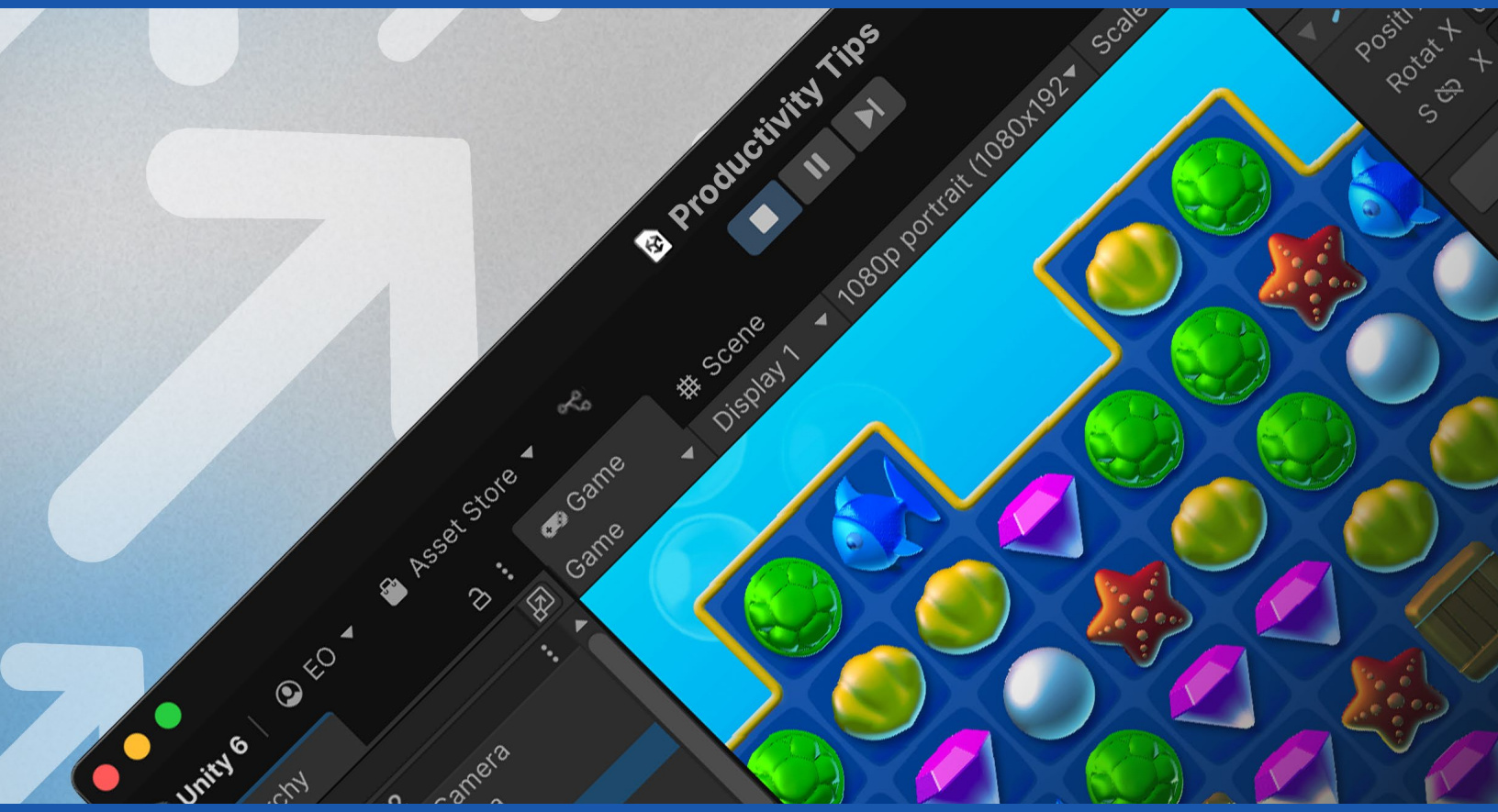
→ 전자책

Unity 6로 생산성 높이기 팁



Unity®

목차

들어가는 말

이 가이드는 아티스트와 프로그래머가 Unity 툴셋으로 더 빠르게 작업하는 데 필요한 다양한 팁을 제공합니다. Unity 6의 최신 기능과 함께, 오랫동안 Unity에서 활용되고 있는 효율적인 작업 단계와 워크플로를 소개합니다.

혼자 작업하든 팀과 함께하든 Unity를 매일 활용하다 보면 1초의 작업이나 1번의 마우스 클릭이 하나둘씩 쌓여서 큰 차이를 만들게 됩니다. 이제 시간을 절약하고 생산성을 높여 보세요. Unity 개발 숙련도에 관계없이 누구나 이 가이드를 참조하면 게임 개발의 모든 단계에서 워크플로 속도를 높일 수 있습니다.

