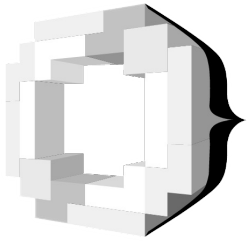
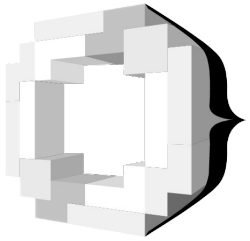
2023 유니티 중급반 6주차

오파츠 12기 여정인, 유태환



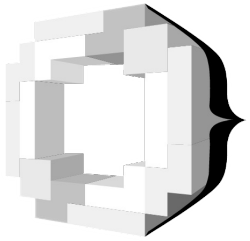
목차

- 유니티 UI 알아보기
  - UGUI? UI Toolkit?
- UGUI로 간단한 UI 구성 방법 알아보기



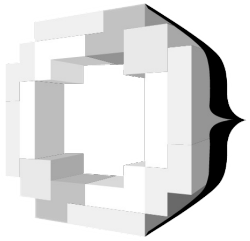
유니티 UI

- 유니티 2021 버전에서 지원하는 공식 UI 시스템은 두 가지 입니다
  - UGUI
  - UI Toolkit



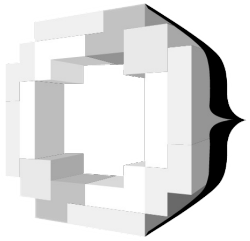
UGUI

- 기존 NGUI를 대체하는 유니티의 GUI 시스템
- 유니티 엔진에서 공식적으로 지원하는 UI!
  - UI를 제작할 때 가장 먼저 고려할 옵션
  - 유니티의 기본 개념을 잘 익혀 둬야 함



UI Toolkit

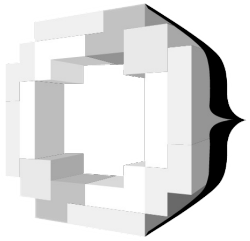
- 유니티 2021 버전부터 새롭게 도입된 새로운 UI 시스템
- 2021 버전 기준으로 Stable 하지 않음
  - 계속해서 기능이 추가, 변경, 삭제되고 있어요
  - 하지만 2022버전 이후 유니티에서는 UI Toolkit 사용을 권장하고 있어요
- 표준 웹 기술에서 영감을 받았다고 함



UI Toolkit

- UXML (Unity XML)

```
<?xml version="1.0" encoding="utf-8"?>
<UXML
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="UnityEngine.UIElements"
    xsi:noNamespaceSchemaLocation="../UIElementsSchema/UIElements.xsd"
    xsi:schemaLocation="UnityEngine.UIElements ../UIElementsSchema/UnityEngine.UIElements.xsd">

    <Label text="Select something to remove from your suitcase:"/>
    <Box>
        <Toggle name="boots" label="Boots" value="false" />
        <Toggle name="helmet" label="Helmet" value="false" />
        <Toggle name="cloak" label="Cloak of invisibility" value="false"/>
    </Box>
    <Box>
        <Button name="cancel" text="Cancel" />
        <Button name="ok" text="OK" />
    </Box>
</UXML>
```

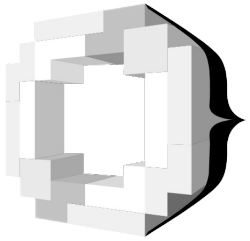


UI Toolkit

- USS (Unity Style Sheet)

```
Button {
  width: 200px;
}
```

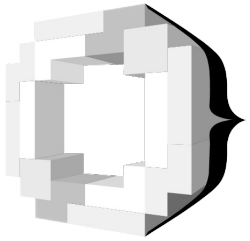
```
.button {
}

.button__icon {
}

.button--small {
}

.button--small .button__icon {
}
```



UI Toolkit

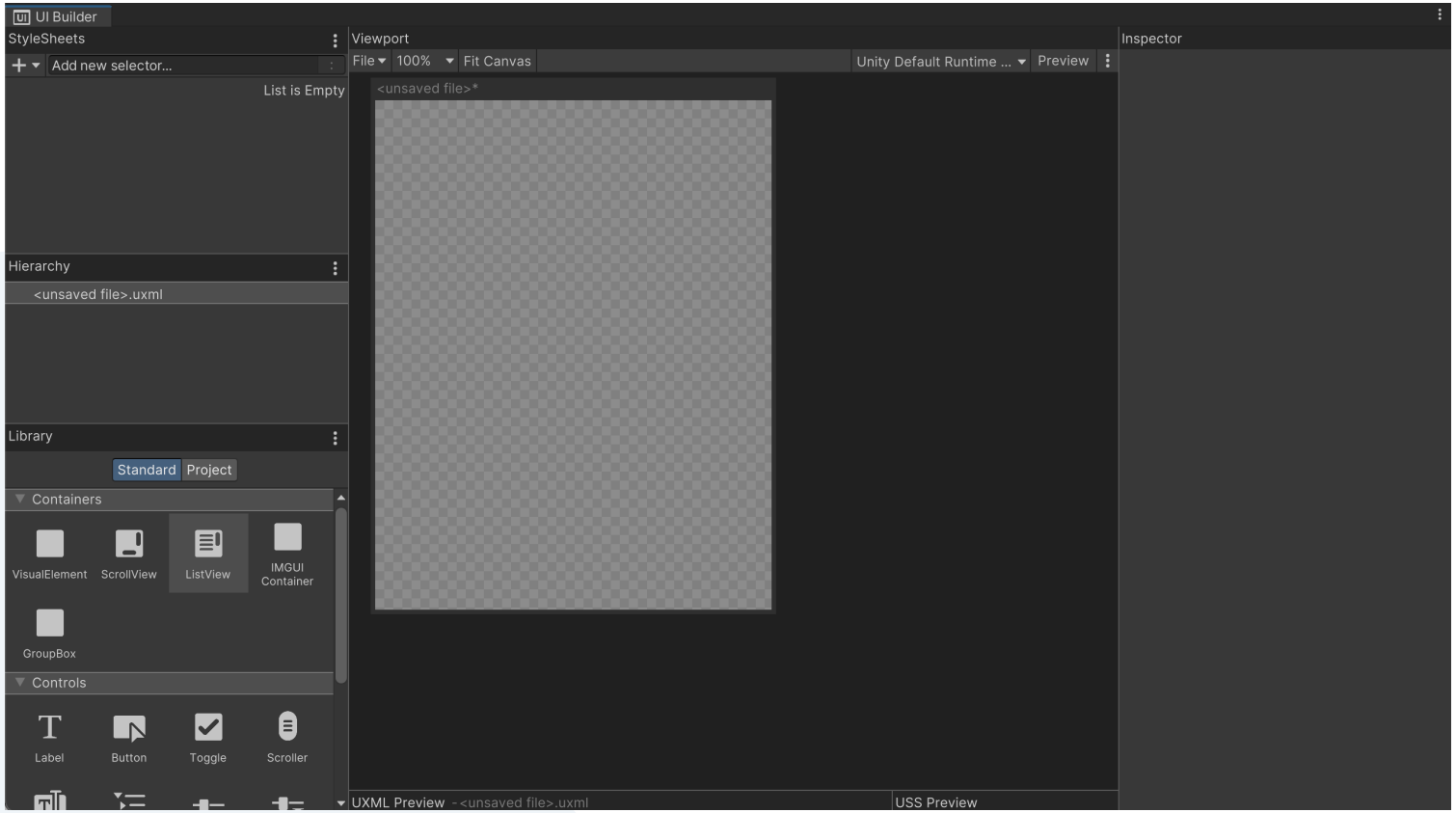
- UXML은 HTML에서,
- USS는 CSS에서 많은 부분을 가져왔어요
- 웹 개발에 익숙하다면 유니티를 크게 몰라도

