

Unity

Game Engine

▶ 게임 엔진(Game Engine)이란?

- 2D나 3D 그래픽을 출력하기 위한 그래픽 엔진, 물리 엔진, 충돌 검출과 충돌 반응, 사운드 출력, 스크립트 작성, 애니메이션, 인공 지능, 네트워크, 스트리밍, 메모리 관리, 쓰레딩, 씬 그래프 등의 **게임 개발에 필요한 요소를 미리 만들어 둔 개발 도구**
- **다양한 플랫폼에서 실행** 할 수 있게 해준다
- **재사용**에 염두 하여 개발되기에 하나의 게임에 종속되지 않고 **여러 종류의 게임**을 만들 수 있게 해준다

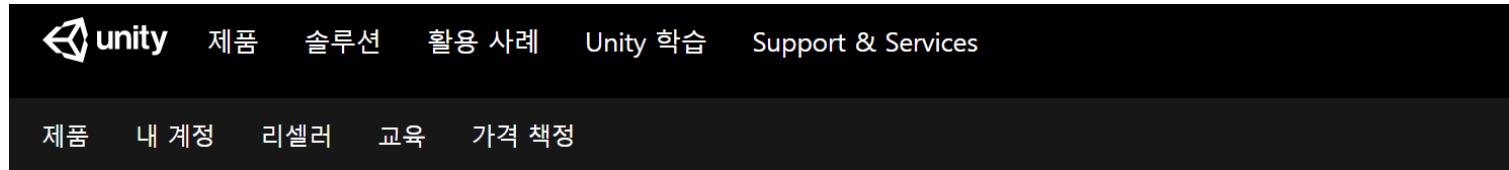
▶ 유니티 엔진(Unity Engine)

- 멀티 플랫폼용 게임 개발을 할 수 있게 해주는 **2D 및 3D 게임 엔진**
- **Component 디자인 패턴** 기반 엔진
- **물리 기반 엔진**
- 대부분의 모든 기능이 **개인 개발자**(지난해 매출 또는 자본금 10만 달러 미만)에게 **무료**
- 개발 환경과 실행 환경을 통합하여 **개발 도중 언제든지 실행하여 테스트가 가능**
- **C#, java** 스크립트를 사용하여 개발
- 세계적으로 사용자가 많아 **개발을 위한 자료(레퍼런스)**가 많다
- **Asset Store**에서 **자료(리소스)**를 구하기 쉽다

Unity 개발 환경 만들기

▶ 다운로드

- Unity Hub를 이용하면 **프로젝트 버전 관리가 용이**하다
- 주소 : <https://unity3d.com/kr/get-unity/download>



Unity 다운로드

세상에서 가장 사랑받는 2D/3D 멀티플랫폼 게임 및 인터랙티브 콘텐츠 개발 엔진 Unity를 다운로드하러 오신 것을 환영합니다!

먼저 본인에게 알맞은 Unity 버전을 선택한 다음 다운로드하세요.

Unity 선택 및 다운로드

Unity Hub 다운로드

여기에서 새로운 Unity Hub에 대해 자세히 알아보세요..

Unity 개발 환경 만들기

▶ 설치

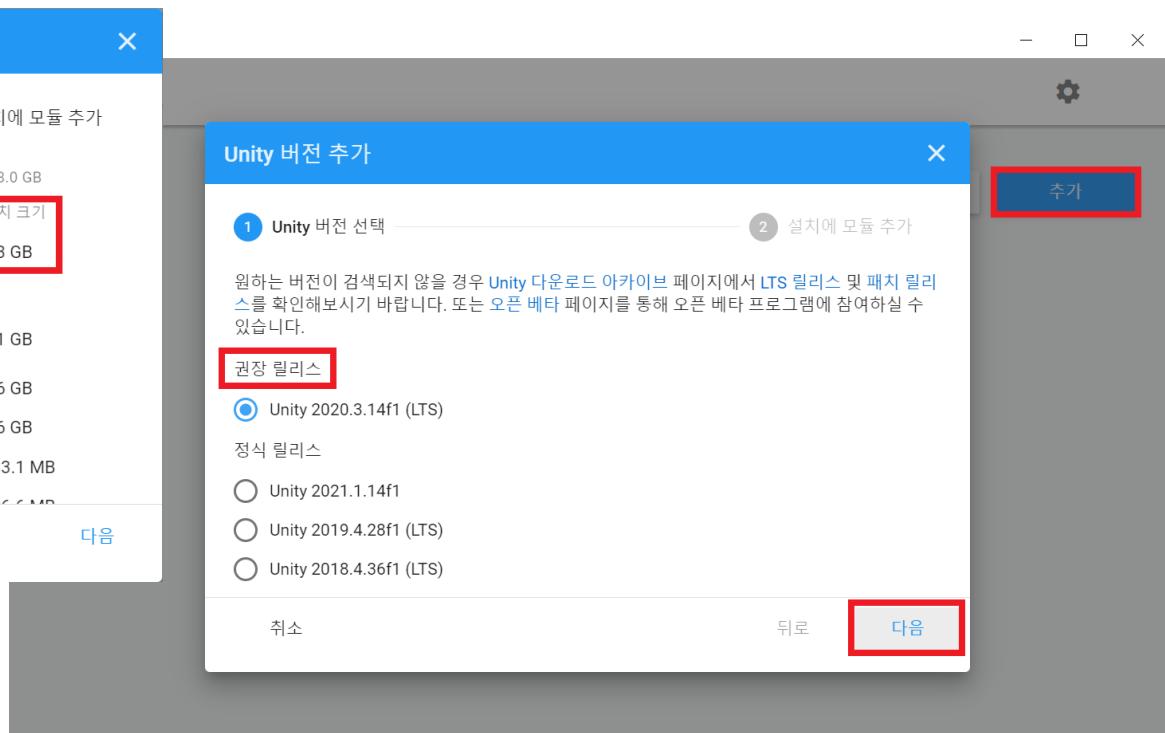
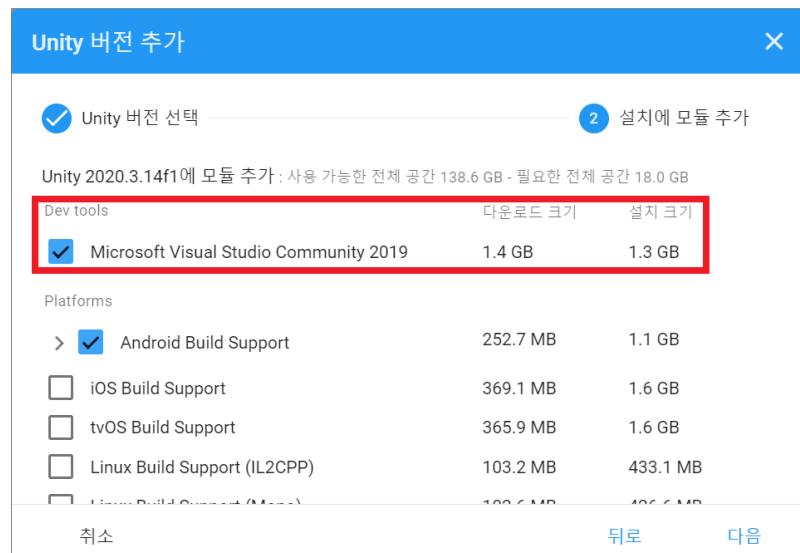
- Unity Hub를 설치 후 실행
- Unity 사용을 위한 라이선스 획득을 위해, 계정 생성 후 로그인 필수



Unity 개발 환경 만들기

▶ 설치

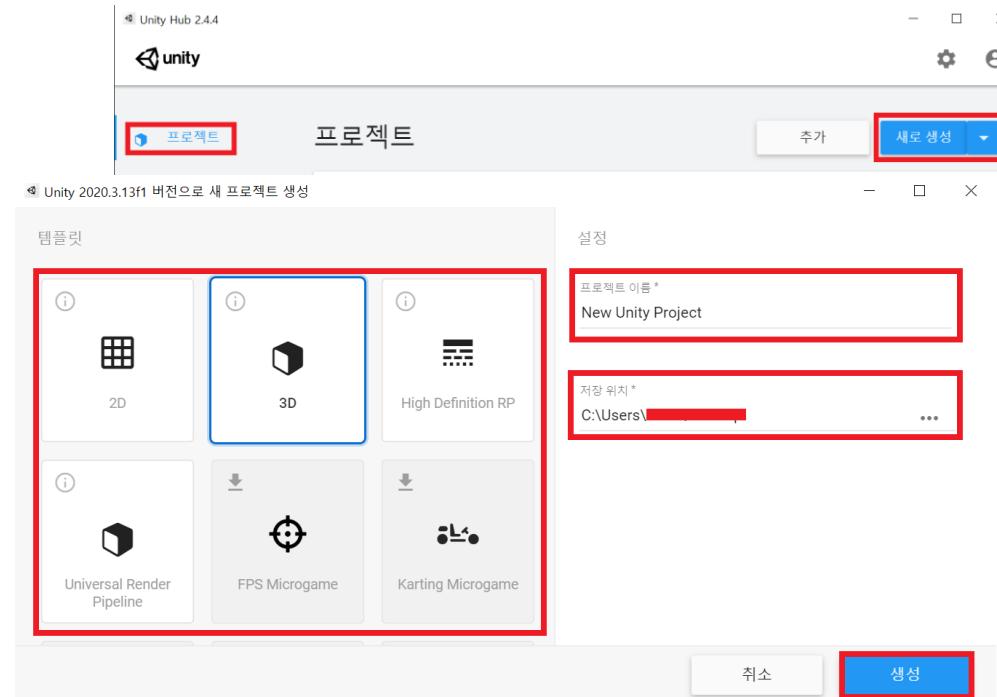
- 설치 탭의 [추가] 버튼을 선택하여 **권장 릴리스**를 설치
- Dev tools의 **Visual Studio Community** 체크 확인, 해당 버전의 **Visual Studio**를 이용하여 스크립트 관리가 가능
- 모바일 게임을 만들 경우 **Android, iOS Build Support**를 추가



Unity 개발 환경 만들기

▶ 프로젝트 생성

- [새로 생성]을 눌러 새로운 프로젝트를 생성
- [새로 생성] 옆의 화살표를 눌러 원하는 Unity 버전으로 프로젝트 생성이 가능하다
- 프로젝트명과 프로젝트가 생성되는 경로(저장 위치)에 한글이 있으면 안된다
- Unity에서 UI의 Label.text에 들어가는 문자 이외에는 한글을 사용하면 안된다



Unity 개발 환경 만들기

▶ SRP(Scriptable Render Pipeline)

- 렌더링 파이프라인을 스크립트로 제어할 수 있게 해준다
- 유니티가 제공하는 파이프라인에 국한되지 않고, 만들려고 하는 프로젝트에 최적화된 파이프라인을 구성할 수 있게 해준다

▶ HDRP(High-Definition Render Pipeline)