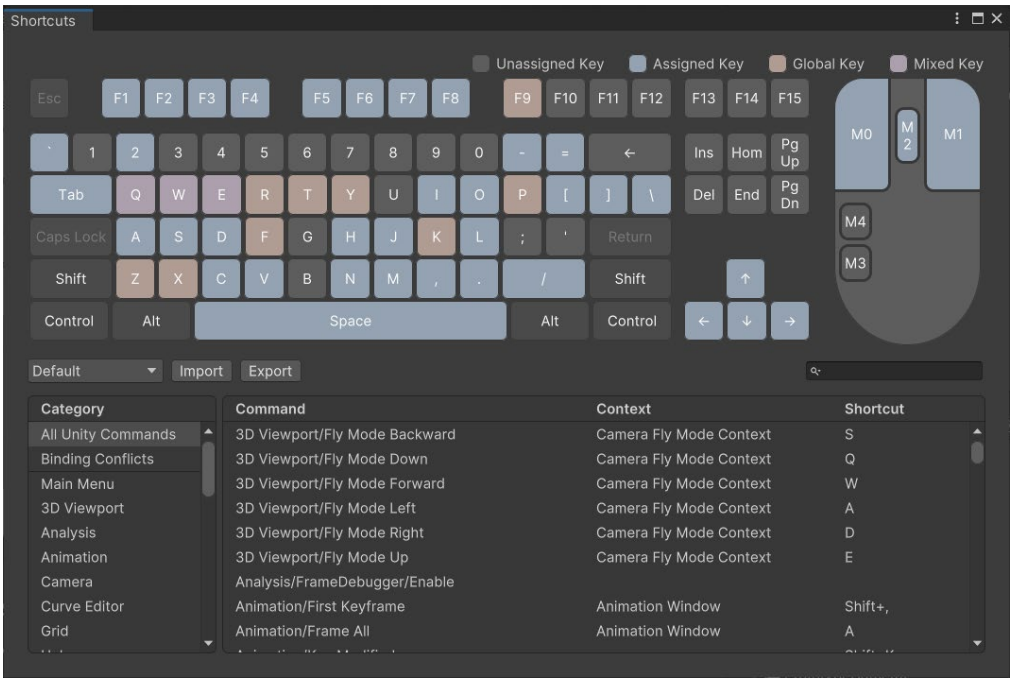
Unity 6의 새로운 기능에 대해 알아보려면 Unity 6 웹 페이지를 살펴보거나 Unity 기술 자료의 Unity 6.0의 새로운 기능 섹션 및 Unity 6.1의 새로운 기능 섹션을 참조하세요.

이 가이드가 도움이 되기를 바랍니다.

에디터 워크플로

자주 사용하는 키보드 단축키 손쉽게 커스터마이즈하기

단축키 관리자는 에디터 단축키를 확인하고, 커스터마이즈하고, 관리할 수 있는 인터랙티브 비주얼 인터페이스입니다. 여기에서 Global, 씬(Scene) 뷰, 특정 에디터 툴 같은 다양한 컨텍스트에 단축키를 할당하고, 자주 사용하는 툴에 대한 기존의 바인딩을 시각화할 수 있습니다. 팀 또는 기기 간에 단축키 프로파일을 임포트/익스포트할 수도 있습니다.



단축키 관리자

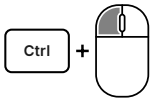
다음은 Unity 메인 메뉴에서 단축키 관리자에 액세스하는 방법입니다.

— Windows 및 **Linux**: **Edit > Shortcuts** 선택

— macOS: **Unity > Shortcuts** 선택

일반적인 단축키

아래는 몇 가지 일반적인 기본 단축키입니다.

액션	Windows	Mac
선택된 프레임	F	
항목 복제	Ctrl + D	⌘ + D
게임 오브젝트 삭제	Shift + Del	⌘ + Del
보기/이동/회전/사각/트랜스폼	Q W E R T	
피벗 모드 토글	Z	
피벗 회전 토글	X	
버텍스 스냅	V	
스냅	Ctrl + 	⌘ + 
씬 뷰에서 선택한 오브젝트 분리	Shift + H	
최대화 토글	Shift + Space	
컨텍스트에서 프리팹 편집	P	

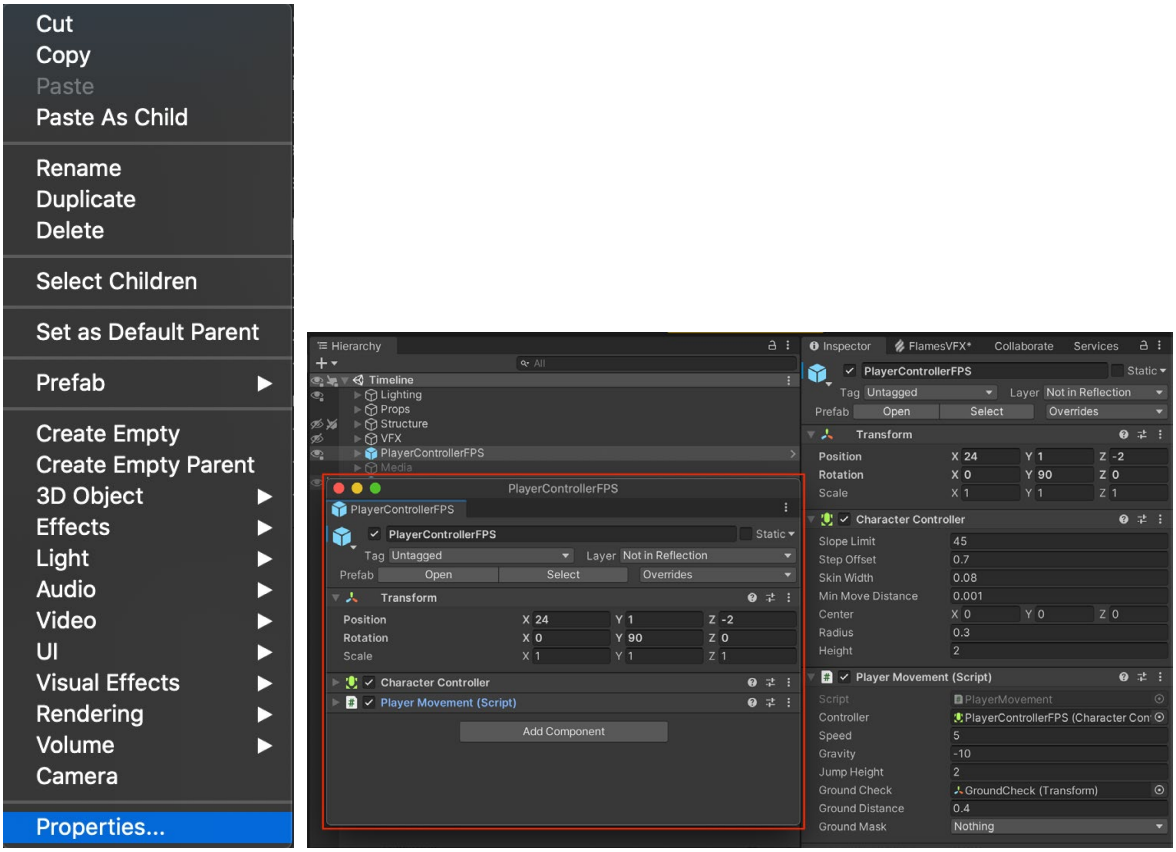
Command	Shortcut
View	Q
Move	W
Rotate	E
Scale	R
Rect	T
Transform	Y
Toggle Pivot Position	Z
Toggle Pivot Orientation	X