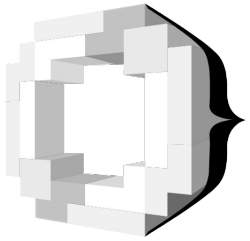
  유니티 UI 개발을 간단히 할 수 있도록 디자인했다고 해요



UI Toolkit

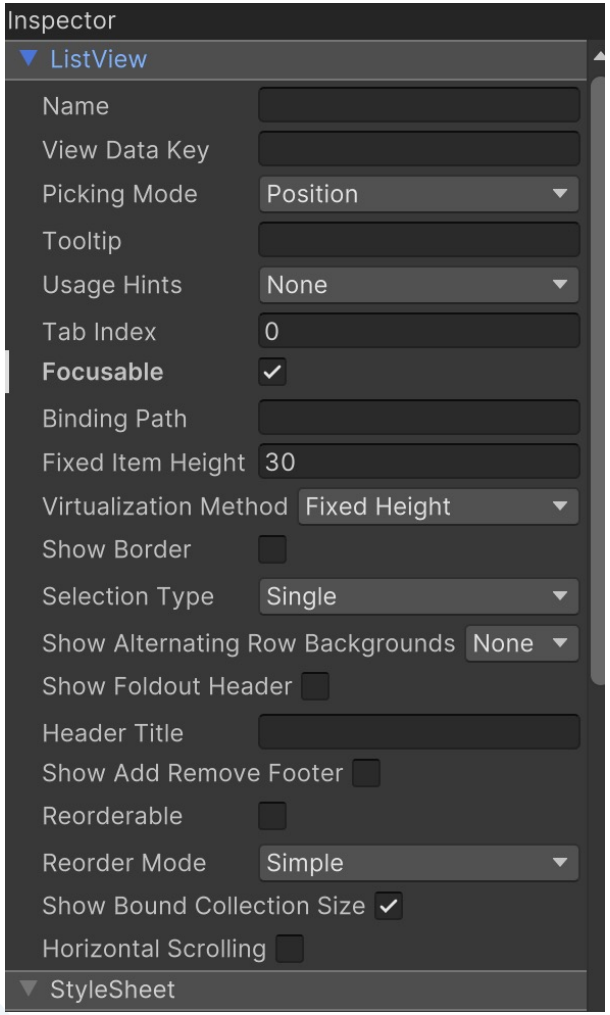
- UI 에디터도 제공하고 있습니다

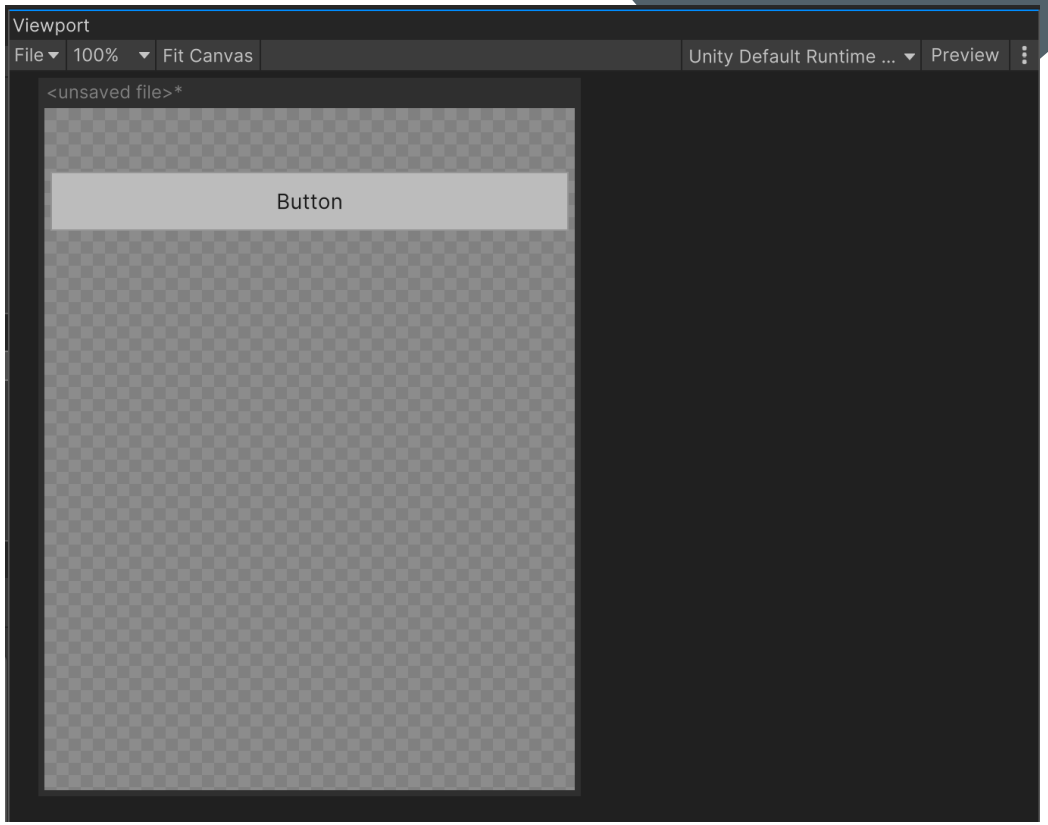


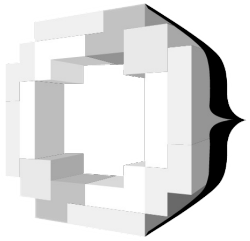


UI Toolkit

- 잠깐 살펴봅시다





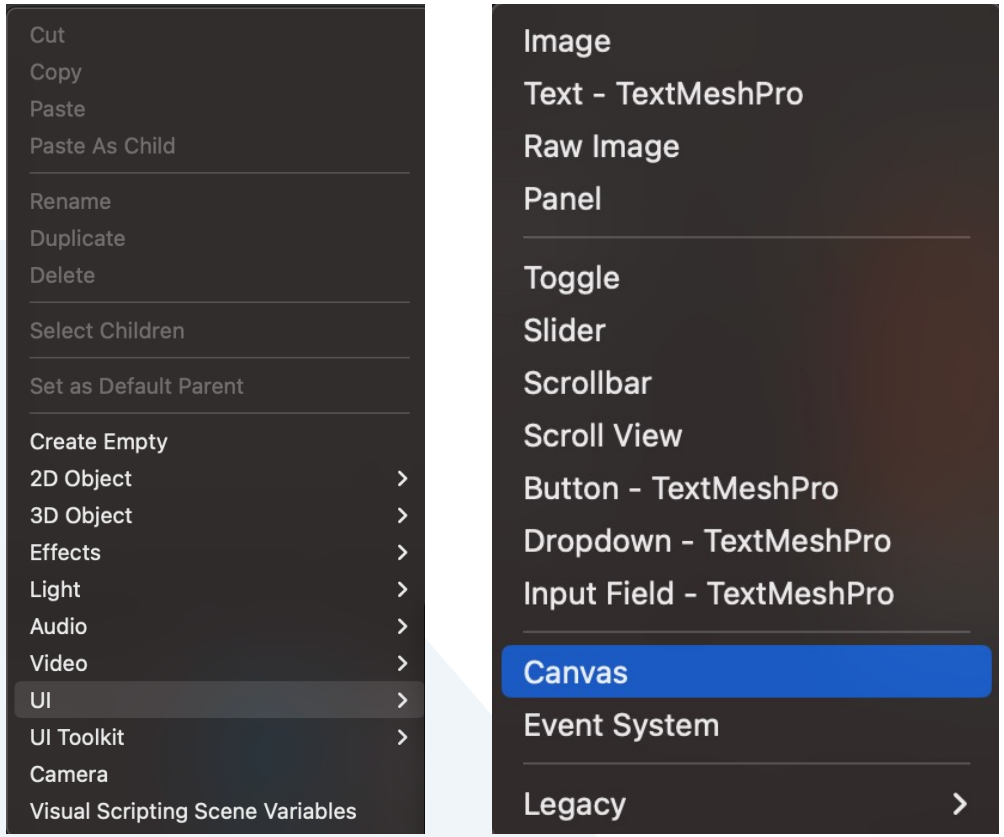


UGUI vs UI Toolkit

- UI Toolkit에는 아직 개발 도중인 기능들이 있습니다.
- 오히려 UGUI에 더 풍부한 기능이 있을 수 있어요.
- 그러나 에디터 UI 수정에는 UI Toolkit이 추천됩니다.
- 실제 게임에 적용되는 UI 제작에는 장점을 따져서 선택합시다

UGUI

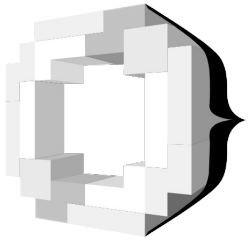
- 오늘 수업에서는 UGUI를 기준으로 설명하겠습니다



Hierarchy에서
Canvas를 생성해주세요!

UGUI의 UI 컴포넌트들은 기본적으로
Canvas에서 렌더링 됩니다.

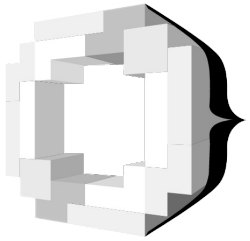
Canvas는 여러 개 존재할 수 있어요



UGUI

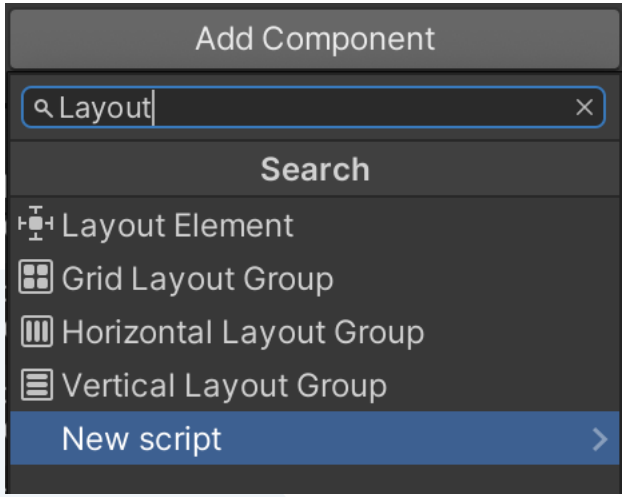
- Canvas Scaler 설정

반응형 UI를 위해서는 Scale With Screen Size를 사용합니다.
가로 화면(Landscape) – Match Height (Match value 1)
세로 화면(Portrait) - Match Width (Match value 0)
실제 게임을 예시로 보겠습니다.

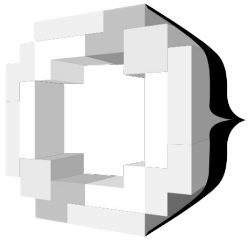


레이아웃

- 유니티에서 기본적으로 제공하는 레이아웃입니다

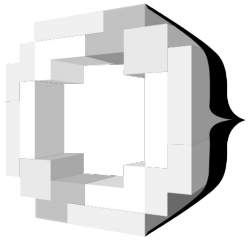


직관적인 아이콘과 이름을 보고 어떨 때 사용할 수 있을 지 알 수 있어요!
복잡하고 구조적인 UI를 제작할 때 좋아요!
잠깐 직접 확인해봅시다.



레이아웃

- Layout Group에는 문제가 있어요
  - Dirty Flag 시스템을 사용합니다
  - 레이아웃에 있는 UI Element 중 단 하나만 변경되어도 Dirty Flag가 설정될 수 있습니다.
  - 이때 레이아웃 전체 서치하고, 다시 렌더링합니다 (모든 Element에 대해 GetComponent 호출) -> 성능에 문제!



레이아웃

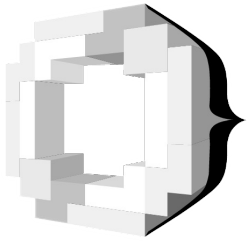
- 유니티 공식문서에서 제안하는 놀라운 해결책

Solution: Avoid layout groups when possible.

Use Anchors for proportional layouts. On hot UIs with a dynamic number of UI Elements, consider writing your own code to calculate layouts. Be sure to use this on demand, rather than for every single change.

(?)

- 실행 도중에 변경이 없다면 그냥 레이아웃을 쓰는 게 좋아요
- 하지만 변경이 있다면 레이아웃을 직접 구현하는 게 좋아요 (or 다른 라이브러리 쓰기)

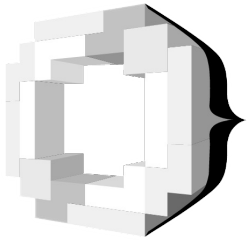


레이아웃

- 저는 Panel로 레이아웃을 구현합니다
- Canvas 속의 여러 Canvas들로 구현해도 돼요
  - 다시 그려줘야 하는 주기가 비슷한 것들끼리 묶는게 좋다고 해요

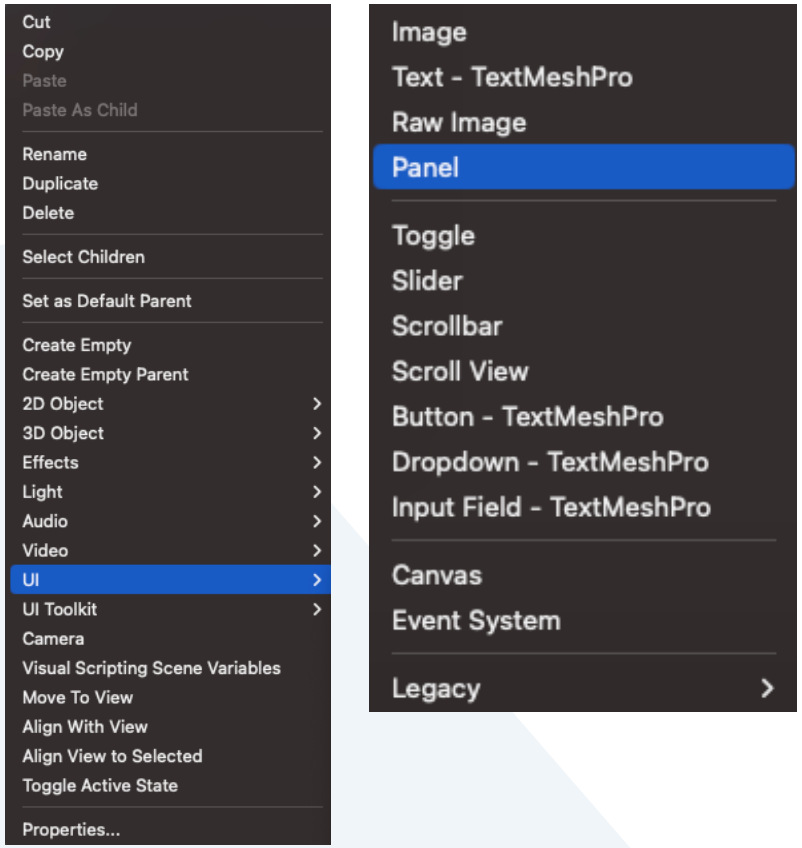
    -> 성능 향상!
  - 예) (절대 변경 안되는 타이틀) (버튼 누르면 바뀌는 메뉴 화면)

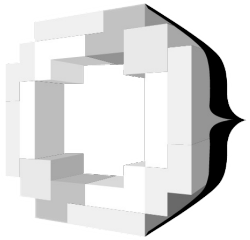
    (계속해서 바뀌는 로딩 바)



