# Unity Basic Skills

## Unity Graphic User Interface (UGUI)

Created in 2020-10-25

Last Updated 2020-11-09

Unity Version 2020.1.10f1

*Index*

- 프로젝트 기본 설정
- Unity GUI
- Canvas
- Visual Components

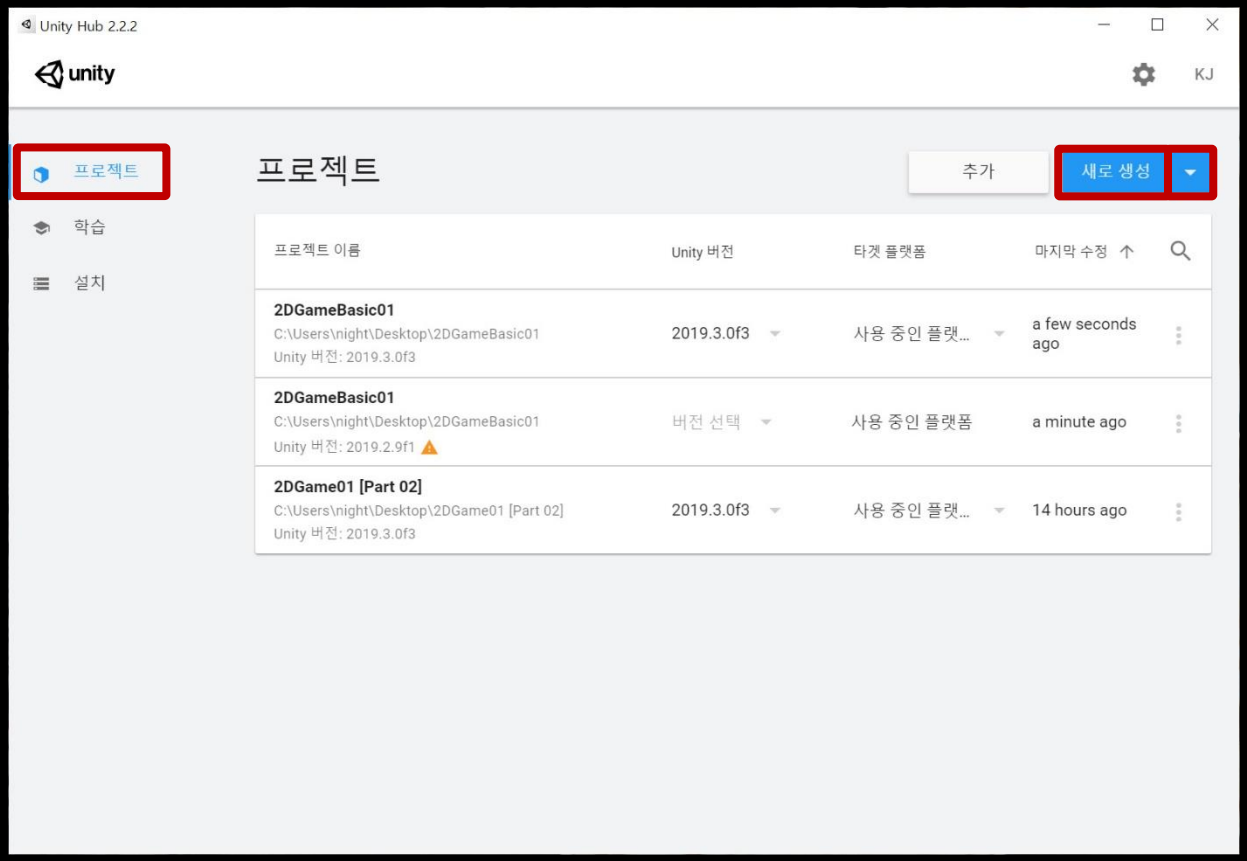
# 프로젝트 기본 설정

- 프로젝트 생성
- 게임화면 설정

# 프로젝트 기본 설정

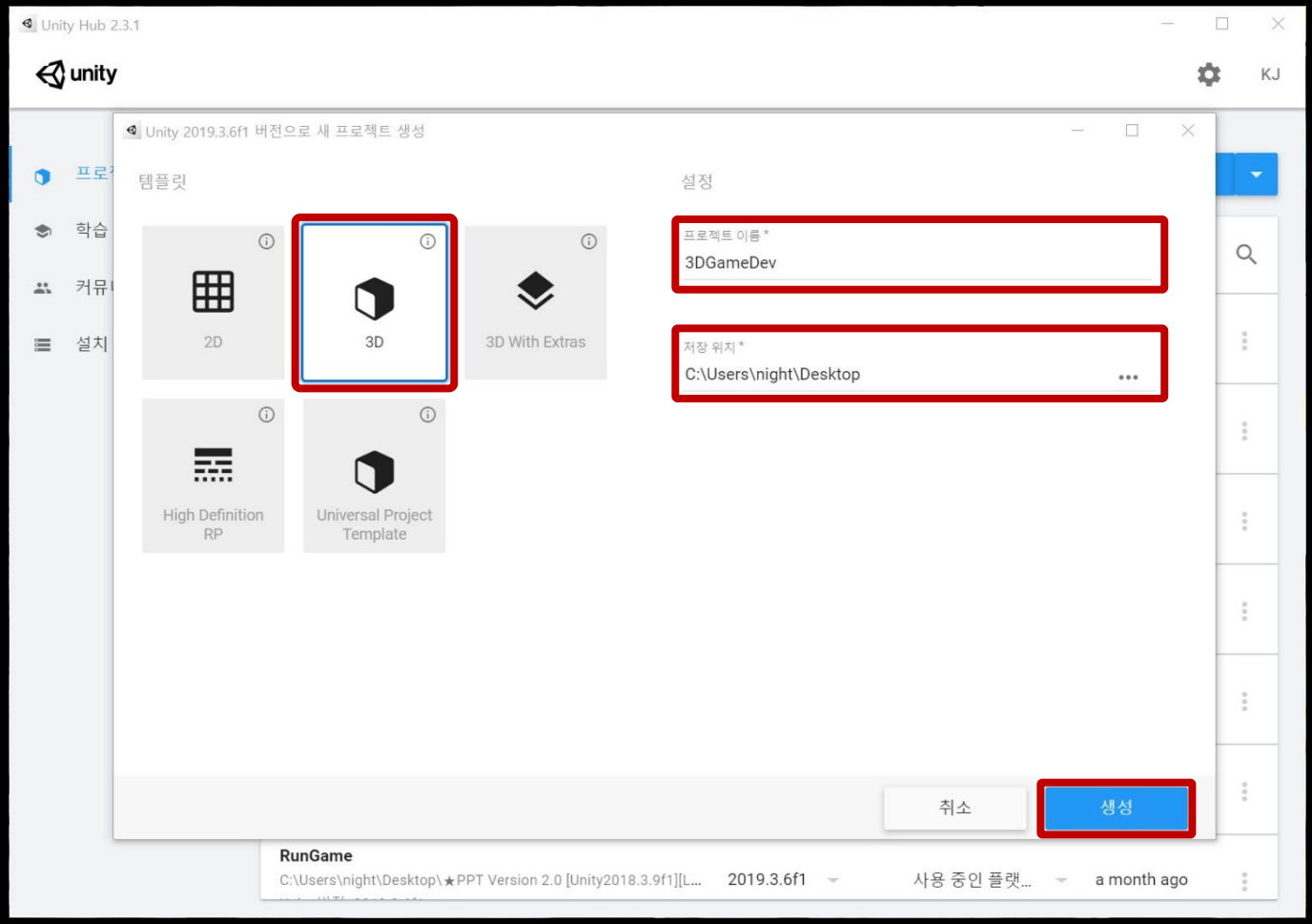
## 프로젝트 생성

- 프로젝트 생성
  - Unity Hub 실행 후 "새로 생성" 버튼을 눌러 새로운 프로젝트를 생성
  - 여러 개의 버전이 설치되어 있을 경우 "▼" 버튼을 눌러 버전 선택



# 프로젝트 기본 설정

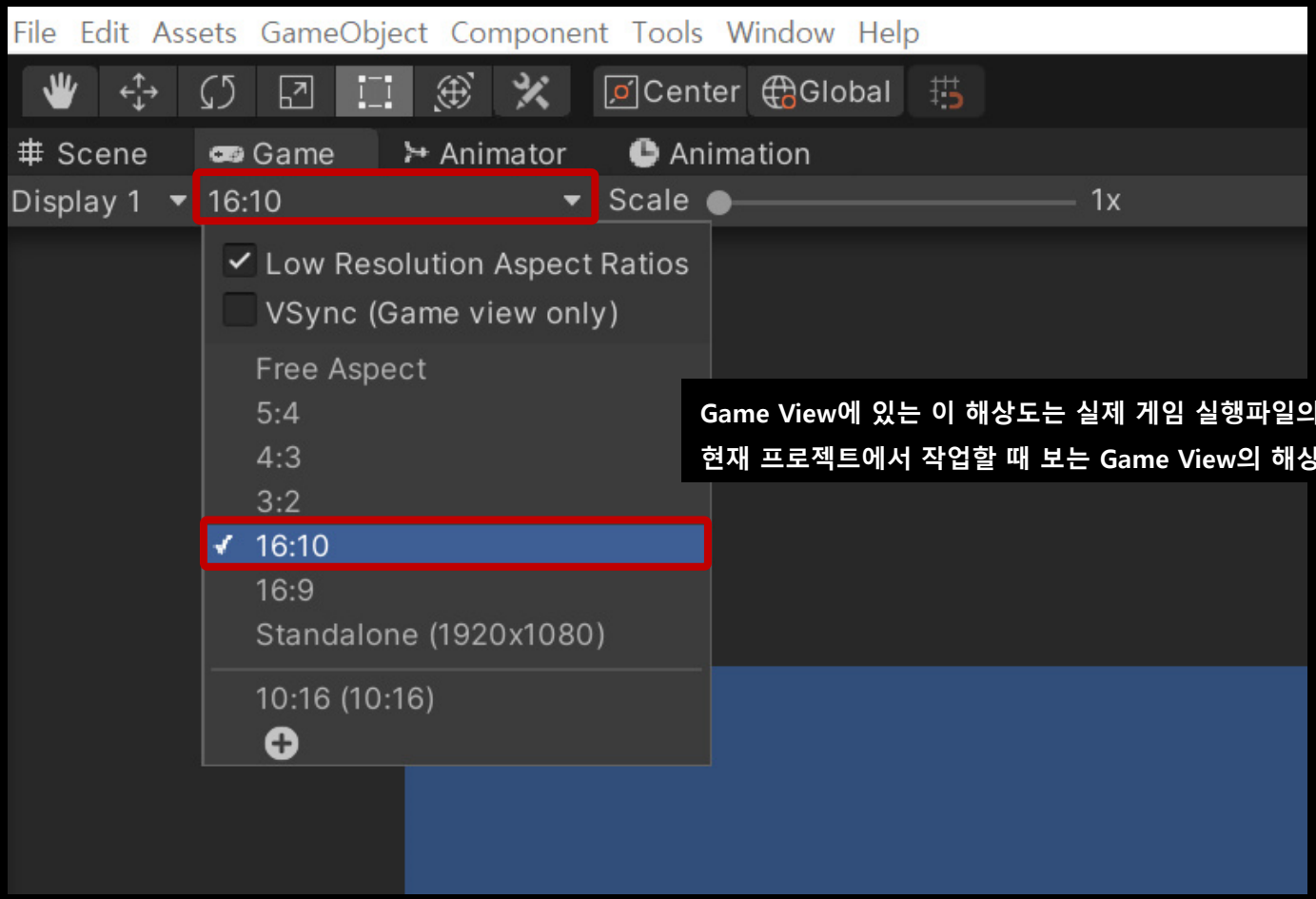
- 프로젝트 생성 (계속)
  - 원하는 템플릿(2D, 3D, Etc..)을 선택
  - 프로젝트 이름과 저장 위치를 설정한 후 "생성" 버튼을 눌러 프로젝트 생성



# 프로젝트 기본 설정

## 게임화면 설정

- Game View 바로 아래 Drop List 클릭 후 원하는 해상도 설정 (16:10)



# Unity GUI

- Unity UI
- Canvas GameObject
- EventSystem GameObject
- RectTransform Component

# Unity GUI

- **Unity UI**
  - OnGUI
    - UGUI가 나오기 전인 Unity 4.5 버전까지 사용했던 Unity Built-In UI
    - 시각적 개발을 지원하지 않음 (UI 배치, 스타일 설정 등 100% 코드 기반으로 작동)
    - 표면 레이어를 기반으로 하기 때문에 실행 효율이 매우 낮음
  - NGUI (Next-Gen User Interface)
    - UGUI가 나오기 이전에 많이 사용된 타사 유료 플러그인(2020. 10. 25 기준 $95)
    - 레퍼런스가 다양하며, 다양한 크기의 화면에 최적화 시키기 쉽다
    - 정적인 화면에 최적화가 잘 되어 있다
    - 동적 할당을 할 때 Sprite를 다시 그리기 때문에 비효율적이다

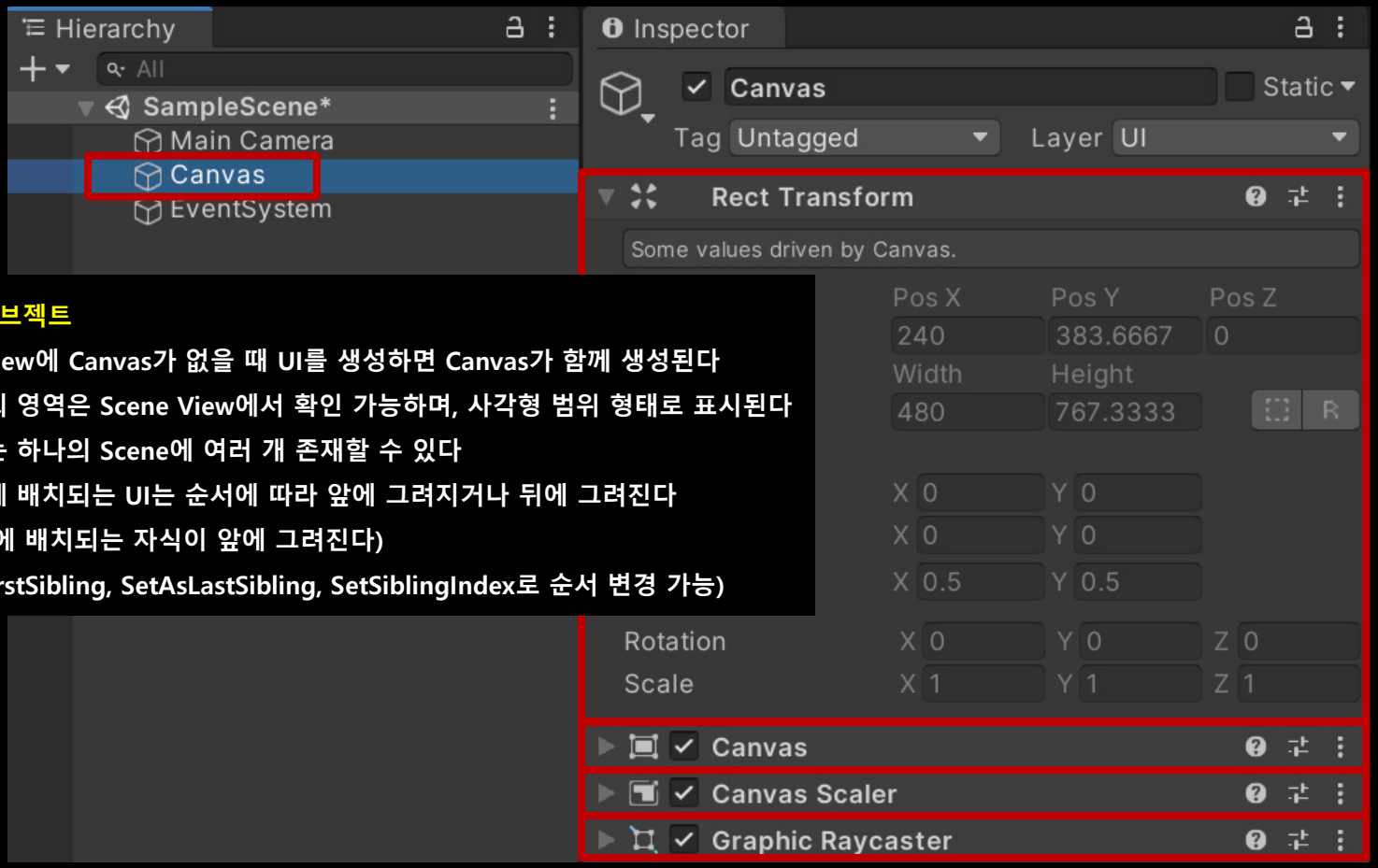
# Unity GUI

- UGUI (Unity Graphic User Interface)
  - Unity에서 제공하는 Unity Built-In UI (4.6 이후)
  - 추가 구매 없이 바로 사용 가능
  - 직관적인 UI 구성요소 간의 Depth 조절 가능
  - Canvas 단위로 Draw Call이 관리된다
  - Sprite Atlas 관리 (폴더 단위로도 가능)
  - Particle Rendering 문제가 존재한다
  - Tweening을 기본으로 지원하지 않는다
  - 동적 할당이 쉽고 편하며, 효율이 NGUI보다 좋다
  - UI 확장 에셋을 쉽게 구할 수 있으며, Unity에서도 지속적으로 관리해준다

# Unity GUI

## Canvas GameObject

- Canvas는 모든 UI 요소(게임 오브젝트, 컴포넌트)를 배치하는 영역
- RectTransform, Canvas, CanvasScaler, GraphicRaycaster 컴포넌트 보유



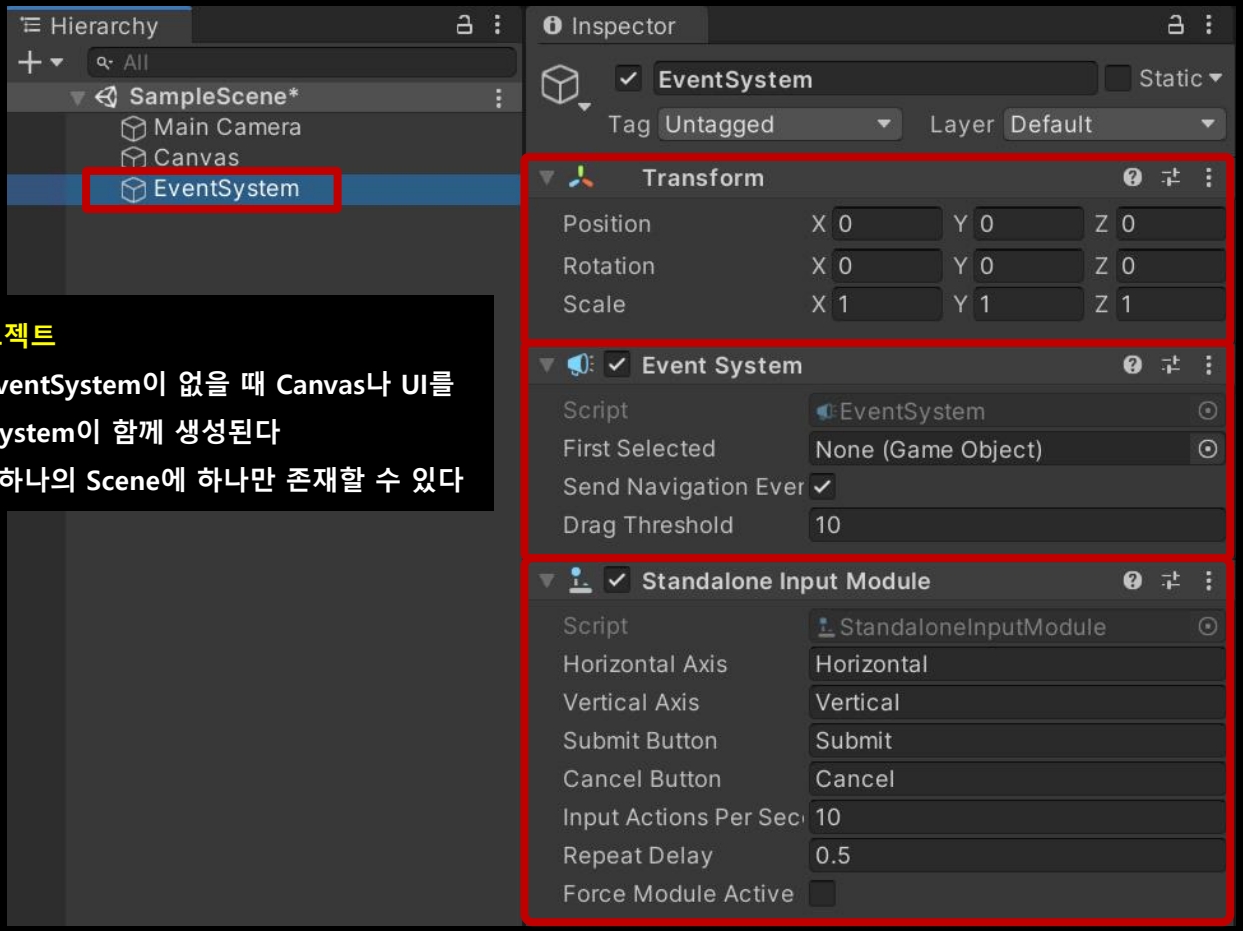

**Canvas 오브젝트**

- Scene View에 Canvas가 없을 때 UI를 생성하면 Canvas가 함께 생성된다
- Canvas의 영역은 Scene View에서 확인 가능하며, 사각형 범위 형태로 표시된다
- Canvas는 하나의 Scene에 여러 개 존재할 수 있다
- Canvas에 배치되는 UI는 순서에 따라 앞에 그려지거나 뒤에 그려진다
  (더 아래에 배치되는 자식이 앞에 그려진다)
  (SetAsFirstSibling, SetAsLastSibling, SetSiblingIndex로 순서 변경 가능)

# Unity GUI

## EventSystem GameObject

- EventSystem은 Graphic Raycaster를 이용해 충돌된 오브젝트의 이벤트를 검출하는 수단. 상호작용이 가능한 UI의 이벤트 처리를 담당



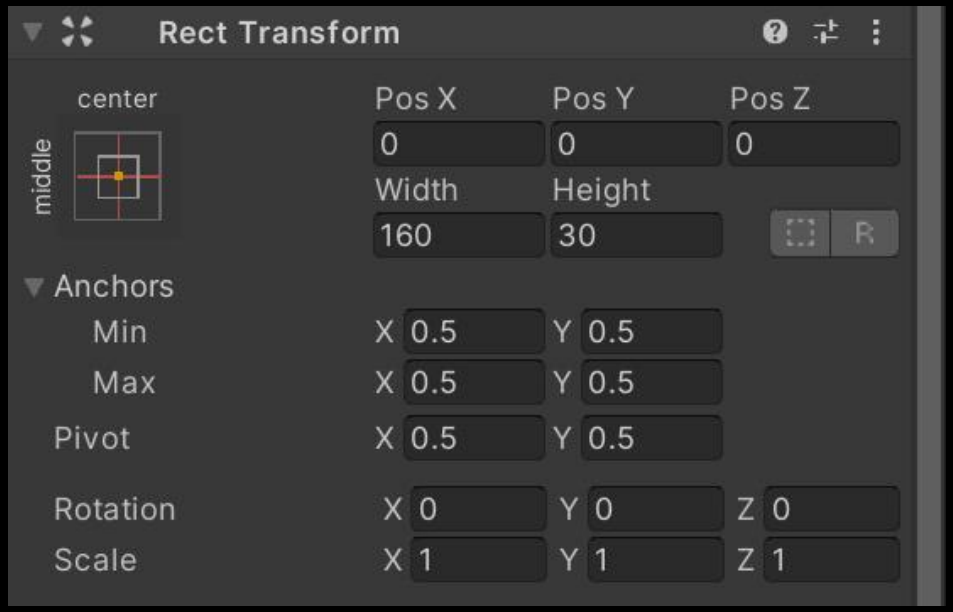

**EventSystem 오브젝트**

- Scene View에 EventSystem이 없을 때 Canvas나 UI를 생성하면 EventSystem이 함께 생성된다
- EventSystem은 하나의 Scene에 하나만 존재할 수 있다

# Unity GUI

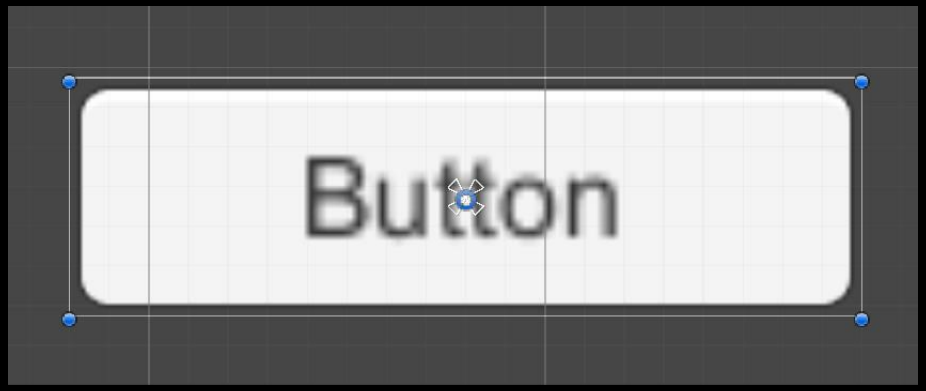
## RectTransform Component

- "RectTransform"은 "Transform"을 상속받아 만든 UI 전용 Transform
  - Transform 컴포넌트와 마찬가지로 위치(position), 회전(rotation), 크기(scale)를 가지고 있으며, 사각형의 치수를 결정하는 폭(width), 높이(height)를 가지고 있다
  - Pivot과 Anchors를 설정할 수 있어 UI 배치 및 관리를 더욱 편리하게 할 수 있다
  - RectTransform을 사용하는 경우 Resizing은 Width, Height를 통해서만 하고, Scale 변수는 건드리지 않는다

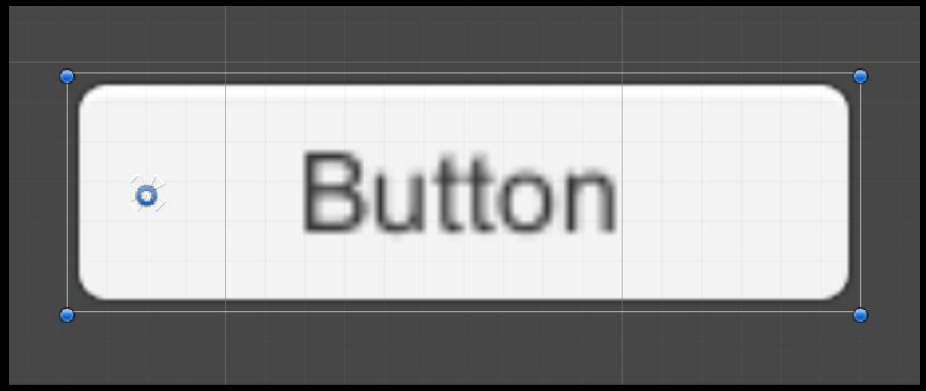


# Unity GUI

- 피벗 (Pivot)
  - RectTransform 컴포넌트를 가지고 있는 오브젝트 본인의 중심점
  - 중심점의 위치에 따라 위치를 설정했을 때 배치되는 지점,
    회전할 때 돌아가는 축, 크기 변화 등이 다르게 설정된다



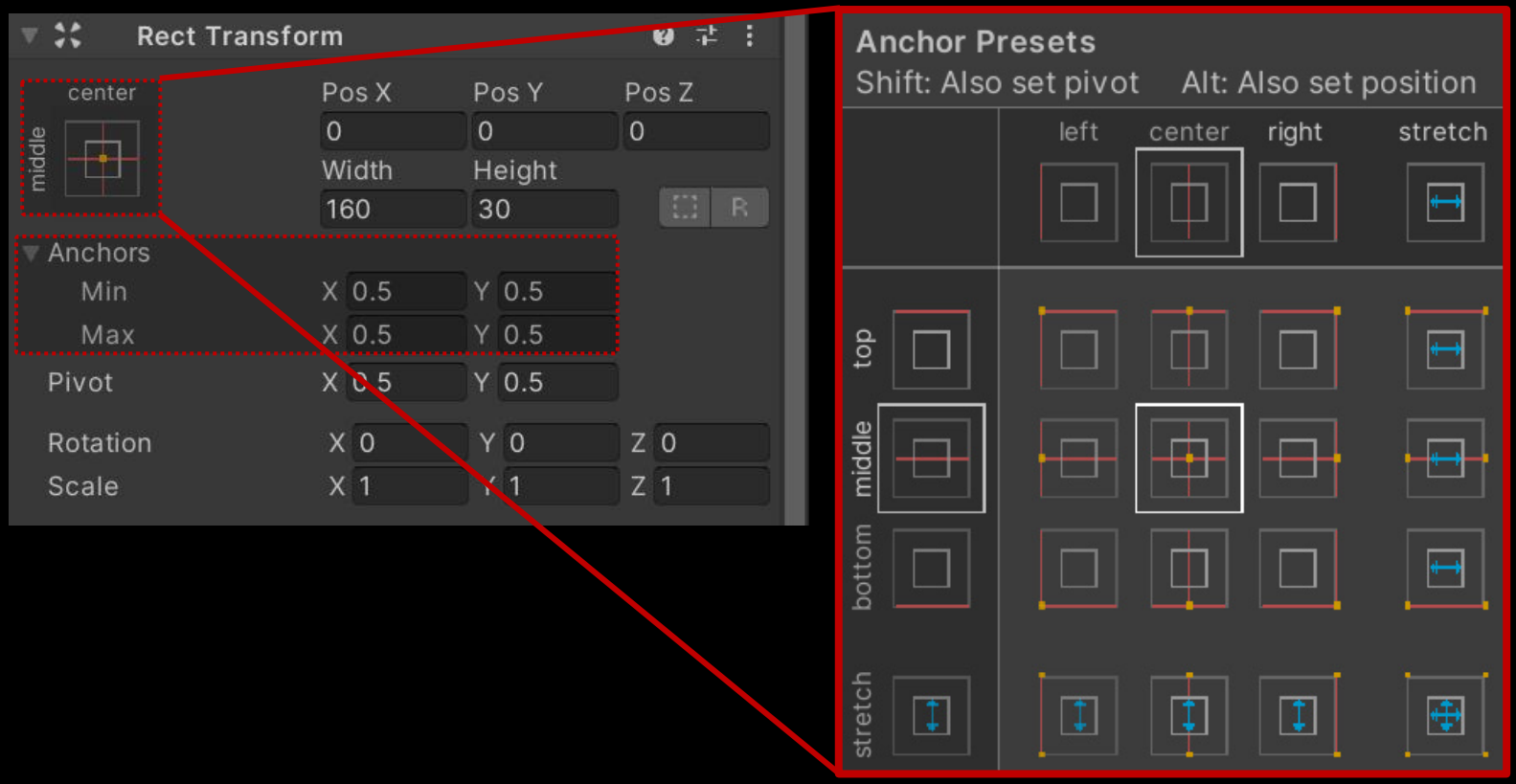

Pivot (0.5, 0.5)




Pivot (0.1, 0.5)

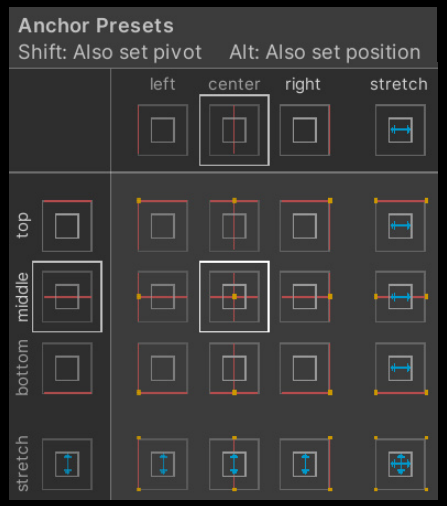
# Unity GUI

- 앵커 (Anchors)
  - 부모/자식 관계일 때(두 오브젝트 모두 RectTransform이 있어야 한다)
    자식 오브젝트가 부모의 특정 변이나 꼭지점을 기준으로 고정되게 하는 기능
  - 부모 오브젝트의 크기가 바뀔 때 고정된 축의 여백은 그대로 유지된다
  - Anchor에 제공되는 Anchor Presets을 이용하면 Anchor 설정이 굉장히 편리하다



# Unity GUI

- 단축키를 이용한 Anchor / Pivot / Position 설정




**단축키 : Mouse Click**

**< 기본 앵커 프리셋 (앵커 위치만 설정) >**


**단축키 : Alt + Click**

**< 앵커 위치와 객체 위치 동시 설정 >**


**단축키 : Shift + Click**

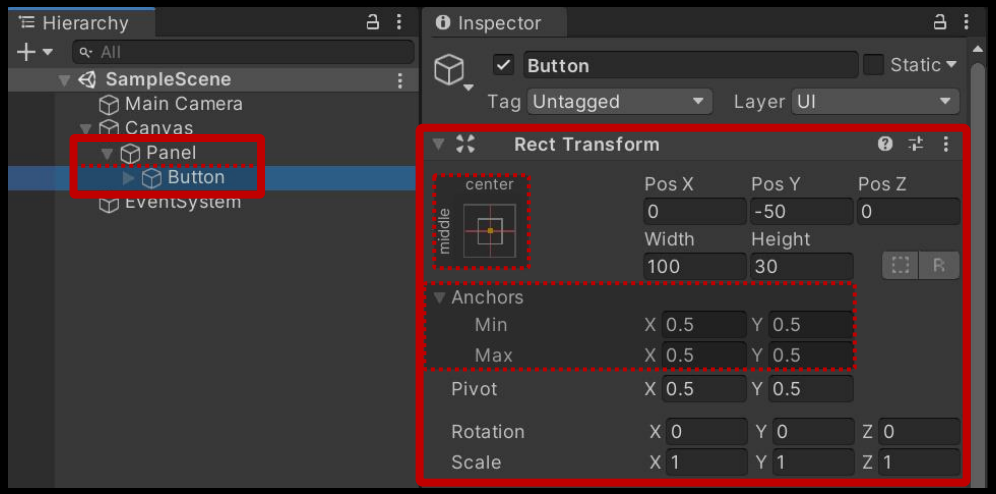
**< 앵커 위치와 객체 피봇 동시 설정) >**

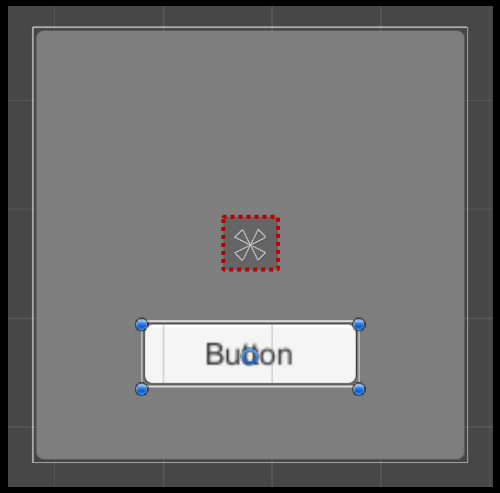

**단축키 : Alt + Shift + Click**

**< 앵커 위치와 객체 피봇, 객체 위치 동시 설정 >**

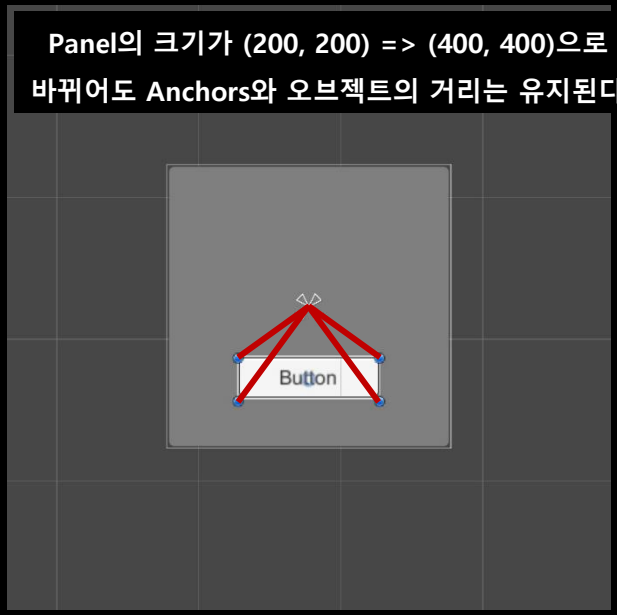
# Unity GUI

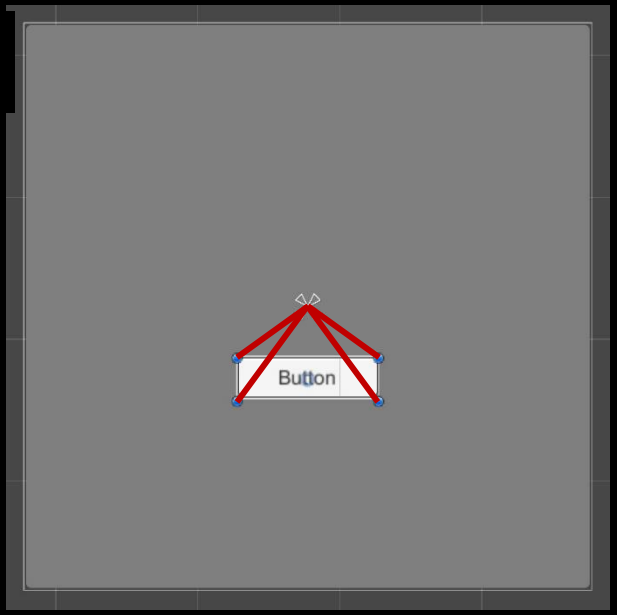
- Anchors 위치에 따른 오브젝트 제어 (중앙)



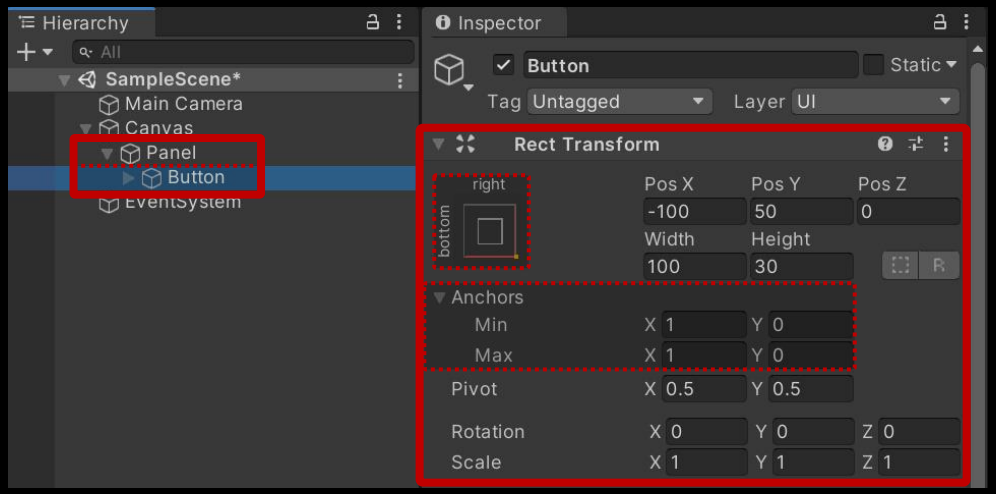

Panel의 크기가 (200, 200) => (400, 400)으로 바뀌어도 Anchors와 오브젝트의 거리는 유지된다.

