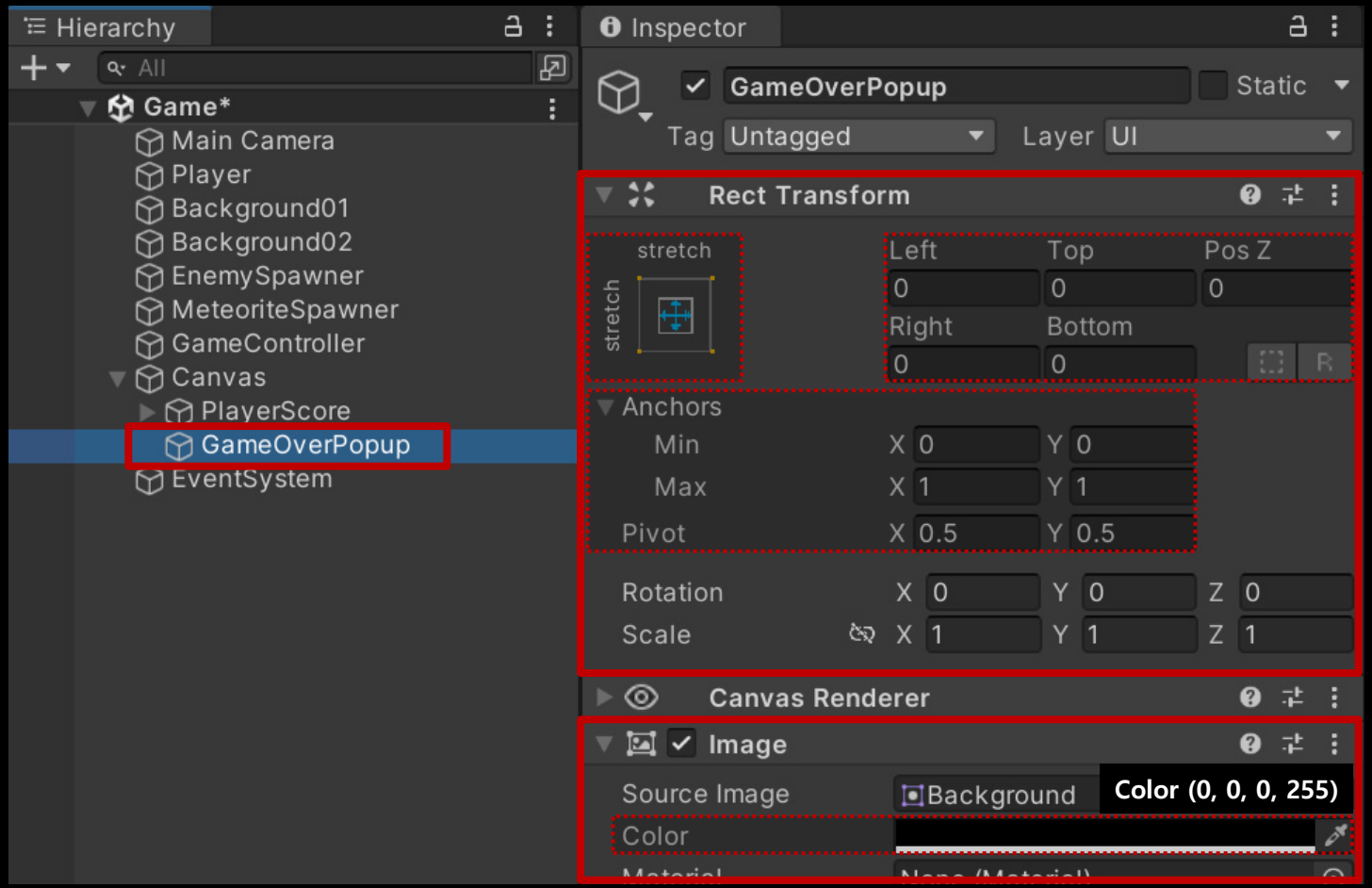
- 결과 화면



게임오버 UI 구현

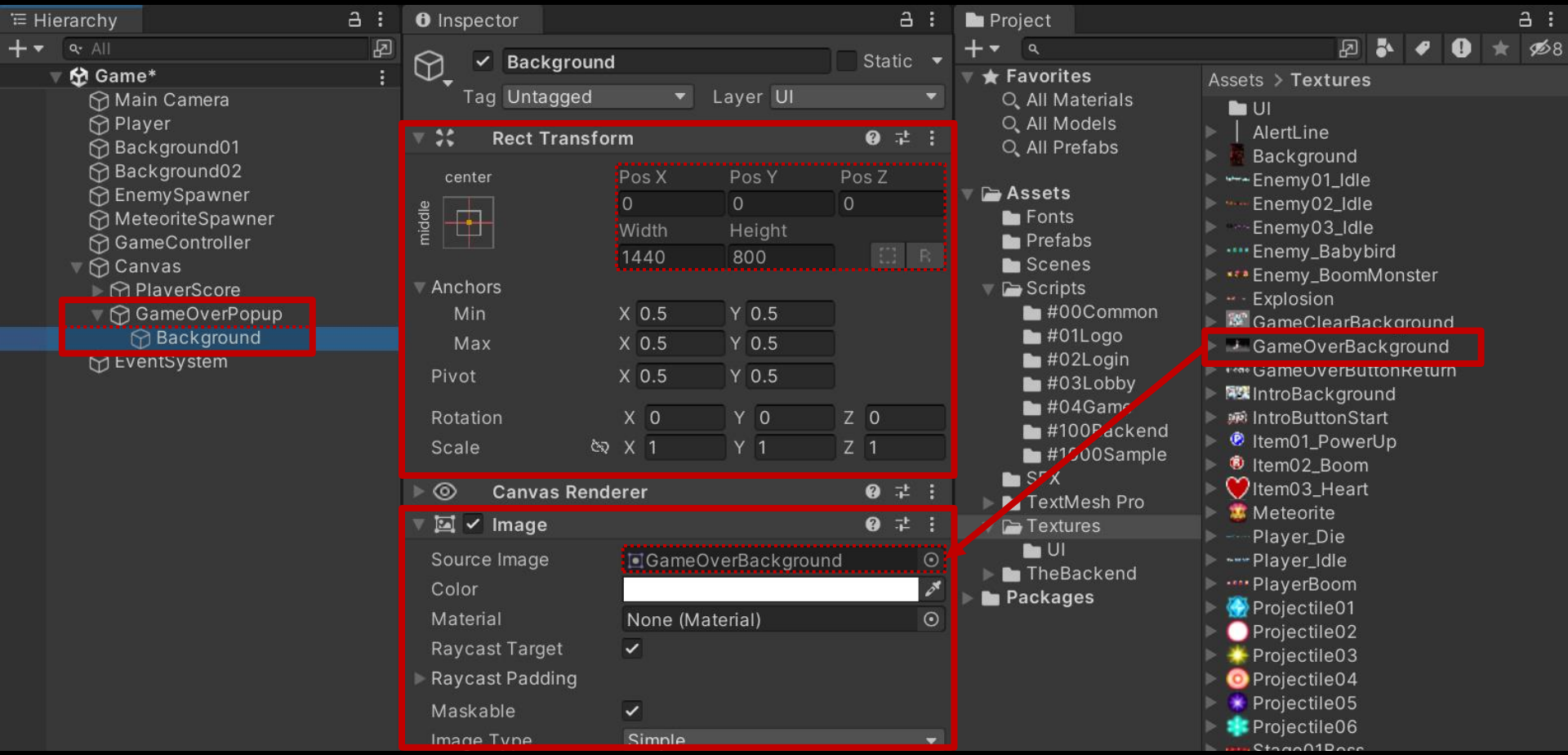
게임오버 UI 출력

- 게임오버 UI들을 관리하는 Panel UI 생성 및 설정
  - GameObject - UI - Panel



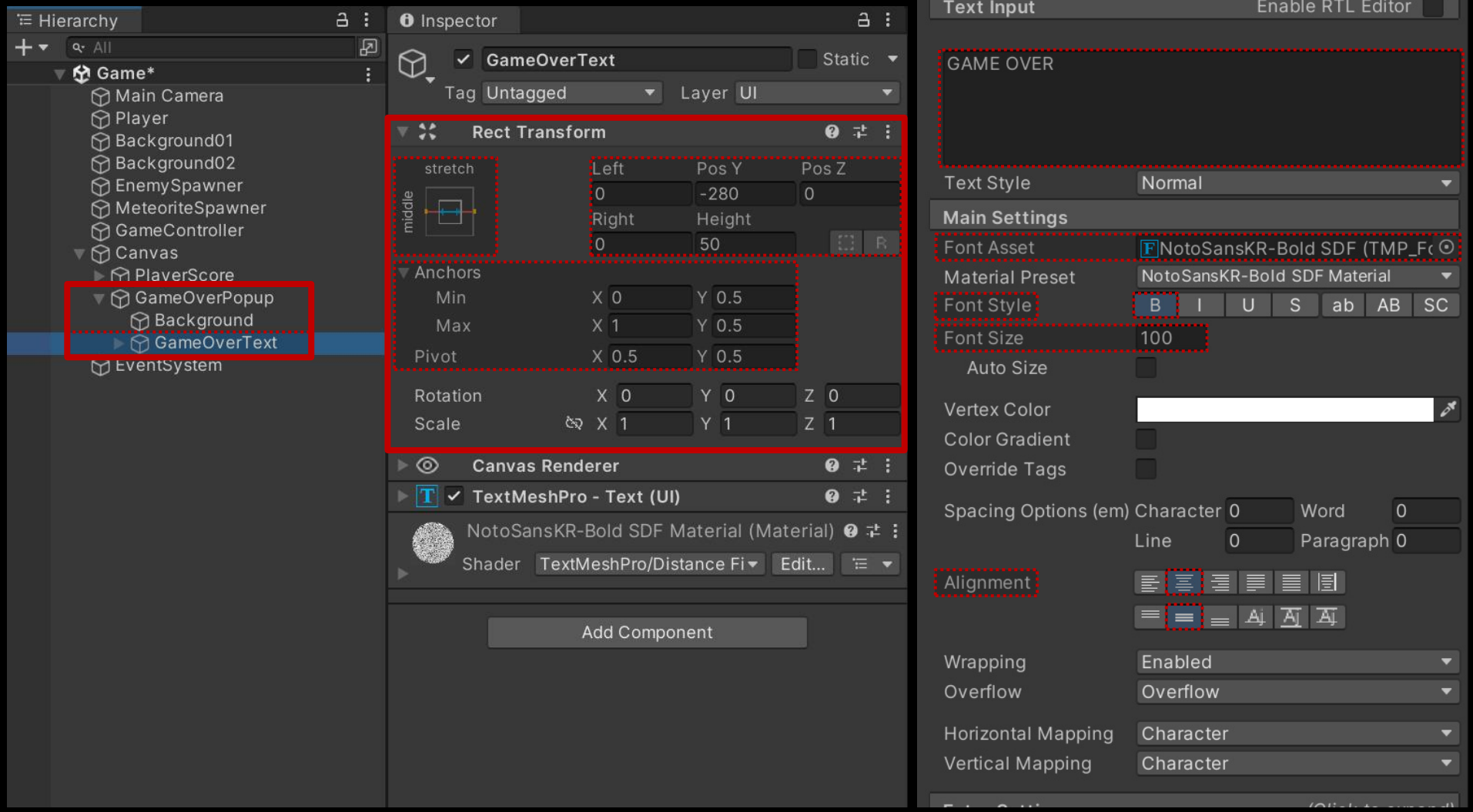
게임오버 UI 구현

- 게임오버 배경을 출력하는 Image UI 생성 및 설정
  - GameObject - UI - Image



게임오버 UI 구현

- "GAME OVER" 텍스트를 출력하는 "Text - TextMeshPro" UI 생성 및 설정
  - GameObject - UI - "Text - TextMeshPro"



게임오버 UI 구현

- "SCORE" 텍스트를 출력하는 "Text - TextMeshPro" UI 생성 및 설정
  - GameObject - UI - "Text - TextMeshPro"

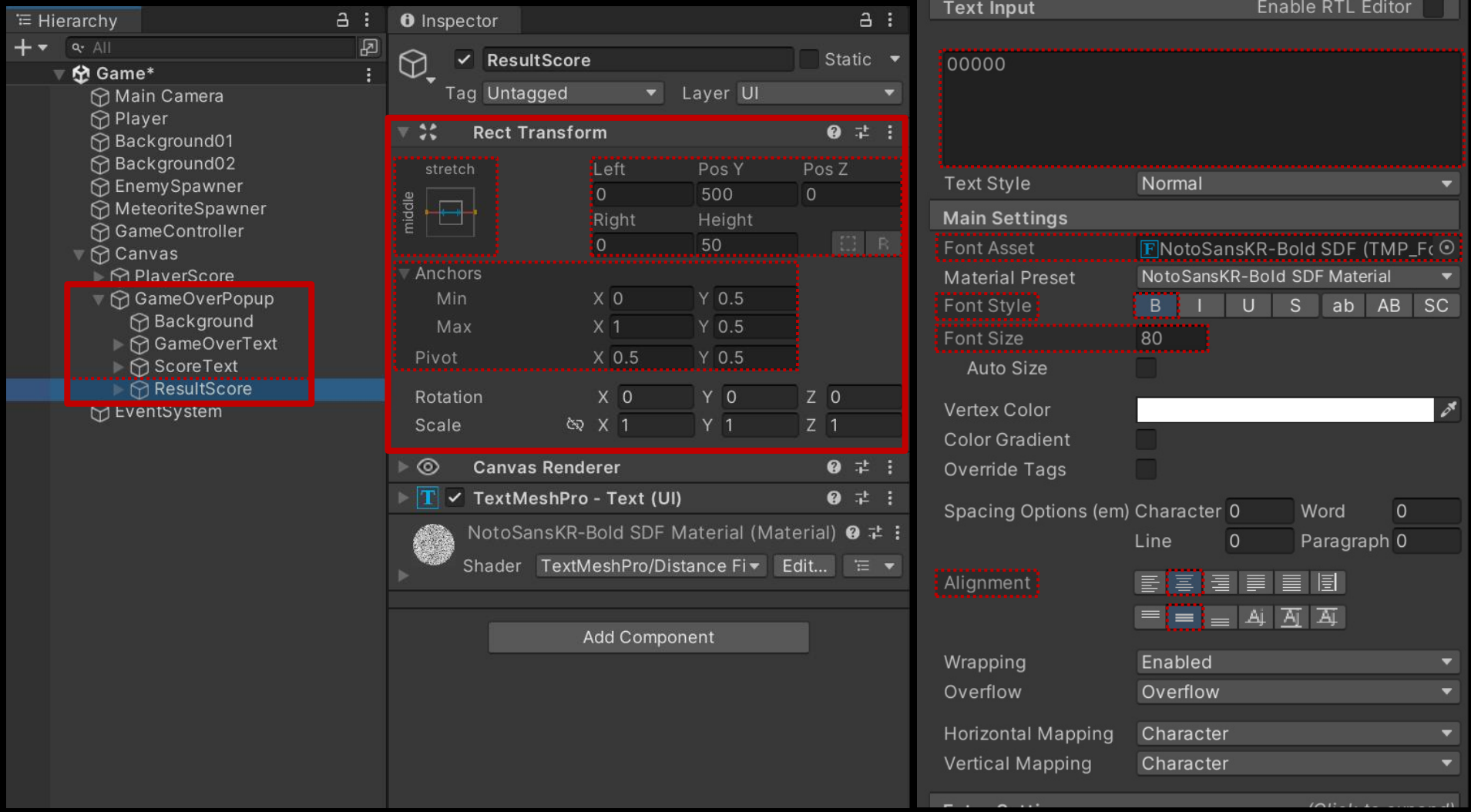
Hierarchy
All
Game*
Main Camera
Player
Background01
Background02
EnemySpawner
MeteoriteSpawner
GameController
Canvas
PlayerScore
GameOverPopup
Background
GameOverText
ScoreText
EventSystem