일반 에디터 단축키

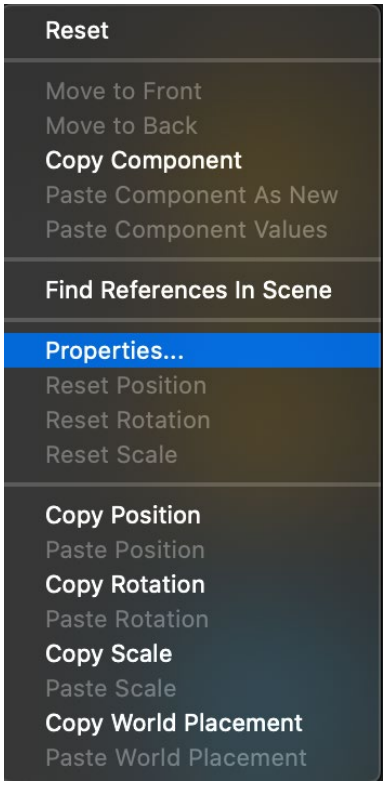
여러 인스펙터로 작업

전용 인스펙터(Focused Inspector) 창에서는 한 번에 여러 개의 인스펙터 창을 표시하여 특정 게임 오브젝트, 컴포넌트, 에셋의 프로퍼티를 볼 수 있습니다. 전용 인스펙터 창은 씬에서 다른 항목을 선택하더라도 항상 선택된 항목의 프로퍼티를 표시합니다.

게임 오브젝트 또는 컴포넌트를 오른쪽 클릭하고 **Properties**를 선택합니다. 그러면 다른 창처럼 위치를 옮기거나 고정하거나 크기를 조정할 수 있는 플로팅 인스펙터 창이 나타납니다.



두 게임 오브젝트를 비교하는 전용 인스펙터



여러 개의 전용 인스펙터 창을 동시에 열면 씬을 변경하는 동안 여러 게임 오브젝트를 참고하거나 나란히 두고 비교할 수 있습니다.

또한 게임 오브젝트의 특정 컴포넌트만 표시할 수도 있으며, 이 경우 창이 차지하는 스크린 공간을 줄일 수 있습니다.

프리셋으로 선호하는 기본 설정 정의

프리셋을 사용하면 인스펙터의 컴포넌트 또는 에셋 기본 상태를 커스터마이즈하고 재사용할 수 있습니다. 프리셋을 만들면 Unity가 컴포넌트 또는 에셋의 현재 설정을 **.preset 에셋**으로 저장합니다.