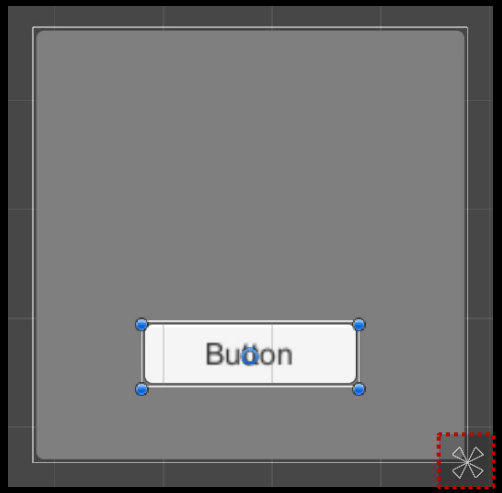


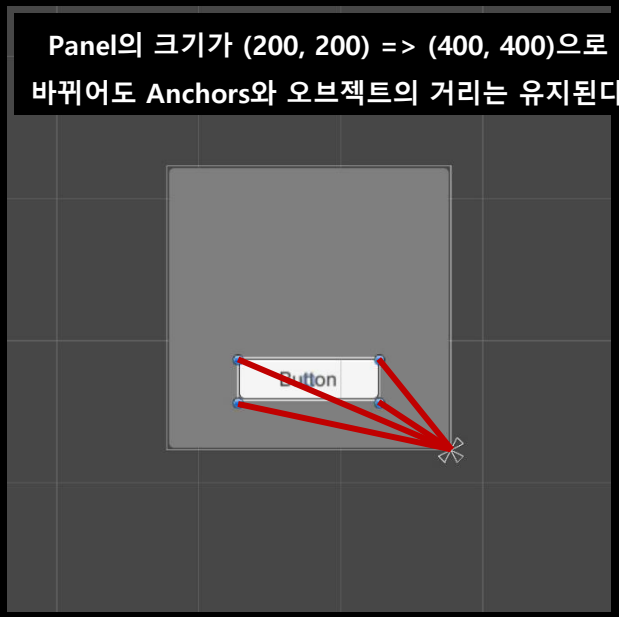
# Unity GUI

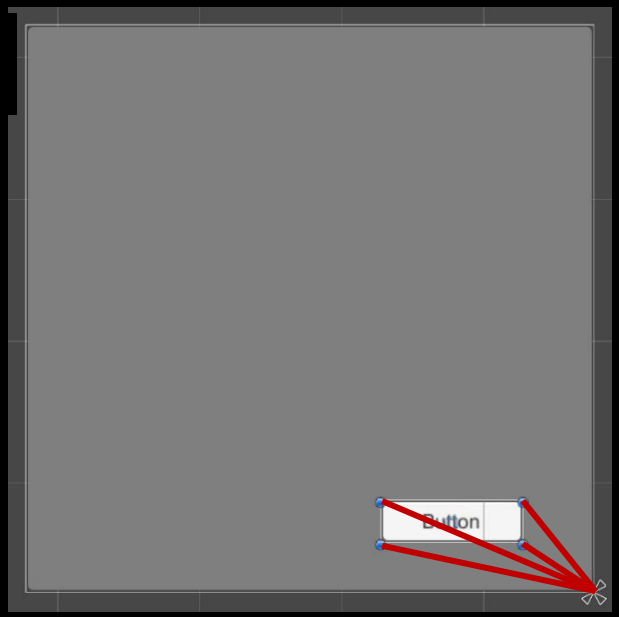
- Anchors 위치에 따른 오브젝트 제어 (오른쪽 하단)



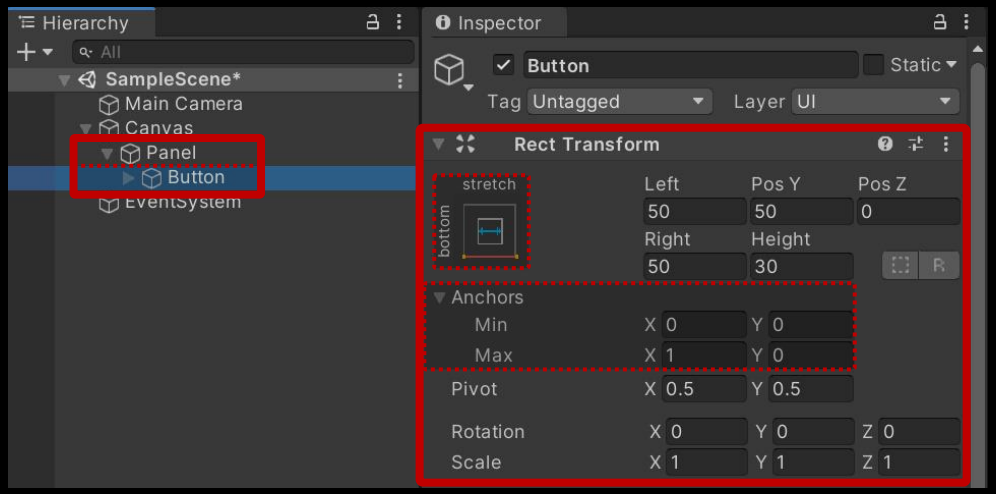

Panel의 크기가 (200, 200) => (400, 400)으로 바뀌어도 Anchors와 오브젝트의 거리는 유지된다.

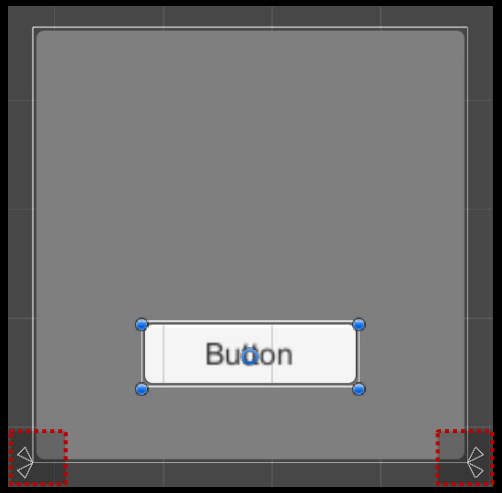


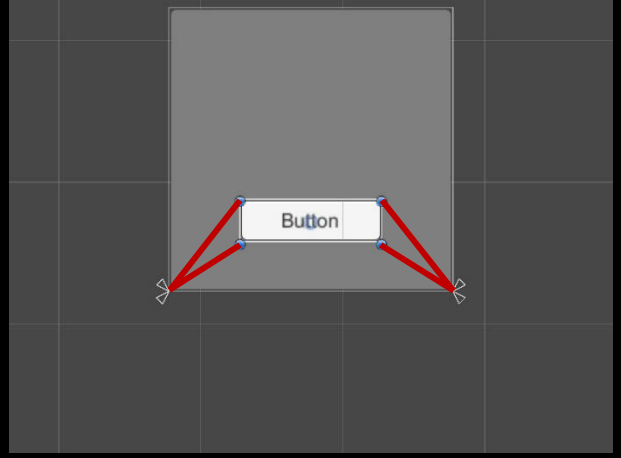
# Unity GUI

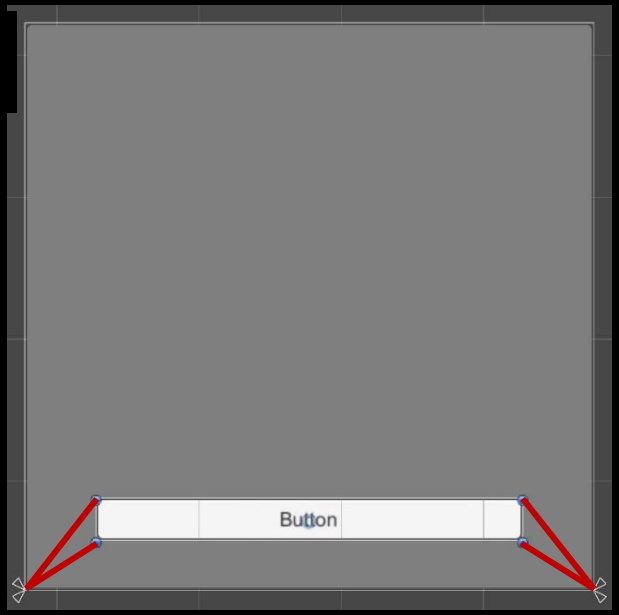
- Anchors 위치에 따른 오브젝트 제어 (가로는 최대, 하단)



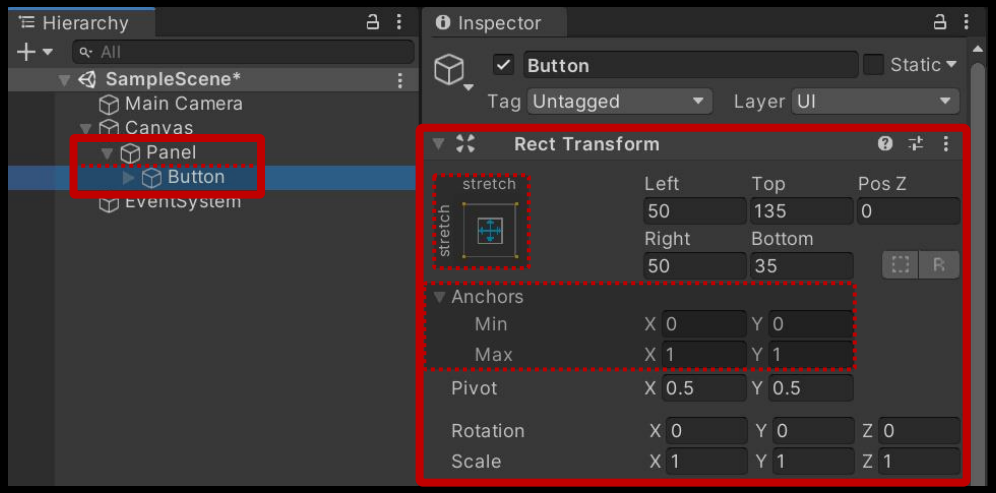

Panel의 크기가 (200, 200) => (400, 400)으로 바뀌어도 Anchors와 오브젝트의 거리는 유지된다.

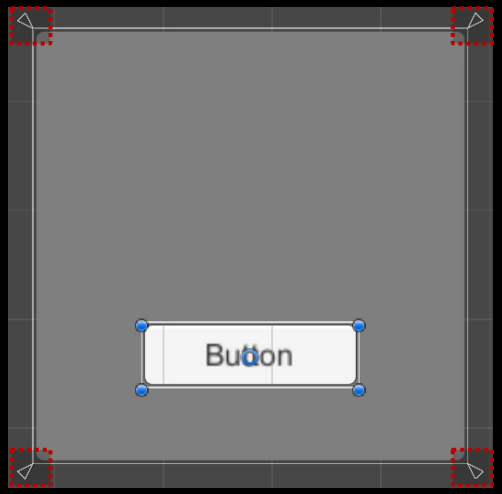


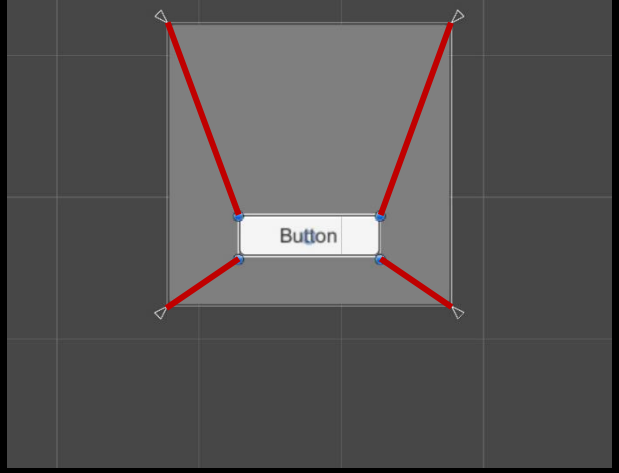
# Unity GUI

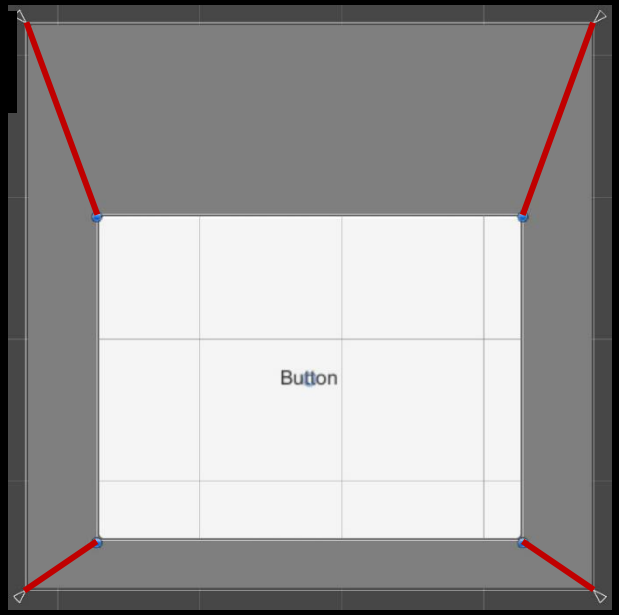
- Anchors 위치에 따른 오브젝트 제어 (가로, 세로 최대)




Panel의 크기가 (200, 200) => (400, 400)으로 바뀌어도 Anchors와 오브젝트의 거리는 유지된다.



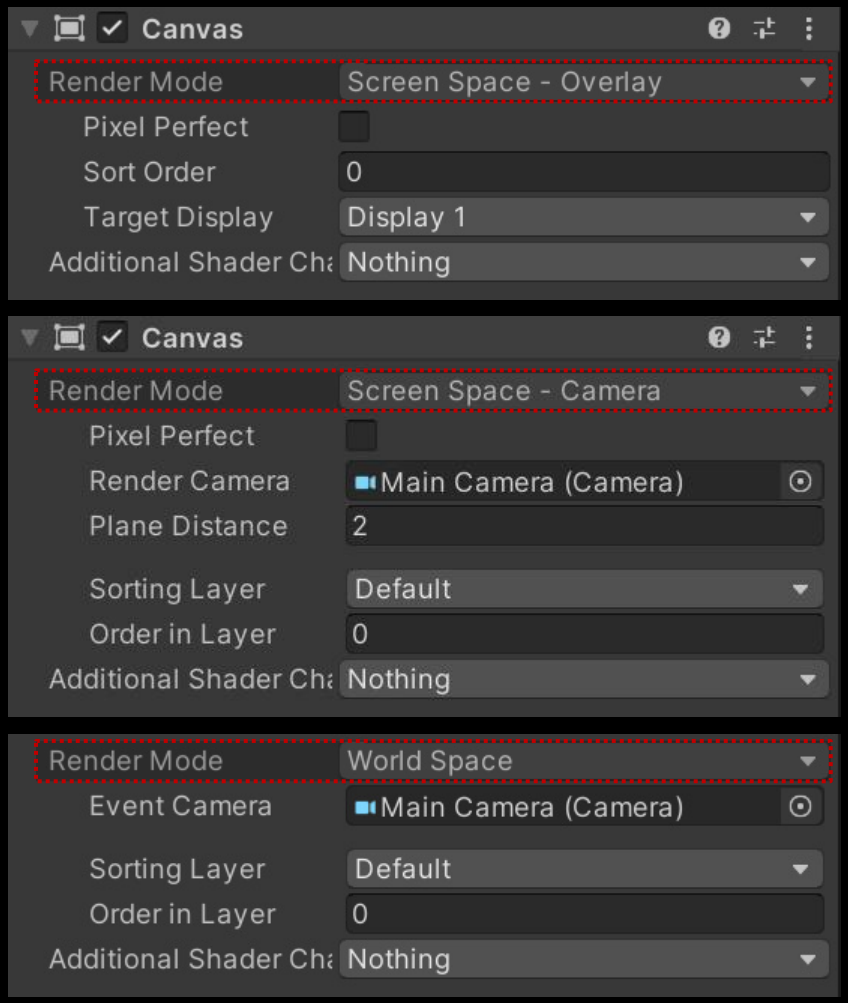
# Canvas

- Canvas
- Canvas Scaler
- Graphic Raycaster

# Canvas

## Canvas

- Canvas 컴포넌트는 UI가 배치되고, 화면에 렌더링되는 추상 공간을 나타냄

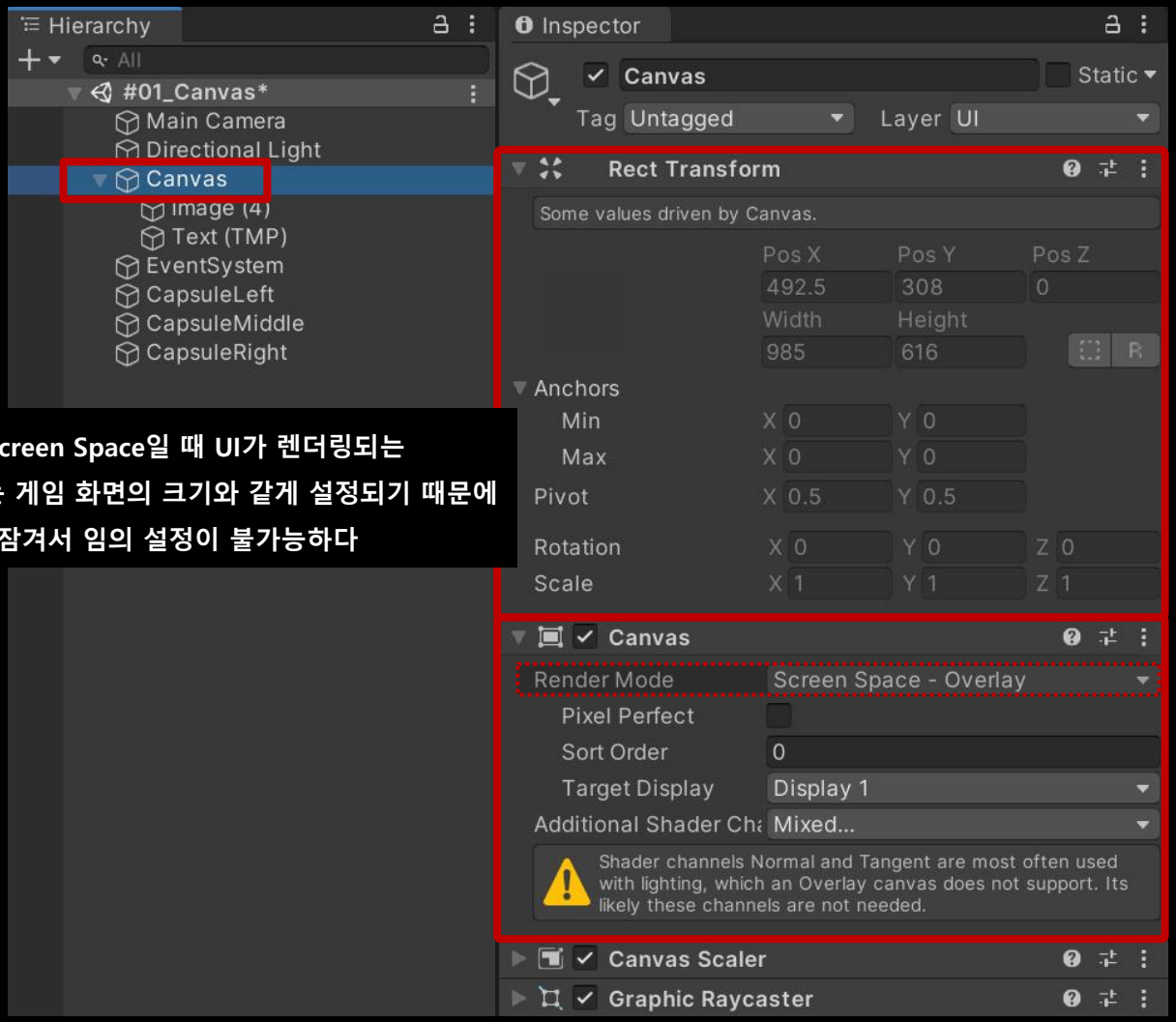


# Canvas

- Canvas Component Variables
  - Render Mode
    - UI가 화면에 렌더링 되는 방법 (Screen Space - Overlay / Camera, World Space)
  - Pixel Perfect [“Screen Space - Overlay / Camera” modes only]
    - Anti-Aliasing 없이 UI를 정밀하게 렌더링 할 때 사용
  - Render Camera [“Screen Space - Camera” mode only]
    - UI를 렌더링 하는데 사용되는 카메라
  - Plane Distance [“Screen Space - Camera” mode only]
    - 카메라(Render Camera)와 UI 사이의 거리
  - Event Camera [“World Space” mode only]
    - UI 이벤트를 처리하는데 사용되는 카메라

# Canvas

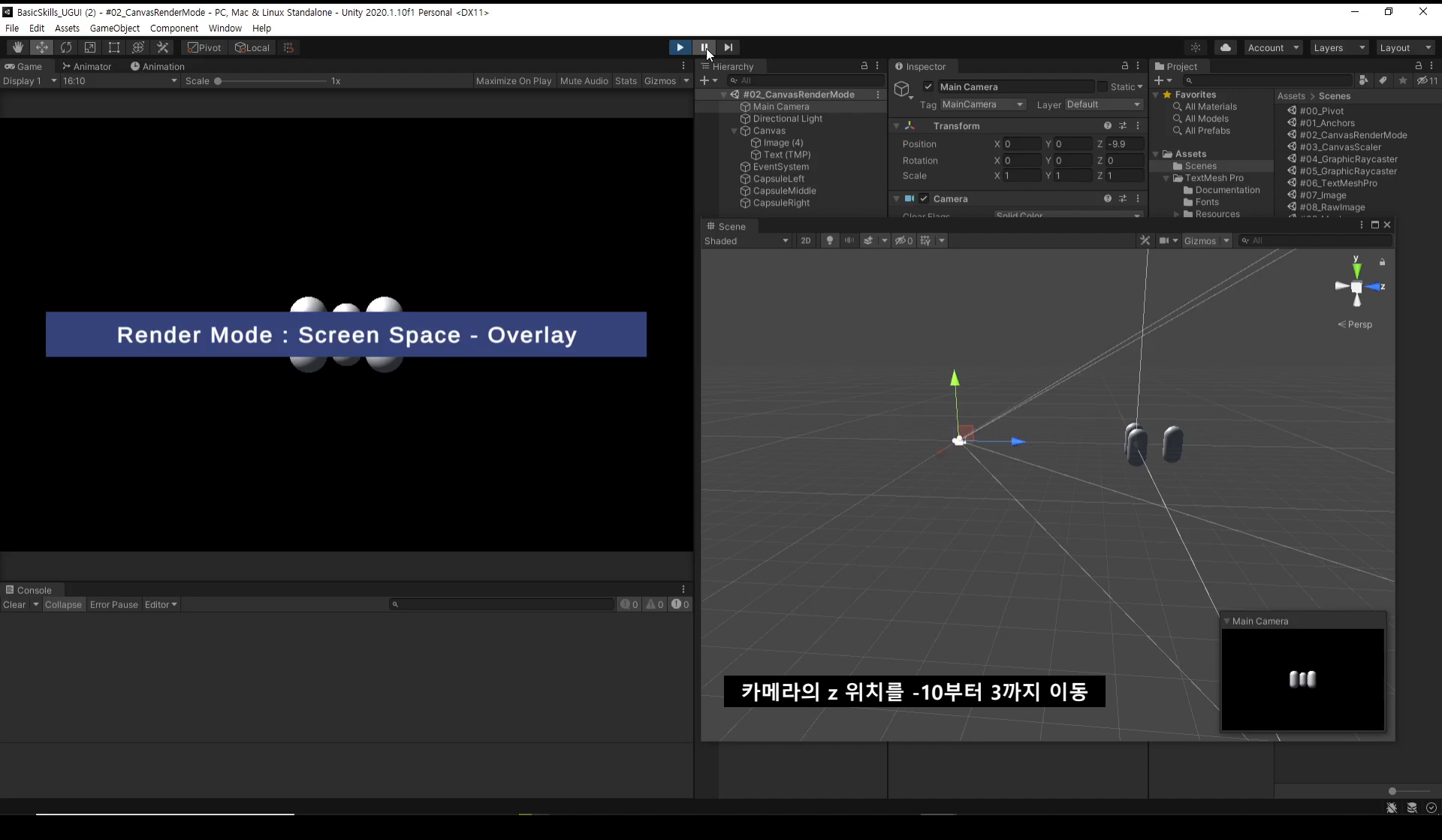
- Render Mode [Screen Space - Overlay]
  - 가장 흔히 사용되는 UI 배치 방법으로 UI가 월드의 오브젝트보다 앞에 그려진다




**Render Mode가 Screen Space일 때 UI가 렌더링되는**
**추상 공간의 크기는 게임 화면의 크기와 같게 설정되기 때문에**
**RectTransform이 잠겨서 임의 설정이 불가능하다**

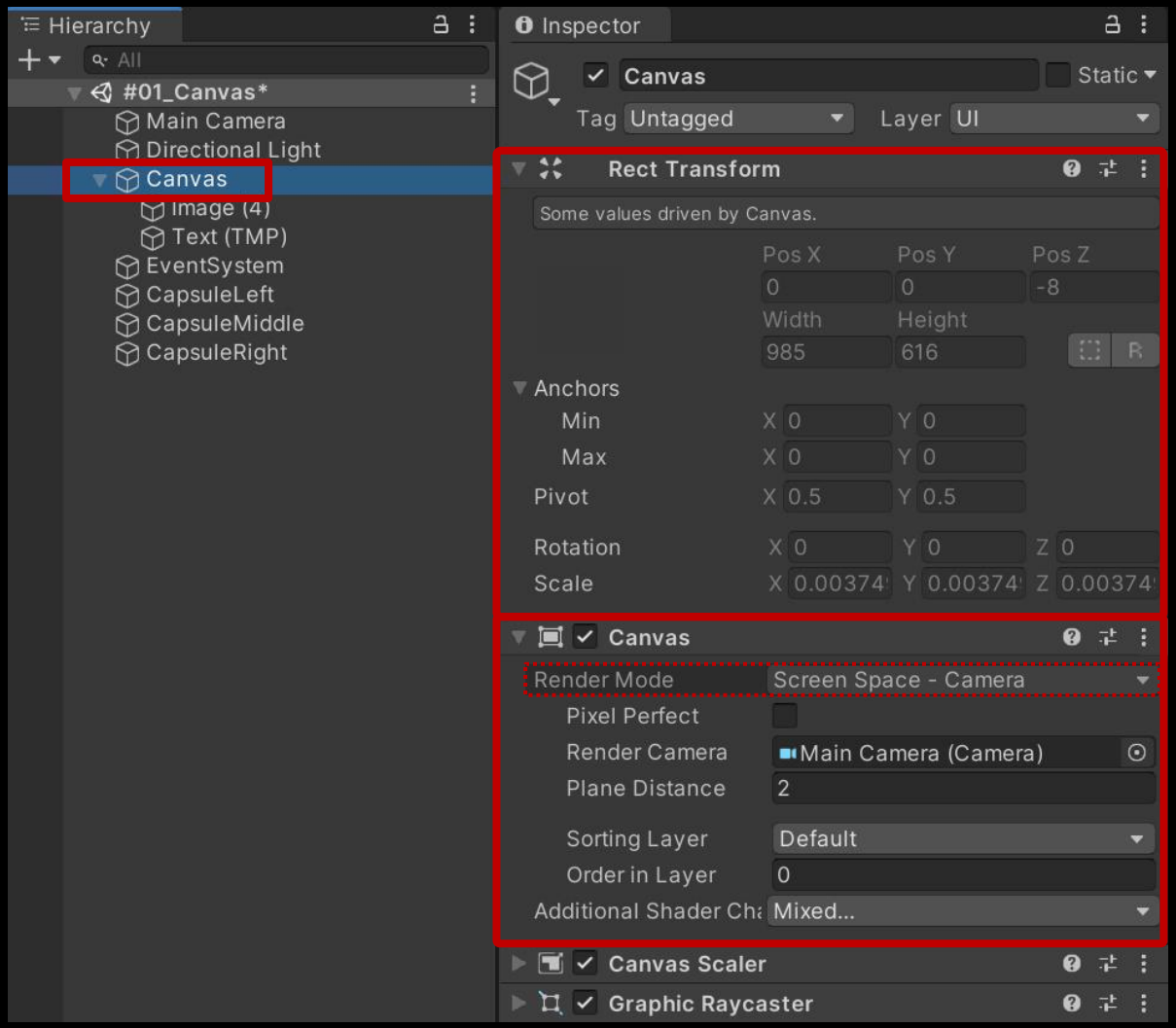
# Canvas

- Render Mode [Screen Space - Overlay] (계속)



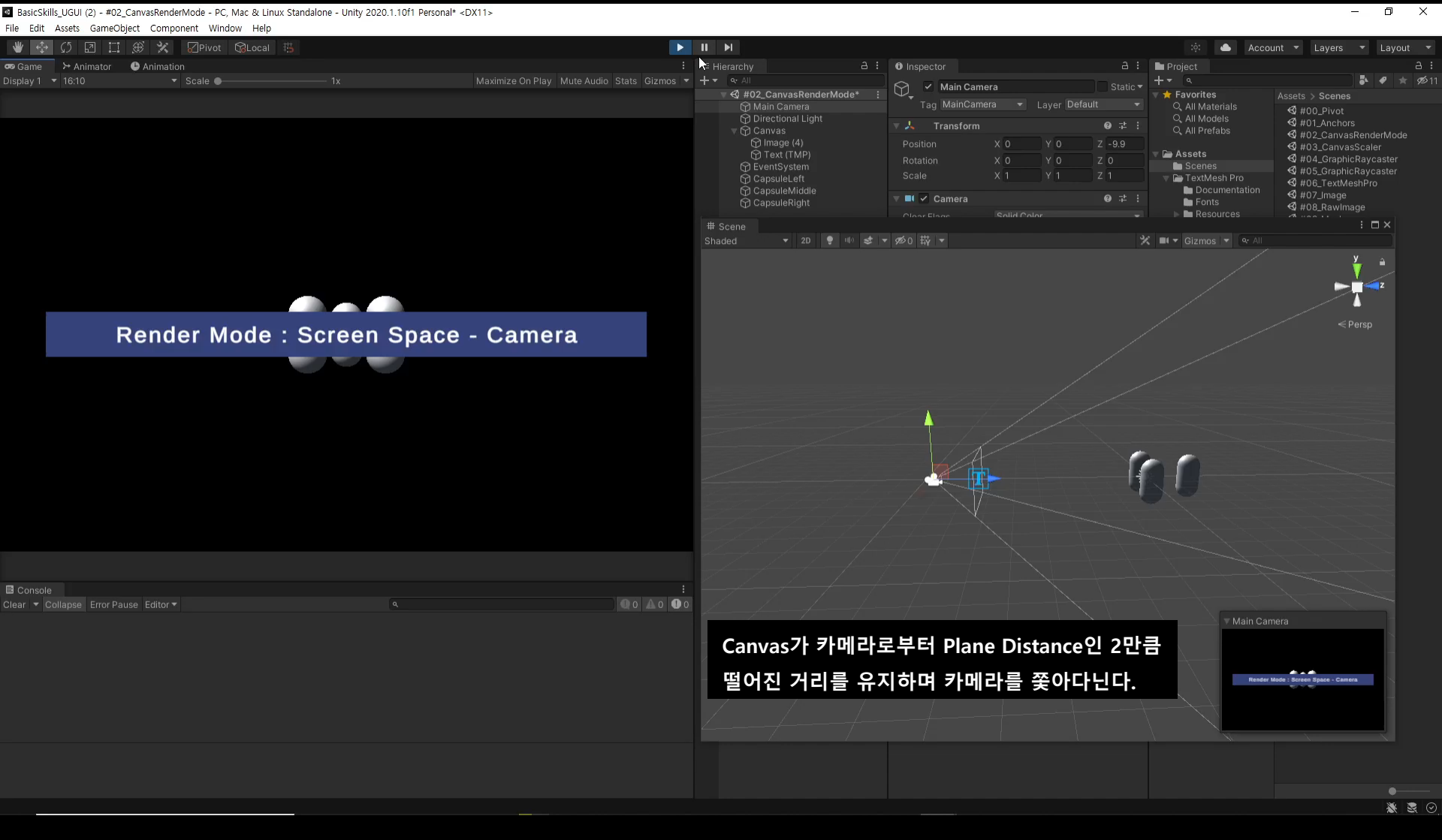
# Canvas

- Render Mode [Screen Space - Camera]
  - UI가 그려지는 위치는 Render Camera와의 거리(Plane Distance)로 설정된다



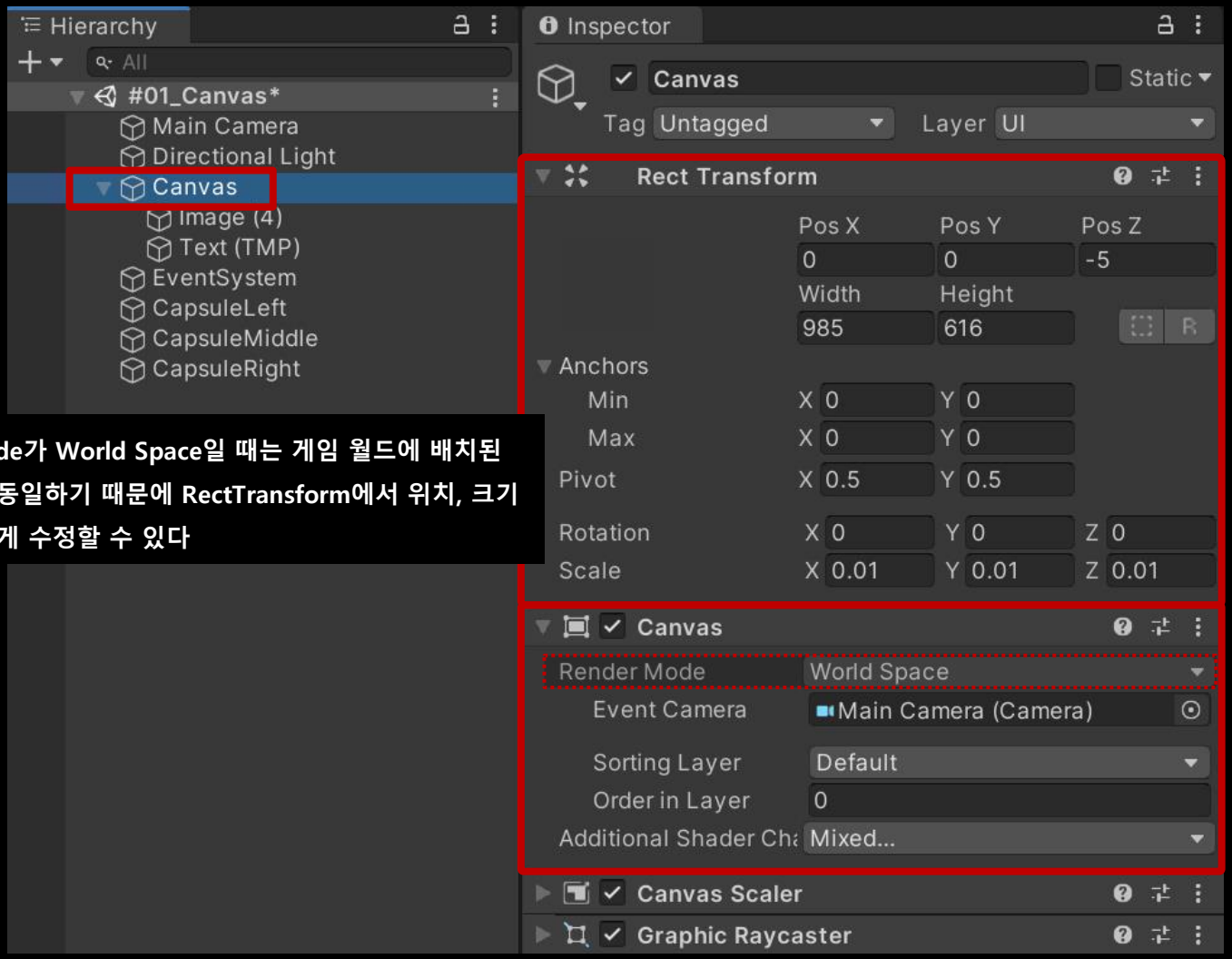
# Canvas

- Render Mode [Screen Space - Camera] (계속)



# Canvas

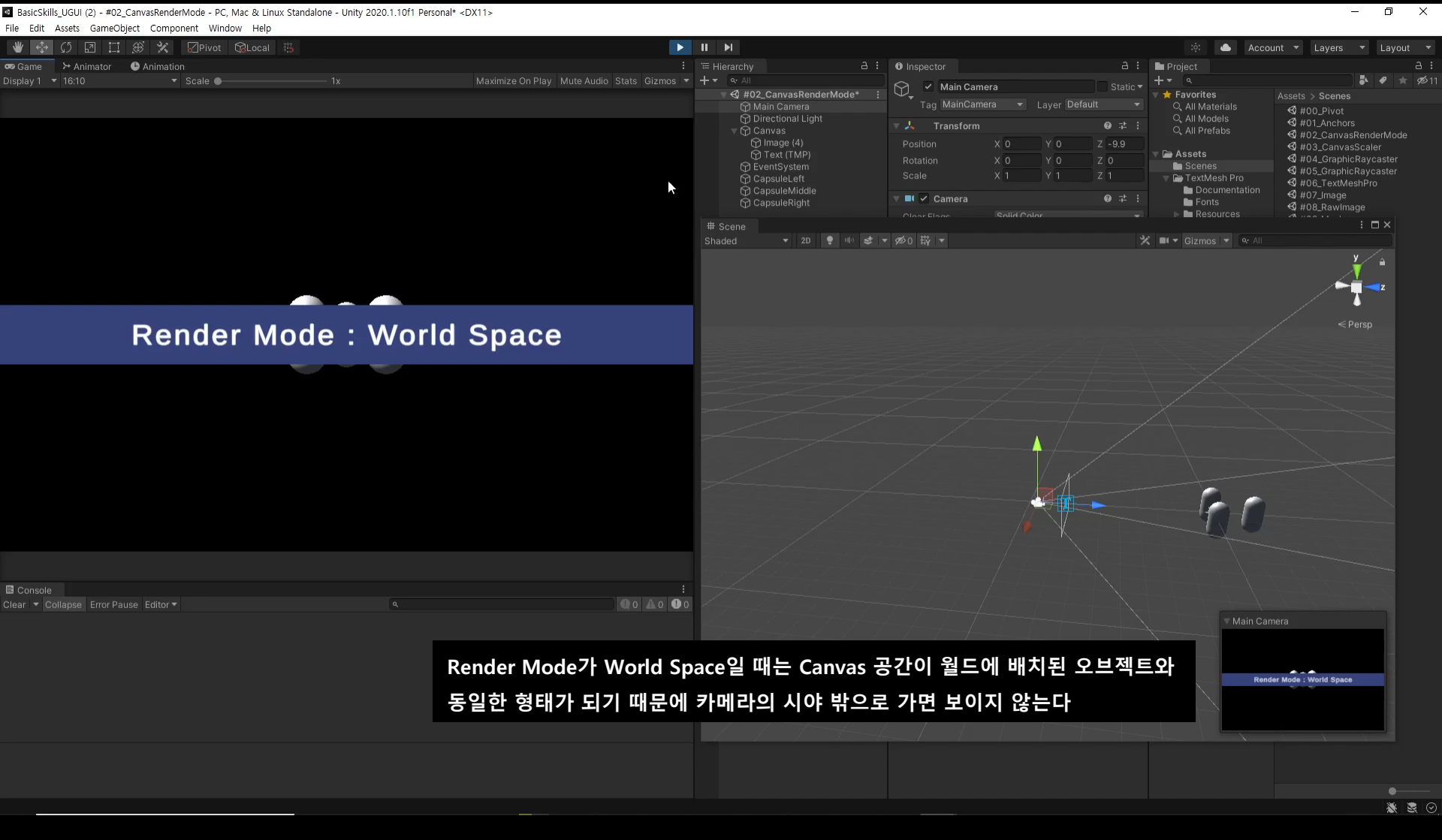
- Render Mode [World Space]
  - 월드에 배치된 오브젝트와 동일하게 카메라의 시야 내에서만 화면에 보인다




Render Mode가 World Space일 때는 게임 월드에 배치된 오브젝트와 동일하기 때문에 RectTransform에서 위치, 크기 등을 자유롭게 수정할 수 있다

# Canvas

- Render Mode [World Space] (계속)

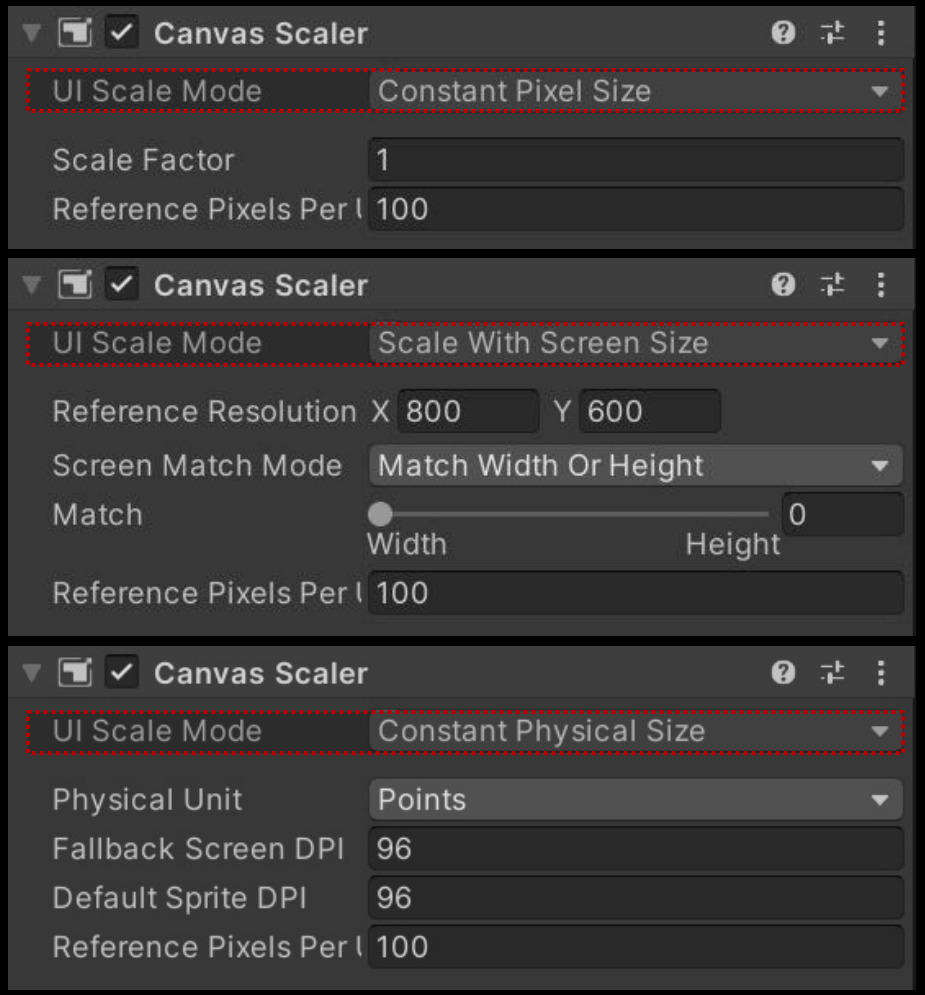



Render Mode가 World Space일 때는 Canvas 공간이 월드에 배치된 오브젝트와 동일한 형태가 되기 때문에 카메라의 시야 밖으로 가면 보이지 않는다

