

# 유니티 사용자를 위한 언리얼 시작 가이드

언리얼 액터와 유니티 게임 오브젝트는 다르다!

# 발표 목차

---

- 언리얼과 유니티 에디터 비교
- 언리얼 엔진의 게임 시작 플로우
- 엔진 아키텍처의 비교
- 프로젝트 관리
- 블루프린트 스크립트에 대한 소고

# 시작하기 전.

---

- 언리얼과 유니티 공식 비교 문서
  - <https://docs.unrealengine.com/latest/KOR/GettingStarted/FromUnity/index.html>
  - 단편적인 비교는 이 문서를 참고.
- 언리얼 엔진 기능에 대한 설명은 공식 문서 참고
  - <https://docs.unrealengine.com/latest/INT/index.html>
  - 상당히 많은 문서가 한글화되어 있음!

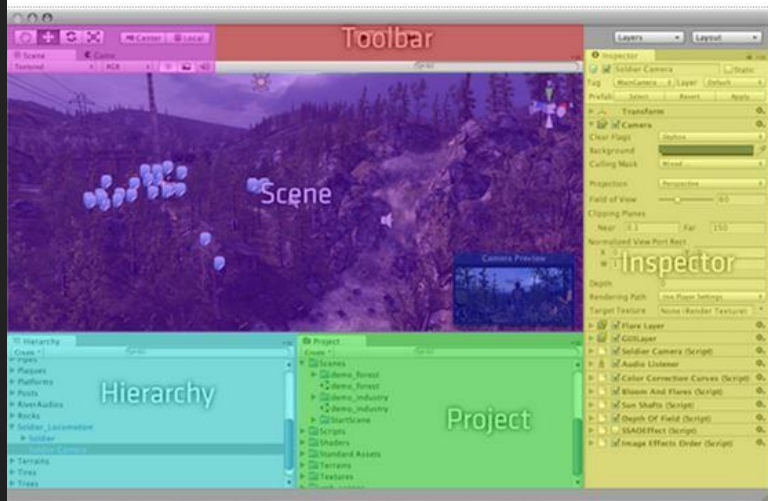
단지 보기에 유사할 뿐 기능이 같은 것은 아님!  
제대로 사용하려면 차이점을 정확히 파악해야함

---

# 에디터 비교

# 에디터 비교

## Unity Editor



## Unreal Editor



# 대응되는 용어

---

- 동일한 기능
  - 뷰포트 윈도우 = 씬 뷰
  - 콘텐츠 브라우저 윈도우 = 프로젝트 뷰
  - 프로젝트 설정 메뉴
- 유사한 기능
  - 디테일 윈도우와 인스펙터 뷰
  - 월드 아웃라이너 윈도우와 계층 뷰
- 언리얼에만 있는 기능
  - 월드 세팅 윈도우 : 작업하는 월드에 대한 설정, 유니티는 월드 관리에 대한 개념이 없음.
  - 모드 윈도우 : [GameObject > Create Other] + 다양한 추가 기능

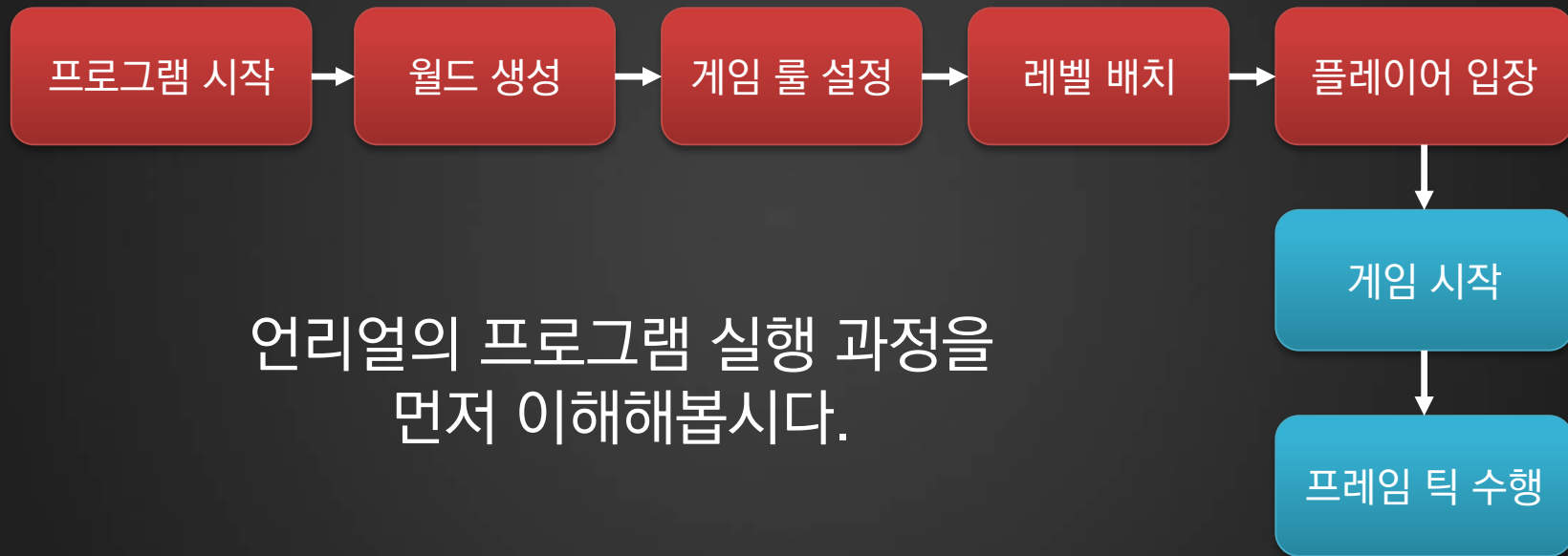
# 에디터 구성이 서로 다른 이유

---

- 언리얼의 액터는 유니티의 게임 오브젝트와 다르다!
- 게임 제작 방식
  - 유니티 : 게임 오브젝트를 사용해 개발자가 게임을 만들어나가는 방식
  - 언리얼 : 게임 제작에 필요한 전체 워크플로우를 갖춰놓고 미리 준비된 요소를 업그레이드하면서 게임을 만들어나가는 방식

언리얼을 사용하려면 언리얼의 제작 방식을 이해해야 함

# 언리얼 프로그램의 진행 과정



언리얼의 프로그램 실행 과정을  
먼저 이해해봅시다.



---

# 언리얼 엔진 게임 시작 플로우

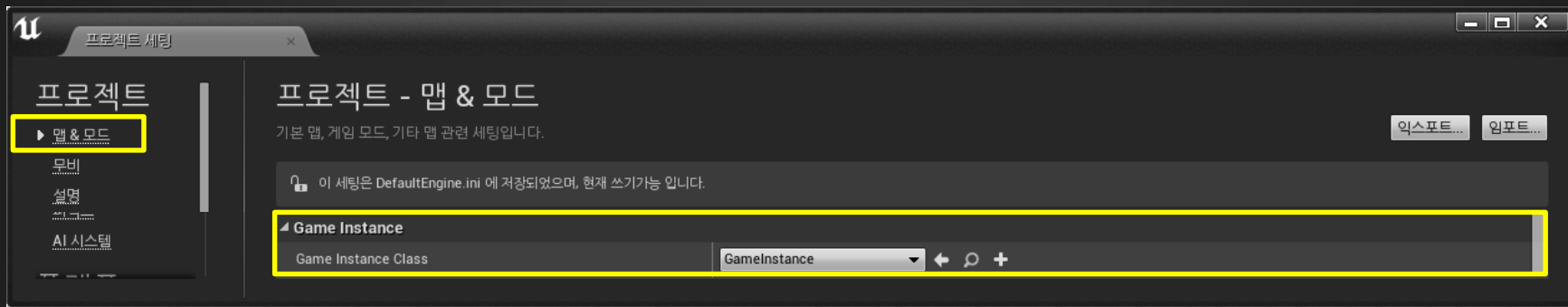
# 첫 번째 단계 : 게임 프로그램의 실행

---

- 게임 인스턴스 : 실행된 게임 프로그램
  - PIE ( Play In Editor ) : 에디터에서 실행된 게임
- 게임 인스턴스의 역할
  - 전체 게임 프로그램의 관리 ( 게임 시작, 게임 종료, 서비스 접속 등등 )
- Custom 게임 인스턴스에서 주로 할 일
  - 게임의 스플래시 이미지 , 종료 UI 등의 표시
  - 월드의 로딩
  - 서비스 접속 시작 등등..