레이아웃

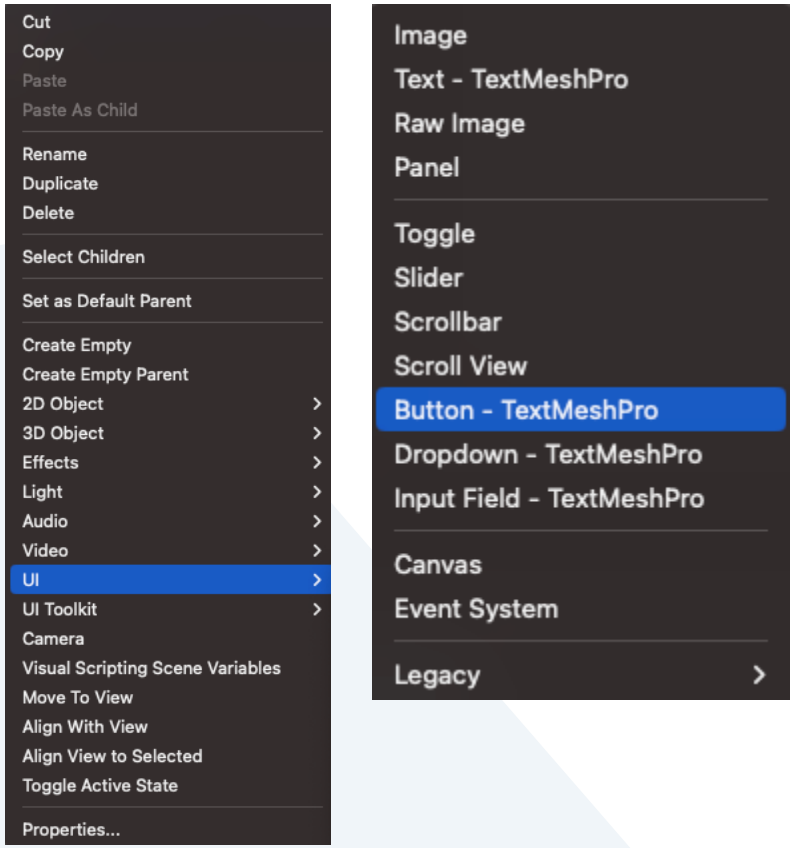
- 아까 만들어 준 Canvas를 우클릭하고 Panel을 추가해줘요

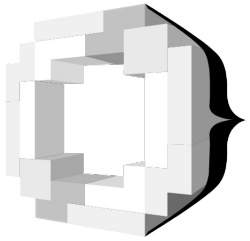




레이아웃

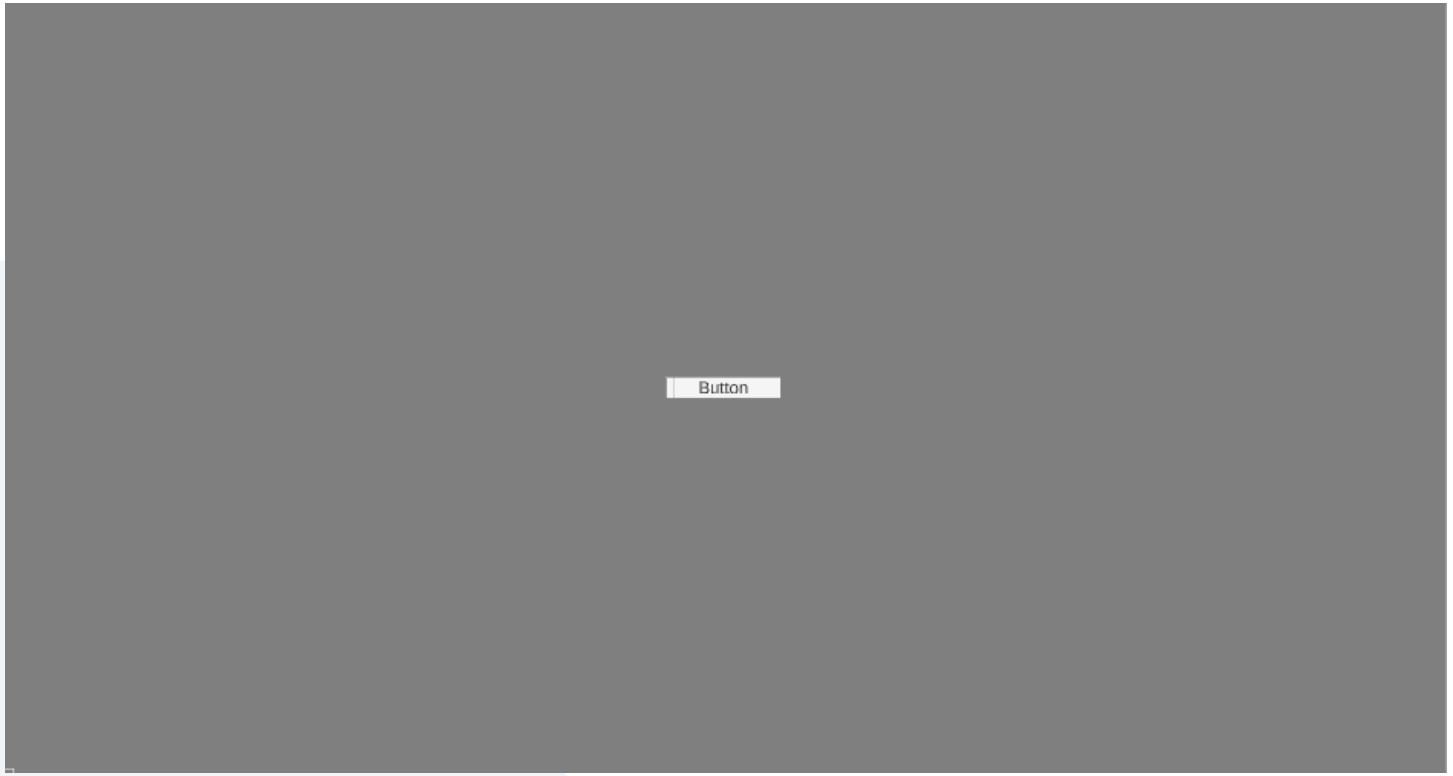
- 다시 Panel을 우클릭하고 Button을 하나 추가해봅시다

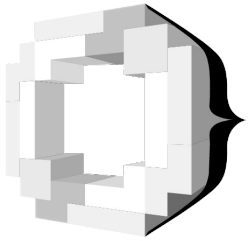




레이아웃

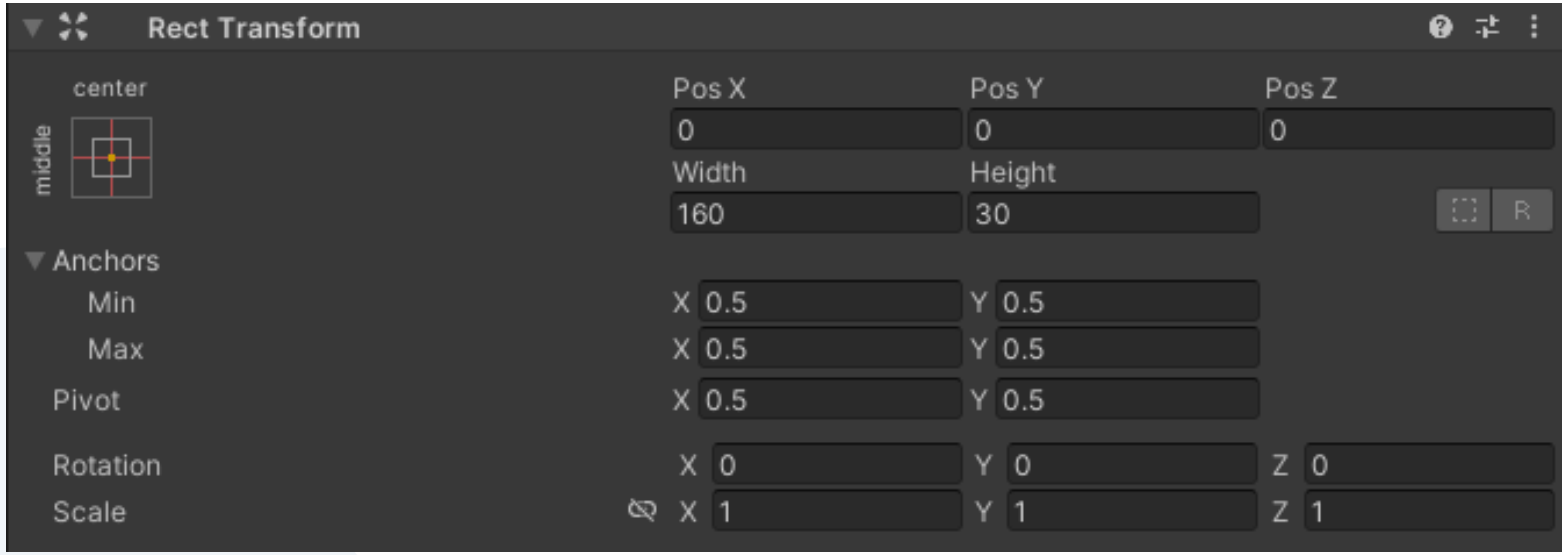
- Button이 생겼어요! 이제 위치를 조정해봅시다.

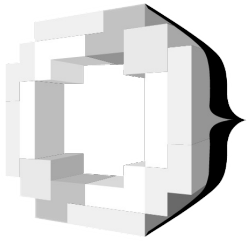




레이아웃

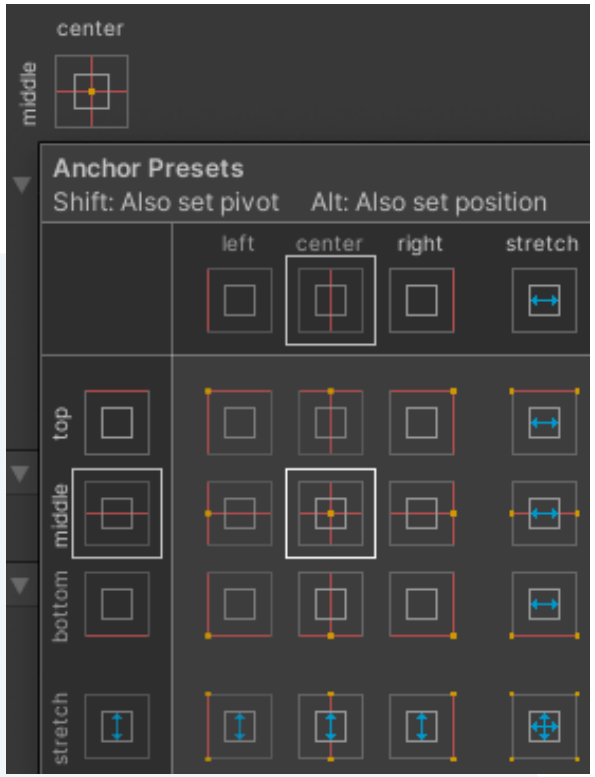
- UI Element들은 모두 Rect Transform을 가집니다





레이아웃

- 왼쪽의 그림을 클릭하면 아래 창을 볼 수 있어요

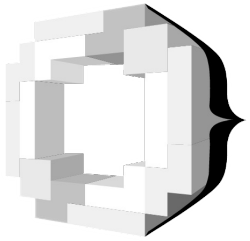


그냥 클릭: Anchor 변경

Shift + 클릭: Pivot, Anchor 변경

Alt + 클릭: Anchor, Position 변경

Shift + Alt + 클릭: Pivot, Anchor, Position 변경



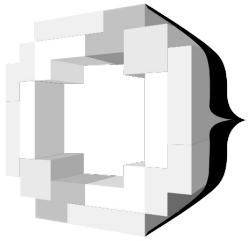
레이아웃

- Pivot? Anchor? Position?

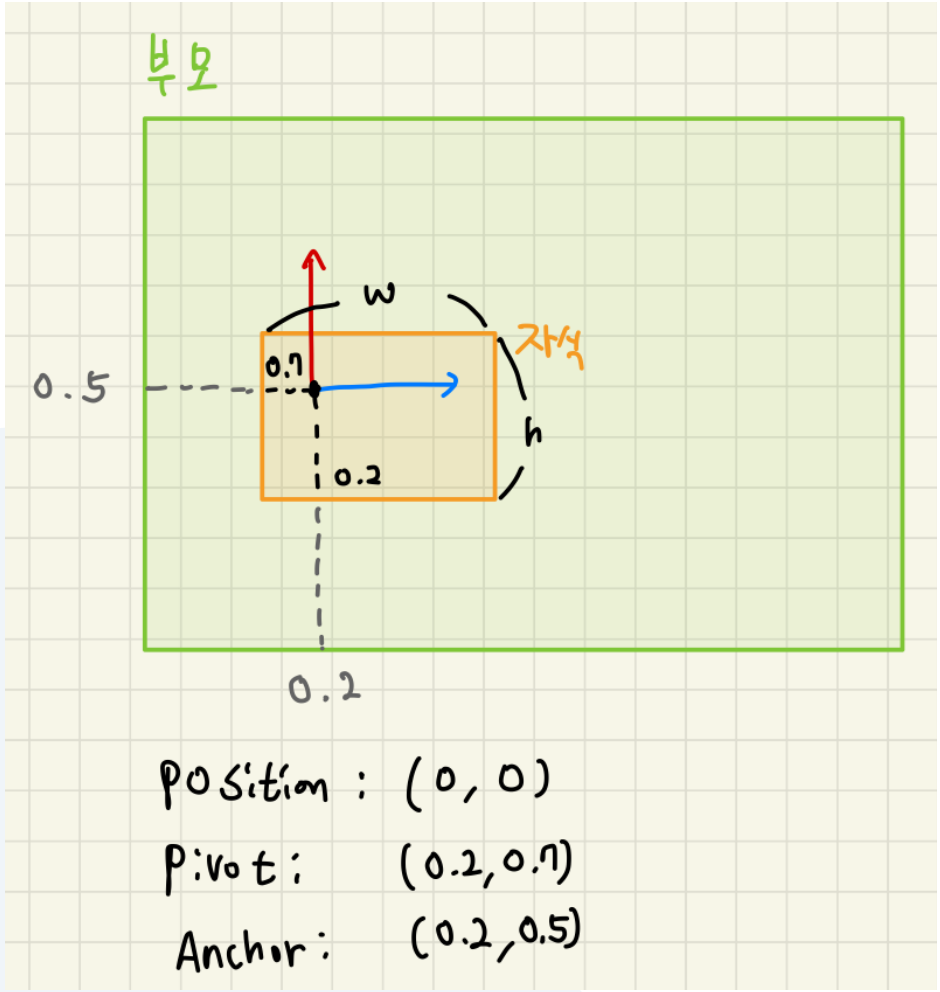
Pivot은 내 기준에서 원점의 위치입니다.

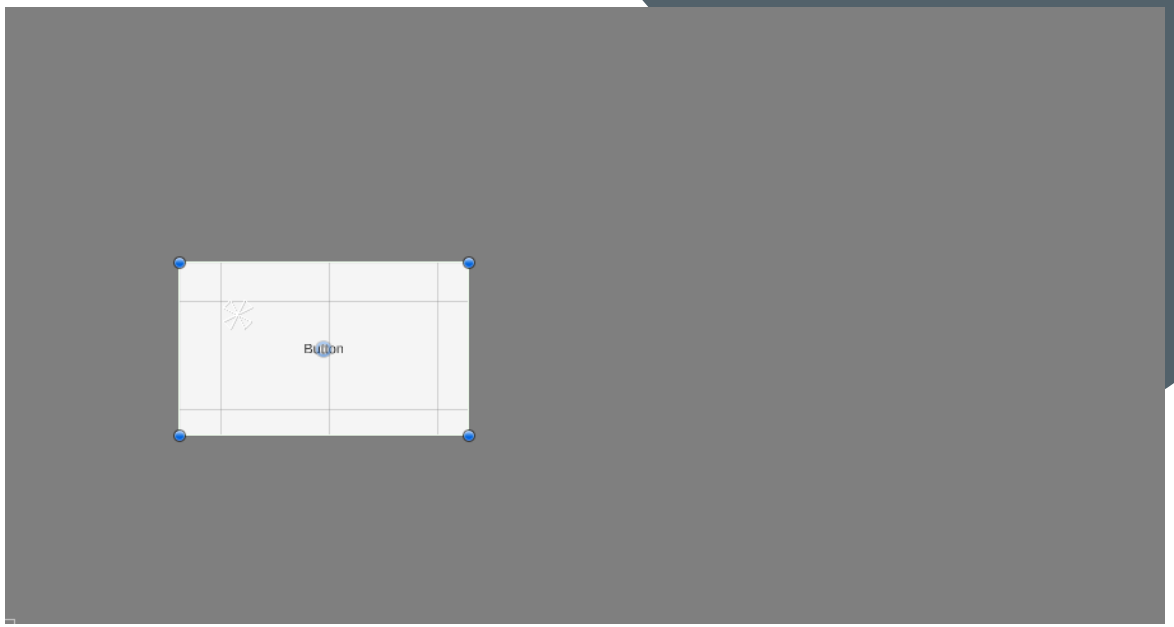
Anchor은 부모 기준에서 원점의 위치입니다.