# Canvas

## Canvas Scaler

- Canvas에 배치된 모든 UI 요소의 크기, 픽셀 밀도를 제어하는데 사용

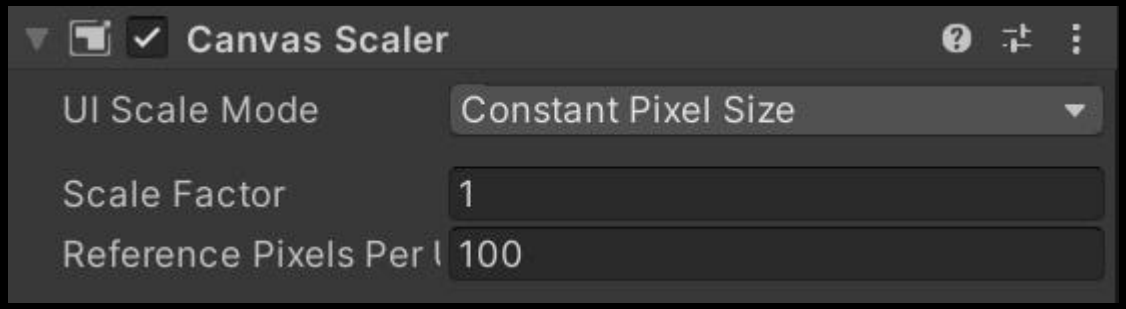



Tip. Canvas Scaler는 글꼴 크기, 이미지 테두리 등 Canvas 아래에 배치된 모든 UI에 영향을 주는데 Canvas Scaler 옵션이 바뀌었을 경우 이미 배치가 완료된 UI의 크기, 위치 등이 바뀔 수 있기 때문에 Canvas Scaler를 먼저 설정해야 한다

Tip. Canvas Scaler는 게임에 존재하는 모든 Scene에 사용되고 있는 Canvas 오브젝트를 모두 설정해야 한다

# Canvas

- UI Scale Mode [Constant Pixel Size]
  - 화면 크기에 관계 없이 UI의 위치나 크기가 픽셀에 대한 단순한 배율로 지정된다




**Scale Factor**

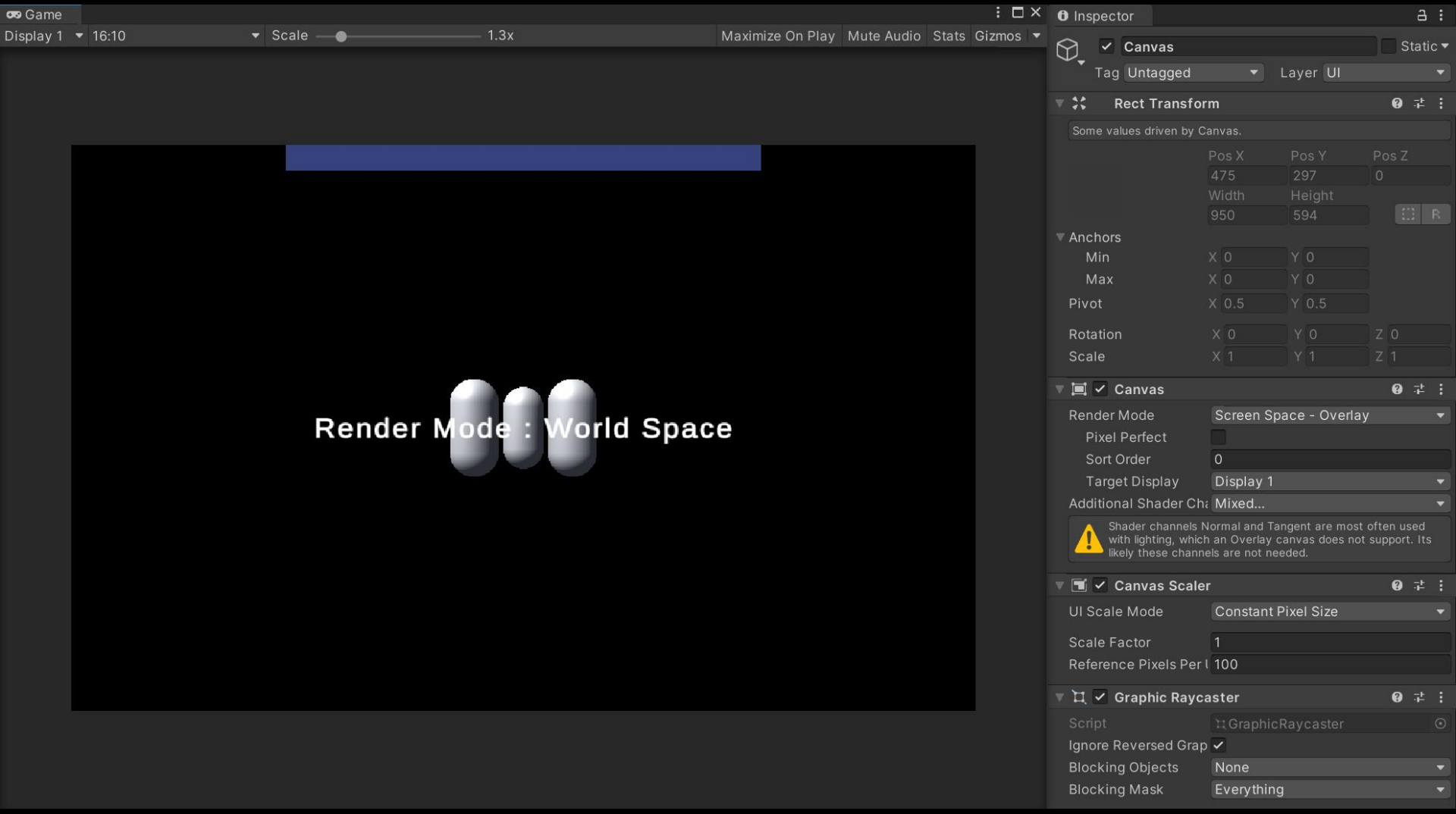
- Canvas 아래에 배치되는 모든 UI 요소의 화면 내 비율

**Reference Pixels Per Unit**

- Image 컴포넌트를 가지는 UI의 경우 Sprite에 'Pixels Per Unit' 설정이 있으면, Sprite의 1 픽셀 = UI의 1 Unit

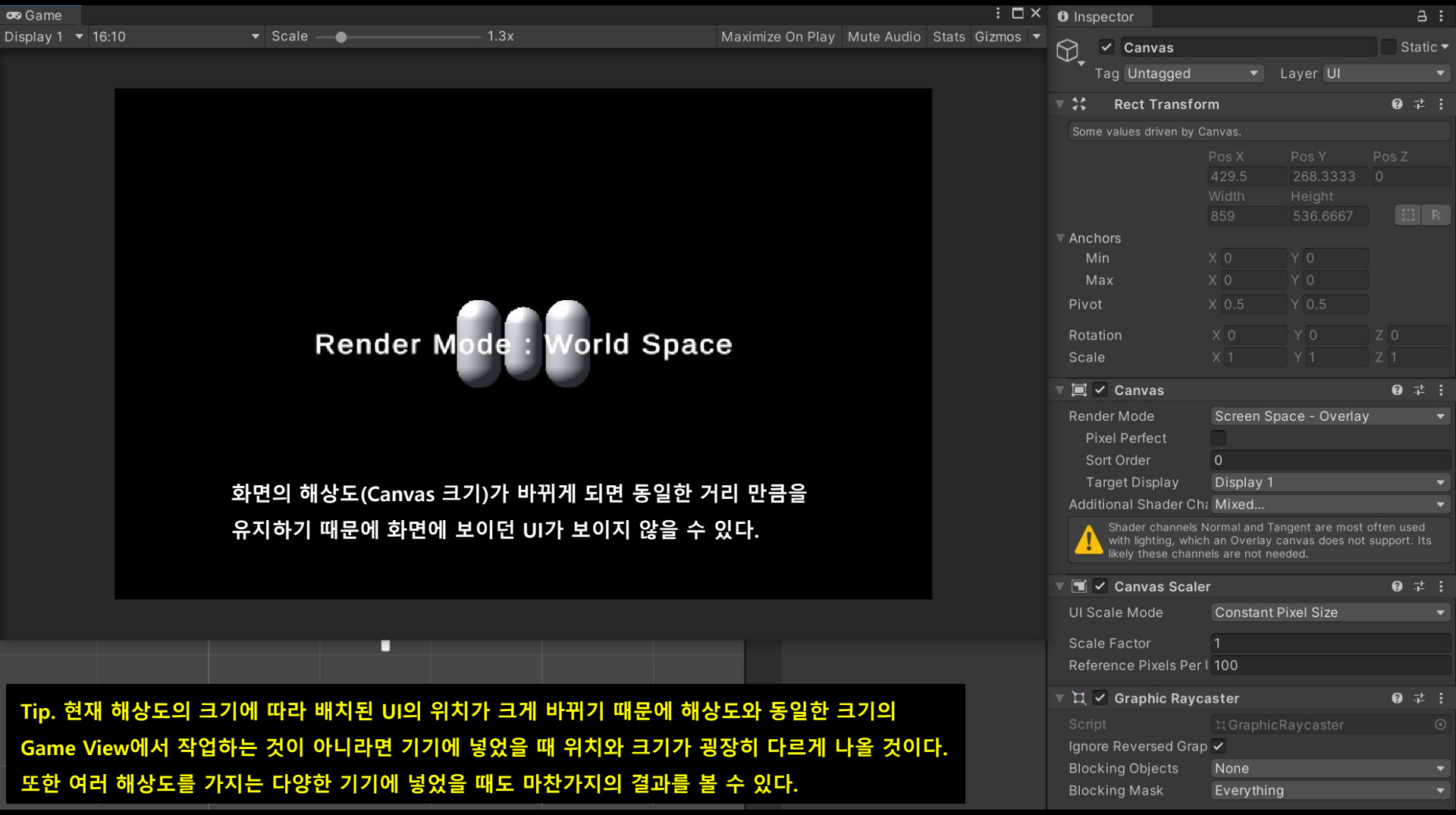
# Canvas

- UI Scale Mode [Constant Pixel Size] (계속)



# Canvas

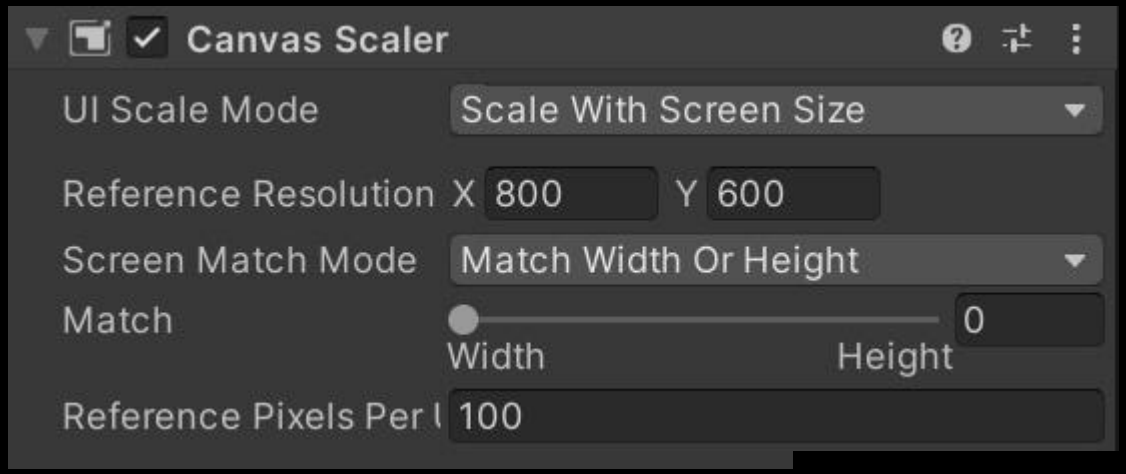
- UI Scale Mode [Constant Pixel Size] (계속)




**Tip. 현재 해상도의 크기에 따라 배치된 UI의 위치가 크게 바뀌기 때문에 해상도와 동일한 크기의 Game View에서 작업하는 것이 아니라면 기기에 넣었을 때 위치와 크기가 굉장히 다르게 나올 것이다. 또한 여러 해상도를 가지는 다양한 기기에 넣었을 때도 마찬가지의 결과를 볼 수 있다.**

# Canvas

- UI Scale Mode [Scale With Screen Size]
  - 화면의 크기에 따라 UI의 위치나 크기가 함께 수정된다




Tip. 모바일 게임을 제작할 때는 기기별 화면 비율이 제각각이기 때문에 Scale With Screen Size 모드가 가장 많이 쓰인다

**Reference Resolution**

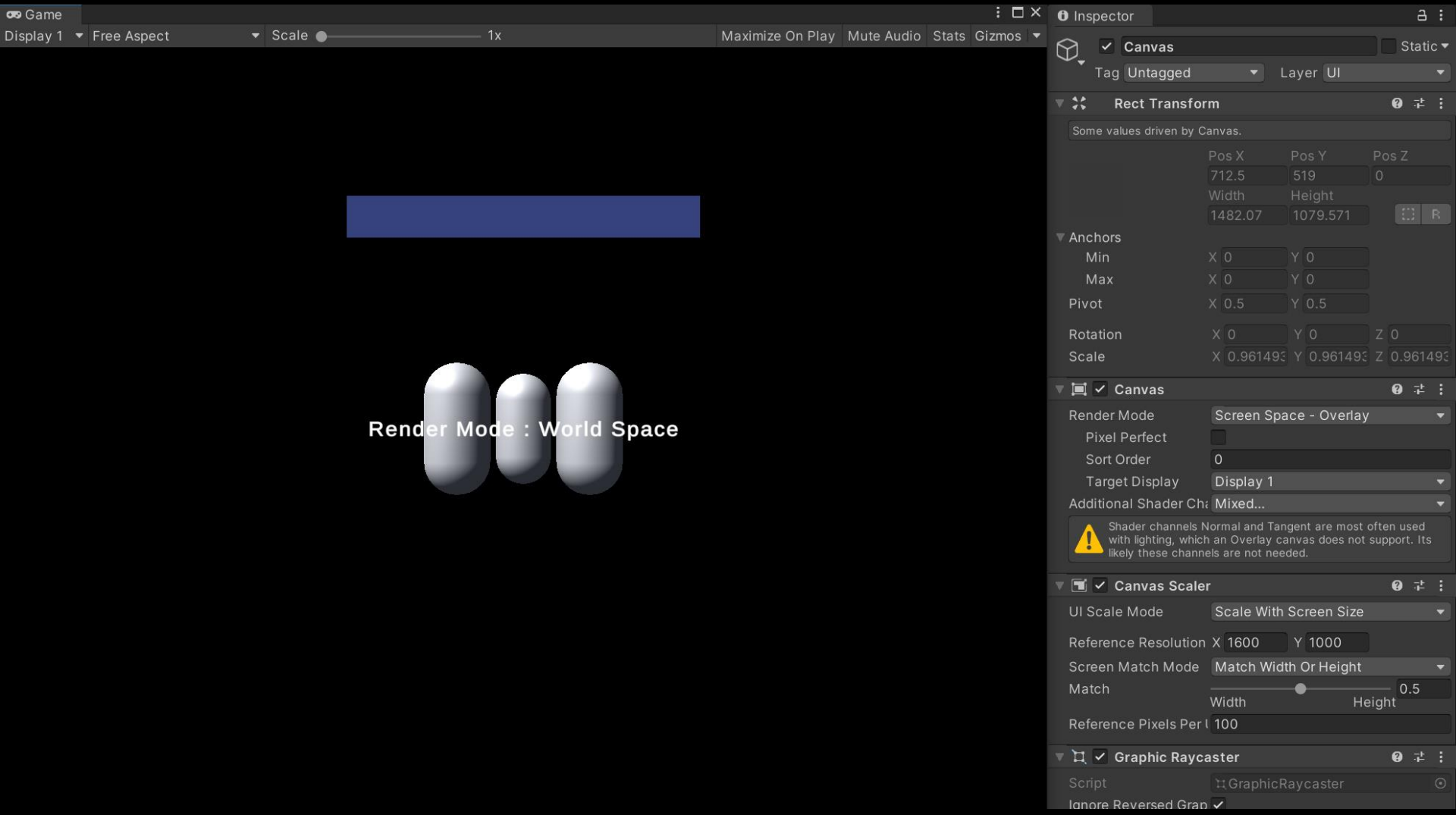
- UI의 적정 해상도 크기를 설정

**Screen Match Mode**

- 현재 해상도의 종횡비가 Reference Resolution과 같지 않을 때 Canvas 영역 크기를 설정할 때 사용되는 모드
  - + Match Width Or Height : Canvas 영역의 Width 또는 Height를 기준으로 Canvas 영역을 설정
    (Match 변수는 Width 또는 Height 중 어느 쪽을 더 정확하게 맞출 지 설정. 보통 0.5로 설정함)
  - + Expand : Canvas 크기가 Reference Resolution보다 작아지지 않도록 Canvas 영역을 수평 또는 수직으로 확장
  - + Shrink : Canvas 크기가 Reference Resolution보다 커지지 않도록 Canvas를 수평 또는 수직으로 자름

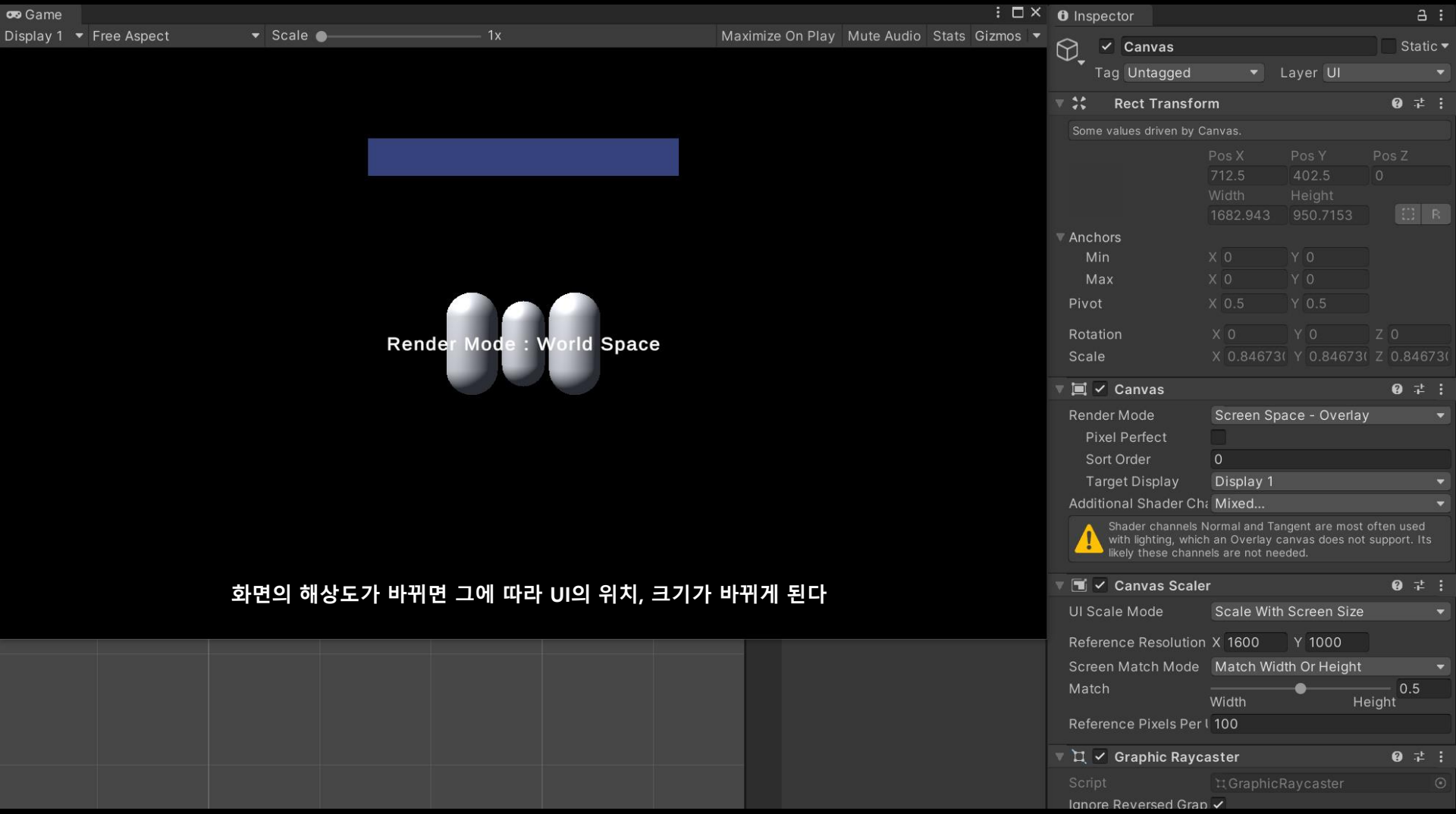
# Canvas

- UI Scale Mode [Scale With Screen Size] (계속)



# Canvas

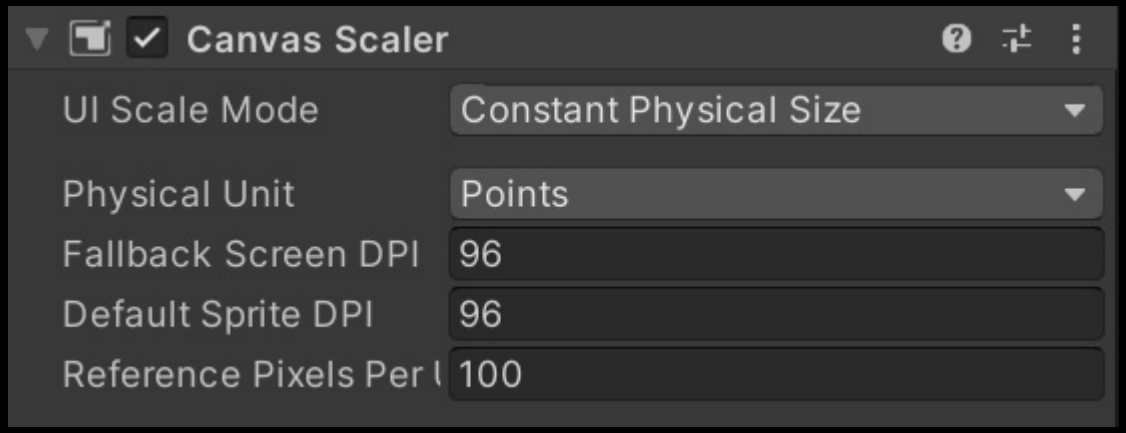
- UI Scale Mode [Scale With Screen Size] (계속)




화면의 해상도가 바뀌면 그에 따라 UI의 위치, 크기가 바뀌게 된다

# Canvas

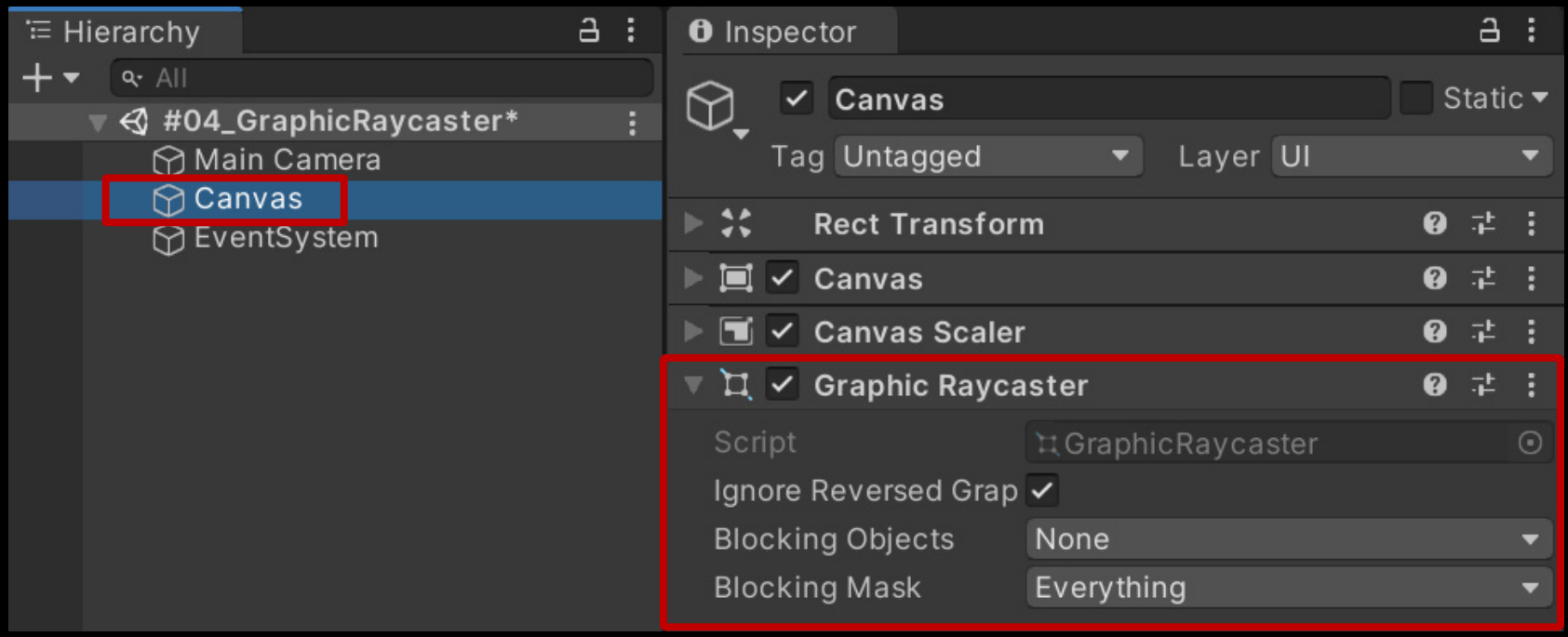
- UI Scale Mode [Constant Physical Size]
  - 화면의 크기에 관계 없이 UI 요소가 동일한 물리적인 크기로 유지된다



# Canvas

## Graphic Raycaster

- Canvas 오브젝트 하위에 배치된 UI 요소들에 광선을 쏴서 충돌처리를 한다




"Canvas" 컴포넌트의 Render Mode가 Screen Space - Overlay일 경우
카메라를 사용하지 않고 렌더링 과정에서 그려진 오브젝트들 위에 UI를 덧그린다.
이때 Canvas 안을 검색하는 Raycaster가 Graphic Raycaster이며,
EventSystem이 이벤트를 검출하는 수단으로 사용된다.

# Canvas

- Graphic Raycaster의 충돌 정보 얻어오기
  - Graphic Raycaster의 광선 충돌 정보를 얻어오는 스크립트 생성 및 작성
    - C# Script 생성 후 스크립트의 이름을 "GraphicRaycasterTest"로 변경
    - 스크립트를 "Canvas" 오브젝트의 컴포넌트로 적용

```
1   using System.Collections.Generic;
2   using UnityEngine;
3   using UnityEngine.UI;
4   using UnityEngine.EventSystems;
5
6   public class GraphicRaycasterTest : MonoBehaviour
7   {
8       private GraphicRaycaster graphicRaycaster;
9
10      private void Awake()
11      {
12          graphicRaycaster = GetComponent<GraphicRaycaster>();
13      }
14
```

# Canvas

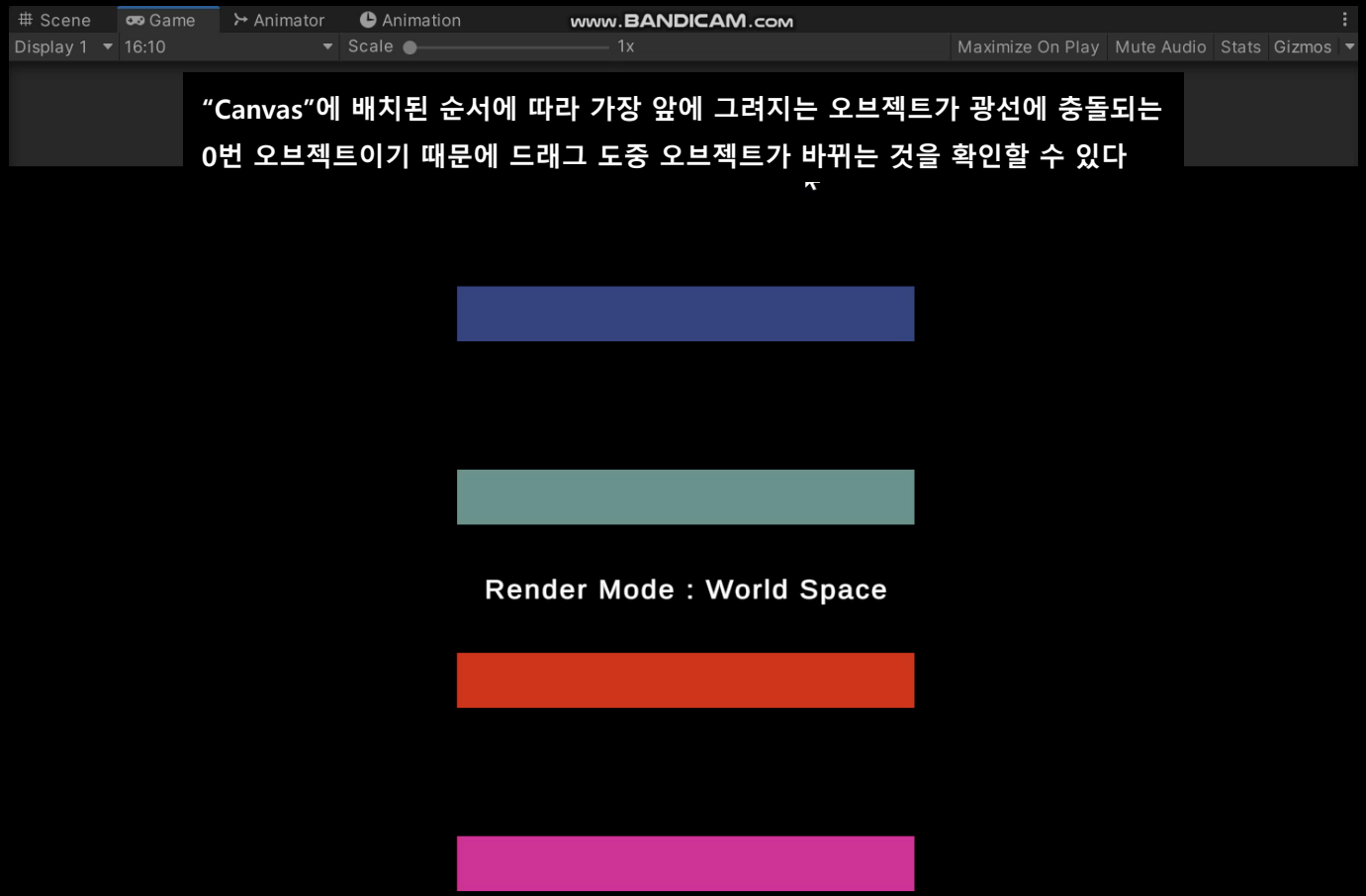
- Graphic Raycaster의 광선 충돌 정보를 얻어오는 스크립트 생성 및 작성 (계속)

```
15      private void Update()
16      {
17          var ped = new PointerEventData(null);
18
19          ped.position = Input.mousePosition;
20
21          List<RaycastResult> results = new List<RaycastResult>();
22
23          graphicRaycaster.Raycast(ped, results);
24
25          if ( results.Count <= 0 )
26          {
27              return;
28          }
29
30          Debug.Log("<color=blue>Hit : </color>"+results[0].gameObject.name);
31
32          if ( Input.GetMouseButton(0) )
33          {
34              results[0].gameObject.transform.position = ped.position;
35          }
36      }
37  }
```

광선에 부딪힌 오브젝트 중 가장 앞에서 부딪히는
0번을 대상으로 이름을 출력하거나 위치를 변경함

# Canvas

- 결과 화면



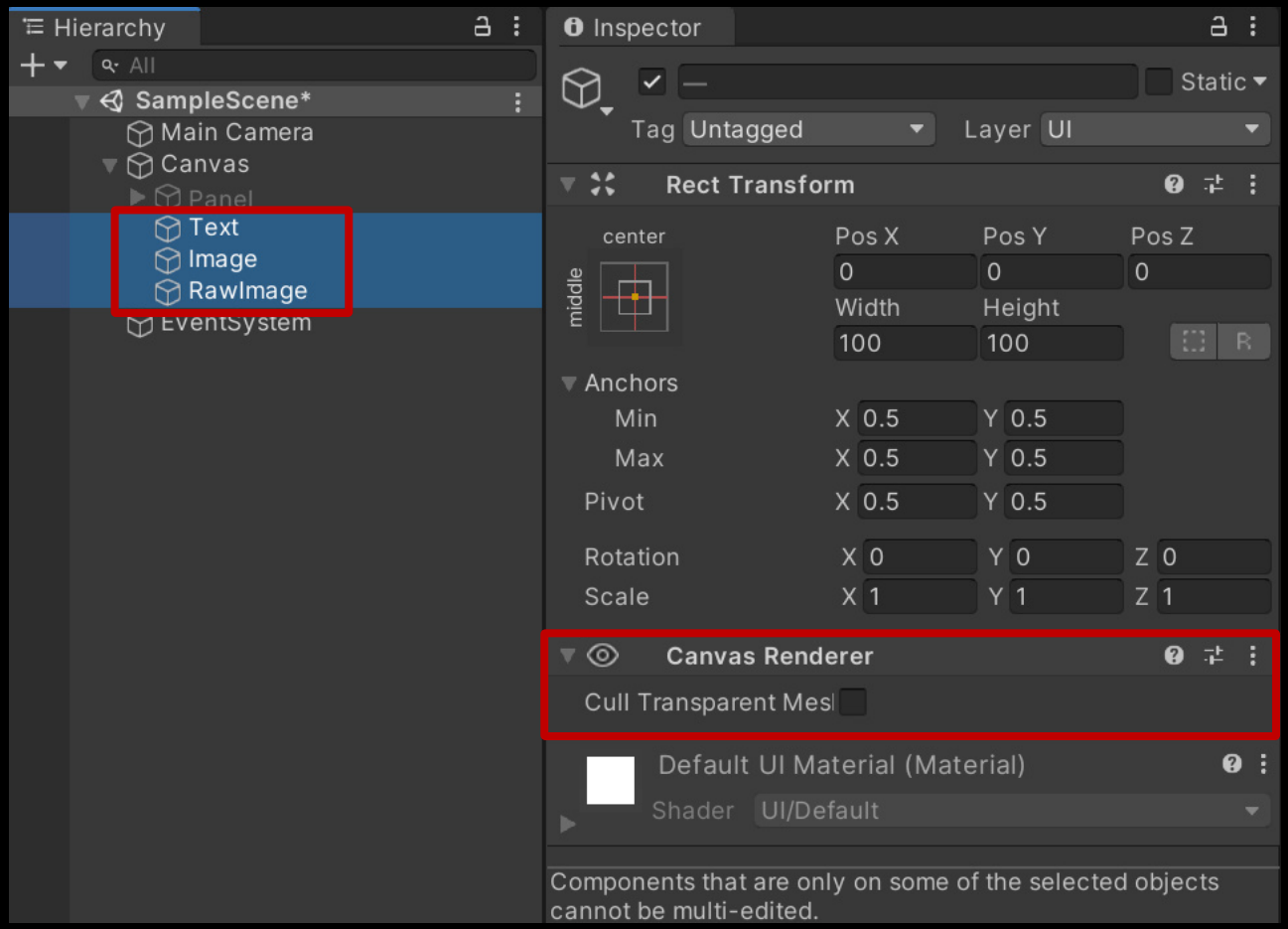

원하는 UI 오브젝트를 배치해서 실습을 진행한다

# Visual Components

- CanvasRenderer
- Text - TextMeshPro
- Image
- Mask
- UI Effects [Outline, Shadow]
- RawImage

# Visual Components

- **CanvasRenderer**
  - Canvas에 배치되어 화면에 보여지는 UI에 필수로 포함되는 컴포넌트
    - 화면에 보여지는 UI Component는 "Text", "Image", "RawImage"

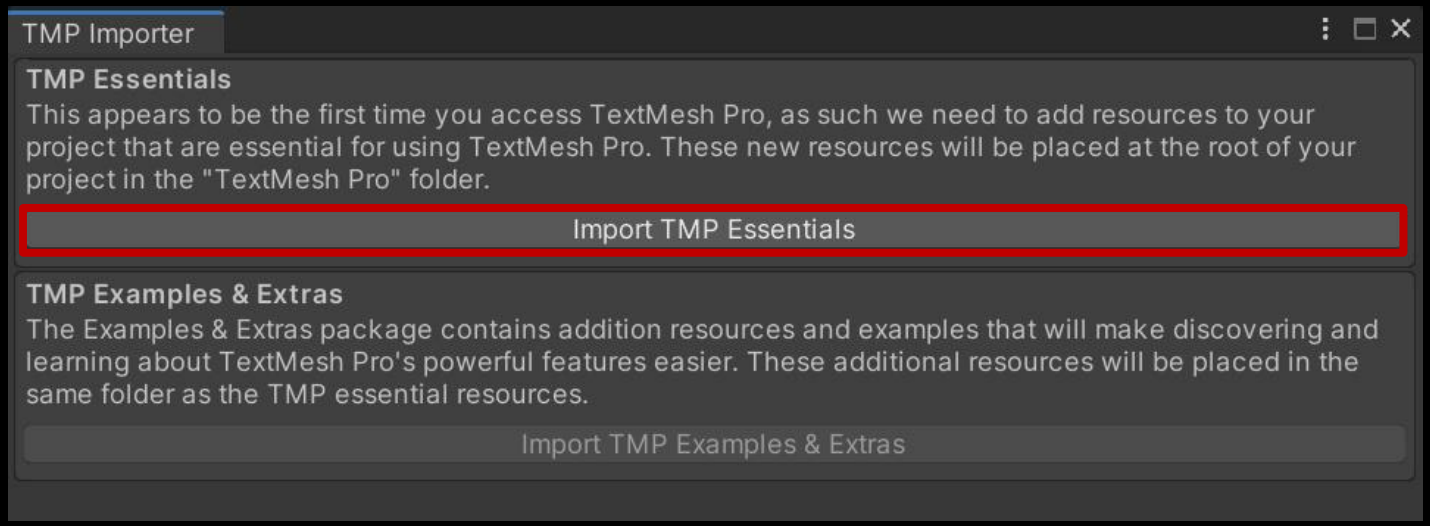


# Visual Components

Tip. 현재 "Toggle" 오브젝트만
"Text - TextMeshPro"를 제공하지 않는다.