게임의 상황에 맞게 상속받아 처리

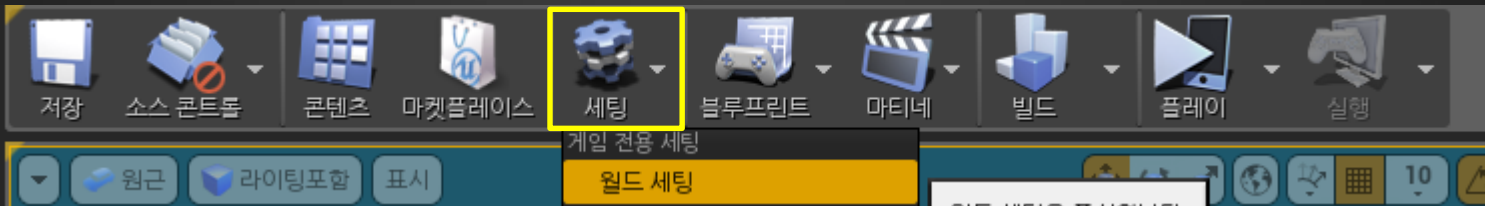
# 게임 인스턴스의 설정



인스턴스 초기화 시 Init 이벤트가 호출  
블루프린트에서 가장 먼저 접근 되는 이벤트

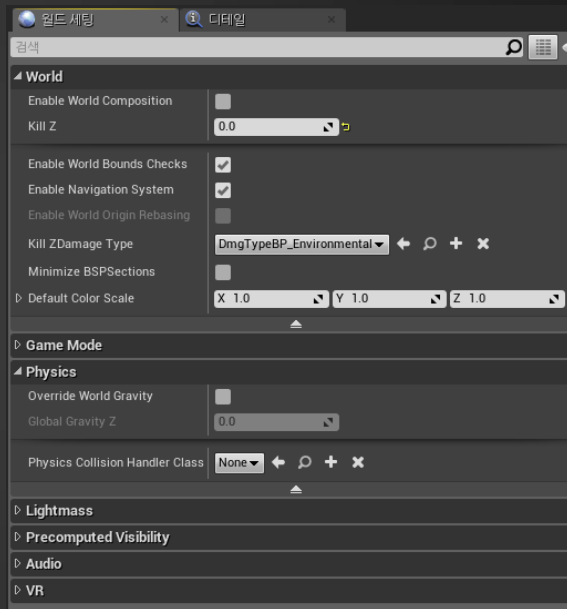
## 두 번째 단계 : 월드의 생성

- 언리얼에서 월드란?
  - 게임을 실행하기 위한 가상 세계의 공간
  - 공간 내에 액터를 배치해 게임에 필요한 기본 환경을 구축하는 역할
  - 따라서 언리얼에서 액터를 생성할 때는 반드시 월드에서 생성해야 한다.
- 월드의 설정
  - [프로젝트 설정 > 맵&모드] Default Map에 저장된 정보로 실행
  - 맵 별로 월드 세팅 윈도우에서 월드의 환경과 사용할 기능들을 지정할 수 있음



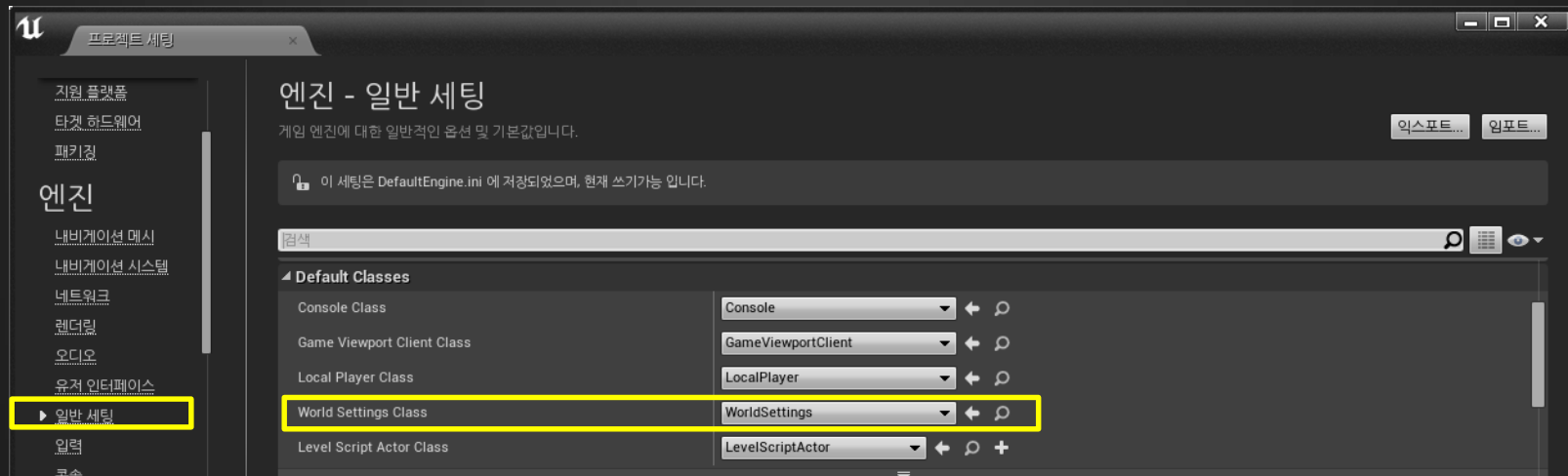
# 월드의 설정

- 월드란 액터가 활동하는 환경
  - 언리얼에서 액터는 반드시 월드에서 생성한다.
  - 그래서 SpawnActor 는 World의 기능
- 월드의 설정
  - 액터 동작에 영향을 미치는 환경과 데이터를 설정
  - 물리 시스템에서 중력을 얼마로 할 것인가?
  - 길찾기 기능을 사용할 것인가?
  - 라이트맵 데이터의 보관.
  - 오클루전 컬링 기능의 적용 여부



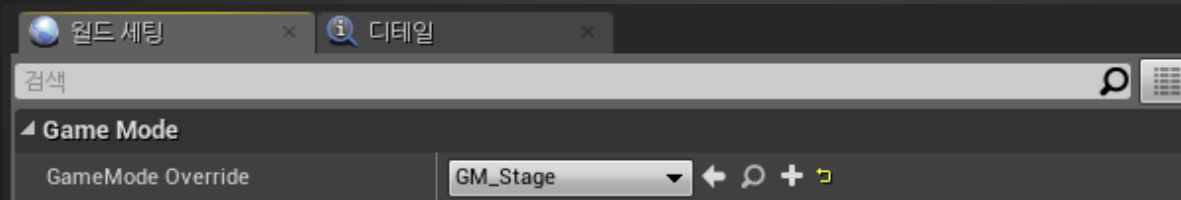
# 월드의 커스터마이징

- 기본으로 제공되는 월드보다 특이한 설정을 하고 싶은 경우
  - C++에서 WorldSettings 클래스를 상속받아서 추가 기능을 구현.