Position은 원점 기준에서의 좌표 위치입니다.

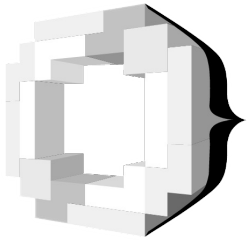


레이아웃

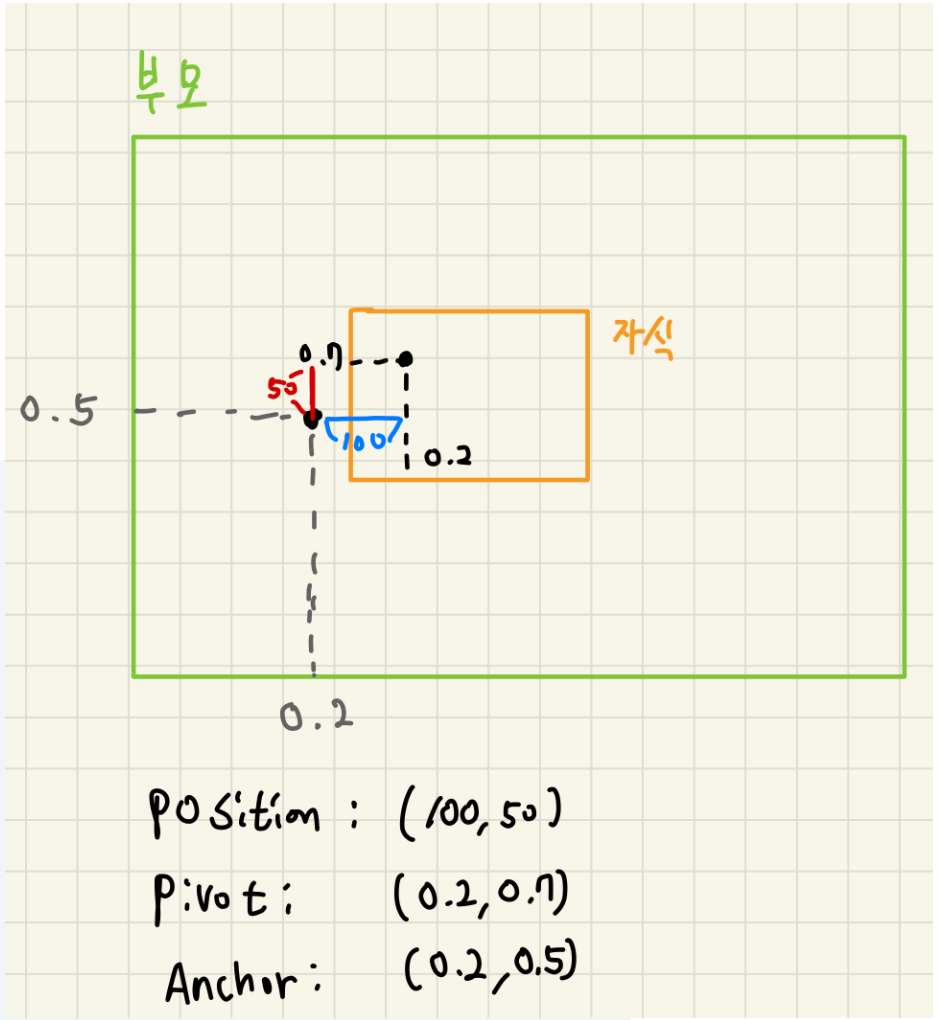

부모
w
자식
0.5
0.7
h
0.2
0.2
position : (0, 0)
pivot : (0.2, 0.7)
Anchor : (0.2, 0.5)

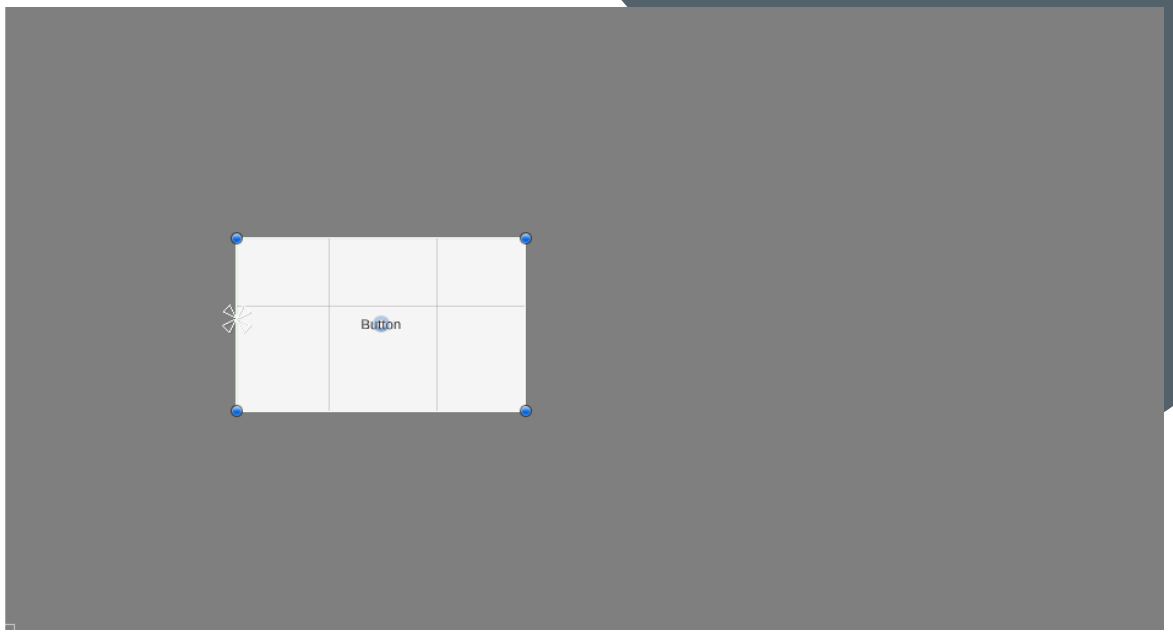

Button

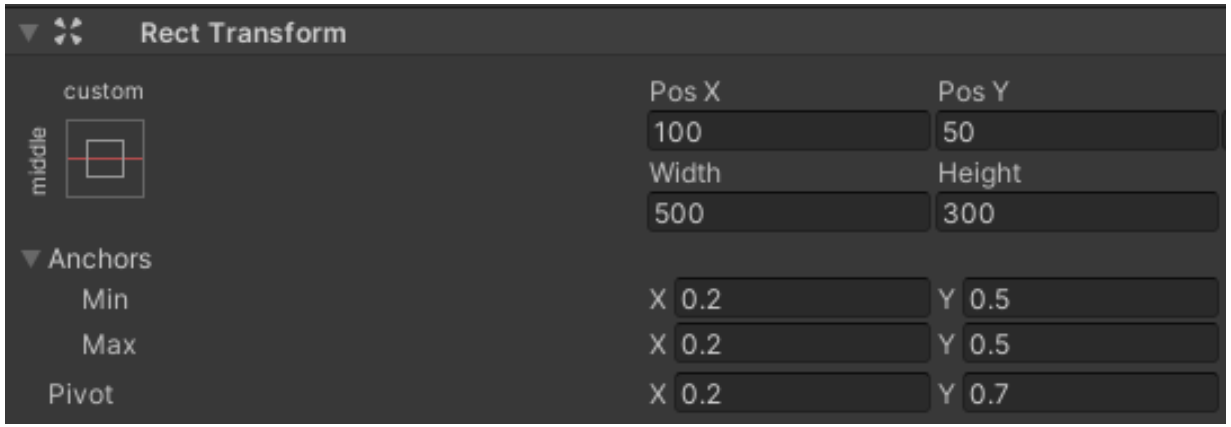

Rect Transform
custom
middle
Pos X
0
Pos Y
0
Width
500
Height
300
Anchors
Min
X 0.2
Y 0.5
Max
X 0.2
Y 0.5
Pivot
X 0.2
Y 0.7

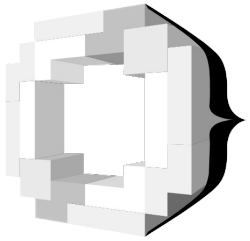


레이아웃


부모
자식
0.5
0.7
50
100
0.2
0.2
position : (100, 50)
Pivot : (0.2, 0.7)
Anchor : (0.2, 0.5)

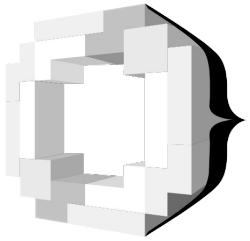

Button


Rect Transform
custom
middle
Pos X
100
Pos Y
50
Width
500
Height
300
Anchors
Min
X 0.2
Y 0.5
Max
X 0.2
Y 0.5
Pivot
X 0.2
Y 0.7



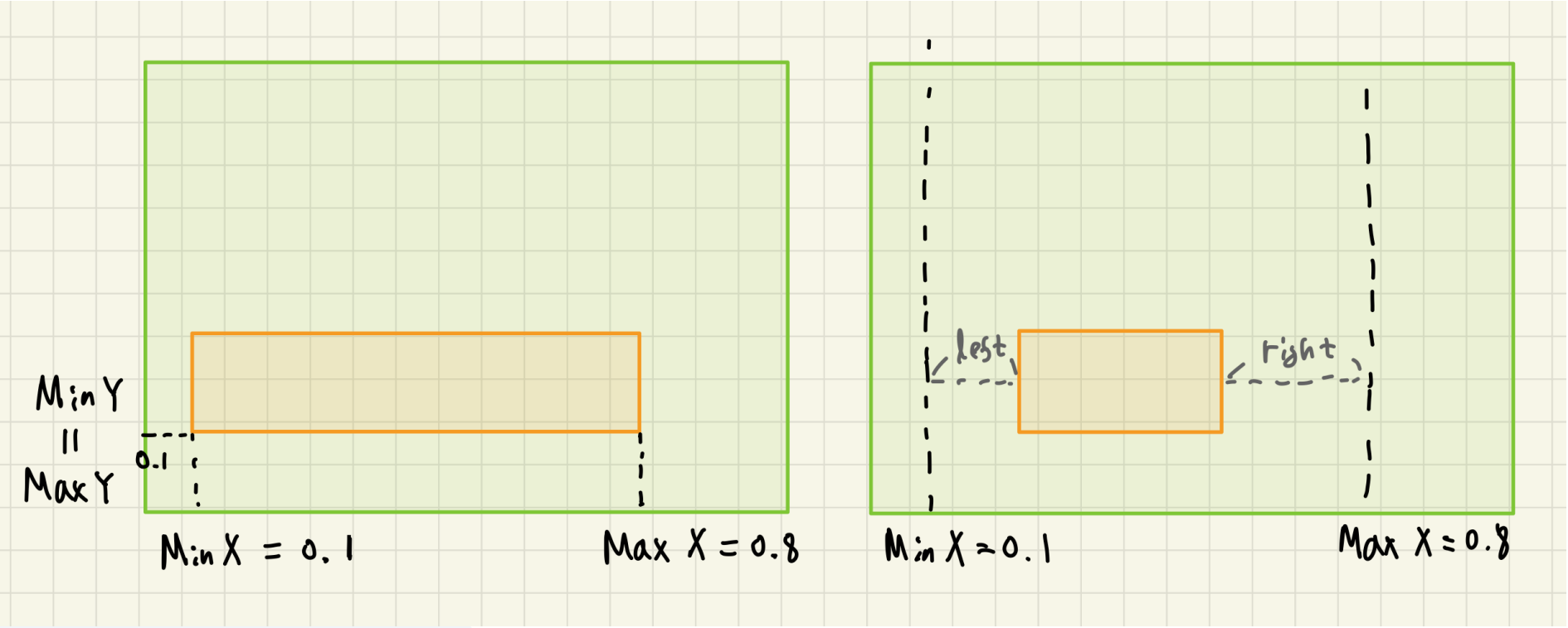
레이아웃

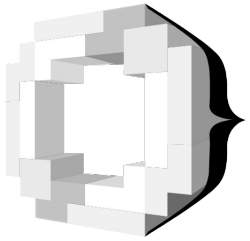
- Anchor에는 삼각형 네 개로 그 위치가 표시됩니다.
- Anchor에는 Min, Max값을 설정할 수 있습니다.
- Min, Max 값이 같으면 좌표값으로 위치를 정할 수 있습니다.
- Min, Max 값이 다르면 Left, Right, Top, Bottom 등
  부모에 대한 상대 값으로 좌표와 Width, Height가 정해집니다.