Inspector
ScoreText
Static
Tag Untagged
Layer UI
Rect Transform
stretch
middle
Left 0
Pos Y 600
Pos Z 0
Right 0
Height 50
Anchors
Min X 0 Y 0.5
Max X 1 Y 0.5
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Canvas Renderer
TextMeshPro - Text (UI)
NotoSansKR-Bold SDF Material (Material)
Shader TextMeshPro/Distance Fi
Edit...
Add Component

TextMeshPro - Text (UI)
Text Input
Enable RTL Editor
SCORE
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 80
Auto Size
Vertex Color
Color Gradient
Override Tags
Spacing Options (em) Character 0 Word 0
Line 0 Paragraph 0
Alignment
Wrapping Enabled
Overflow Overflow
Horizontal Mapping Character
Vertical Mapping Character

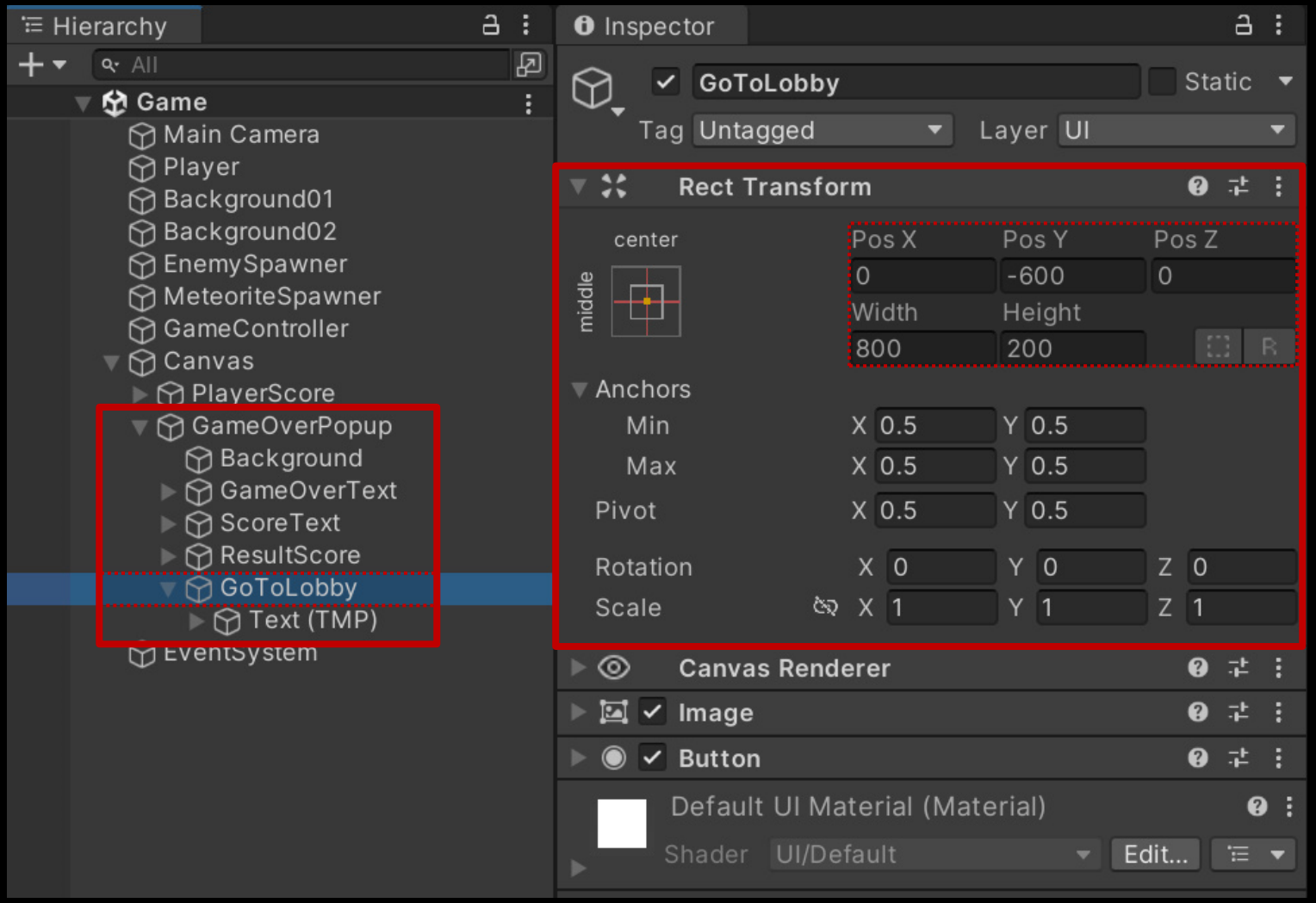
게임오버 UI 구현

- 획득한 점수를 출력하는 "Text - TextMeshPro" UI 생성 및 설정
  - GameObject - UI - "Text - TextMeshPro"



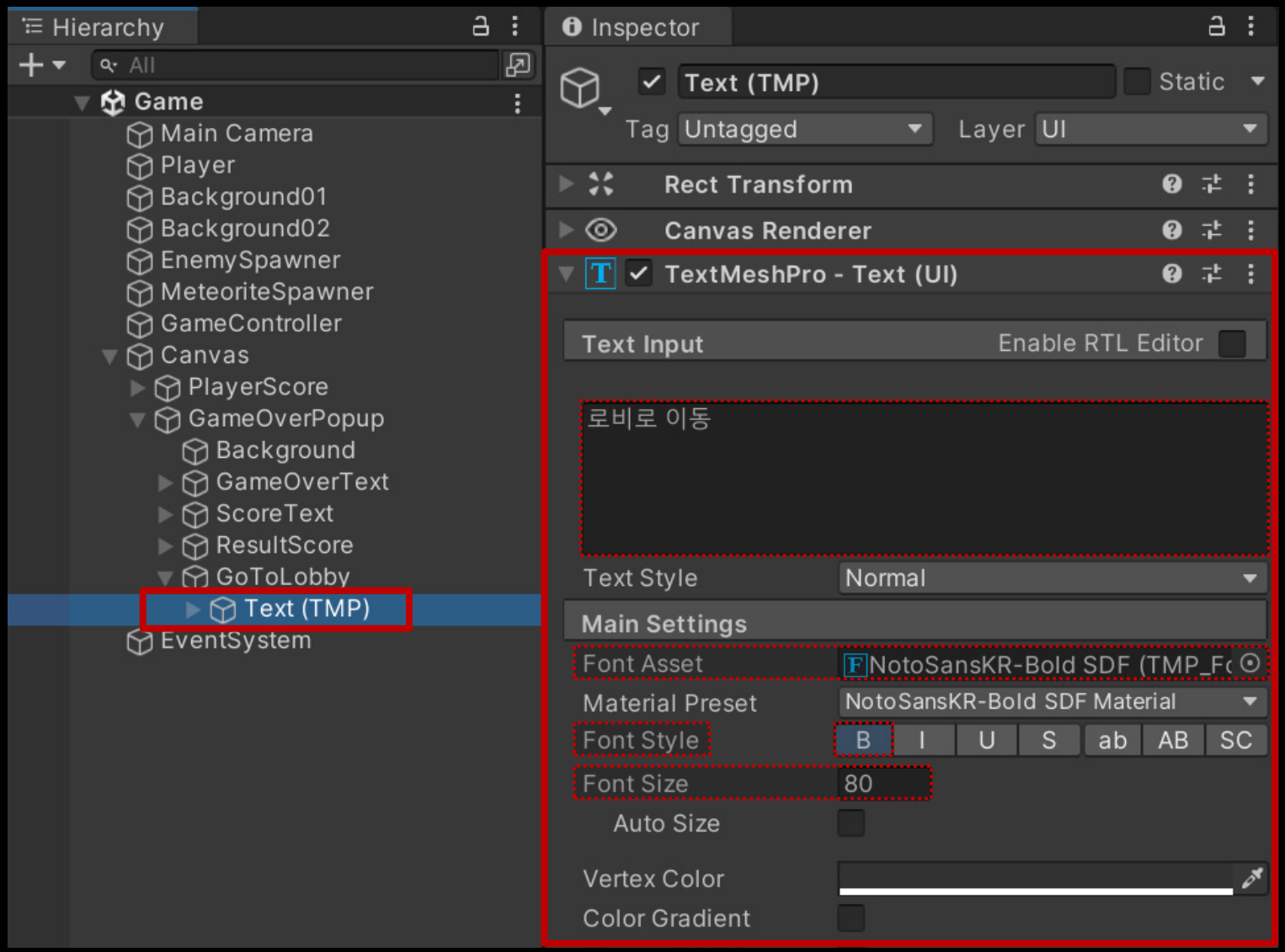
게임오버 UI 구현

- 로비로 이동하는 "Button - TextMeshPro" UI 생성 및 설정
  - GameObject - UI - "Button - TextMeshPro"



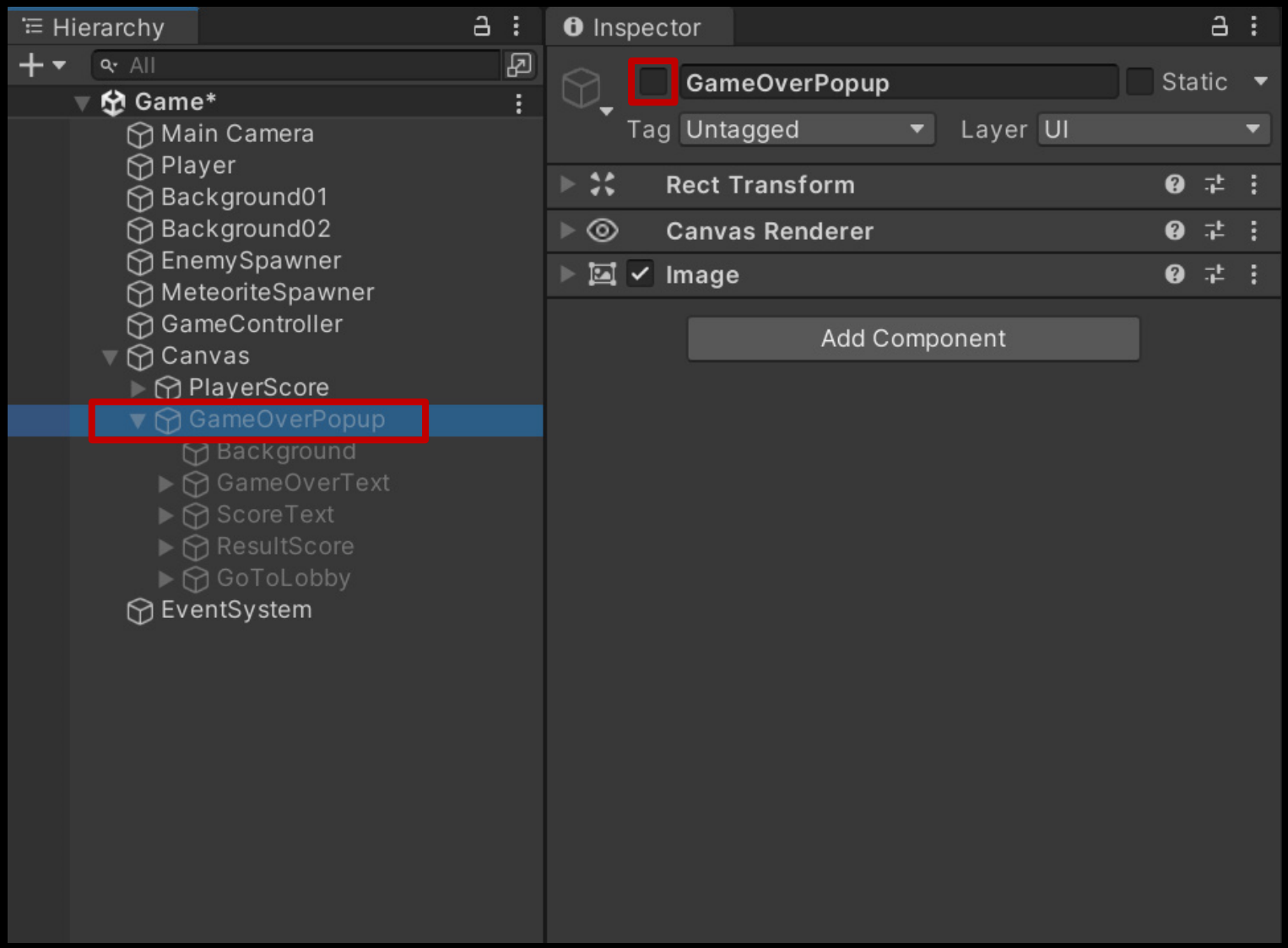
게임오버 UI 구현

- 로비로 이동하는 "Button - TextMeshPro" UI 생성 및 설정 (계속)



게임오버 UI 구현

- GameOverPopup 오브젝트 비활성화



게임오버 UI 구현

- 게임오버 되었을 때 호출하는 OnGameOver() 메소드 정의
  - GameUIController Script 수정

```
1   using UnityEngine;
2   using TMPro;
3
4   public class GameUIController : MonoBehaviour
5   {
6       [SerializeField]
7       private GameController  gameController;
8
9       [Header("InGame")]
10      [SerializeField]
11      private TextMeshProUGUI textScore;
12
13      [Header("Game Over")]
14      [SerializeField]
15      private GameObject      panelGameOver;
16      [SerializeField]
17      private TextMeshProUGUI textResultScore;
18
```