# 세 번째 단계 : 게임의 룰 생성

- 언리얼에서 게임 모드란?
  - 게임을 즐길 공간이 생성되면, 게임 시작을 위해 규칙을 정해야 한다.
  - 게임의 규칙을 정하고 관리를 담당하는 관리자가 필요. 이것이 게임 모드
    - 예) Free For All , Death Match 등의 룰 설정
  - 공정하게 심판을 봐야 하기 때문에, 서버에서 처리하도록 설계되어 있음.
- 게임 모드를 지정하는 방법 : 월드 세팅에서 게임 모드를 지정



# 게임 모드가 하는 일들

- 게임 세션의 관리
  - 플레이어의 게임 입장 / 퇴장, 전체 게임의 시작 타이밍 설정
- 게임 룰의 지정
  - 승리 조건, 스코어 등의 데이터 관리
  - 추가적인 규칙의 설정 예) “우리 편은 총을 맞아도 HP는 달지 않는다”
- 플레이어의 설정





# 네 번째 단계 : 레벨의 설치

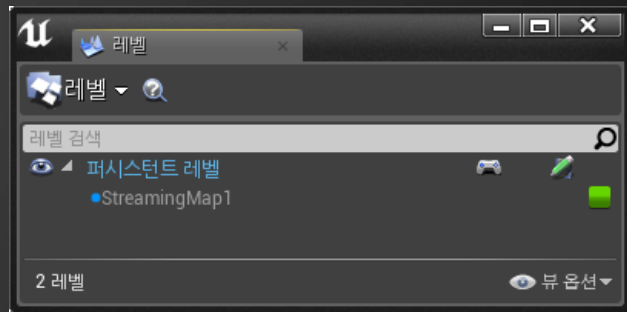
---

- 레벨이란?
  - 게임의 진행을 위해 월드에 설치된 각종 물체들의 집합
- 레벨과 월드의 차이는?
  - 월드는 환경에 대한 설정
  - 레벨은 월드에 배치한 물체들의 집합을 의미
    - 월드에 배치된 장애물, 적, 라이트등 모든 물체.
    - 게임 진행을 위해 월드에서 자기 역할을 수행하는 물체가 액터.

월드는 액터가 머무는 환경, 레벨은 액터의 집합체

# 레벨의 정의와 종류

- 축구 게임의 예시
  - 축구를 시작하기 전에 축구 골대가 완성되고 심판이 준비되어야 한다.
  - 축구 골대, 잔디, 라인, 심판들은 각각 액터며 이들은 레벨에서 관리
  - 게이머가 조종하는 플레이어는 레벨이 아님.
- 레벨의 종류
  - 퍼시스턴트 레벨이란?
    - 에디터에서 개발자가 설정한, 처음에 로딩되는 레벨
  - 스트리밍 레벨이란?
    - 게임 진행시 추가적으로 로딩될 레벨
    - 유니티와 달리 이전에 로딩된 레벨의 부분 해제 가능



# 레벨의 초기화가 끝나면

---

1. 게이머를 대변할 플레이어를 초기화.
2. 플레이어는 준비되었다는 신호를 준다.
  - 플레이어 관련 액터들의 BeginPlay 이벤트 발생
3. 레벨에 속한 액터들도 모두 준비되었다는 신호를 준다.
  - 레벨 내 액터들의 BeginPlay 이벤트 발생
4. 게임 모드는 모두 준비가 완료되었으니 게임을 시작한다는 신호를 보낸다.

이제부터 본격적인 게임이 시작됨

# 게임 시작 프로세스 정리

---

- 플레이 버튼을 누르면 실행하는 순서
  1. 게임 인스턴스의 시작
  2. 게임 인스턴스는 엔진을 통해 지정한 월드를 로딩
  3. 월드의 초기화
    1. 월드에서 게임을 시작하기 위한 게임 모드 생성
    2. 월드에 저장된 레벨 정보를 불러들이고 레벨 내 액터를 초기화
    3. 플레이어 초기화
    4. 모든 액터의 시작
    5. 게임 모드의 완료 신호
  4. 게임의 본격적인 시작

# 발생되는 중요 이벤트

---

- 게임 시작시 발생하는 중요 이벤트
  1. **GameInstance의 Init 이벤트**
  2. 프로젝트 설정에서 지정한 기본 맵 경로를 참고해 월드 생성
  3. 월드의 초기화
    1. 월드 설정을 참고해 게임 모드 생성
    2. 레벨 내 액터의 Initialize Component 이벤트 호출
    3. 각 플레이어 액터의 Initialize Component 이벤트 호출
    4. 모든 액터의 BeginPlay 이벤트 호출
    5. **GameMode의 StartPlay 함수 호출 ( C++에서만 호출 가능 )**
  4. 프레임마다 Tick 이벤트 호출

# 중간 정리

## 게임 인스턴스

### 월드

#### 1. 게임 모드 생성

플레이어  
설정 정보  
(클래스타입)

#### 2. 레벨 액터 스폰

액터

액터

액터

액터

#### 3. 플레이어 스폰

컨트롤러

폰

HUD

스테이트

# 용어 정리

---

- 엔진 : 게임에 관련된 기능을 생성해주는 공장
- 게임 인스턴스 : 게임이 동작하는 프로그램
- 월드 : 엔진에 의해 만들어진 가상 세계. 여기서 게임이 진행
- 게임 모드 : 월드에서 게임을 진행하기 위해 만든 룰
- 레벨 : 게임 진행을 위해 월드에 배치한 액터의 묶음
  - 퍼시스턴트 레벨 : 제작자가 에디터를 통해 배치한 첫 레벨
  - 스트리밍 레벨 : 실시간으로 월드에서 추가 로딩 가능한 레벨
- 플레이어 : 게이머를 대변하는 액터의 묶음. 컨트롤러와 폰으로 구성.

---

# 게임 플레이 프레임 워크



# 언리얼에서 플레이어란?

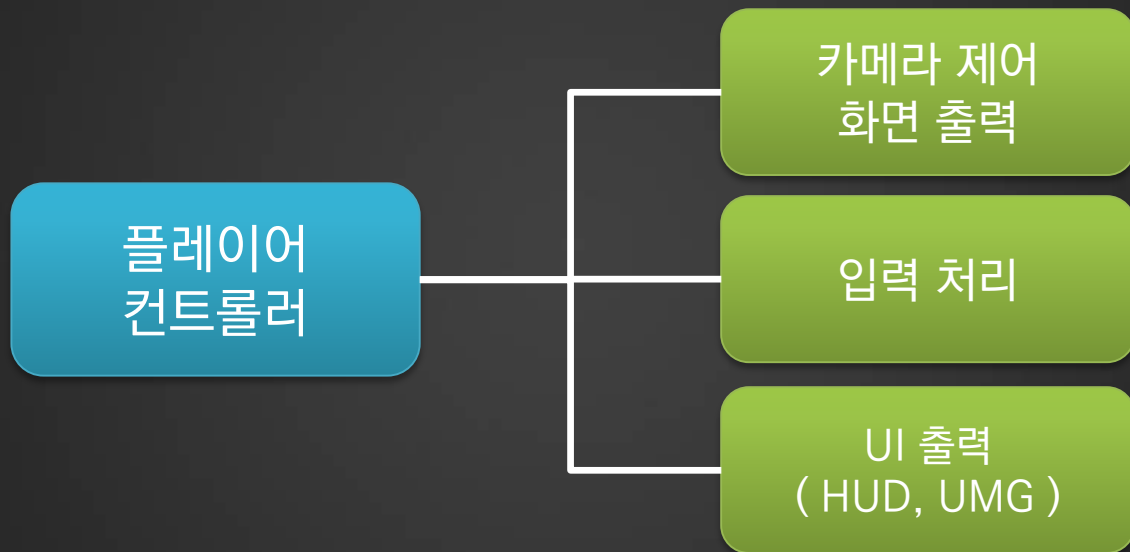
- 현실 세계의 게이머를 대변해주는 액터 조합
- 플레이어는 폰과 플러에어 컨트롤러로 구분된다.
  1. 폰(Pawn)이란?
    - 플레이어가 조종 가능한 액터의 기본 형
    - 움직임에 대한 로직 수행
  2. 플레이어 컨트롤러(Controller)가 하는 일
    - 현실 세계 게이머의 의지를 입력 기기를 통해 폰에게 전달
    - 가상 세계의 카메라 시야를 현실 세계 게이머의 TV, 모니터에 전달
- 플레이어마다 숫자 아이디를 가지고 있다.
  - 로컬 플레이어는 무조건 0번을 배정받음.

