게임엔진프로그래밍응용

11. 탑다운 슈터 게임 3

청강문화산업대학교 게임콘텐츠스쿨

반 경 진

코로나 유의사항

공지사항

- 2023학년도 1학기 방역 및 학사운영 방안
  https://www.ck.ac.kr/archives/193175
- 2023학년도 1학기 국가공휴일 및 대학 행사 수업 대체 일정 공지
  https://www.ck.ac.kr/archives/193109

온라인 수업 저작권 유의 사항

온라인 수업 저작권 유의 사항

온라인수업 저작권 유의사항 안내

강의 저작물을 다운로드, 캡처하여
교외로 유출하는 행위는
불 법 입 니 다

저작권자의 허락 없이 저작물을 복제, 공중송신 또는 배포하는 것은
저작권 침해에 해당하며 저작권법에 처벌받을 수 있습니다.

강의 동영상과 자료 일체는 교수 및 학교의 저작물로서 저작권이 보호됩니다.
수업자료를 무단 복제 또는 배포, 전송 시 민형사상 책임을 질 수 있습니다.

Index

UI, 게임 매니저

UI, 게임 매니저

- HUD UI

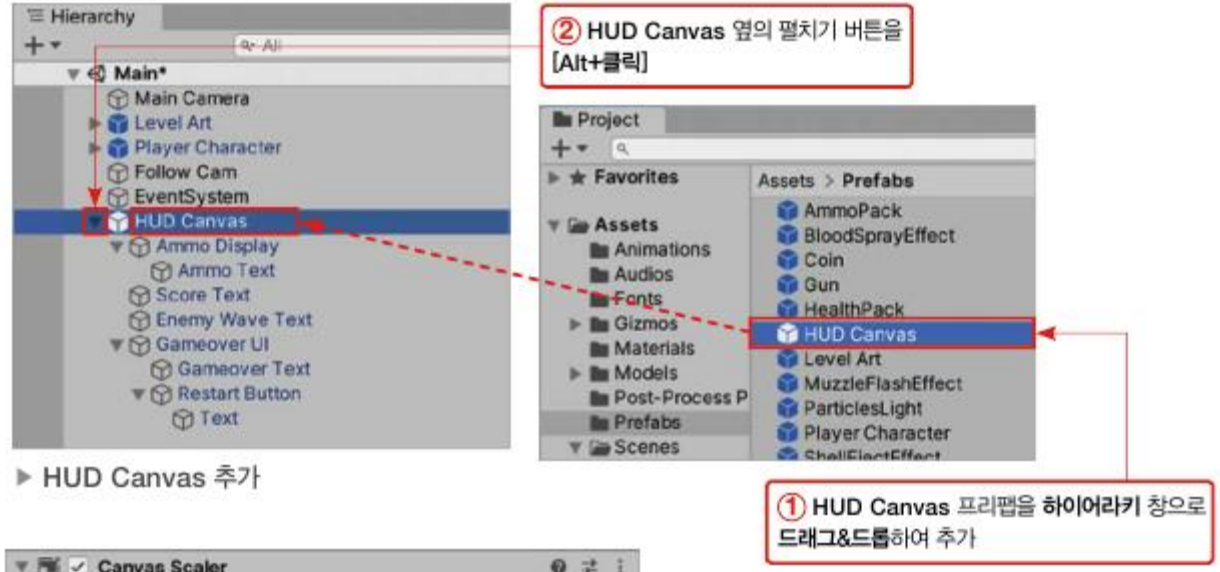
▶ HUD Canvas의 모습

UI, 게임 매니저

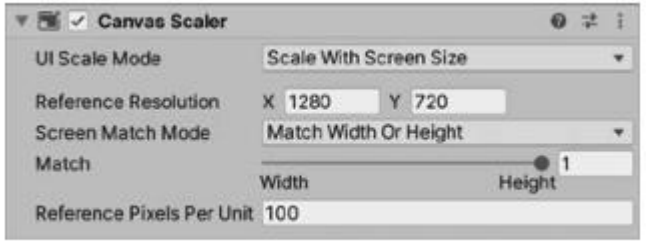
[과정 01] HUD Canvas 추가

① Prefabs 폴더의 **HUD Canvas** 프리팹을 **하이어라키** 창으로 **드래그&드롭**

② **하이어라키** 창에서 **HUD Canvas** 옆의 **펼치기 버튼**을 **[Alt+클릭]**하여 자식 리스트 모두 펼치기



▶ HUD Canvas 추가



▶ HUD Canvas의 캔버스 스케일러 컴포넌트

- **Ammo Display** : 남은 탄알을 표시하는 창
  - **Ammo Text** : 탄알을 표시하는 텍스트
- **Score Text** : 점수를 표시하는 텍스트
- **Enemy Wave Text** : 현재 적 웨이브와 남은 적 수를 표시하는 텍스트
- **Gameover UI** : 게임오버 시 활성화되는 패널
  - **Gameover Text** : 게임오버 안내 텍스트
  - **Restart Button** : 게임 재시작 버튼
    - **Text** : 버튼의 텍스트

▶ HUD Canvas의 UIManager

UI, 게임 매니저

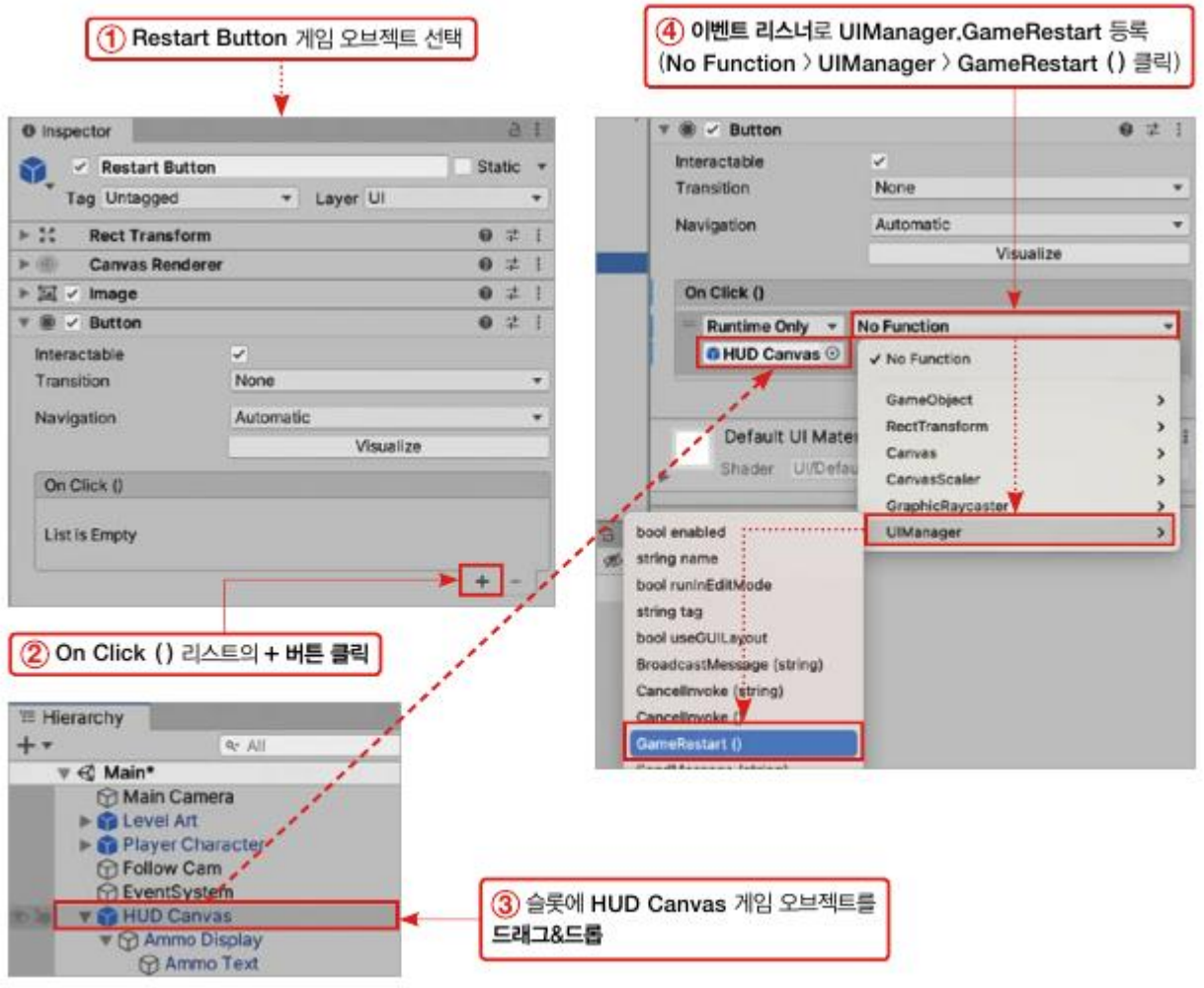
[과정 01] 재시작 버튼 설정

① **하이어라키** 창에서 **Restart Button** 게임 오브젝트 선택

② **Button** 컴포넌트의 **On Click ()** 리스트의 **+ 버튼 클릭**

③ 생성된 슬롯에 **HUD Canvas** 게임 오브젝트를 **드래그&드롭**

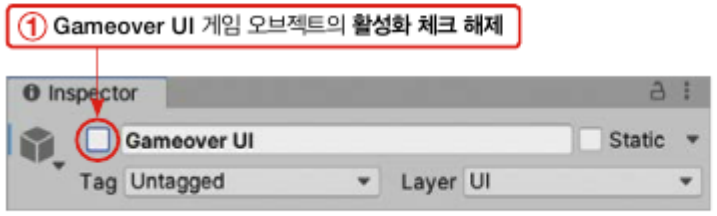
④ **이벤트 리스너**로 `UIManager.GameRestart` 등록(**No Function** 〉 **UIManager** 〉 **GameRestart()** 클릭)



▶ 재시작 버튼 설정

[과정 02] Gameover UI 비활성화

① **하이어라키** 창에서 **Gameover UI** 게임 오브젝트 선택 〉 **인스펙터** 창에서 **활성화 체크 해제**

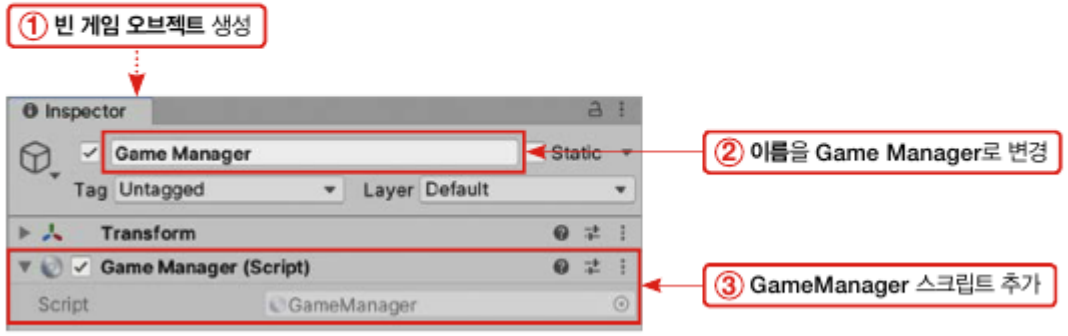


▶ Gameover UI 비활성화

UI, 게임 매니저

[과정 01] 게임 매니저 추가

① **빈 게임 오브젝트** 생성(+ > **Create Empty**)
② 생성된 빈 게임 오브젝트의 **이름**을 **Game Manager**로 변경
③ **Game Manager** 게임 오브젝트에 **GameManager** 스크립트 추가(**Scripts** 폴더의 **GameManager** 스크립트를 **Game Manager** 게임 오브젝트로 **드래그&드롭**)