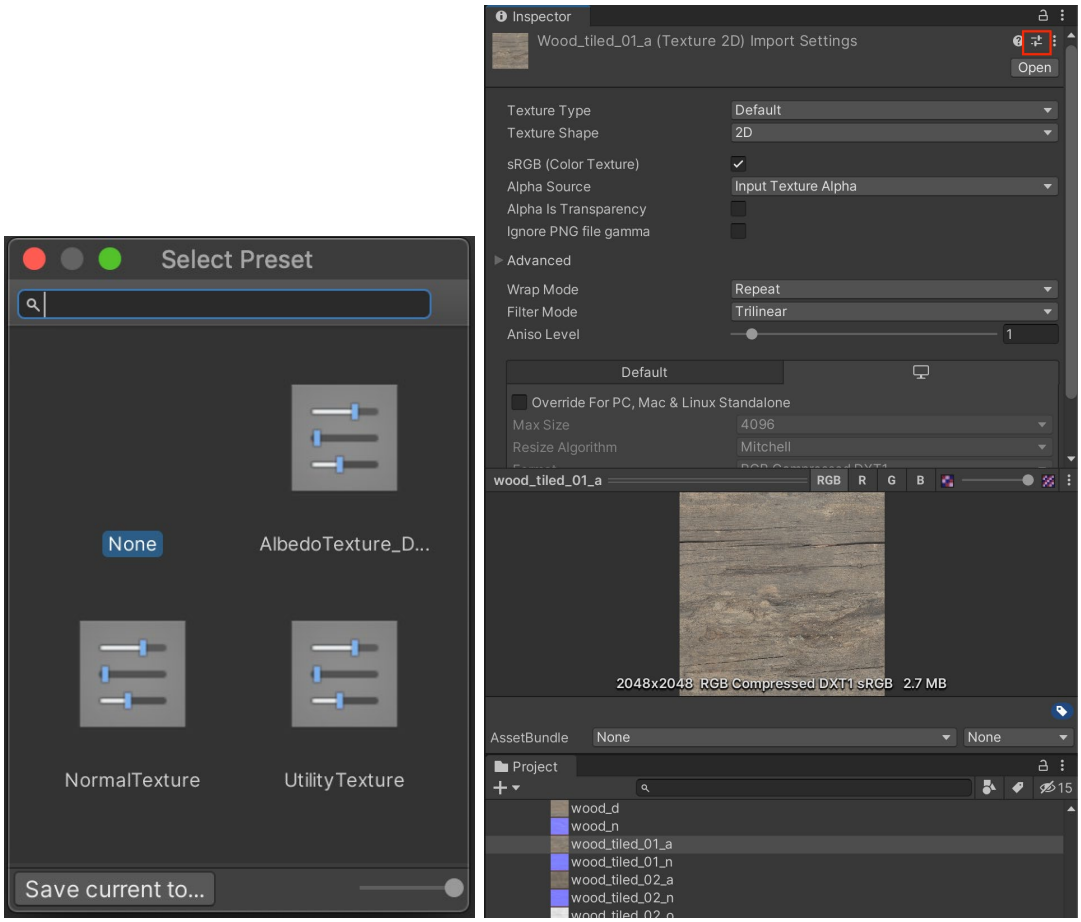
이 구성을 동일한 유형의 다른 컴포넌트 또는 에셋에 적용할 수 있습니다.

프리셋을 사용하여 표준 설정을 강제하거나 새로운 에셋에 원하는 기본 설정을 적용할 수 있습니다. 프리셋을 통해 팀 전체가 일관된 표준 설정을 사용할 수 있으므로, 프로젝트 결과에 영향을 미칠 수 있는 흔한 설정 오류를 미연에 방지합니다.

프리셋 아이콘(빨간 상자 표시)

프리셋을 만드는 방법은 다음과 같습니다.

1. 컴포넌트의 오른쪽 상단에 있는 **프리셋** 아이콘을 클릭합니다.
2. **Save current to...**를 클릭하여 프리셋을 에셋으로 저장한 다음,
3. 사용 가능한 프리셋을 클릭하여 일련의 값을 로드합니다.



2D 텍스처의 용도(알베도, 일반, 유틸리티)에 따라 다른 임포트 설정을 포함하는 프리셋 예시

프리셋을 활용하는 그 밖의 방법은 다음과 같습니다.

— **기본값으로 게임 오브젝트 생성:** 계층(Hierarchy) 창에 프리셋 에셋을 드래그 앤 드롭하면 해당 컴포넌트가 프리셋 값으로 미리 채워진 새로운 게임 오브젝트를 생성할 수 있습니다.

— **특정 유형과 프리셋 연결: Preset Manager**(**Project Settings > Preset Manager**)에서 유형당 하나 이상의 프리셋을 지정하고 새로운 컴포넌트를 만들면 지정된 프리셋 값이 기본값으로 설정됩니다.

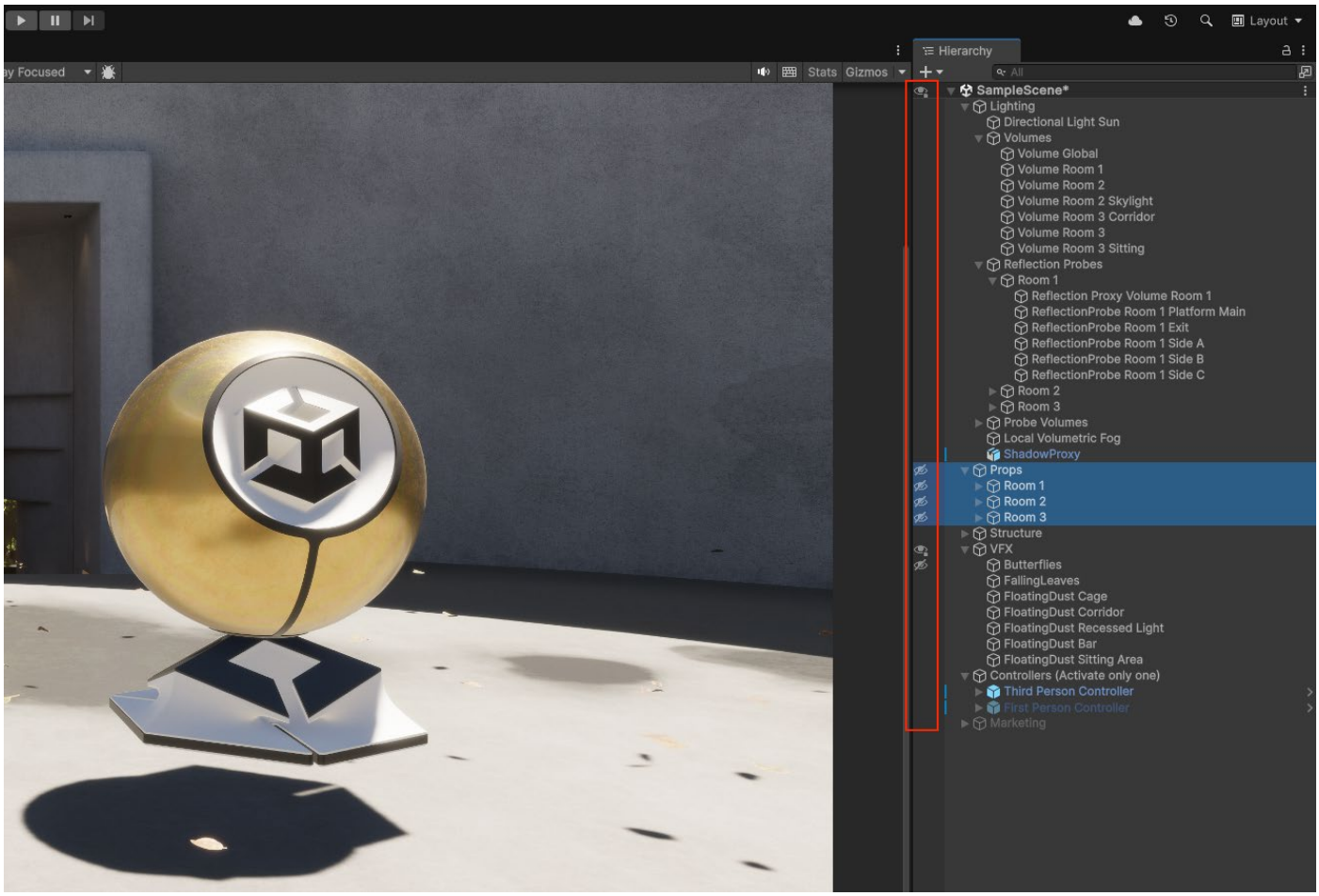
— **필터를 사용하여 여러 프리셋 중 선택:** 동일한 유형에 대해 여러 개의 프리셋을 만들고 Preset Manager에서 필터를 사용하여 어느 프리셋을 적용할지 결정할 수 있습니다.