현실 세계의 사람



# 플레이어 컨트롤러가 하는 일

---

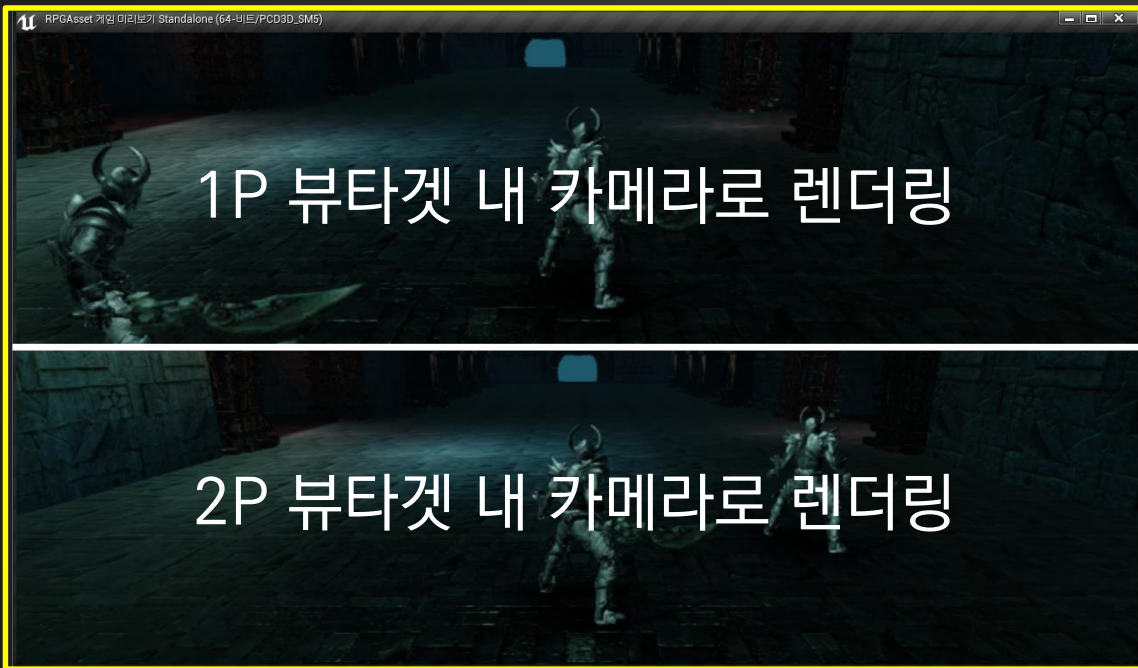


# 언리얼의 화면 출력 시스템

---

- 게임 뷰포트 클라이언트(GameViewportClient)
  - 게임 인스턴스가 관리하는 화면 출력 영역. 인스턴스와 1:1 매칭
  - 플레이어에게 영역을 분양
    - 1인용 게임인 경우 전체 화면을 분양
    - 플레이어가 추가되면 화면을 분할해서 각 플레이어에게 분양
- 뷰 타겟(View Target)
  - 플레이어에 분양된 화면은 컨트롤러에 연결된 카메라를 통해 채워지게 됨.
  - 플레이어 카메라는 특정 액터에 위치해야 함
  - 플레이어 카메라를 포함하는 액터를 특별히 뷰 타겟(ViewTarget) 이라고 함
  - 컨트롤러가 제어하는 폰에 카메라가 있으면 폰은 자동으로 컨트롤러의 뷰 타겟이 됨

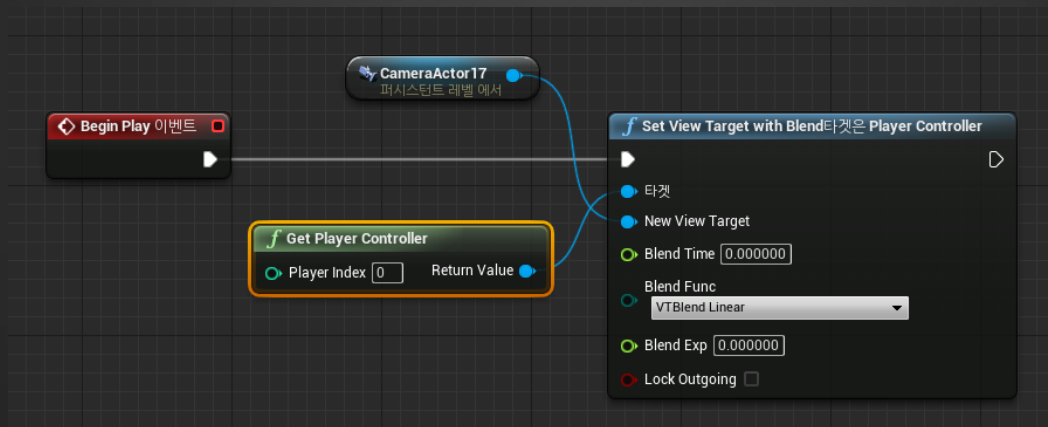
# 게임 뷰포트 클라이언트란?



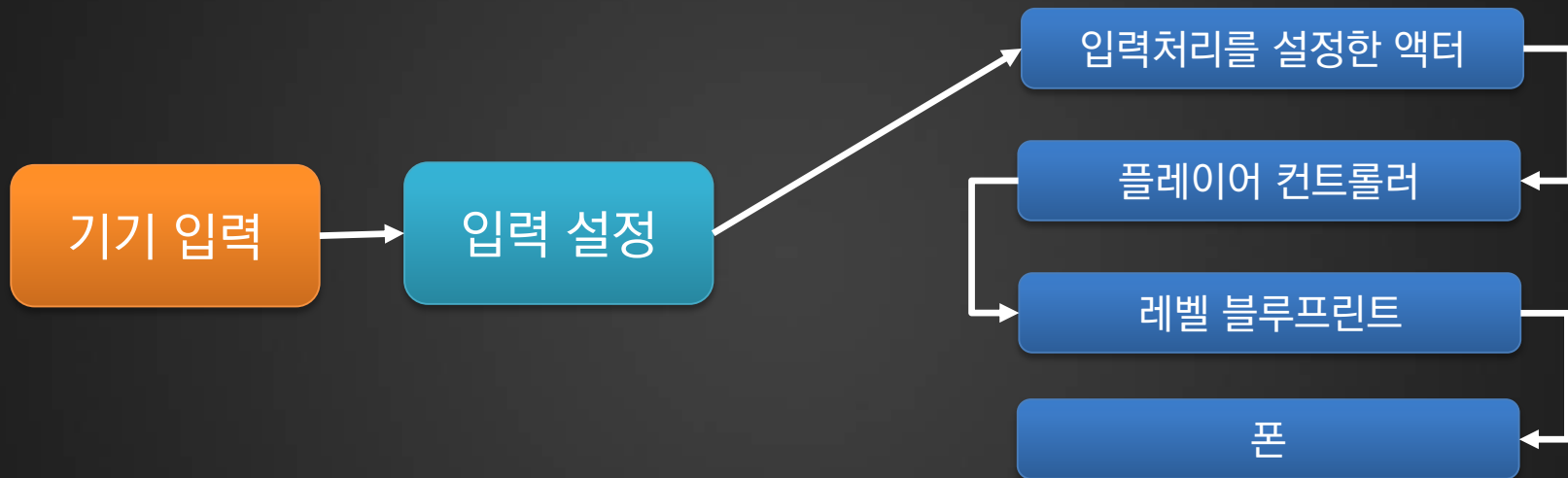
→ GameViewportClient

# 뷰 타겟(View Target)

- 플레이어의 카메라를 담당하는 액터
- 폰에 카메라 컴포넌트를 배치하면 폰은 자동으로 뷰 타겟으로 설정된다.
- 카메라 시야를 폰과 관련 없이 만들고 싶은 경우