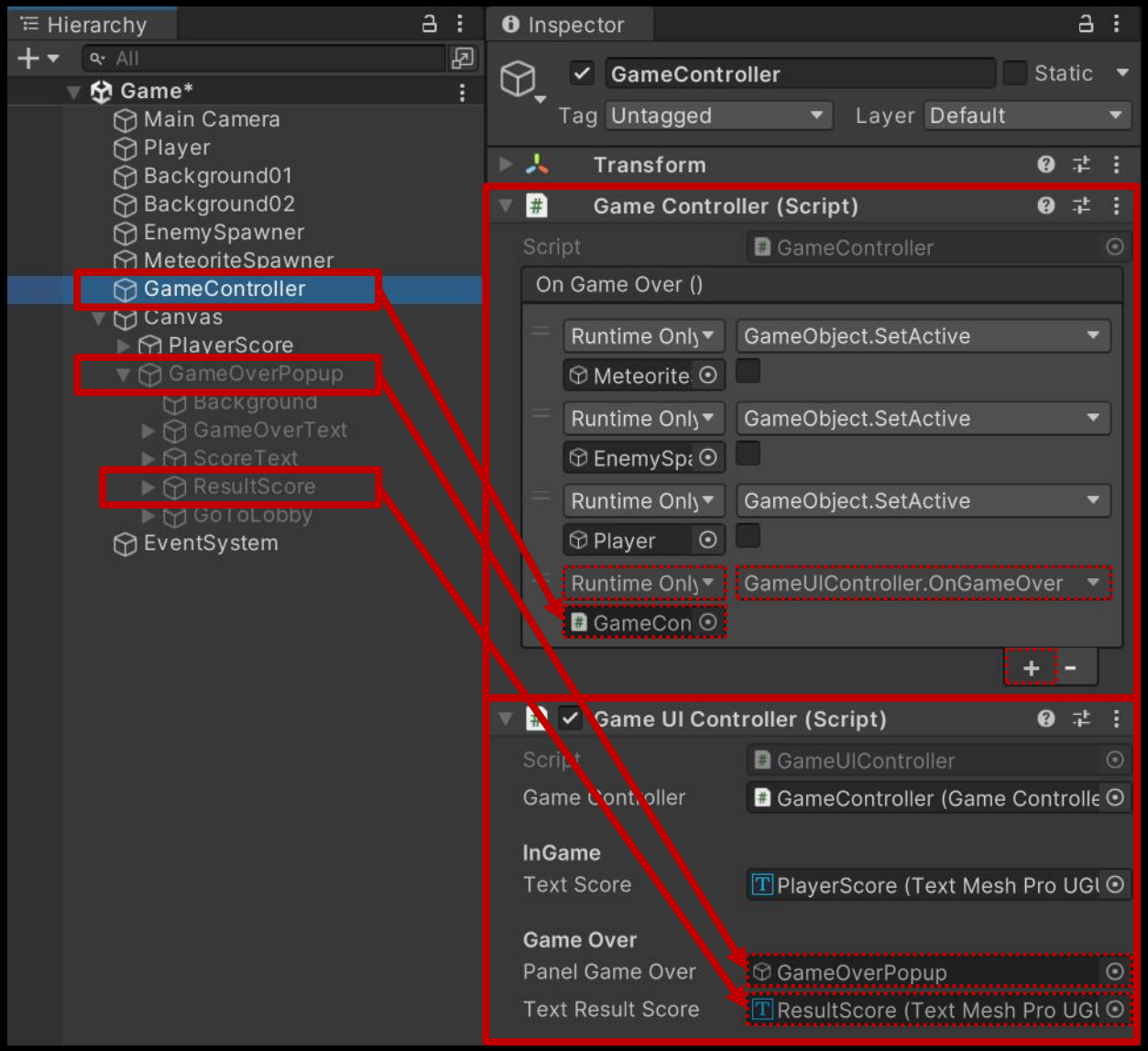
게임오버 UI 구현

- GameUIController Script 수정

```
19      private void Update()...
23
24      public void OnGameOver()
25      {
26          // GameOver Panel UI 활성화
27          panelGameOver.SetActive(true);
28          // 획득 점수 출력
29          textResultScore.text = gameController.Score.ToString();
30      }
31
32      public void BtnClickGoToLobby()
33      {
34          Utils.LoadScene(SceneNames.Lobby);
35      }
36  }
```

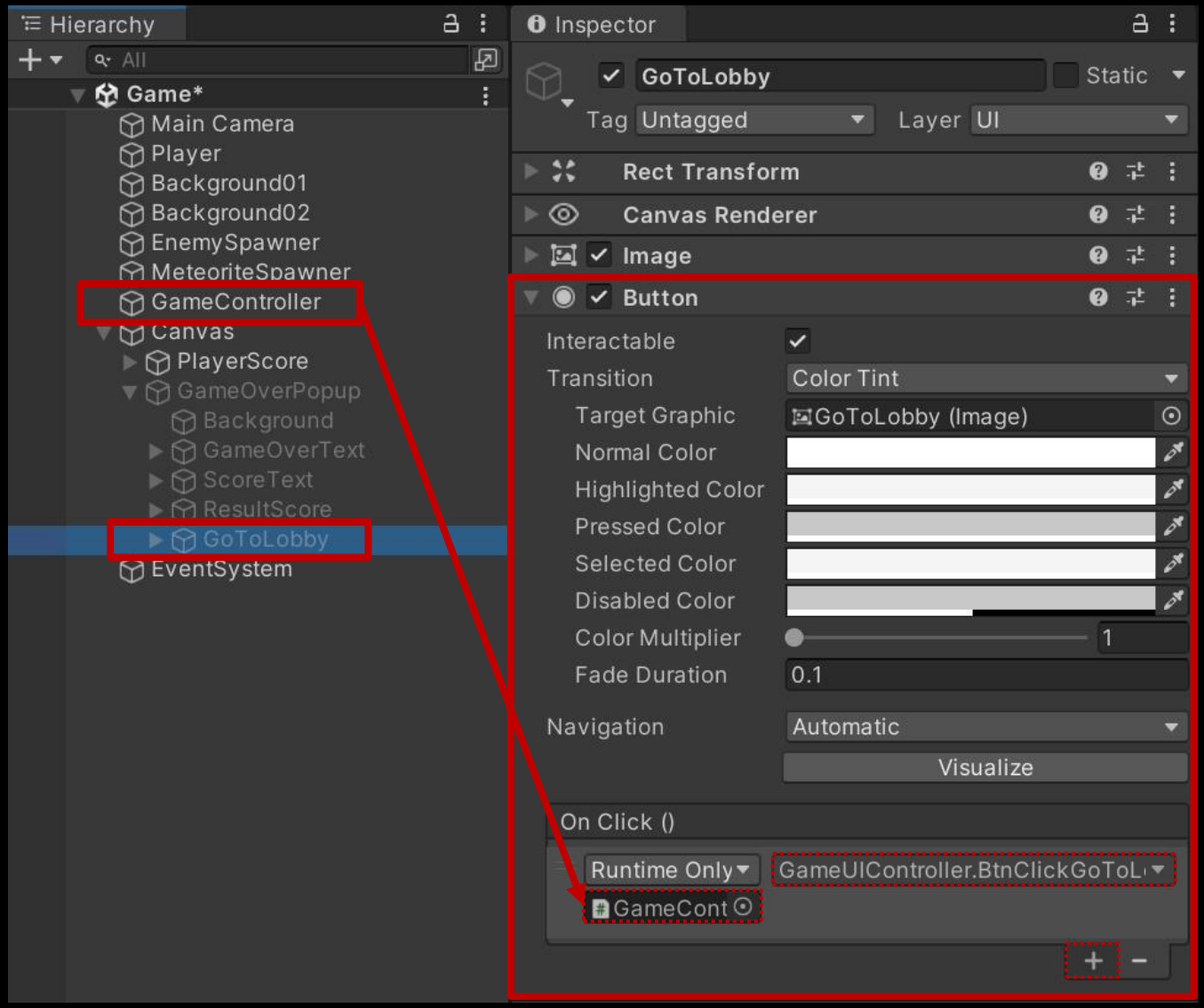
게임오버 UI 구현

- GameController 오브젝트의 컴포넌트 변수 설정



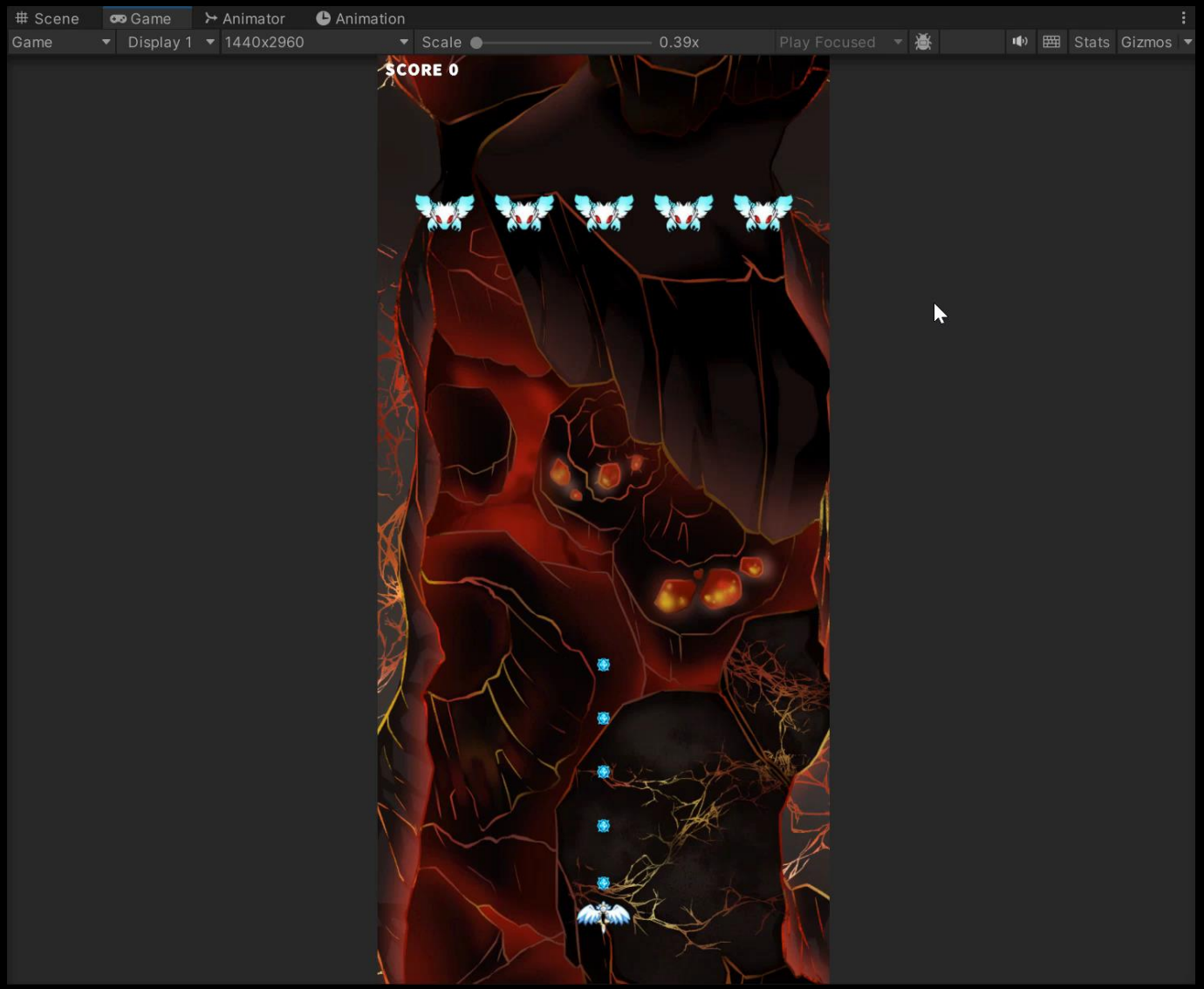
게임오버 UI 구현

- GotoLobby 오브젝트의 "Button" 컴포넌트 OnClick 이벤트 등록



게임오버 UI 구현

- 결과 화면



게임오버 UI 구현

- **Scale Effect**
  - Text UI의 크기 애니메이션을 제어하는 스크립트 생성 및 작성
    - C# Script 생성 후 스크립트의 이름을 "ScaleEffect"로 변경

```
1  using System.Collections;
2  using UnityEngine;
3  using TMPro;
4
5  public class ScaleEffect : MonoBehaviour
6  {
7      [SerializeField]
8      [Range(0.01f, 10f)]
9      private float           effectTime;     // 크기 확대/축소 되는 시간
10     private TextMeshProUGUI effectText;     // 크기 확대/축소 효과에 사용되는 텍스트
11
12     private void Awake()
13     {
14         effectText = GetComponent<TextMeshProUGUI>();
15     }
16
```