▶ 게임 매니저 추가

- 싱글턴으로 존재
- 점수 관리
- 게임오버 상태 관리
- UIManager를 이용해 점수와 게임오버 UI 갱신

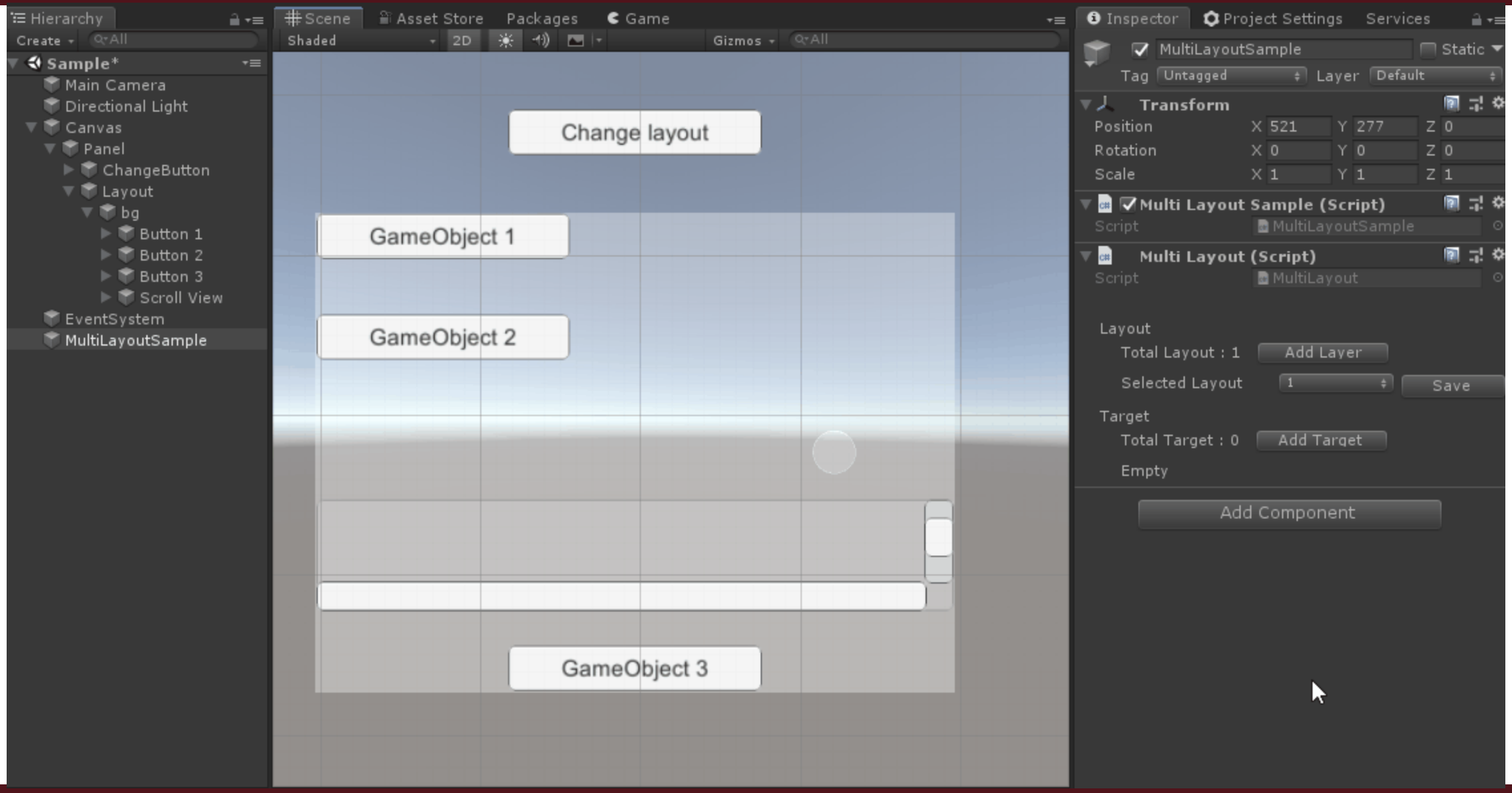
UI, 게임 매니저

- Game Package Manager for Unity (https://github.com/nhn/gpm.unity) 소개
  https://assetstore.unity.com/packages/tools/utilities/game-package-manager-147711
  - WebView
  - LogViewer
  - AssetManagement
  - UI
  - Profiler
  - Adapter
  - DLST

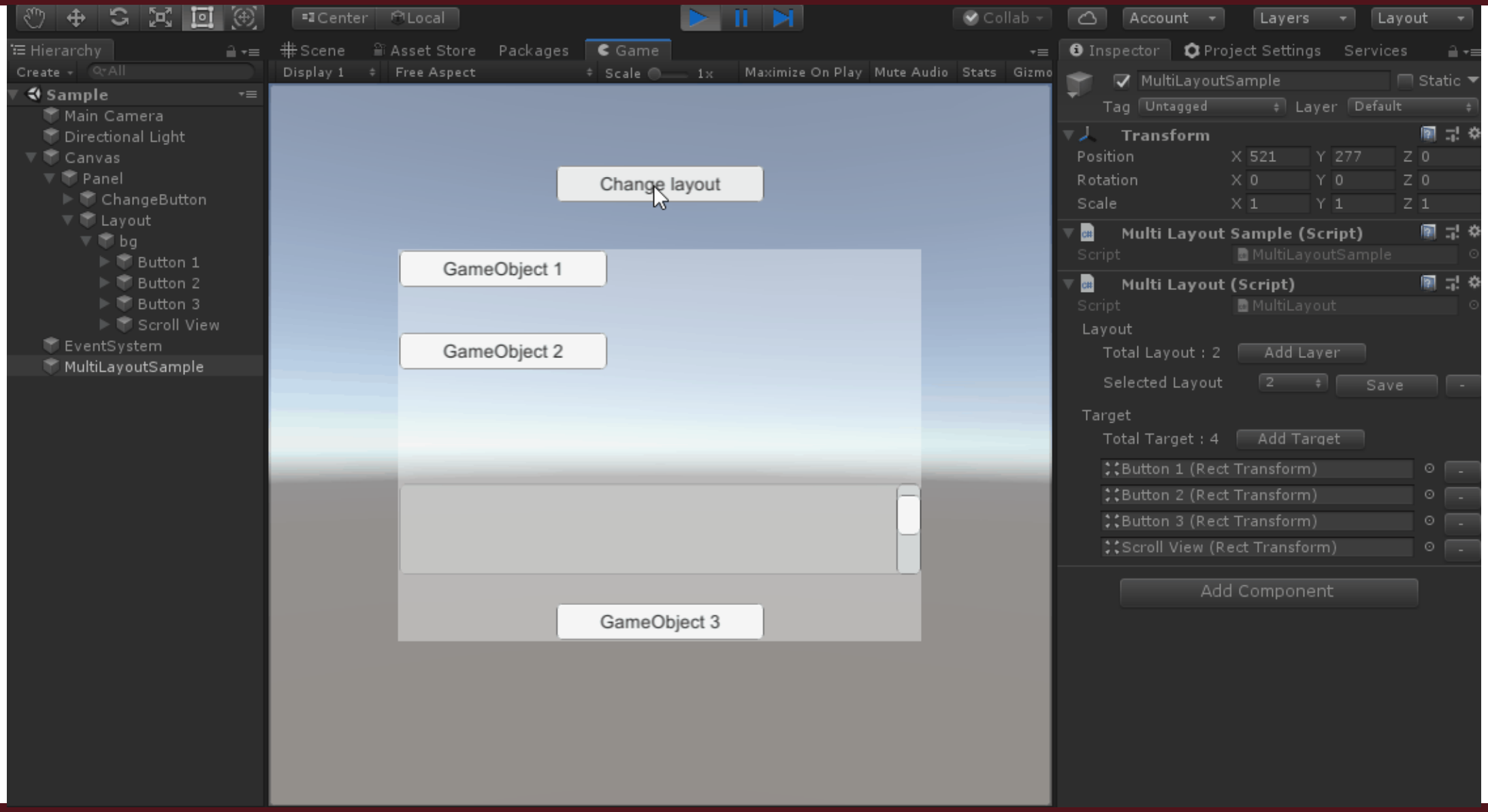
UI, 게임 매니저

- Game Package Manager for Unity
  - UI
    1. Multi Layout
    2. Infinite Scroll
    3. DraggableRect
    4. ContentSizeSetter
    5. WrapLayoutGroup

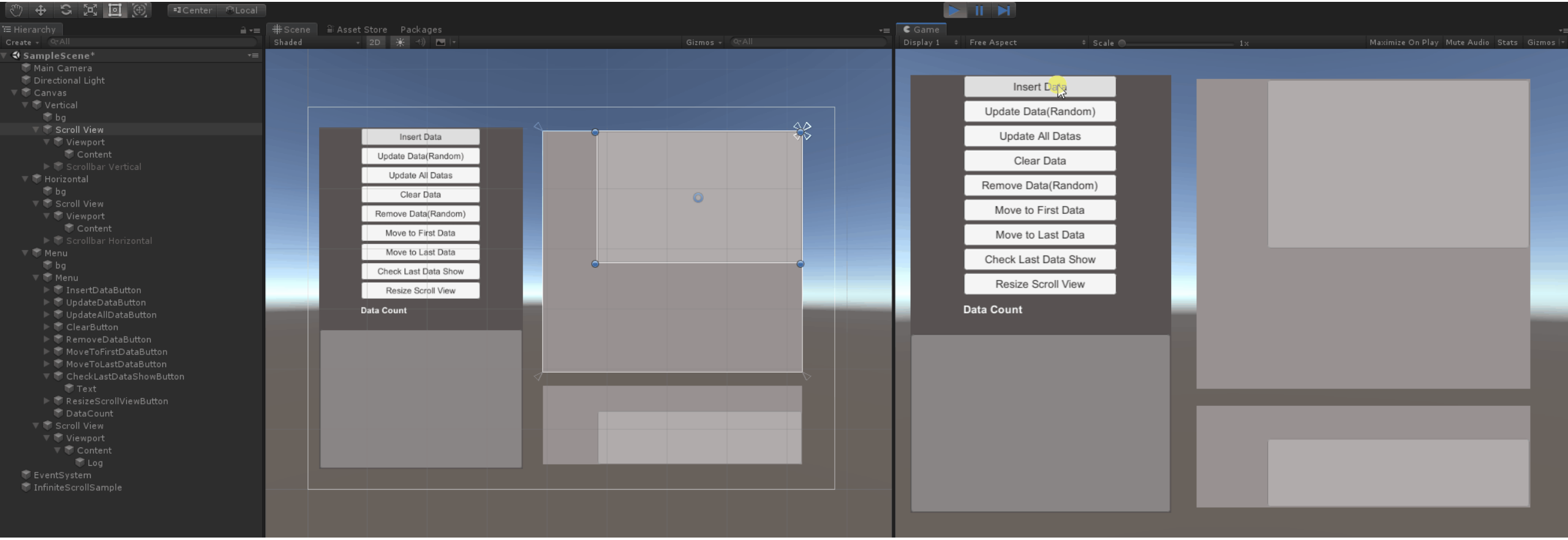
UI, 게임 매니저


Hierarchy
Create
All
Sample*
Main Camera
Directional Light
Canvas
Panel
ChangeButton
Layout
bg
Button 1
Button 2
Button 3
Scroll View
EventSystem
MultiLayoutSample
Scene
Asset Store
Packages
Game
Shaded
2D
Gizmos
All
Change layout
GameObject 1
GameObject 2
GameObject 3
Inspector
Project Settings
Services
MultiLayoutSample
Static
Tag
Untagged
Layer
Default
Transform
Position
X 521
Y 277
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Multi Layout Sample (Script)
Script
MultiLayoutSample
Multi Layout (Script)
Script
MultiLayout
Layout
Total Layout : 1
Add Layer
Selected Layout
1
Save
Target
Total Target : 0
Add Target
Empty
Add Component