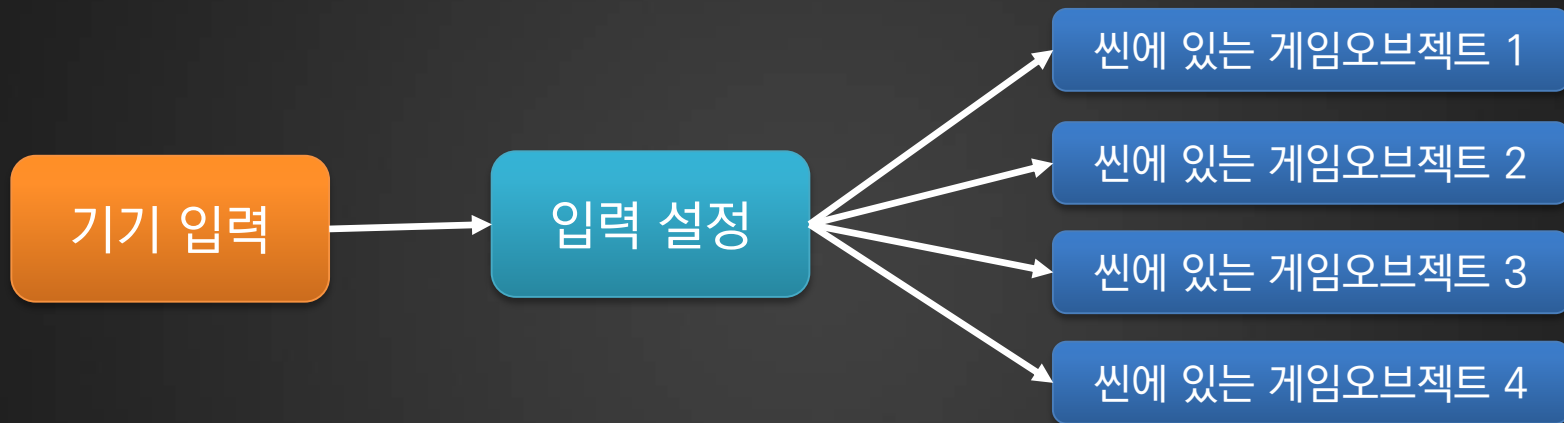


# 언리얼 입력 시스템



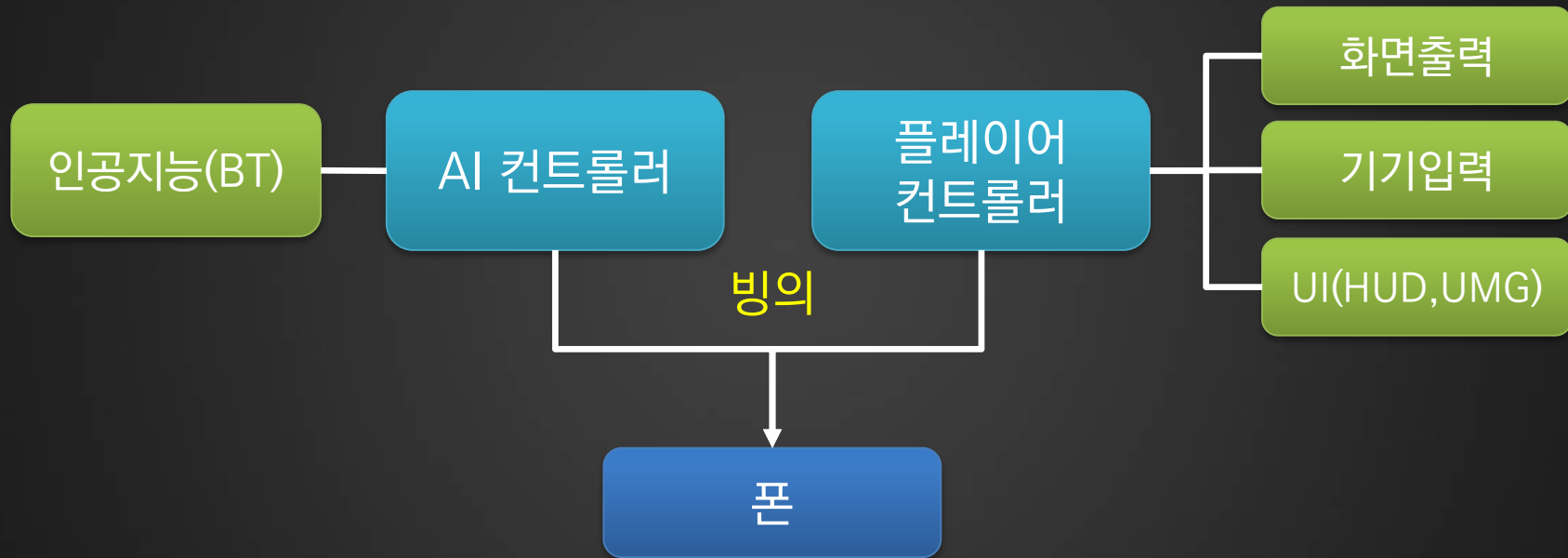
우선 순위 높은 액터가 입력을 처리하면 뒤에는 전달되지 않음

# 유니티 입력 시스템



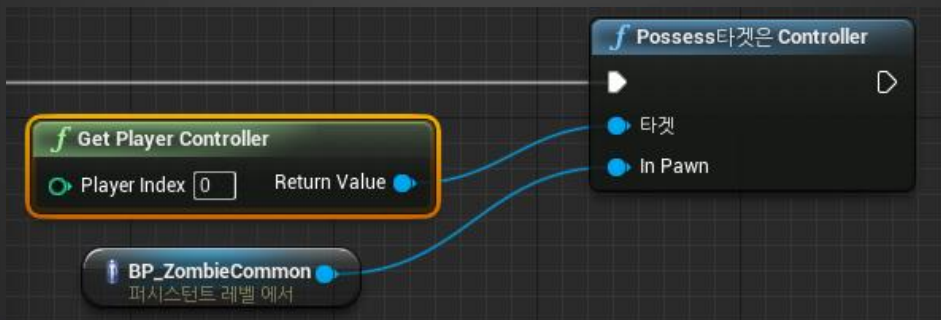
특정 게임 오브젝트에 입력 이벤트가 전달되지 않도록  
막을 방법이 없음

# 플레이어의 구성과 NPC로의 확장





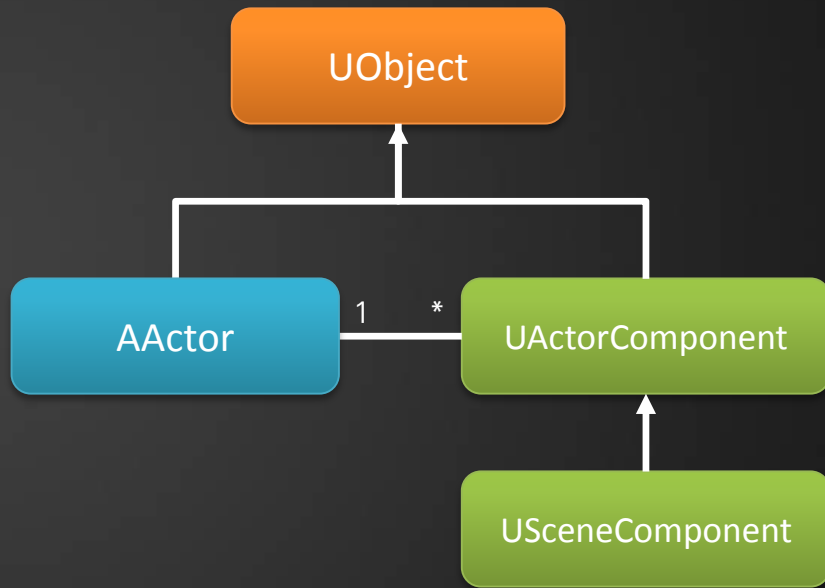
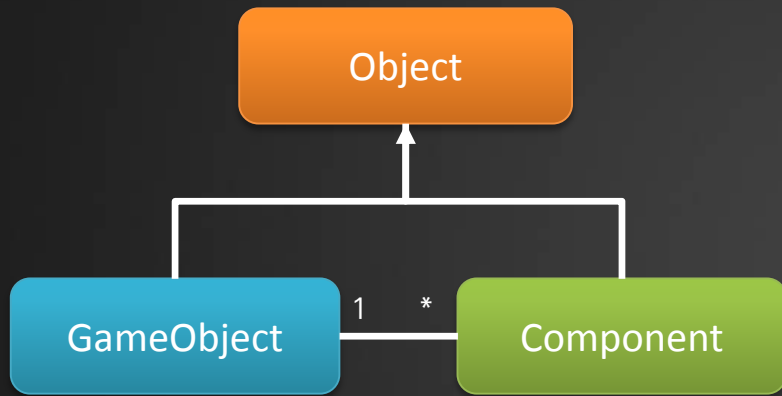
# 빙의(Possession)