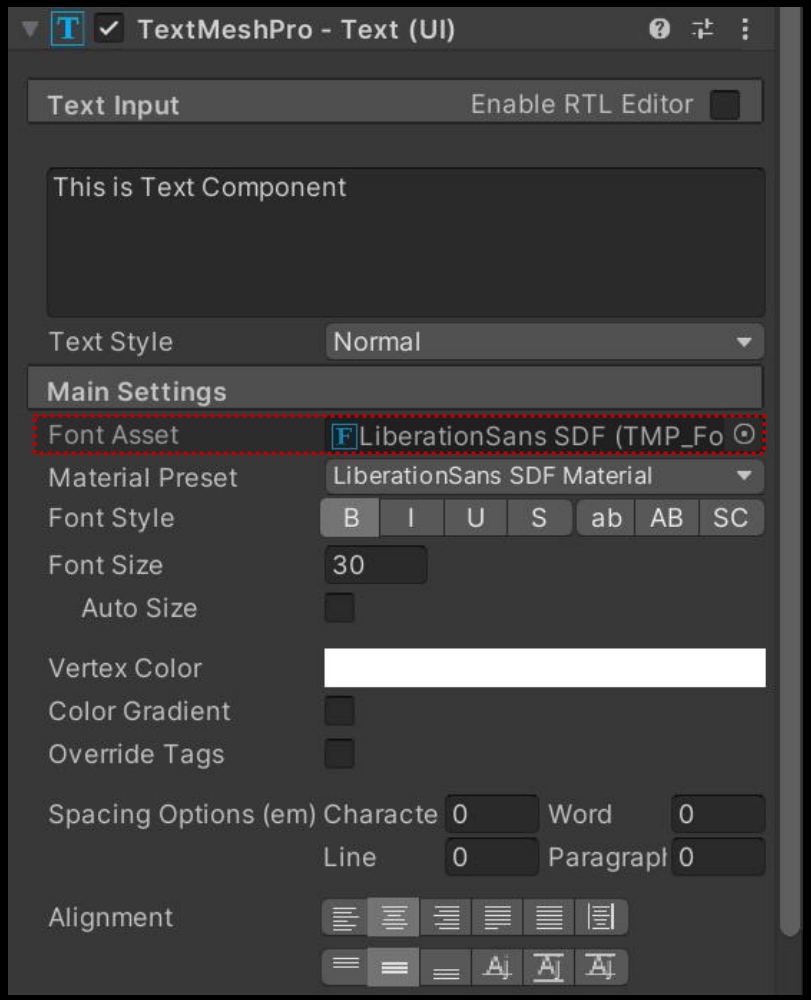
## Text - TextMeshPro

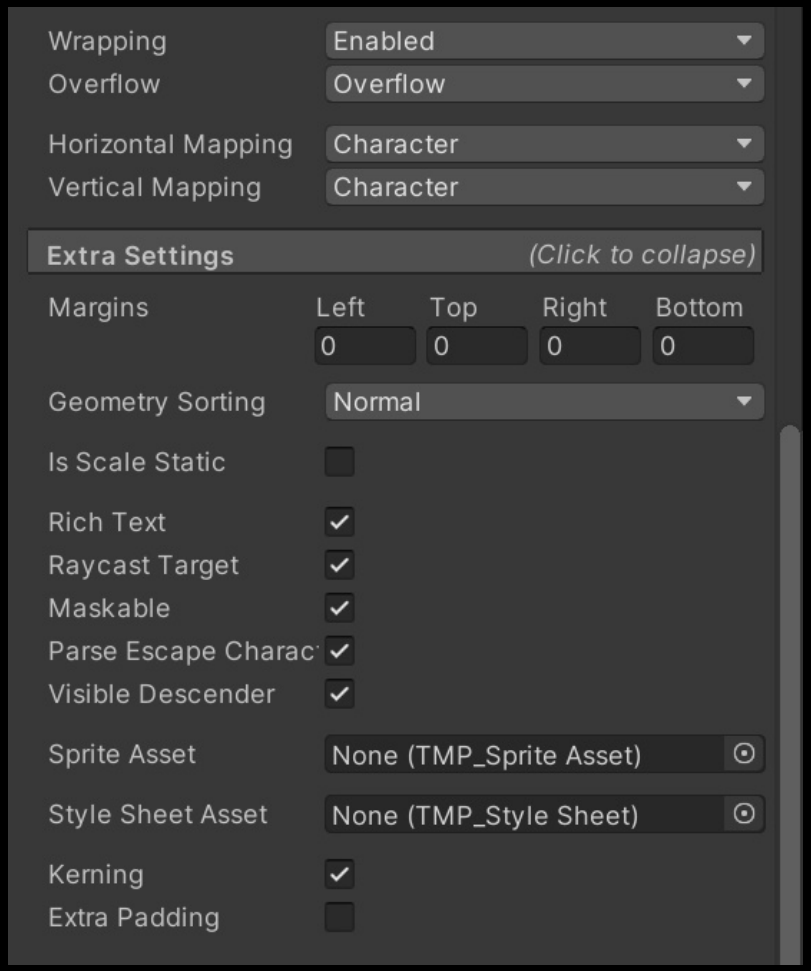
- 게임 화면에 텍스트를 표시하는 UI 컴포넌트
  - "InputField", "Button", "Dropdown"과 같이 화면에 텍스트를 표시하는 모든 UI에 "Text - TextMeshPro" 컴포넌트가 포함되어 있다
  - "Text" 컴포넌트는 Legacy로 곧 사용이 중지될 예정으로 텍스트를 포함하는 다른 UI도 "Text - TextMeshPro"가 붙은 것을 대체하여 사용한다
  - 최초에 생성할 때는 "TextMeshPro" 에셋을 Import 하기 때문에 아래 그림과 같이 TMP Importer View가 활성화된다



# Visual Components

- "Text - TextMeshPro" Variables

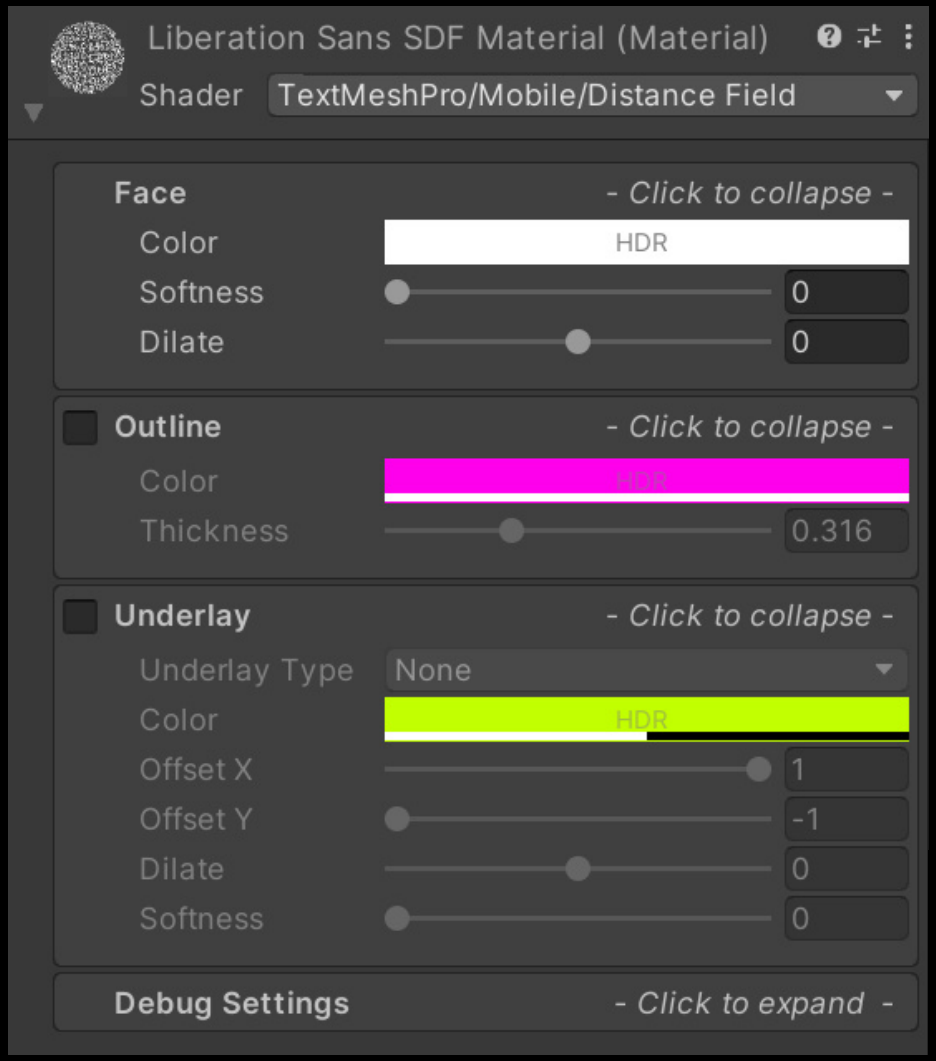



**Tip. Text - TextMeshPro의 기본 폰트는 영어만 지원하기 때문에**
**Font Asset을 새로 만들어 한글, 특수문자를 사용할 수 있도록 설정한다.**

# Visual Components

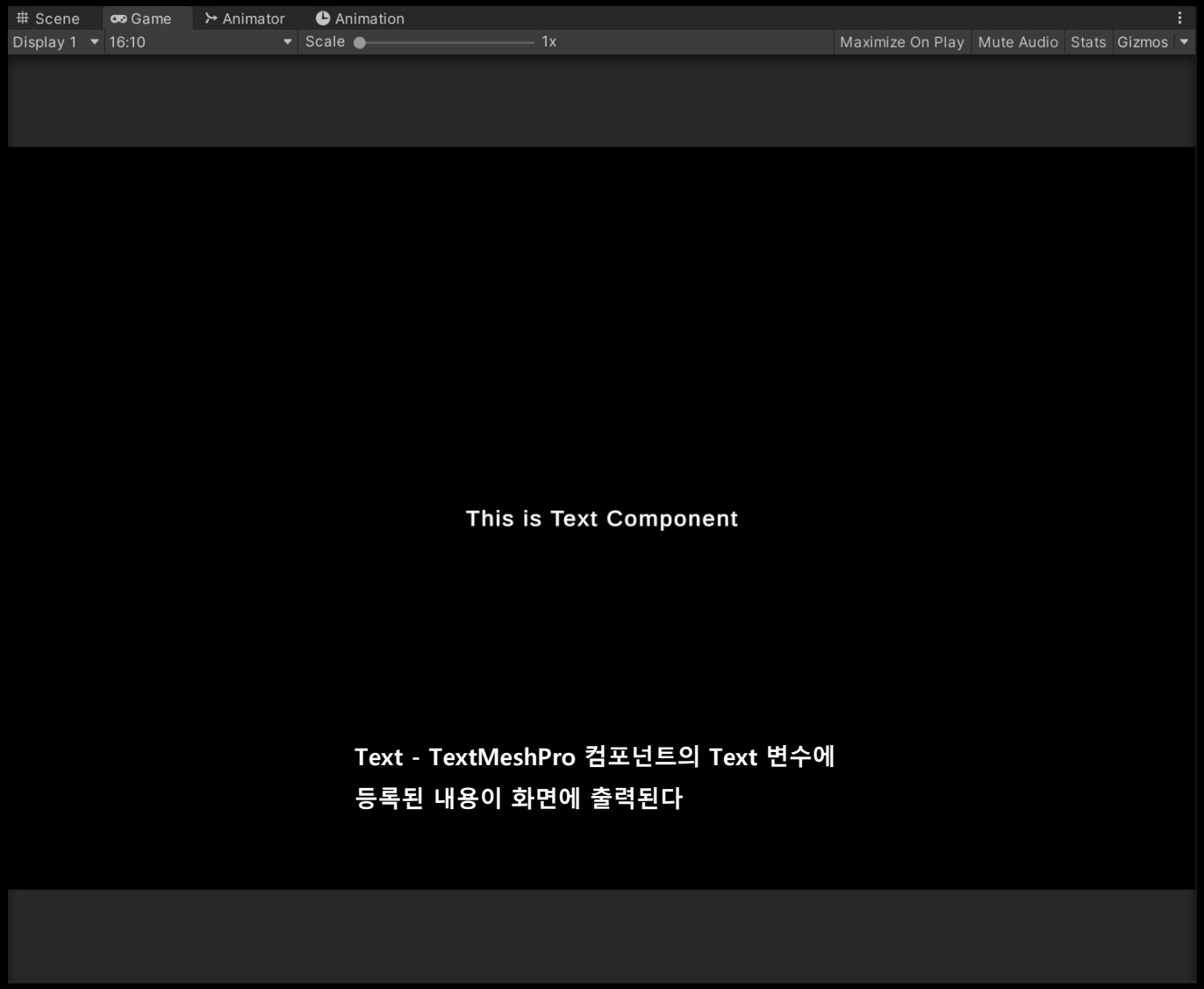
- "Text - TextMeshPro" Material




기존의 "Text" 컴포넌트는 "Outline", "Shadow"와 같이 추가 컴포넌트를 적용해야 일부 지원하는 텍스트 효과를 "Text-TextMeshPro" 컴포넌트는 Material에 있는 파라미터를 설정해 효과를 추가할 수 있다.

# Visual Components

- 결과 화면




**Text - TextMeshPro 컴포넌트의 Text 변수에**
**등록된 내용이 화면에 출력된다**

# Visual Components

- Text - TextMeshPro 컴포넌트를 제어하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "TMPTest"로 변경
  - 스크립트를 "Text (TMP)" 오브젝트의 컴포넌트로 적용

```
1  using UnityEngine;
2  using TMPro;
3
4  public class TMPTest : MonoBehaviour
5  {
6      private TextMeshProUGUI textSample;
7
8      private void Awake()
9      {
10         textSample = GetComponent<TextMeshProUGUI>();
11
12         textSample.text         = "This font don't support korean.";
13         textSample.color        = Color.red;
14         textSample.fontSize     = 40;
15         textSample.fontStyle    = FontStyles.Bold | FontStyles.Italic;
16     }
17 }
```

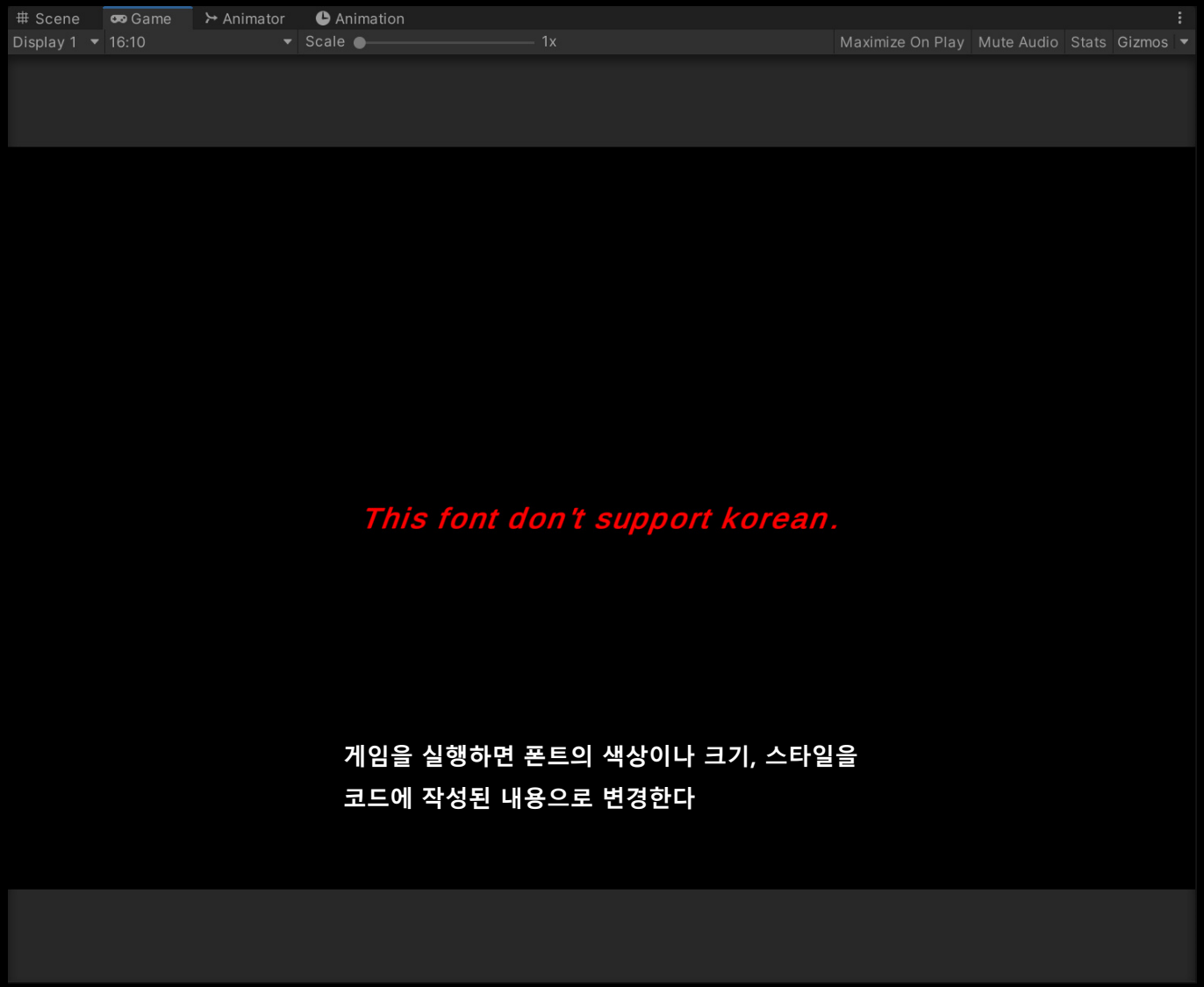
TextMeshPro에서 제공하는 클래스를 사용하려면 using TMPro; 추가

Text - TextMeshPro 컴포넌트는 TextMeshProUGUI 클래스 사용

TextMeshProUGUI 클래스의 text 변수가 화면에 출력되는 문자열이 저장되는 변수

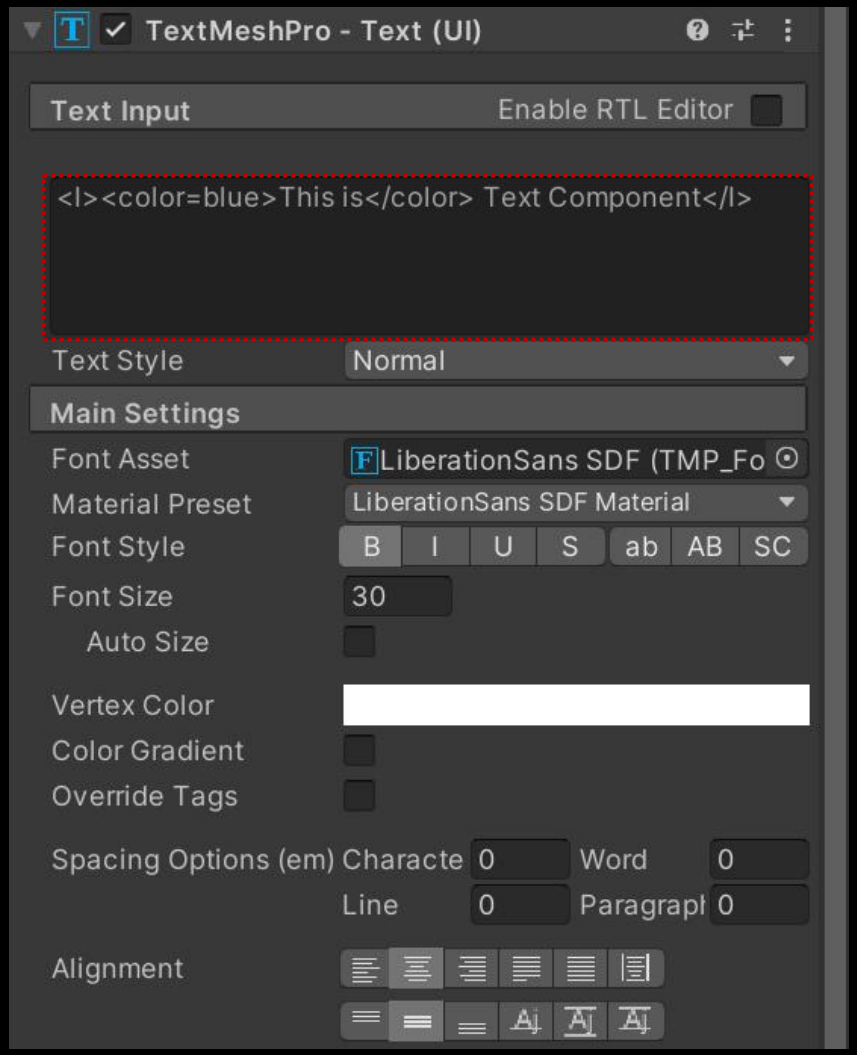
# Visual Components

- 결과 화면



# Visual Components

- 서식 있는 텍스트 (Rich Text) 지원




HTML과 유사하게 태그를 사용해 폰트의 스타일을 바꿀 수 있다

지원되는 태그

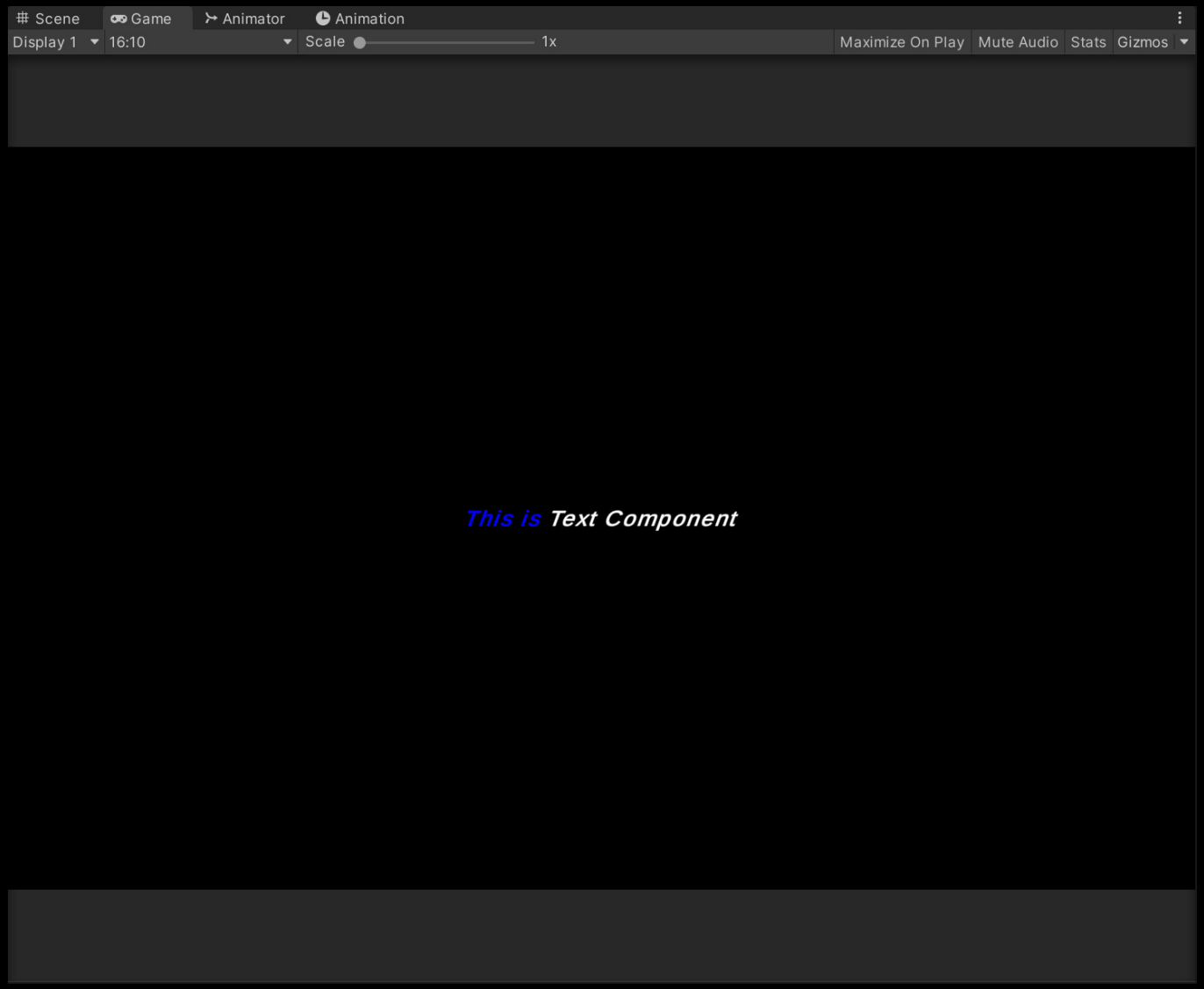
- b : 텍스트를 굵게
- i : 텍스트 기울이기
- size : 텍스트 크기
- color : 텍스트 색상

<size=10>과 같이 시작부분을 설정했으면

</size>와 같이 종료 부분도 설정해야 한다

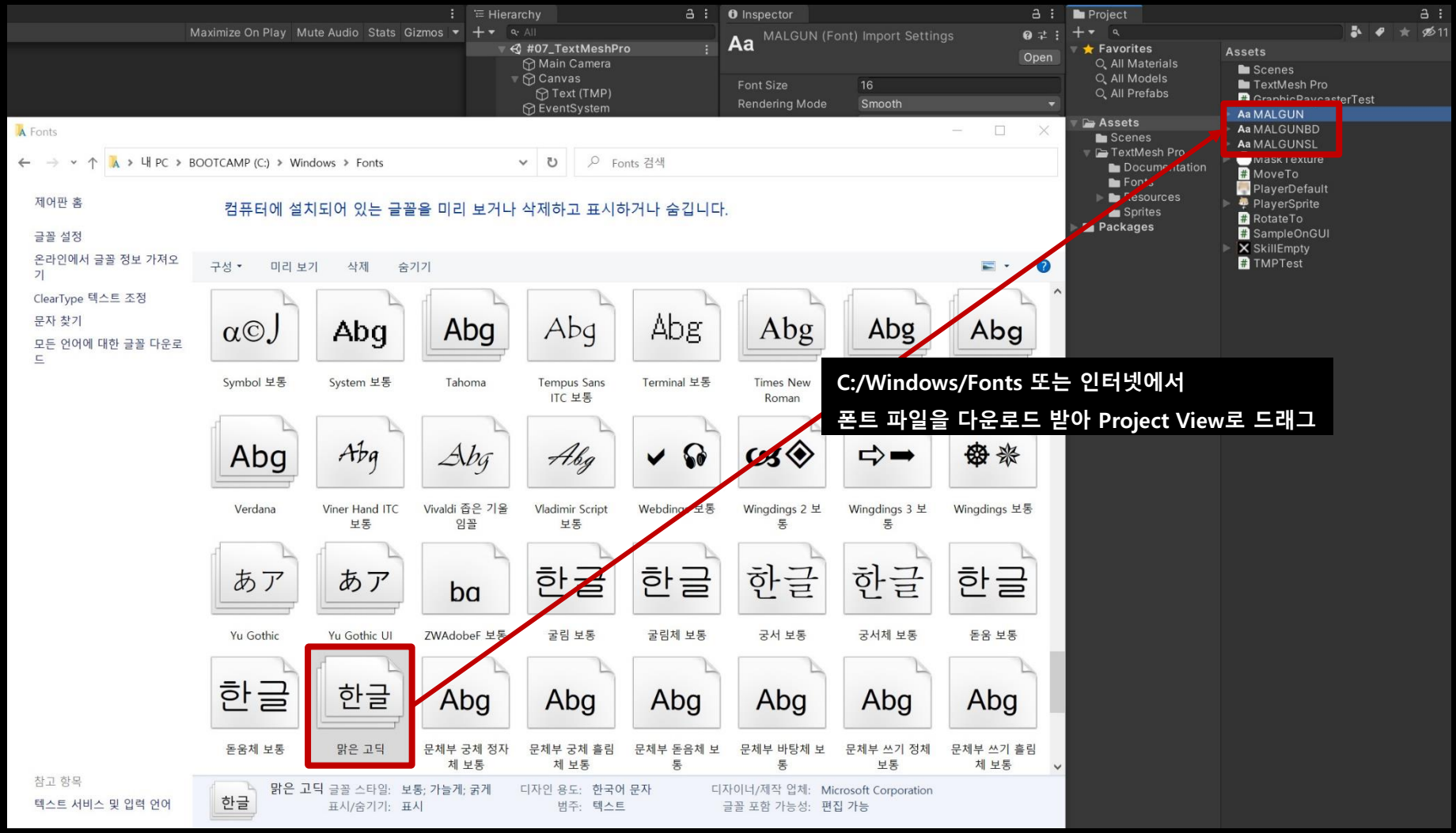
# Visual Components

- 결과 화면



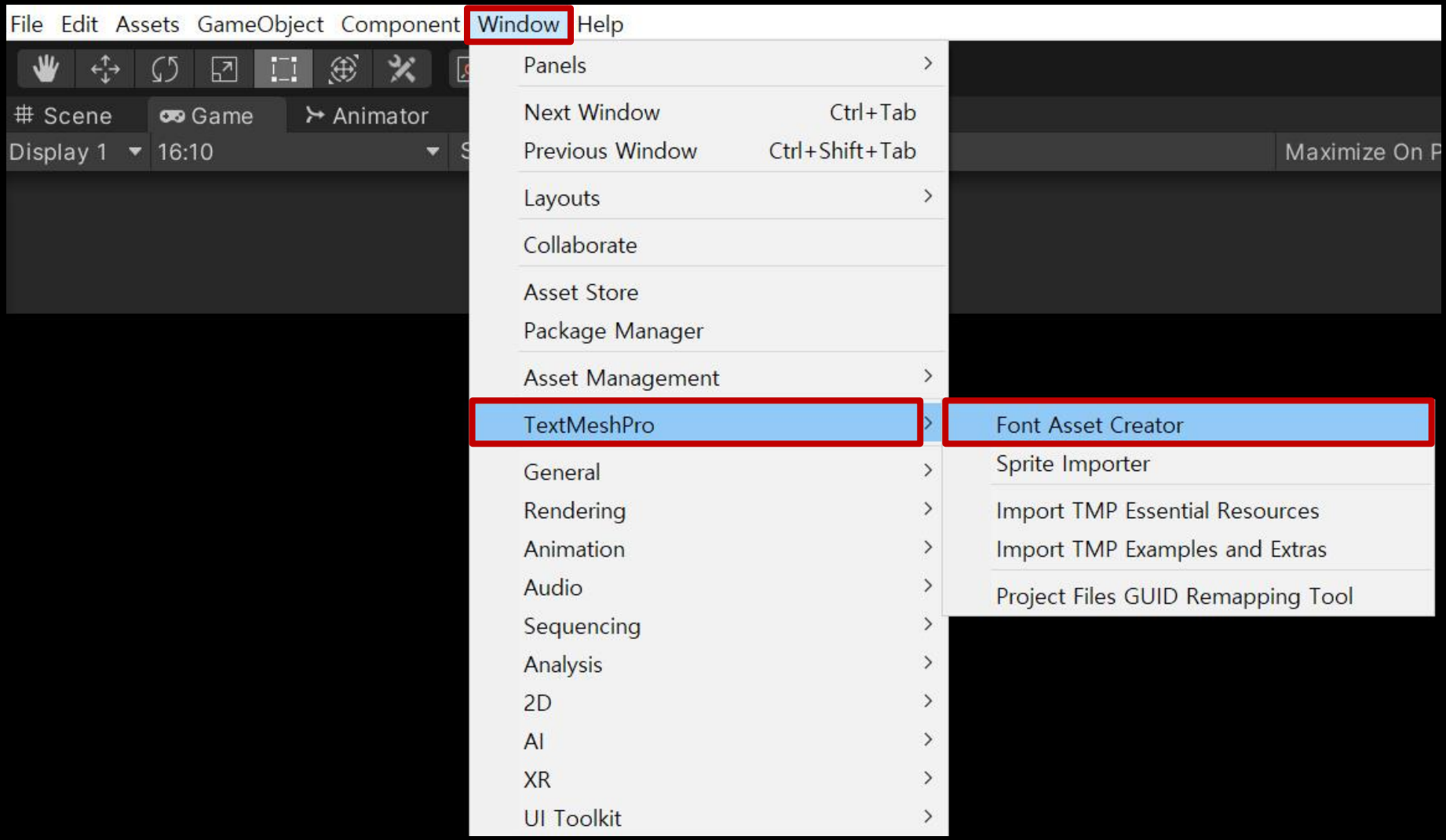
# Visual Components

- Text - TextMeshPro 폰트 에셋 생성
  - 폰트로 사용할 Font 파일 불러오기



# Visual Components

- Window - TextMeshPro - Font Asset Creator



# Visual Components

- 해당 폰트 정보를 Atlas 형태로 Material에 저장하는 과정

1. 폰트로 사용할 파일을 등록
2. Atlas Resoultion을 설정해 텍스트가 저장될 이미지 크기를 설정
(크기에 따라 용량이 설정된다. 하지만 너무 크기가 작으면 텍스트의 해상도가 낮게 설정된다.)
3. Character Set을 "Custom Characters"로 설정해 작성 가능한 텍스트를 직접 설정한다.

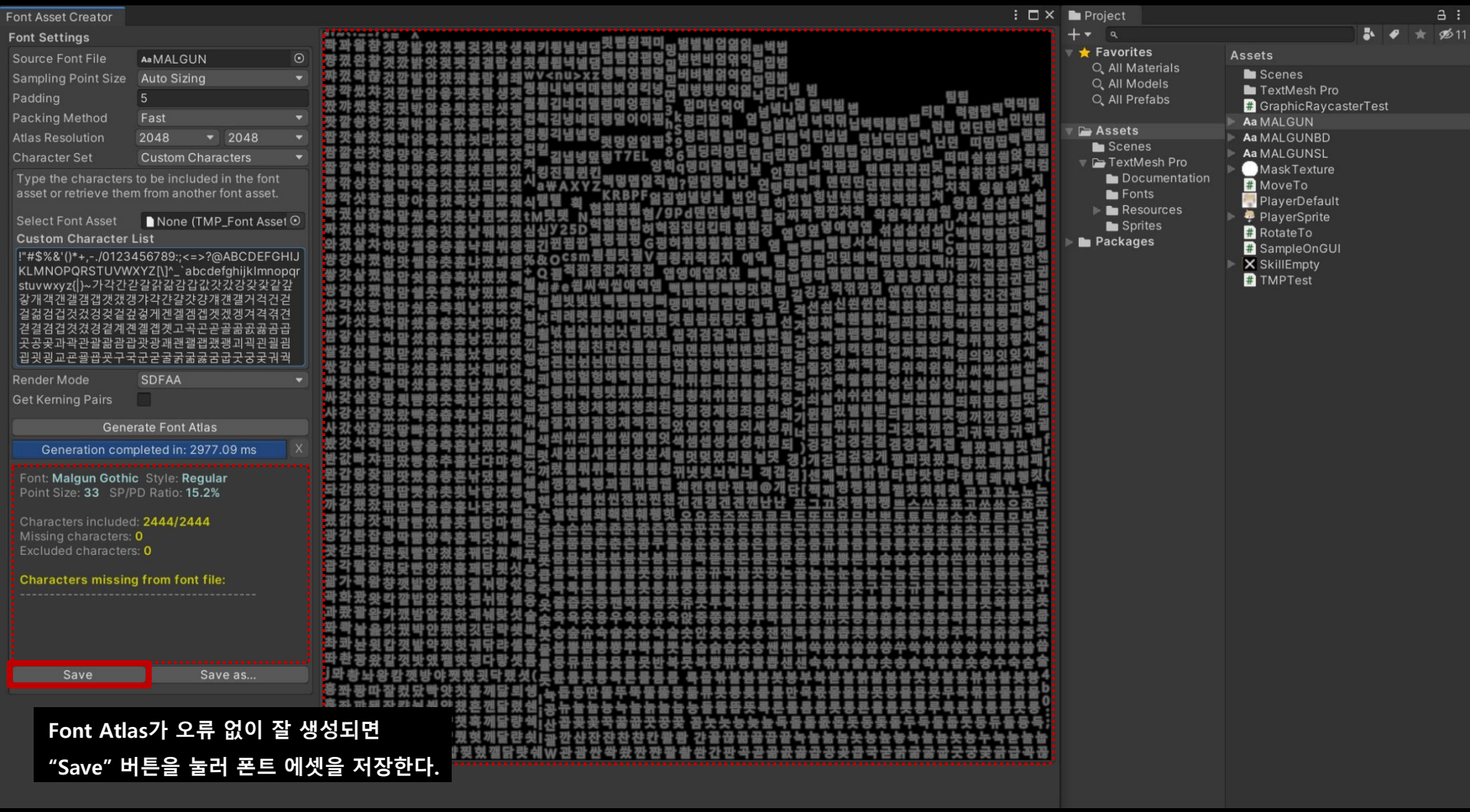
TextMeshPro용
커스텀 한글
2350자

우리가 Custom Character List에 등록한 글자들만 제대로 출력되고, 나머지는 ㅁㅁ와 같이 출력된다.