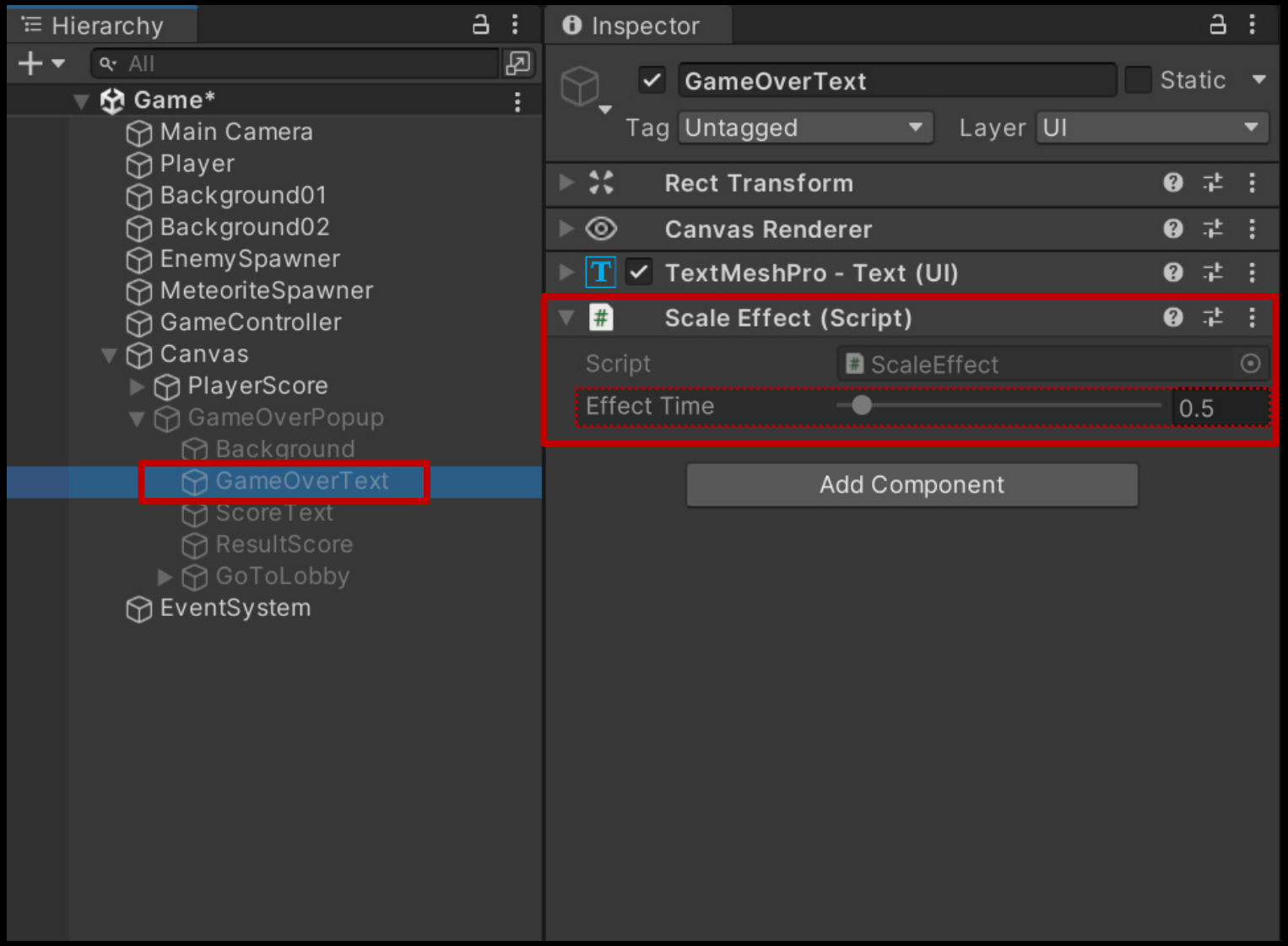
게임오버 UI 구현

- Text UI의 크기 애니메이션을 제어하는 스크립트 생성 및 작성 (계속)

```
17      public void Play(float start, float end)
18      {
19          StartCoroutine(Process(start, end));
20      }
21
22      private IEnumerator Process(float start, float end)
23      {
24          float current = 0;
25          float percent = 0;
26
27          while ( percent < 1 )
28          {
29              current += Time.deltaTime;
30              percent = current / effectTime;
31
32              effectText.fontSize = Mathf.Lerp(start, end, percent);
33
34              yield return null;
35          }
36      }
37  }
```

게임오버 UI 구현

- GameOverText 오브젝트에 "ScaleEffect" 컴포넌트 추가 및 설정



게임오버 UI 구현

- "GAME OVER" 텍스트 크기 축소 애니메이션 재생
  - GameUIController Script 수정

```
1   using UnityEngine;
2   using TMPro;
3
4   public class GameUIController : MonoBehaviour
5   {
6       [SerializeField]
7       private GameController  gameController;
8
9       [Header("InGame")]
10      [SerializeField]
11      private TextMeshProUGUI textScore;
12
13      [Header("Game Over")]
14      [SerializeField]
15      private GameObject      panelGameOver;
16      [SerializeField]
17      private TextMeshProUGUI textResultScore;
18
19      [Header("Game Over UI Animation")]
20      [SerializeField]
21      private ScaleEffect     effectGameOver;
22
```

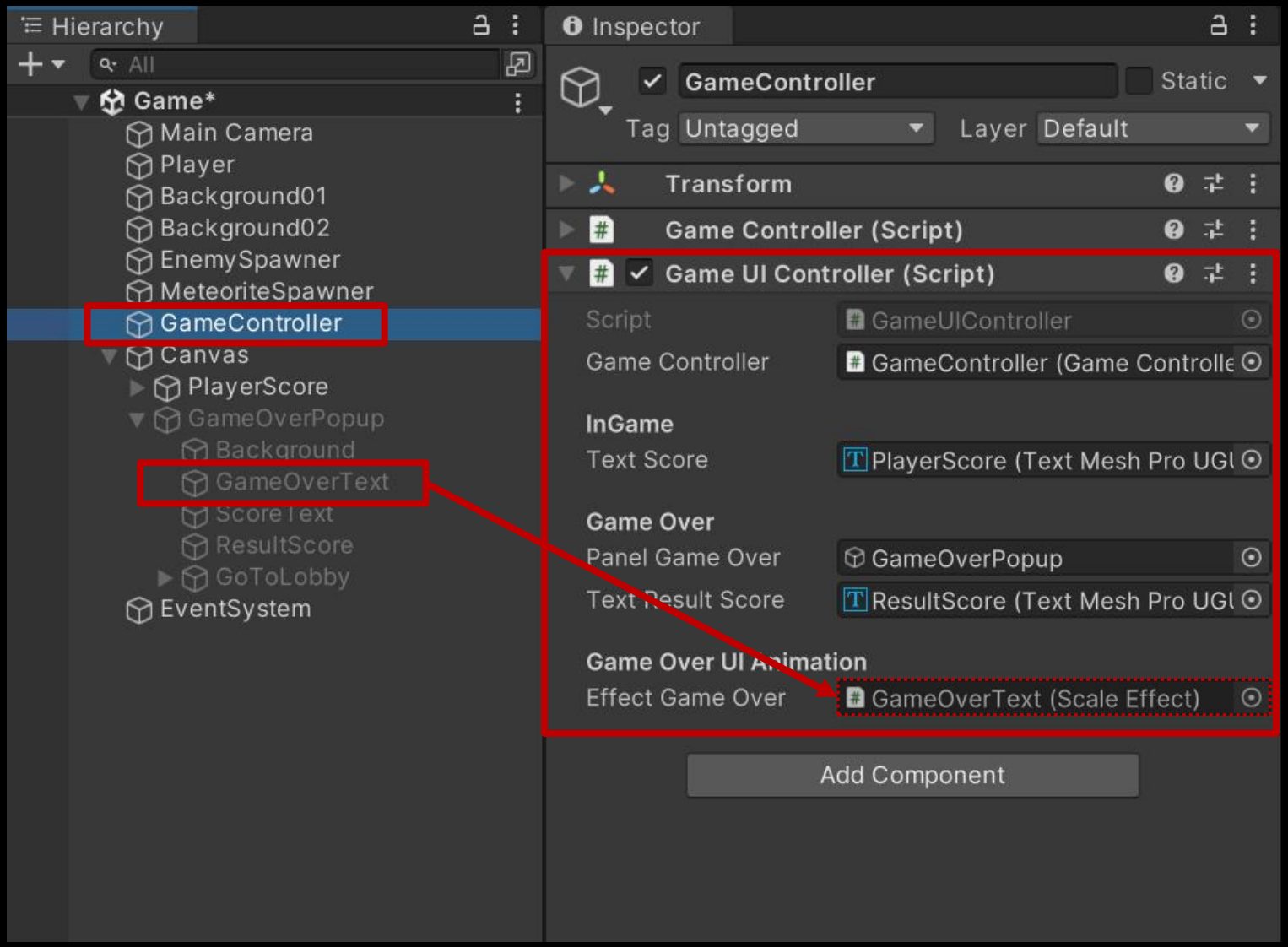
게임오버 UI 구현

- GameUIController Script 수정 (계속)

```
23      private void Update()...
27
28      public void OnGameOver()
29      {
30          // GameOver Panel UI 활성화
31          panelGameOver.SetActive(true);
32          // 획득 점수 출력
33          textResultScore.text = gameController.Score.ToString();
34          // "GAME OVER" 텍스트 크기 축소 애니메이션
35          effectGameOver.Play(200, 100);
36      }
37
38      public void BtnClickGoToLobby()...
42  }
```

게임오버 UI 구현

- GameController 오브젝트의 "GameUIController" 컴포넌트 변수 설정



게임오버 UI 구현

- **Counting Effect**
  - 점수 카운팅 애니메이션을 제어하는 스크립트 생성 및 작성
    - C# Script 생성 후 스크립트의 이름을 "CountingEffect"로 변경

```
1  using System.Collections;
2  using UnityEngine;
3  using TMPro;
4
5  public class CountingEffect : MonoBehaviour
6  {
7      [SerializeField]
8      [Range(0.01f, 10f)]
9      private float           effectTime;     // 카운팅 되는 시간
10     private TextMeshProUGUI effectText;     // 카운팅 효과에 사용되는 텍스트
11
12     private void Awake()
13     {
14         effectText = GetComponent<TextMeshProUGUI>();
15     }
16
```

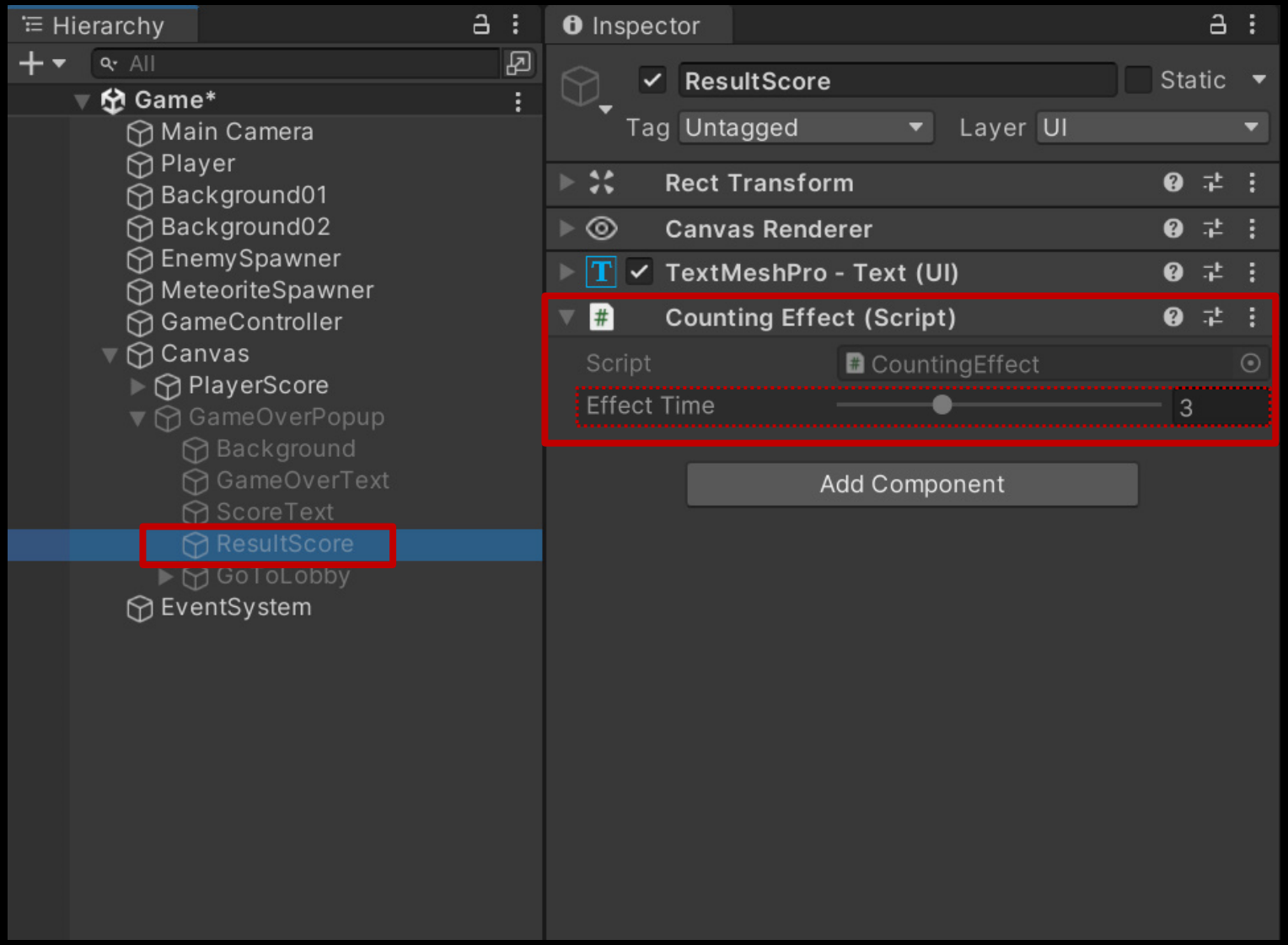
게임오버 UI 구현

- 점수 카운팅 애니메이션을 제어하는 스크립트 생성 및 작성 (계속)

```
17      public void Play(int start, int end)
18      {
19          StartCoroutine(Process(start, end));
20      }
21
22      private IEnumerator Process(int start, int end)
23      {
24          float current = 0;
25          float percent = 0;
26
27          while ( percent < 1 )
28          {
29              current += Time.deltaTime;
30              percent = current / effectTime;
31
32              effectText.text = Mathf.Lerp(start, end, percent).ToString("F0");
33
34              yield return null;
35          }
36      }
37  }
```

게임오버 UI 구현

- ResultScore 오브젝트에 "CountingEffect" 컴포넌트 추가 및 설정



게임오버 UI 구현

- 획득 점수 카운팅 애니메이션 재생
  - GameUIController Script 수정

```
1   using ...
3
4   public class GameUIController : MonoBehaviour
5   {
6       [SerializeField]
7       private GameController  gameController;
8
9       [Header("InGame")]
10      [SerializeField]
11      private TextMeshProUGUI textScore;
12
13      [Header("Game Over")]
14      [SerializeField]
15      private GameObject      panelGameOver;
16      [SerializeField]
17      private TextMeshProUGUI textResultScore;
18
19      [Header("Game Over UI Animation")]
20      [SerializeField]
21      private ScaleEffect     effectGameOver;
22      [SerializeField]
23      private CountingEffect  effectResultScore;
24
```