- SRP의 하나로 고해상도를 지원해 주는 렌더링 파이프라인

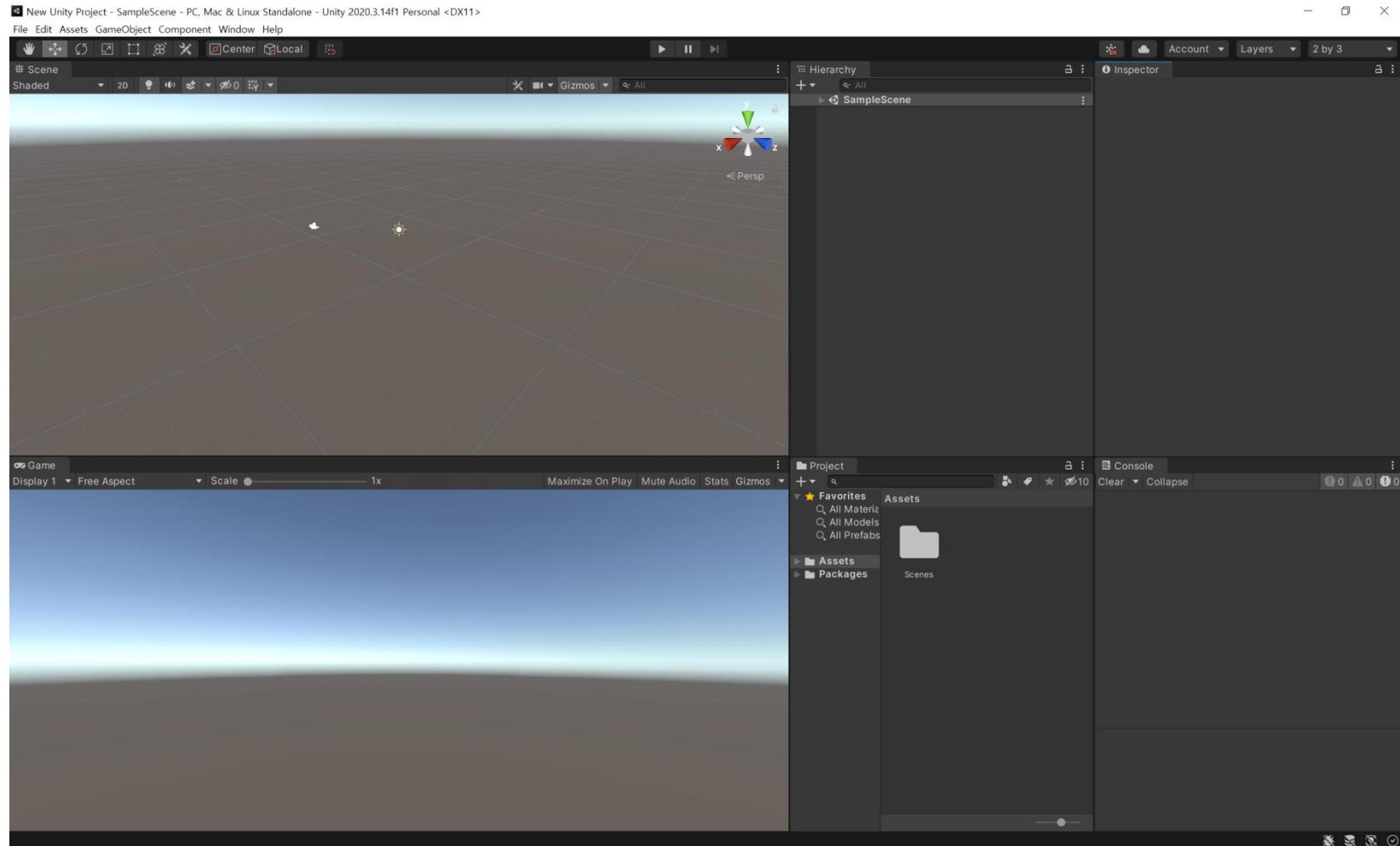
▶ URP(Universal Render Pipeline)

- 2019.3버전 이후 LWRP(Light Weight Render Pipeline)가 URP로 변경
- 경량화된 렌더링 파이프라인
- 모바일 디바이스에서 고품질의 그래픽을 구현이 가능

프로젝트 생성하기

템플릿을 **3D**로 선택
프로젝트 생성

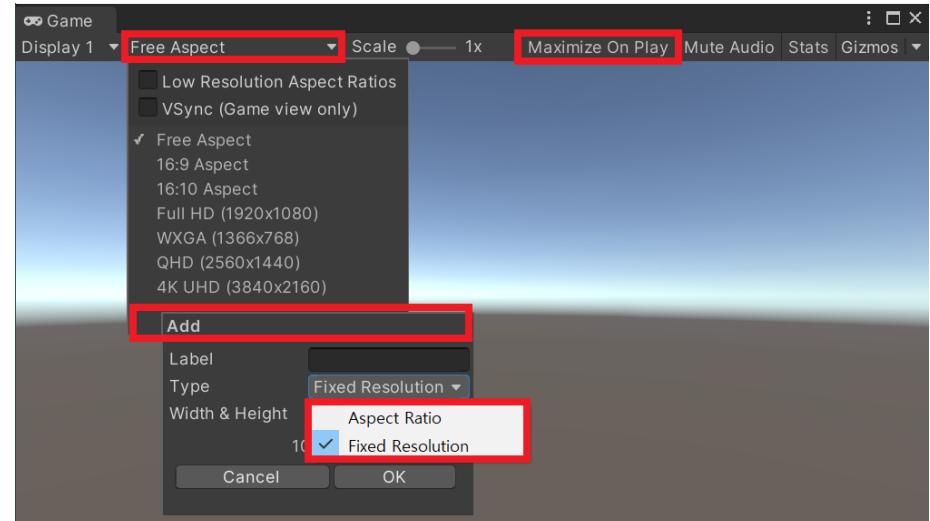
인터페이스



인터페이스

▶ Game(게임) 창

- 실제 애플리케이션의 실행 화면을 볼 수 있다
- 기본적으로 Game 창의 크기에 따라 화면의 비율이 변한다
- [Free Aspect]를 눌러 원하는 해상도, 화면 크기를 추가 및 설정
- Low Resolution Aspect Ratios : 구형 디바이스의 저 해상도 환경으로 에뮬레이트
- VSync : 게임 보기에 우선 순위 부여, 영상 녹화에 끊김 없이 재생될 수 있게 도와 준다
- Aspect Ratio : 해상도(비율)
- Fixed Resolution : 고정 크기
- [Maximized On Play]로 테스트 플레이 시, 최대화면으로 자동으로 변경되어 실행, 종료 하면 원래의 크기로 돌아온다



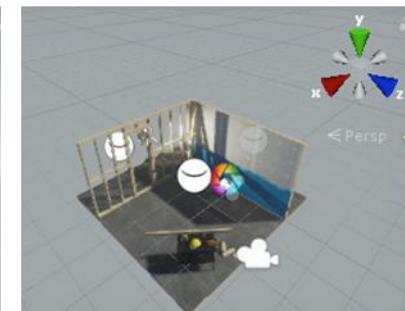
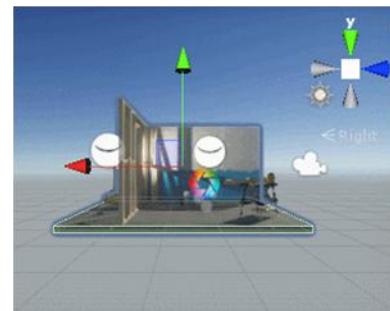
인터페이스

▶ Scene(씬) 뷰

- 게임 월드에 표기되는 모든 게임오브젝트의 유형을 선택 및 배치할 수 있다
- 일종의 맵(Map)이나 레벨(Level)에 해당한다
- Unity에서 작업하기 위해서 가장먼저 씬 뷰에서 오브젝트 선택, 조작 및 수정하는 스킬을 익혀야 한다

▶ Scene Gizmo(씬 기즈모)

- 씬 뷰의 우측 상단에 표시된다
- 씬 뷰 카메라가 현재 바라보고 있는 방향을 나타낸다
- 각 축을 클릭하여 씬 뷰 카메라가 해당 축을 기준으로 보도록 할 수 있다
- 씬 기즈모의 큐브 또는 아래의 문자를 클릭하여 씬 뷰 카메라 모드를 원근 모드(Perspective Mode), 직교 모드(Orthographic Mode)로 변경할 수 있다



인터페이스

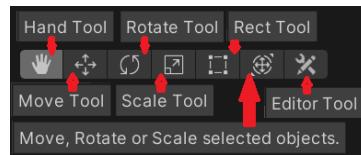
▶ Toolbar

- Unity 엔진에서 사용되는 도구의 모음



▶ Scene 편집 도구

- Hand Tool(단축키 : Q) : 씬 뷰 카메라 이동
➤ 마우스 휠을 클릭한 상태로 이동
- 화살표 방향키, Shift + 화살표 방향키
- Move Tool(단축키 : W) : GameObject를 화살표 방향으로 평행이동(Translate)
- Rotate Tool(단축키 : E) : GameObject 를 회전
- Scale Tool(단축키 : R) : GameObject 의 크기 조절
- Rect Tool(단축키 : T) : 2D GameObject 의 크기 조절
- Move, Rotate, Scale(단축키 : Y) : 이동, 회전, 크기 조절이 합쳐져 있다
- Editor Tool : 스크립트를 이용하여 커스텀 툴을 만들어 사용할 수 있다



인터페이스

▶ Transform Gizmo Toggle

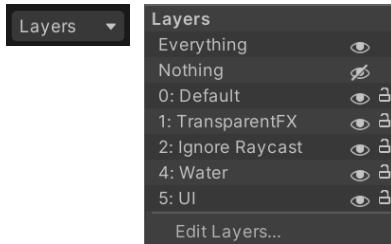
- 씬 뷰에 영향을 준다



- Pivot/Center : 선택한 오브젝트의 중심 좌표를 어떻게 표시할 것인가에 대한 옵션
- Pivot : 설정한 원점 좌표의 좌표 축이 표시
- Center : GameObject의 중앙 좌표 축 표시
- Global/Local : 선택한 GameObject의 좌표축을 글로벌 또는 로컬 좌표로 표시
- Grid Snapping : 선택한 GameObject를 1단위씩 움직이도록 한다

▶ Layers

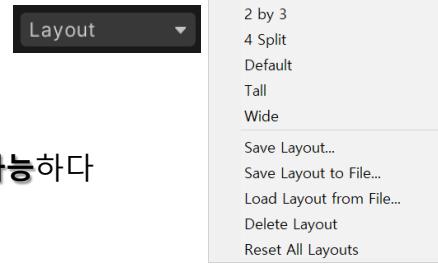
- 씬(Scene) 뷰에서 보이거나 숨기려는 레이어를 설정할 수 있다



인터페이스

▶ Layout

- 화면 구성을 원하는 구성으로 설정할 수 있다
- 몇 개의 기본 레이아웃이 설정되어 있어 쉽게 변경이 가능하다
- 내가 설정해 둔 레이아웃을 저장해두고 언제든지 가져와서 사용이 가능하다



▶ Unity Cloud

- 유니티 서비스 관련, 직접적인 개발과는 관련이 없지만 유저들의 성향을 분석하고 광고 수익을 극대화하거나 개발 생산성을 높이는데 유용한 툴을 제공



- Collab : 프로젝트 협업 서비스
- Cloud : 유니티 클라우드와 프로젝트를 연동
- Account : 유니티 계정 관리 웹 페이지 이동 링크

▶ Play Mode

- 게임 창에서 실제 애플리케이션이 어떻게 재생되는지 확인할 수 있다
- **플레이모드 실행 중 수정한 내용은 플레이 모드 종료 시에 모두 초기화** 된다, 따라서 **수정 내용은 플레이모드 종료 후 적용** 하여야 한다

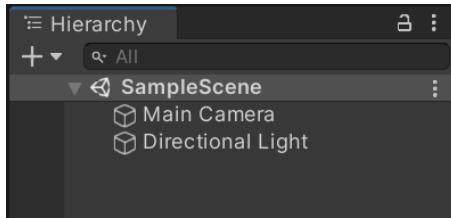


- Play : 현재 게임 씬(Scene)을 시작 또는 종료, Ctrl
- Pause : 현재 플레이 중인 씬(Scene)을 일시 정지
- Step : 게임을 한 프레임 단위로 진행

인터페이스

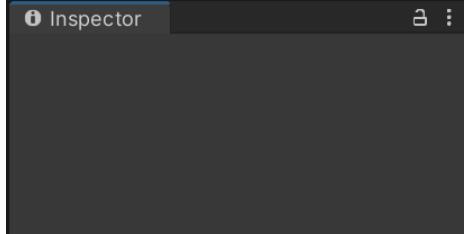
▶ Hierarchy(하이어라키) 창

- 현재 씬(Scene)에서 사용되는 모든 오브젝트를 나열한다
- 계층 구조로 이루어져 있어 화살표 버튼을 이용하여 펼치고 닫을 수 있다
- 쉽게 계층 구조 변경이 가능하다