— **관리자 설정 저장 및 재사용:** 관리자 창에 프리셋을 사용하면 설정을 재사용할 수 있습니다. 예를 들어 동일한 태그 및 레이어와 물리 설정을 다시 적용하려는 경우 프리셋을 사용하면 다음 프로젝트에서 설정 시간을 절약할 수 있습니다.

씬 가시성으로 씬 뷰 정리

씬의 규모가 커짐에 따라 특정 오브젝트를 일시적으로 숨겨서 게임 오브젝트를 더 쉽게 선택하고 편집할 수 있습니다.

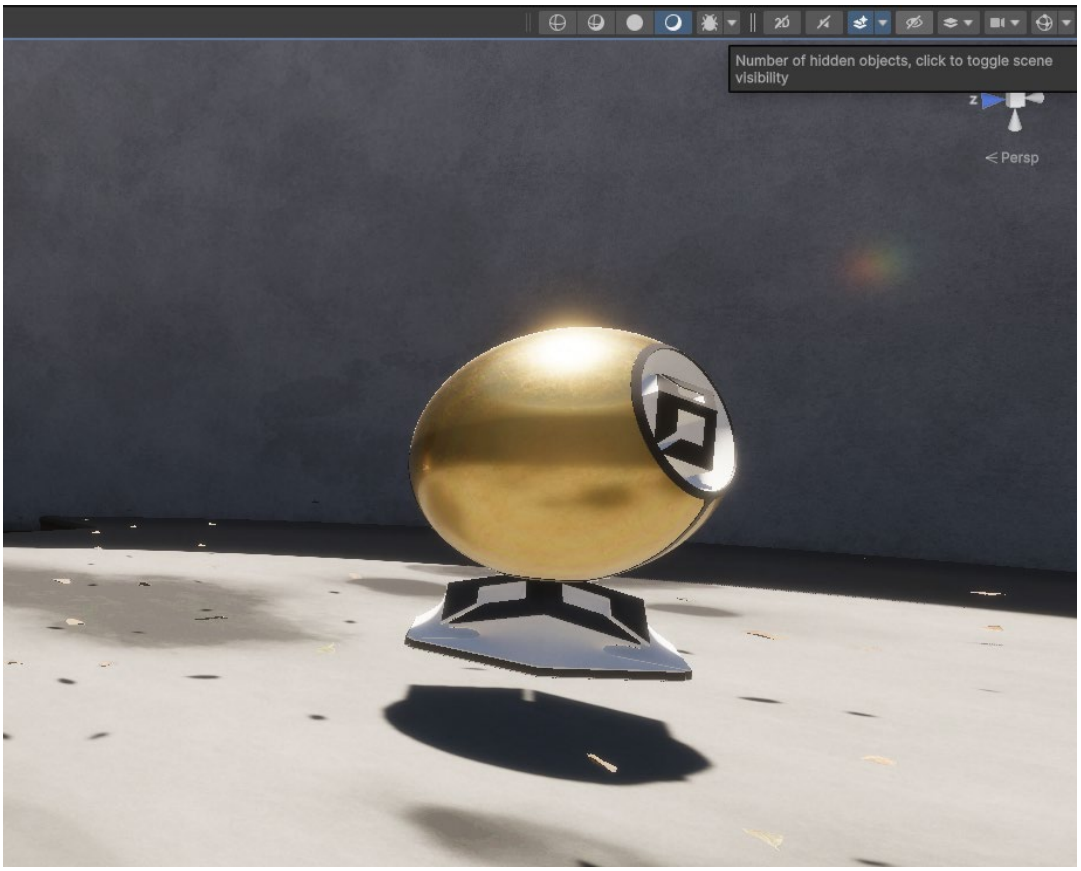
게임 오브젝트를 비활성화하는 대신(비활성화하면 의도치 않은 동작으로 이어질 수 있음) 씬 가시성 컨트롤을 사용하여 씬 뷰의 가시성을 토글할 수 있습니다. **계층** 창에서 게임 오브젝트 옆의 눈 아이콘을 클릭하여 씬 가시성을 토글합니다. 이렇게 하면 씬 뷰에서 오브젝트가 숨겨지며, 오브젝트의 활성 상태나 인게임 가시성에는 영향을 미치지 않습니다.