레이아웃

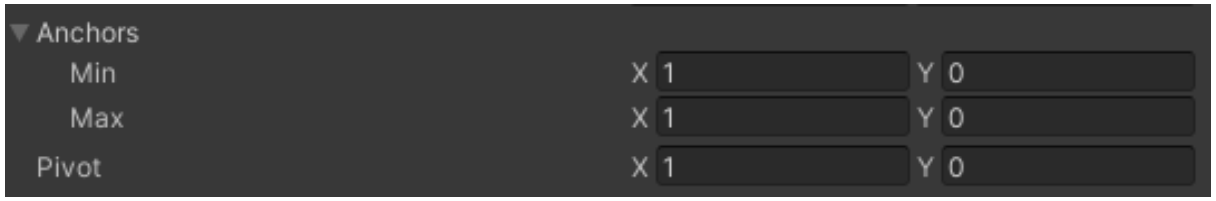
- Anchor의 Min, Max 값이 다를 때는 아래와 같아요.

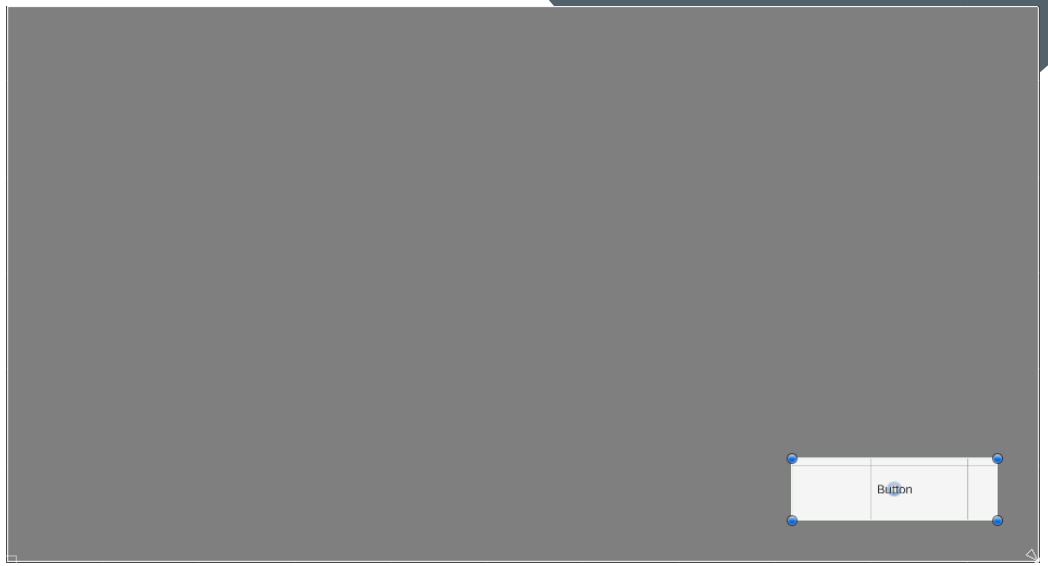




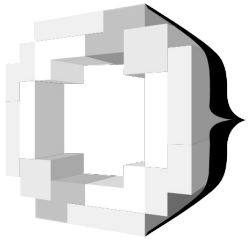
레이아웃

- 버튼을 오른쪽 아래에 배치해볼까요?



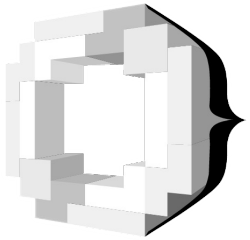


이제 위치와 너비, 높이는 Anchor, Pivot으로 정해진 원점에 대해 변경됩니다.



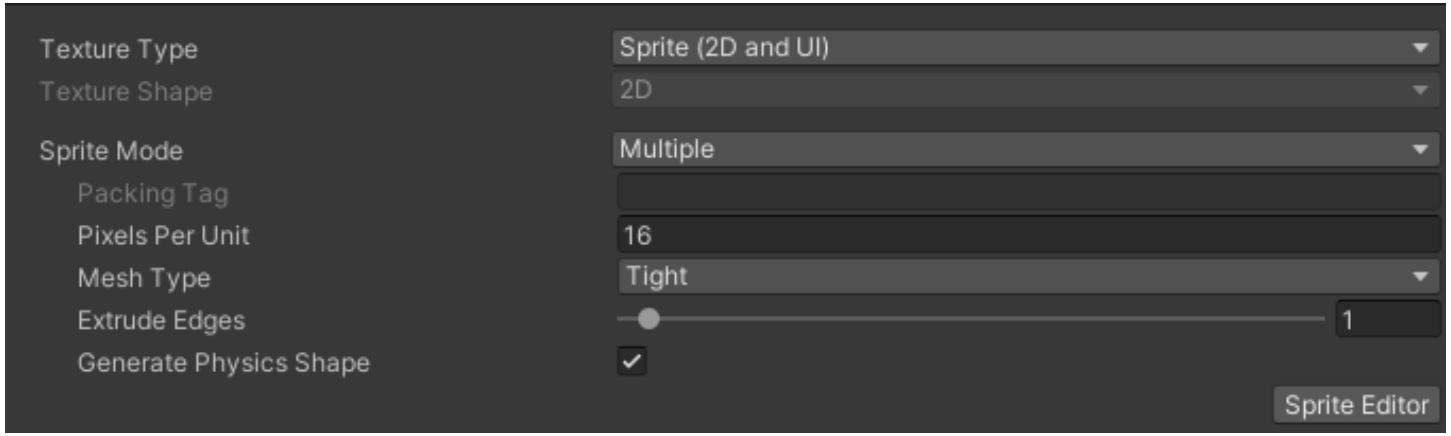
버튼 꾸미기

- 버튼이 조금 밋밋하네요. 에셋을 사용해서 꾸며줍시다.

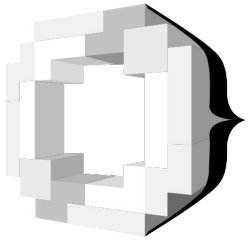


버튼 꾸미기

- Sprites 폴더의 UI - 16x16 UI Tileset 에셋이에요

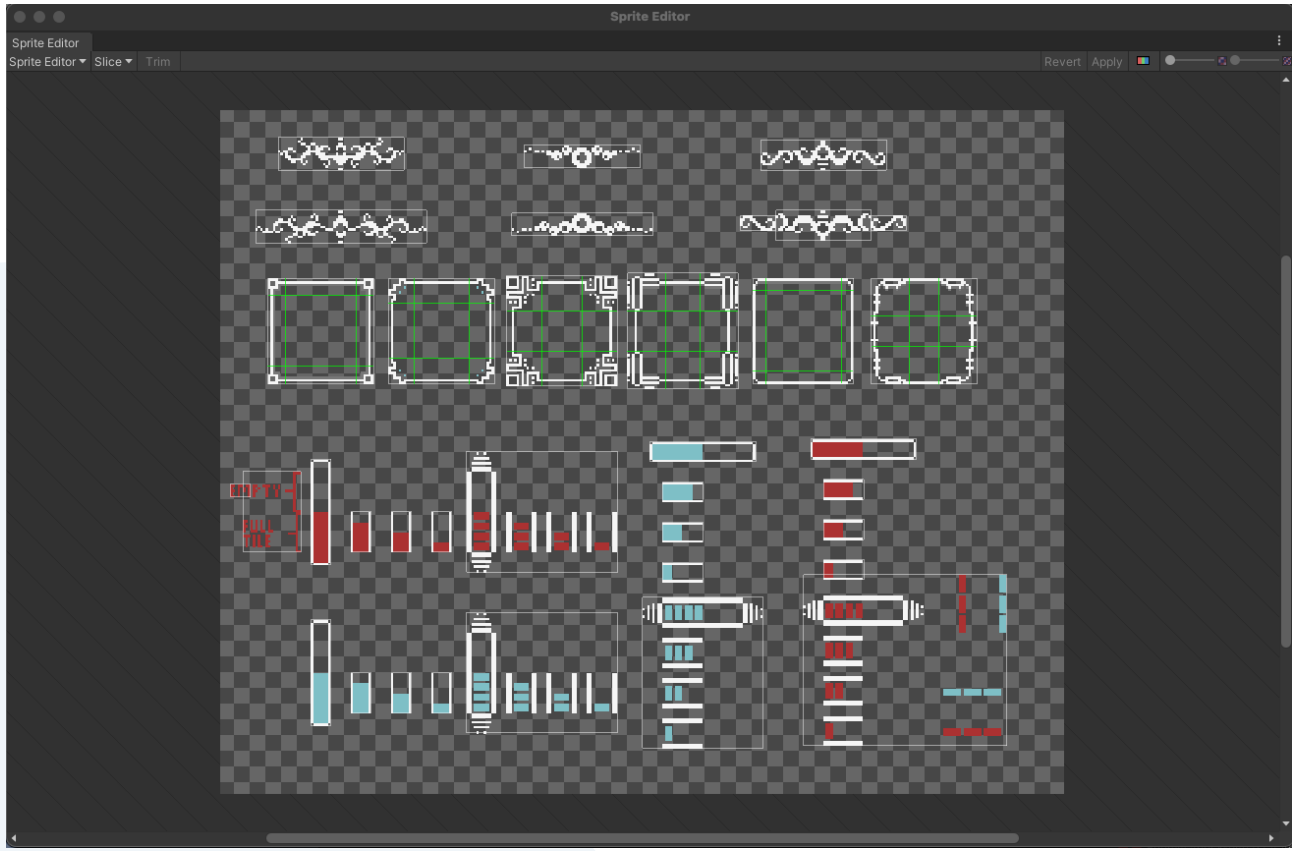


우측 하단의 Sprite Editor를 클릭해봅시다.

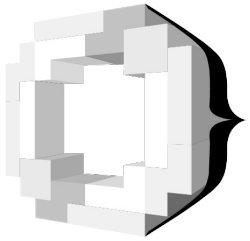


버튼 꾸미기

- 여러 스프라이트가 한 파일에 몰려있어서 나눠줘야해요

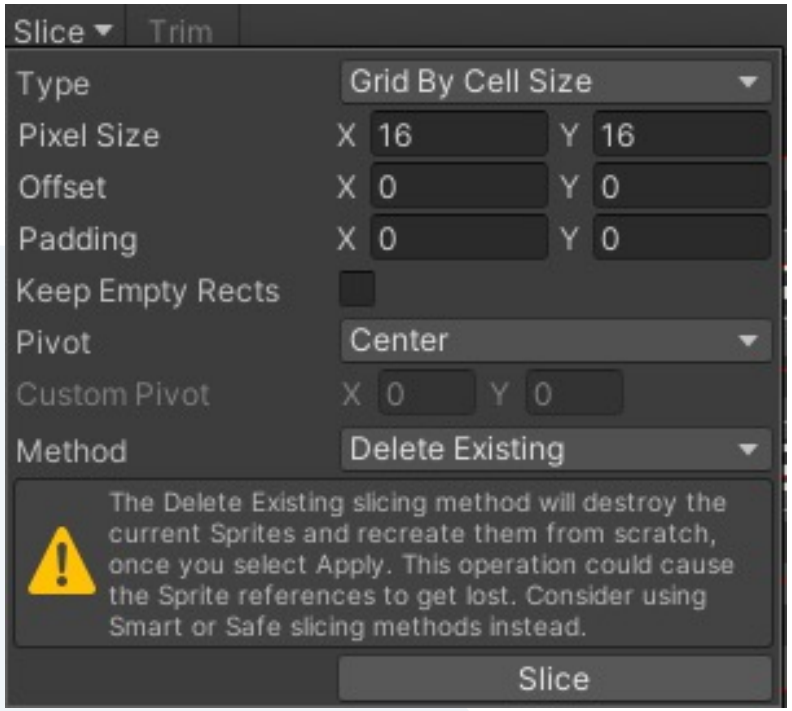


여기선 제가 미리 나눠줬어요.



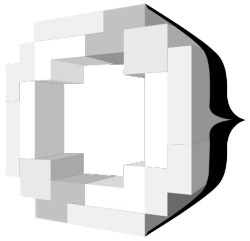
버튼 꾸미기

- Sprite Editor의 왼쪽 상단 Slice 버튼을 누르면 아래처럼 창이 떠요



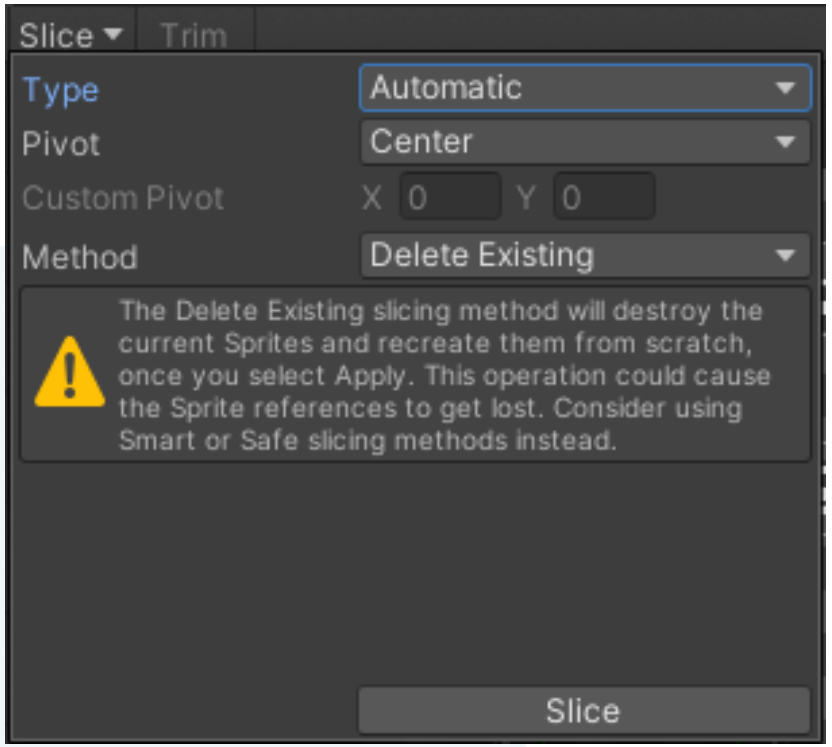
잘 만들어진 에셋이라면 스프라이트들이 규칙적으로
일정 픽셀마다 배치되어있어요.