설정이 완료되면 "Generate Font Atlas" 버튼을 눌러 폰트 Atlas 생성

# Visual Components

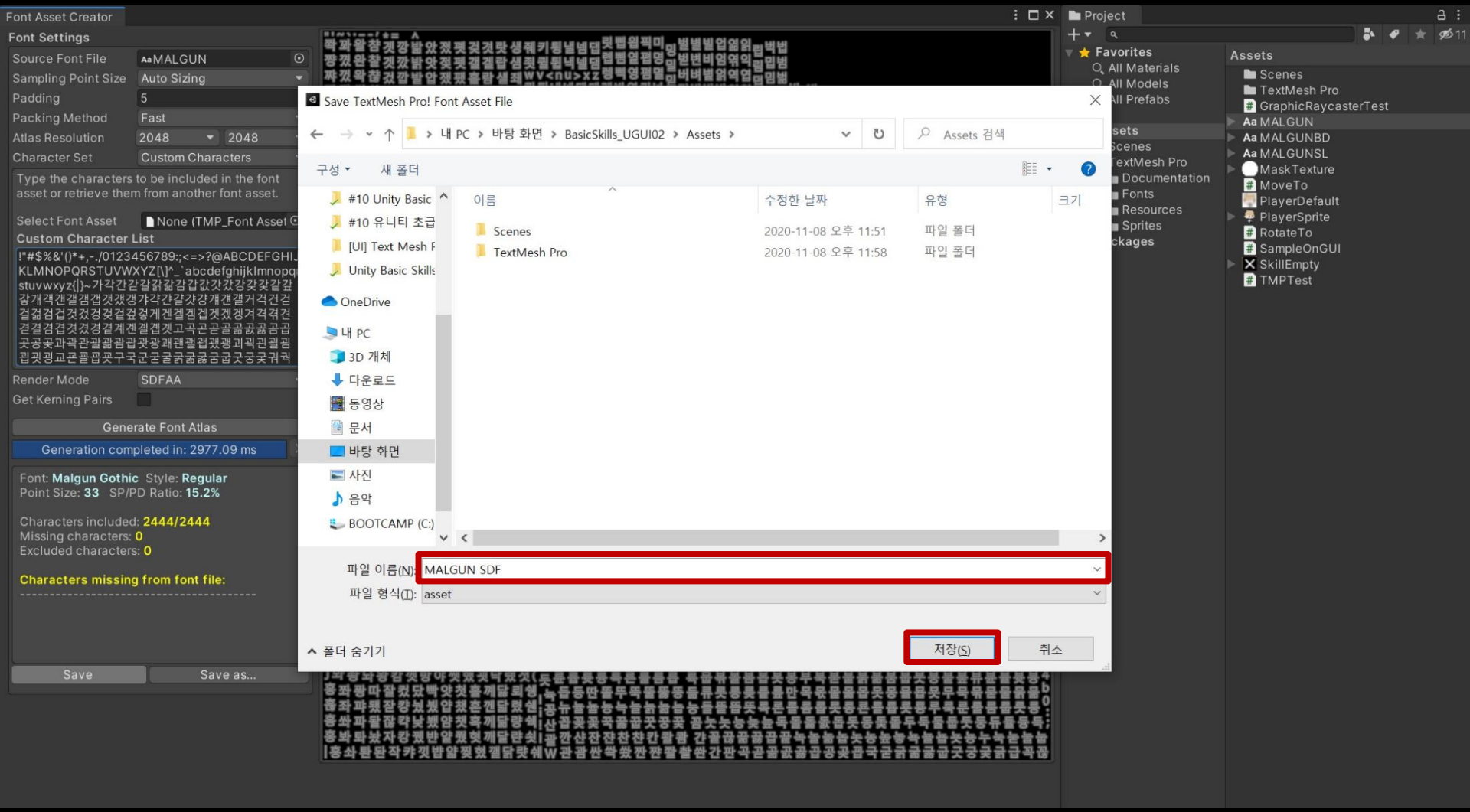
- 생성이 완료된 폰트 에셋




**Font Atlas가 오류 없이 잘 생성되면**
**"Save" 버튼을 눌러 폰트 에셋을 저장한다.**

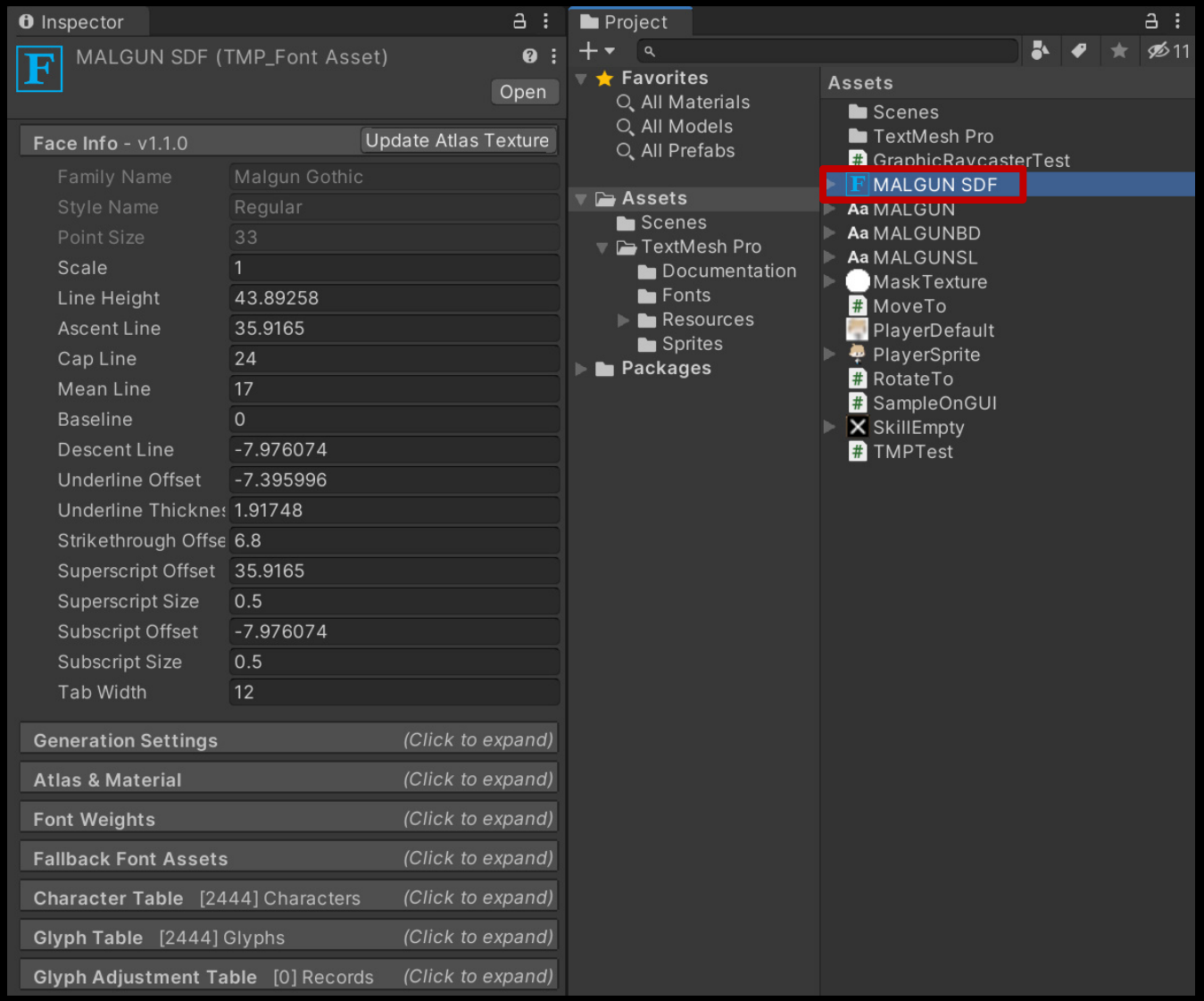
# Visual Components

- 원하는 이름을 입력하고 폰트 에셋 저장



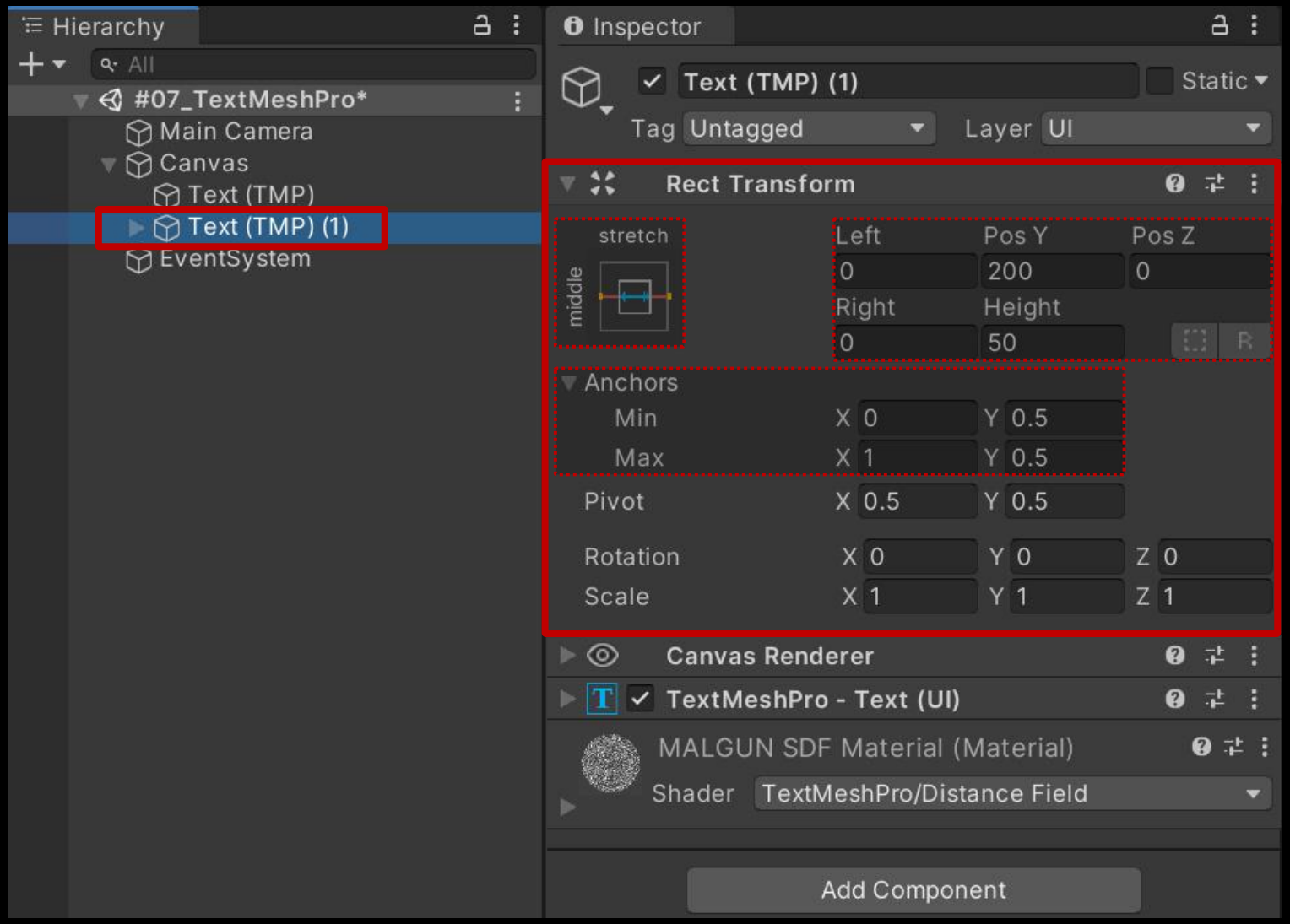
# Visual Components

- 저장된 폰트 에셋



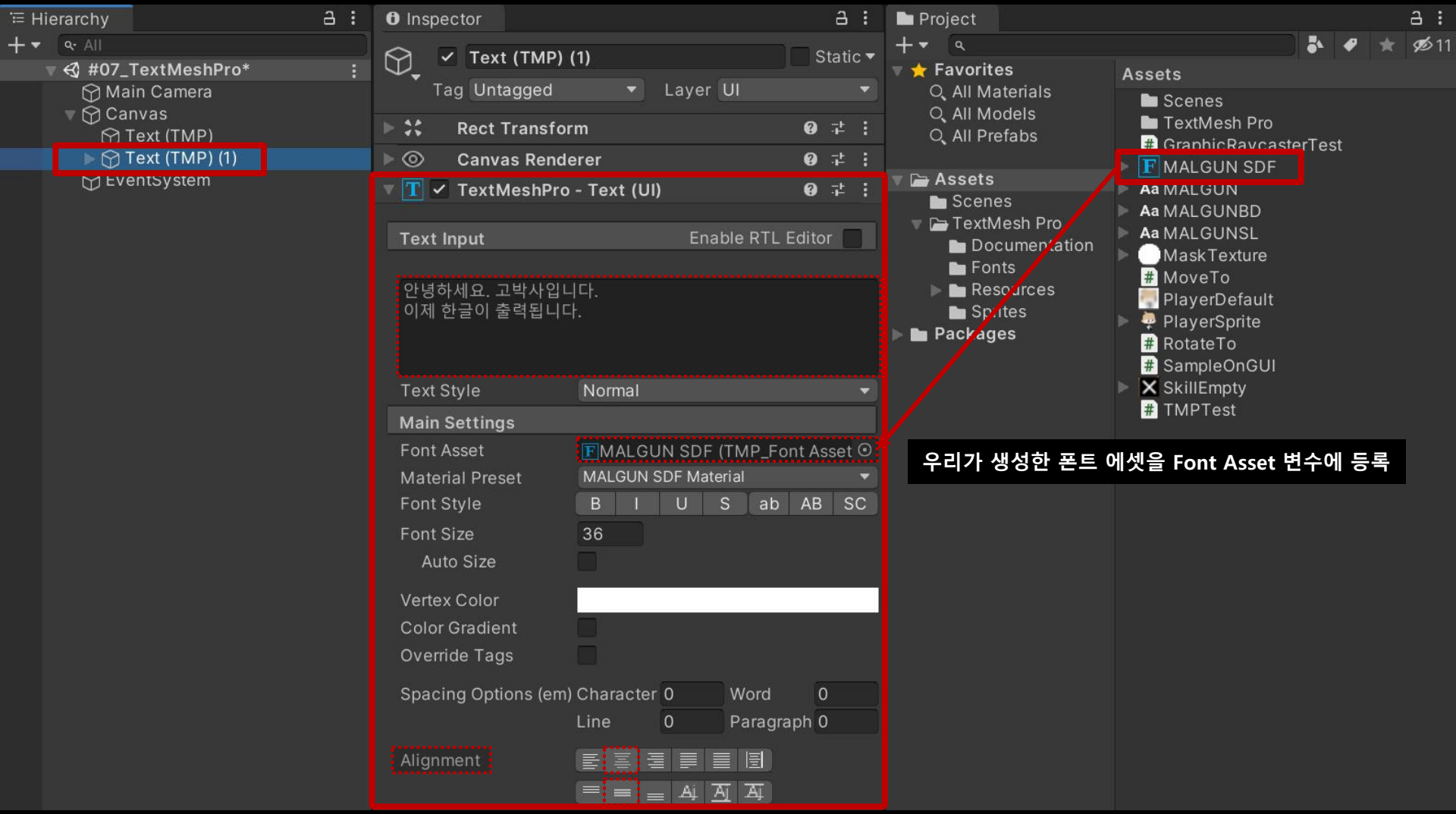
# Visual Components

- 저장된 한글 폰트를 이용해 Text - TextMeshPro 출력



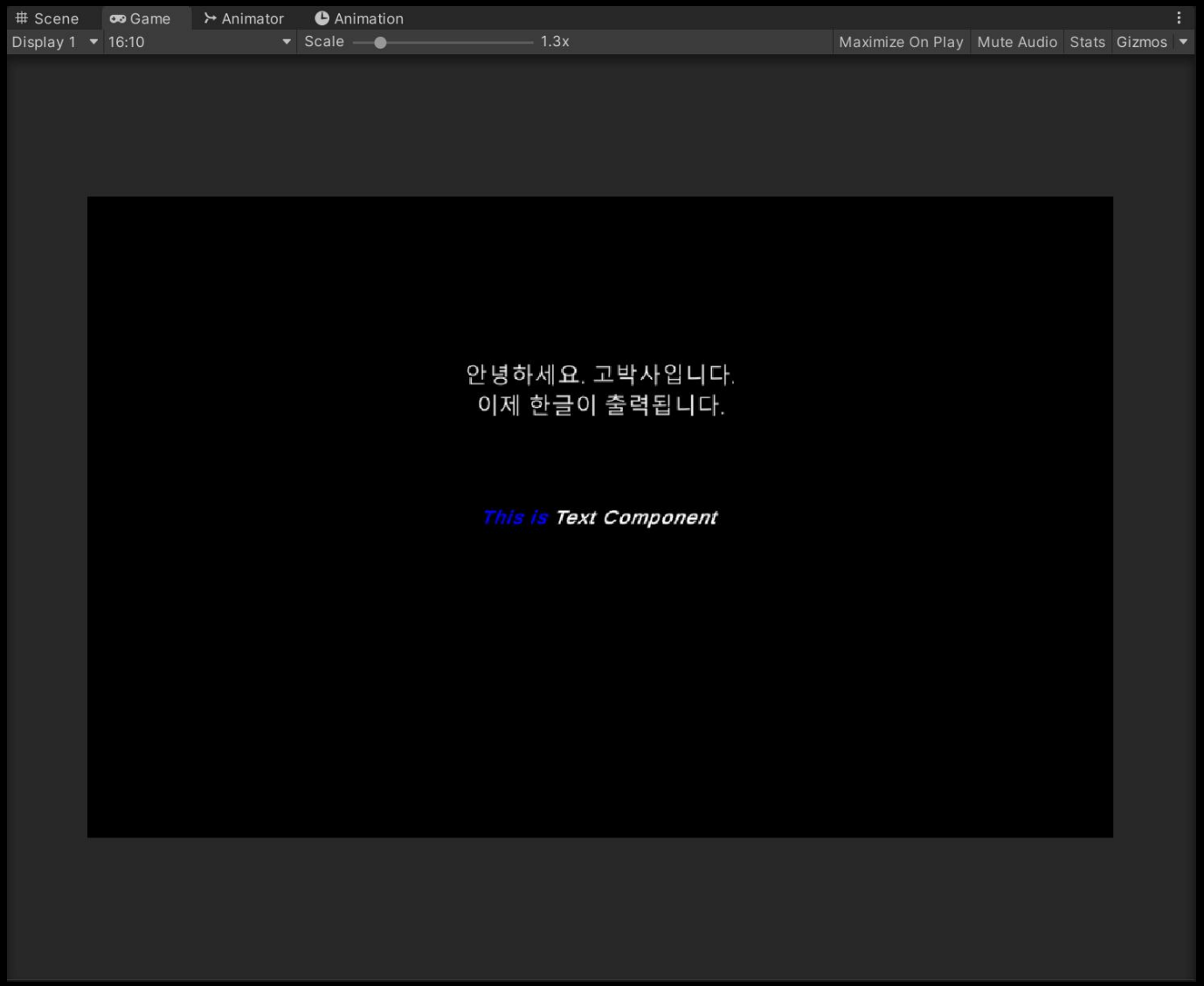
# Visual Components

- 저장된 한글 폰트를 이용해 Text - TextMeshPro 출력 (계속)



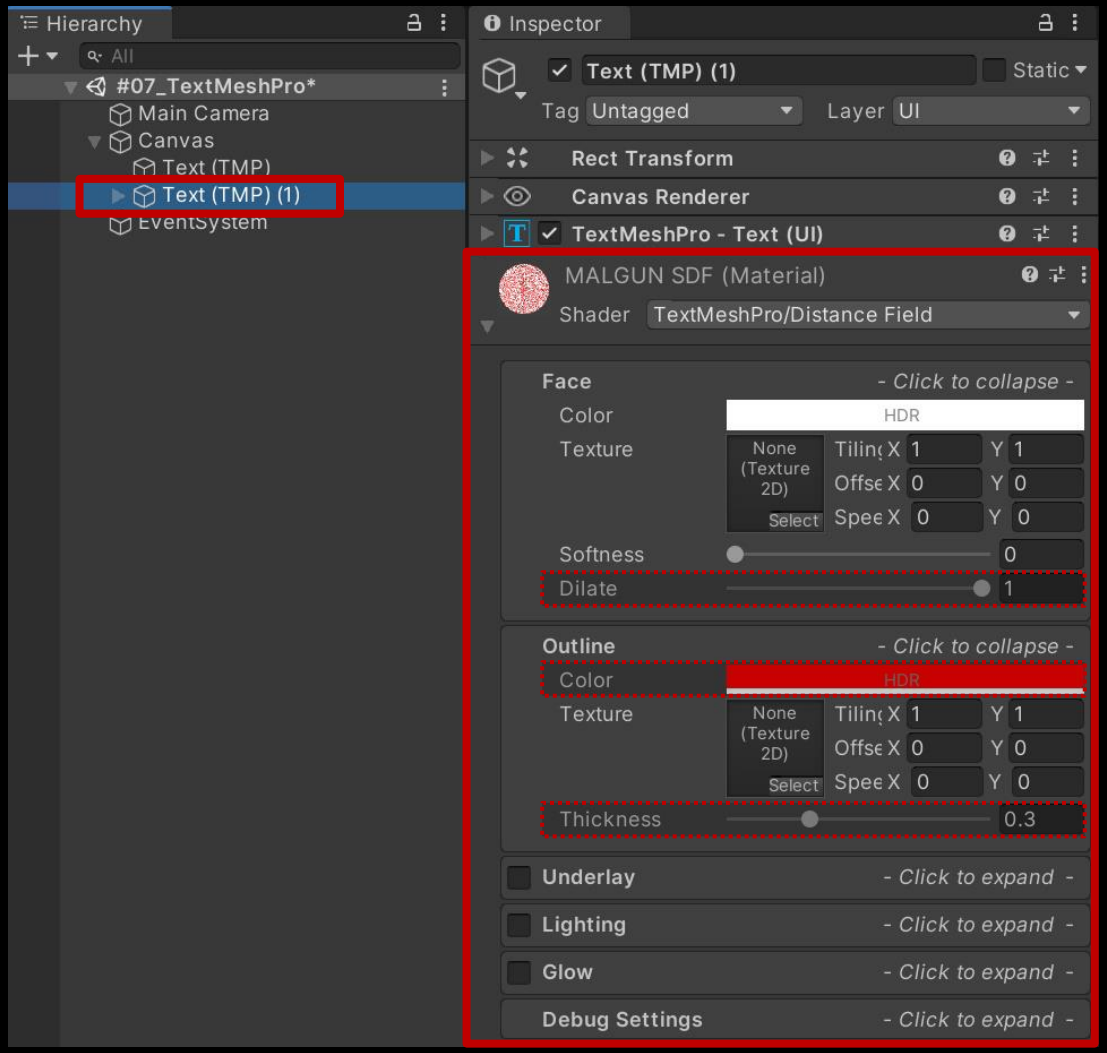
# Visual Components

- 결과 화면



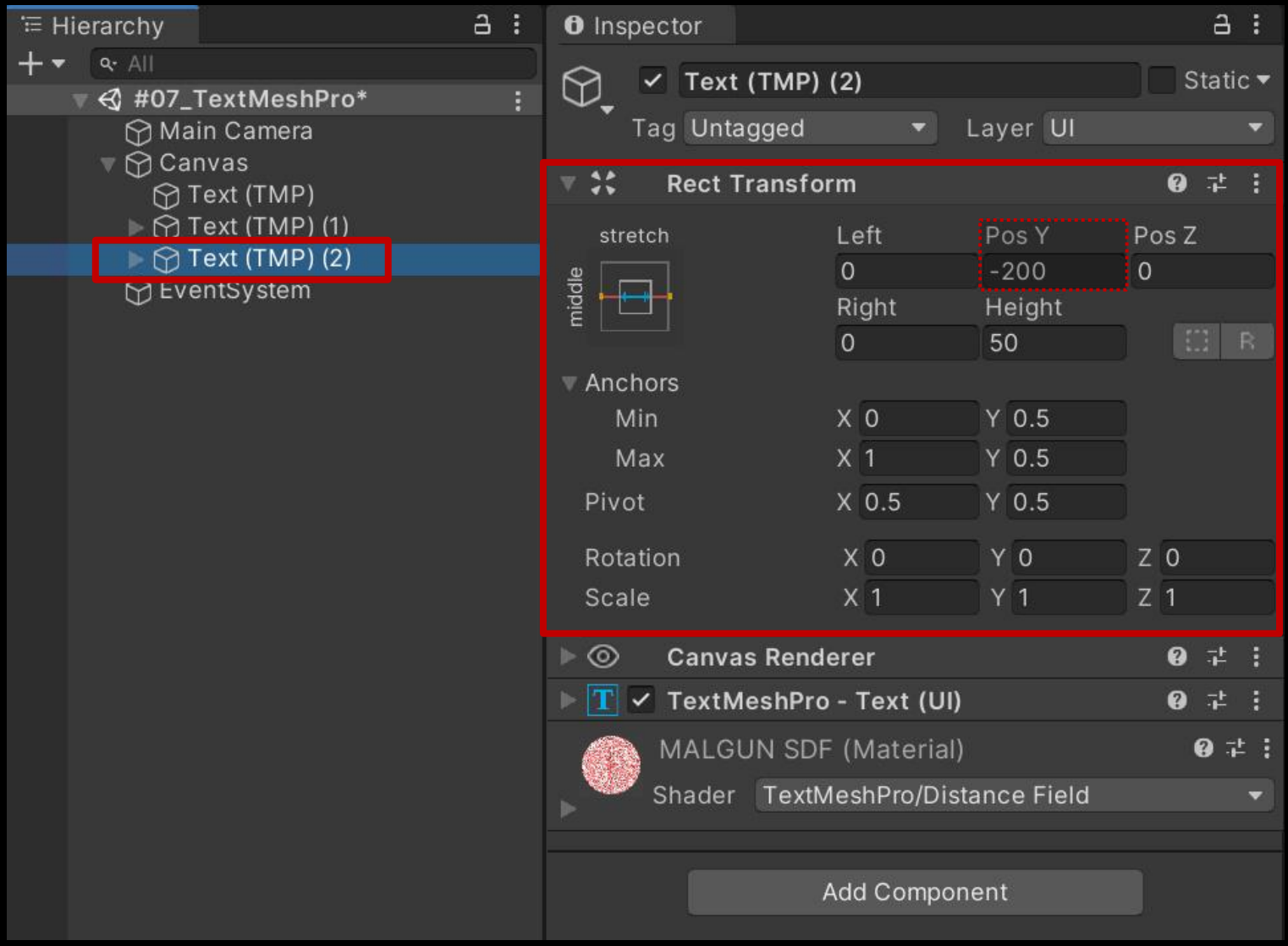
# Visual Components

- Text - TextMeshPro Material 사용
  - 폰트 에셋 Material의 Face, Outline 설정



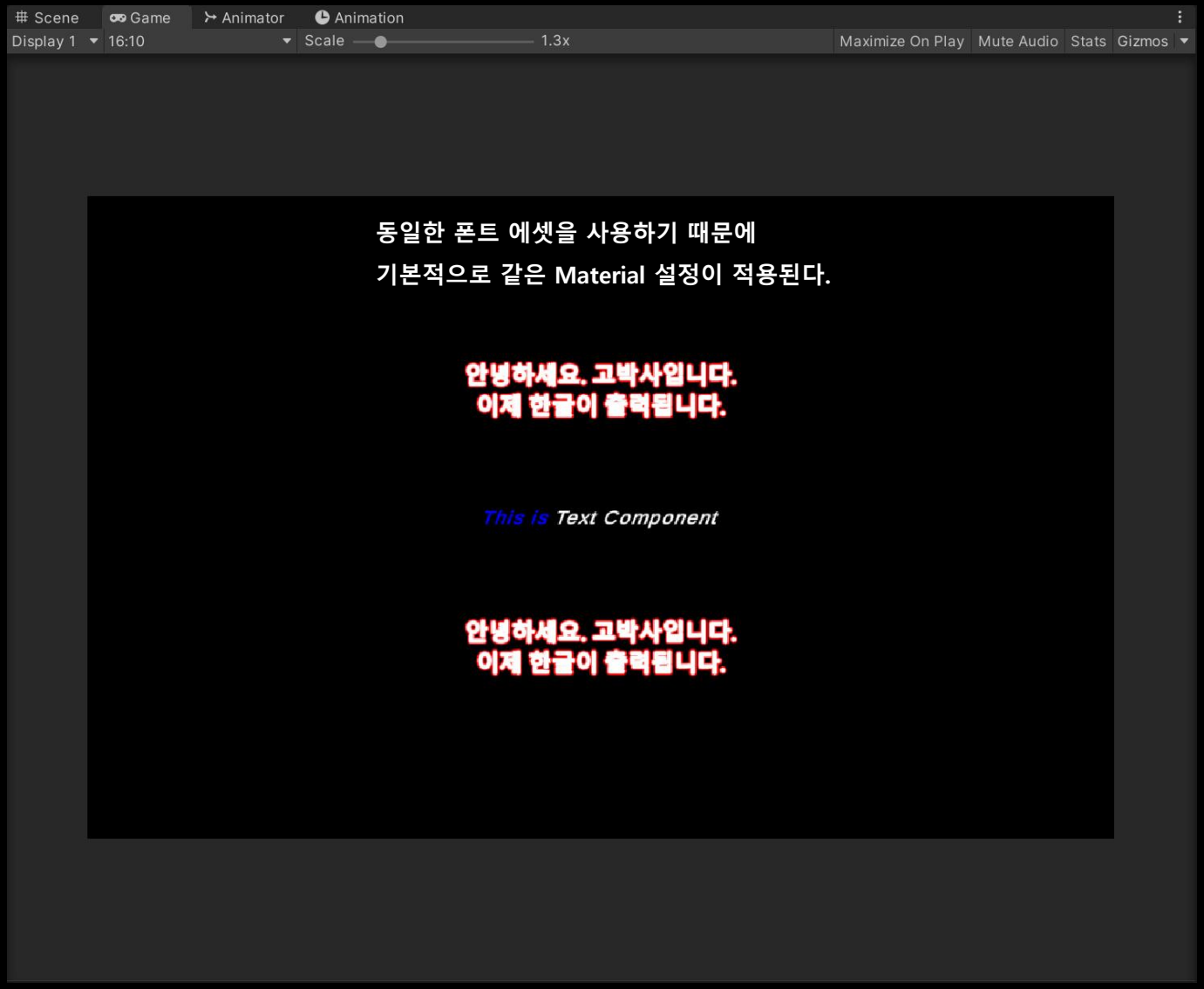
# Visual Components

- Text (TMP) (1)을 복제한 후 위치 설정



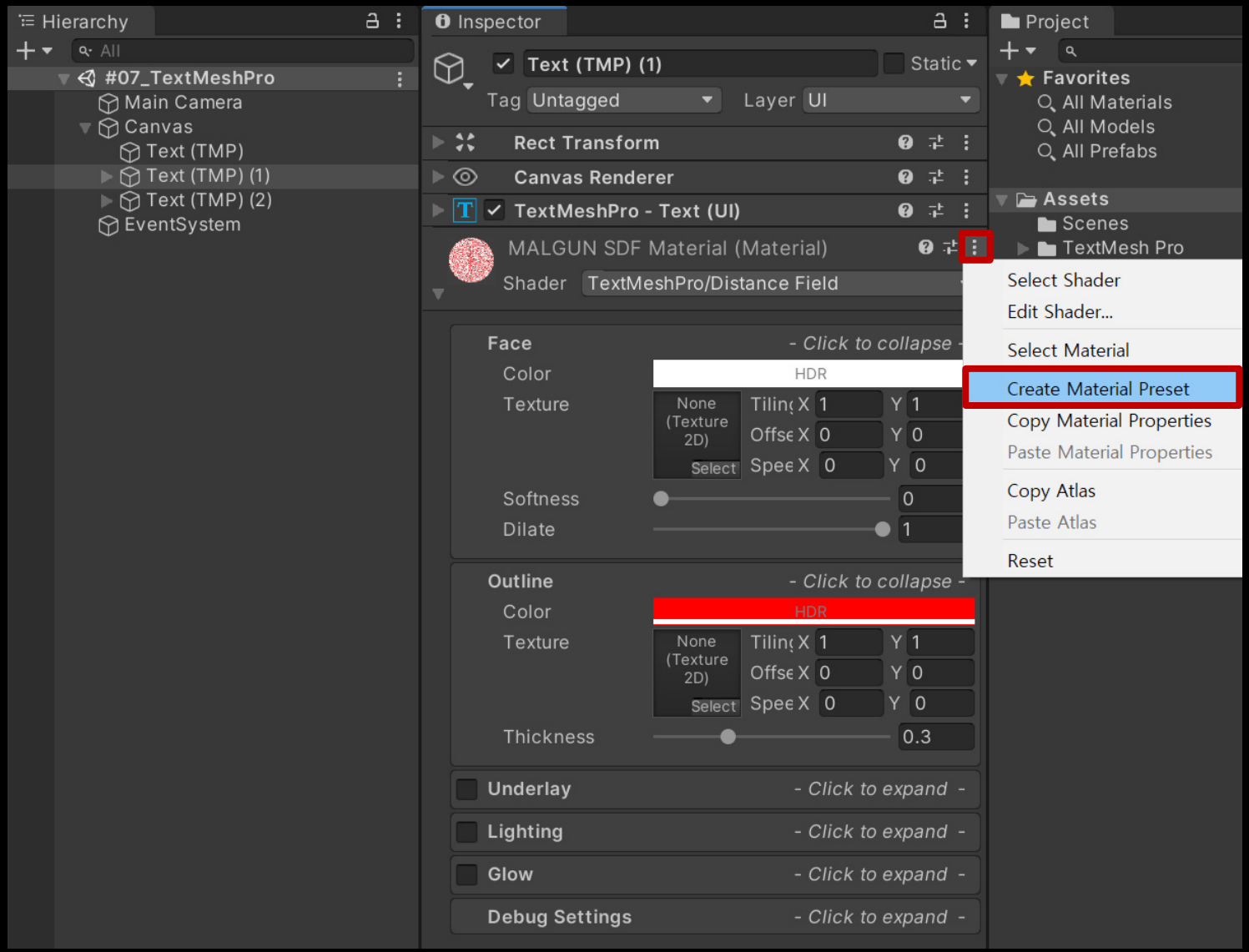
# Visual Components

- 결과 화면



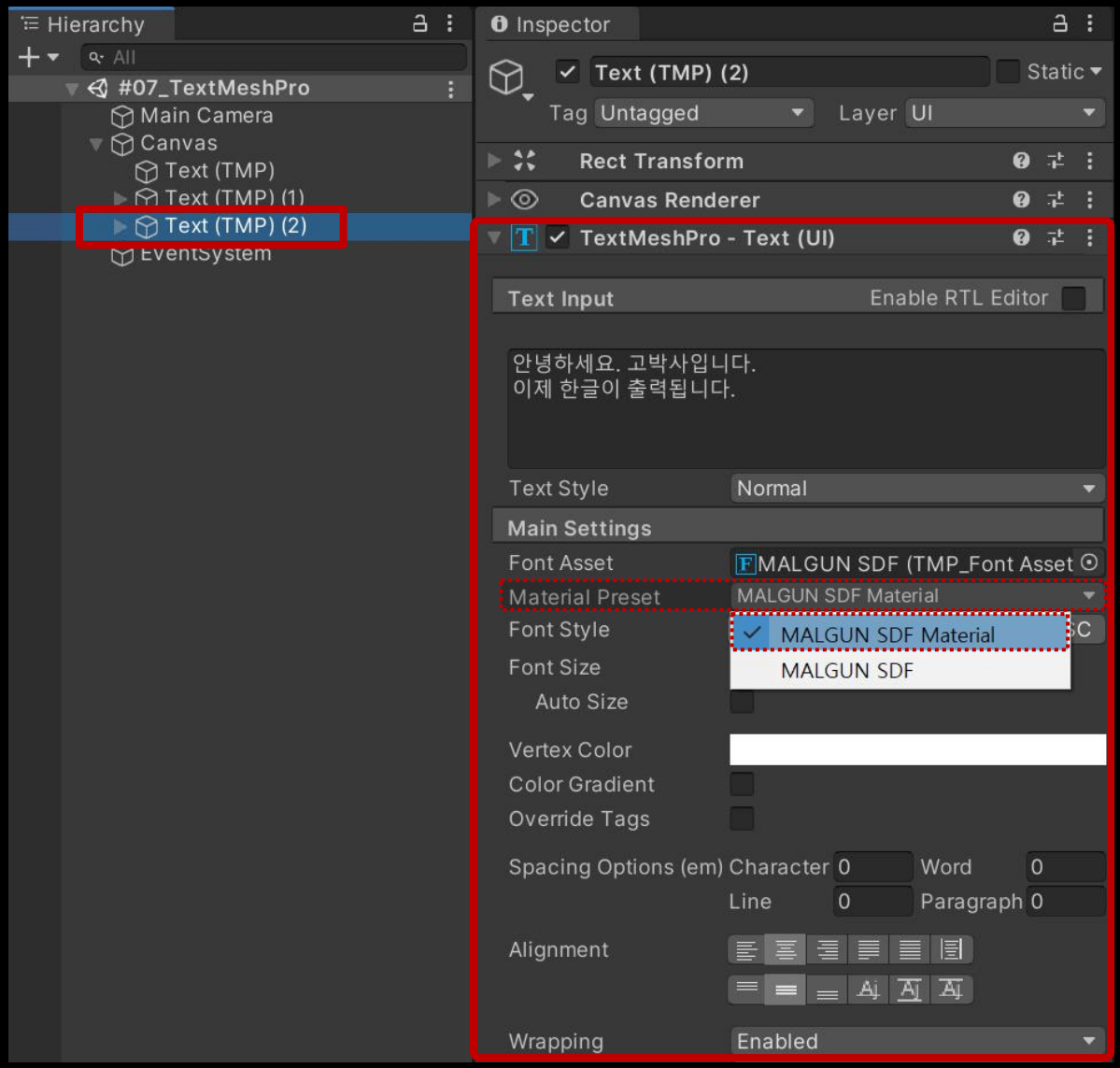
# Visual Components

- Material 메뉴 - Create Material Preset로 새로운 Material 프리셋 생성



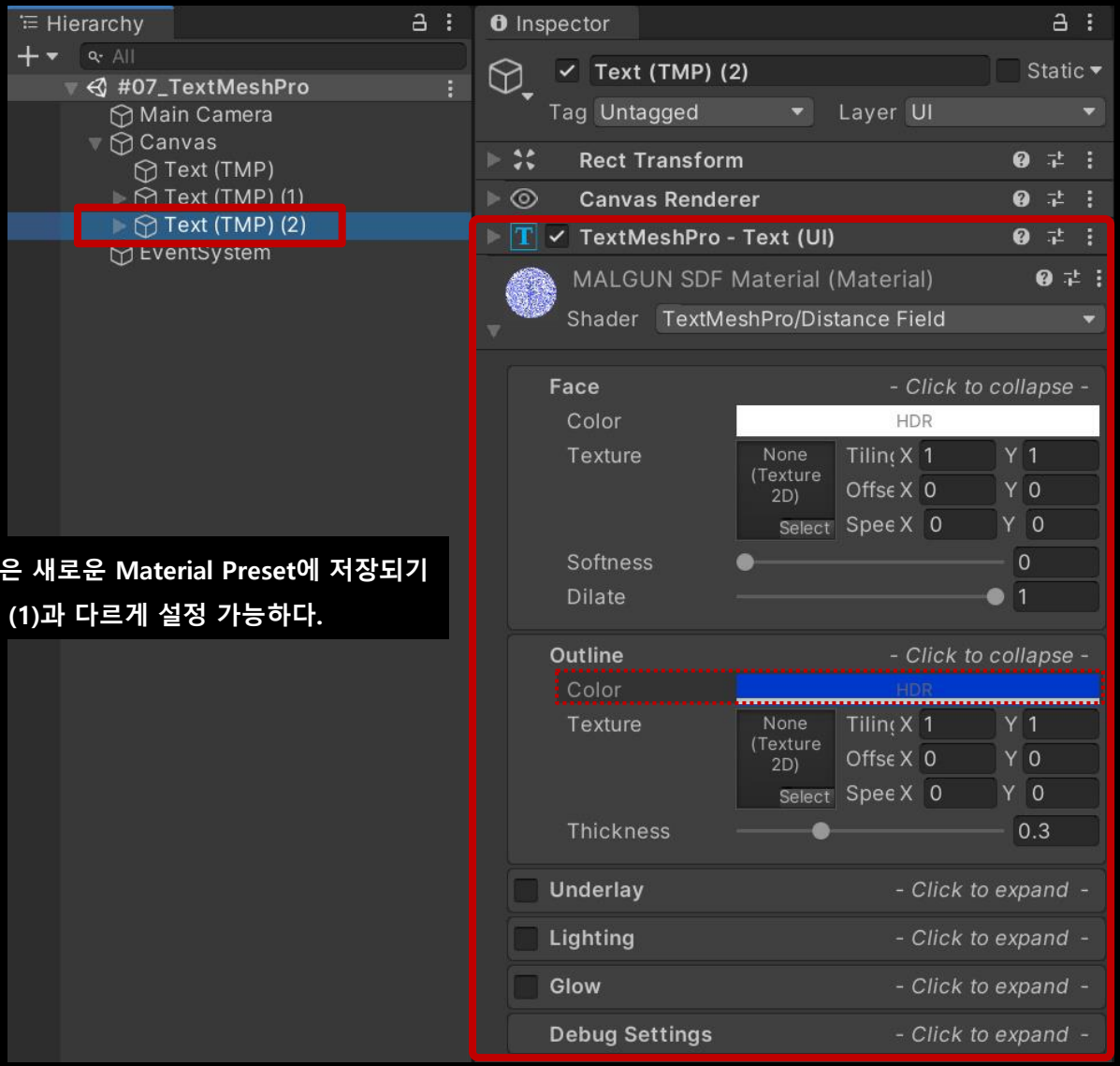
# Visual Components

- Material Preset 변수에 새로 추가한 Material 프리셋 설정



# Visual Components

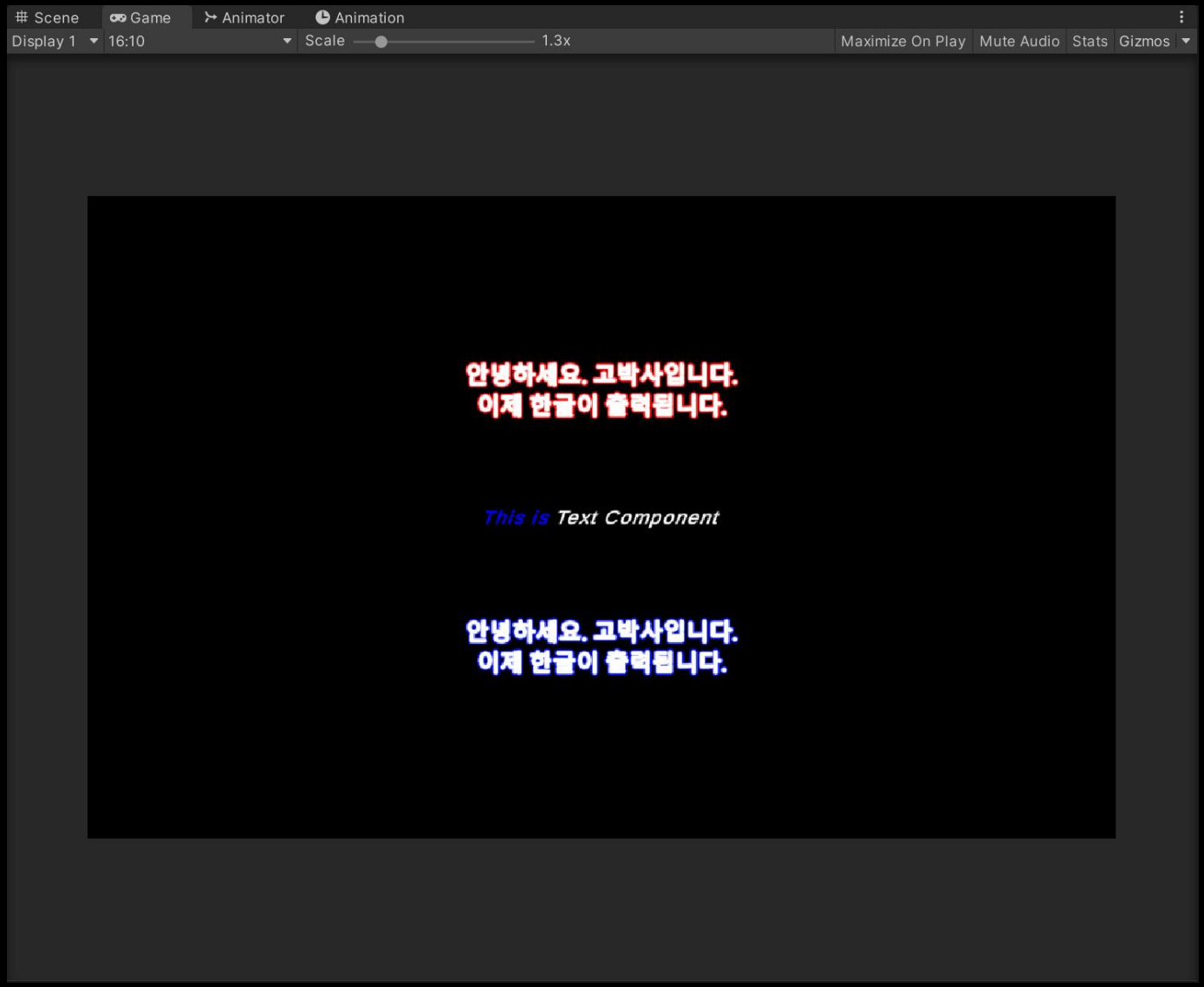
- Material Preset 변수에 새로 추가한 Material 프리셋 설정




현재 Material 설정은 새로운 Material Preset에 저장되기 때문에 Text (TMP) (1)과 다르게 설정 가능하다.