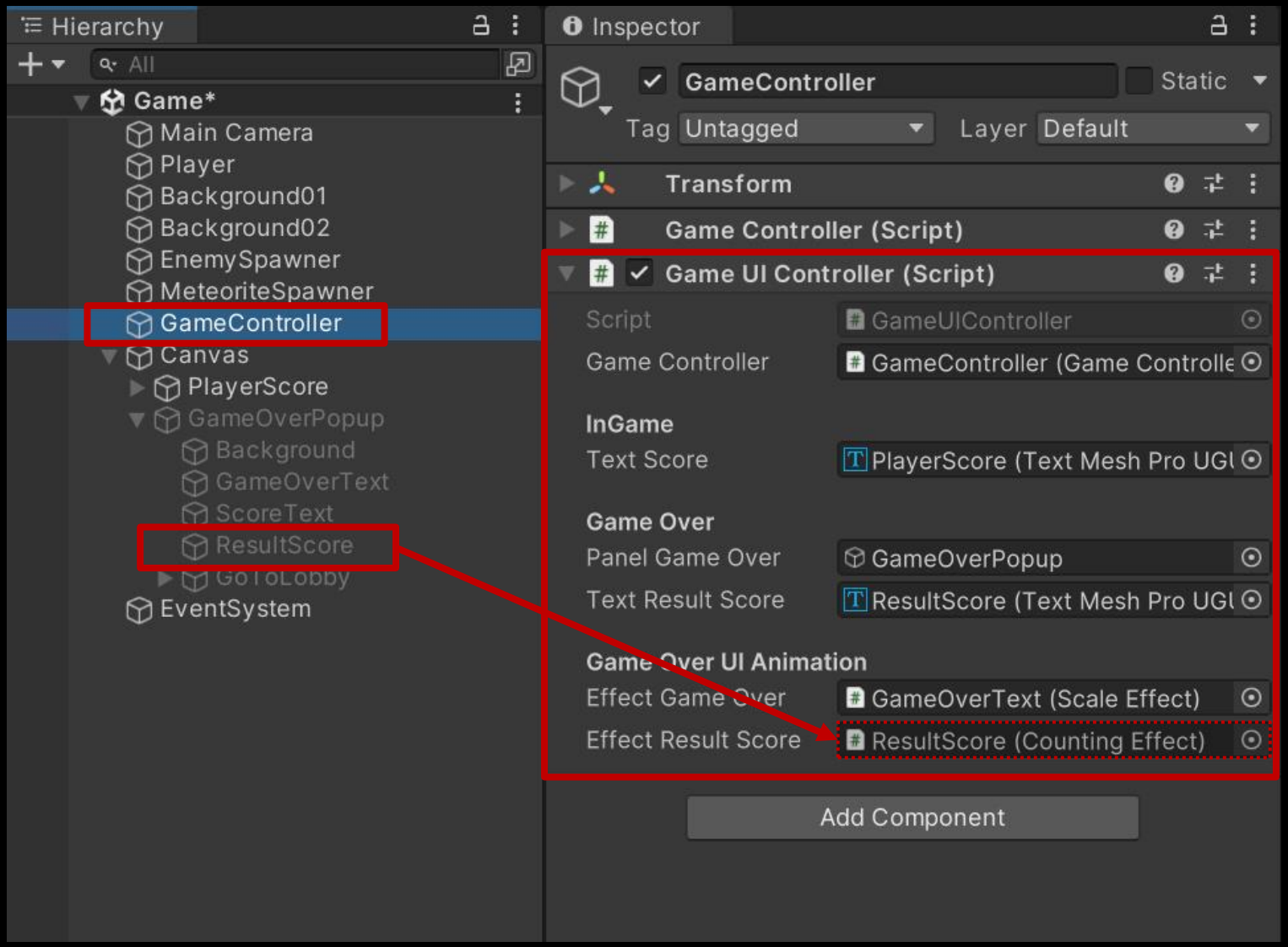
게임오버 UI 구현

- GameUIController Script 수정

```
25      private void Update()...
29
30      public void OnGameOver()
31      {
32          // GameOver Panel UI 활성화
33          panelGameOver.SetActive(true);
34          // 획득 점수 출력
35          textResultScore.text = gameController.Score.ToString();
36          // "GAME OVER" 텍스트 크기 축소 애니메이션
37          effectGameOver.Play(200, 100);
38          // 0 -> gameController.Score까지 점수를 카운팅하는 애니메이션
39          effectResultScore.Play(0, gameController.Score);
40      }
41
42      public void BtnClickGoToLobby()...
46  }
```

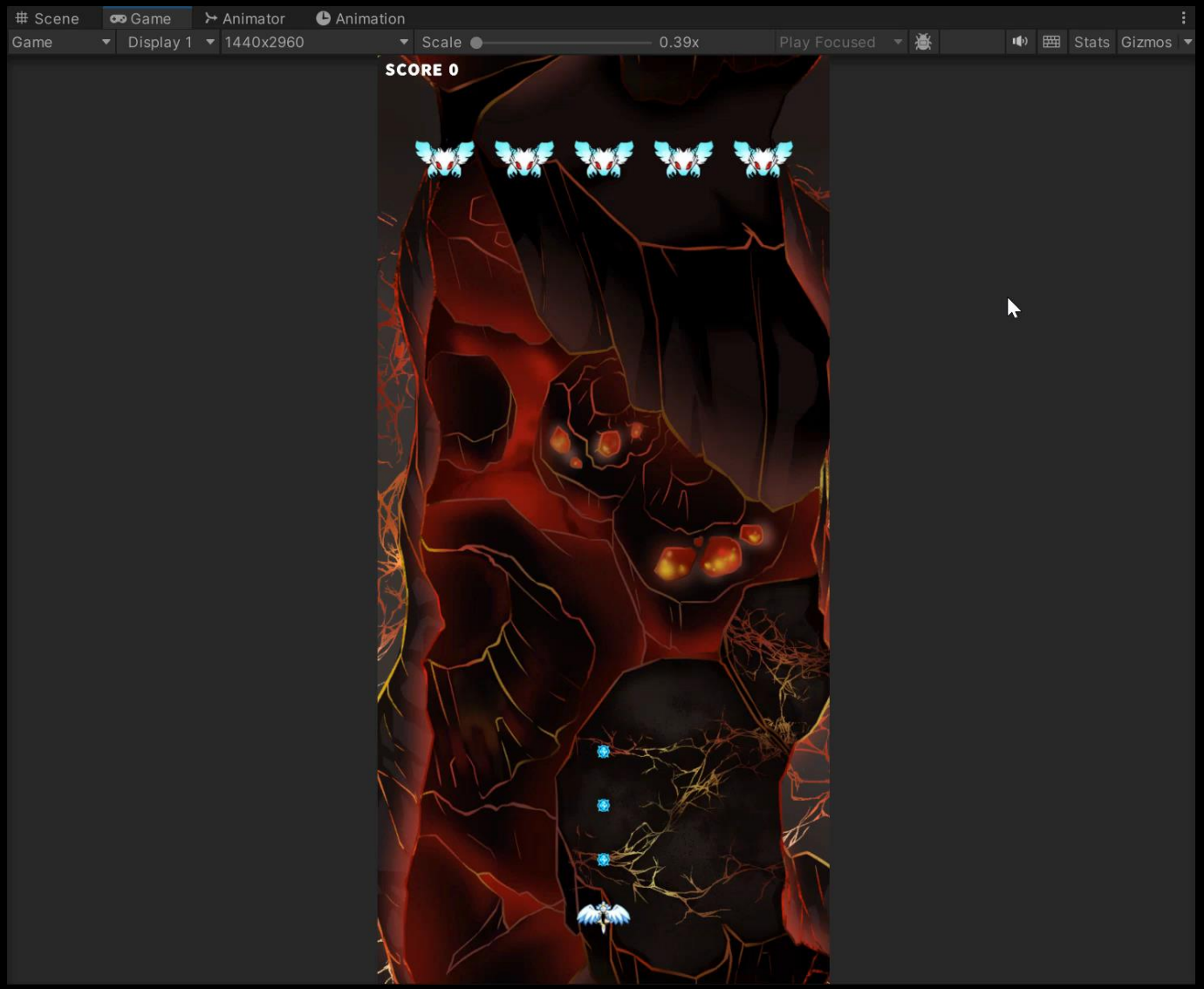
게임오버 UI 구현

- GameController 오브젝트의 "GameUIController" 컴포넌트 변수 설정



게임오버 UI 구현

- 결과 화면



랭킹으로 사용할 유저 정보 데이터 추가

- 랭킹 등록을 위해 필요한 기능들
- 일일 최고 점수 데이터 추가
- 랭킹 생성

랭킹으로 사용할 유저 정보 데이터 추가

- 랭킹 등록을 위해 필요한 기능들
  - 계정 생성 / 로그인 시스템을 이용해 등록한 유저

Backnd Console
ProjectA
관리자
ProjectA 개발 모드
즐겨찾기
대시보드
서버 설정
프로젝트
인증 정보
푸시
스토어 정보
소셜
예산 알림
요금제
뒤끝베이스
유저 관리
유저 접근 관리
게임 정보 관리
랭킹 관리
우편 관리
푸시 관리
쿠폰 관리
차트 관리
확률 관리
로그 관리
영수증 검증
유저 관리
게임 유저 생성
삭제
SDK 문서
콘솔 가이드
대상
회원번호
상세 검색
검색
검색 조건 초기화

번호	회원번호	회원 아이디	닉네임	가입일	최종 접속일	국가	접속 OS
1	2d622750-04f5-11ee-a3c4-73605164c040	user02	–	2023.06.07 14:35	2023.06.07 14:35	–	Windows 10 &#4010...
2	7814bb60-04f4-11ee-bb76-8582e94b5d40	user01	고박사	2023.06.07 14:30	2023.06.07 16:15	–	Windows 10 &#4010...

1
10개씩 보기

랭킹으로 사용할 유저 정보 데이터 추가

- 랭킹 데이터를 저장하기 위한 private 테이블

Backnd Console

ProjectA

관리자

ProjectA 개발 모드

즐겨찾기

대시보드

서버 설정

프로젝트

인증 정보

푸시

스토어 정보

소셜

예산 알림

요금제

뒤끝베이스

유저 관리

유저 접근 관리

게임 정보 관리

랭킹 관리

우편 관리

푸시 관리

쿠폰 관리

차트 관리

확률 관리

로그 관리

영수증 검증

길드 관리

캐시 관리

게임 정보 관리 | 테이블 생성 | 삭제

전체

테이블명으로 검색

SDK 문서 | 콘솔 가이드

테이블 | 데이터

*테이블은 최대 100개까지 생성할 수 있습니다.

테이블명	설명	분류	스키마 정의	상태
USER_DATA		private	스키마 미정의	활성

1

10개씩 보기

회사소개 이용약관 서비스수준협약 개인정보처리방침

랭킹으로 사용할 유저 정보 데이터 추가

- 테이블 내부에 레벨, 점수 등과 같이 순위의 기준이 되는 숫자형 데이터 row

Backnd Console
ProjectA
관리자
ProjectA 개발 모드
게임 정보 관리 행 생성 삭제 테이블 초기화
SDK 문서 콘솔 가이드
즐겨찾기
대시보드
서버 설정
프로젝트
인증 정보
푸시
스토어 정보
소셜
예산 알림
요금제
뒤끝베이스
유저 관리
유저 접근 관리
게임 정보 관리
랭킹 관리
우편 관리
푸시 관리
쿠폰 관리
차트 관리
확률 관리
로그 관리
영수증 검증
길드 관리
캐시 관리
테이블
데이터
테이블 선택
USER_DATA
상세 검색
검색
검색 조건 초기화

r_id	inDate	owner_inDate	updatedAt	client_date	exp...	gold	heart	jewel	level
1c20-04f2-11ee-a716-9db41e8f5999	2023-06-07T05:19:14.305Z	2023-06-07T05:19:13.634Z	2023-06-07T05:19:14.305Z	2023-06-07T05:19:13.192Z	0	0	30	0	1
5940-04f2-11ee-b4fa-a998ca032138	2023-06-07T05:18:59.437Z	2023-06-07T05:18:58.772Z	2023-06-07T05:18:59.437Z	2023-06-07T05:18:58.352Z	0	0	30	0	1
8290-04f2-11ee-87d2-03970512c2d3	2023-06-07T05:18:42.149Z	2023-06-07T05:18:41.465Z	2023-06-07T05:18:42.149Z	2023-06-07T05:18:41.070Z	0	0	30	0	1

1
10개씩 보기
회사소개 이용약관 서비스수준협약 개인정보처리방침
© AFI, Inc. All rights reserved.

랭킹으로 사용할 유저 정보 데이터 추가

- 랭킹을 생성할 때 private 테이블의 "숫자형" 데이터를 컬럼으로 사용 가능

랭킹 생성

*새로운 랭킹을 사용하기 위해서는 SDK 5.4.0 이상이 필요합니다.

유형: 유저 랭킹 | 길드 랭킹

랭킹명*

초기화 기간: 일간 | 주간 | 월간 | 누적랭킹 | 일회성 랭킹

랭킹 종료 시 컬럼 초기화*: 적용 | 미적용

랭킹 항목 *

테이블: USER_DATA

컬럼: experience

experience
level
jewel
heart
gold

추가 항목

정렬 기준

랭킹 보상

보상 우편 제목

확인 | 취소

Tip. 랭킹 생성/등록을 하려면 "게임 정보 기능"으로 등록한 private 테이블의 "숫자형" 데이터가 있어야 함

랭킹으로 사용할 유저 정보 데이터 추가

일일 최고 점수 데이터 추가

- 일일 최고 점수를 저장하는 정수형 변수 선언
  - UserGameData Script 수정

```
1   [System.Serializable]
2   public class UserGameData
3   {
4       public  int     level;          // Lobby Scene에 보이는 플레이어 레벨
5       public  float   experience;     // Lobby Scene에 보이는 플레이어 경험치
6       public  int     gold;           // 무료 재화
7       public  int     jewel;          // 유료 재화
8       public  int     heart;          // 게임 플레이에 소모되는 재화
9       public  int     dailyBestScore; // 일일 최고 점수
10
11      public void Reset()
12      {
13          level           = 1;
14          experience      = 0;
15          gold            = 0;
16          jewel           = 0;
17          heart           = 30;
18          dailyBestScore  = 0;
19      }
20  }
```

랭킹으로 사용할 유저 정보 데이터 추가

- 유저 정보 추가/불러오기/갱신에 일일 최고 점수 데이터 추가
  - BackendGameData Script 수정

```
30      /// <summary> 뒤끝 콘솔 테이블에 새로운 유저 정보 추가
33      public void GameDataInsert()
34      {
35          // 유저 정보를 초기값으로 설정
36          userGameData.Reset();
37
38          // 테이블에 추가할 데이터로 가공
39          Param param = new Param()
40          {
41              { "level",      userGameData.level },
42              { "experience", userGameData.experience },
43              { "gold",       userGameData.gold },
44              { "jewel",      userGameData.jewel },
45              { "heart",      userGameData.heart },
46              { "dailyBestScore", userGameData.dailyBestScore }
47          };
48
49          // 첫 번째 매개변수는 뒤끝 콘솔의 "게임 정보 관리" 탭에 생성한 테이블 이름
50          Backend.GameData.Insert("USER_DATA", param, callback => ...);
66      }
67
```