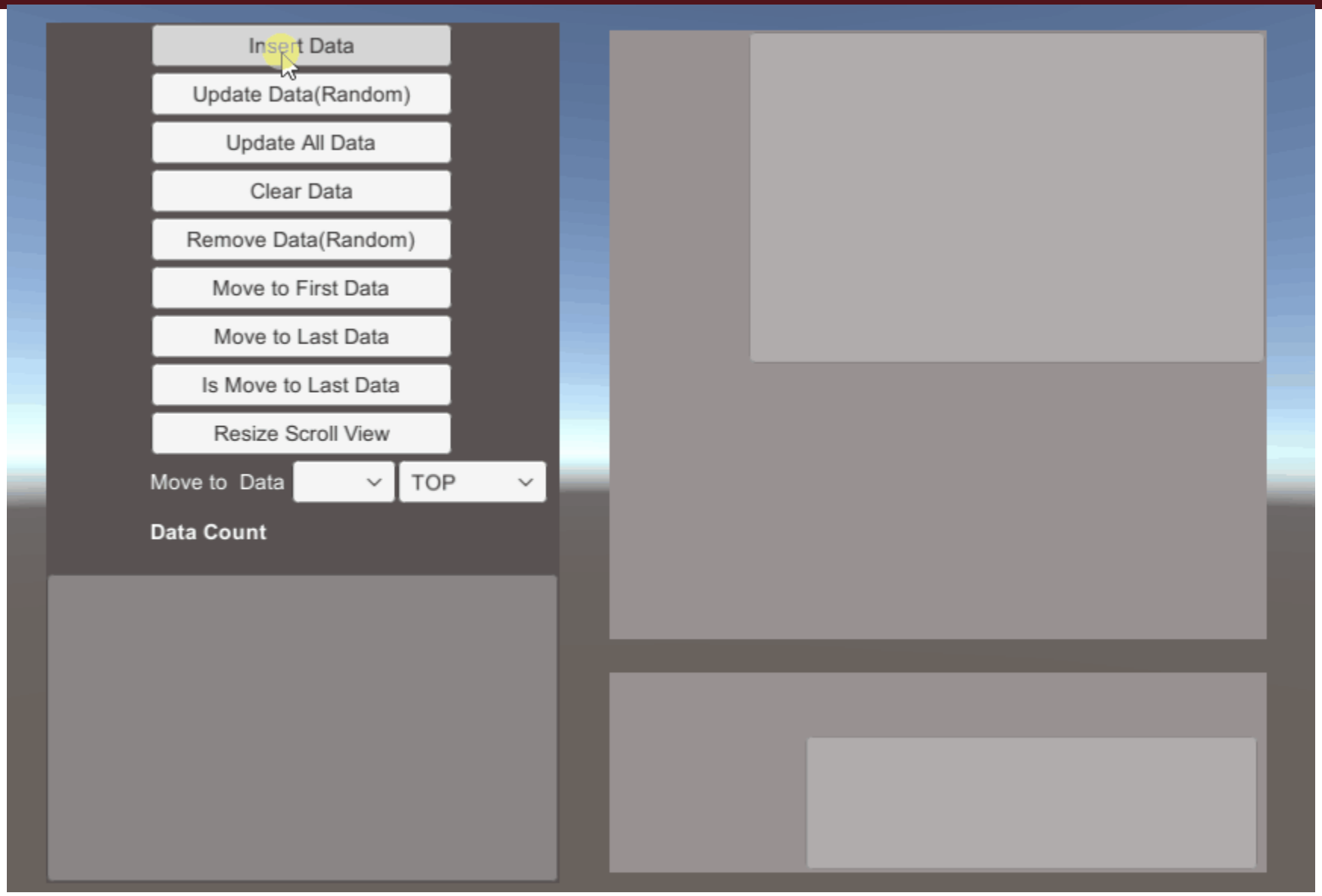
UI, 게임 매니저


Center
Local
Collab
Account
Layers
Layout
Hierarchy
Create
Sample
Main Camera
Directional Light
Canvas
Panel
ChangeButton
Layout
bg
Button 1
Button 2
Button 3
Scroll View
EventSystem
MultiLayoutSample
Scene
Asset Store
Packages
Game
Display 1
Free Aspect
Scale
1x
Maximize On Play
Mute Audio
Stats
Gizmo
Change layout
GameObject 1
GameObject 2
GameObject 3
Inspector
Project Settings
Services
MultiLayoutSample
Static
Tag
Untagged
Layer
Default
Transform
Position
Rotation
Scale
Multi Layout Sample (Script)
Script
MultiLayoutSample
Multi Layout (Script)
MultiLayout
Layout
Total Layout : 2
Add Layer
Selected Layout
Save
Target
Total Target : 4
Add Target
Button 1 (Rect Transform)
Button 2 (Rect Transform)
Button 3 (Rect Transform)
Scroll View (Rect Transform)
Add Component

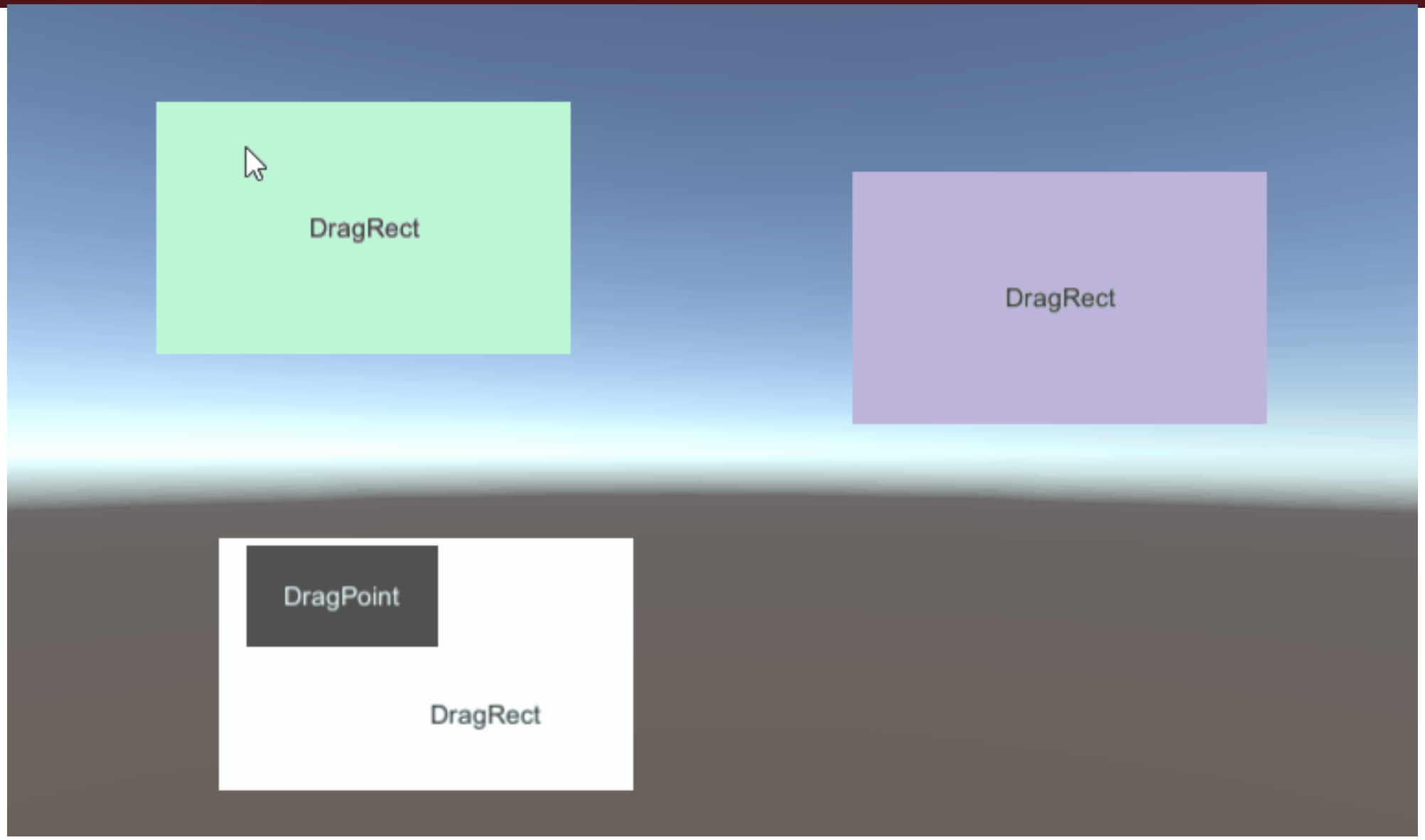
UI, 게임 매니저


Hierarchy
Create
SampleScene*
Main Camera
Directional Light
Canvas
Vertical
bg
Scroll View
Viewport
Content
Scrollbar Vertical
Horizontal
bg
Scroll View
Viewport
Content
Scrollbar Horizontal
Menu
bg
Menu
InsertDataButton
UpdateDataButton
UpdateAllDataButton
ClearButton
RemoveDataButton
MoveToFirstDataButton
MoveToLastDataButton
CheckLastDataShowButton
Text
ResizeScrollViewButton
DataCount
Scroll View
Viewport
Content
Log
EventSystem
InfiniteScrollSample
Scene
Asset Store
Packages
Shaded
2D
Gizmos
Insert Data
Update Data(Random)
Update All Datas
Clear Data
Remove Data(Random)
Move to First Data
Move to Last Data
Check Last Data Show
Resize Scroll View
Data Count
Game
Display 1
Free Aspect
Scale
Maximize On Play
Mute Audio
Stats
Gizmos
Insert Data
Update Data(Random)
Update All Datas
Clear Data
Remove Data(Random)
Move to First Data
Move to Last Data
Check Last Data Show
Resize Scroll View
Data Count