▶ Inspector(인스펙터) 창

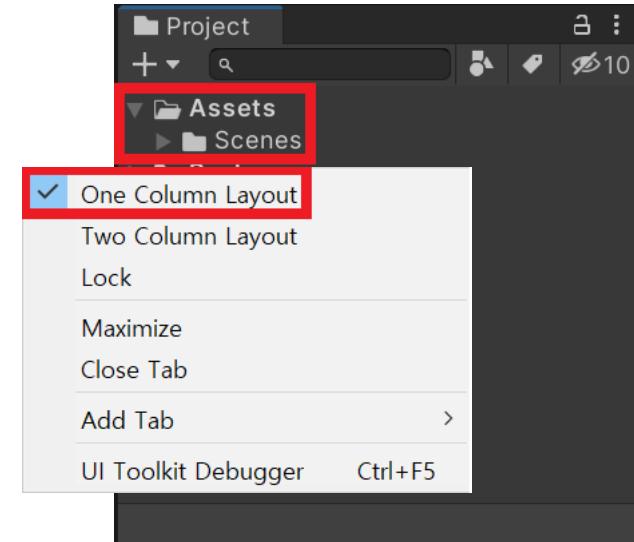
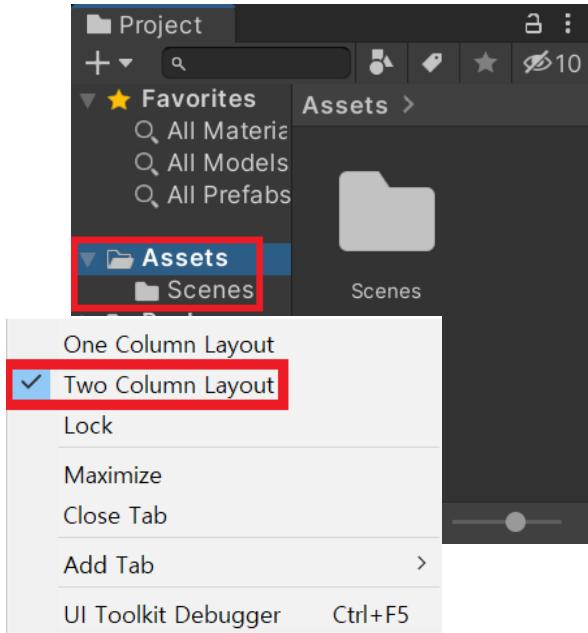
- 선택한 개체의 정보 확인 및 속성을 수정할 수 있다



인터페이스

▶ Project(프로젝트) 창

- 현재 프로젝트에서 사용될 모든 리소스(Asset)들을 정리 및 관리
- [...]버튼으로 두 종류의 Layout을 선택할 수 있다

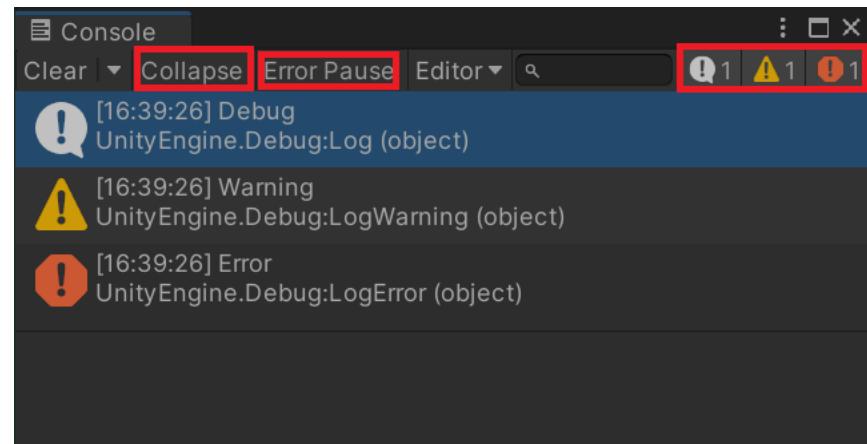
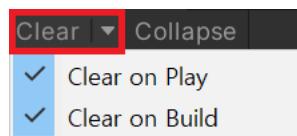


인터페이스

▶ Console(콘솔) 창

- Debug, Warning, Error 로그 출력
- 우측 상단의 각 로그 아이콘을 선택하여 원하는 형식의 로그만을 출력할 수 있다
- Collapse : 같은 내용의 로그를 묶어 개수로 표현
- Error Pause : LogError가 발생하면 플레이 상태를 일시 정지 시킨다
- Clear on Play : 플레이 시, 이전까지의 모든 로그를 제거
- Clear on Build : 프로젝트 빌드 시, 이전까지의 모든 로그를 제거

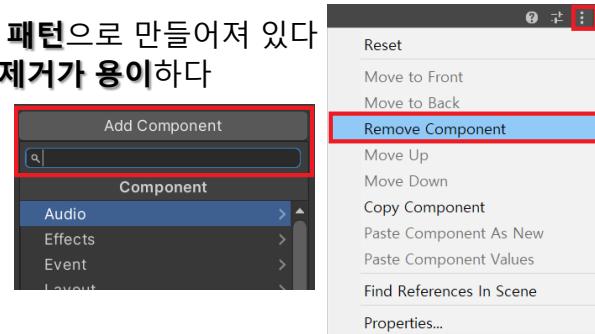
```
Debug.Log( "Debug" );
Debug.LogWarning( "Warning" );
Debug.LogError( "Error" );
```



컴포넌트

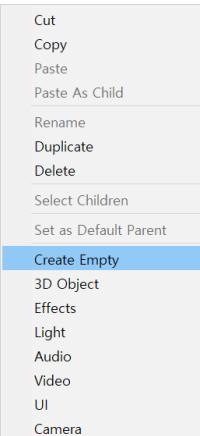
▶ 컴포지트(컴포넌트) 디자인 패턴

- 유니티는 각 오브젝트를 구성하는 요소를 **컴포넌트**로 관리하는 **디자인 패턴**으로 만들어져 있다
- 원하는 **컴포넌트**를 검색하여 손쉽게 **추가**하거나 필요없는 컴포넌트의 제거가 용이하다



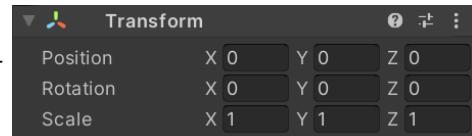
▶ GameObject

- **Hierarchy** 창에 표시되는 모든 오브젝트를 칭하는 용어
- 특정 오브젝트를 구성하는 빈 껍데기
- **Hierarchy** 창에서 마우스 오른쪽 클릭을 하면 보이는 메뉴에서 **Create Empty**를 선택
- 단축키 :
 - Ctrl + Shift + N(Create Empty)
 - Alt + Shift + N(선택 오브젝트의 자식 계층으로 생성)



▶ Transform

- 모든 **GameObject**가 기본적으로 가지고 있는 **컴포넌트**
- Position : 오브젝트의 **좌표** 정보
- Rotation : 오브젝트의 **회전** 정보
- Scale : 오브젝트의 **크기** 정보



컴포넌트

▶ Camera

- 월드를 실제 게임 화면에 보이도록 출력한다
- Hierarchy 창에서 마우스 오른쪽 클릭 메뉴에서 **Camera**를 선택하여 쉽게 추가가 가능하다
- 하나의 씬(Scene)에 하나 이상의 카메라 배치가 가능
- **카메라는 되도록 적게 배치**해야 한다

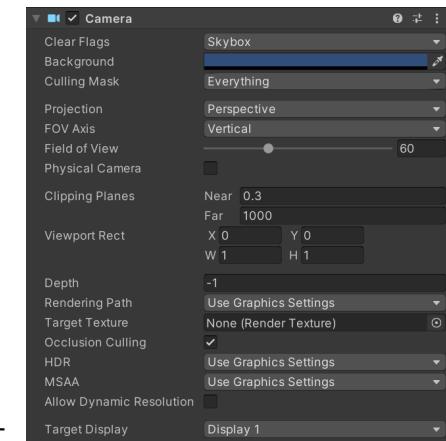
- **Clear Flag** : 어떤 도구로 **화면을 지울(Clear)** 것인지를 지정, **지정한 도구로 화면을 칠한다**
 - Skybox : 스카이 박스라는 배경 전용 쉐이더를 이용하여 지운다
 - Solid Color : **Background**에서 지정한 색으로 지운다
 - Depth Only : 여러 카메라의 영상을 **Depth 순서(오름차순)**로 덮어 써어 출력한다

- **Culling Mask** : 선택한 Layer만 해당 카메라에서 **출력**

- **Projection** : 원근(Perspective), 직교(Orthographic) 모드의 카메라로 출력
 - Field of View : 원근 카메라에서 사용, FOV Axis에 선택한 축을 이용하여 카메라의 시야각을 설정
 - Size : 직교 카메라에서 사용, 카메라의 크기를 변경

- **Clipping Plane** : 클리핑 영역을 위한 **근경(Near)과 원경(Far)의 영역**을 지정

- **Viewport Rect** : 카메라 뷰가 **화면에 보이게 될 영역**을 지정, 0 ~ 1값을 가진다



컴포넌트

- **Rendering Path** : 카메라에서 사용할 **렌더링 방법**을 지정
 - > Use Graphic Setting : Player Setting에 설정해 둔 **Rendering Path** 방식을 사용
 - > Vertex Lit : 해당 카메라로 렌더링된 모든 개체는 Vertex-Lit(정점 라이트)로 렌더링 된다, **가장 간단한 쉐이더 중 하나로 오래된 하드웨어에서 사용할 수 있게 설계 되었다**
 - > Forward : 픽셀당 빛을 계산하는 방법, **DX9에 대응하는 가장 일반적인 엔더링 방법**으로 유니티가 제공하는 렌더링 기능을 대부분 사용 가능하지만 **빛과 물체의 수만큼 계산량이 증가**한다
 - > Deferred Lighting : 모든 오브젝트의 조명 렌더링을 한 번에 처리하기 때문에 **조명의 수에 제한이 없다**, **Orthographic 카메라에서는 Forward로 동작 한다**
- **Target Texture** : Render Texture를 사용할 때 이용된다
- **Occlusion Culling** : **다른 오브젝트 뒤에 가려진 오브젝트가 렌더링 되지 않도록 한다**
- **HDR** : High Dynamic Range(**색상과 밝기를 사실적으로 묘사하는 기술**) 렌더링을 활성화 한다
- **MSAA** : 안티 엘리어싱 기법 중 하나인 **멀티 샘플링**을 사용한다
- **Allow Dynamic Resolution** : **동적으로 해상도를 조절하여 애플리케이션의 프레임 속도를 일관되게 유지되도록 한다**
- **Target Display** : 최대 8대의 모니터 중에 해당 **카메라가 출력할 모니터를 설정할 수 있다**(**PC, Mac, Linux 환경에서만 가능**)

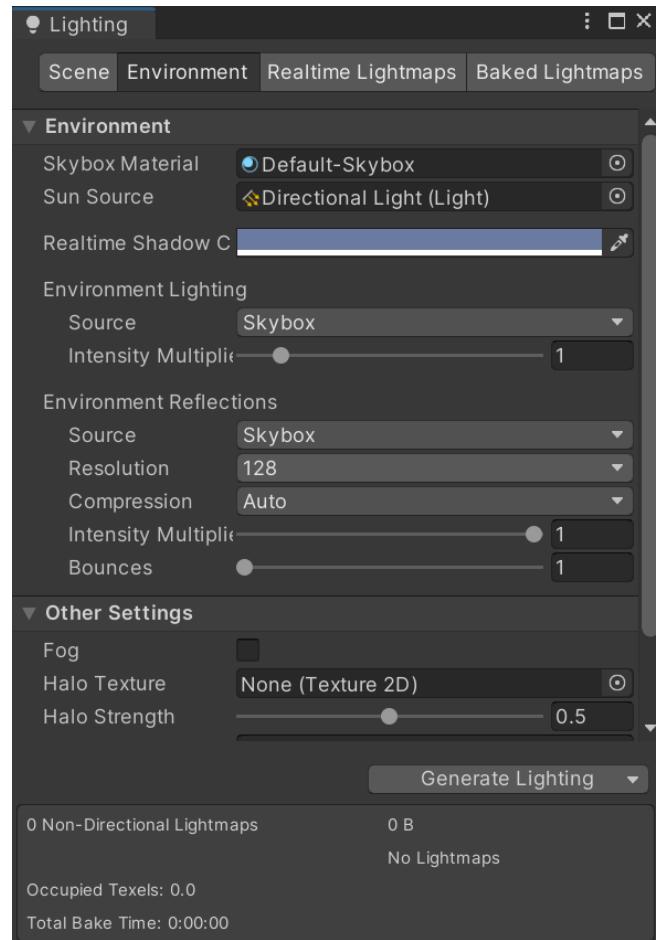
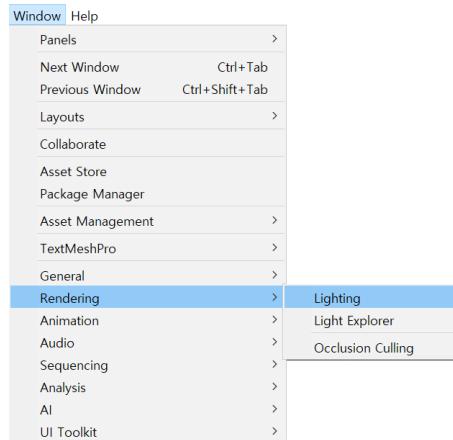
컴포넌트