씬 가시성 컨트롤을 사용하여 씬 뷰에서 오브젝트를 숨깁니다.

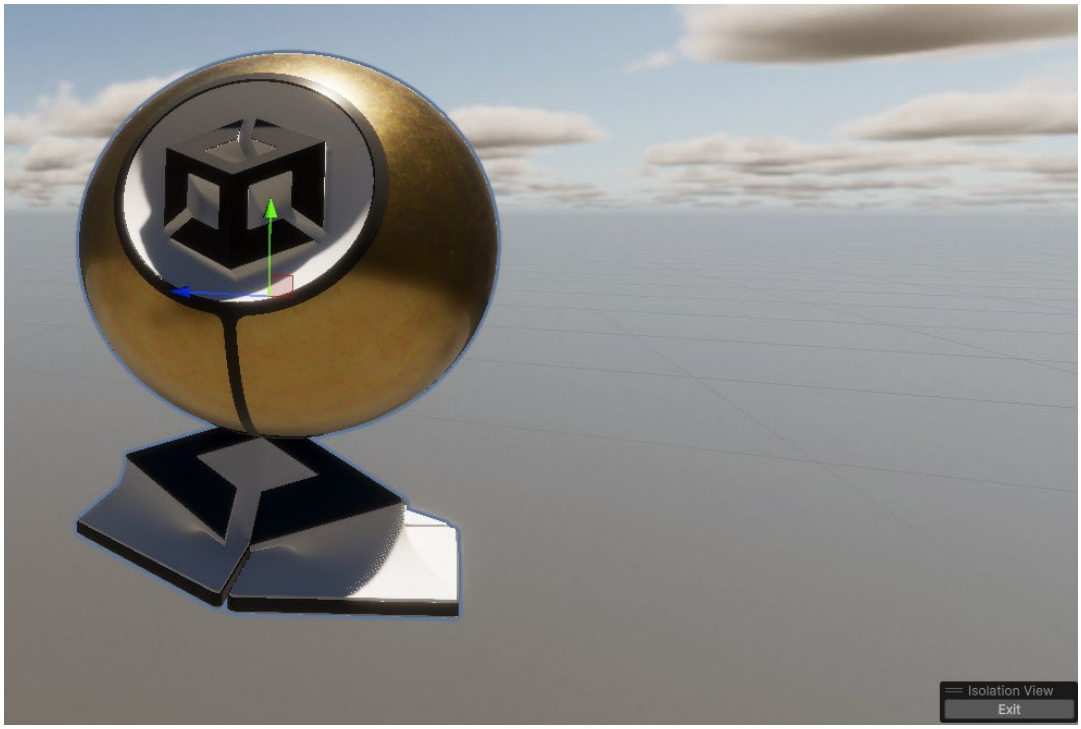
숨겨진 오브젝트가 부모인지 자식인지에 따라 계층 구조의 상태 아이콘이 달라집니다.

아이콘	상태
	게임 오브젝트는 표시하지만 게임 오브젝트의 일부 자식은 숨깁니다.
	게임 오브젝트는 숨기지만 게임 오브젝트의 일부 자식은 표시합니다.
	게임 오브젝트와 게임 오브젝트의 자식을 표시하지만 게임 오브젝트에 마우스 커서를 올려놓을 때만 표시됩니다.
	게임 오브젝트와 게임 오브젝트의 자식을 숨깁니다.



씬 뷰 컨트롤 바를 토글하여 전역 씬 가시성을 오버라이드합니다.

아이솔레이션 뷰를 사용하여 특정 오브젝트와 그 자식 오브젝트에만 집중할 수도 있습니다. 계층 창에서 게임 오브젝트를 선택하고 **Shift+H**를 눌러 아이솔레이션 뷰를 켜거나 끌 수 있습니다. 이렇게 하면 종료할 때까지 다른 씬 가시성 설정이 오버라이드됩니다.



게임 오브젝트를 집중하여 편집할 수 있는 아이솔레이션 뷰

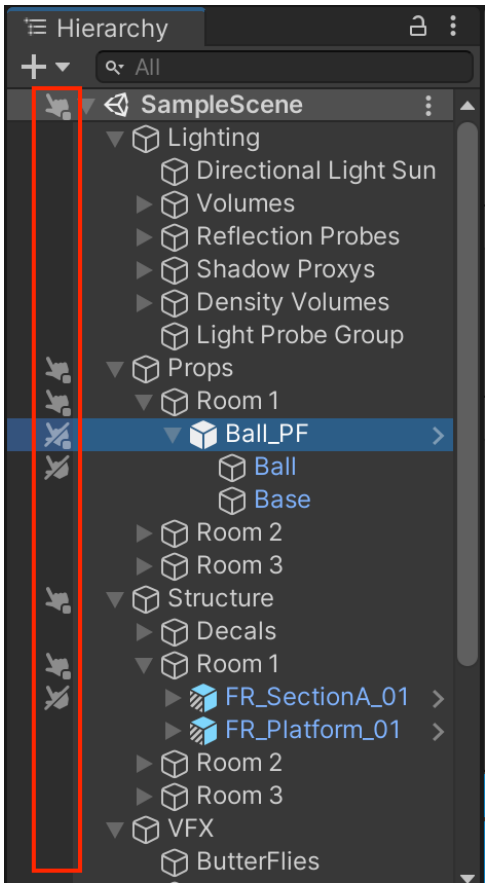
Shift+스페이스바 단축키를 사용하여 언제든지 뷰포트를 최대화하고 나머지 에디터를 숨길 수 있습니다.