# Visual Components

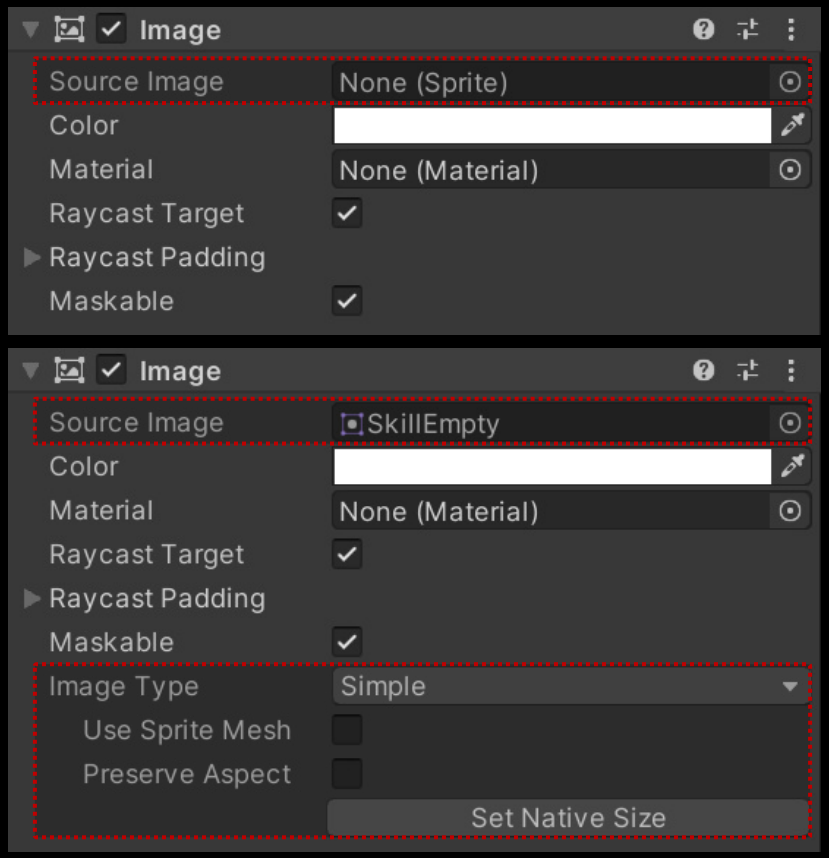
- 결과 화면



# Visual Components

## Image

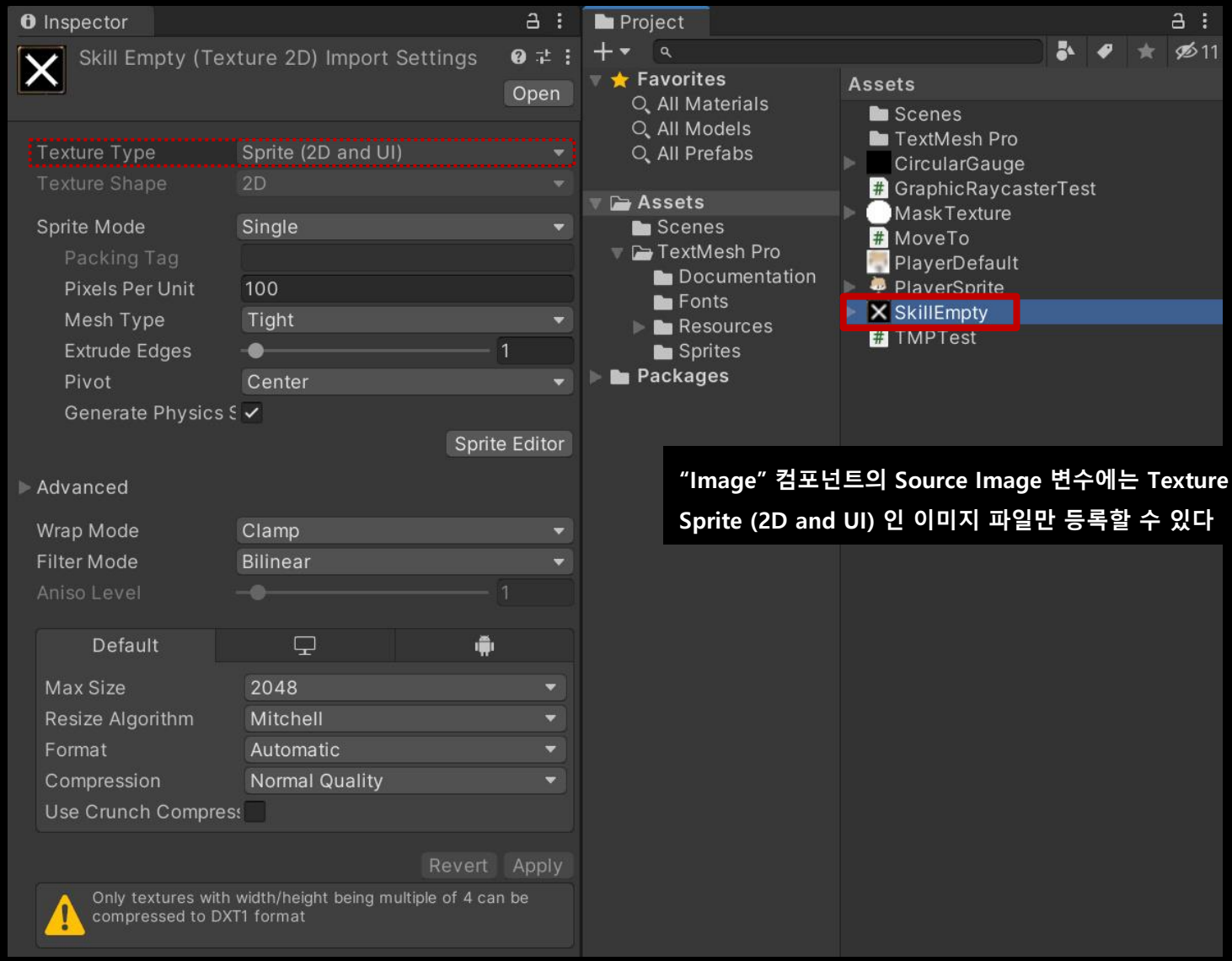
- 게임 화면에 이미지를 표시하는 UI 컴포넌트 (Sprite 2D and UI Type)
  - "Button", "Toggle", "Slider", "Scrollbar", "Dropdown", "InputField", "Scroll View"와 같이 화면에 이미지를 표시하는 모든 UI에 "Image" 컴포넌트가 포함되어 있다




**Source Image 변수에 등록되는 이미지가 게임 화면에 출력되며, 이미지가 등록되어 있어야 Image Type과 같은 추가 변수가 Inspector View에 뜬다**

# Visual Components

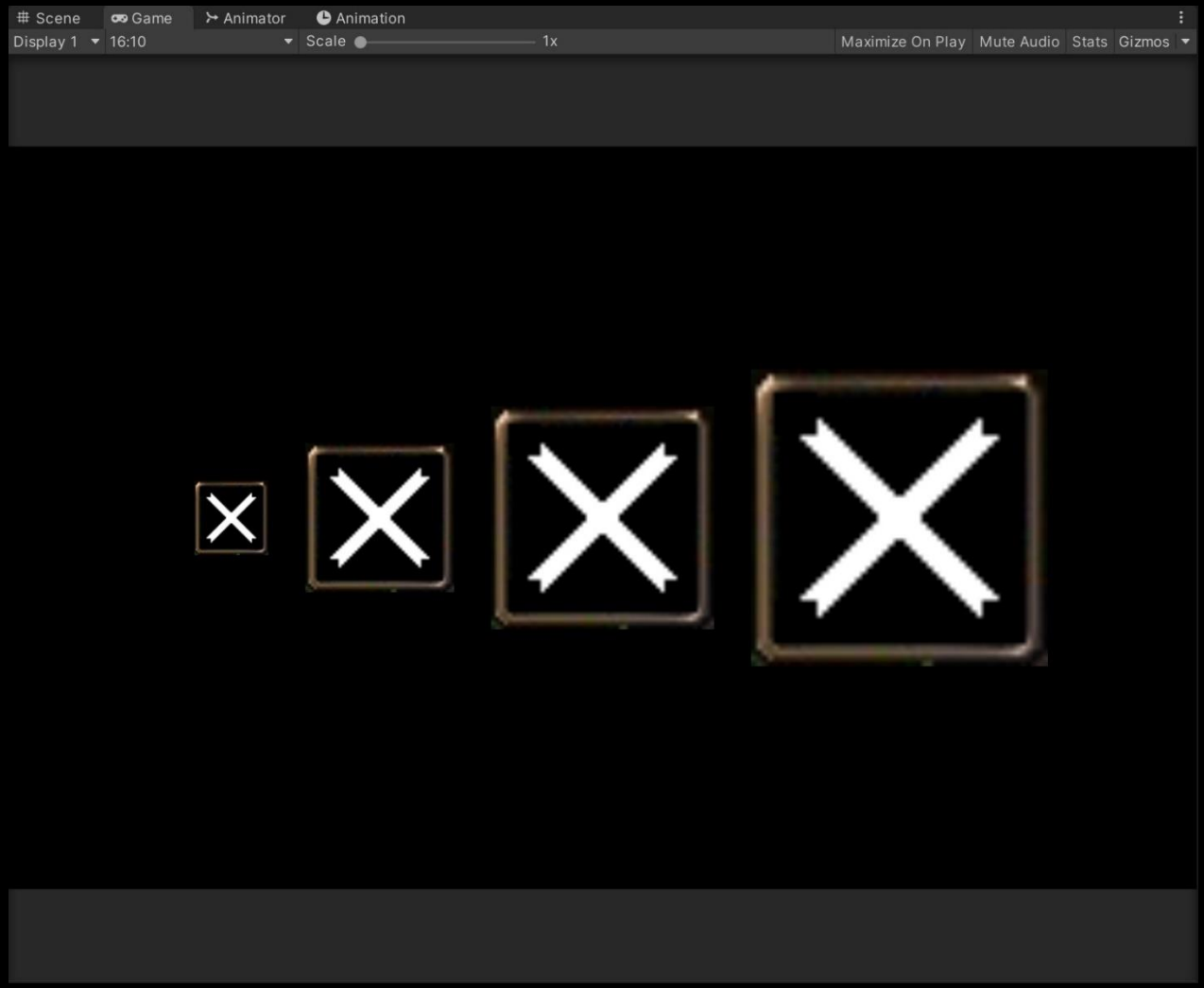
- Texture Type - Sprite (2D and UI)




"Image" 컴포넌트의 Source Image 변수에는 Texture Type이 Sprite (2D and UI) 인 이미지 파일만 등록할 수 있다

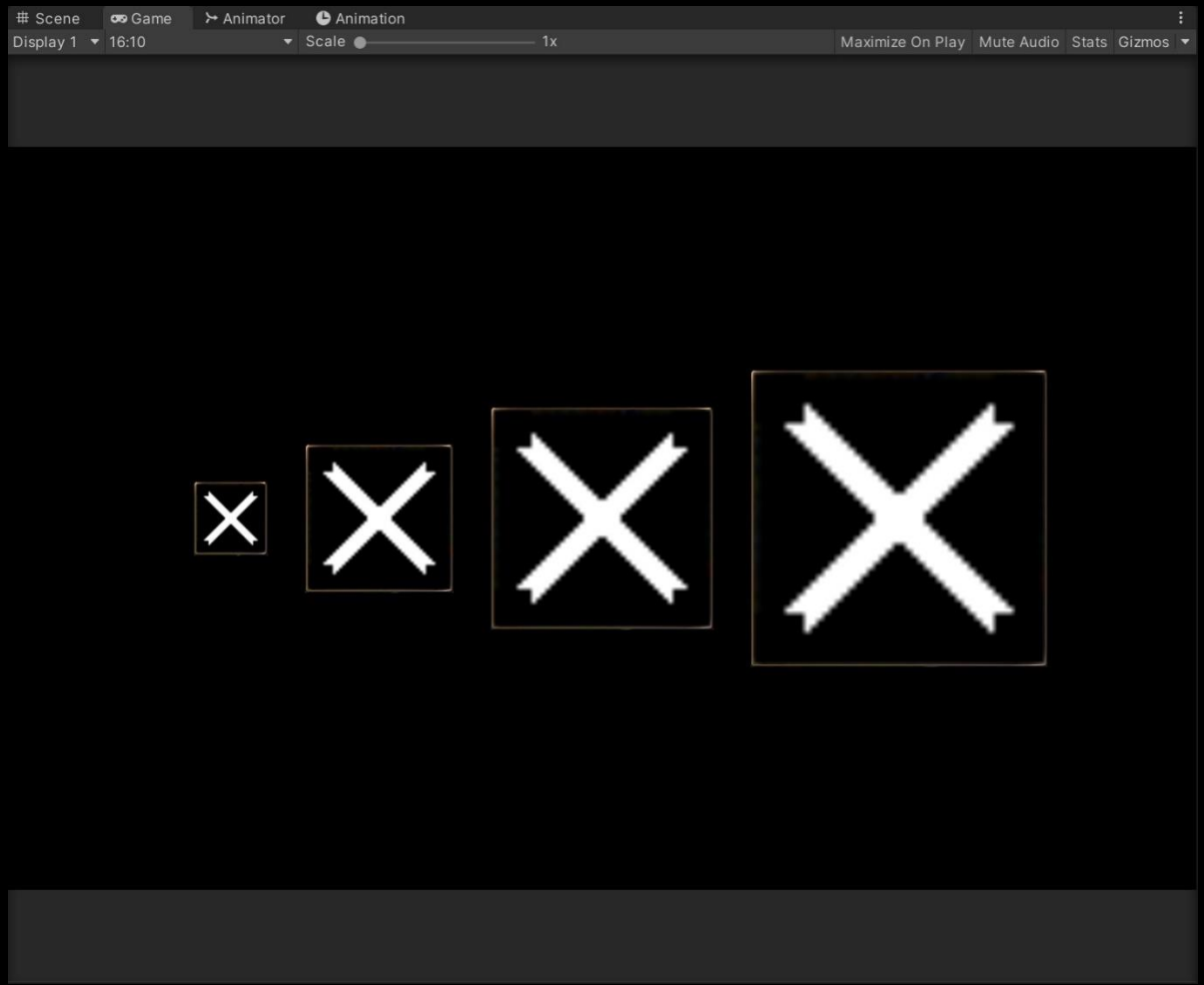
# Visual Components

- Image Type [Simple]
  - 가장 일반적으로 사용되는 타입으로 이미지의 크기를 동일하게 조정



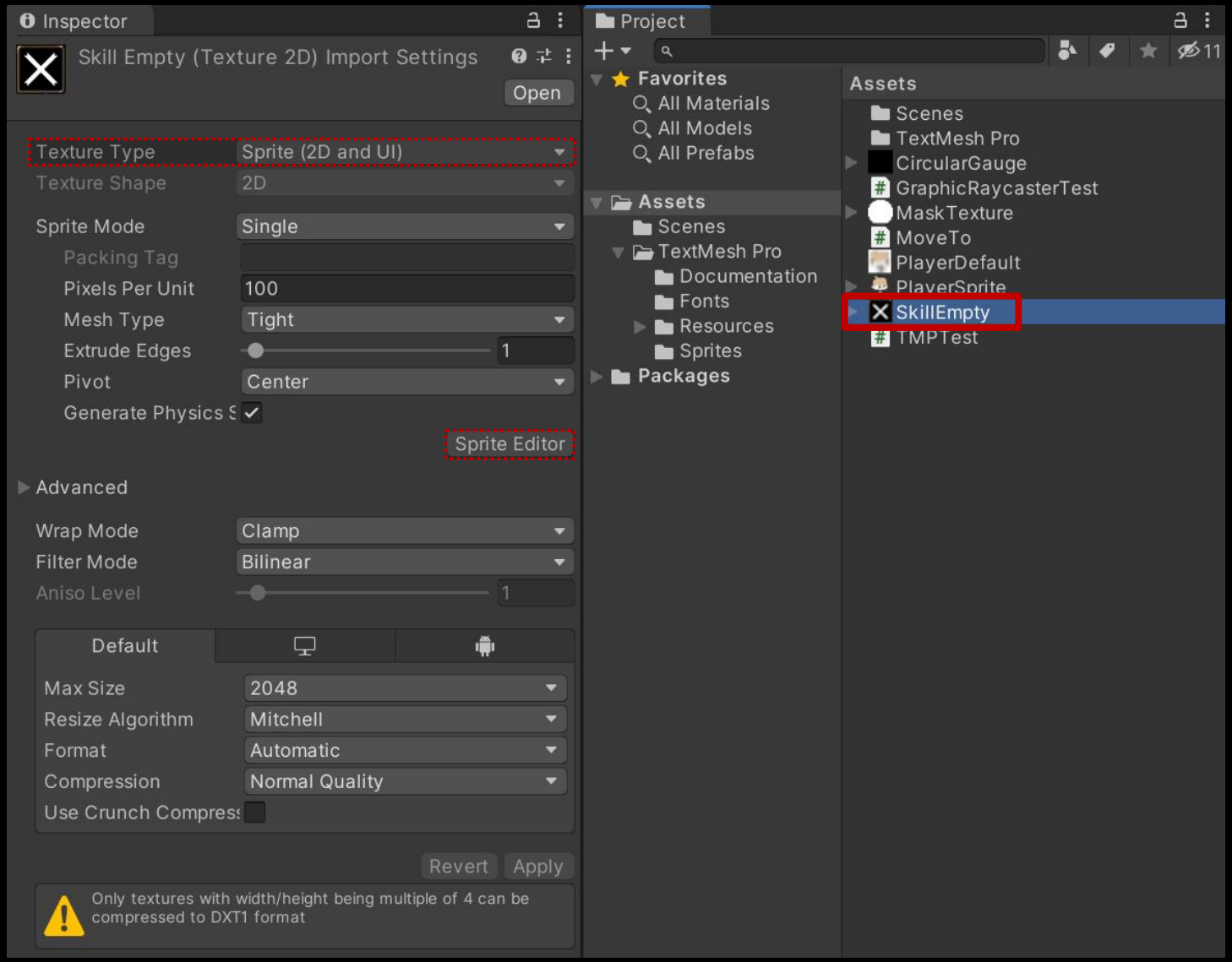
# Visual Components

- Image Type [Sliced]
  - 이미지의 크기가 바뀌었을 때 미리 설정한 모서리(3x3 Sprite Division Mode)는 크기가 그대로 유지되고, 중앙 부분만 크기가 늘어나거나 줄어든다



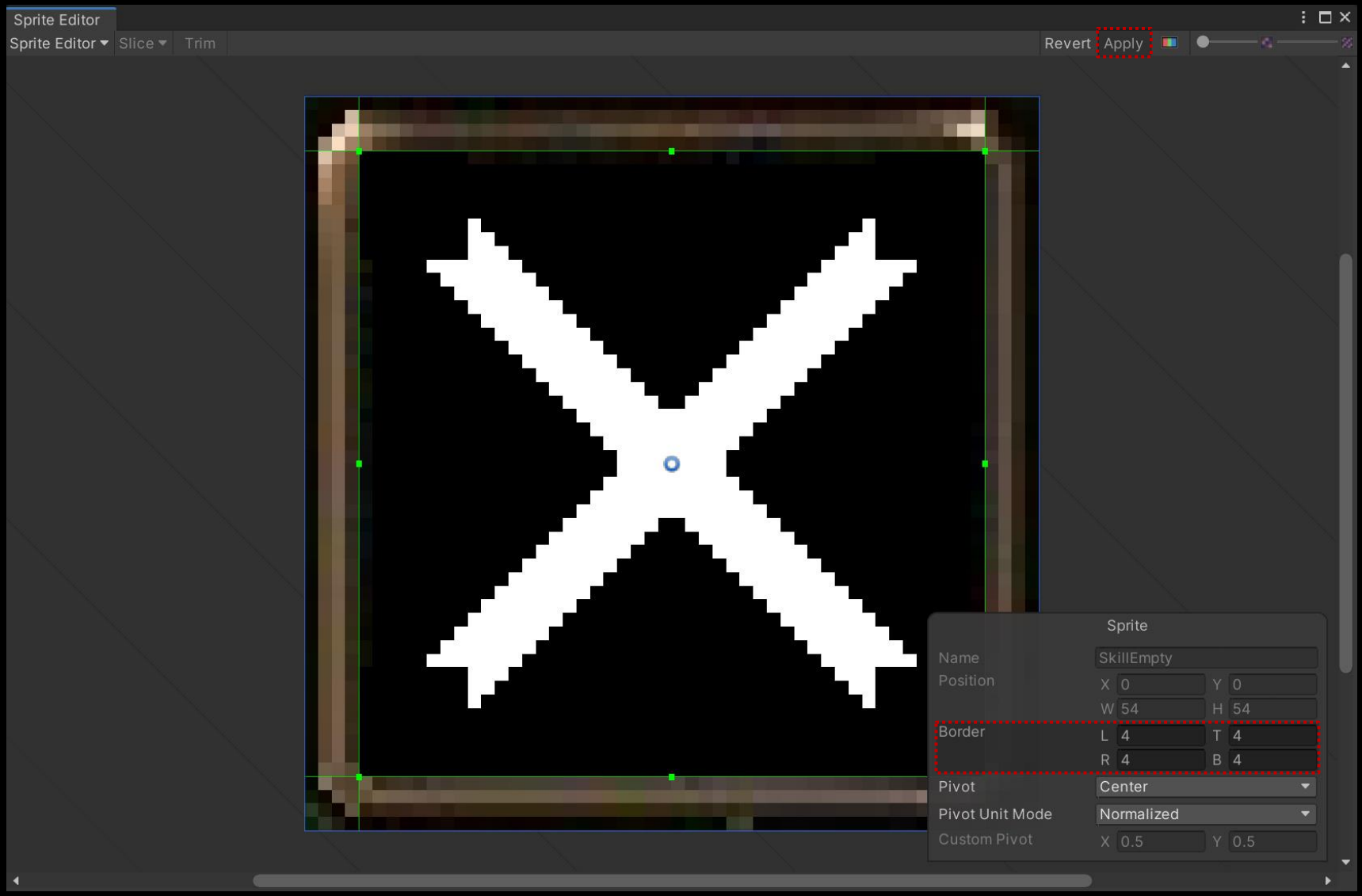
# Visual Components

- 3x3 Sprite Division Mode



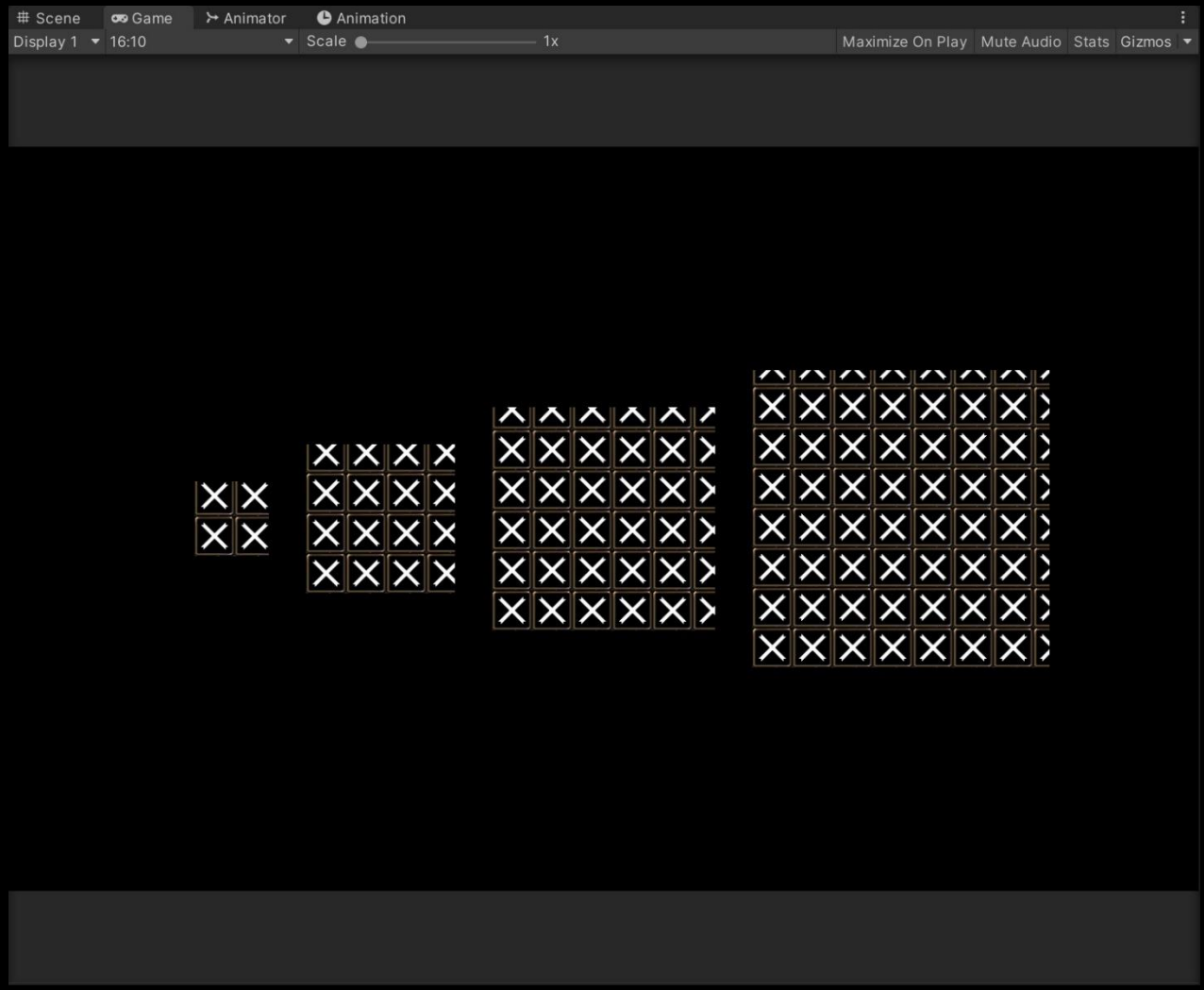
# Visual Components

- 3x3 Sprite Division Mode (계속)



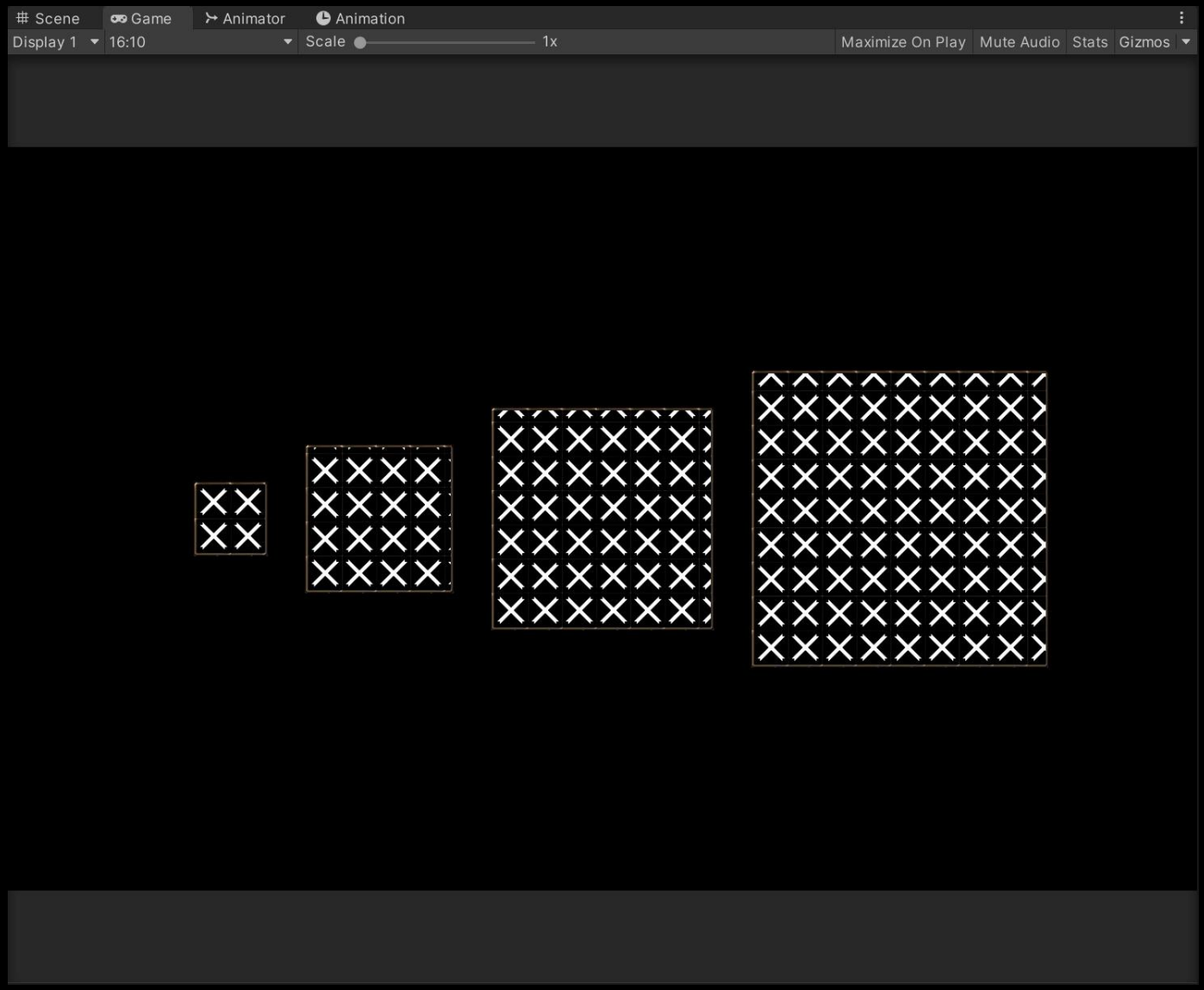
# Visual Components

- Image Type [Tiled]
  - 이미지를 바둑판 형태로 배열하여 출력



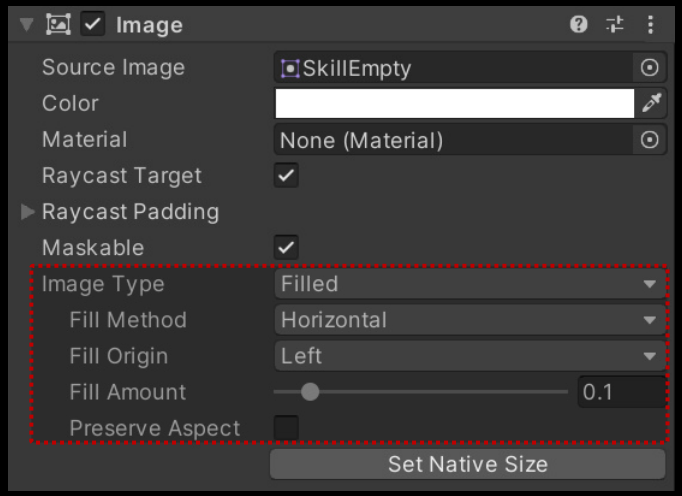
# Visual Components

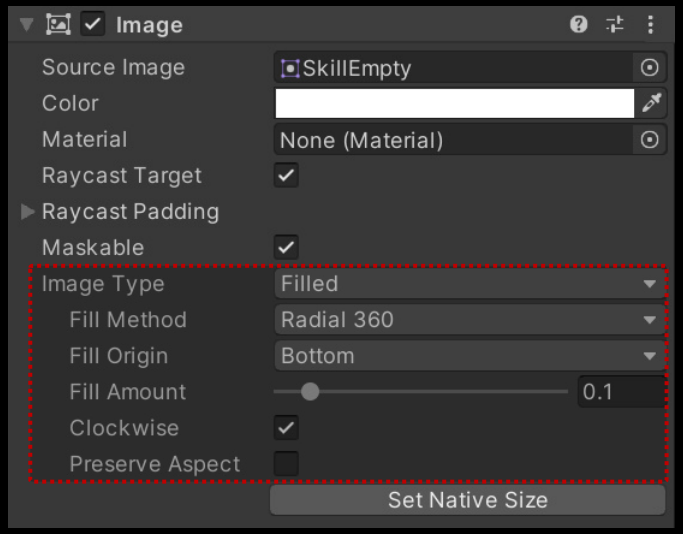
- 3x3 Sprite Division Mode가 설정되어 있으면 중앙 부분만 바둑판 형태로 출력



# Visual Components

- Image Type [Filled]
  - 이미지를 원하는 방향, 원하는 형태로 채워서 출력

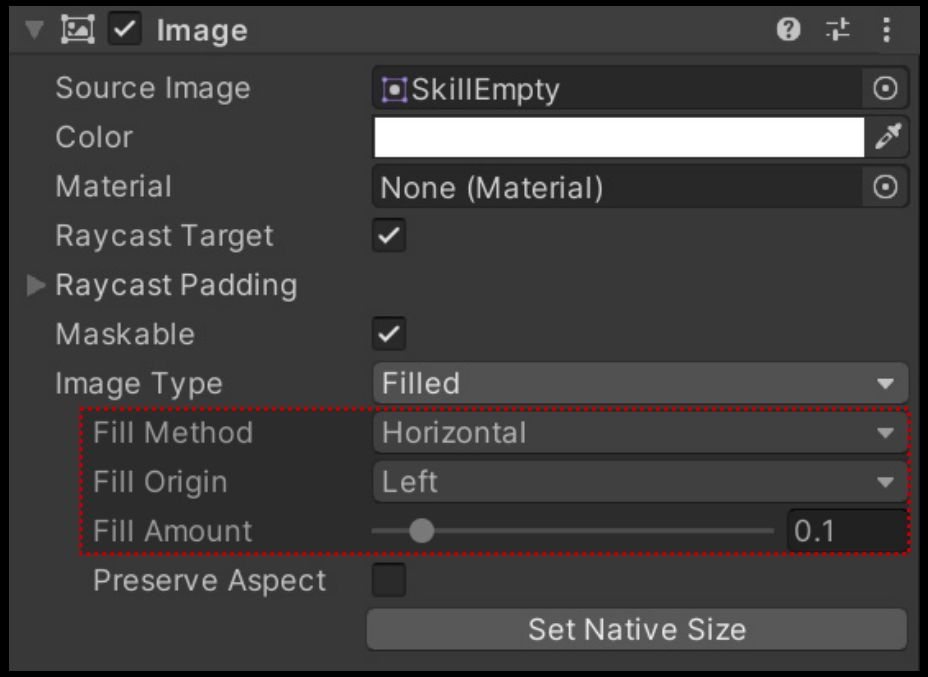



  - Filled Mode Variables
    - Fill Method : 채워지는 방식
    - Fill Origin : 채워지는 시작 방향
    - Fill Amount : 채워진 양
    - Clockwise : 채워지는 방향 (시계 방향, 반 시계 방향)
      - Fill Method가 Radial 일 때만 사용

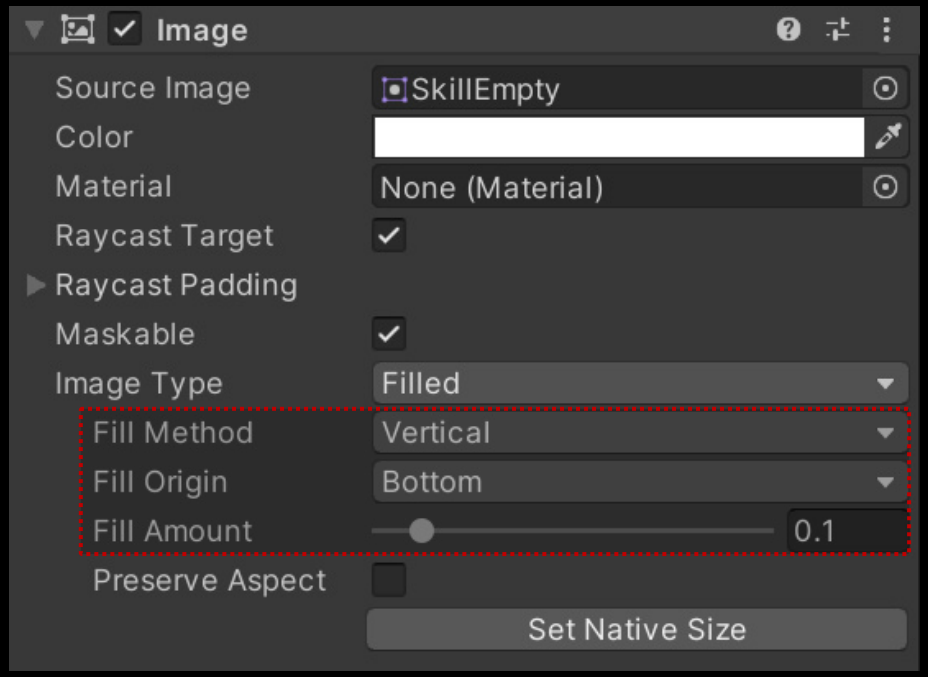
# Visual Components

- Fill Method : Horizontal
  - 가로 방향으로 이미지가 채워진다
  - Fill Origin : Right, Left



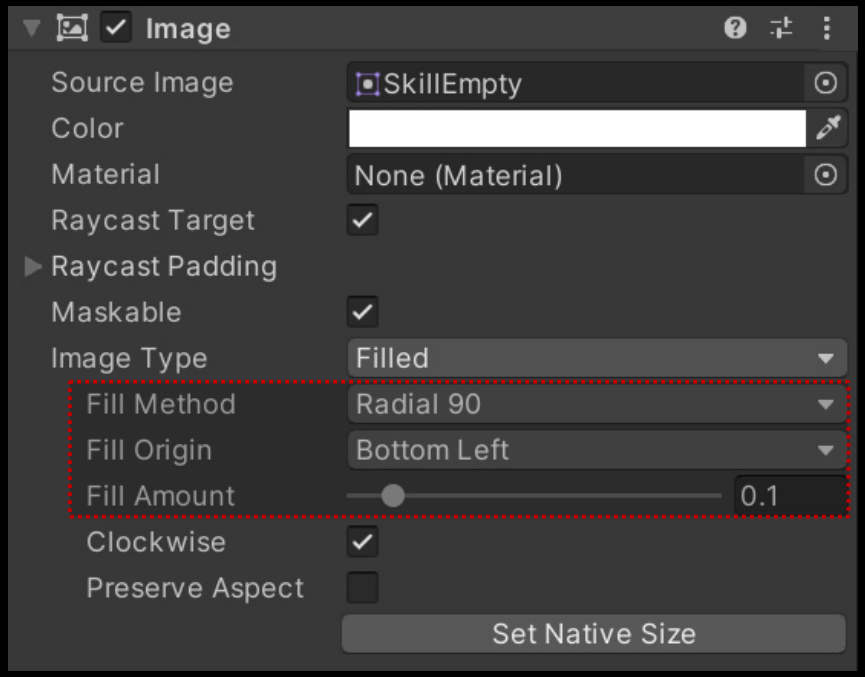

# Visual Components

- Fill Method : Vertical
  - 세로 방향으로 이미지가 채워진다
  - Fill Origin : Bottom, Top




# Visual Components

- Fill Method : Radial 90
  - 이미지의 네 모서리 중 하나를 기준으로 0-90도로 이미지가 채워진다
  - Fill Origin : Bottom Left, Top Left, Bottom Right, Top Right



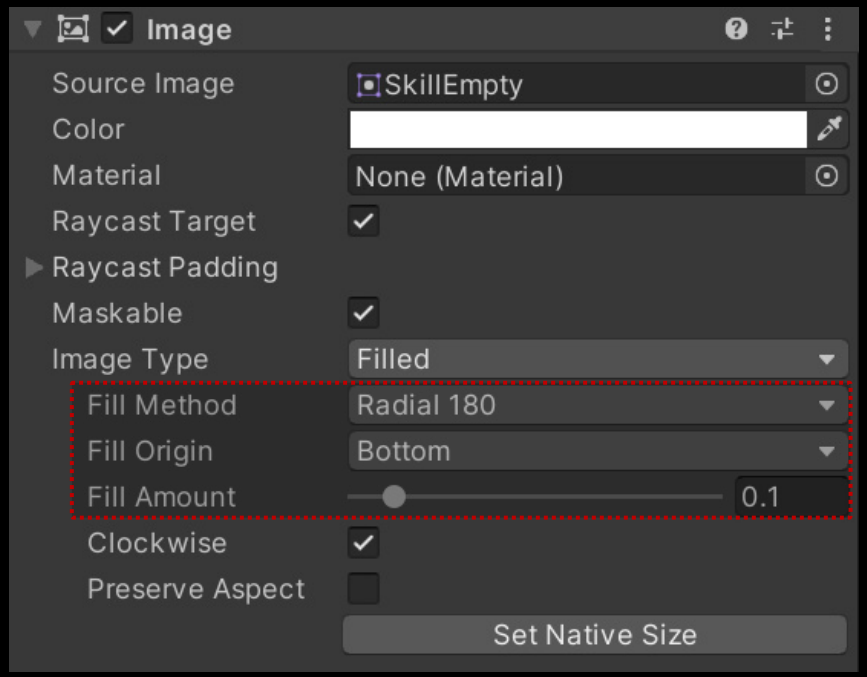

**Tip. Fill Method가 Radial 일 때**
**회전 방향 (Clockwise) 옵션이 생긴다**

**Clockwise = true**
**(시계 방향)**

**Clockwise = false**
**(반 시계 방향)**

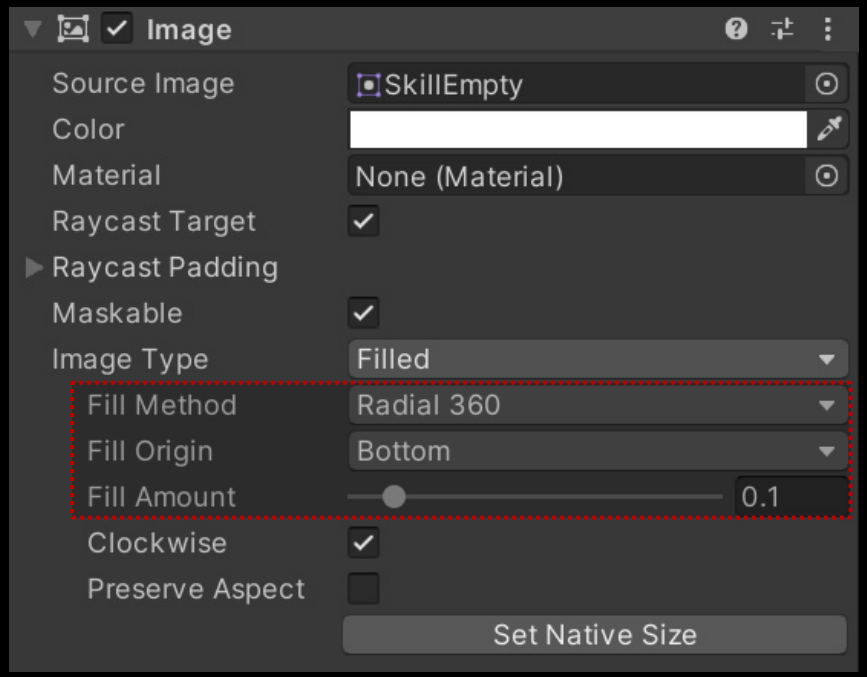
# Visual Components

- Fill Method : Radial 180
  - 이미지의 네 변의 중점 중 하나를 기준으로 0-180도로 이미지가 채워진다
  - Fill Origin : Bottom, Top, Left, Right