▶ Skybox

- Windows => Rendering => Lighting => Environment
- Material의 Shader를 Skybox로 선택하여 생성
- 직접 원하는 스카이 박스를 만들어 적용 가능

▶ Material

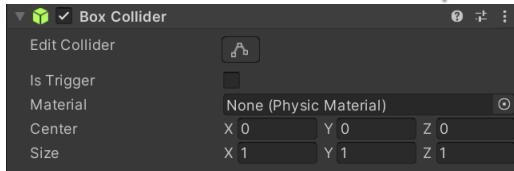
- Project 창에서 마우스 오른쪽 클릭 => Create => Material
- Shader에 따라 속성이 다르다
- 렌더링을 효과적으로 처리하기 위하여 물체의 표면에 대한 정보를 별도로 관리하기 위한 객체



Physic(물리) 작용

▶ Collider

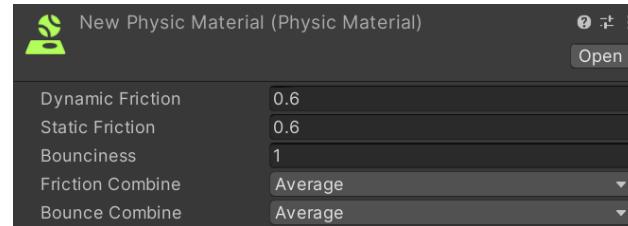
- 오브젝트(물체) 간의 충돌 처리를 하기 위한 영역
- 대표적으로 Box Collider, Sphere Collider, Capsule Collider가 많이 사용된다



- Edit Collider : 활성화하면 씬(Scene)에서 Collider의 크기 등을 수정 할 수 있다
- Is Trigger : 오브젝트 간의 물리적 충돌을 하지 않고 충돌 판정만을 할지를 설정
- Material : 해당 오브젝트에 물리적인 재질을 적용
- Center : 충돌 영역의 중심 좌표
- Size : 충돌 영역의 크기

▶ Physic Material

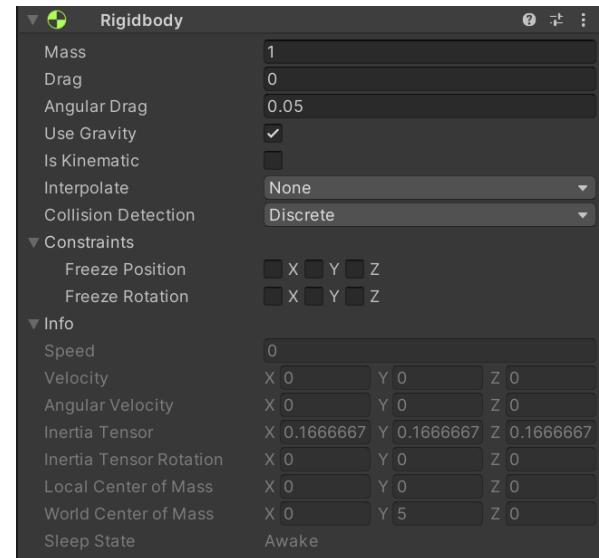
- Project 창에서 마우스 오른쪽 클릭 => Create => Physic Material
- Dynamic Friction : 이동 중에 사용되는 마찰력, 0 ~ 1의 값을 사용하며 0이면 얼음과 같은 느낌을 준다
- Static Friction : 정지된 상태에서 사용되는 마찰력, 0 ~ 1의 값을 사용
- Bounciness : 탄성도, 0 ~ 1의 값을 가지며 에너지 효율을 의미
- Friction Combine : 두 물체간의 마찰력 결과 처리 방법
- Bounce Combine : 두 물체간의 탄성력 결과 처리 방법



Physic(물리) 작용

▶ Rigidbody

- 오브젝트에 **강체(rigid body, 剛體)**를 적용한다
- **물리 제어로 동작**을 하기 위해서는 필수
- Mass : 오브젝트의 **질량**(기본 단위는 Kg), **낙하 속도에 영향 주지 않음**
- Drag : 힘에 의해 **이동할 때** 오브젝트에 **작용하는 공기 저항력**
- Angular Drag : 오브젝트가 토크 **회전할 때** 미치는 **공기 저항력**
- Use Gravity : 오브젝트에 **중력을** 적용
- Is Kinematic : 오브젝트는 **물리 엔진으로 작동하지 않고 Transform으로만 조작**, HingeJoint가 추가된 Rigidbody를 애니메이션처리 할 경우 유용하다



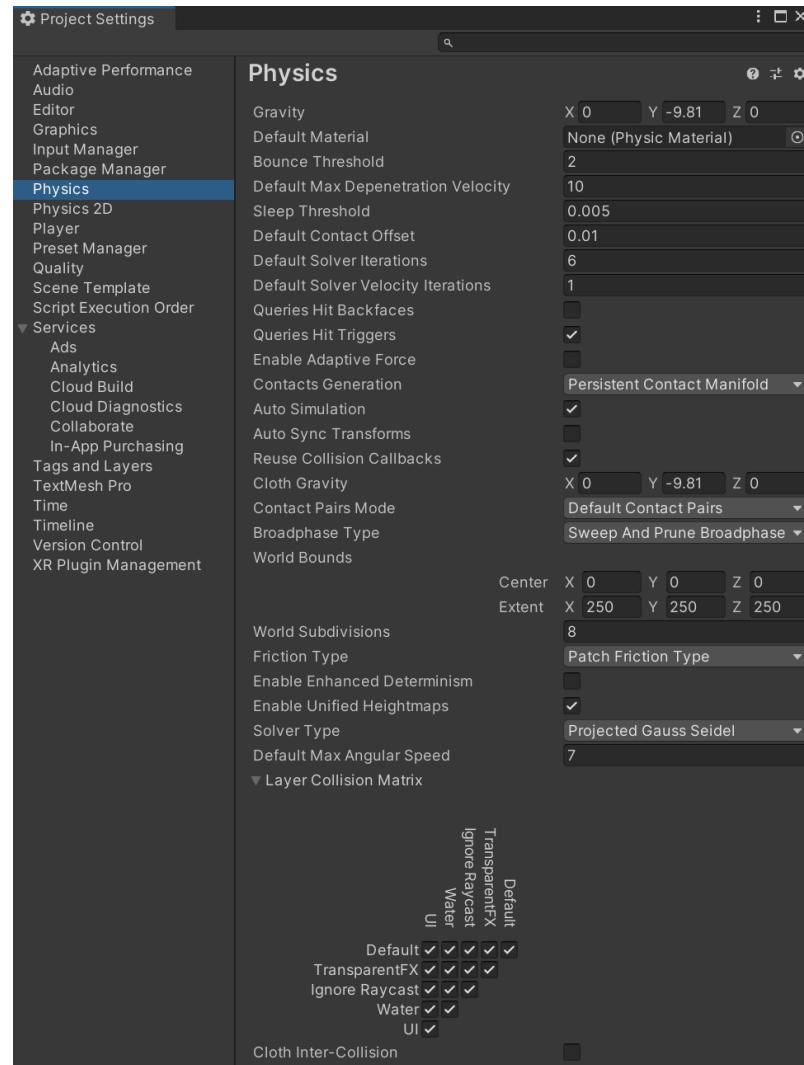
Physic(물리) 작용

- **Interpolate** : Rigidbody의 움직임이 어색할 경우 사용
 - **None** : 보간이 적용되지 않음
 - **Interpolate** : 이전 프레임의 Transform에 맞게 움직임을 부드럽게 처리
 - **Extrapolate** : 다음 프레임의 Transform을 추정하여 움직임을 부드럽게 처리
- **Collision Detection** : 오브젝트가 빠르게 움직여 충돌 하지 않고 뚫고 지나치는 것을 방지
 - **Discrete** : 불연속 충돌 검사, 일반적인 충돌에 사용
 - **Continuous** : 스위핑 기반의 연속 출동 검사, **Rigidbody**가 있는 Collider에는 불연속 충돌 검사를 하고 **없는 Collider**에는 연속 충돌 검사를 한다
 - **Continuous Dynamic** : 스위핑 기반의 연속 출동 검사, 해당 **Rigidbody**는 연속 충돌 검사를 하고 다른 **Rigidbody**는 불연속 충돌 검사를 한다
 - **Continuous Speculative** : **Rigidbody**와 **Collider**간의 추측성 충돌 검사, Kinematic에서 설정할 수 있는 유일한 모드, 다른 스위핑 기반의 연속 충돌 검사에 비해 자원을 적게 사용한다
- **Constraints** : 움직임을 제약
 - **Freeze Position** : 월드 좌표계의 X, Y, Z 축 이동을 선택적으로 중지
 - **Freeze Rotation** : 로컬 좌표계의 X, Y, Z 축 회전을 선택적으로 중지

Physic(물리) 작용

▶ Physic Manager

- 3D 물리를 위한 전역 설정
- Edit => Project Setting => Physics
- 2D 물리는 Physics 2D에서 설정



Physic(물리) 작용

- Gravity : 중력 가속도, 물체의 낙하 속도를 제어
- Default Material : 기본 Physic Material 설정, None이면 각 Collider에서 처리
- Bounce Threshold : 충돌하는 두 물체의 상대 속도가 설정한 값보다 낮으면 튕기지 않는다, 너무 작은 값으로 설정하지 않는 것을 권장
- Default Max Depenetration Velocity : 오브젝트끼리 침투된 상태에서 실행 시, 오브젝트가 튀어나오는 속도 값, 너무 작은 값으로 설정하지 않는 것을 권장
- Sleep Threshold : Rigidbody의 운동 에너지를 질량으로 나눈 값이 설정된 값보다 작은 경우 모드로 변경하여 업데이트 되지 않게 하여 자원 낭비를 줄인다
- Default Contact Offset : 충돌 체크하기 위한 물체 간의 거리, 0에 가까운 값을 설정하면 Jitter가 발생 할 수 있다
- Default Solver Iterations : Solver(관절의 움직임이나 겹치는 구성 요소 간의 접촉 관리와 같은 상호작용을 하는 작은 물리 엔진) 프로세서의 수
- Default Solver Velocity Iterations : Solver가 실행하는 속도 프로세스의 수

Physic(물리) 작용

- Queries Hit Backfaces : 물리 쿼리(예:Physics.Raycast)가 MeshCollider의 **후면 삼각형(폴리곤)**을 감지하기 위해 사용
- Queries Hit Triggers : 물리 Hit tests(예:Raycasts, SphereCasts, SphereTests)가 Trigger 설정된 Collider를 감지하기 위하여 사용
- Enable Adaptive Force : 오브젝트들이 힘(Force)에 영향을 받을 경우 보다 사실적인 동작을 하게 만든다
- Contacts Generation
 - Legacy Contacts Generation : 분할 축 정리를 이용한 컨택트를 생성, Unity 5.5이하의 버전일 경우 PCM보다 더 효율적일 수 있음
 - Persistent Contacts Manifold(PCM) : 물리 프레임마다 더 적은 컨택트를 생성하고 프레임 간에 더 많은 컨택트 데이터를 공유
- Auto Sync Transforms : Transform의 변경 사항을 물리 시스템과 자동으로 동기화
- Reuse Collision Callbacks : 충돌(Collision) 발생시 Callback을 재사용하여 GC호출을 줄인다
- Cloth Gravity : 천(Cloth)에 작용하는 중력 가속도

Physic(물리) 작용

- Contact Pairs Mode
 - > Default Contact Pairs : 운동학(Kinematic)-운동학(Kinematic) 및 운동학(Kinematic)-정적(Static)을 제외한 모든 접촉(Contact)에 대하여 충돌 및 Trigger 이벤트를 수신
 - > Enable Kinematic Kinematic Pairs : 운동학(Kinematic)-운동학(Kinematic) 접촉(Contact)에 대하여 충돌 및 Trigger 이벤트를 수신
 - > Enable Kinematic Static Pairs : 운동학(Kinematic)-정적(Static) 접촉(Contact)에 대하여 충돌 및 Trigger 이벤트를 수신
 - > Enable All Contact Pairs : 모든 접촉 (Contact)에 대한 충돌 및 Trigger 이벤트를 수신
- Broadphase Type : 물리 시뮬레이션에 사용할 Broad-phase 알고리즘
 - > Sweep and Prune Broadphase : sweep-and-prune Broadphase 충돌 알고리즘 사용, 단일 축을 따라 오브젝트를 정렬하여 멀리 떨어진 Contact Pairs를 배제, 평평(flat)한 월드에 많은 오브젝트가 있는 경우 성능향상을 볼 수 있다
 - > Multibox Pruning Broadphase : 그리드(Grid)를 이용하며 각 그리드 셀은 sweep-and-prune을 수행한다
 - > Automatic Box Pruning : 세계 경계(Bounds)와 세분화(Subdivisions) 수를 자동으로 계산하는 제외하고 Multibox Pruning 알고리즘과 방식은 유사하다
- World Bounds : Multibox Pruning Broadphase를 사용할 때 멀리 있는 개체가 서로 영향이 미치지 않도록 세계(World)를 둘러싸는 2D 그리드(Grid)를 정의
- World Subdivisions : 2D 그리드(Grid)에서 X, Z축을 이용해 만든 셀의 수, Multibox Pruning Broadphase 설정에서만 사용