만약 16x16 픽셀 단위로 배치되어 있다면
Grid By Cell Size 옵션을 선택해 주고
Pixel Size를 설정해 준 후 Slice를 누르면 끝이에요



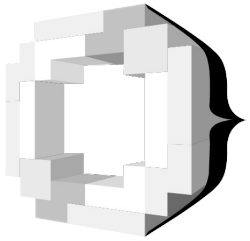
버튼 꾸미기

- 여기선 규칙적으로 배치가 되어있지 않으므로 직접 나눠줬어요



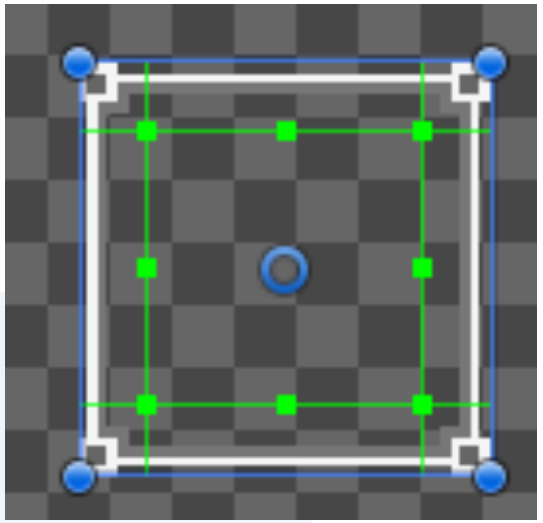
Automatic 옵션을 선택해주고
Slice를 시도해 볼 수 있어요.

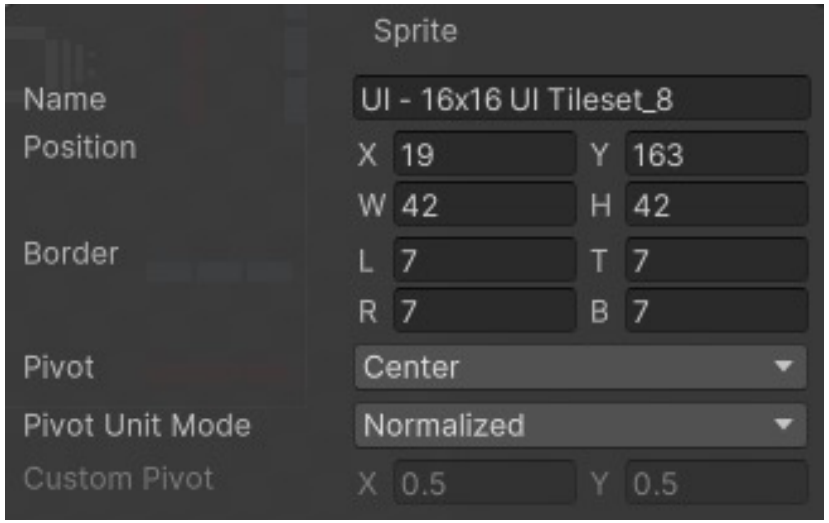
보통은 잘 안됩니다…
저는 일단 Automatic으로 Slice된 상태에서
수정을 해줬어요.



버튼 꾸미기

- Sprite Editor의 왼쪽 상단 Slice 버튼을 누르면 아래처럼 창이 떠요

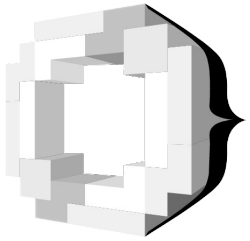




이렇게 쪼개 줄 수 있어요.

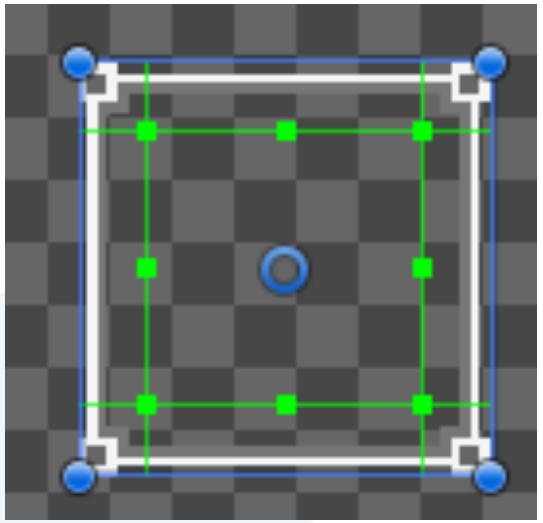
Position의 X, Y, W, H 값은 어떻게 쪼개 줄 것인가 하는 정보에요(회색 사각형)

Border은 9-Slicing을 위한 값들이에요(초록색 선들)

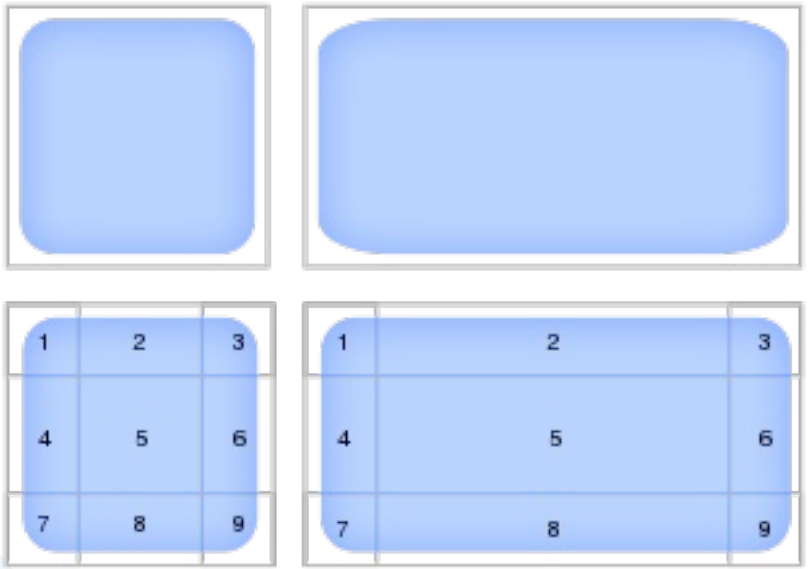


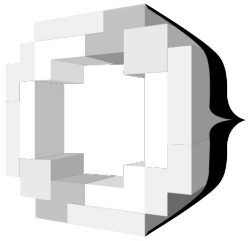
버튼 꾸미기

- 이 에셋들은 나중에 크기가 변할겁니다. 보통은 늘어날거에요



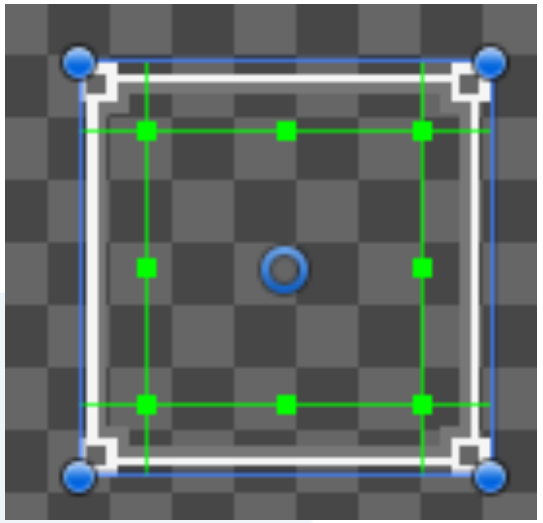
이 때 픽셀들이 깨지는 것을 방지하기 위한 방법 중 하나가
9-Slicing이에요

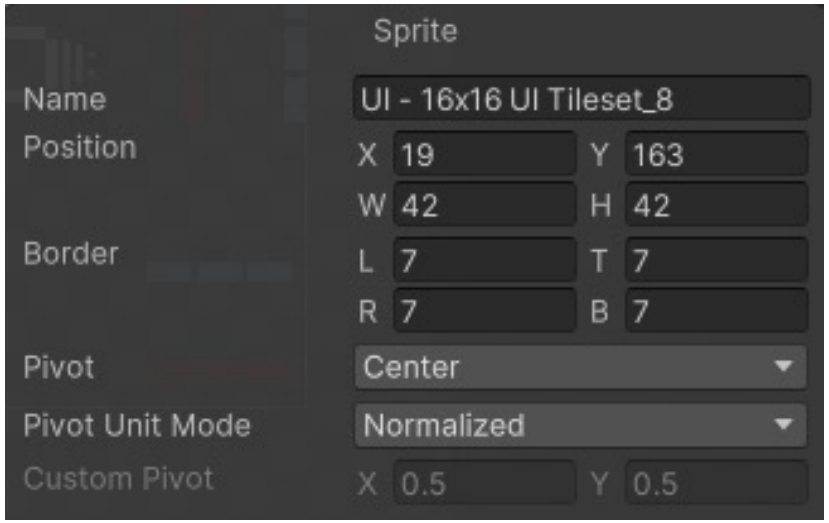


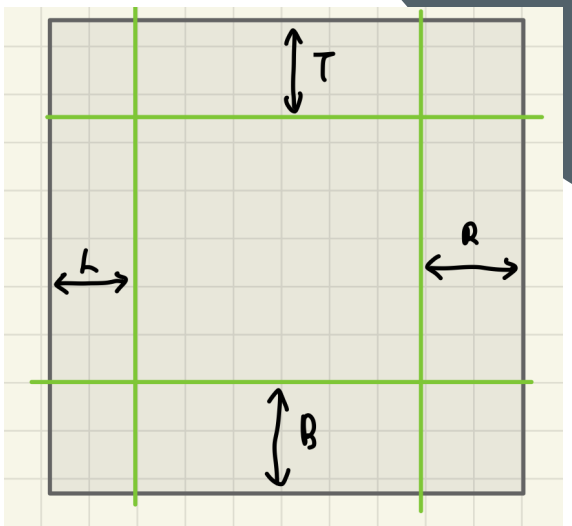


버튼 꾸미기

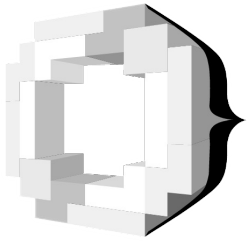
- 어떻게 적용되는지는 직접 적용해보며 살펴봅시다





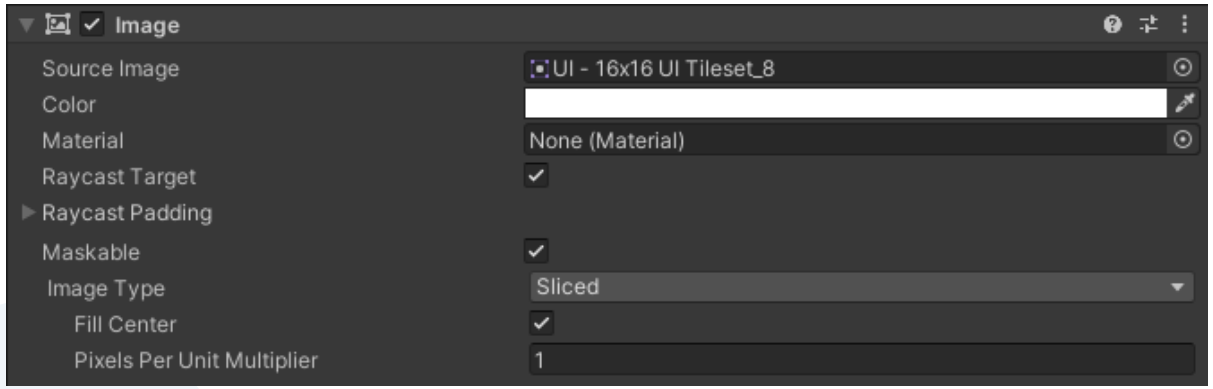


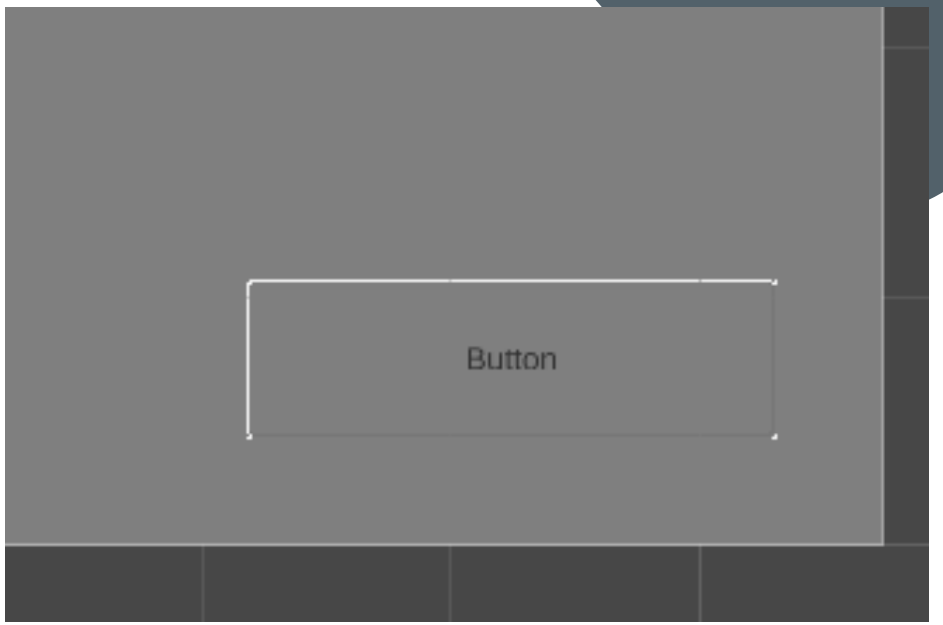
일단은 적당히 L, R, T, B 값을 설정해줍시다.
보통 나중에 크기가 변해도 픽셀이 깨지지 말아야 할 코너 부분을
Border 바깥에 위치하도록 값을 정해줍니다.