대규모 스트리밍 월드를 만들거나 런타임에 여러 씬을 효과적으로 관리해야 하는 경우 Unity에서 여러 씬으로 작업할 수도 있습니다. 라이트맵, NavMesh 데이터, 오클루전 컬링 데이터 같은 데이터를 동시에 여러 씬에 베이크할 수 있습니다. 에디터 내에서 또는 런타임에 스크립트를 사용하여 여러 씬을 편집할 수도 있습니다.

씬 뷰에서 잘못된 오브젝트 선택 방지

Unity에서는 씬 가시성을 관리하는 방식처럼 씬 뷰에서 특정 게임 오브젝트의 선택 가능 여부를 제어할 수 있습니다.

씬 가시성 툴바에서 게임 오브젝트의 선택 가능 여부를 토글하여 씬 뷰에서 특정 게임 오브젝트가 선택되는 것을 차단할 수 있습니다. 이는 복잡한 대규모 씬에서 의도치 않은 게임 오브젝트를 선택하고 편집하는 일을 방지하는 데 유용합니다.



계층 구조 선택 가능 여부

전체 브랜치 또는 단일 오브젝트를 대상으로 선택 가능 여부를 토글할 수 있으므로, 게임 오브젝트를 선택할 수 있더라도 자식이나 부모 오브젝트는 선택이 불가능할 수 있습니다. 다음은 선택 가능 여부 상태를 표시하는 아이콘과 각 아이콘에 대한 설명입니다.

아이콘	상태
	게임 오브젝트는 선택할 수 있지만 게임 오브젝트의 일부 자식은 선택할 수 없습니다.
	게임 오브젝트는 선택할 수 없지만 게임 오브젝트의 일부 자식은 선택할 수 있습니다.
	게임 오브젝트와 게임 오브젝트의 자식을 선택할 수 있습니다. 마우스 커서를 올려놓을 때만 게임 오브젝트가 표시됩니다.
	게임 오브젝트나 게임 오브젝트의 자식을 선택할 수 없습니다.

인스펙터에서 곧바로 에셋 미리 보기

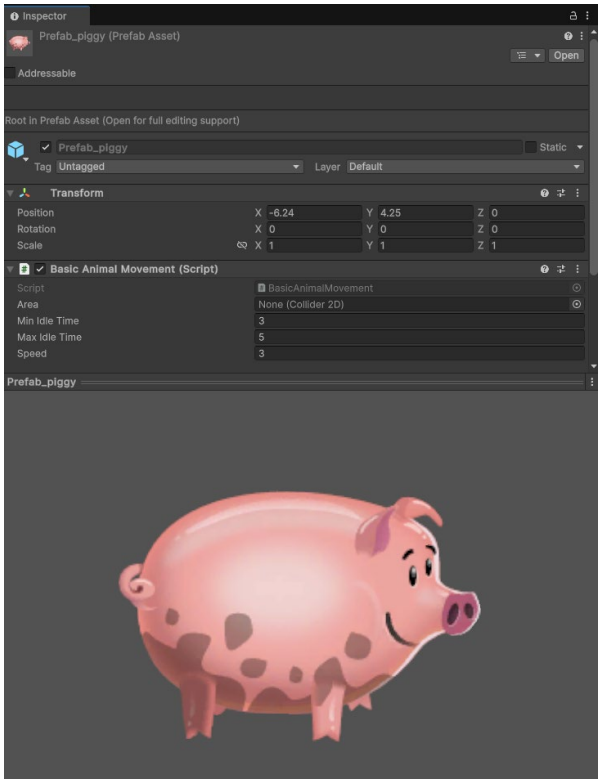
에셋 미리 보기는 3D 모델, 머티리얼, 텍스처 같은 에셋의 미리 보기 또는 썸네일을 볼 수 있는 비주얼 패널로, Unity 에디터의 인스펙터 창 하단에 있습니다. 에디터에서 작업할 때 에셋 미리 보기를 활용하면 빠르고 인터랙티브한 방법으로 에셋의 비주얼을 살펴볼 수 있습니다.

모델, 프리팹 같은 3D 에셋은 인스펙터 창 하단의 에셋 미리 보기를 클릭하고 드래그하여 오브젝트를 회전한 다음 다양한 각도에서 살펴볼 수 있습니다. 에셋을 씬에 추가하지 않은 상태로 에셋의 지오메트리, 머티리얼, 방향을 빠르게 확인하는 용도로 사용하세요.

에셋 미리 보기가 너무 작게 표시된다면 인스펙터 창에서 미리 보기 영역 위쪽의 구분선을 드래그하여 크기를 조절할 수 있습니다.

애니메이션 모델 같은 에셋은 해당 에셋에 기본 애니메이션이 있는 경우 에셋 미리 보기에 간단한 기본 애니메이션이 표시됩니다. 모션과 리깅을 빠르게 살펴보는 용도로 사용하세요.