Physic(물리) 작용

- **Friction Type** : 마찰 알고리즘
 - Patch Friction Type : 낮은 Solver Iterations에서 가장 안정적인 결과를 가지는 **기본 마찰 알고리즘**
 - One Directional Friction Type : 쿠롬(Coulomb) 마찰 모델을 단순화 시킨 알고리즘, 양방향(two-directional) 모델에 비해서 정확도가 떨어진다
 - Two Directional Friction Type : 단방향(one-directional) 모델과 유사하지만 두 방향에서 동시에 마찰을 적용, 모든 접촉(Contact)점에 적용되기 때문에, 비례하여 많은 비용이 듈다
- **Enable Unified Heightmaps** : **Terrain** 충돌을 **Mesh** 충돌과 같은 방식으로 처리
- **Solver Type** : 사용할 PhysX 시뮬레이션 선택
 - Projected Gauss Seidel : **기본 PhysX Solver**
 - Temporal Gauss Seidel : 기본 Solver 시뮬레이션 중 일부 불규칙한 동작이 발생할 경우 사용
- **Layer Collision Matrix** : Layer 간의 **충돌 처리 여부** 설정
- **Cloth Inter-Collision**
 - Distance : 천(Cloth)에 구체 **입자를 설정**, 시뮬레이션 중 **구가 겹치지 않게** 한다
 - Stiffness : 천(Cloth)이 겹치며 **충돌**할 때 발생하는 **반발력**, 분리된 상태를 유지하기 위해 사용

따라하기

▶ Main Camera

- position(0, 3, -10)
- Rotation(0, 0, 0)

▶ Create => Physic Material 생성

- Bounciness : 1

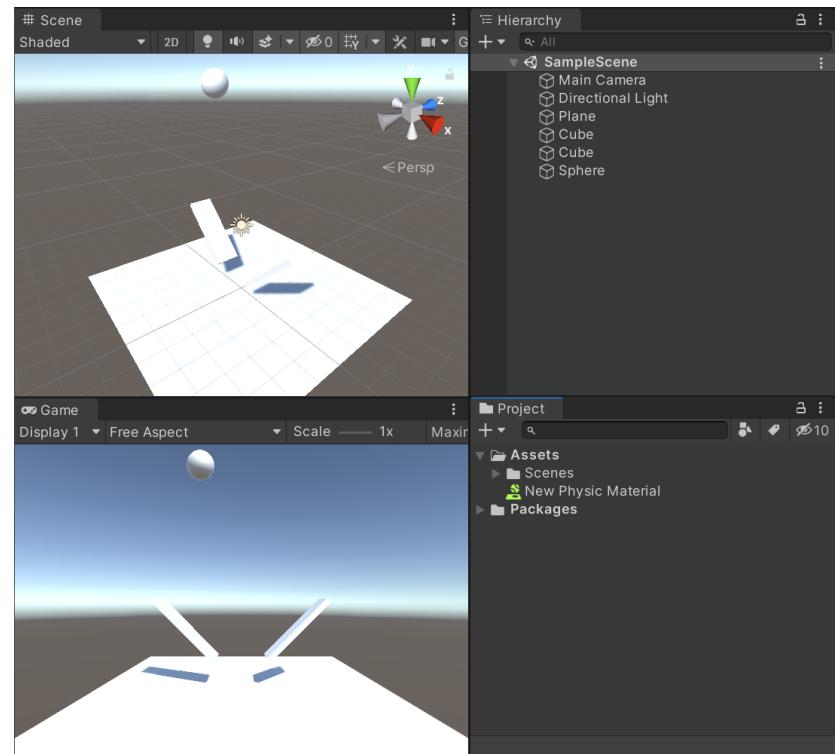
▶ 3D Object => Plane 생성

▶ 3D Object => Cube 2개 생성

- position(2, 2, 0), position(-2, 2, 0)
- Rotation(0, 0, 45), Rotation(0, 0, -45)
- Scale(3, 0.2, 1)
- Collider Material에 New Physic Material 적용

▶ 3D Object => Sphere 생성

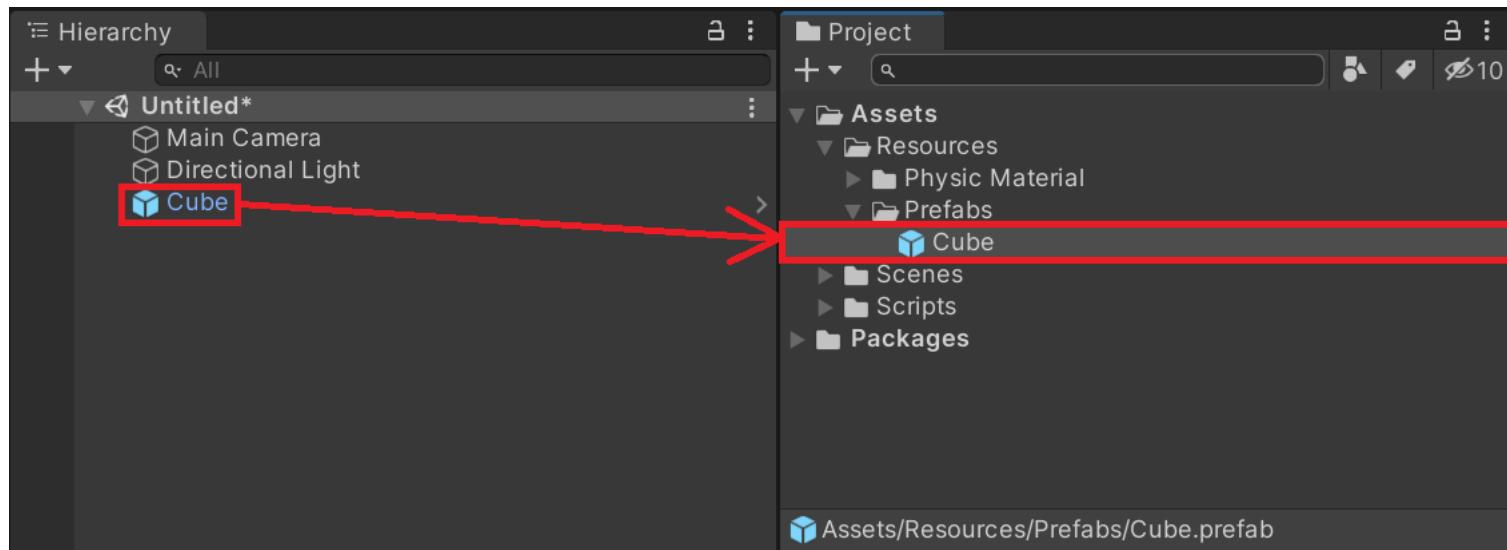
- position(-1.5, 8, 0)
- Collider Material에 New Physic Material 적용



Prefab(프리팹)

▶ Prefab(프리랩)이란?

- 재사용 가능한 GameObject
- 같은 GameObject를 복제하여 사용해야 할 경우 Prefab을 사용
- Base가 되는 Prefab의 속성을 변경하면 모든 copy들의 속성은 자동으로 변경
- Hierarchy의 게임 오브젝트를 Project의 Assets 폴더로 Drag&Drop하여 만들 수 있다



Script(스크립트)

▶ 브로드캐스트(Broadcast)

- 등록된 모두에게 메시지를 보낸 이벤트 처리를 한다
- 메시지를 받은 오브젝트가 메시지에 명시된 기능을 가지고 있다면 해당 기능을 실행, 기능을 가지고 있지 않다면 메시지를 무시
- 컴포넌트의 특정 기능을 간접적으로 실행
- 누가 메시지를 받을지 신경 쓰지 않는다
- 누가 메시지를 보냈는지 신경 쓰지 않는다
- 메시지의 독립성을 유지

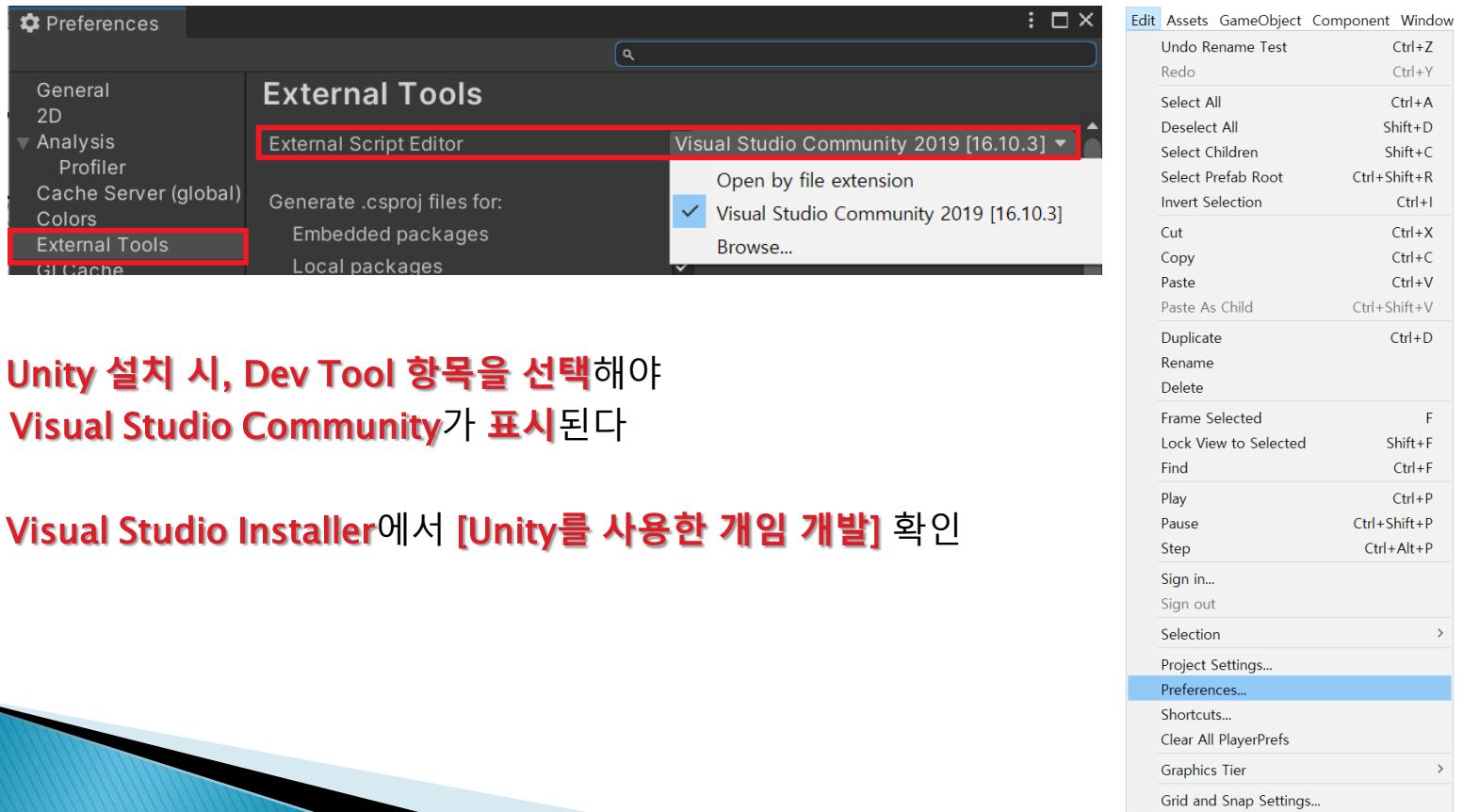
▶ MonoBehaviour

- 유니티의 모든 컴포넌트(Components)는 MonoBehaviour를 상속 받는다
- 원하는 스크립트를 컴포넌트(Component)로 사용하기 위해서는 반드시 MonoBehaviour를 상속 받아야 한다
- 유니티는 씬(Scene)에 등록된 모든 컴포넌트에게 메시지를 보내는 브로드캐스팅 방법을 사용한다