랭킹으로 사용할 유저 정보 데이터 추가

- BackendGameData Script 수정 (계속)

```
68      /// <summary> 뒤끝 콘솔 테이블에서 유저 정보를 불러올 때 호출
71      public void GameDataLoad()
72      {
73          Backend.GameData.GetMyData("USER_DATA", new Where(), callback =>
74          {
75              // 게임 정보 불러오기에 성공했을 때
76              if ( callback.IsSuccess() )
77              {
78                  Debug.Log($"게임 정보 데이터 불러오기에 성공했습니다. : {callback}");
79
80                  // JSON 데이터 파싱 성공
81                  try
82                  {
83                      LitJson.JsonData gameDataJson = callback.FlattenRows();
84
85                      // 받아온 데이터의 개수가 0이면 데이터가 없는 것
86                      if ( gameDataJson.Count <= 0 )...
90                      else
91                      {
92                          // 불러온 게임 정보의 고유값
93                          gameDataRowInDate           = gameDataJson[0]["inDate"].ToString();
94                          // 불러온 게임 정보를 userData 변수에 저장
95                          userGameData.level          = int.Parse(gameDataJson[0]["level"].ToString());
96                          userGameData.experience     = float.Parse(gameDataJson[0]["experience"].ToString());
97                          userGameData.gold           = int.Parse(gameDataJson[0]["gold"].ToString());
98                          userGameData.jewel          = int.Parse(gameDataJson[0]["jewel"].ToString());
99                          userGameData.heart          = int.Parse(gameDataJson[0]["heart"].ToString());
100                         userGameData.dailyBestScore = int.Parse(gameDataJson[0]["dailyBestScore"].ToString());
101
102                         onGameDataLoadEvent?.Invoke();
103                     }
104                 }
```

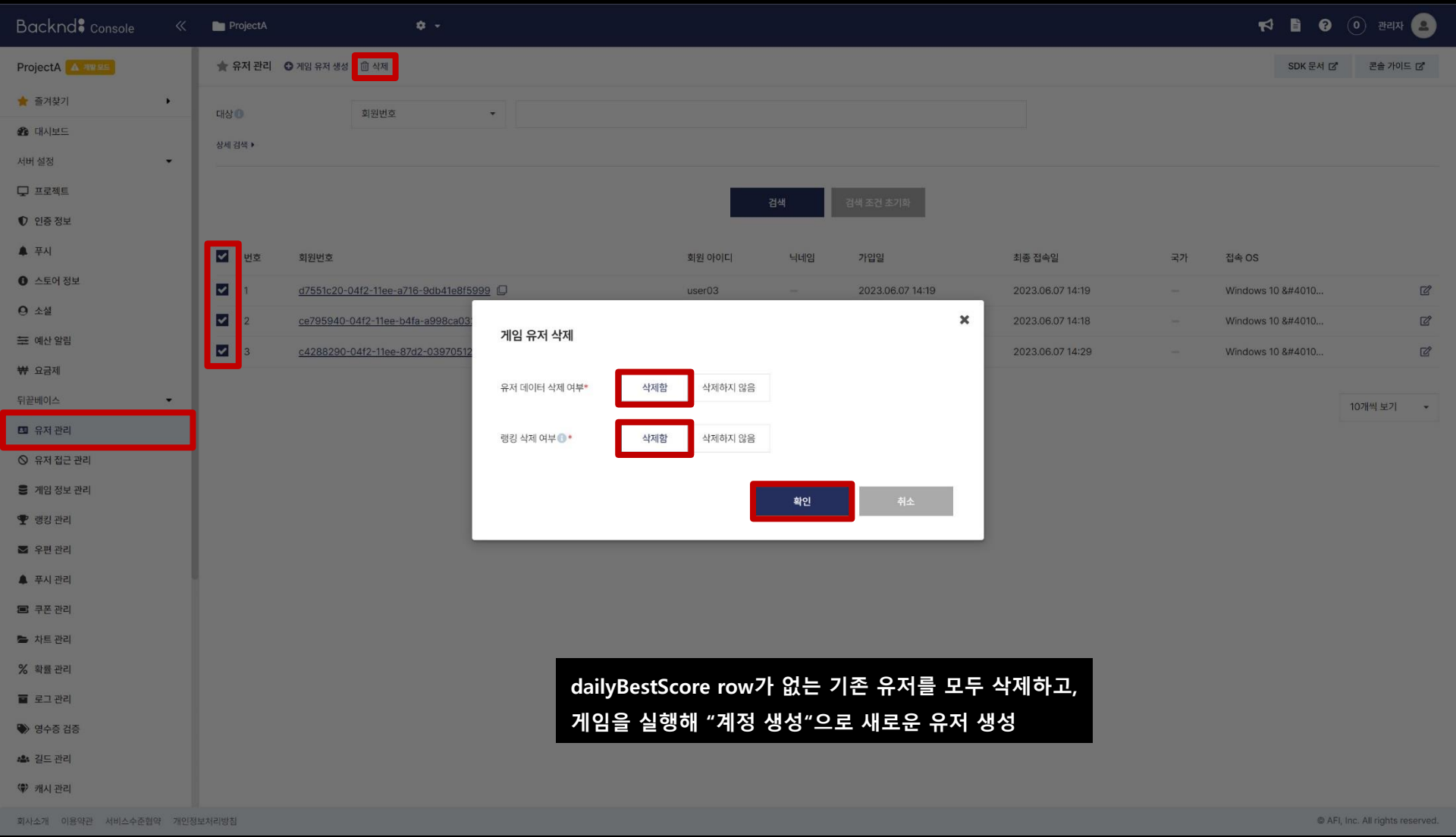
랭킹으로 사용할 유저 정보 데이터 추가

- BackendGameData Script 수정 (계속)

```
122     /// <summary> 뒤끝 콘솔 테이블에 있는 유저 데이터 갱신
125     public void GameDataUpdate(UnityAction action=null)
126     {
127         if ( userGameData == null )...
133
134         Param param = new Param()
135         {
136             { "level",          userGameData.level },
137             { "experience",     userGameData.experience },
138             { "gold",           userGameData.gold },
139             { "jewel",          userGameData.jewel },
140             { "heart",          userGameData.heart },
141             { "dailyBestScore", userGameData.dailyBestScore }
142         };
143
144         // 게임 정보의 고유값(gameDataRowInDate)이 없으면 에러 메시지 출력
145         if ( string.IsNullOrEmpty(gameDataRowInDate) )...
149         // 게임 정보의 고유값이 있으면 테이블에 저장되어 있는 값 중 inDate 컬럼의 값과
150         // 소유하는 유저의 owner_inDate가 일치하는 row를 검색하여 수정하는 UpdateV2() 호출
151         else...
169     }
170 }
```

랭킹으로 사용할 유저 정보 데이터 추가

- 기존 코드로 등록했던 유저 삭제



dailyBestScore row가 없는 기존 유저를 모두 삭제하고,
게임을 실행해 "계정 생성"으로 새로운 유저 생성

랭킹으로 사용할 유저 정보 데이터 추가

- 새로 생성한 계정의 row 데이터

Backnd Console

ProjectA

관리자

ProjectA 개발 모드

즐겨찾기
대시보드
서버 설정
프로젝트
인증 정보
푸시
스토어 정보
소셜
예산 알림
요금제
뒤끝베이스
유저 관리
유저 접근 관리
게임 정보 관리
랭킹 관리
우편 관리
푸시 관리
쿠폰 관리
차트 관리
확률 관리
로그 관리
영수증 검증
길드 관리
캐시 관리

게임 정보 관리 | 행 생성 | 삭제 | 테이블 초기화

SDK 문서 | 콘솔 가이드

테이블 | 데이터

테이블 선택: USER_DATA

상세 검색

검색 | 검색 조건 초기화

	inDate	owner_inDate	updatedAt	client_date	dail...	exp...	gold	heart	jewel	level
11ee-bb76-8582e94b5d40	2023-06-07T05:30:53.541Z	2023-06-07T05:30:52.822Z	2023-06-07T05:30:53.541Z	2023-06-07T05:30:52.426Z	0	0	0	30	0	1

1

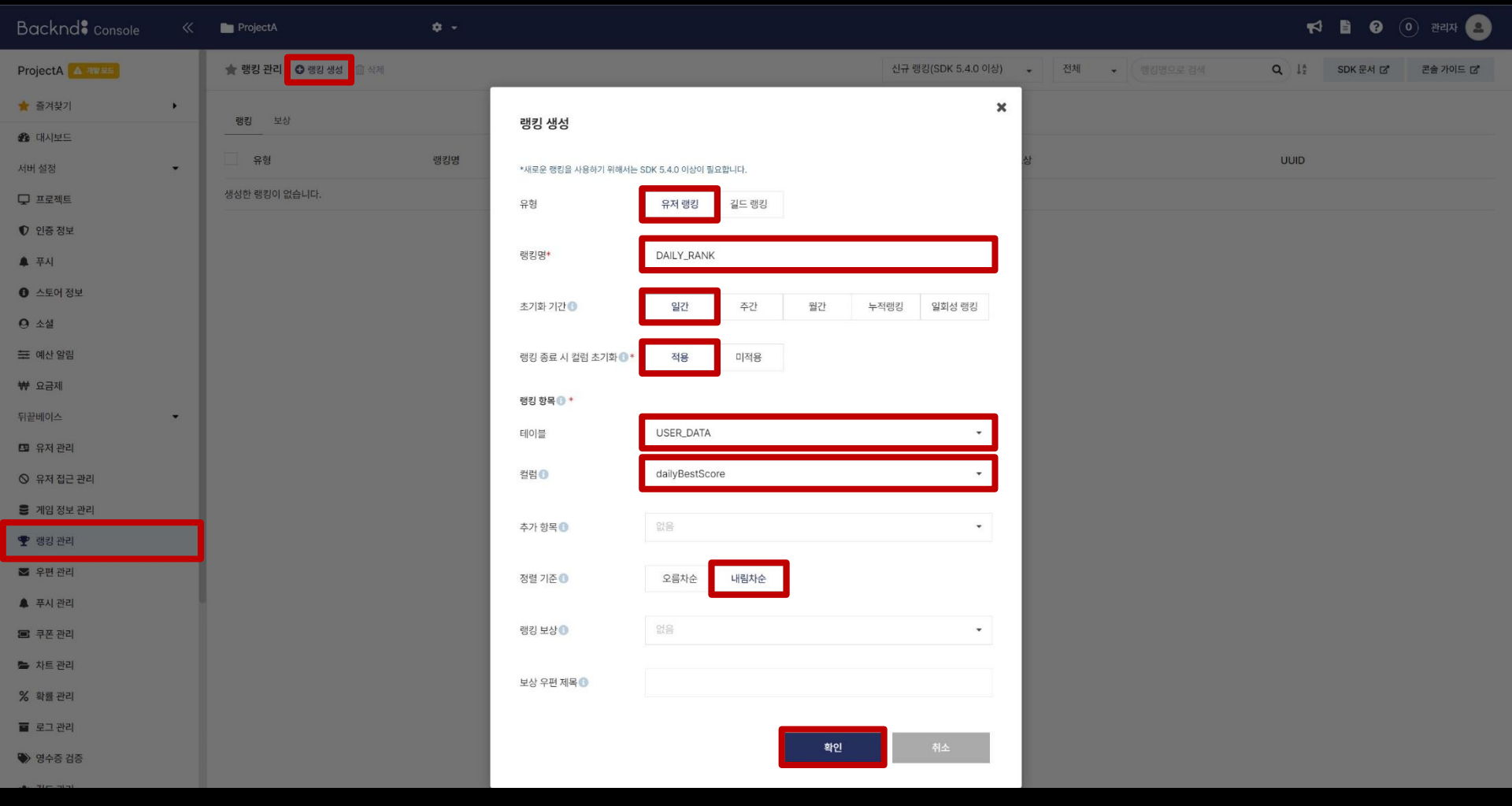
10개씩 보기

회사소개 이용약관 서비스수준협약 개인정보처리방침

랭킹으로 사용할 유저 정보 데이터 추가

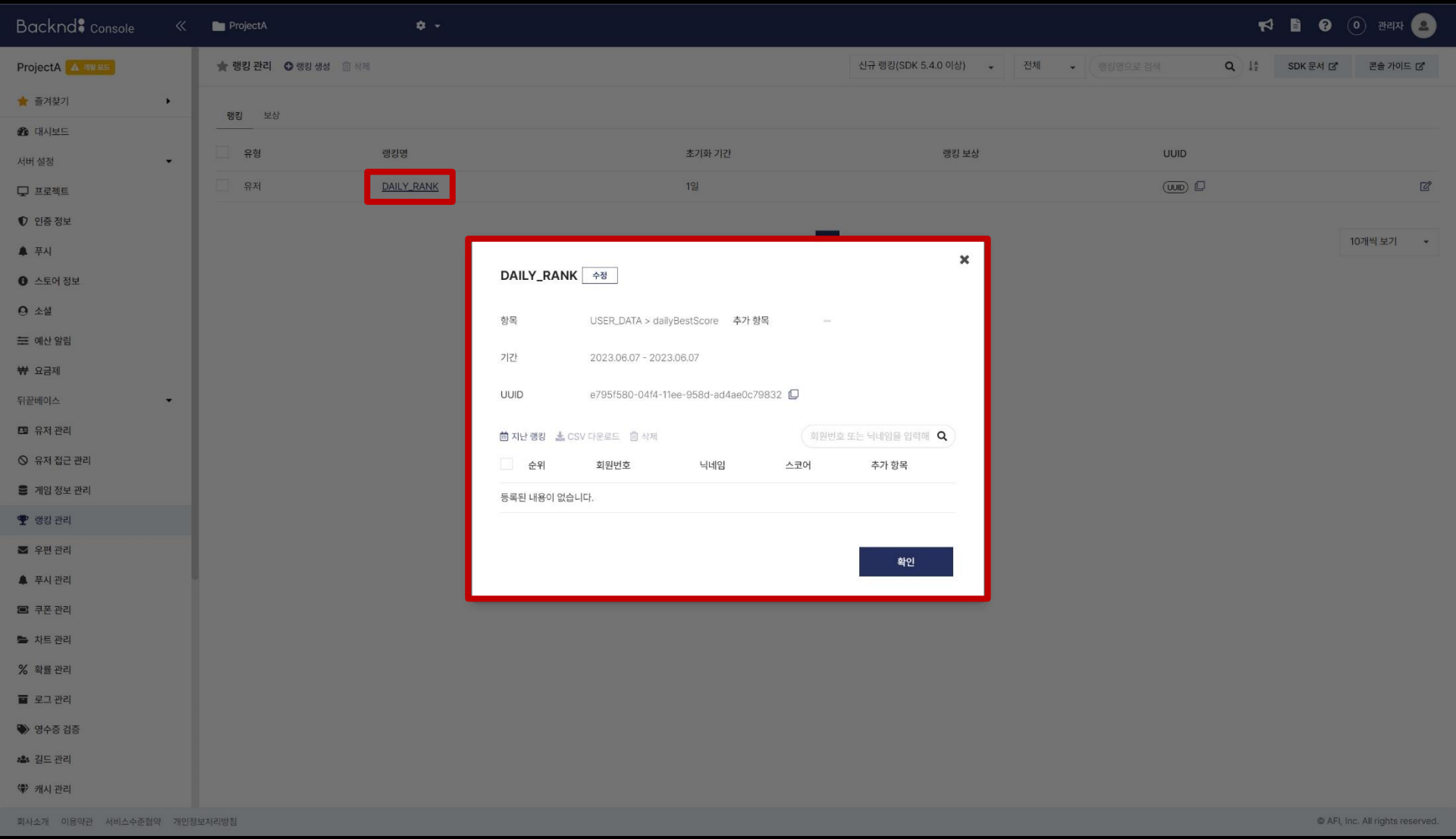
랭킹 생성

- 일일 최고 점수 랭킹 생성



랭킹으로 사용할 유저 정보 데이터 추가

- 일일 최고 점수 랭킹 생성 (계속)