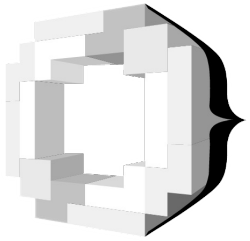
버튼 꾸미기

- Button의 Source Image를 아까 쪼개 준 Sprite 중 하나로 설정해줍니다.





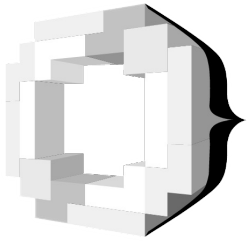
확실히 버튼의 모양이 우리가 원하는 모양은 아닌 것 같네요..



버튼 꾸미기

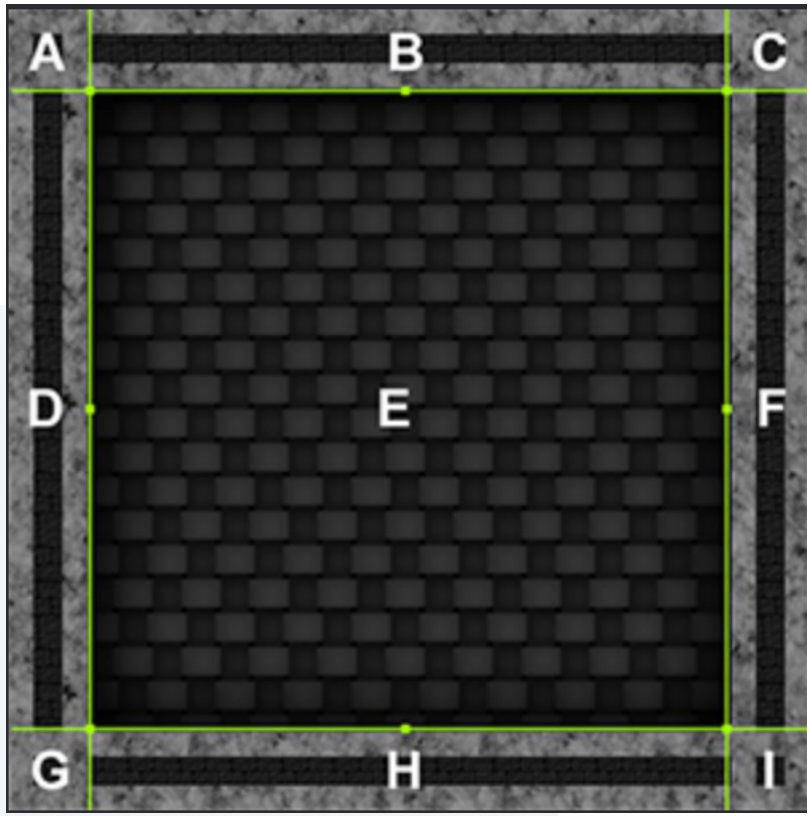
- 이 부분을 잘 설정해주어야 해요.

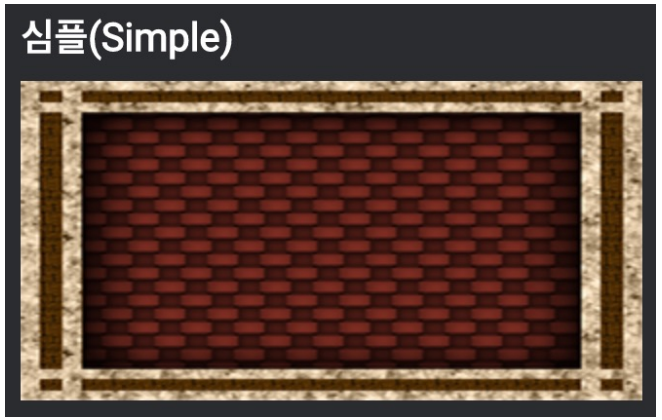
Image Type에는 Simple, Sliced, Tiled, Filled가 있어요.

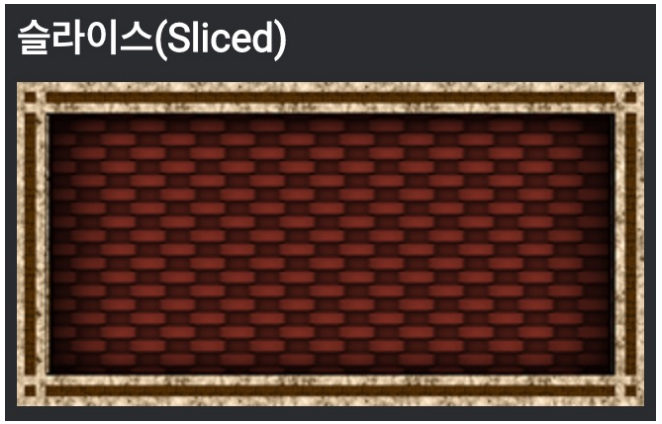


버튼 꾸미기

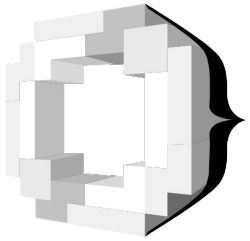
- 유니티 공식문서에 좋은 예제가 있네요





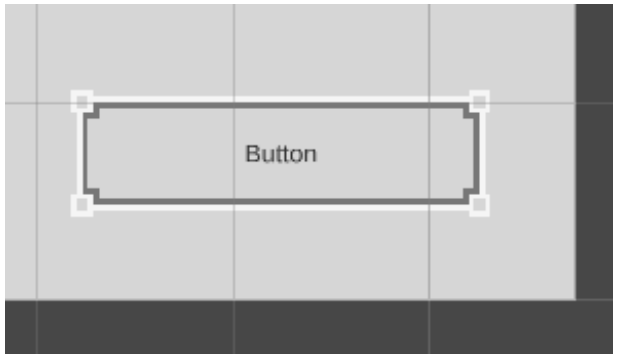


Filled는 원형으로
채우는 옵션입니다.
보통 스킬 쿨 타임
표시할 때 자주 쓰여요.



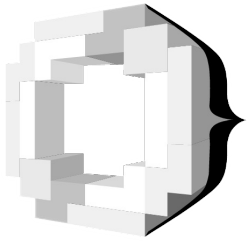
버튼 꾸미기

- 여기서는 Tiled를 사용하겠습니다.



Pixels Per Unit Multiplier를
3/16으로 설정해줬어요.

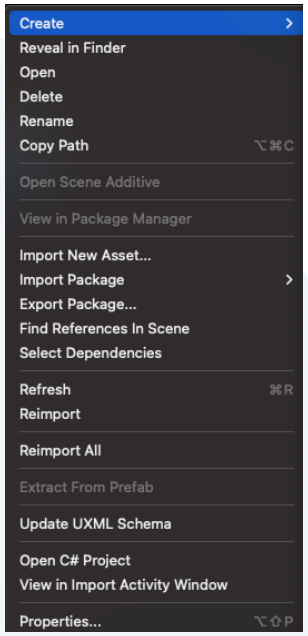
이는 원래 에셋의 크기가
(16 * 3) x (16 * 3)
정도였기 때문이에요.

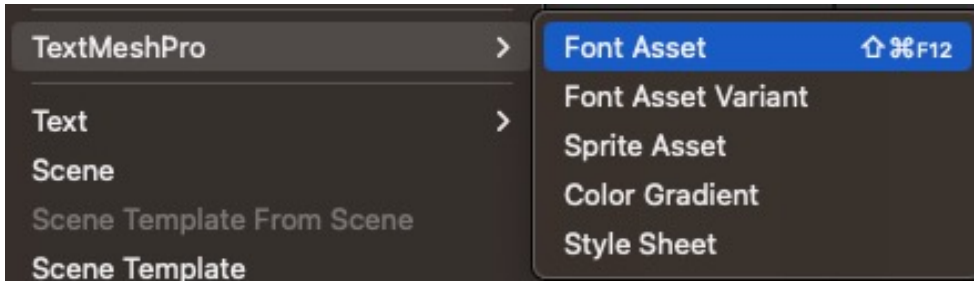


버튼 꾸미기

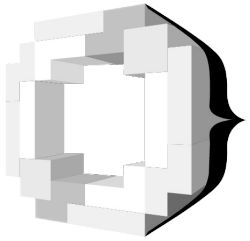
- Font도 변경해봅시다.

TextMeshPro 텍스트의 폰트를 변경해주려면 font 파일을
TextMeshPro Font 에셋으로 변환해줘야 해요.



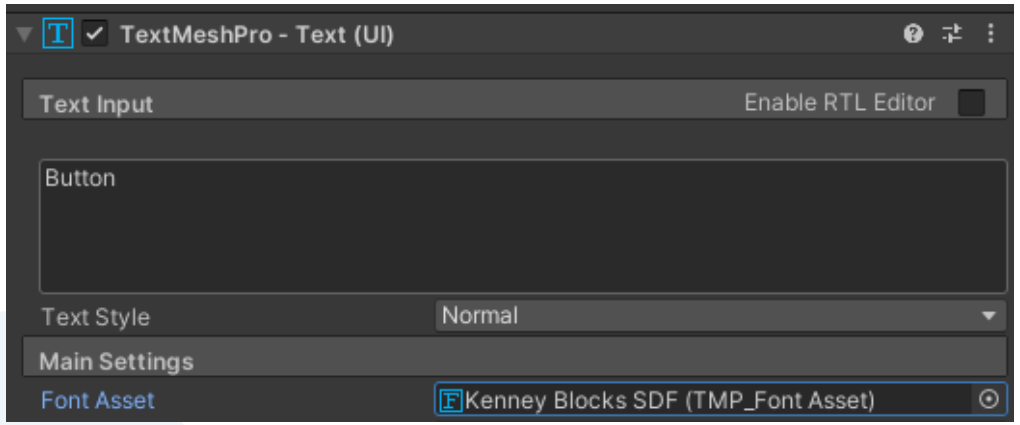


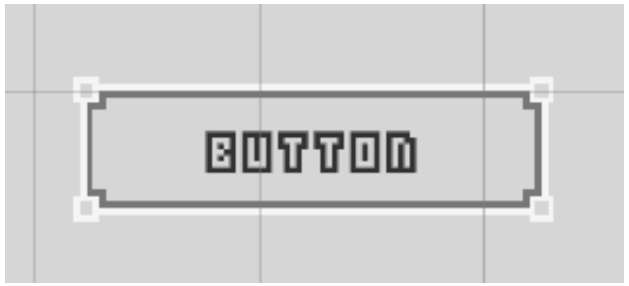
Font 파일을 우클릭하고 Create -> TextMeshPro -> Font Asset를
클릭하면 같은 폴더에 왼쪽과 같이 생긴 에셋이 생겨요!



버튼 꾸미기

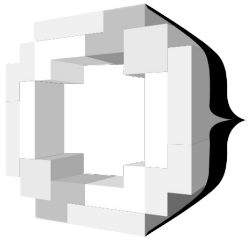
- Font도 변경해봅시다.





Text(TMP)의 Font Asset을 변경해주면…

잘 변경되었네요!