Script(스크립트)

▶ External Script Editor

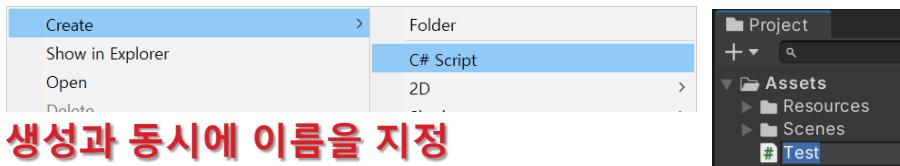
- 유니티 메뉴 Edit => Preference => External Tools => External Script Editor
- 원하는 Editor를 선택하여 사용



Script(스크립트)

▶ Script 생성

- Project 창 => 마우스 오른쪽 버튼 => Create => C# Script

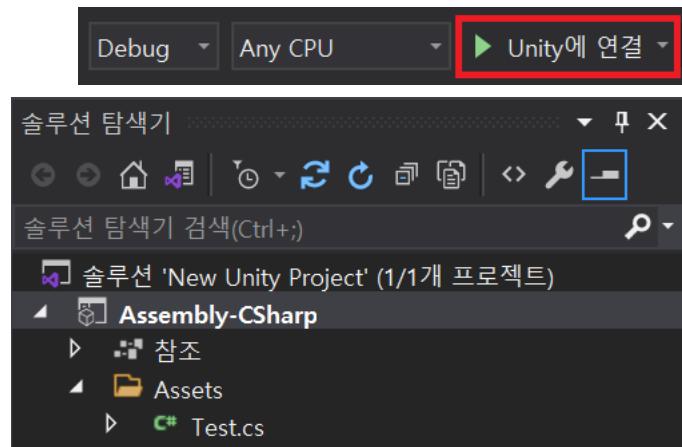


생성과 동시에 이름을 지정

- 생성 후에 파일 이름을 변경할 경우 반드시 **스크립트의 class 명을 파일명과 통일**하여야 한다
- 생성된 파일을 실행하여 Script를 수정할 수 있다
- **Visual Studio의 디버그 매뉴에서 Unity와의 연결을 확인** 가능하며, **솔루션 탐색기에서 해당 프로젝트에서 생성된 모든 스크립트 확인**이 가능하다

▶ Visual Studio 디버깅

- **Visual Studio에서 디버깅 시작(F5) 후, Unity프로젝트**로 돌아가 **Play**하면 Visual Studio를 이용하여 브레이크 포인트를 걸어 디버깅이 가능하다



Script(스크립트)

- ❖ `Start()`, `Update()` 함수, 특히 `Update()` 같이 매번 호출되는 함수는 **내용이 없어도 호출에 많은 비용**이 들기 때문에 **사용하지 않으면 반드시 삭제**하도록 하자

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

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

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

Script(스크립트)

▶ Awake()

- 씬(Scene)이 시작할 때 **한번**, Start() 호출 전에 호출된다
- 게임 오브젝트가 비활성화 상태면 활성화 될 때까지 호출되지 않는다

▶ OnEnable()

- Start() 호출 전, **매번** 오브젝트 활성화 직후 호출

▶ OnDisable()

- **매번** 오브젝트 비활성화 직후 호출

▶ Start()

- **첫 번째 프레임 업데이트 전에 한번** 호출
- 게임 오브젝트가 비활성화 상태면 활성화 될 때까지 호출되지 않는다

▶ FixedUpdate()

- 프레임 속도와 관계없이 **일정한 프레임**에 호출
- Edit => Project Setting => Time => Fixed Timestep에서 호출 간격 수정 가능
- 모든 물리 계산은 FixedUpdate() 후 즉시 완료, 움직임 계산을 적용할 때 Time.deltaTime을 곱하지 않아도 된다

▶ Update()

- **프레임 당 한번** 호출

▶ LateUpdate()

- **Update()가 끝난 후 프레임 당 한번** 호출, Update()에서 수행된 모든 계산은 LateUpdate()가 시작할 때 완료된다

Script(스크립트)

▶ OnTriggerXXX()

- 반드시 접촉자 혹은 대상은 **Rigidbody**를 지니고 있어야 한다
- Collider의 [**Is Trigger**] **True** 충돌 판정
- `OnTriggerEnter()` : 오브젝트 사이의 **접촉일 일어난 순간** 호출
- `OnTriggerStay()` : 오브젝트가 **접촉 상태, 매 프레임** 호출
- `OnTriggerExit()` : 오브젝트 사이의 **접촉이 해제되는 순간** 호출

▶ OnCollisionXXX()

- 반드시 접촉자 혹은 대상은 **Rigidbody**를 지니고 있어야 한다
- Collider의 [**Is Trigger**] **False** 충돌 판정
- `OnCollisionEnter()` : 오브젝트 사이의 **접촉일 일어난 순간** 호출
- `OnCollisionStay()` : 오브젝트가 **접촉 상태, 매 프레임** 호출
- `OnCollisionExit()` : 오브젝트 사이의 **접촉이 해제되는 순간** 호출

Script(스크립트)

▶ OnMouseXXX()

- 오브젝트에 반드시 **Collider**가 있어야 한다
- Layaer가 **Ignore RayCast**이면 안된다
- 해당 오브젝트에 관한 마우스 이벤트 발생 시 호출
- **OnMouseDown()** : 오브젝트에서 **마우스 버튼이 눌려진 순간** 호출
- **OnMouseDrag()** : 오브젝트에서 **마우스 버튼이 눌려진 상태 또는 눌려진 상태에서 이동 시(오브젝트를 벗어나도 상관 없다), 매 프레임** 호출
- **OnMouseEnter()** : **마우스 포인터가 오브젝트 위에 진입할 경우** 호출
- **OnMouseExit()** : **마우스 포인터가 오브젝트 위에서 벗어날 경우** 호출
- **OnMouseOver()** : **마우스 포인터가 오브젝트 위에 있을 경우, 매 프레임** 호출
- **OnMouseUp()** : 오브젝트에서 **마우스 버튼이 눌려졌다 때는(오브젝트에서 벗어나도 상관 없다) 순간** 호출
- **OnMouseUpAsButton()** : 오브젝트에서 **마우스 버튼이 눌려졌다 오브젝트 위에서 때는 순간** 호출

Script(스크립트)

▶ OnApplicationPause()

- 일시 정지가 감지된 프레임의 끝, 실질적으로 일반 프레임의 업데이트 사이에 호출
- 일시 정지 상태를 가리키는 그래픽스를 표시하기 위해 호출된 후에 한 프레임이 추가로 업데이트 실행
- 해당 애플리케이션의 활성화(선택된) 상태를 확인할 수 있다

▶ OnApplicationQuit()

- 애플리케이션 종료 전, 모든 오브젝트에서 호출

▶ OnDestroy()

- 오브젝트를 삭제 할 경우, 오브젝트가 존재했을 때의 마지막 프레임에 대한 모든 프레임 업데이트를 마친 후 호출

따라하기

▶ 새 씬(Scene) 만들기

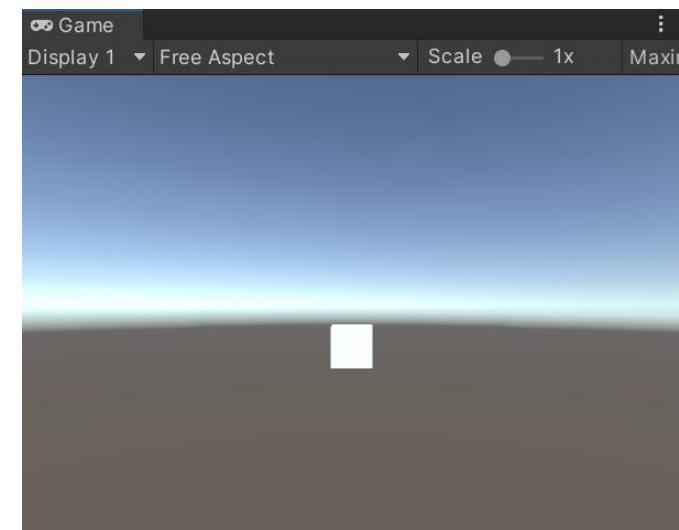
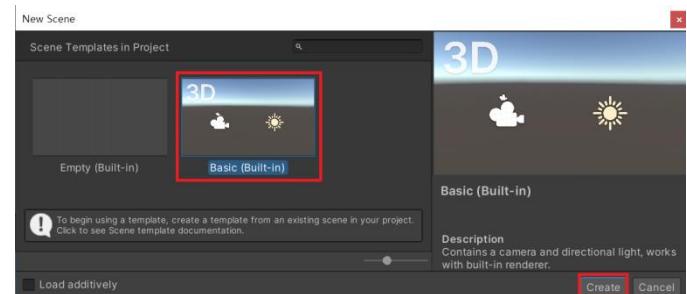
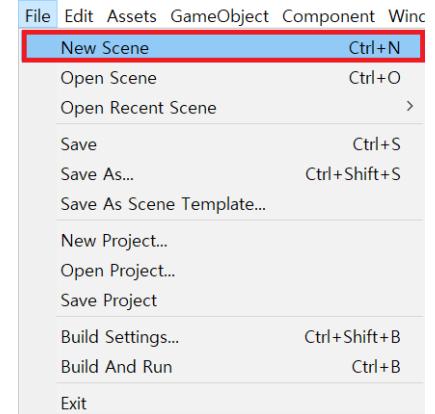
- File => New Scene

▶ 3D Object => Cube 2개 생성

- position(0, 0, 0)
- CollisionTest.cs 컴포넌트 추가

OnMouseXXX() 마우스 이벤트 확인

**CollisionTest.cs 파일은
[예제소스] 폴더 확인**



회전

▶ 오일러 각(Euler Angle)

- 수학자 오일러(Euler)가 고안한 개념
- 3차원 공간의 절대적 좌표를 기준으로 물체의 회전을 측정하는 방법
- X, Y, Z 축을 기준으로 회전하기 때문에 직관적이고 조작이 용이하다
- 180도가 넘는 회전의 표현이 가능하다
- 3번에 나누어 계산하기 때문에 비용이 크며 짐벌 락(Gimbal Lock)이 발생할 수 있다

▶ 쿼터니언(Quaternion)

- 방향(orientation)과 회전(rotation)을 표현한다
- 4개의 성분(x, y, z, w)로 이루어져 있으며 각 성분은 축이나 각도가 아니다
- 벡터(x, y, z)와 스칼라($w : \text{roll}$)를 의미한다
- 180도가 넘는 회전을 표현하지 못한다
- 각 축을 한번에 계산하여 비용이 적고 짐벌 락(Gimbal Lock)이 발생하지 않는다

▶ 짐벌 락(Gimbal Lock)

- 두 개의 회전 축이 서로 겹쳐지는 현상

// 오일러 각 값으로 쿼터니언 회전.

Quaternion.Euler(angleX, angleY, angleZ);

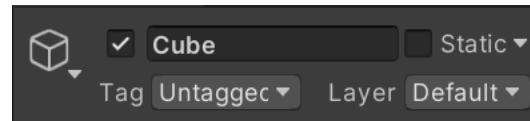
// 특정 축(axis)을 기준으로 설정한 각(angle)만큼 회전.