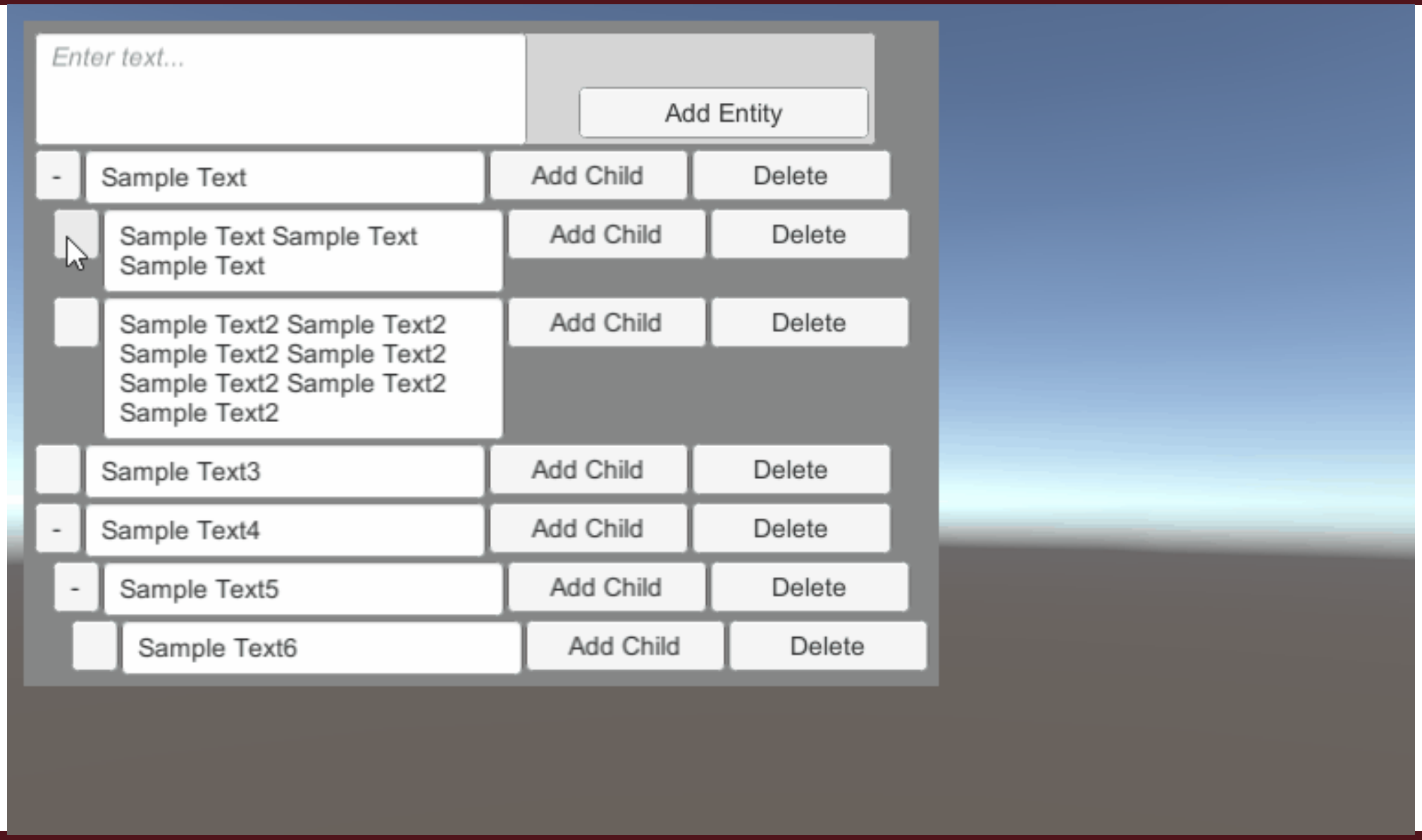
UI, 게임 매니저


Insert Data
Update Data(Random)
Update All Data
Clear Data
Remove Data(Random)
Move to First Data
Move to Last Data
Is Move to Last Data
Resize Scroll View
Move to Data
TOP
Data Count

UI, 게임 매니저



UI, 게임 매니저


Enter text...
Add Entity
-
Sample Text
Add Child
Delete
Sample Text Sample Text
Sample Text
Add Child
Delete
Sample Text2 Sample Text2
Sample Text2 Sample Text2
Sample Text2 Sample Text2
Sample Text2
Add Child
Delete
Sample Text3
Add Child
Delete
-
Sample Text4
Add Child
Delete
-
Sample Text5
Add Child
Delete
Sample Text6
Add Child
Delete

UI, 게임 매니저

좀비 생성

좀비 생성

[과정 01] ZombieData 스크립트 열기

① **Scripts** 폴더의 **ZombieData** 스크립트를 **더블 클릭**으로 열기

열린 스크립트는 다음과 같습니다.

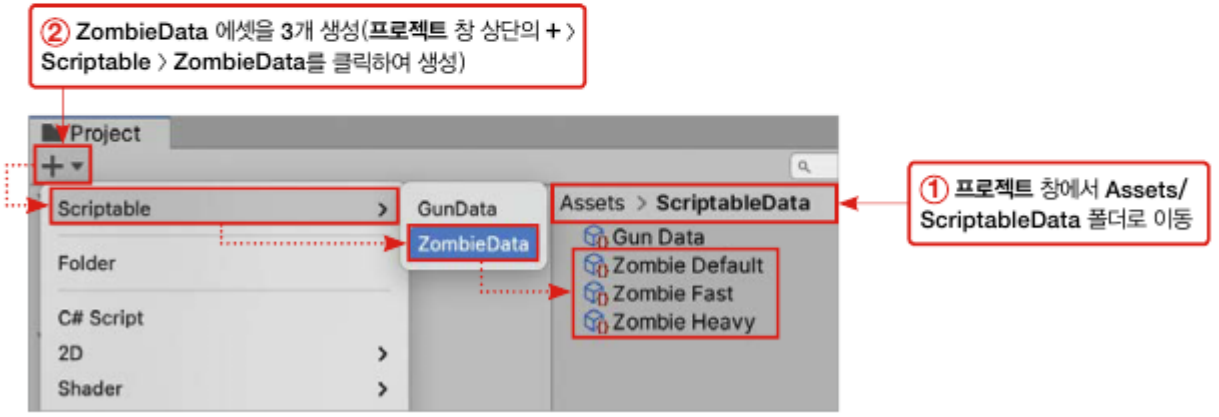
```
using UnityEngine;

// 좀비 생성시 사용할 셋업 데이터
[CreateAssetMenu(menuName = "Scriptable/ZombieData", fileName = "Zombie Data")]
public class ZombieData : ScriptableObject {
    public float health = 100f; // 체력
    public float damage = 20f; // 공격력
    public float speed = 2f; // 이동 속도
    public Color skinColor = Color.white; // 피부색
}
```

[과정 02] 좀비 셋업 데이터 생성하기

① **프로젝트** 창에서 **Assets/ScriptableData** 폴더로 이동

② **ScriptableData** 폴더에 **ZombieData** 에셋을 다음과 같은 이름으로 **3**개 생성(**프로젝트** 창 상단의 **+** 〉 **Scriptable** 〉 **ZombieData**를 클릭하여 생성)

- **Zombie Default**
- **Zombie Fast**
- **Zombie Heavy**



▶ 좀비 셋업 데이터 생성하기

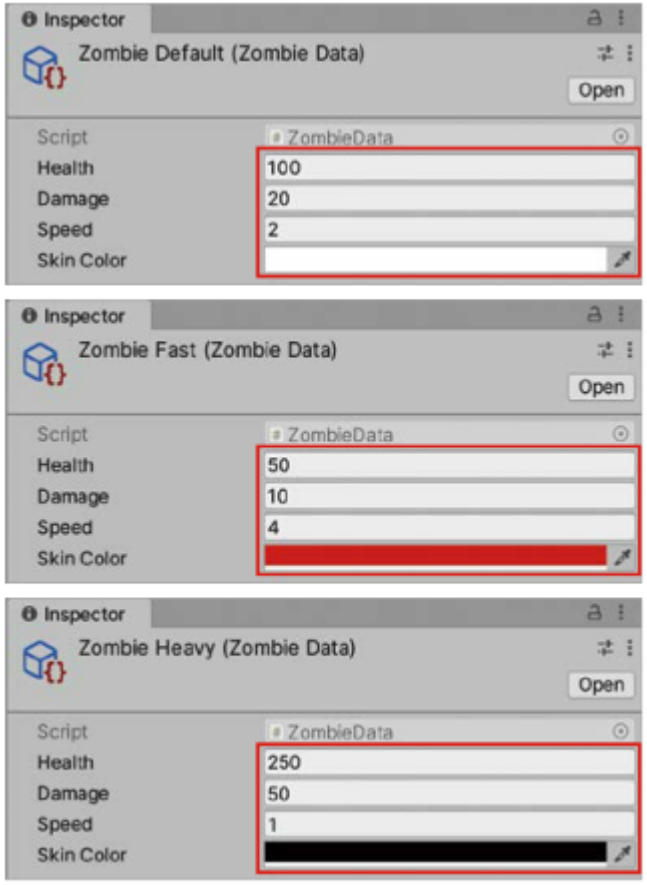
좀비 생성

[과정 03] 좀비 셋업 데이터 필드 생성하기

① Zombie Default는 기본값을 사용

② Zombie Fast의 설정은 다음과 같이 변경
- Health를 50으로 변경
- Damage를 10으로 변경
- Speed를 4로 변경
- SkinColor를 (255, 0, 0)으로 변경

③ Zombie Heavy의 설정은 다음과 같이 변경
- Health를 250으로 변경
- Damage를 50으로 변경
- Speed를 1로 변경
- SkinColor를 (0, 0, 0)으로 변경



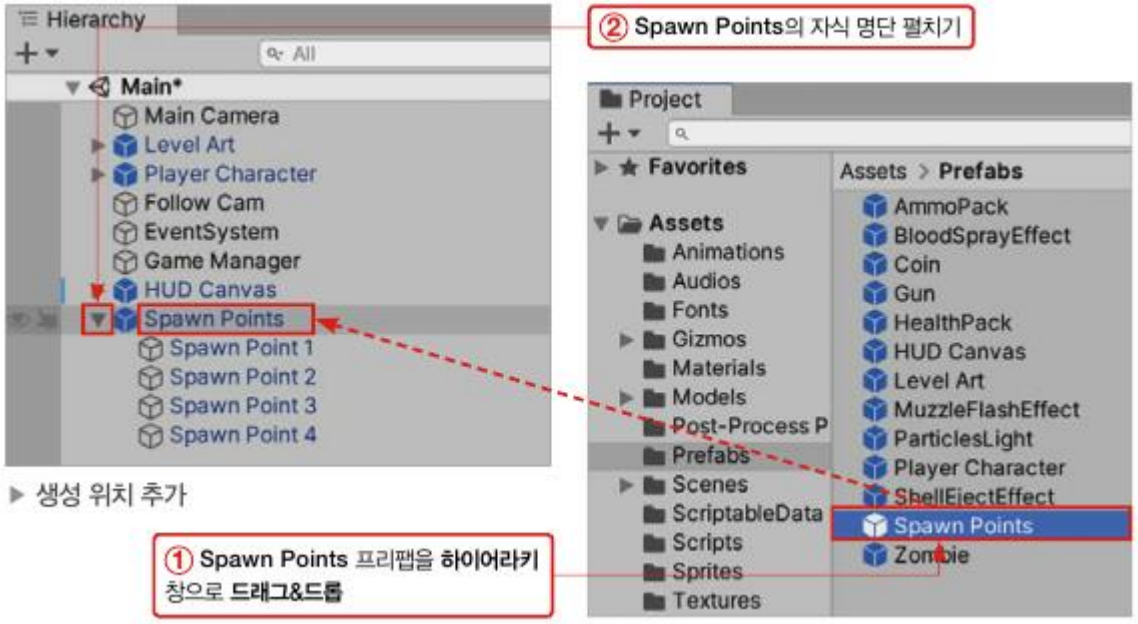
▶ 좀비 셋업 데이터 필드 생성하기

좀비 생성

[과정 01] 생성 위치 추가

① Prefabs 폴더의 Spawn Points 프리팹을 **하이어라키** 창으로 **드래그&드롭**

② **하이어라키** 창에서 Spawn Points 게임 오브젝트의 자식 명단 펼치기



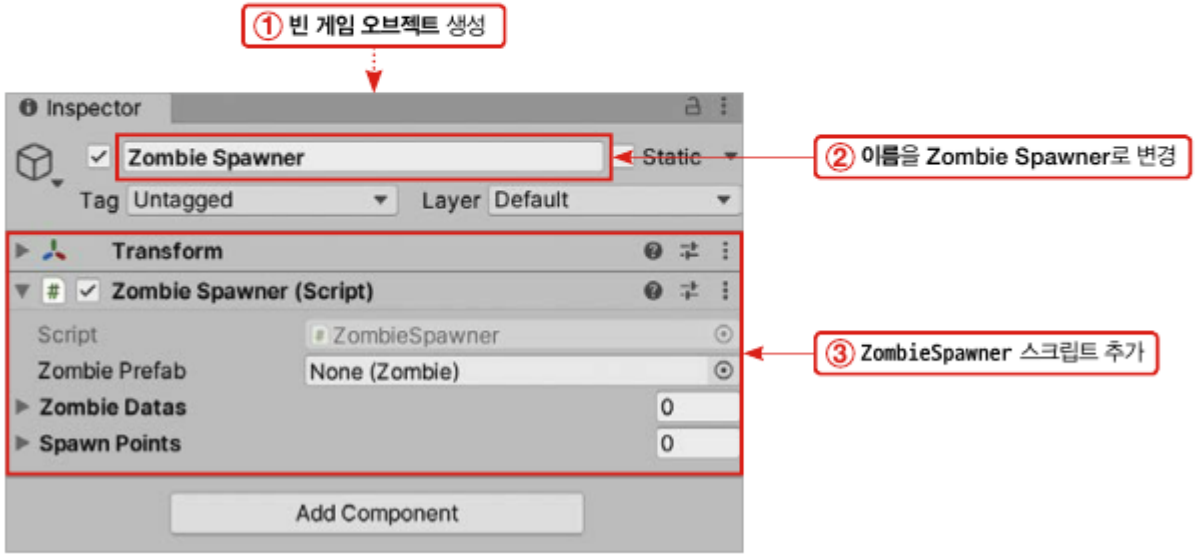
▶ 생성 위치 추가

[과정 02] 좀비 생성기 추가

① **빈 게임 오브젝트** 생성(+ 〉 Create Empty)