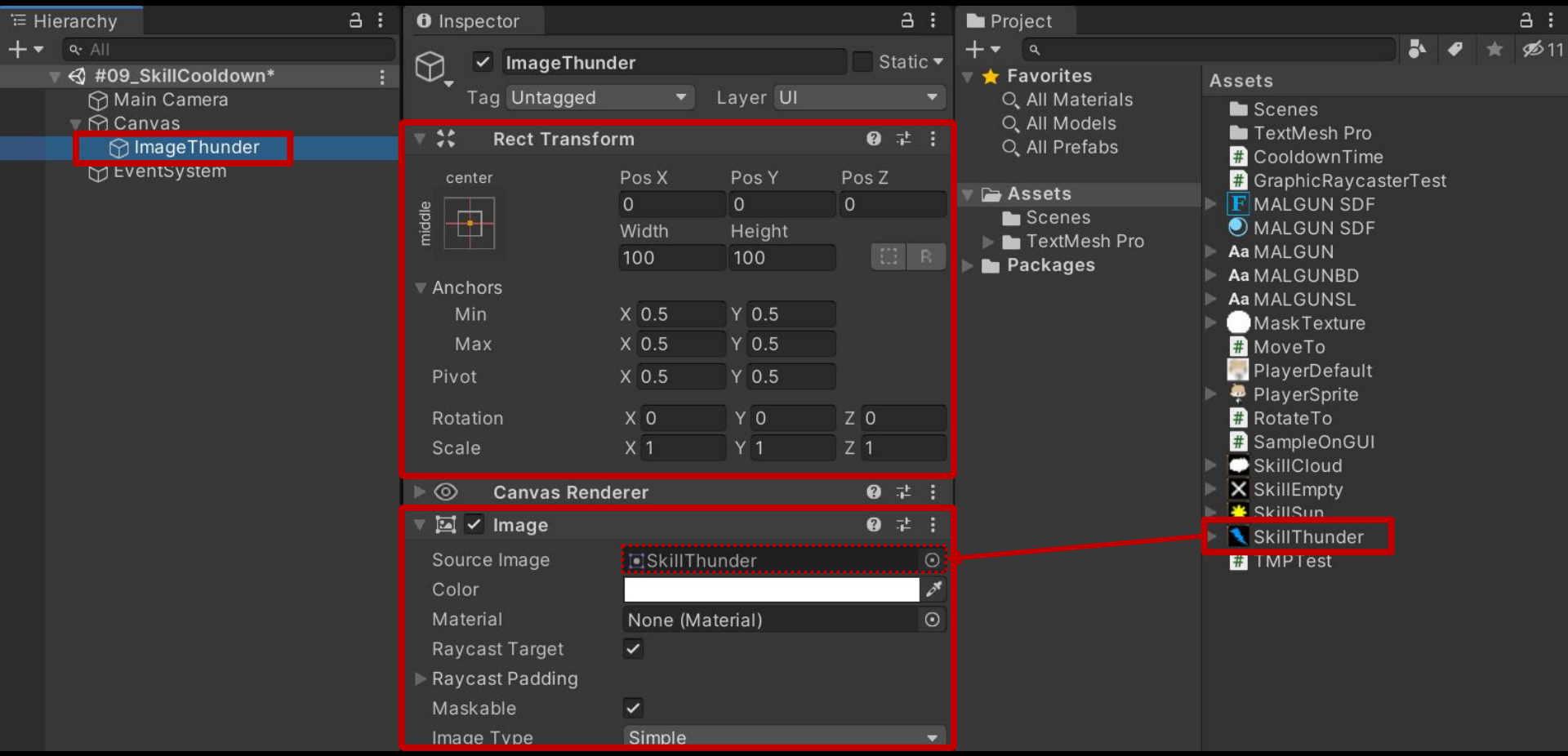

# Visual Components

- Fill Method : Radial 360
  - 이미지의 네 변의 중점 중 하나를 기준으로 0-360도로 이미지가 채워진다
  - Fill Origin : Bottom, Top, Left, Right




# Visual Components

- Skill Cooldown
  - 스킬 이미지 표현을 위한 Image UI 생성
    - GameObject - UI - Image



# Visual Components

- 스킬 쿨타임이 적용 중일 때 쿨타임 표현을 위한 Image UI 생성
  - GameObject - UI - Image

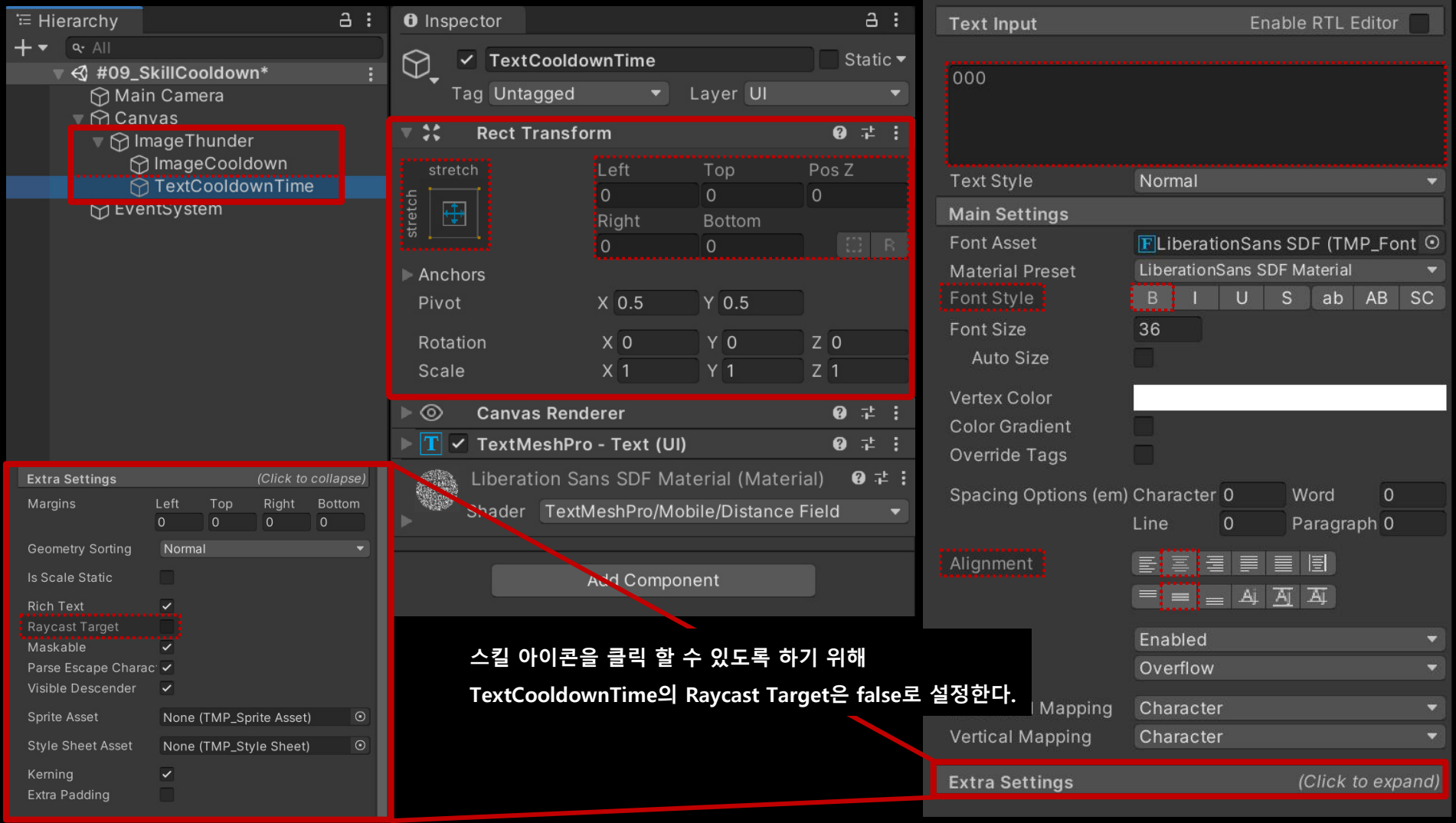
Canvas 오브젝트에 배치되는 순서에 따라 UI의 그려지는 순서가 결정된다.
쿨다운은 스킬 아이콘보다 앞에 보여야 하기 때문에 더 아래에
배치되어야 하고, 해당 스킬의 쿨다운을 표현하기 때문에 자식으로 설정

Anchors를 전체로 선택한 후 스킬 아이콘의 테두리만큼
여백을 남기고 스킬 아이콘 내부를 채워준다.

스킬 아이콘을 클릭 할 수 있도록 하기 위해
ImageCooldown의 Raycast Target은 false로 설정한다.

# Visual Components

- 스킬 쿨타임 시간 표현을 위한 Text - TextMeshPro UI 생성
  - GameObject - UI - "Text - TextMeshPro"

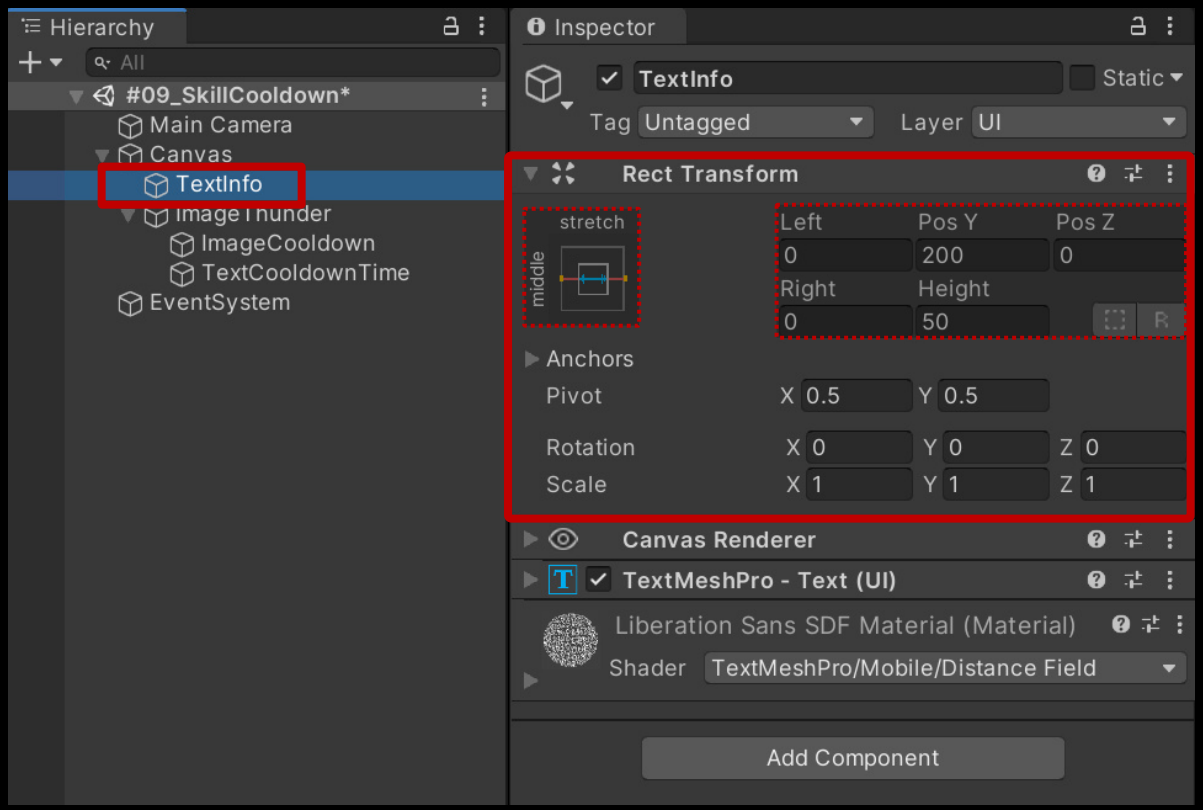


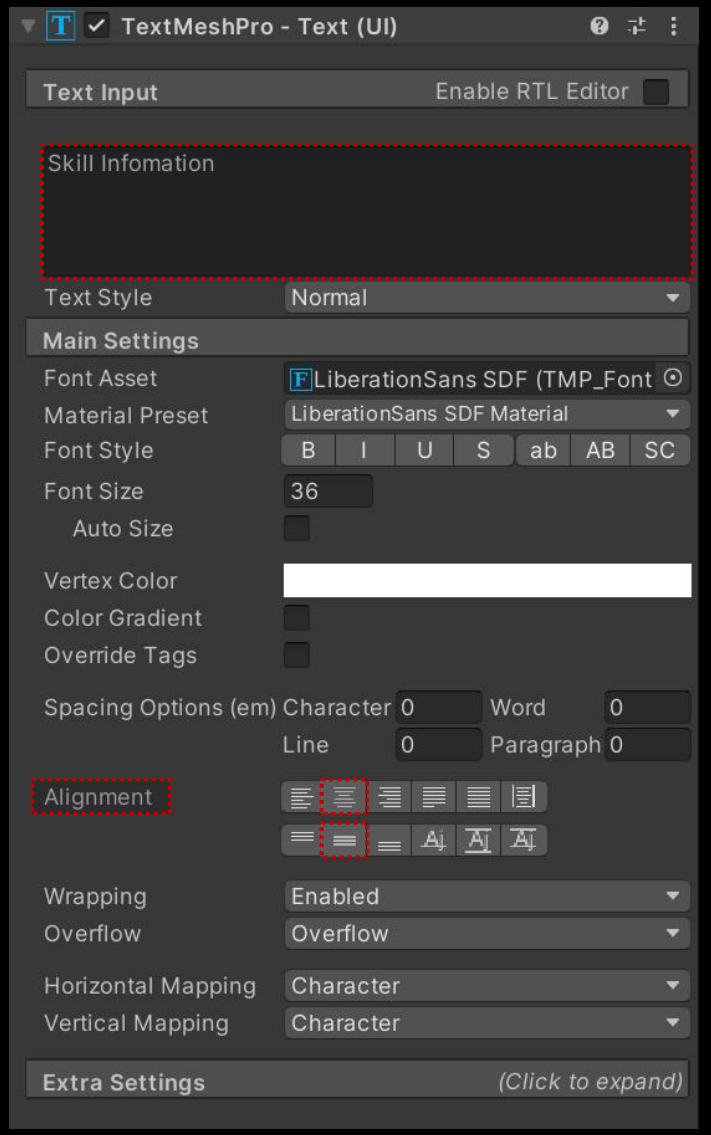

스킬 아이콘을 클릭 할 수 있도록 하기 위해

**TextCooldownTime의 Raycast Target은 false로 설정한다.**

# Visual Components

- 스킬 사용 정보 표현을 위한 Text - TextMeshPro UI 생성
  - GameObject - UI - "Text - TextMeshPro"



# Visual Components

- 스킬을 사용했을 때 쿨타임을 제어하는 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "CooldownTime"으로 변경
  - 스크립트를 "ImageThunder" 오브젝트의 컴포넌트로 적용

```
1  using System.Collections;
2  using UnityEngine;
3  using UnityEngine.UI;
4  using TMPro;
5
6  public class CooldownTime : MonoBehaviour
7  {
8      [SerializeField]
9      private TextMeshProUGUI textInfo;              // 스킬 시전 정보 출력
10     [SerializeField]
11     private TextMeshProUGUI textCooldownTime;      // 쿨다운 시간을 표시하는 Text UI
12     [SerializeField]
13     private Image           imageCooldownTime;     // 쿨다운 시간을 나타내는 Image UI
14     [SerializeField]
15     private string          skillName;             // 스킬 이름
16     [SerializeField]
17     private float           maxCooldownTime;       // 최대 쿨타임 시간
18
19     private float           currentCooldownTime;   // 현재 쿨타임 시간
20     private bool            isCooldown;            // 현재 쿨타임이 적용중인지 체크
21
22     private void Awake()
23     {
24         SetCooldownIs(false);
25     }
26
```

# Visual Components

- 스킬을 사용했을 때 쿨타임을 제어하는 스크립트 생성 및 작성 (계속)

```
27      /// <summary>
28      /// 외부에서 스킬 사용 후 쿨타임을 적용할 때 호출하는 메소드
29      /// </summary>
30      public void StartCooldownTime()
31      {
32          // 이미 스킬을 사용해서 쿨타임이 남아있으면 종료
33          if ( isCooldown == true )
34          {
35              textInfo.text = $"{skillName} CooldownTime : {currentCooldownTime.ToString("F1")}";
36              return;
37          }
38
39          textInfo.text = $"Use Skill : {skillName}";
40
41          StartCoroutine("OnCooldownTime", maxCooldownTime);
42      }
43
```

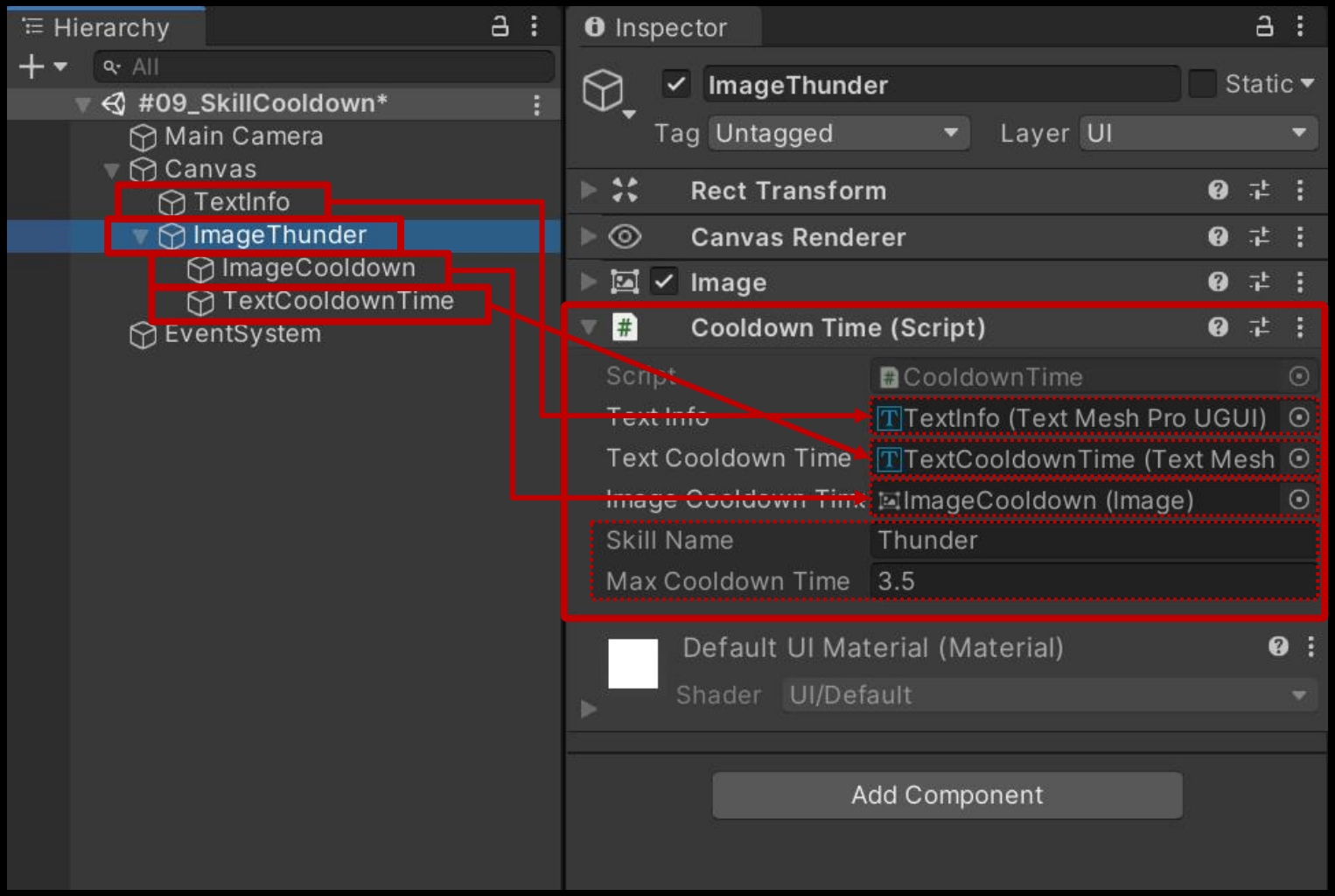
# Visual Components

- 스킬을 사용했을 때 쿨타임을 제어하는 스크립트 생성 및 작성 (계속)

```
44      /// <summary>
45      /// 실제 스킬의 쿨다운 타임을 제어하는 코루틴 메소드
46      /// </summary>
47      private IEnumerator OnCooldownTime(float maxCooldownTime)
48      {
49          // 쿨다운 시간 저장
50          currentCooldownTime = maxCooldownTime;
51
52          SetCooldownIs(true);
53
54          while ( currentCooldownTime > 0 )
55          {
56              currentCooldownTime -= Time.deltaTime;
57              // Image UI의 fiilAmount를 조절해 채워지는 이미지 모양 설정
58              imageCooldownTime.fillAmount = currentCooldownTime/maxCooldownTime;
59              // Text UI에 쿨다운 시간 표시
60              textCooldownTime.text = currentCooldownTime.ToString("F1");
61
62              yield return null;
63          }
64
65          SetCooldownIs(false);
66      }
67
68      private void SetCooldownIs(bool boolean)
69      {
70          isCooldown                  = boolean;
71          textCooldownTime.enabled    = boolean;
72          imageCooldownTime.enabled   = boolean;
73      }
74  }
```

# Visual Components

- ImageThunder 오브젝트의 "CooldownTime" 컴포넌트 변수 설정



# Visual Components

- 스킬 시전을 제어하는 디버그용 스크립트 생성 및 작성
  - C# Script 생성 후 스크립트의 이름을 "DebugSkillInput"으로 변경
  - 스크립트를 "DebugSkillInput" 오브젝트의 컴포넌트로 적용

빈 오브젝트 생성 후 컴포넌트로 적용

```
1  using System.Collections.Generic;
2  using UnityEngine;
3  using UnityEngine.UI;
4  using UnityEngine.EventSystems;
5
6  public class DebugSkillInput : MonoBehaviour
7  {
8      [SerializeField]
9      private GraphicRaycaster    graphicRaycaster;
10     [SerializeField]
11     private CooldownTime[]      cooldownTime;
12
13     private List<RaycastResult> raycastResults;
14     private PointerEventData    pointerEventData;
15
16     private void Awake()
17     {
18         raycastResults   = new List<RaycastResult>();
19         pointerEventData = new PointerEventData(null);
20     }
21
```

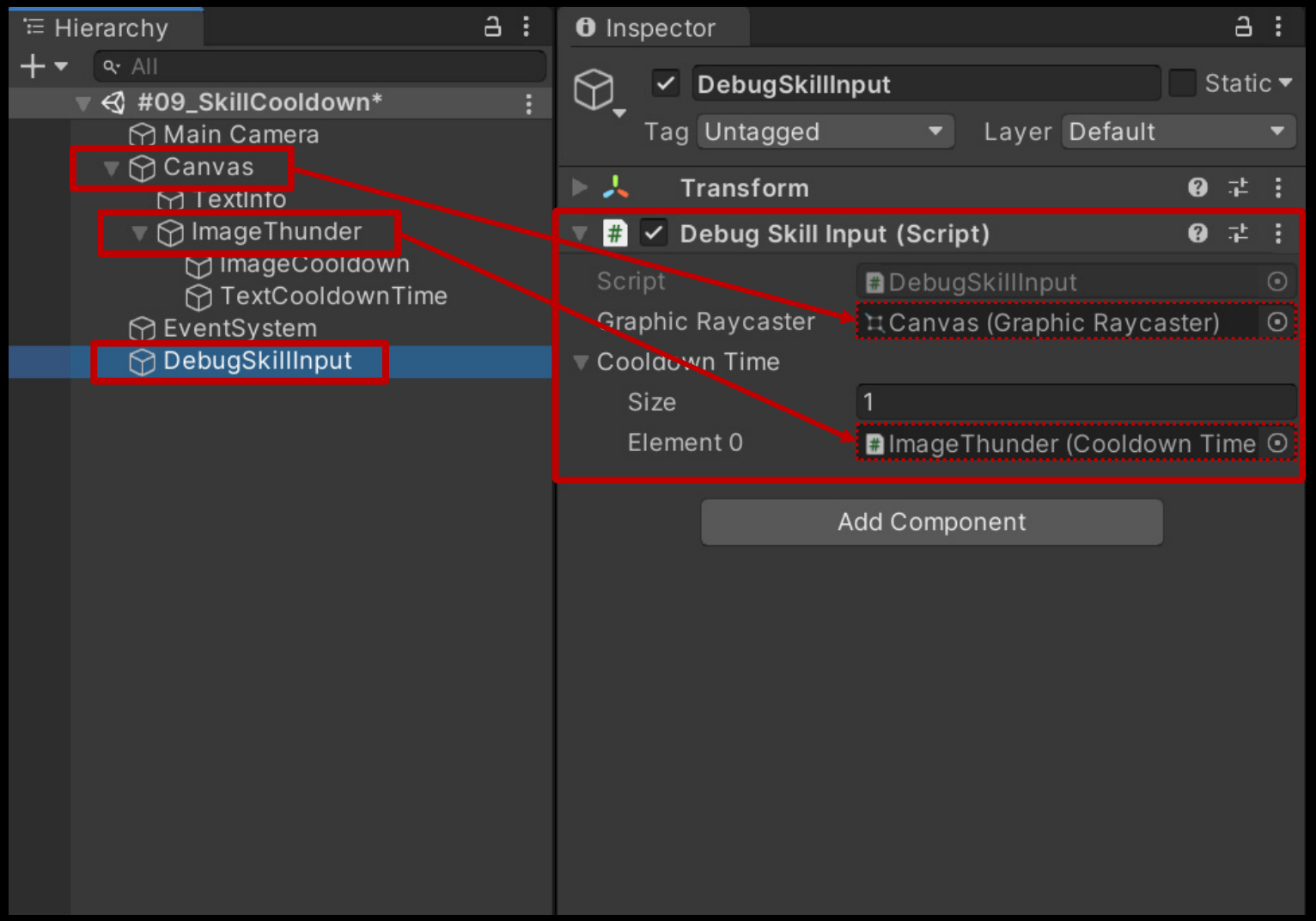
# Visual Components

- 스킬 시전을 제어하는 디버그용 스크립트 생성 및 작성 (계속)

```
22      private void Update()
23      {
24          // 1 ~ 9 숫자키를 눌러 스킬 시전
25          for ( int i = 0; i < cooldownTime.Length; ++ i )
26          {
27              if ( Input.GetKeyDown((i+1).ToString()) )
28              {
29                  cooldownTime[i].StartCooldownTime();
30              }
31          }
32
33          // 스킬 아이콘을 마우스로 클릭해서 스킬 시전
34          if ( Input.GetMouseButtonDown(0) )
35          {
36              raycastResults.Clear();
37
38              pointerEventData.position = Input.mousePosition;
39              graphicRaycaster.Raycast(pointerEventData, raycastResults);
40
41              if ( raycastResults.Count > 0 )
42              {
43                  // as CooldownTime으로 형 변환을 시도해 만약 스킬이 아닌 다른 UI를 눌렀다면 null이 저장되도록 한다.
44                  CooldownTime cool = raycastResults[0].gameObject.GetComponent<CooldownTime>() as CooldownTime;
45                  if ( cool != null )
46                  {
47                      cool.StartCooldownTime();
48                  }
49              }
50          }
51      }
52  }
```