Quaternion.AngleAxis(angle, axis);

GameObject

➤ Properties

- New GameObject : 새 게임 오브젝트를 생성
- gameObject : 해당 스크립트를 컴포넌트로 가지고 있는 게임 오브젝트
- gameObject.name : 게임 오브젝트의 이름
- gameObject.layer : 게임오브젝트의 레이어
- gameObject.tag : 게임 오브젝트의 태그
- gameObject.activeSelf : 게임 오브젝트의 **로컬 활성화 상태**
- gameObject.activeInHierarchy : Hierarchy에서 게임 오브젝트의 활성화 상태, **부모 게임 오브젝트가 비활성화이면 자식 게임 오브젝트도 비활성화**
- gameObject.transform : 게임 오브젝트의 Transform



GameObject

➤ Methods

- GC 이슈와 많은 비용으로 GetComponent() 함수는 **Update()**에서 호출 자제하는 것이 좋다
- gameobject.AddComponent<T>() : 게임 오브젝트에 **T 형식의 컴포넌트를 추가**
- gameobject.GetComponent<T>() : 게임 오브젝트에서 **T 형식의 컴포넌트를 가져온다**, 가져오지 못할 경우 Null을 반환
- gameobject.GetComponentInChildren<T>() : 게임 오브젝트 **자신 또는 자식에** 있는 **T 형식의 컴포넌트를 가져온다**, 가져오지 못할 경우 Null을 반환
- gameobject.GetComponentInParent<T>() : 게임 오브젝트 **자신 또는 부모에** 있는 **T 형식의 컴포넌트를 가져온다**, 가져오지 못할 경우 Null을 반환
- gameobject.TryGetComponent<T>(out value) : 게임 오브젝트에서 **T 형식의 컴포넌트를 value에 가져온다**, 가져왔다면 **True** 아니면 **False**를 반환
- gameobject.SetActive(bool) : 게임 오브젝트의 **활성화 상태를 변경**(True:활성화, False:비활성화)
- Gameobject.CreatePrimitive(PrimitiveType.*) : **기본**으로 지원하는 **3D오브젝트를 생성**

특정 오브젝트 찾기

▶ Find

- Hierarchy에 있는 특정 게임 오브젝트 또는 컴포넌트를 찾아 올 수 있다
- Hierarchy를 순차적으로 내려가며 검색하여 비용이 많이 들기 때문에 **Update()**에서 호출 하지 않는 것이 좋다

- `GameObject.Find("name")` : 이름을 이용하여 활성화된 게임 오브젝트를 찾는다
- `GameObject.FindWithTag("tag_name")` : 태그를 이용하여 활성화된 게임 오브젝트를 찾는다
- `GameObject.FindGameObjectsWithTag("tag_name")` : 태그를 이용하여 활성화된 모든 게임 오브젝트를 찾는다

- `transform.Find("child_name")` : 자신의 자식 계층만을 이름을 이용하여 찾는다, 비활성화된 자식도 찾을 수 있다
- `transform.GetChild(child_index)` : 자신의 자식 계층만을 번호로 찾는다, 자식 계층의 첫 번째가 0번째 인덱스, 비활성화된 자식도 찾을 수 있다

- `FindObjectOfType<컴포넌트>()` : 해당 컴포넌트를 가지고 있는 처음 발견된 활성화 게임 오브젝트의 컴포넌트 반환
- `FindObjectsOfType<컴포넌트>()` : 해당 컴포넌트를 가진 모든 활성화된 게임 오브젝트의 컴포넌트 배열로 반환

오브젝트 생성 / 삭제

▶ Instantiate(), Destroy()

- 프리팹(Prefab) 또는 Hierarchy에 있는 GameObject를 복제(생성) 및 삭제할 수 있다
- 게임 오브젝트의 생성 및 삭제에는 많은 비용이 들기 때문에 Update()에서 호출 하지 않는 것이 좋다
- `Object Instantiate(Object original, Vector3 position, Quaternion rotation, Transform parent)`
 - > original : 만들려는(복제하려는) 베이스 오브젝트
 - > position : 만들어진 오브젝트의 생성 위치
 - > rotation : 만들어진 오브젝트의 회전
 - > parent : 만들어진 오브젝트의 부모
- `public GameObject prefab;`
`void Start()`
`{`
 `var go1 = Instantiate(prefab);`
 `var go2 = Instantiate<Transform>(prefab.transform);`
`}`
- `Destroy(GameObject obj, float t)`
 - > obj : 삭제 하려는 게임 오브젝트
 - > t : 삭제 딜레이 시간
- `public GameObject go1;`
`public GameObject go2;`
`private void OnDisable()`
`{`
 `Destroy(go1);`
 `Destroy(go2, 3f);`
`}`

리소스 가져오기

▶ Resources.Load()

- Resources 폴더에 있는 여러 리소스 파일을 쉽게 가져올 수 있다
- Resources 폴더는 중복으로 만드는 것이 가능하다

- T Load<T>(string path)

- T형식의 지정한 이름과 같은 리소스 파일을 가져온다
- path : 가져 오려는 리소스의 경로와 이름(확장자 제외)

```
public GameObject res;  
void Start()  
{  
    res = Resources.Load<GameObject>("Prefabs/Cube");  
}
```

- T[] LoadAll<T>(string path)

- 해당 폴더의 T형식의 모든 리소스 파일을 가져온다
- path : 가져 오려는 리소스들이 있는 폴더의 경로

```
public GameObject[] arr_res;  
void Start()  
{  
    arr_res = Resources.LoadAll<GameObject>("Prefabs");  
}
```



데이터 관리

▶ PlayerPrefs

- 유니티에서 지원하는 데이터 관리 클래스
- OnApplicationQuit()이 호출될 때 데이터가 저장된다
- Windows 환경에서 레지스트리(Registry)에 데이터가 저장된다
- 경로 : HKEY_CURRENT_USER/Software/Unity/UnityEditor/[CompanyName]/[projectName]
- int, float, string 타입의 값을 읽고 쓰는 것이 가능하다

- `XXX GetXXX(string key, XXX defaultValue)`
 - key : 데이터의 키
 - defaultValue : 해당 키로 저장된 값이 없을 경우의 디폴트 값
 - `PlayerPrefs.GetInt("Int Value", 0);`

- `void SetXXX(string key, XXX value)`
 - key : 데이터의 키
 - value : 저장할 데이터의 값
 - `PlayerPrefs.SetInt("Int Value", 1);`

- `bool HasKey(string key)`
 - 해당 키를 가진 데이터의 유무 확인

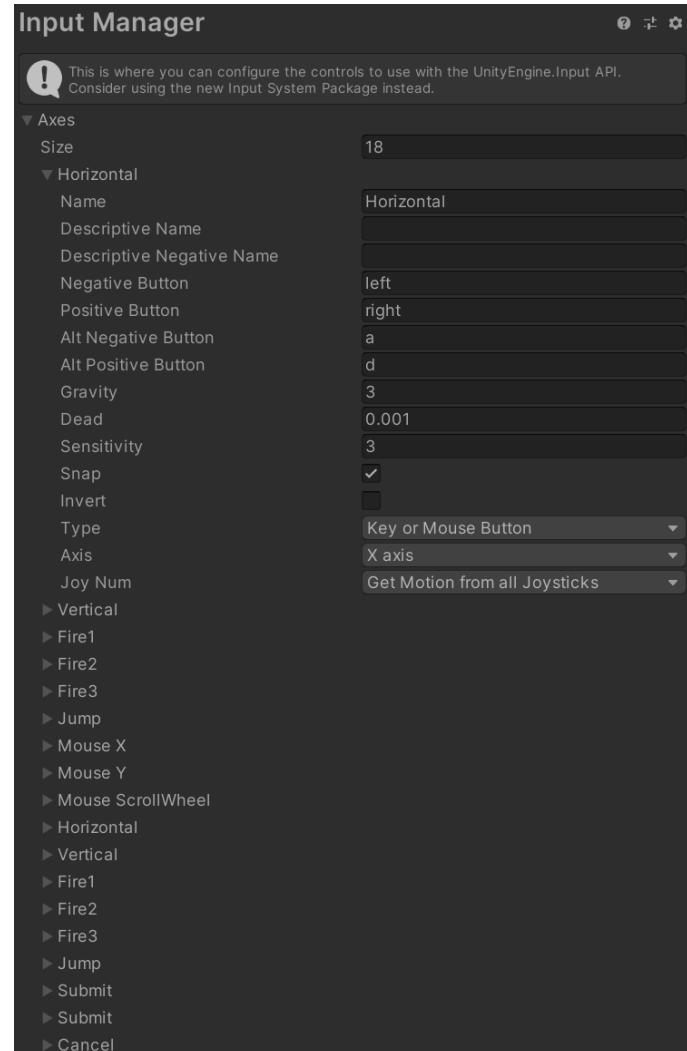
- `void Save()`
 - 원하는 때에 데이터를 저장한다

- `void DeleteKey(string key), void DeleteAll()`
 - 해당 키를 가진 또는 모든 데이터를 제거(삭제) 한다

Input

▶ Input Manager

- 메뉴 Edit=>Project Settings=>Input Manager
 - 입력 축(Axis), 버튼 등을 정의하여 사용 가능
 - 인풋의 결과값을 -1 ~ 1 범위의 값을 반환
-
- **Name** : 설정한 버튼의 이름(스크립트에서 참조)
 - **Negative Button, Positive Button**
 - : 음수(-1 ~ 0), 양수 값(0 ~ 1)을 알리는 기본(메인) 버튼
키보드, 조이스틱, 마우스 사용
 - **Alt Negative Button, Alt Positive Button**
 - : 대체(서브) 컨트롤 버튼
 - **Gravity** : 입력이 없을 때 중립(0)으로 가는 속도(초 단위)
 - **Dead** : 아날로그 스틱의 이동 값이 설정 값보다 작으면 0으로 처리
 - **Sensitivity**
 - : 축이 대상 값으로 향하기 위한 속도(초 단위)
디지털 장치(키보드) 전용
 - **Snap** : 활성화 되어 있다면, 반대 방향 입력 때 축의 값은 0으로 초기화
 - **Invert** : 음수, 양수 값이 반대로 적용
 - **Type** : 축 제어를 위한 입력 타입
 - **Key or Mouse Button** : 키(키보드) 또는 마우스 버튼
 - **Mouse Movement** : 마우스의 이동
 - **Joystick Axis** : 조이스틱의 아날로그 스틱
 - **Axis** : 이 축을 제어하는 연결된 기기의 축
 - **Joy Num** : 축을 제어할 연결된 조이스틱을 선택



Input

▶ 키보드

키 패밀리	명명 규칙
문자 키	a, b, c...
숫자 키	1, 2, 3...
화살표 키	up, down, left, right
숫자패드 키	[1], [2], [3], [+], [equals] ...
수정 키	right shift, left shift, right ctrl, left ctrl, right alt, left alt, right cmd, left cmd
특수 키	backspace, tab, return, escape, space, delete, enter, insert, home, end, page up, page down
기능 키	f1, f2, f3...

▶ 조이스틱

버튼 원점	명명 규칙
조이스틱의 특정 버튼입니다.	joystick button 0, joystick button 1, joystick button 2...
특정 조이스틱의 특정 버튼입니다.	joystick 1 button 0, joystick 1 button 1, joystick 2 button 0...

▶ 마우스

- mouse 0(왼쪽), mouse 1(오른쪽), mouse 2(휠), ... (마우스의 추가 버튼)

Input

- ▶ `float GetAxis(string axisName)`
 - **-1 ~ 1 범위의 값을 반환**, 부드러운 이동을 위해 사용
 - `axisName` : **Input Manager**에 설정해둔 버튼 이름을 사용
- ▶ `float GetAxisRaw(string axisName)`
 - **-1, 0, 1 값을 반환**, 즉각적인 반응을 위해 사용
 - `axisName` : **Input Manager**에 설정해둔 버튼 이름을 사용
- ▶ `bool GetButtonXXX(string buttonName)`
 - 해당 **버튼이 눌려졌는지** 알 수 있다
 - `GetButtonDown`, `GetButton`, `GetButtonUp`가 있으며 각각 눌려 졌을때, 누르고 있을 때, 누른 손을 뗐을 때
 - `buttonName` : **Input Manager**에 설정해둔 버튼 이름을 사용
- ▶ `bool GetKeyXXX(string name/KeyCode key)`
 - 해당 **버튼이 눌려졌는지** 알 수 있다
 - `GetKeyDown`, `GetKey`, `GetKeyUp`가 있으며 각각 눌려 졌을때, 누르고 있을 때, 누른 손을 뗐을 때
 - `name` : **키 이름**
 - `key` : **가상 키 코드**
- ▶ `bool GetMouseButtonXXX(int button)`
 - **마우스의 버튼이 눌려졌는지** 알 수 있다
 - `GetMouseButtonDown`, `GetMouseButton`, `GetMouseButtonUp`가 있으며 각각 눌려 졌을때, 누르고 있을 때, 누른 손을 뗐을 때
 - `button` : 마우스 버튼의 인덱스(**0 ~ 6**)
- ▶ `Touch GetTouch(int index)`
 - 모바일 기기에서 **터치 정보**를 알 수 있다
 - `Input.touchCount`를 이용하여 **총 터치하고 있는 수**를 알 수 있다
 - `index` : 알아 볼 터치의 인덱스(**0~n**)

UI

▶ OnGUI

- 현재는 **사용되지 않는 유니티 UI** 함수
- 렌더링 및 연산 **비용이 크다**

▶ NGUI

- **외부** 플러그인 **에셋(Asset)**
- OnGUI의 문제 및 NGUI의 편의성 등으로 **많이 사용되었다**
- 현재는 UGUI를 사용하기 때문에 **사용되지 않는다**