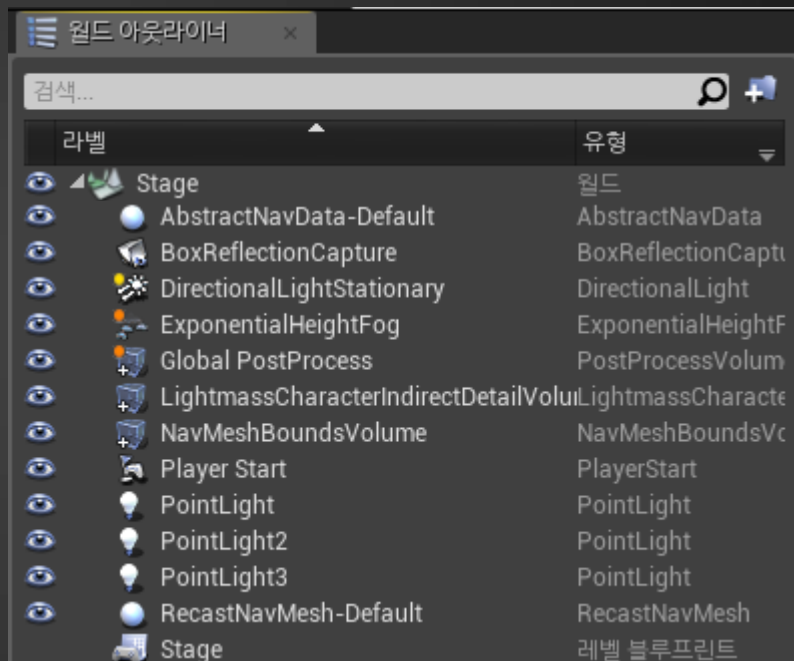
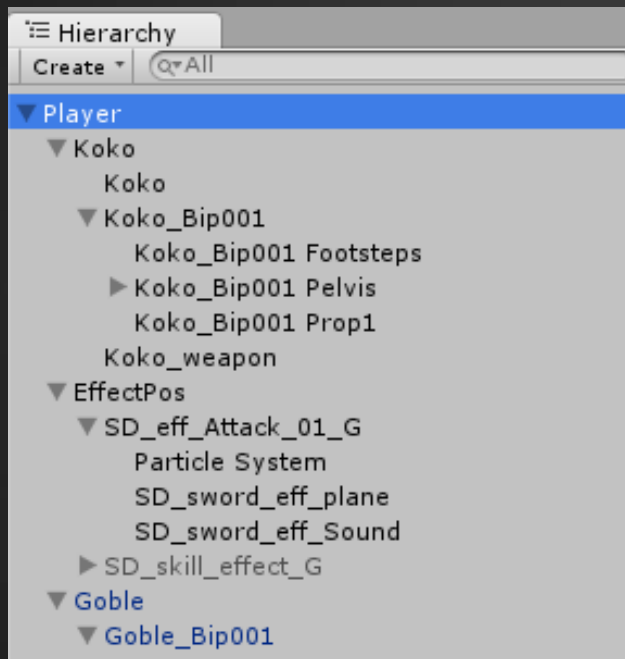
---

# 엔진 아키텍처의 비교

# 코어 오브젝트 구성



# 유니티 계층 뷰 vs 언리얼 월드 아웃라이너



# 유니티 게임 오브젝트의 특징

---

- 3차원 공간에 독립적으로 존재하는 물체의 표현
- 빈 게임 오브젝트는 약방의 감초
  - 캐릭터, 배경, 적
  - 캐릭터의 본
  - 게임 관리자
- 게임 오브젝트의 조합
  - 복잡한 객체의 제작을 위해 게임 오브젝트간의 계층 구조를 직접 설계해야 한다.

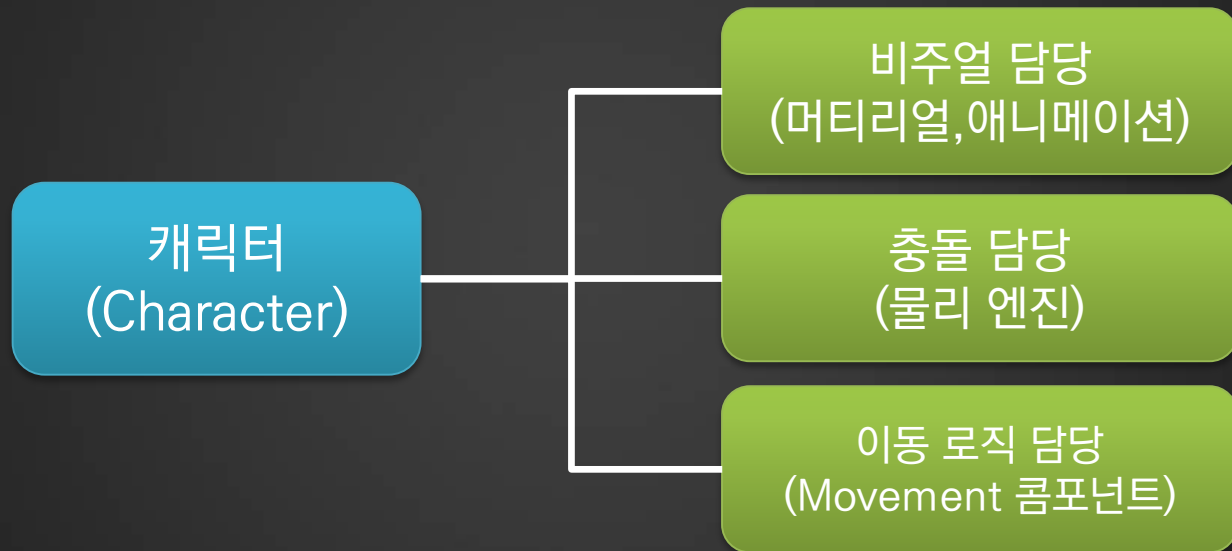
유니티 개발자 별로 다양한 캐릭터 제작 방식이 있음

# 언리얼 액터

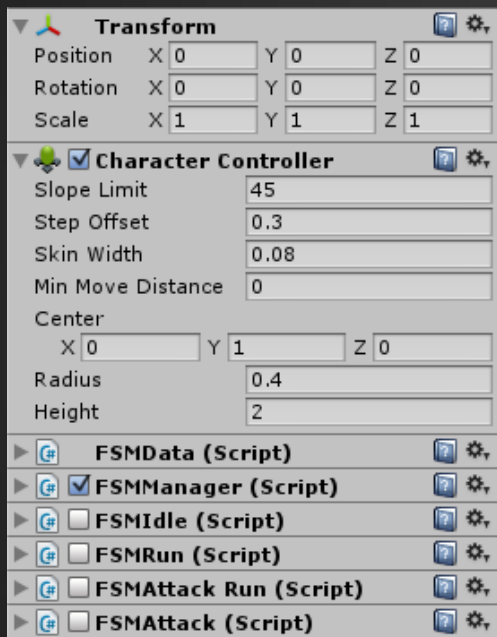
---

- 언리얼 액터의 정의
  - 게임 내에서 명확한 자기 역할과 기능을 가지는 단위
  - 복잡한 물체의 설계를 위한 계층 구조는 액터 내에서 해결.
  - 따라서 월드 아웃라이너는 목록형
  - 본은 스켈레탈 메시 시스템에서 별도로 관리
  - 게임 관리자 역할은 이미 지정되어 있음
- 유니티 게임 오브젝트와의 차별점
  - 유니티는 설계를 오로지 게임 오브젝트로 해결 해야함.
  - 언리얼은 역할에 따라 다양한 액터를 제공하며, 사용자가 직접 파생해서 자신만의 액터를 제작하는 것이 가능.