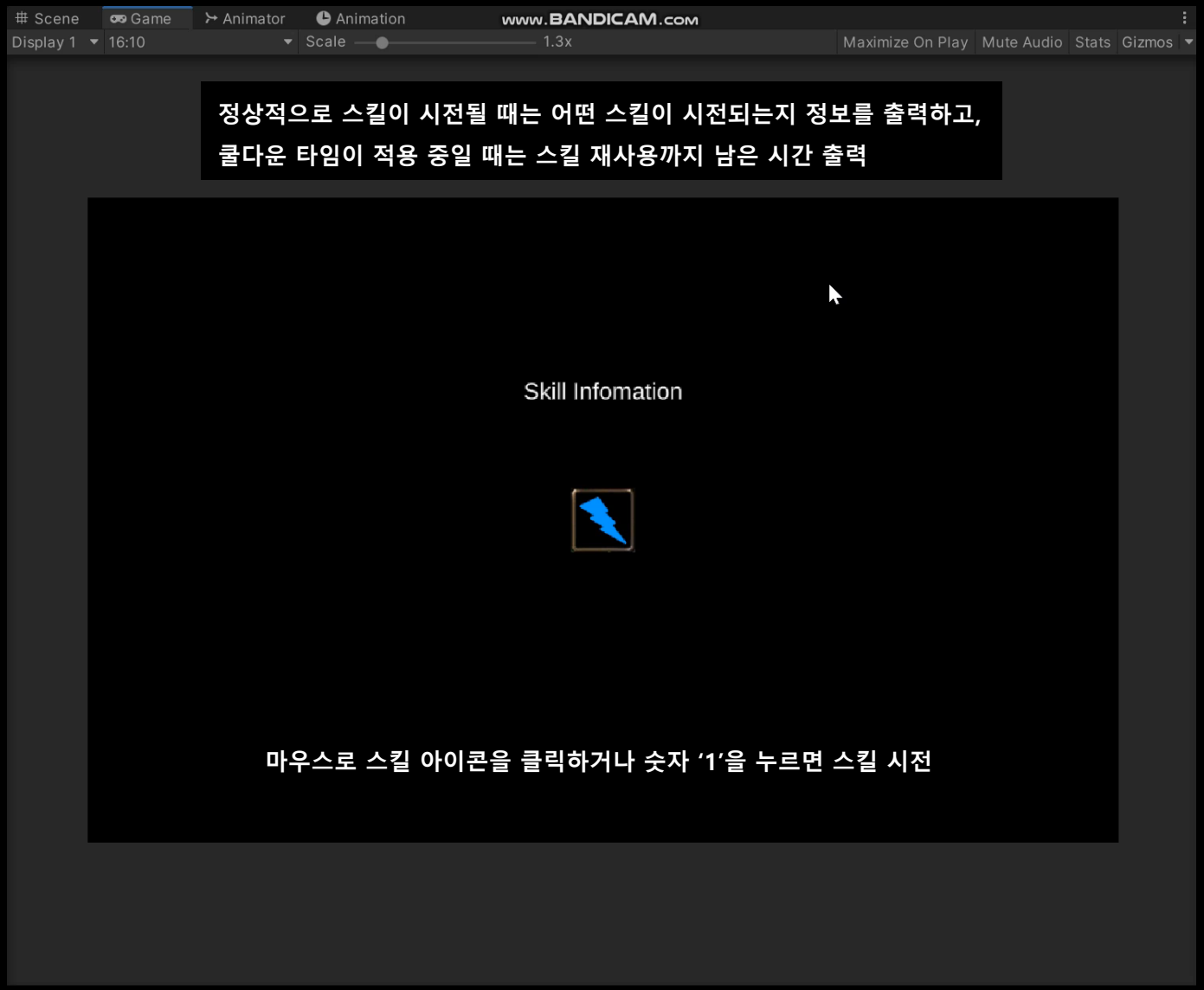
# Visual Components

- DebugSkillInput 오브젝트의 "DebugSkillInput" 컴포넌트 변수 설정



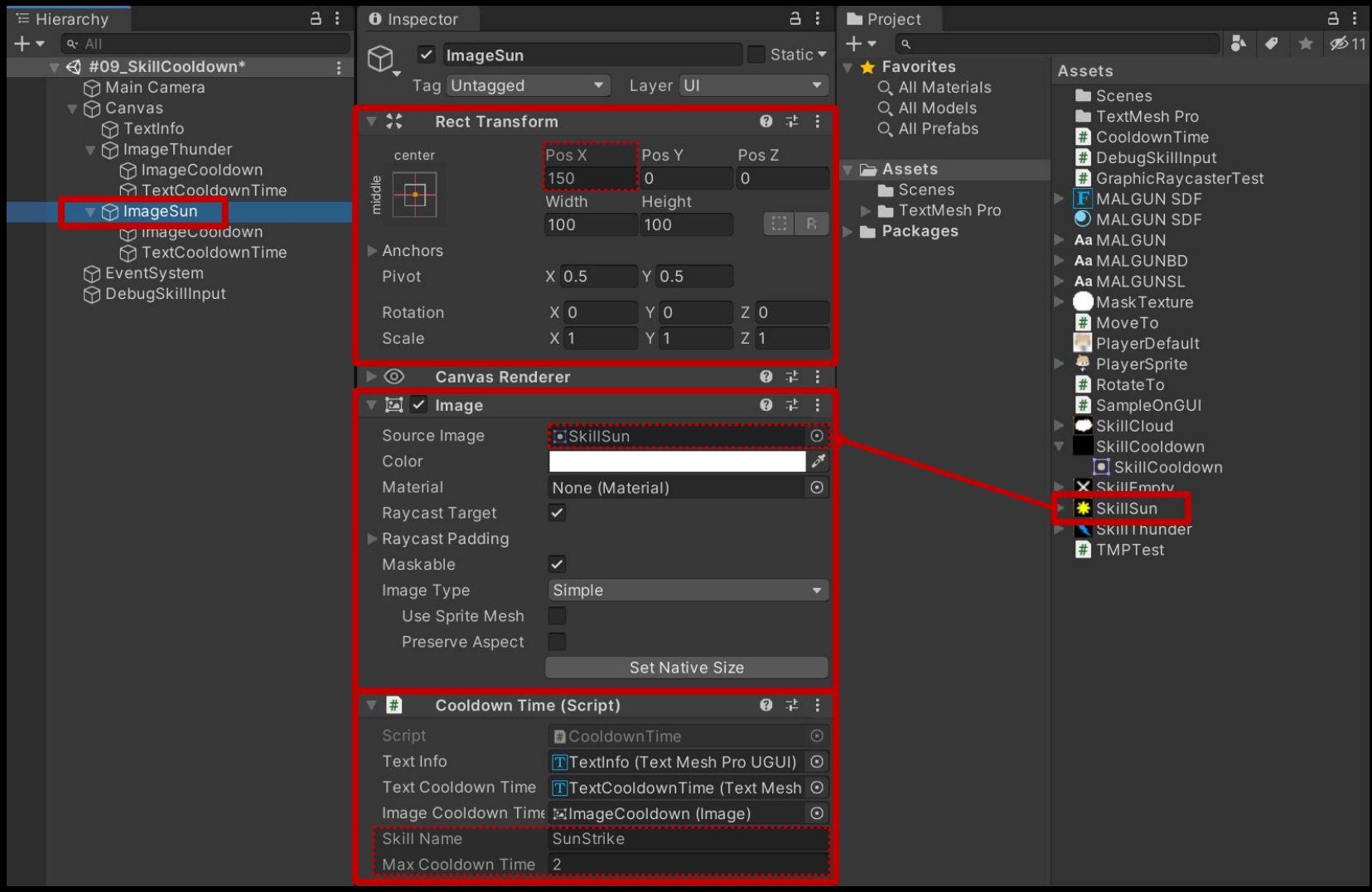
# Visual Components

- 결과 화면



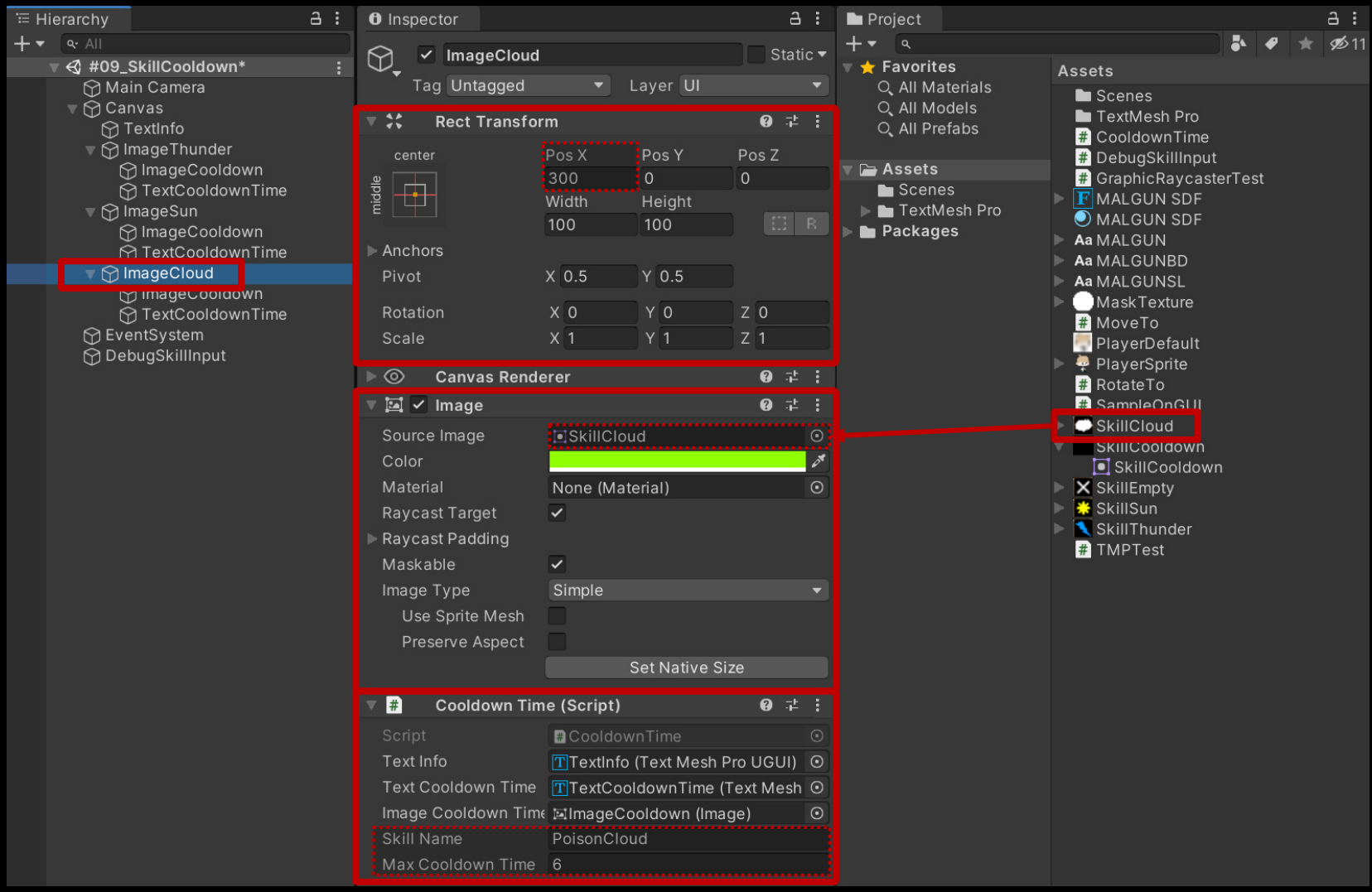
# Visual Components

- 스킬 추가 [SunStrike]
  - Ctrl+D로 "ImageThunder" 오브젝트를 복제



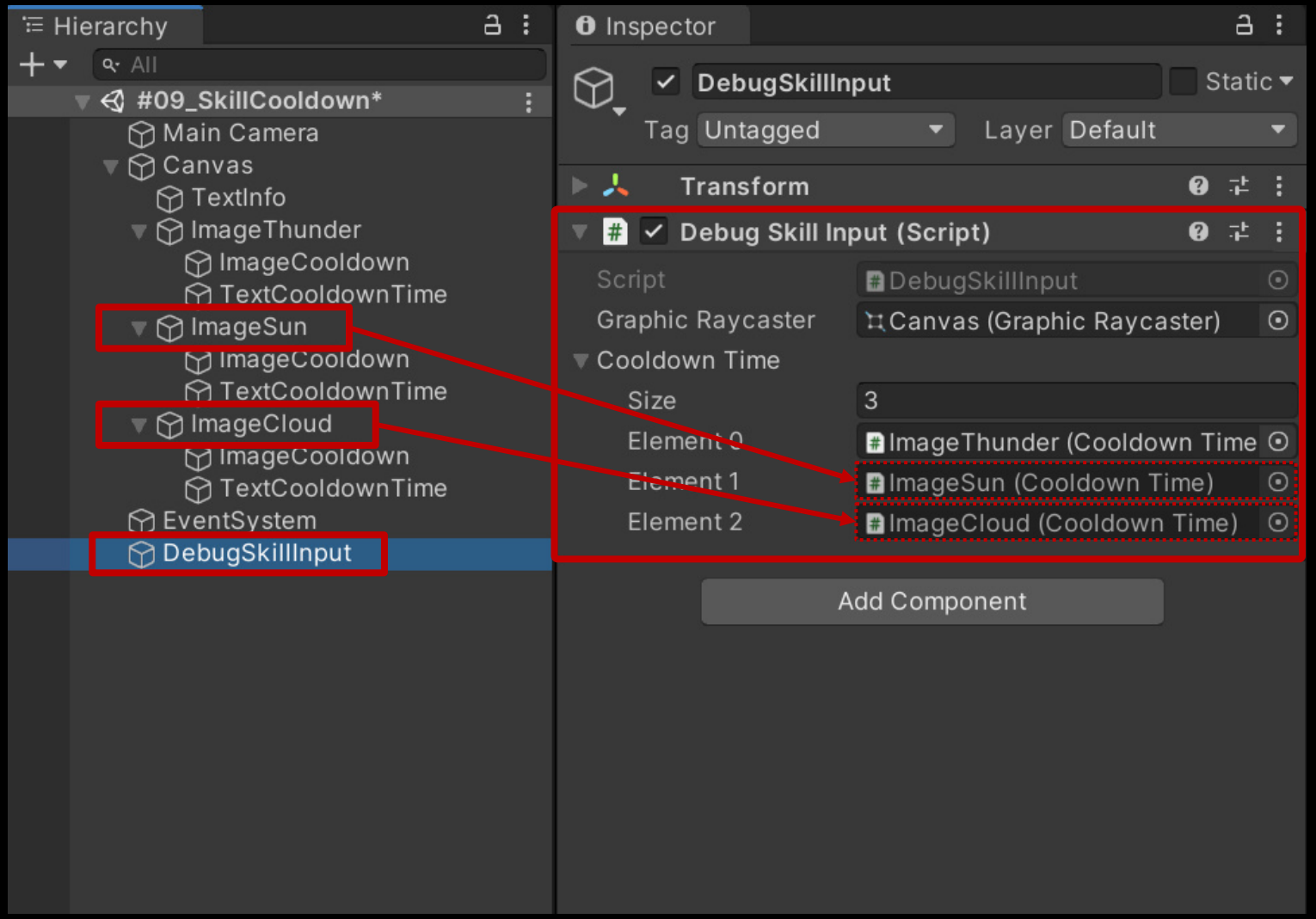
# Visual Components

- 스킬 추가 [PoisonCloud]
  - Ctrl+D로 "ImageThunder" 오브젝트를 복제



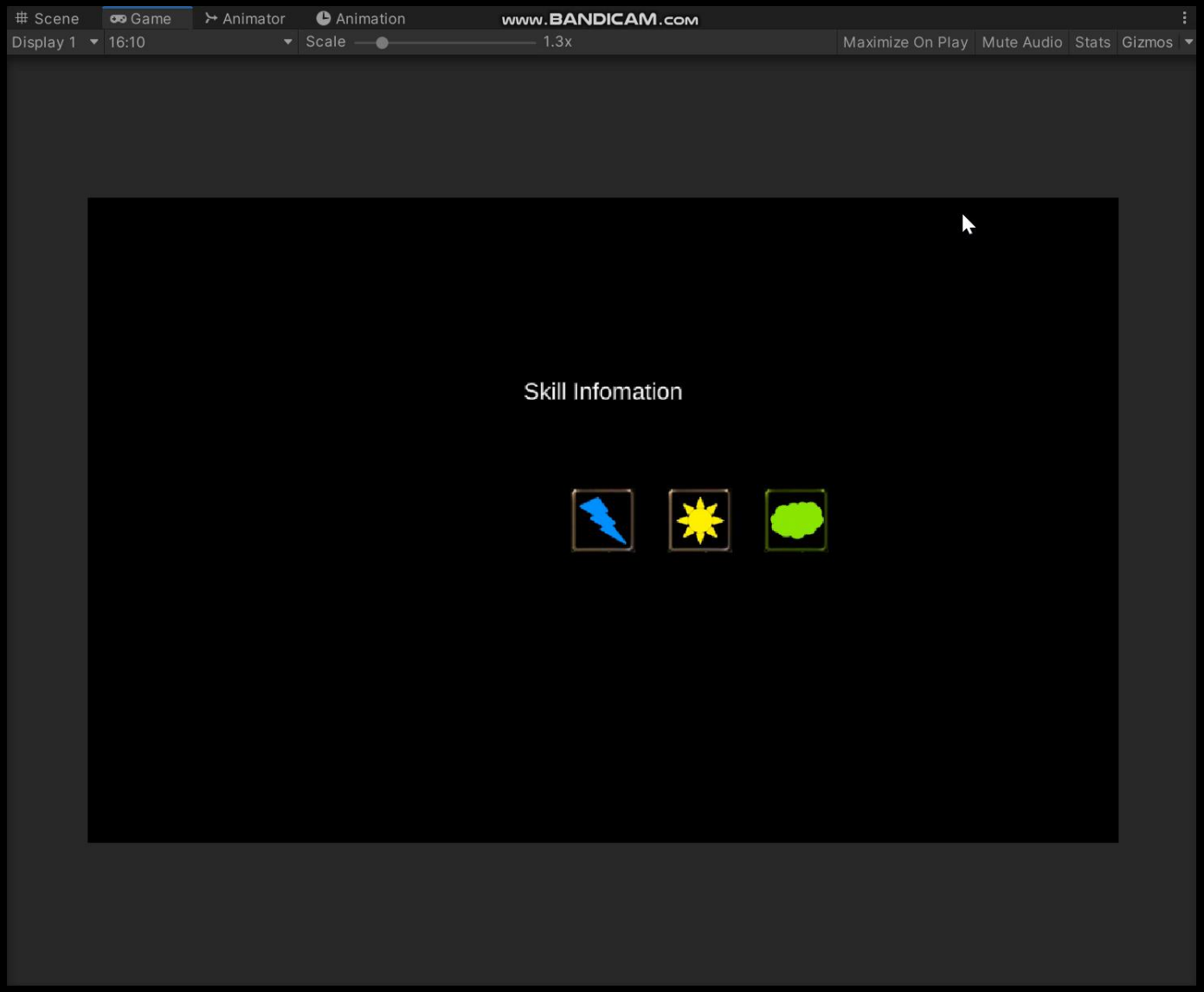
# Visual Components

- DebugSkillInput 오브젝트의 "DebugSkillInput" 컴포넌트 변수 설정



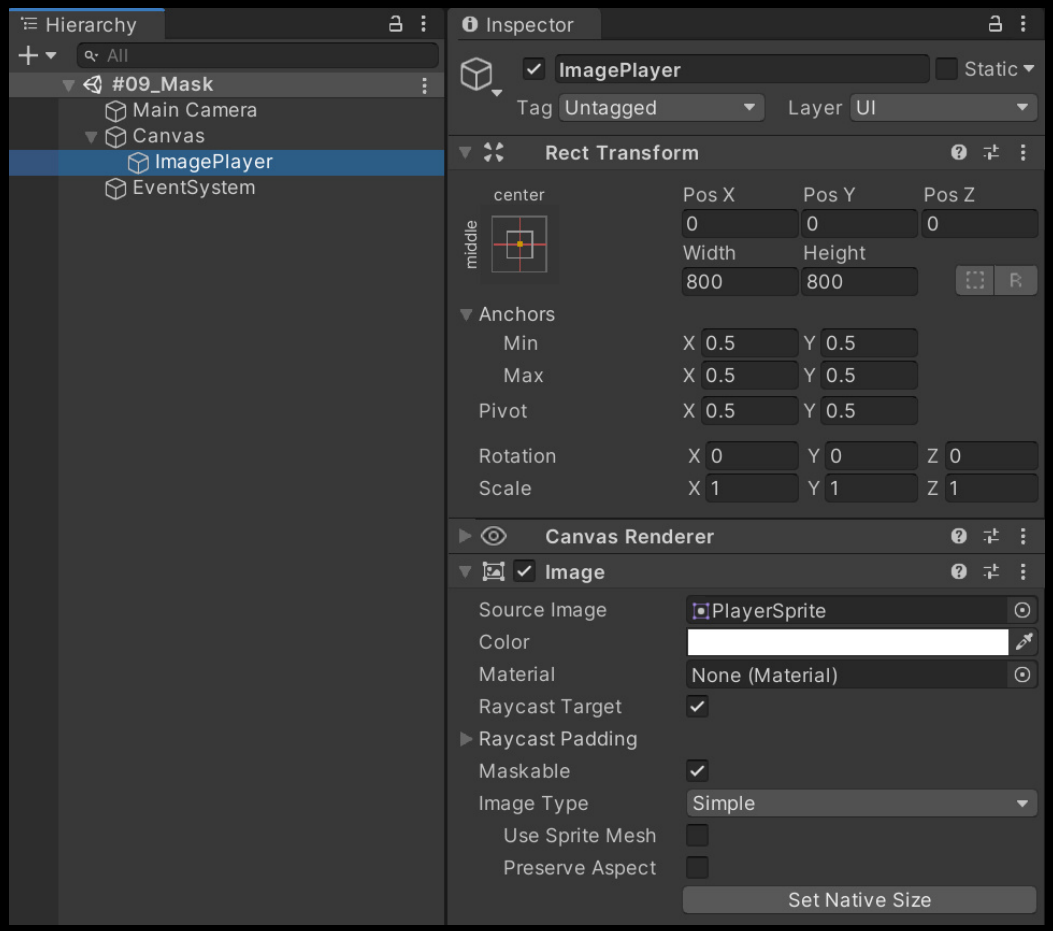
# Visual Components

- 결과 화면



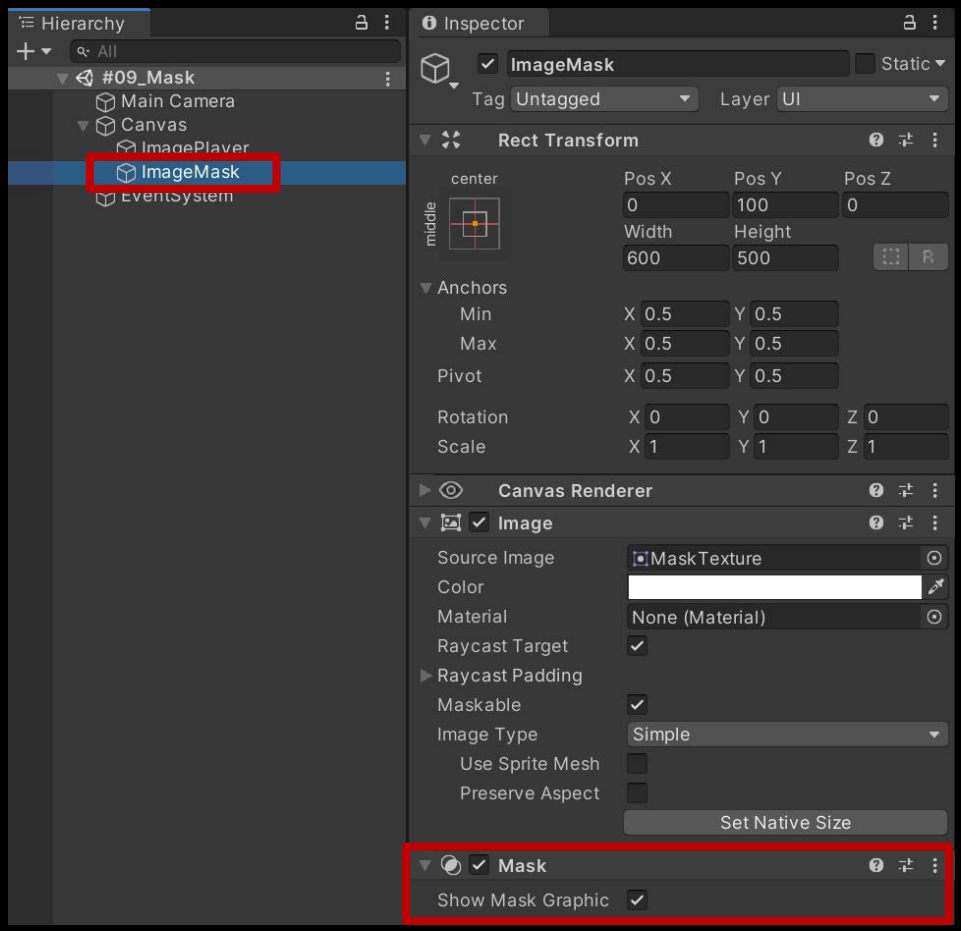
# Visual Components

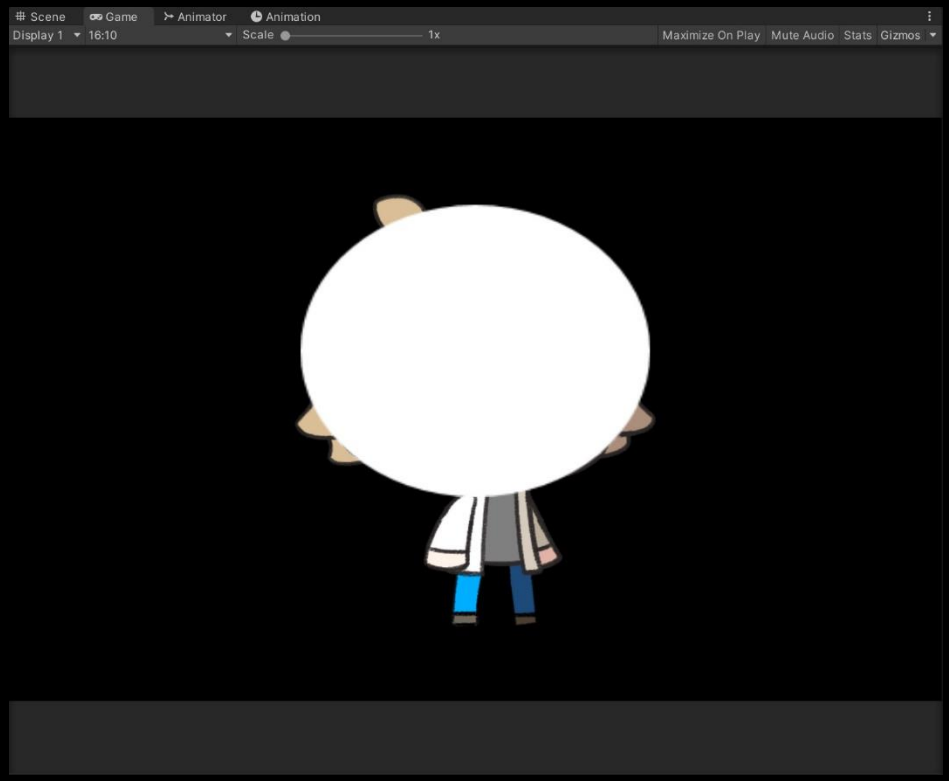
- **Mask**
  - "Mask" 컴포넌트를 가지고 있는 Image UI의 자식으로 배치되는 UI들을 부모 오브젝트의 모양에 한정해서 보여주는 것




# Visual Components

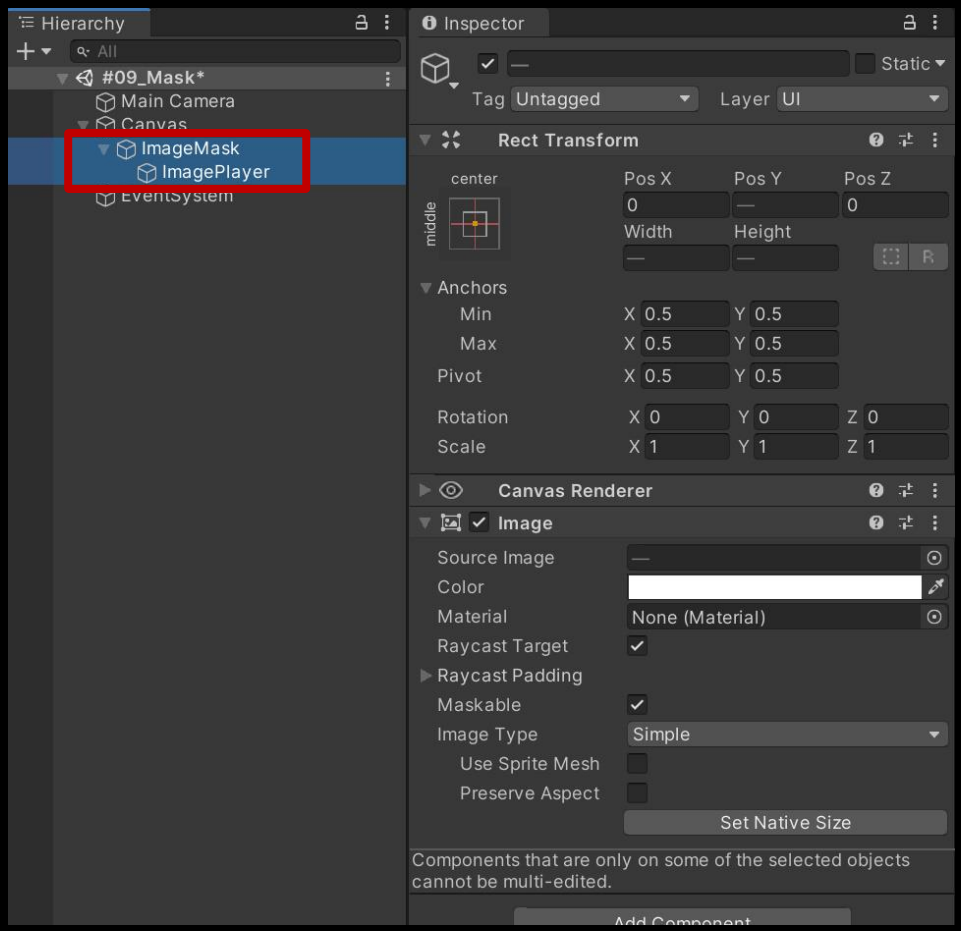
- 결과 화면 (계속)
  - Image UI를 생성하고, Mask 컴포넌트 추가



# Visual Components

- 결과 화면 (계속)
  - 부모/자식 관계가 되었을 때



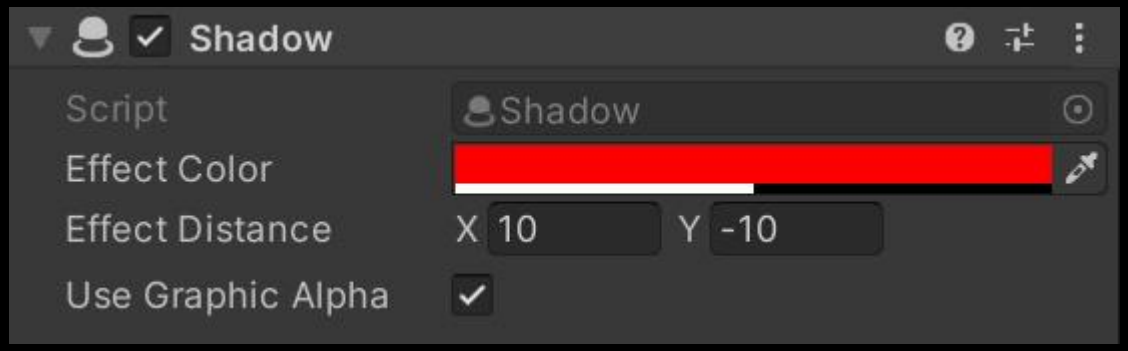

# Visual Components

## UI Effects [Outline, Shadow]

Tip. 최적화가 잘 되어 있지 않아 꼭 필요한 경우가 아니라면 사용을 권장하지는 않는다.

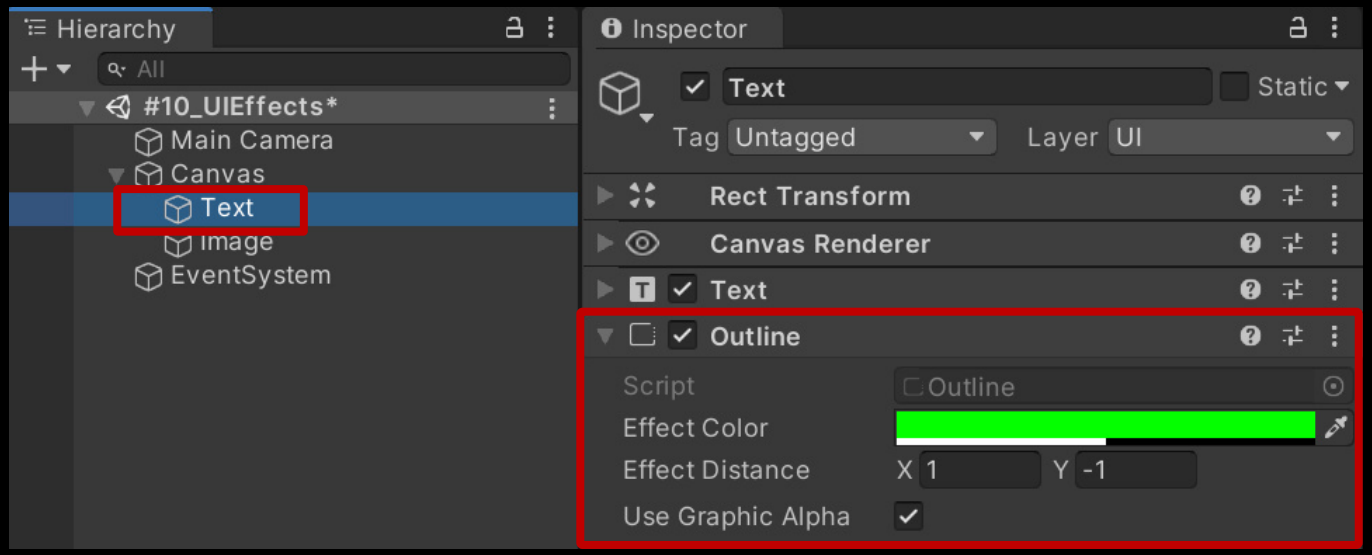
- Outline : "Text", "Image"의 외곽선을 표시

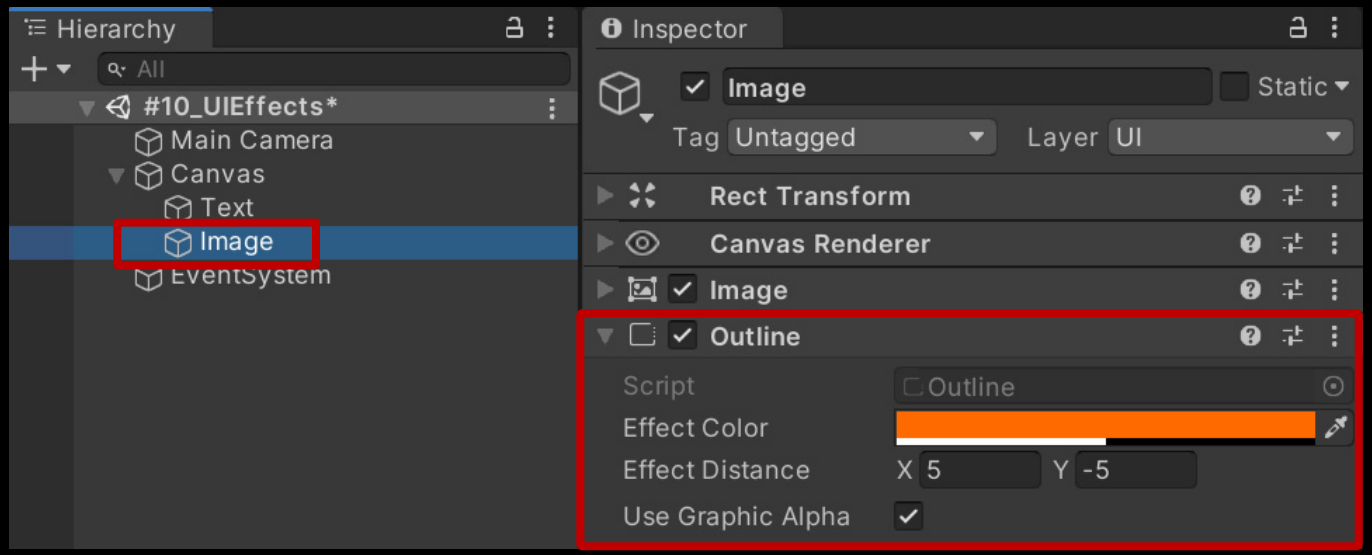

- Shadow : "Text", "Image"의 그림자를 표시



# Visual Components

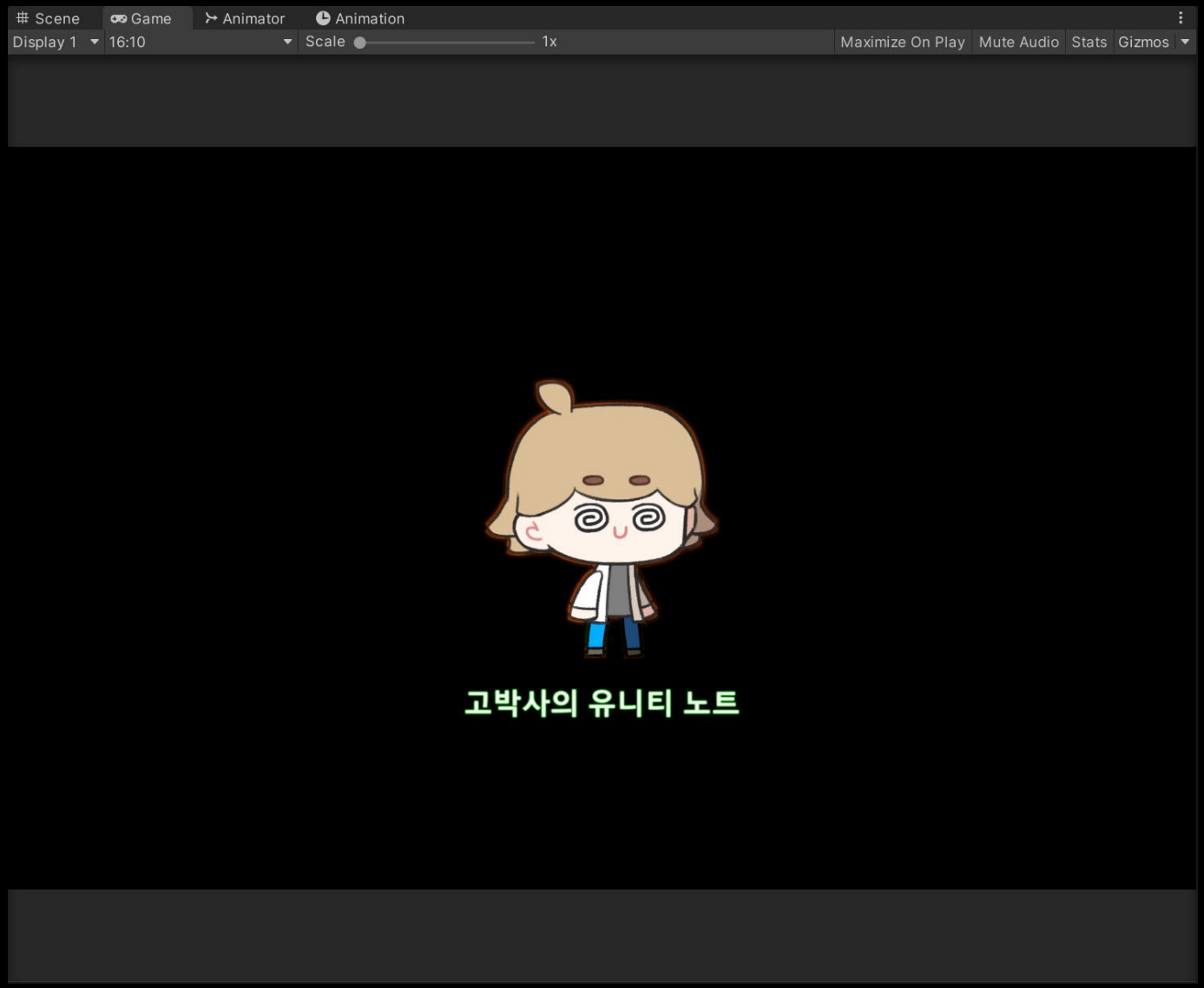
- Outline 예제



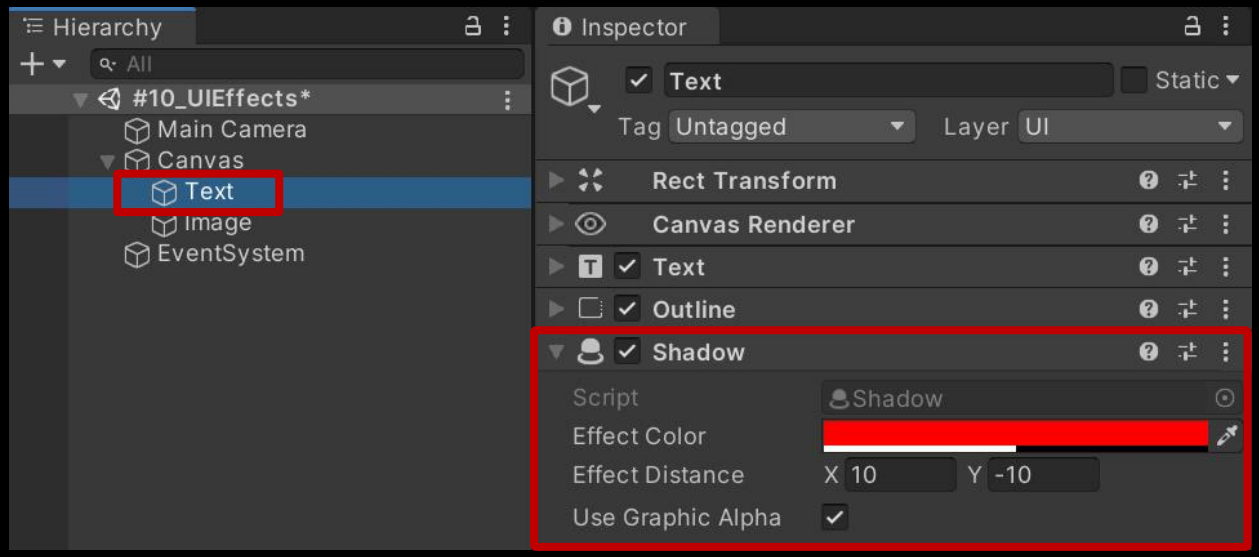
# Visual Components

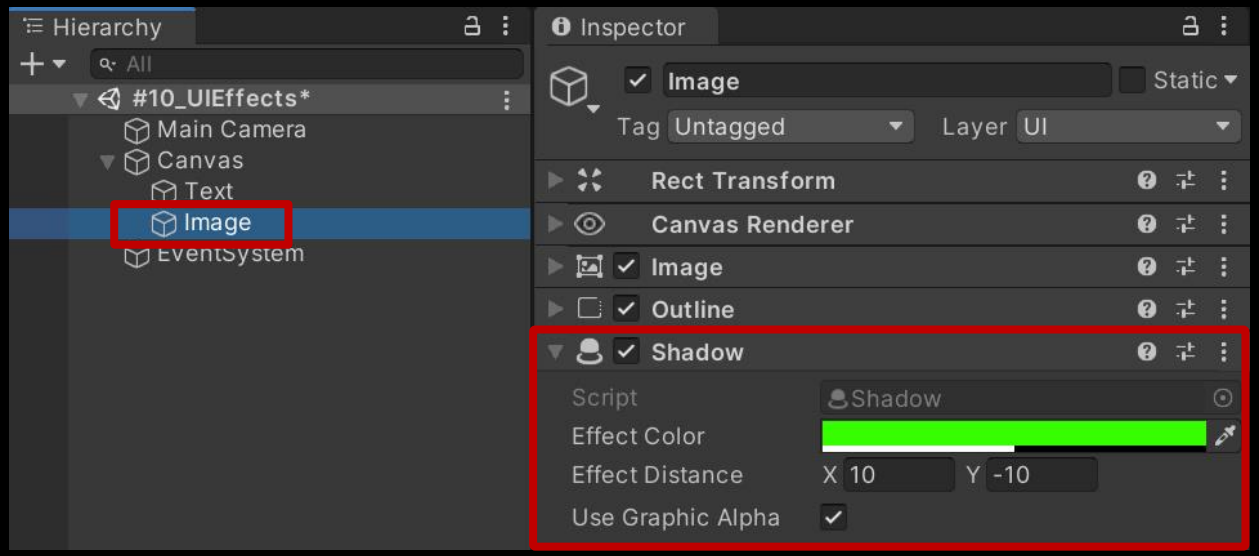
- 결과 화면 [Outline]



# Visual Components

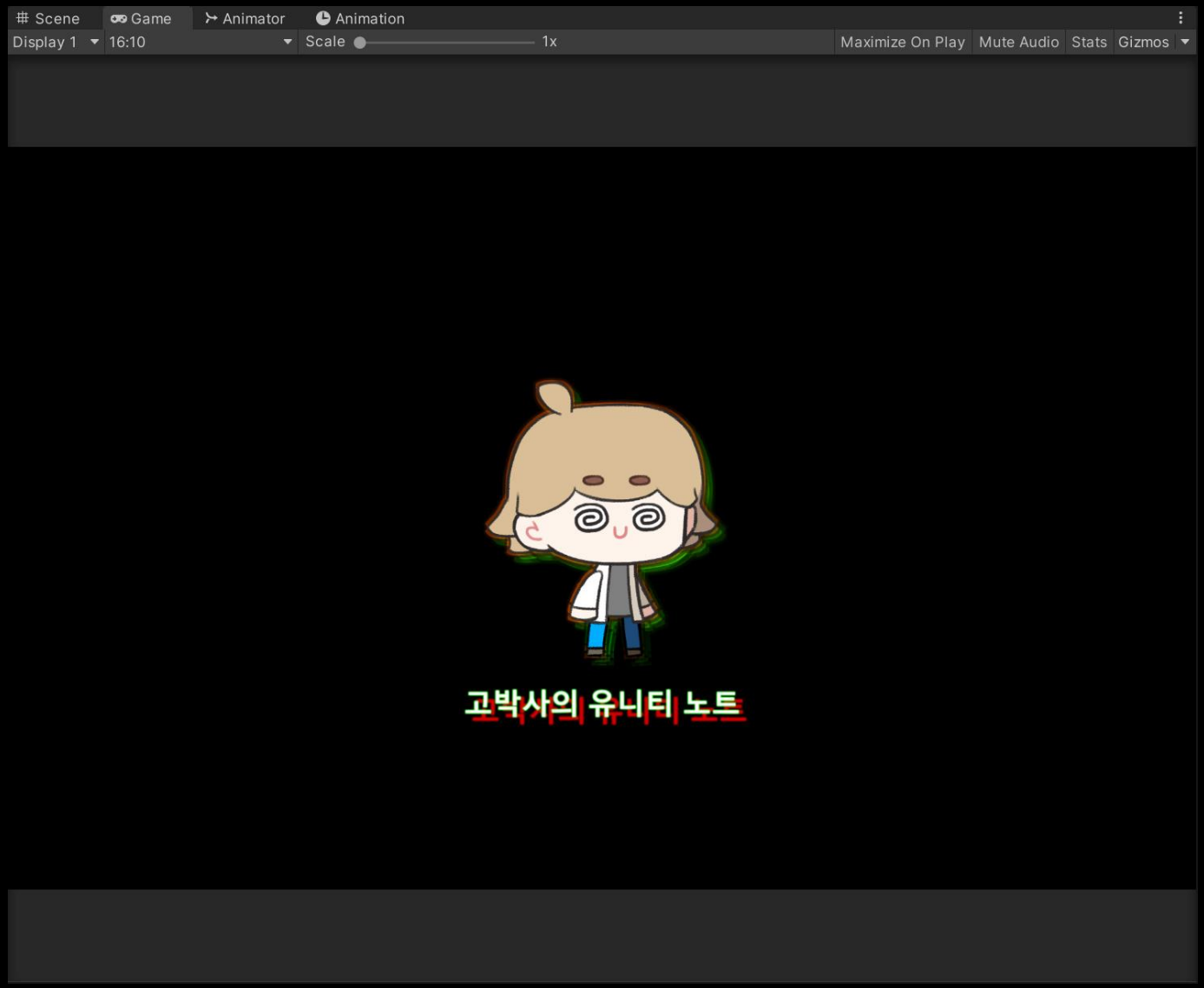
- Outline, Shadow 예제



# Visual Components

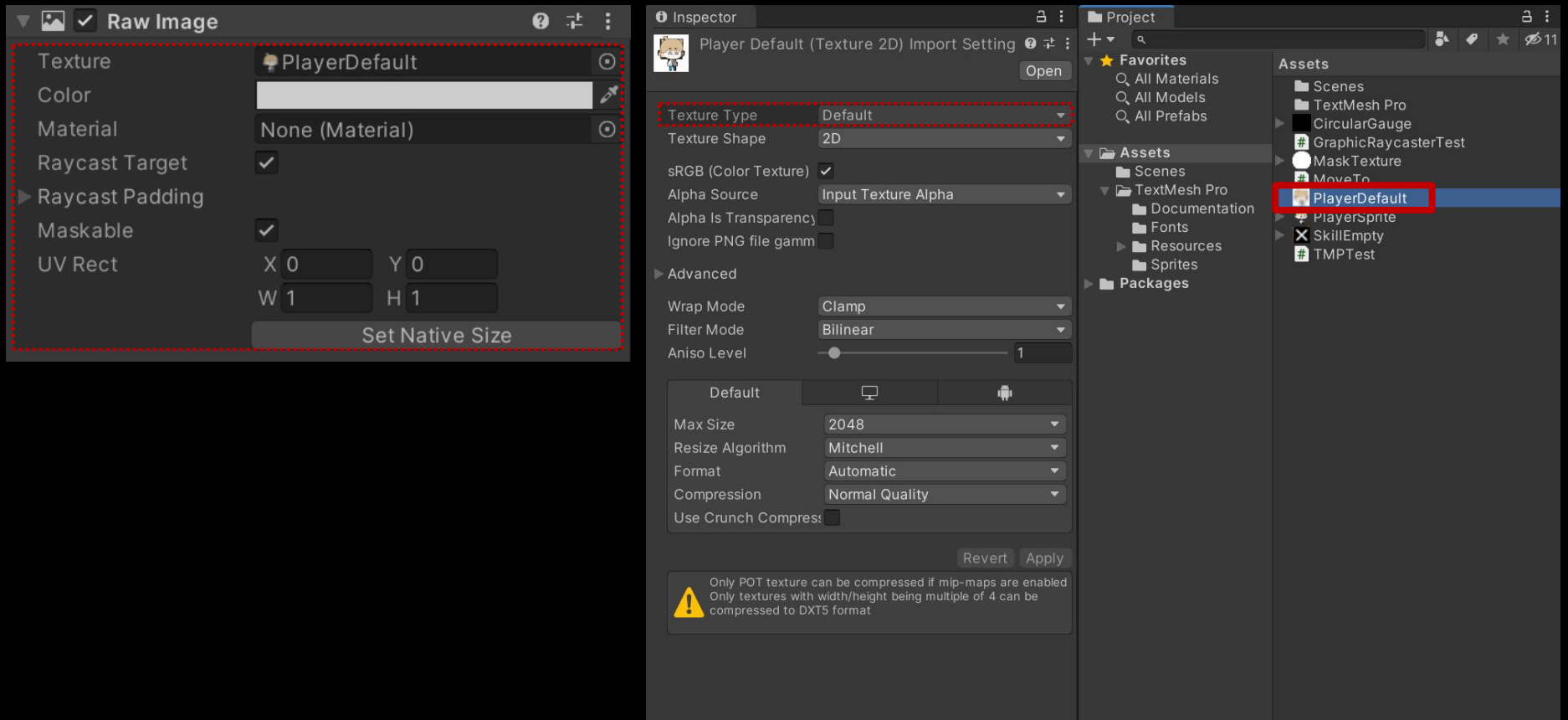
- 결과 화면 [Outline, Shadow]



# Visual Components

## RawImage

- 게임 화면에 이미지를 표시하는 UI 컴포넌트 (Default Type)
  - 텍스처의 타입이 Sprite (2D and UI)가 아닐 때 사용
  - 이미지를 표현하는 대부분의 UI는 "Image" 컴포넌트를 이용한다



# Visual Components

- RawImage 결과 화면