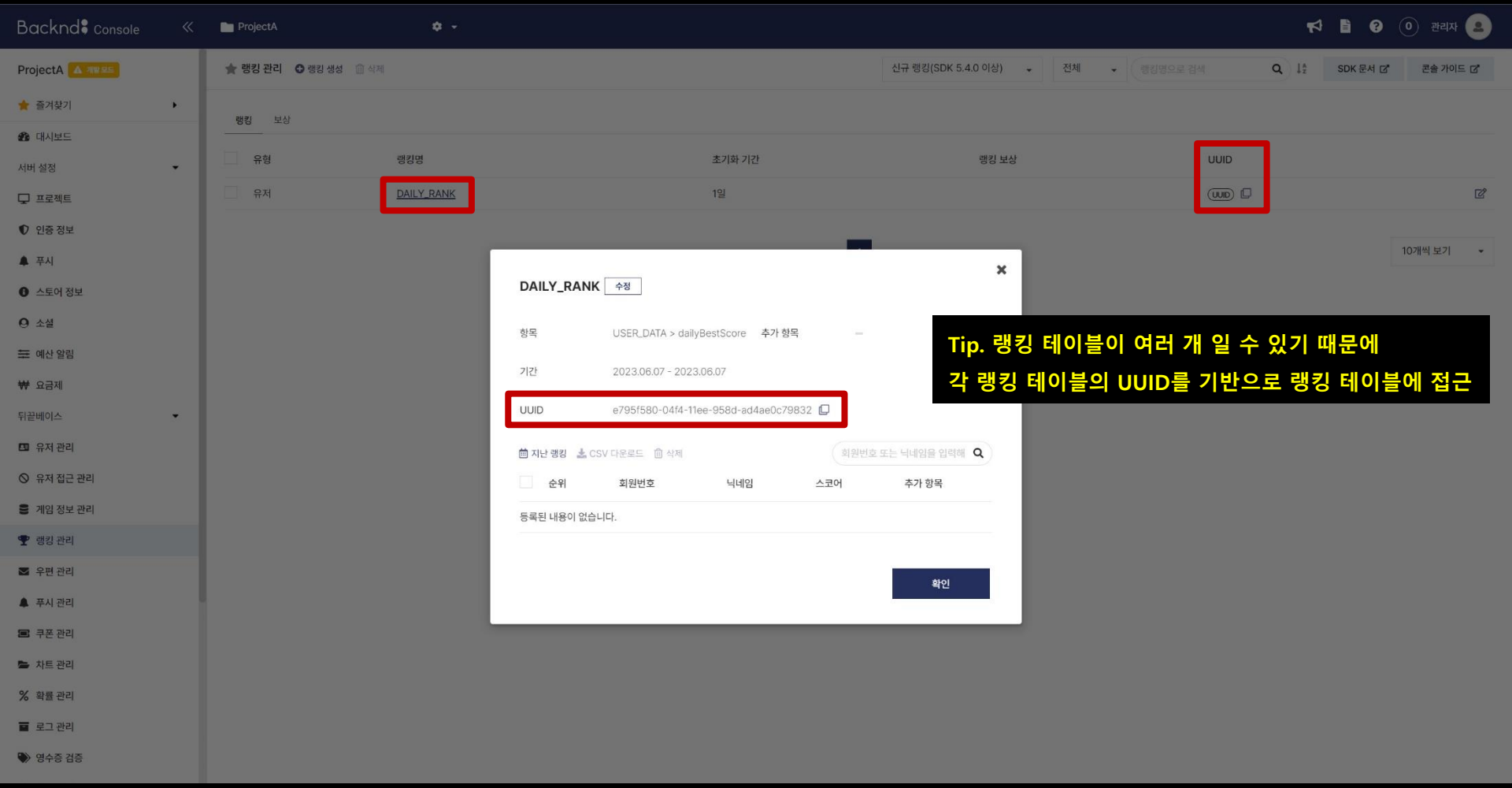
랭킹 데이터 등록

- 랭킹 데이터 등록
- 랭킹 데이터 확인 및 갱신

랭킹 데이터 등록

랭킹 데이터 등록

- 랭킹 테이블의 UUID



랭킹 데이터 등록

- 공용으로 사용하는 상수를 선언하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "Constants"로 변경

```
1  public static class Constants
2  {
3      public static readonly string   USER_DATA_TABLE = "USER_DATA";
4      public static readonly string   DAILY_RANK_UUID = "e795f580-04f4-11ee-958d-ad4ae0c79832";
5  }
```

랭킹 데이터 등록

- 일일 최고 점수를 랭킹에 등록하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "DailyRankRegister"로 변경

```
1  using BackEnd;
2  using UnityEngine;
3
4  public class DailyRankRegister : MonoBehaviour
5  {
6      public void Process(int newScore)
7      {
8          UpdateMyRankData(newScore);
9      }
10
11     private void UpdateMyRankData(int newScore)
12     {
13         string rowInDate = string.Empty;
14
15         // 랭킹 데이터를 업데이트하려면 게임 데이터에서 사용하는 데이터의 inDate 값이 필요
16         Backend.GameData.GetMyData(Constants.USER_DATA_TABLE, new Where(), callback =>
17         {
18             if ( !callback.IsSuccess() )
19             {
20                 Debug.LogError($"데이터 조회 중 문제가 발생했습니다 : {callback}");
21                 return;
22             }
23
```

랭킹 데이터 등록

- 일일 최고 점수를 랭킹에 등록하는 스크립트 생성 및 작성 (계속)

```
24              Debug.Log($"데이터 조회에 성공했습니다 : {callback}");
25
26              if ( callback.FlattenRows().Count > 0 )
27              {
28                  rowInDate = callback.FlattenRows()[0]["inDate"].ToString();
29              }
30              else
31              {
32                  Debug.LogError("데이터가 존재하지 않습니다.");
33                  return;
34              }
35
36              Param param = new Param()
37              {
38                  { "dailyBestScore", newScore }
39              };
40
41              Backend.URank.User.UpdateUserScore(Constants.DAILY_RANK_UUID, Constants.USER_DATA_TABLE, rowInDate, param, callback =>
42              {
43                  if ( callback.IsSuccess() )
44                  {
45                      Debug.Log($"랭킹 등록에 성공했습니다 : {callback}");
46                  }
47                  else
48                  {
49                      Debug.LogError($"랭킹 등록 중 오류가 발생했습니다 : {callback}");
50                  }
51              });
52          });
53      }
54  }
```

== backend method ==

Backend.URank.User.UpdateUserScore(string rankUuid, string tableName, string rowInDate, Param param);

tableName 테이블의 rowInDate row 데이터를 param 값으로 갱신하고, rankUuid 랭킹 테이블의 랭킹 정보 갱신

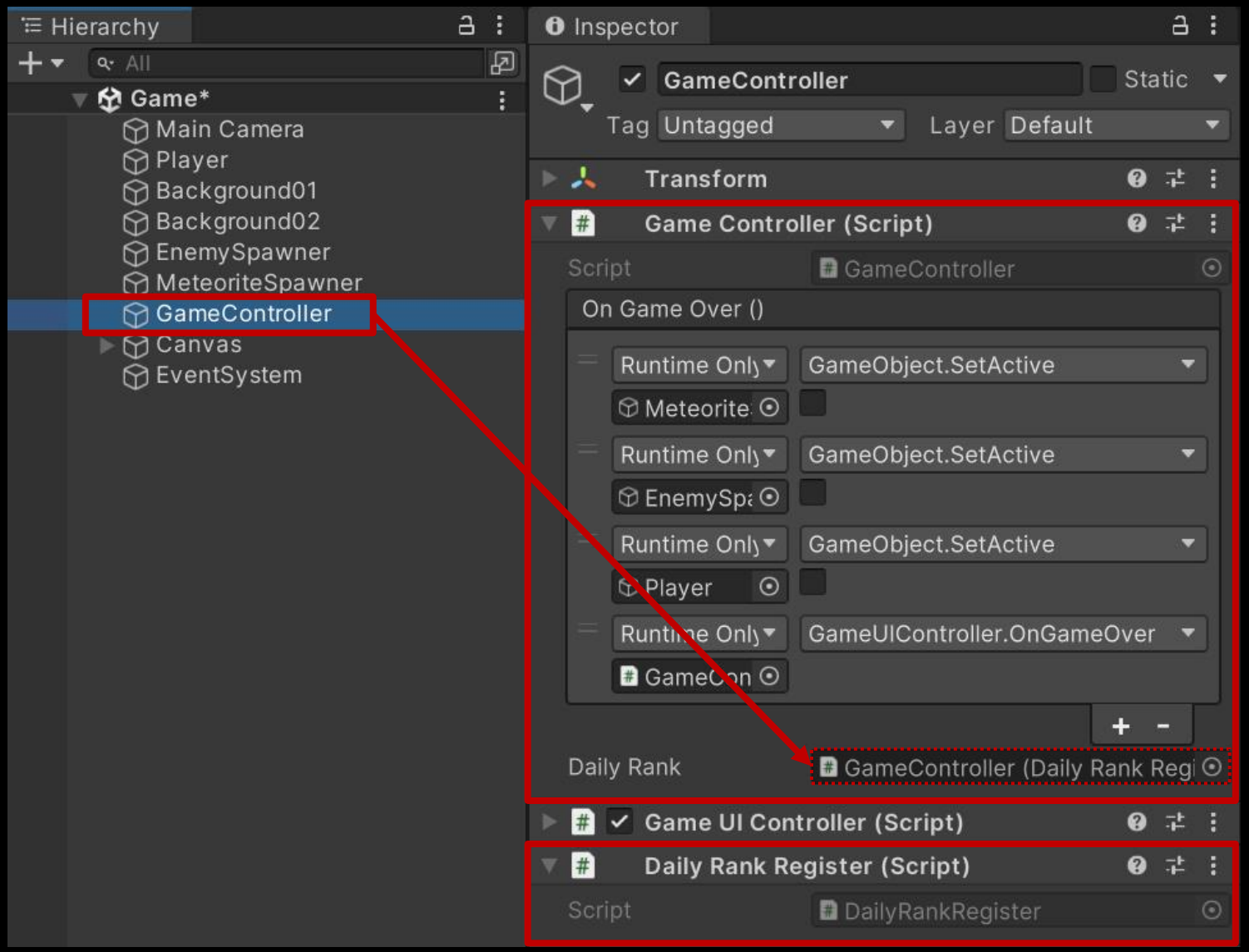
랭킹 데이터 등록

- 게임오버 되었을 때 score를 랭킹에 등록
  - GameController Script 수정

```
4   public class GameController : MonoBehaviour
5   {
6       [SerializeField]
7       private UnityEvent          onGameOver;     // 게임오버 되었을 때 호출할 메소드 등록 및 실행
8       [SerializeField]
9       private DailyRankRegister   dailyRank;
10
11      private int     score = 0;
12
13      public  bool    IsGameOver { set; get; } = false;
14      public  int     Score...
19
20      public void GameOver()
21      {
22          // 중복 처리 되지 않도록 bool 변수로 제어
23          if ( IsGameOver == true ) return;
24
25          IsGameOver = true;
26
27          // 게임오버 되었을 때 호출할 메소드들을 실행
28          onGameOver.Invoke();
29
30          // 현재 점수 정보를 바탕으로 랭킹 데이터 갱신
31          dailyRank.Process(score);
32
33          // 경험치 증가 및 레벨업 여부 검사
34          // (현재 레벨 시스템에 대한 설정이 없기 때문에 경험치의 최대치를 100으로 가정)
35          // (게임을 한번 플레이할 때마다 경험치는 25씩 증가)
36          BackendGameData.Instance.UserGameData.experience += 25;
37          if ( BackendGameData.Instance.UserGameData.experience >= 100 )...
42
43          // 게임 정보 업데이트
44          BackendGameData.Instance.GameDataUpdate();
45      }
46  }
```