# 언리얼에서의 캐릭터 구성



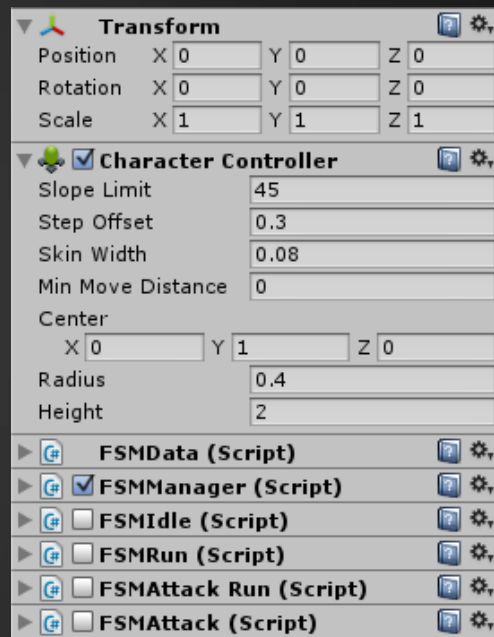
# 유니티 컴포넌트 vs 언리얼 컴포넌트





# 유니티에서의 콤포넌트란?

- 게임 오브젝트에 종속된 객체
- 콤포넌트별로 독립된 로직이 있음
- 게임 오브젝트 안에서는 콤포넌트는 모두 평등한 존재
- 게임 오브젝트에는 로직을 넣을 수 없어서 콤포넌트에서 이를 해결해야 함.
- 콤포넌트가 많아 질 경우 이를 관리할 리더를 따로 설계해야 함. ( Script Execution Order 설정 )



# 언리얼에서 콤포넌트란?

- 액터에 종속된 객체 ( SubObject )
- 콤포넌트별로 독립된 로직이 있음
- 언리얼에서 콤포넌트는 **두 가지 타입**으로 구분됨
  1. 부모로부터 상대적으로 위치한 타입 ( SceneComponent )
  2. 위치와 상관 없이 로직만을 수행하는 타입 ( ActorComponent )
    - 유니티 MonoBehaviour와 일치하는 콘셉
- 유니티와 다른 점
  - 액터에 로직을 부여하는 공간이 별도로 있음.
  - 액터 로직을 콤포넌트에 의존할 필요가 없음.
  - 액터 로직은 콤포넌트 로직보다 먼저 실행됨.



# 유니티에서의 게임 관리

---

- 유니티는 게임을 관리해주는 관리자가 없음.
- 유니티 엔진이 제공해주는 기능들
  - 프로그램 실행 시 먼저 시작되는 씬의 설정 ( 0번 씬 )
  - 씬이 바뀌어도 유지되는 게임 오브젝트의 설정 ( DontDestroyOnLoad )
  - 로직은 콤포넌트에서만 구현 가능
  - 콤포넌트의 우선 순위 설정 ( Script Execution Order )
  - 어플리케이션 실행과 관련된 콤포넌트의 이벤트 ( OnApplicationQuit )

유니티 개발자별로 다양한 게임의 관리 방식이 존재

# 엔진 간 이벤트 함수 비교

## 유니티의 콤포넌트 이벤트

- 에디터함수(OnDrawGizmos)
- 생성자(Awake)
- 시작(Start)
- 활성화(OnEnable)
- 프레임(Update)
- 물리프레임(FixedUpdate)
- 애니메이션 이후 프레임(LateUpdate)
- 비활성화(OnDisable)
- 종료(OnDestroy)
- 프로그램 종료(OnApplicationQuit)

## 언리얼의 게임 인스턴스 이벤트

- 프로그램 시작 ( Init )
- 프로그램 종료 ( Shutdown )

## 언리얼의 액터 이벤트

- 에디터함수(Construction Script)
- 시작(Begin Play)
- 종료(Destroyed)
- 프레임(Tick)

## 언리얼의 콤포넌트 이벤트

- 시작(Initialize Component)
- 프레임(Tick)

# 정리

---

- 유니티의 특징

- 빈 씬에 카메라 컴포넌트가 들어간 빈 게임 오브젝트를 제공
- 컴포넌트와 제공되는 이벤트를 활용해 개발자들이 스스로 제작
- 게임 오브젝트와 계층 구조, 스크립트 실행 순서를 활용해 모든 것을 해결 해야함
- 자유도는 높으나 구조를 잘못 잡으면 프로젝트가 커지면서 고생할 가능성이 높아짐.

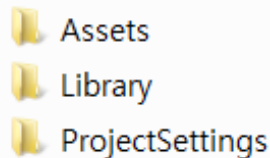
- 언리얼의 특징

- 프로그램 시작부터 게임 시작까지 일련의 과정을 체계적으로 제공
- 중요한 요소들은 이미 정의되어 있으며, 이를 상속받아 확장하는 것이 가능
- 액터와 컴포넌트 모두 로직을 소유하며, 액터 로직이 먼저 실행됨.
- 자유도는 적으나, 가이드라인만 잘 따르면 좋은 구조로 게임이 만들어짐.

---

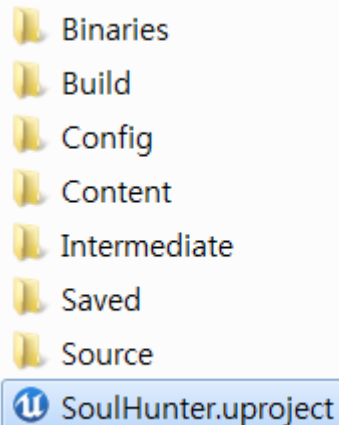
# 프로젝트 관리

# 프로젝트 디렉토리 구성



Assets  
Library  
ProjectSettings

유니티



Binaries  
Build  
Config  
Content  
Intermediate  
Saved  
Source  
SoulHunter.uproject

언리얼

# 언리얼 각 폴더의 의미

- 프로젝트 파일 : 단순하지만 프로젝트에 사용할 추가 모듈을 지정할 수 있음.
- Binaries 폴더 : 빌드한 최종 결과물이 저장
- Build 폴더 : 플랫폼 빌드시 필요한 설정이 들어 있음
- Config 폴더 : 프로젝트 설정.
  - 유니티의 ProjectSettings와 유사.
- Content 폴더 : 프로젝트에서 사용할 애셋의 관리 장소
  - 유니티의 Asset과 유사.
- Intermediate 폴더 : 프로젝트 진행에서 산출된 중간 결과물
  - 유니티의 Library와 유사.
- Saved 폴더 : 변환된 애셋, 세이브 데이터, 스크린 샷 등 보관
- Source 폴더 : C++ 프로젝트의 소스 보관



# 에디터 런칭 과정

---

1. Build 폴더를 참고해 에디터 실행
2. 에디터는 Config와 Content 폴더를 참고해 프로젝트 로딩
3. 프로젝트 작업 중 Intermediate, Saved 폴더에 관련 데이터를 저장
4. 플랫폼 빌드에서 변환된 애셋은 Saved 폴더에 저장

Intermediate와 Saved 폴더는 유니티의 Library폴더

# 버전 컨트롤을 위한 중요 폴더

---

- 프로젝트 파일 : 필수
- Binaries 폴더 : 옵션. C++ 프로젝트 기반인 경우 VS가 없는 사람을 위해 필요
- Build 폴더 : 필수
- Config 폴더 : 필수
- Content 폴더 : 필수
- Intermediate 폴더 : 불필요, 에디터에서 재생성 가능
- Saved 폴더 : 불필요 , 에디터에서 재생성 가능
- Source 폴더 : 필수

# 프로젝트 리소스 관리

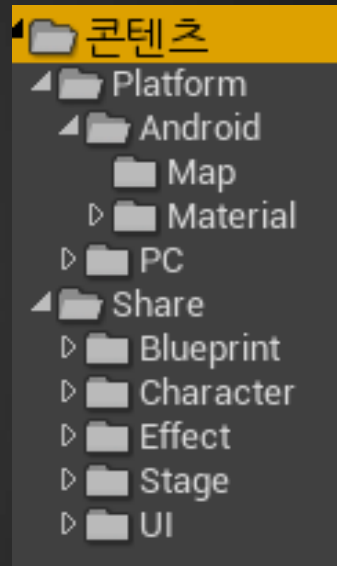
---

- 유니티의 리소스 관리 방법
  - GUID 기반의 관리
- 언리얼의 리소스 관리 방법
  - 경로 기반의 관리.
  - 메타 파일이 필요 없음.