② 생성된 빈 게임 오브젝트의 **이름**을 Zombie Spawner로 변경

③ Zombie Spawner 게임 오브젝트에 `ZombieSpawner` 스크립트 추가(Scripts 폴더의 `ZombieSpawner` 스크립트를 Zombie Spawner 게임 오브젝트로 **드래그&드롭**)



▶ 좀비 생성기 추가

좀비 생성

[과정 01] SpawnWave() 메서드 완성

① SpawnWave() 메서드를 다음과 같이 완성

```
private void SpawnWave() {
    // 웨이브 1 증가
    wave++;

    // 현재 웨이브 * 1.5를 반올림한 수만큼 좀비 생성
    int spawnCount = Mathf.RoundToInt(wave * 1.5f);

    // spawnCount만큼 좀비 생성
    for (int i = 0; i < spawnCount; i++)
    {
        // 좀비 생성 처리 실행
        CreateZombie();
    }
}
```

좀비 생성

[과정 01] CreateZombie ( ) 메서드 완성

① **CreateZombie ( )** 메서드를 다음과 같이 완성

```
private void CreateZombie() {
    // 사용할 좀비 데이터 랜덤으로 결정
    ZombieData zombieData = zombieDatas[Random.Range(0, zombieDatas.Length)];

    // 생성할 위치를 랜덤으로 결정
    Transform spawnPoint = spawnPoints[Random.Range(0, spawnPoints.Length)];

    // 좀비 프리팹으로부터 좀비 생성
    Zombie zombie = Instantiate(zombiePrefab, spawnPoint.position, spawnPoint.rotation);

    // 생성한 좀비의 능력치 설정
    zombie.Setup(zombieData);

    // 생성된 좀비를 리스트에 추가
    zombies.Add(zombie);

    // 좀비의 onDeath 이벤트에 익명 메서드 등록
    // 사망한 좀비를 리스트에서 제거
    zombie.onDeath += () => zombies.Remove(zombie);
    // 사망한 좀비를 10초 뒤에 파괴
    zombie.onDeath += () => Destroy(zombie.gameObject, 10f);
    // 좀비 사망 시 점수 상승
    zombie.onDeath += () => GameManager.instance.AddScore(100);
}
```

좀비 생성

[과정 01] ZombieSpawner 컴포넌트 설정

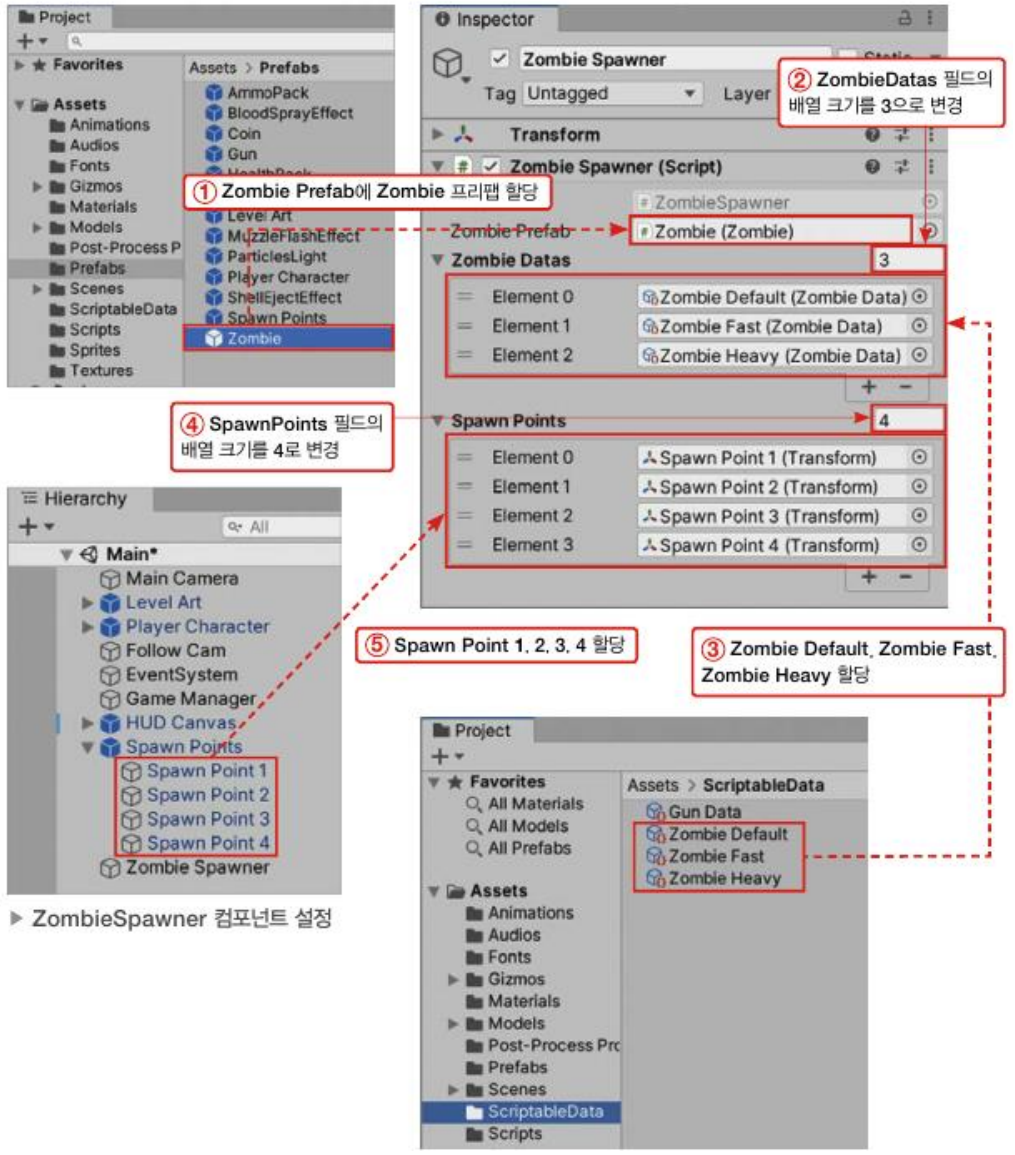
① ZombieSpawner 컴포넌트의 Zombie Prefab에 Prefabs 폴더의 Zombie 프리팹 할당

② ZombieDatas 필드의 배열 크기를 3으로 변경

③ ZombieDatas 필드의 각 원소에 ScriptableData 폴더에 있던 Zombie Default, Zombie Fast, Zombie Heavy 할당

④ SpawnPoints 필드의 배열 크기를 4로 변경

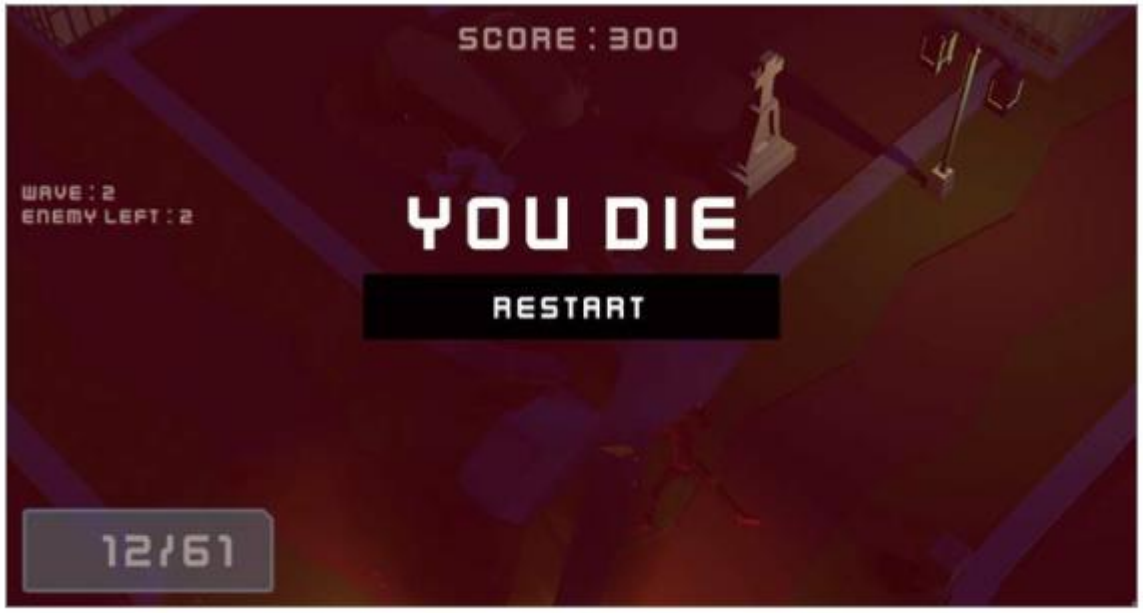
⑤ SpawnPoints 필드의 각 원소에 Spawn Point 1, 2, 3, 4 할당



▶ ZombieSpawner 컴포넌트 설정

좀비 생성

▶ 생성된 좀비들



▶ 게임오버 상태

아이템 생성

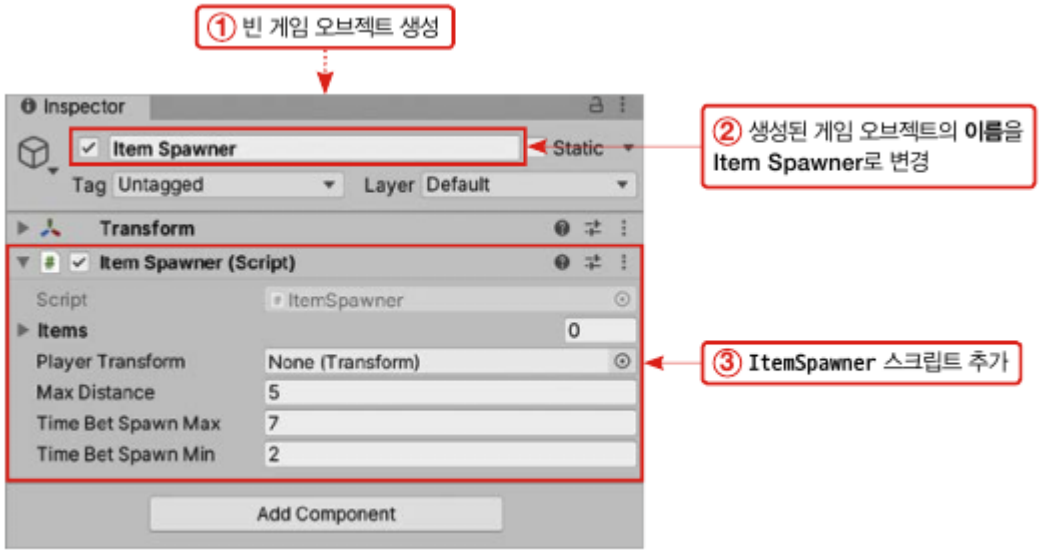
아이템 생성

[과정 01] 아이템 생성기 추가

① 빈 게임 오브젝트 생성(**하이어라키** 창에서 + > **Create Empty**)