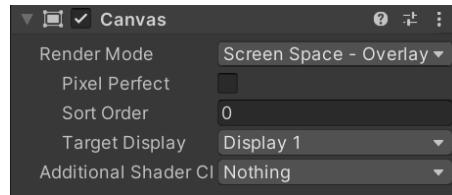
▶ UGUI

- **Unity Graphic User Interface**로 **NGUI 개발자들이 참여**하여 만든 **유니티 자체 UI 툴**이다
- 개발자가 같아서 **NGUI와 사용법이 비슷**하다
- 유니티 자체 UI 툴이기 때문에 훨씬 호환성 및 편의성에서 뛰어나다

UI

▶ Canvas

- UI를 그려주기 위한 영역
- 반드시 모든 UI는 Canvas의 하위에 위치하여야만 한다
- Canvas가 없을 경우 UI를 만들면 자동으로 Canvas가 만들어지며 그 하위에 UI가 위치하게 된다
- 한번에 여러 개의 Canvas가 존재 할 수 있다(최대한 적게 사용하는 것이 좋다)

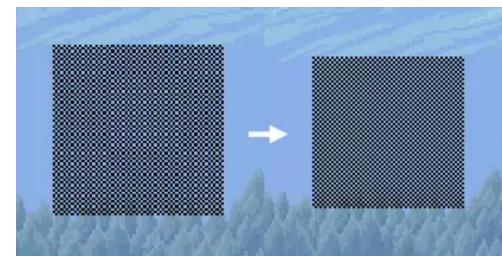


◆ Canvas(Script) RenderMode

- Screen Space – Overlay : UI의 모든 요소가 화면 씬의 위에 렌더링 된다
해상도 및 크기가 변경되면 캔버스가 알아서 크기를 변경한다
- Screen Space – Camera : Screen Space – Overlay와 유사하며 특정 Camera의 주어진 거리(Plane Distance)의 앞에 위치하여 설정한 Camera의 설정 값에 영향을 받아 Camera에서 렌더링 된다
- World Space : UI 오브젝트를 월드 상의 다른 오브젝트처럼 동작하게 한다

- Pixel Perfect : 이미지의 픽셀을 정리하여 깔끔하게 나오도록 한다
Screen Space(Overlay/Camera)에서만 사용 가능

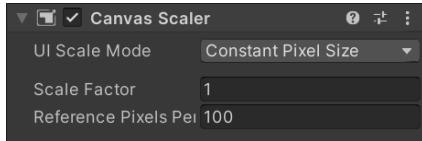
- Additional Shader Channels : Canvas Mesh를 사용할 경우
Shader 채널을 추가할 수 있다



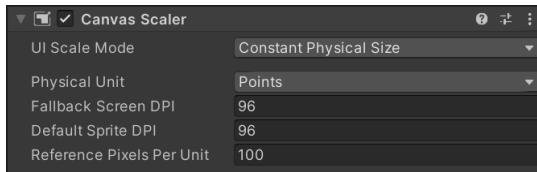
UI

◆ Canvas Scaler(Script)

- UI Scale Mode : UI 요소가 스케일 되는 방법

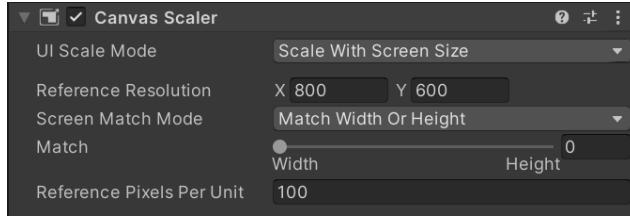


- **Constant Pixel Size** : 화면의 크기와 해상도에 관계없이 UI 요소가 동일한 물리적 크기로 유지된다
- ❖ **Scale Factor** : UI 요소를 이 배율로 스케일 합니다
- ❖ **Reference Pixels Per Unit** : Sprite에 'Pixels Per Unit' 설정이 적용된 경우 Sprite의 1픽셀이 UI의 유닛 하나에 해당 한다



- **Constant Physical Size** : 화면의 크기와 해상도에 관계없이 UI 요소가 동일한 물리적 크기로 유지된다
- ❖ **Physical Unit** : UI 요소의 물리적 이동 및 크기의 단위
- ❖ **Fallback Screen** : 화면 DPI를 알 수 없는 경우 사용
- ❖ **Default Sprite DPI** : 'Pixels Per Unit' 설정이 'Reference Pixel Per Unit' 값과 일치하는 Sprite에 사용할 인치당 픽셀(DPI)
- ❖ **Reference Pixels Per Unit** : Sprite에 'Pixels Per Unit' 설정이 적용된 경우 사용

UI

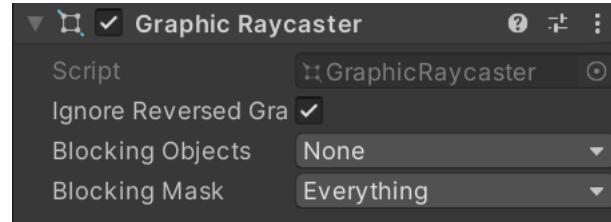


- **Scale With Screen Size** : 화면이 커질수록 UI 요소도 커진다
- ❖ **Reference Resolution** : UI 레이아웃에 적합한 해상도
화면 해상도가 크면 UI가 더 크게, 작으면 UI가 더 작게 스케일 된다
- ❖ **Screen Match Mode**
 - ◆ Match Width or Height : 캔버스의 영역의 너비 또는 높이를 Reference로 사용하여 스케일 하거나 그 사이 값으로 스케일 한다
Match 값으로 너비와 높이의 사용 비율을 설정한다
 - ◆ Expand : 캔버스의 크기가 Reference보다 더 작아지지 않도록 수평 또는 수직으로 확장한다
 - ◆ Shrink : 캔버스의 크기가 Reference보다 더 커지지 않도록 수평 또는 수직으로 자른다
- ❖ **Reference Pixels Per Unit** : Sprite에 'Pixels Per Unit' 설정이 적용된 경우 Sprite의 1픽셀이 UI의 유닛 하나에 해당 한다

UI

◆ Graphic Raycaster(Script)

- Canvas의 UI 기능을 사용하지 않고 **Raycast를 할 경우 사용**된다
- **Ignore Reversed Graphics** : Raycaster가 **후면 그라픽스를 무시**한다
- **Blocking Objects** : **설정한 Collider(2D/3D)**로 **Ray를 막는다**
- **Blocking Mask** : **설정한 Layer**로 **Ray를 막는다**



```
using UnityEngine.EventSystems;
using UnityEngine.UI;

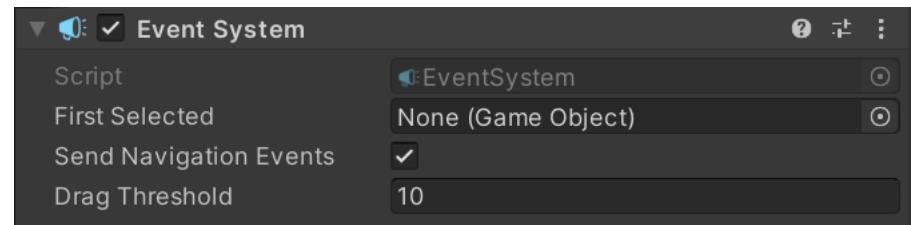
[SerializeField] private GraphicRaycaster graphicRaycaster;
[SerializeField] private EventSystem eventSystem;
private PointerEventData pointerEventData;
private List<RaycastResult> results = new List<RaycastResult>();

void Start()
{
    pointerEventData = new PointerEventData(eventSystem);
    //or pointerEventData = new PointerEventData(null);
}
void Update()
{
    results.Clear();
    pointerEventData.position = Input.mousePosition;
    graphicRaycaster.Raycast(pointerEventData, results);
    if (0 < results.Count) { ... }
}
```

UI

▶ EventSystem

- 모든 **UI 이벤트 처리**를 한다, 없으면 모든 인풋 이벤트가 작동하지 않는다
- **Canvas가 만들어지면 자동으로 생성**
- 매 Update 마다 입력을 확인하여 처리를 한다
- **First Selected** : 처음 선택된 게임 오브젝트(UI 속성)
ex) Input Field를 설정할 경우 시작 시, 선택하지 않아도
바로 문자열 입력이 가능하다
- **Send Navigation Events** : EventSystem이 네비게이션 이벤트
(이동(move), 제출(submit), 취소(cancel))의 허용 여부
- **Drag Threshold** : 픽셀을 드래그 하기 위한 소프트 영역

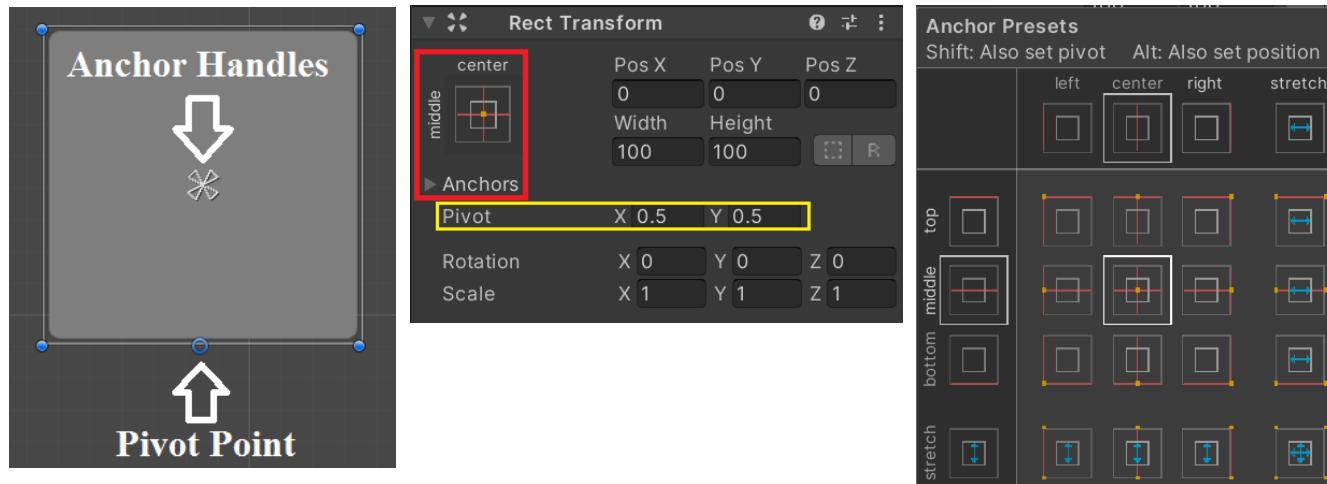


UI

▶ RectTransform

- UI의 Transform

- Pivot으로 해당 오브젝트의 중심을 지정할 수 있다
- Anchors를 이용하여 오브젝트를 특정 위치에 고정할 수 있다
- Anchors는 기본 세팅 외에도 커스텀 값으로 세팅이 가능하다
- Anchors는 화면의 해상도가 변경하면 알아서 크기를 늘이거나 줄여주어 해상도에 의한 UI 배치와 크기 조절을 자동으로 할 수 있다
- 참고 : <https://docs.unity3d.com/kr/530/Manual/UIBasicLayout.html>



실행파일 만들기

▶ Build(빌드)

- 작성한 소스코드를 **실행파일로 만드는 것**
- **멀티 플랫폼 빌드를 지원**하기 때문에 **one source multi build가 가능**하다
- PC, 모바일, 콘솔(PS4/5, Xbox One)을 지원한다
- Nintendo Switch를 빌드하기 위해서는 Nintendo에 **개발자 등록**을 하여 **승인되면** 받을 수 있는 **Nintendo Switch 빌드가 가능한 커스텀 버전의 Unity**를 받을 수 있다고 한다

▶ Build Setting

- **Scene In Build** : 빌드에 포함될 Scene을 등록한다
 - 등록된 Scene을 기준으로 **사용되지 않는 리소스는 빌드에서 자동으로 제외**된다
- **Platform** : 빌드하기 위한 플랫폼을 선택할 수 있다
 - 선택 후 '**Switch Platform**'을 할 경우 해당 **플랫폼 환경에 맞게 개발**을 할 수 있다
- **Player Setting** : 빌드에 필요한 설정을 할 수 있다
 - Product Name(게임 이름), 아이콘, 마우스 커서, 그래픽, 실행 로고 화면(Splash Image) 등 수정이 가능
 - Splash Image의 **유니티 로고는 Pro에서만 제거가 가능**
- **Build/Build And Run** : 빌드를 진행한다