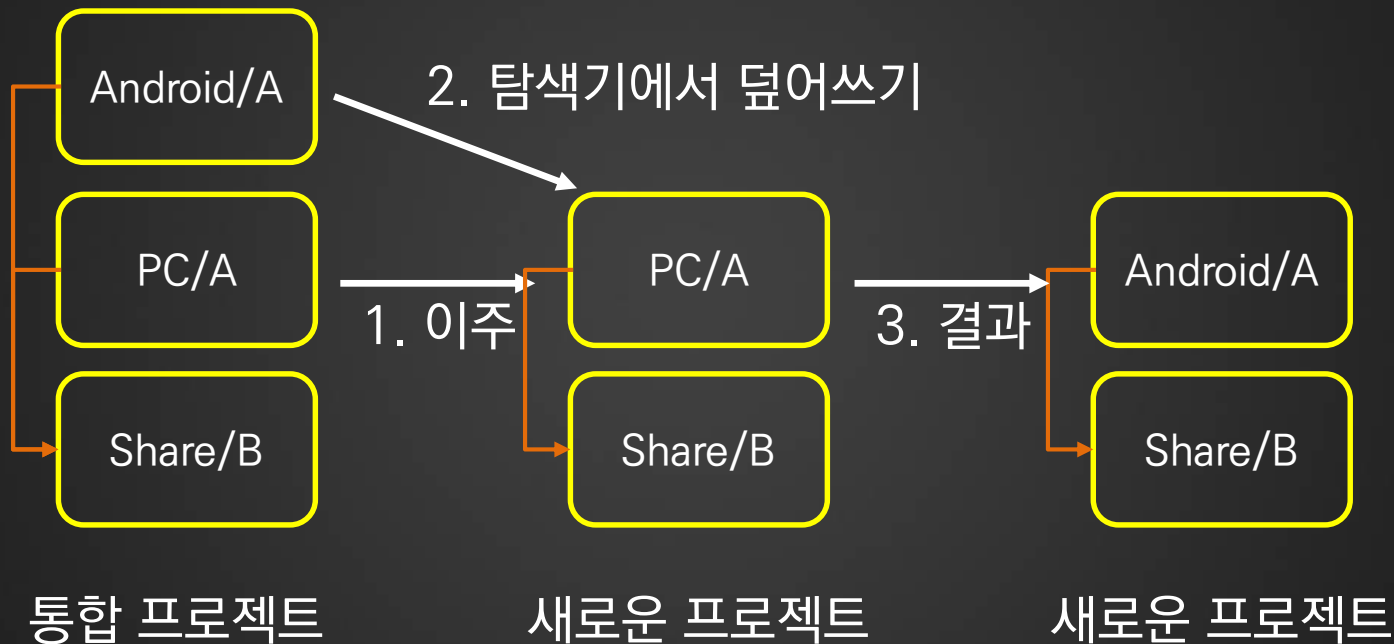
언리얼은 경로 기반으로 관리하기 때문에  
활용하기에 따라 트릭이 가능

# 트릭의 예

- 기본 설정
  - Android 폴더의 A라는 이름의 머티리얼
  - PC 폴더의 A라는 이름의 머티리얼
  - 둘다 Share 폴더에 있는 B라는 Texture 애셋을 사용
- Android만 제외한 새로운 프로젝트를 생성
  - 윈도우 탐색기를 열고 기존 프로젝트의 Android 폴더의 A 머티리얼을 새로운 프로젝트의 PC 폴더의 A 머티리얼에 덮어써도 문제가 발생하지 않음.



# 트릭의 도식화



# 쿠키와 파생 데이터 캐시

---

- 유니티는 임포트시 원본 파일을 그대로 보존.
- 언리얼은 임포트시 원본 파일을 uasset으로 변환.
- 언리얼은 멀티 플랫폼 빌드를 위해 쿠키 컨셉을 제공
- 쿠키이란?
  - uasset 원본 데이터를 플랫폼에 맞게 변환해주는 시스템
- 파생 데이터 캐시(Derived Data Cache)란?
  - 원본 데이터를 참고해 플랫폼에 맞는 포맷으로 변환해 생성한 데이터
  - 파생 데이터 캐시는 팀 안에서 함께 공유해 사용할 수 있음. ( 유니티의 캐시 서버 )

# 애셋의 쿠키 과정



# 정리

---

- 유니티의 특징
  - GUID 기반의 리소스 관리 시스템
  - GUID 키를 에디터에서 생성하므로 관리하기가 까다로움.
  - 캐시 서버를 사용해 플랫폼 변환에 걸리는 시간을 단축
- 언리얼의 특징
  - 경로 기반의 리소스 관리 시스템
  - 쿠키킹과 파생 데이터 캐시 개념 도입
  - 커맨드 라인과 툴을 잘 활용하면 유연한 멀티 플랫폼 프로젝트 생성이 가능



---

# 블루프린트 스크립트에 대한 소고

# 블루프린트 스크립트

---

- 특징
  - 노드 기반의 직관적인 비주얼 인터페이스
  - 프로그래밍에 전문적인 지식이 없어도 로직 제작을 가능하게 제공
  - 유니티 C#에 좌절한 분들에게 적합한 개발 도구 ( Syntax 에러로부터 해방 )
- 실제로 어디까지 사용 가능한가?
  - 엔진의 일부만 사용 가능하지만 간단한 Standalone 게임 정도는 제작 가능.
  - 하지만 실제 서비스되는 상용 프로그램 제작에는 한계가 있음.
    - 부족한 API 기능
    - 누군가 C++로 보완해 준다면 가능하다.

# 언리얼 스크립팅 시스템

---

블루프린트 스크립팅

블루프린트 API

언리얼 C++  
스크립팅

언리얼 C++ 엔진 API

일반 C++

언리얼 VM


# 블루프린트 스크립트는 교육용인가?

- 블루프린트 스크립팅의 장점

- UMG 및 애니메이션 시스템을 사용하기 위해서는 블루프린트를 써야 함
- 실시간으로 컴파일 및 확인 가능 : 생산성 빠름.

- 언리얼의 빌드 시스템

- C++ 코드는 컴파일을 한 후 최종 Shared Object에 모두 묶어서 배포함

이름	수정한 날짜	유형	크기
 libUE4.so	2015-04-08 오후 6:58	SO 파일	127,680KB

- 블루프린트 스크립트 코드는 애셋 내부에 저장
- 블루프린트를 사용한다면 애셋에 로직을 담아 개별적으로 배포하는 것이 가능

# 정리

---

- 블루프린트 스크립트 사용 권장
  - 게임 엔진 위에서 직접 작업한 결과를 변경하고 싶은 아트 직군
  - 게임 내 레벨 디자인을 다양하게 변경하고 싶은 기획 직군
  - 언리얼 엔진의 기능을 빠르게 파악 사용해보고 싶은 분.
  - 프로그래밍 로직에 대해 감을 잡고 싶은 분
    - 블루프린트를 파악한 후에 C# 혹은 C++로 도전
  - 소규모로 단순한 메카닉을 빠르게 프로토타이핑 해보고 싶은 팀
  - 기획 / 아트 / 프로그래밍 파트 모두 전방위적으로 게임 엔진을 활용하고자 하는 팀

블루프린트 교육 문의 : [dustin@indp.kr](mailto:dustin@indp.kr)

감사합니다!