② 생성된 게임 오브젝트의 **이름**을 **Item Spawner**로 변경

③ **Item Spawner** 게임 오브젝트에 **Script** 폴더의 `ItemSpawner` 스크립트 추가



▶ 아이템 생성기 추가

[과정 01] ItemSpawner 컴포넌트 설정

① `ItemSpawner` 컴포넌트의 **Items** 배열 크기를 3으로 변경

② **Prefabs** 폴더의 AmmoPack, Coin, HealthPack 프리팹을 **Items**의 각 원소에 할당

③ **Player Character** 게임 오브젝트를 **PlayerTransform** 필드로 **드래그&드롭**하여 할당

▶ ItemSpawner 컴포넌트 설정

포스트 프로세싱

포스트 프로세싱

포스트 프로세싱 적용 전

포스트 프로세싱 적용 후

▶ 포스트 프로세스 적용 전후 비교

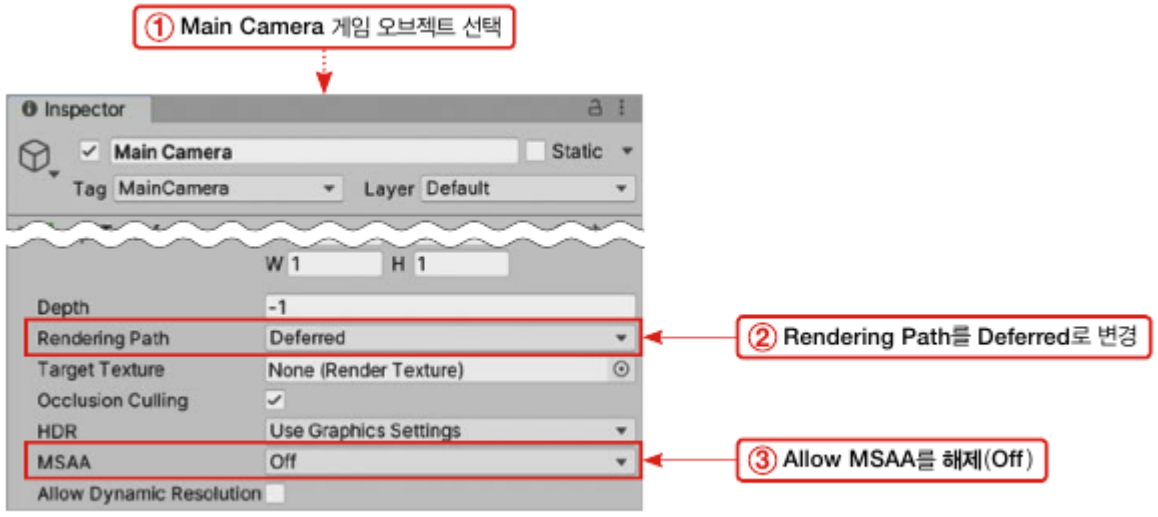
포스트 프로세싱

[과정 01] 카메라 렌더 설정 변경

① 하이어라키 창에서 Main Camera 게임 오브젝트 선택

② Camera 컴포넌트의 Rendering Path를 Deferred로 변경

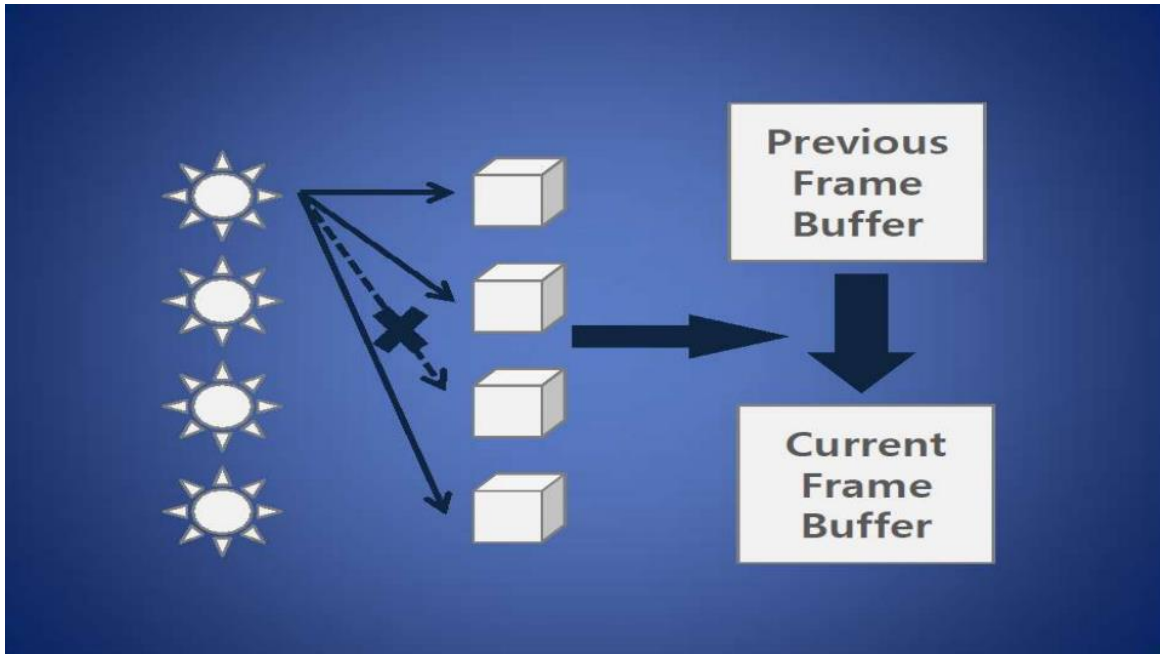
③ Allow MSAA를 해제(Off)

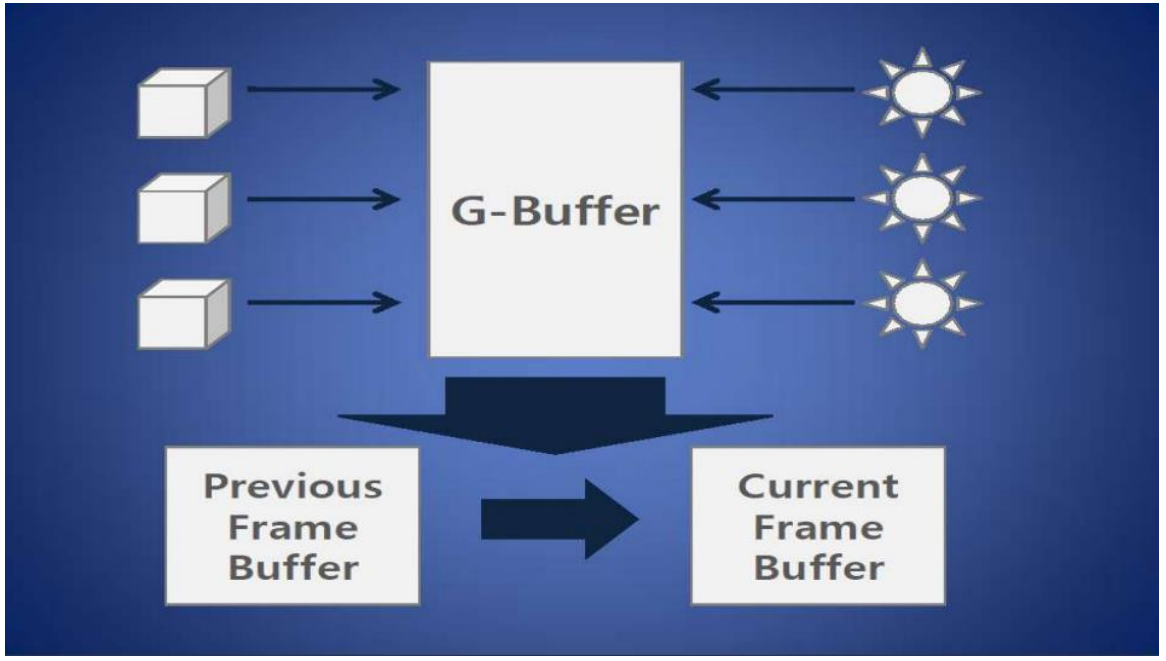


▶ 카메라 렌더 설정 변경

포스트 프로세싱

- 포워드 랜더링 vs 디퍼드 랜더링





포스트 프로세싱

- 포워드 랜더링
  - 각각의 오브젝트를 그릴 때마다 해당 오브젝트에 영향을 주는 모든 라이팅도 함께 계산하는 방식
  - 메모리 사용량이 적고 저사양에서도 비교적 잘 동작
  - 연산 속도가 느리며, 오브젝트 광원이 움직이거나 수가 많아질수록 연산량 급증
  - 유니티에서는 하나의 게임오브젝트에 대해 최대 4개의 광원만 개별 연산 (나머지는 광원과 라이팅 효과는 한 번에 연산)

포스트 프로세싱

- 디퍼드 랜더링(디퍼드 세이딩)
  - 라이팅 연산을 미뤄서(deffered) 실행
  - 첫 번째 랜더링 패스에서는 오브젝트 메시를 그리되 라이팅을 계산하거나 색을 채우지 않음
    대신 오브젝트의 여러 정보를 종류별로 버퍼에 저장
  - 두 번째 랜더링 패스에서 첫 번째 패스의 정보를 활용해 라이팅을 계산하고 최종 컬러를 결정
  - 유니티에서는 개수 제한 없이 광원을 표현할 수 있음(단 MSAA 같은 일부 안티앨리어싱 설정을 제대로 지원하지 않음)

포스트 프로세싱

- Anti-aliasing

포스트 프로세싱

- **Anti-aliasing**

  안티 에일리어싱 방법은 크게 2가지로 나뉘는데,

  첫번째는 샘플 레이트를 높이는 것으로 MSAA(Multi-Sampling Anti-Aliasing)이나 SSAA(Super-Sampling Anti-Aliasing)이 대표적이다.

  두 번째로는 가장자리를 희미하게 만들어 거슬리지 않게 후처리하는 방식으로 엔비디아의 FXAA(Fast Approximate Anti-Aliasing), MFAA(Multi-Frame Anti-Aliasing)와 AMD의 MLAA(Morphologocal Anti-Aliasing)등이 있다. 후처리 안티 에일리어싱은 블러 필터를 사용해 컴퓨터 부담을 줄이면서도 비슷한 결과물을 나타낸다

포스트 프로세싱

- Anti-aliasing - SSAA

화면 전체를 확대하여, 그림의 출력은 이 확대된 가상 화면에서 이루어지며, 원래 출력 대상에 출력을 할 때 각 샘플에 대해 보간된 값을 구해 적용 하며, 그렇기 때문에 연산량은 순수하게 샘플링 배수에 따라 제곱으로 늘어난다.
화면을 두배 확대해서 구현한다면 연산량은 4배로 증가한다. 가령 1920X1080 FHD 화면에 x4 SSAA를 적용한다면 3840X2160 4K 해상도에 해당하는 수준의 연산량을 가지게 된다.
연산량에 걸맞게 가장 확실한 효과를 보여주며, 높은 배수에서는 비등방성필터링과 같은 텍스쳐 필터링 효과가지 겸한다. 하지만 간혹 수직/수평선에서 확연하게 구별되어야 할 색이 섞이게 되어서 선명도가 저하되는 것이 눈에 띄는 경우가 존재한다

포스트 프로세싱

- **Anti-aliasing - MSAA**

  SSAA가 지나치게 큰 성능과 메모리를 요구해서 만들어진 개량방식으로 폴리곤 외곽선이 지나가는 곳만 샘플링 효과를 주는 방식이다.
  폴리곤 외곽선이 지나가는가를 테스트하는 샘플을 먼저 추출하고 외곽선이 지나갈 경우 이 샘플의 위치에서 컬러 샘플을 추출하여 이들을 섞는다.
  만족할 만한 화면 품질과 성능으로 오랫동안 안티 에얼리어싱 기술의 표준으로 자리잡았지만, 4K 해상도가 보급됨으로 인한 성능 제한 및 필요성의 감소, 디퍼드 랜더링 비중이 늘어나면서 호환성을 맞추기 위한 개발 소요 문제들이 생겨나 후처리 기술들에게 많은 자리를 내주고 있다.