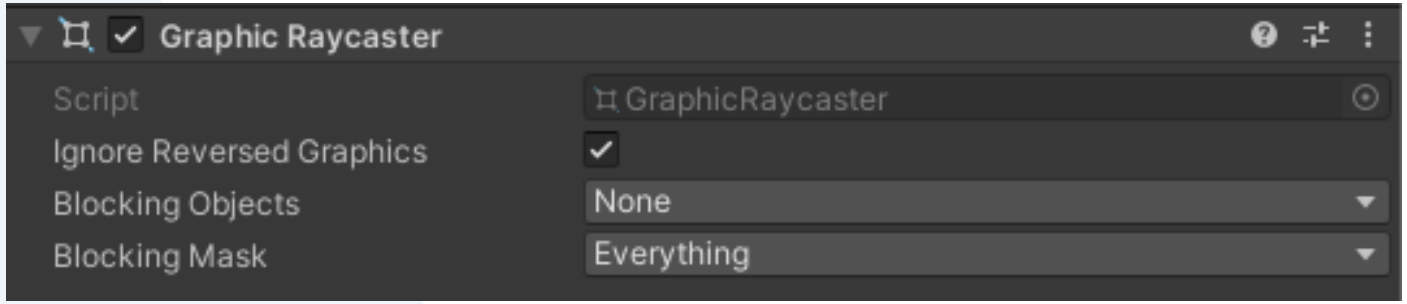


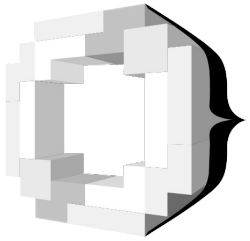
UI 이벤트

- UGUI에서 UI 이벤트는 Ray Cast(광선 쏘기)로 작동합니다
- Canvas의 Graphic Raycaster 컴포넌트,

  Canvas와 같이 생성되는 EventSystem에서 이벤트가 발생한 부분을

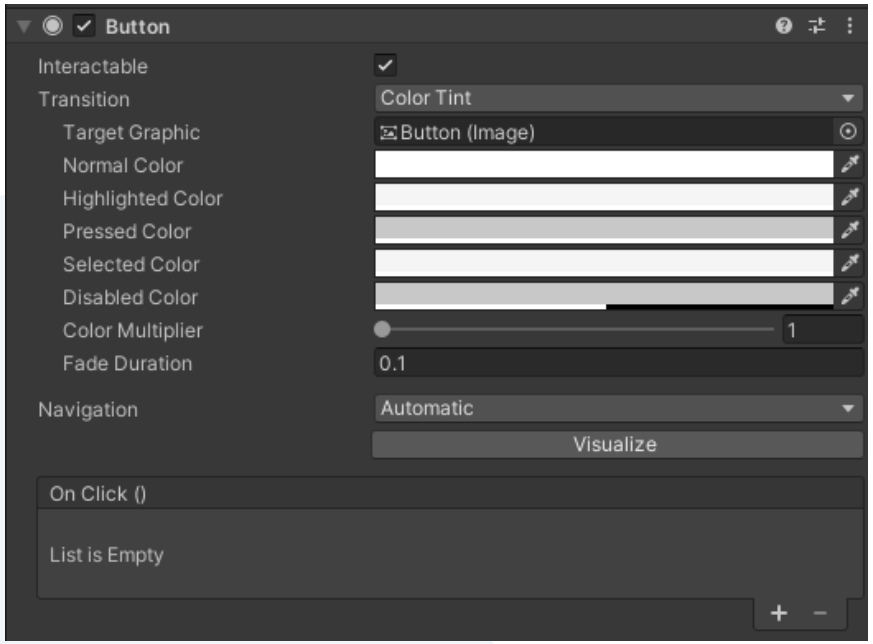
  Ray Cast를 통해서 알아내고 이벤트를 발생시켜요

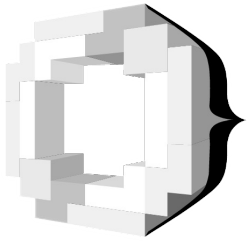




버튼 이벤트

- Button Component의 On Click에서 클릭 이벤트가 발생했을 때 실행할 함수를 추가할 수 있습니다.



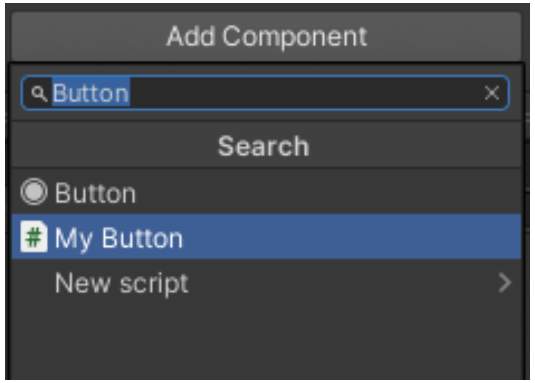


버튼 이벤트

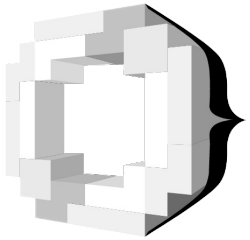
- Button Component의 On Click에서 클릭 이벤트가 발생했을 때 실행할 함수를 추가할 수 있습니다.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MyButton : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void SayHello() {
        Debug.Log("Hello World!");
    }
}
```

MyButton 스크립트를 생성해서,
SayHello라는 함수를 하나 작성하고
이 스크립트를 Button 오브젝트에 붙여주겠습니다.

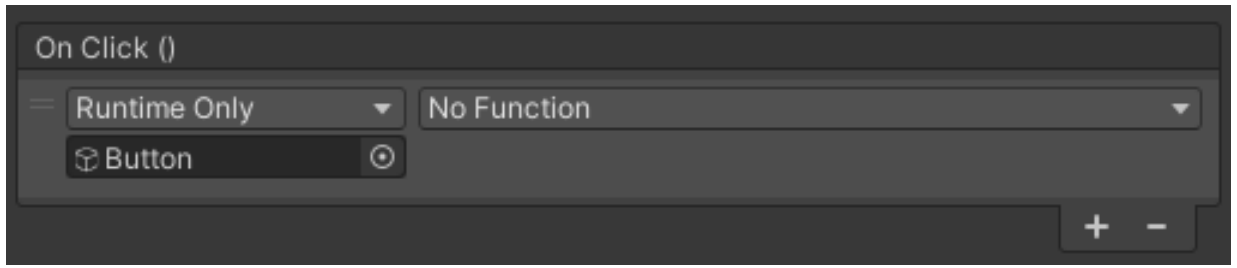


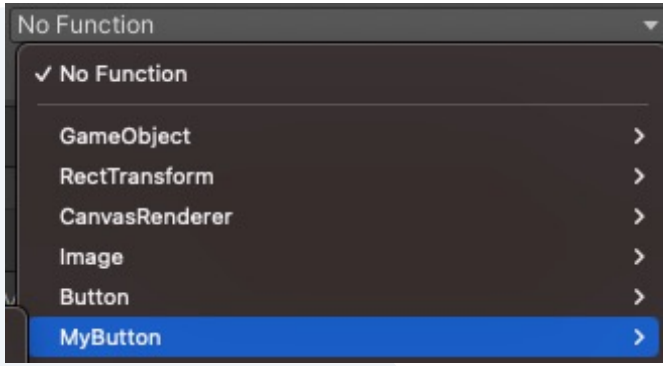
Inspector 맨 하단 Add Component 에서 쉽게 추가할 수 있어요!

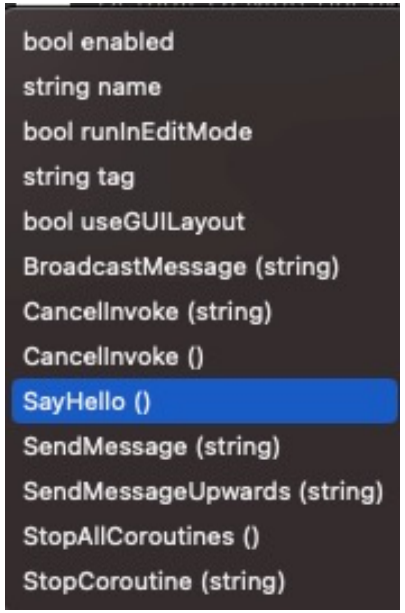


버튼 이벤트

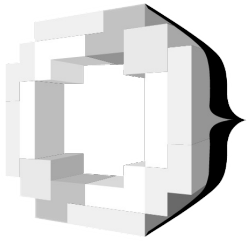
- OnClick의 + 버튼을 누르고 Object에 자기 자신을 드래그&드랍 해줍니다.





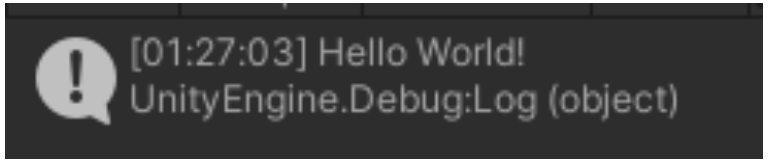


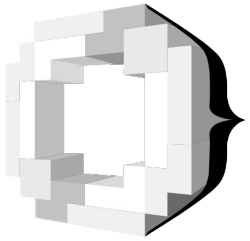
No Function을 클릭해서 MyButton을 선택하고, SayHello를 선택해줍니다.



버튼 이벤트

- 실행 후에 버튼을 클릭해보면 잘 될 거에요!

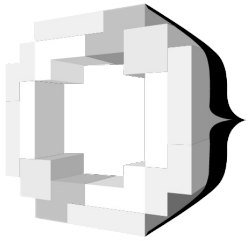




간단하게 알아보는 UI 최적화

- UGUI의 UI Element들은 Dirty Flag에 의해 Rebuild 된다!
  - Layout(Rect Transform) 변경 시 계층 구조 따라가면서 전부 다시 연산작업 실행 될 수 있음
  - 오브젝트 색깔, Material 등 변경 시 해당 오브젝트에 대해서만 다시 연산 됨

Layout Group의 사용을 지양하고, 직접 연산해주는 것이 좋을 수 있어요!
Layout 구성 시에 가능한 것들은 Canvas로 묶어 놓는 것이 좋아요(Canvas 단위로 Rebuild함)
UI에 애니메이션을 붙이면 매 프레임 당 Layout이 Rebuild될 수 있어요..

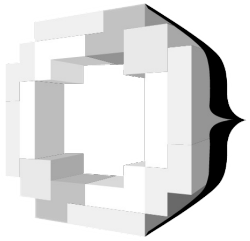


간단하게 알아보는 UI 최적화

- 카메라 영역에 있는 모든 Enabled 오브젝트들은
  Rendering 연산이 진행되고 있을 수 있다!

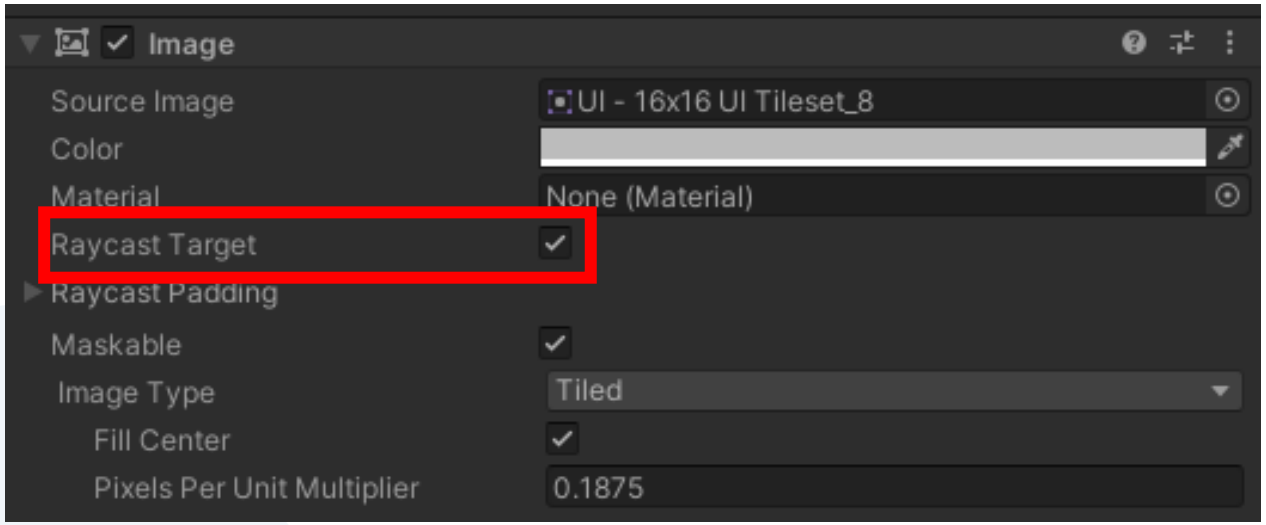
카메라 영역에 있는 Enabled(Inspector 체크표시!) 오브젝트들은 alpha값이 0,
그러니까 완전히 투명하더라도 Rendering 연산에 포함됩니다.
-> 투명해서 안 보이는 오브젝트들은 그냥 Disabled 시키는 방법이 있어요.

UI가 완전히 화면을 가리고 있더라도 그 UI 뒤에 있는 오브젝트들은 보이지 않지만 계속
Rendering 되고 있어요.
-> UI가 화면을 가리지 않고 있다고 가정해도 오브젝트들이 보이지 않게
   카메라를 멀리 치워버리는 방법이 있어요.

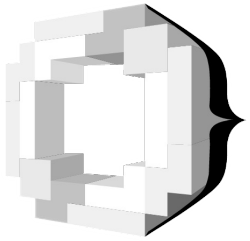


간단하게 알아보는 UI 최적화

- Ray Cast가 필요 없는 UI Element들은 해당 기능을 꺼주자



RayCast Traget에서 제외해주면, 해당 UI Element에 대해 RayCast 및 Event 연산을 하지 않기 때문에 성능상으로 이점이 생길 수 있습니다. (사실 차이가 그렇게 크진 않다고는 함)



해보면 좋아요

- 유니티 기능들은 자세히 안 다뤘어요
  - 유니티 공식 매뉴얼을 살펴봐요
  - https://docs.unity3d.com/kr/2021.3/Manual/UnityManual.html