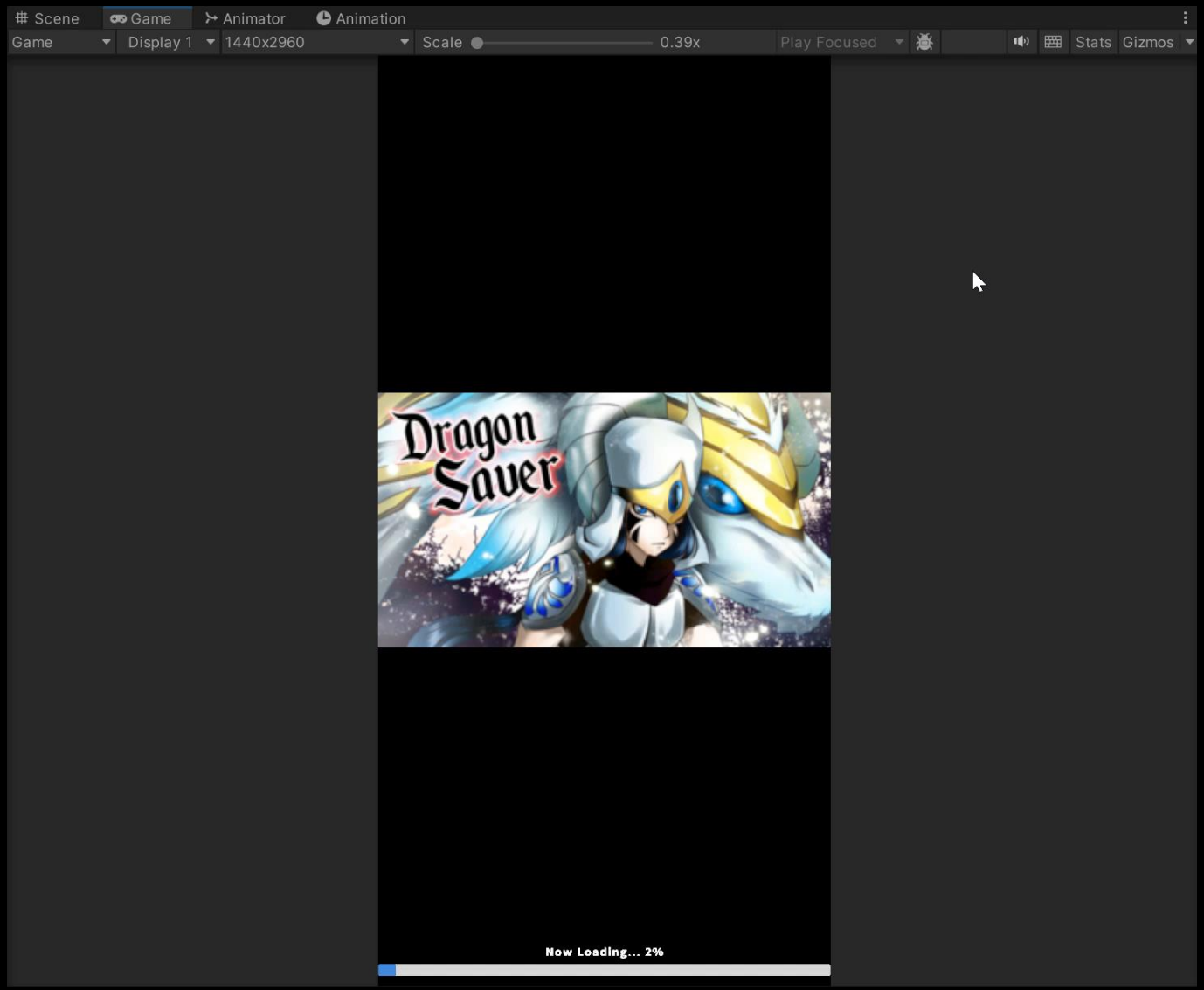
랭킹 데이터 등록

- GameController 오브젝트에 "DailyRankRegister" 컴포넌트 추가 및 설정



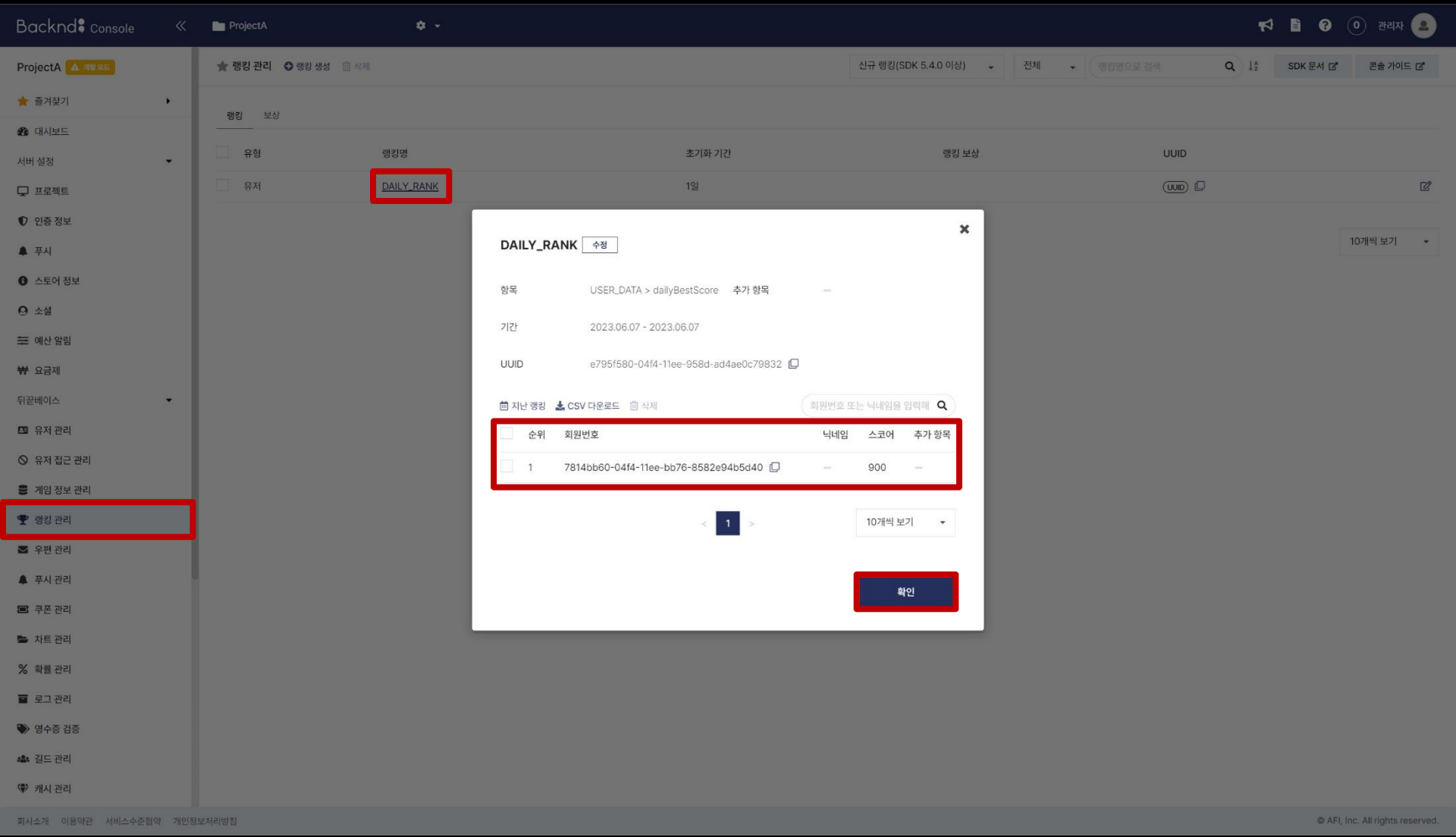
랭킹 데이터 등록

- 결과 화면



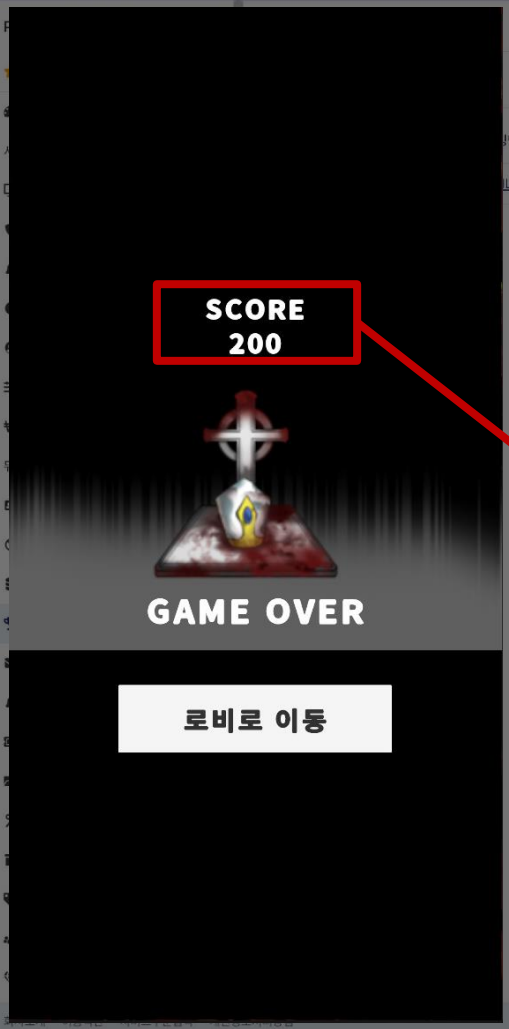
랭킹 데이터 등록

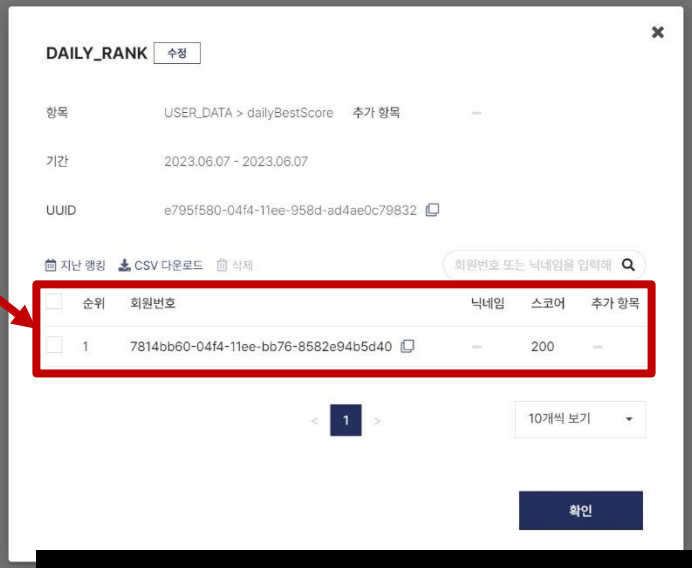
- 결과 화면 (Backend Console)



랭킹 데이터 등록

- 결과 화면 (Backend Console)





현재는 게임오버 될 때마다 아무 조건 없이 랭킹 데이터를 갱신하기 때문에
900 -> 200으로 더 낮은 점수를 획득해도 랭킹 데이터가 갱신된다.

랭킹 데이터 등록

- **랭킹 데이터 확인 및 갱신**
  - 내 랭킹 데이터를 불러와 점수를 비교하고, 최고 점수 일 때만 랭킹 갱신
    - DailyRankRegister Script 수정

```
1   using BackEnd;
2   using UnityEngine;
3
4   public class DailyRankRegister : MonoBehaviour
5   {
6       public void Process(int newScore)
7       {
8           //UpdateMyRankData(newScore);
9           UpdateMyBestRankData(newScore);
10      }
11
12      private void UpdateMyRankData(int newScore)...
55
```

랭킹 데이터 등록

- DailyRankRegister Script 수정 (계속)

```
56  private void UpdateMyBestRankData(int newScore)
57  {
58      Backend.URank.User.GetMyRank(Constants.DAILY_RANK_UUID, callback =>
59      {
60          if ( callback.IsSuccess() )
61          {
62              // JSON 데이터 파싱 성공
63              try...
96          }
97          else
98          {
99              // 자신의 랭킹 정보가 존재하지 않을 때는 현재 점수를 새로운 랭킹으로 등록
100             if ( callback.GetMessage().Contains("userRank") )
101             {
102                 UpdateMyRankData(newScore);
103
104                 Debug.Log($"새로운 랭킹 데이터 생성 및 등록 : {callback}");
105             }
106         }
107     });
108 }
109 }
```

뒷장

== backend method ==

Backend.URank.User.GetMyRank(string rankUuid, int gap=0);

rankUuid 랭킹 테이블에 등록되어 있는 내 랭킹 조회, gap 값을 설정하면 내 랭킹 위/아래 인접 유저 검색 가능

gap이 3일 때 내 랭킹이 10위이면 10-3 ~ 10+3까지 7~13위의 유저 랭킹이 조회된다.

랭킹 데이터 등록

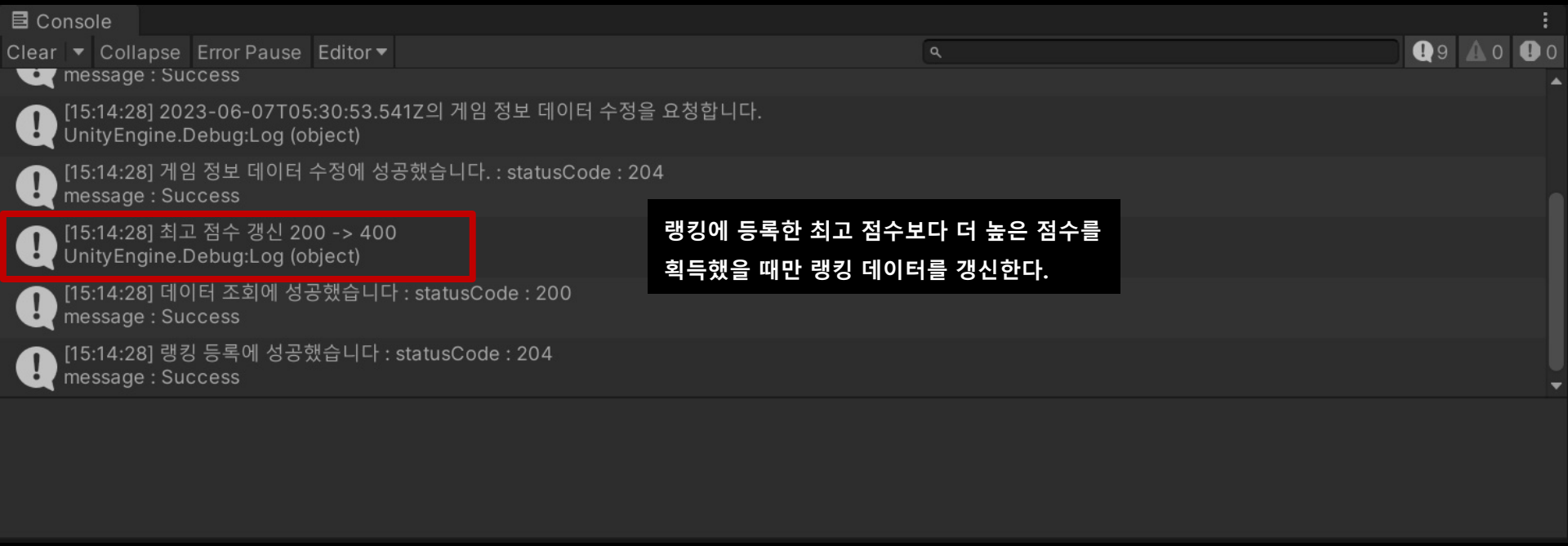
- DailyRankRegister Script 수정 (계속)

```
62
63
64
65
66
67
68
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
```

```
// JSON 데이터 파싱 성공
try
{
    LitJson.JsonData rankDataJson = callback.FlattenRows();

    // 받아온 데이터의 개수가 0이면 데이터가 없는 것
    if ( rankDataJson.Count <= 0 )
    {
        Debug.LogWarning("데이터가 존재하지 않습니다.");
    }
    else
    {
        // 랭킹을 등록할 때는 컬럼명을 "dailyBestScore"로 저장했지만
        // 랭킹을 불러올 때는 컬럼명이 "score"로 통일되어 있다.

        // 추가로 등록한 항목은 컬럼명을 그대로 사용
        int bestScore = int.Parse(rankDataJson[0]["score"].ToString());

        // 현재 점수가 최고 점수보다 높으면
        if ( newScore > bestScore )
        {
            // 현재 점수를 새로운 최고 점수로 설정하고, 랭킹에 등록
            UpdateMyRankData(newScore);

            Debug.Log($"최고 점수 갱신 {bestScore} -> {newScore}");
        }
    }
}
// JSON 데이터 파싱 실패
catch ( System.Exception e )
{
    // try-catch 에러 출력
    Debug.LogError(e);
}
```

랭킹 데이터 등록

- 결과 화면



랭킹에 등록한 최고 점수보다 더 높은 점수를 획득했을 때만 랭킹 데이터를 갱신한다.