포스트 프로세싱

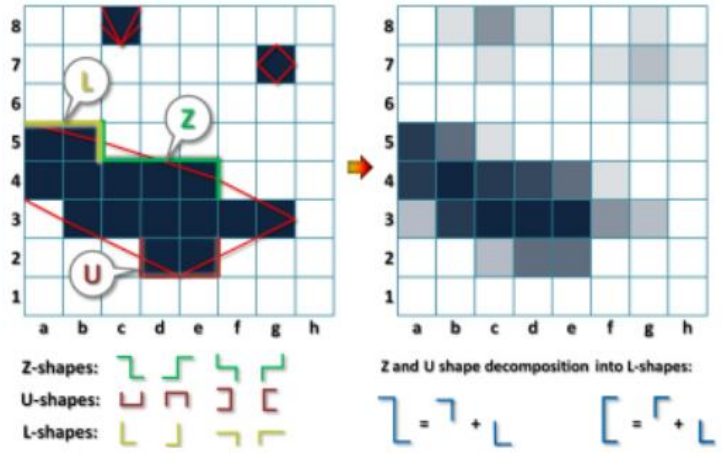
- **Anti-aliasing - FXAA(Fast Approximate Anti-Aliasing)**

FXAA는 전통적인 MSAA와 같은 종류의 고성능 근사값을 제공한다. 일종의 픽셀 셰이더로서 MLAA와 마찬가지로 목표 게임의 렌더링 파이프라인 이후에 이루어지는 후처리 단계이다. 하지만 AMD의 MLAA와 같이 다이렉트 컴퓨트를 사용하는 것이 아니라 단순한 후처리 셰이더를 사용하고, 어떤 GPU 계산 API에도 의존하지 않도록 고안되었다. 따라서 FXAA는 특별히 그래픽카드를 선호하지 않으며, 어떠한 그래픽엔진에도 매우 쉽고 빠르게 적용될 수 있다.
MSAA와 비교하여 FXAA의 목표는 더 빠르고 메모리 점유율이 더 낮으며, 픽셀이 흐릿해지는 현상이 일어나지 않는다는 장점이 있다.
FXAA는 화면의 밝기 값을 분석하여 이것이 급격히 바뀌는 곳을 기반으로 외곽선을 추출하고, 이 외곽선에 90도 방향으로 AA효과를 넣는 방식으로 되어있다. 따라서 이 외곽선의 길이가 충분히 확보되지 않을 경우 효과를 특정하기 힘들어지며 이는 복잡하게 변화하는 구조, 특히 문자 등에서 심각하게 문제가 된다. 전반적으로 고해상도 텍스쳐 등 세밀한 표현이 일어나는 곳에서 심한 블러링 효과가 일어난다.
현재 범용적으로 사용되는 AA기술중에서 가장 성능저하가 적으면서 준수한 품질을 보여주는 기술로 인정받고 있다. (다르게 말하면 실용적인 AA기술들 중에서 속도는 가장 빠르지만 품질은 가장 낮다고도 할 수 있다.)

포스트 프로세싱

- **Anti-aliasing - MLAA(Morphological Anti-Aliasing)**

이미 완성된 이미지에 후처리 형식으로 외곽선의 형태를 파악하여 원래의 형태를 예측, 거기에 적절한 중간색을 넣어 외곽선에 일어나는 계단을 완화시키는 방식이다.
MLAA는 별도의 프레임버퍼 용량은 잡아먹지 않지만 형태 분석을 위한 연산량이 엄청나게 된다. 대신 이 연산은 셀-브로드밴드 엔진 특유의 SPE를 사용할 수 있어, PS3의 VRAM과 범용 코어[19]가 모자라고 대신 SPE가 남아도는 환경에 적합한 안티 에얼리어싱인 것이다.
AMD에서는 이 연산을 CPU대신 자사의 GPU로 가속하는 방식으로 드라이버에 추가하여 AMD의 대표적인 안티 에얼리어싱 기법으로서 알려졌다

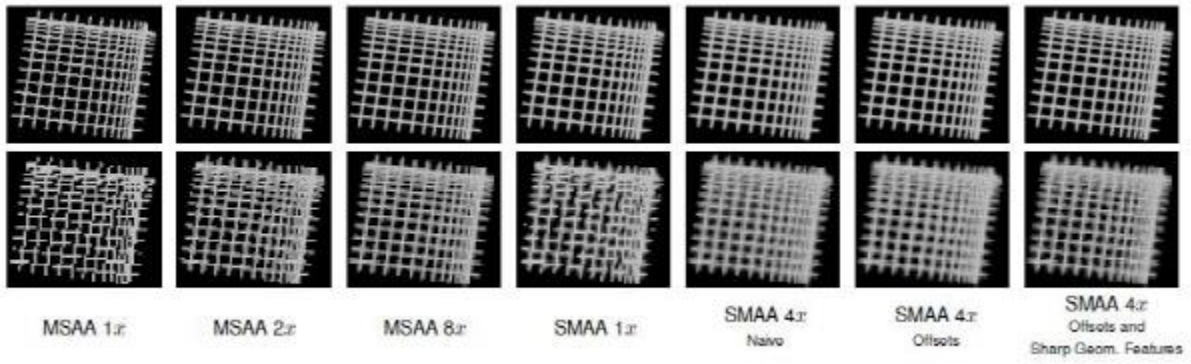


포스트 프로세싱

- **Anti-aliasing - SMAA(Subpixel Morphological AntiAliasing)**

유로그래픽포럼2011에서 크라이엔진3로 처음 SMAA를 시연해보였다.
기존의 MLAA을 진보시킨 것으로 기본적으로는 MLAA의 엄청난 연산량을 해결하기 위하여 형태 분석을 실시간으로 연산하지 않고 적당한 형태에 맞는 필터들을 미리 생성해놓고 형태에 맞게 덮어서 효과를 주는 방식으로 연산량문제를 크게 해결하였으며, 문자에서의 문제해결및 급격히 형태가 변하는 모서리에 대한 대응, 외곽선에서 일어나는 그라데이션을 위한 외곽선 판별알고리즘의 개선등 특정상황들에 맞춰 화질 향상을 위한 상황대응기능들을 추가하였다.

또 여기에 추가적으로 후처리 AA의 최대단점으로 지적되었던 서브픽셀문제를 해결하기 위하여 TAA, MSAA와 결합하는 방식을 추가하였다.



포스트 프로세싱

- **Anti-aliasing - TAA(Temporal Anti-Aliasing)**

NVIDIA의 지포스 600 시리즈에서 새로 적용될 수 있도록 추가된 AA기술.
영화와 같은 화면을 만드는 것을 목표로 하였다.
기본적으로 MSAA+TAA+블러링필터로 이루어진다.[17]
TXAA 2x = MSAA 2x + TAA 2x + 블러링필터
TXAA 4x = MSAA 4x + TAA 2x + 블러링필터
블러링 필터는 CFAA에 사용되는방법과 유사한것으로 추정되며 자신들의 그래픽카드에 전문적으로 적용되도록 만드는 과정에서 MSAA파트에 HDR등 후처리 문제를 해결할수 있도록 만들었다. 또한 TAA의 고스트 문제는 화면 물체의 움직임 벡터를 계산하여 이를 적용하여 적절한 위치에 단일상이 맺히도록 하여 해결하였다.
서브 픽셀에 강력한 대응이 가능하고, 액션 영화와 같이 움직임과정에 부드러움을 느낄수 있다고 주장하였으나 실제 결과물에 있어 FXAA를 능가하는 블러링 효과 때문에 많은 지탄을 받았다. 심한 블러링을 제거 하기 위해서는 블러링필터의 크기와 효과를 줄이면 되지만 이 경우 사실상 MSAA+TAA가 되어버려서 품질적으로 경쟁기술에 비하여 메리트가 없어진다.[18] AA효과를 위하여 블러링 필터를 키우면 블러링이 심해지고 심한 블러링을 줄이기 위해 블러링 필터를 줄이면 AA효과가 줄어드는. CFAA에서 AMD가 한 삽질을 그대로 재현한 기술이라 할수있다.

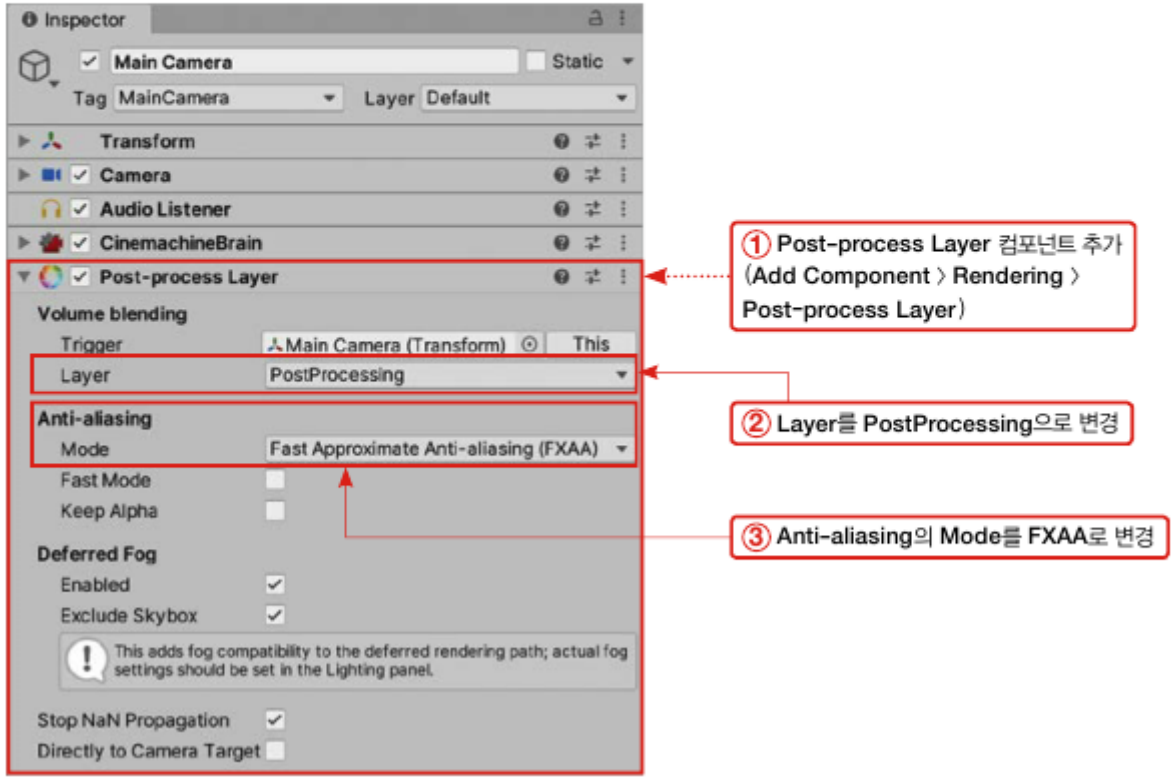
포스트 프로세싱

[과정 01] 카메라에 포스트 프로세스 레이어 추가

① Main Camera 게임 오브젝트에 Post-process Layer 컴포넌트 추가(Add Component 〉 Rendering 〉 Post-process Layer)

② Post-process Layer 컴포넌트의 Layer를 PostProcessing으로 변경

③ Anti-aliasing의 Mode를 FXAA로 변경



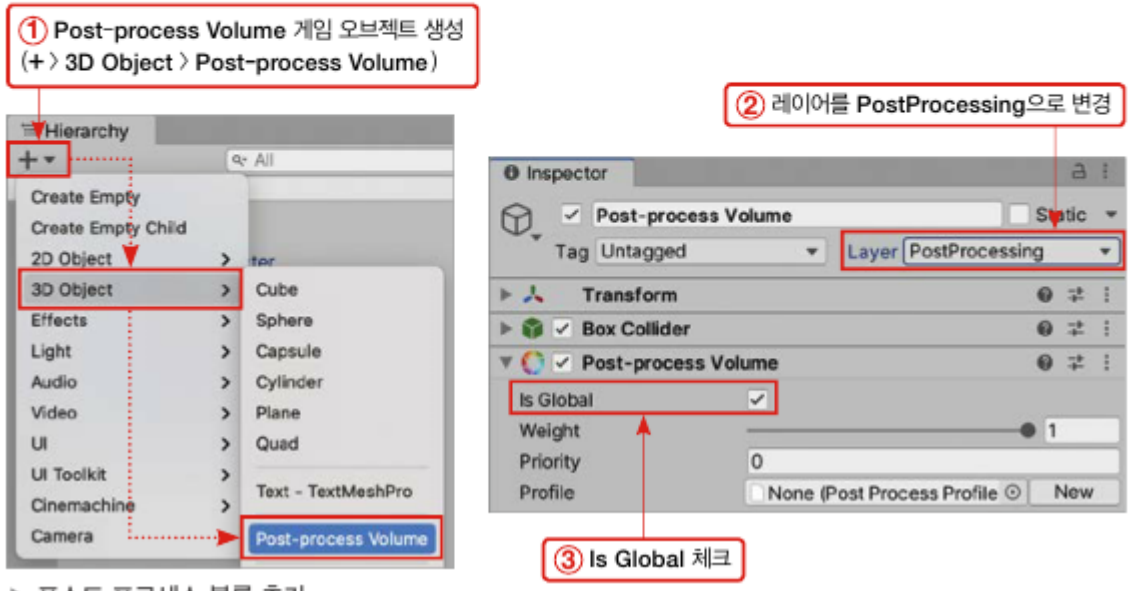
▶ 카메라에 포스트 프로세스 레이어 추가

[과정 02] 포스트 프로세스 볼륨 추가

① Post-process Volume 게임 오브젝트 생성(+ 〉 3D Object 〉 Post-process Volume)

② Post-process Volume 게임 오브젝트의 레이어를 PostProcessing으로 변경

③ Post-process Volume 컴포넌트의 Is Global 체크

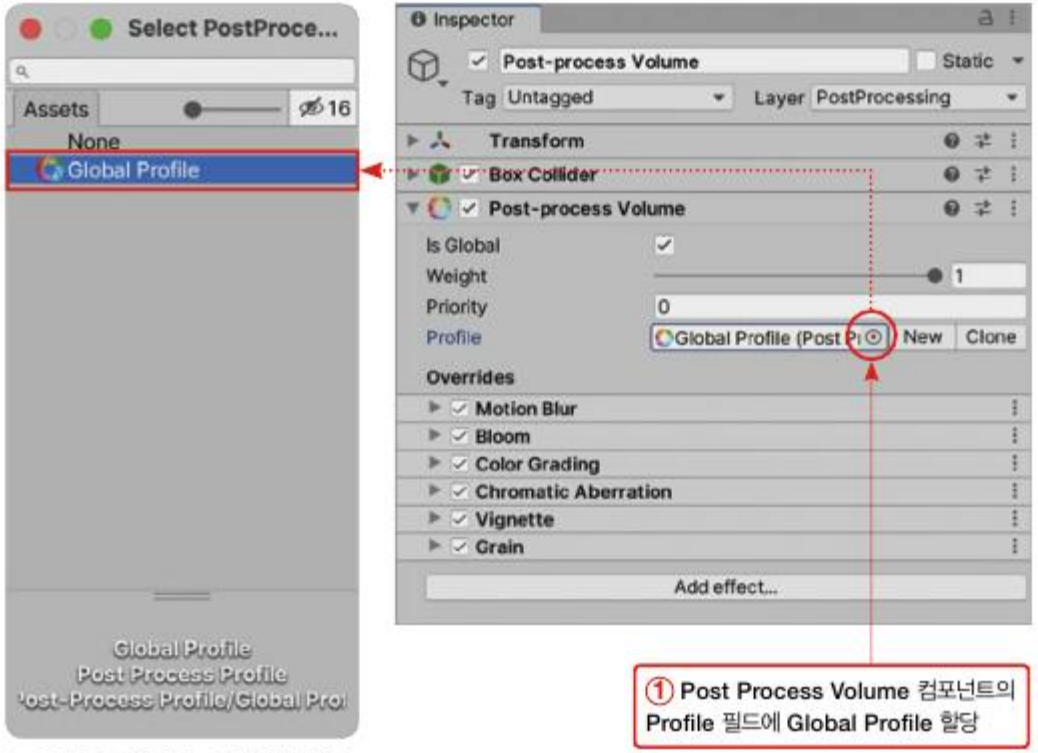


▶ 포스트 프로세스 볼륨 추가

포스트 프로세싱

[과정 03] 포스트 프로세스 프로파일 할당

① Post Process Volume 컴포넌트의 Profile 필드에 Global Profile 할당(Profile 필드 옆의 **선택** 버튼 클릭 > Global Profile 더블 클릭)



▶ 포스트 프로세스 프로파일 할당

▶ 적용된 모습

포스트 프로세싱

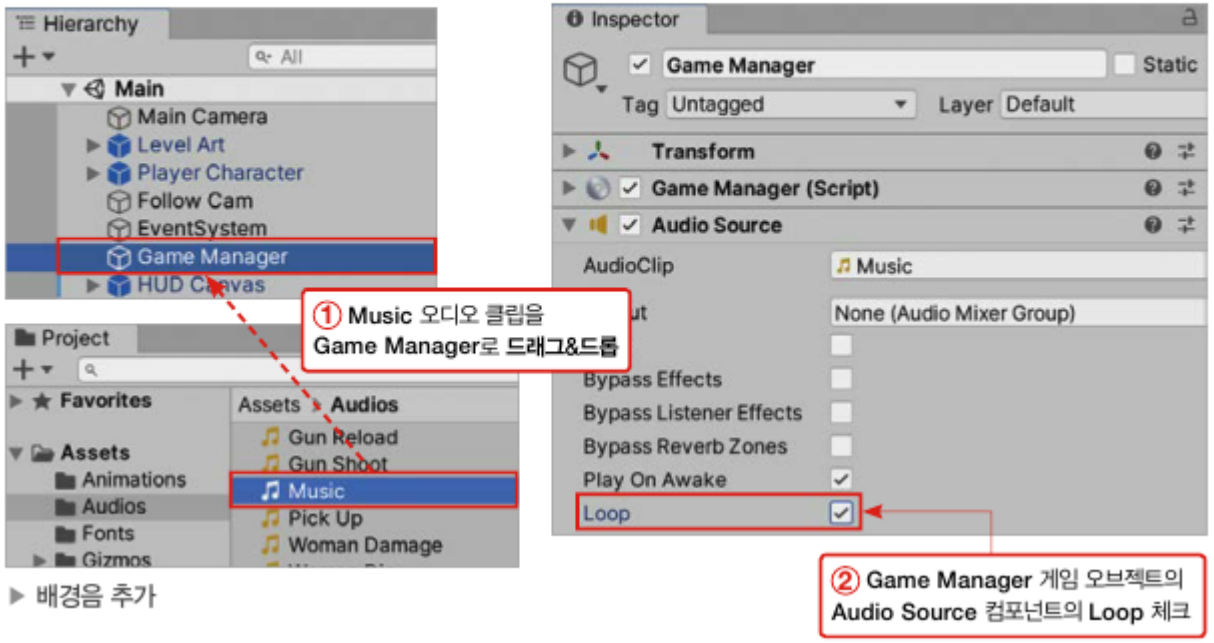
- **모션 블러(Motion Blur)**
  - 빠르게 움직이는 물체에 대한 잔상
- **블룸(Bloom)**
  - 일명 '뽀샤시'
  - 밝은 물체의 경계에서 빛이 산란되는 효과
- **컬러 그레이딩(Color Grading)**
  - 일명 '인스타그램 사진 필터'
  - 최종 컬러, 대비, 감마 등을 교정
- **색 수차(Chromatic Aberration)**
  - 일명 '방사능 중독 효과'
  - 이미지의 경계가 번지고 삼원색이 분리되는 효과
  - 게임에서 방사능이나 독 중독 효과를 표현할 때 주로 사용
- **비네트(Vignette)**
  - 화면 가장자리의 채도와 명도를 낮추는 효과
  - 화면 중심에 포커스를 주고 차분한 느낌을 줄 때 주로 사용
- **그레인(Grain)**
  - 화면에 입자 노이즈 추가
  - 필름 영화 같은 효과를 내거나 공포 분위기를 강화할 때 주로 사용

포스트 프로세싱

[과정 01] 배경음 추가

① Audio 폴더의 Music 오디오 클립을 **하이어라키** 창의 Game Manager로 **드래그&드롭**

② Game Manager 게임 오브젝트의 Audio Source 컴포넌트의 Loop **체크**



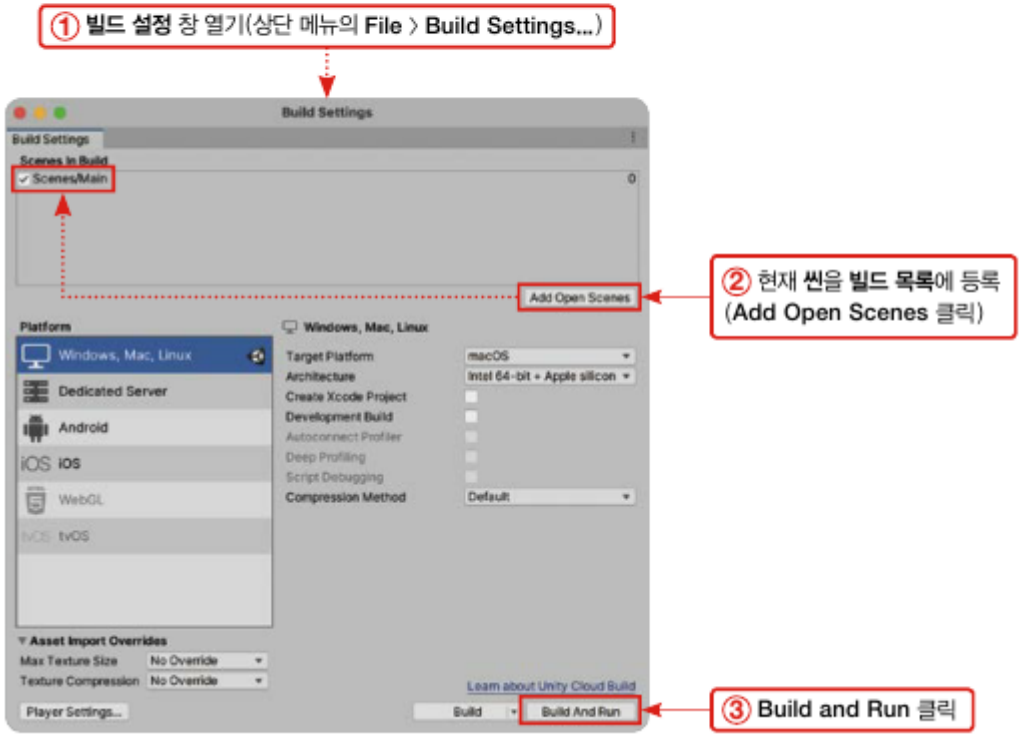
▶ 배경음 추가

[과정 02] 빌드하기

① **빌드 설정** 창 열기(상단 메뉴의 File 〉 Build Settings...)

② 현재 **씬**을 **빌드 목록**에 등록(Add Open Scenes 클릭)

③ Build and Run을 클릭하고 원하는 경로에 빌드



▶ 빌드하기

가상 조이스틱

가상 조이스틱

- UI 실습

Joystick Pack
(https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631)