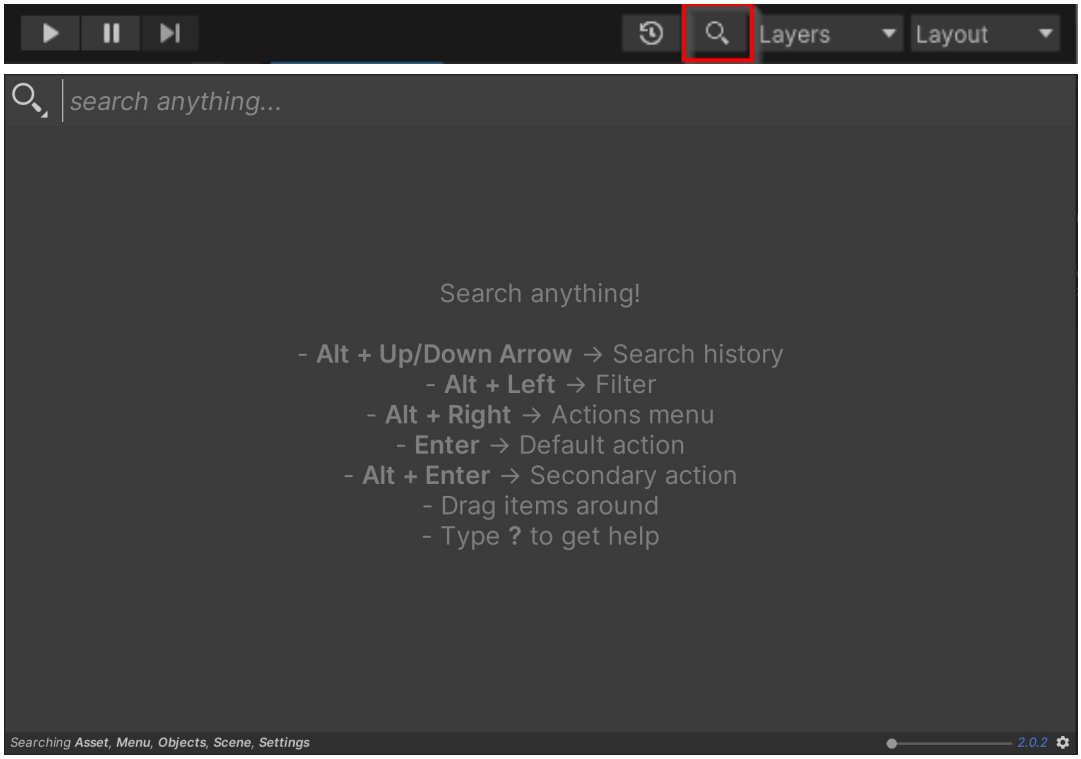
프로젝트(Project) 창에 곧바로 썸네일(에셋 미리 보기의 작은 버전)을 표시할 수도 있습니다. **프로젝트** 창을 **2열 레이아웃**으로 전환한 다음 하단의 미리 보기 슬라이더를 조절하여 썸네일을 표시하세요. 이렇게 하면 에셋을 더 쉽게 살펴보고 정리할 수 있습니다.



해피 하비스트(Happy Harvest) 샘플의 에셋 미리 보기 창

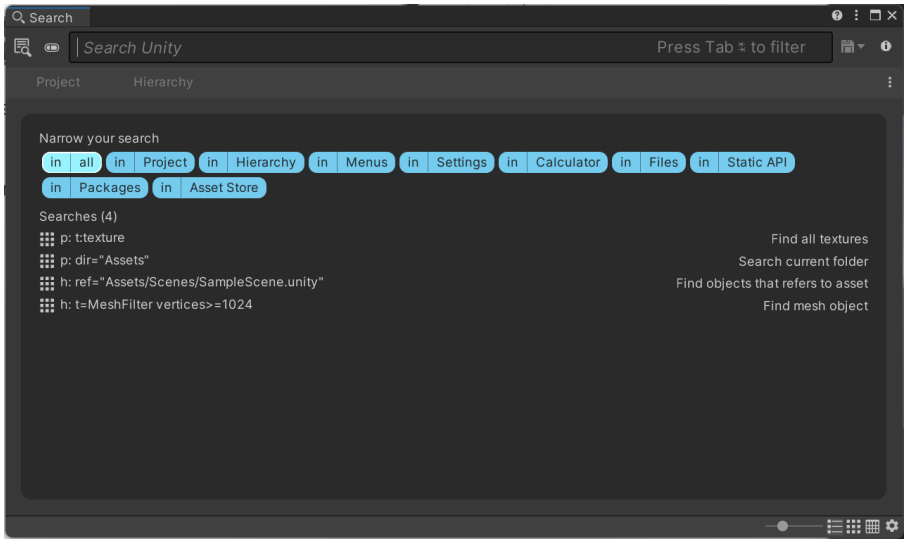
프로젝트의 검색 속도 높이기

Unity에서 에셋, 씬 오브젝트, 메뉴 항목, 패키지, API, 설정 등의 검색 효율을 높이는 방법에는 여러 가지가 있습니다. 계층 뷰와 프로젝트 뷰의 검색 기능 외에도 메인 메뉴 바의 검색 버튼 아이콘을 사용하거나 Windows의 Ctrl+K 단축키 또는 macOS의 Cmd+K 단축키를 사용하여 검색을 활성화할 수 있습니다.

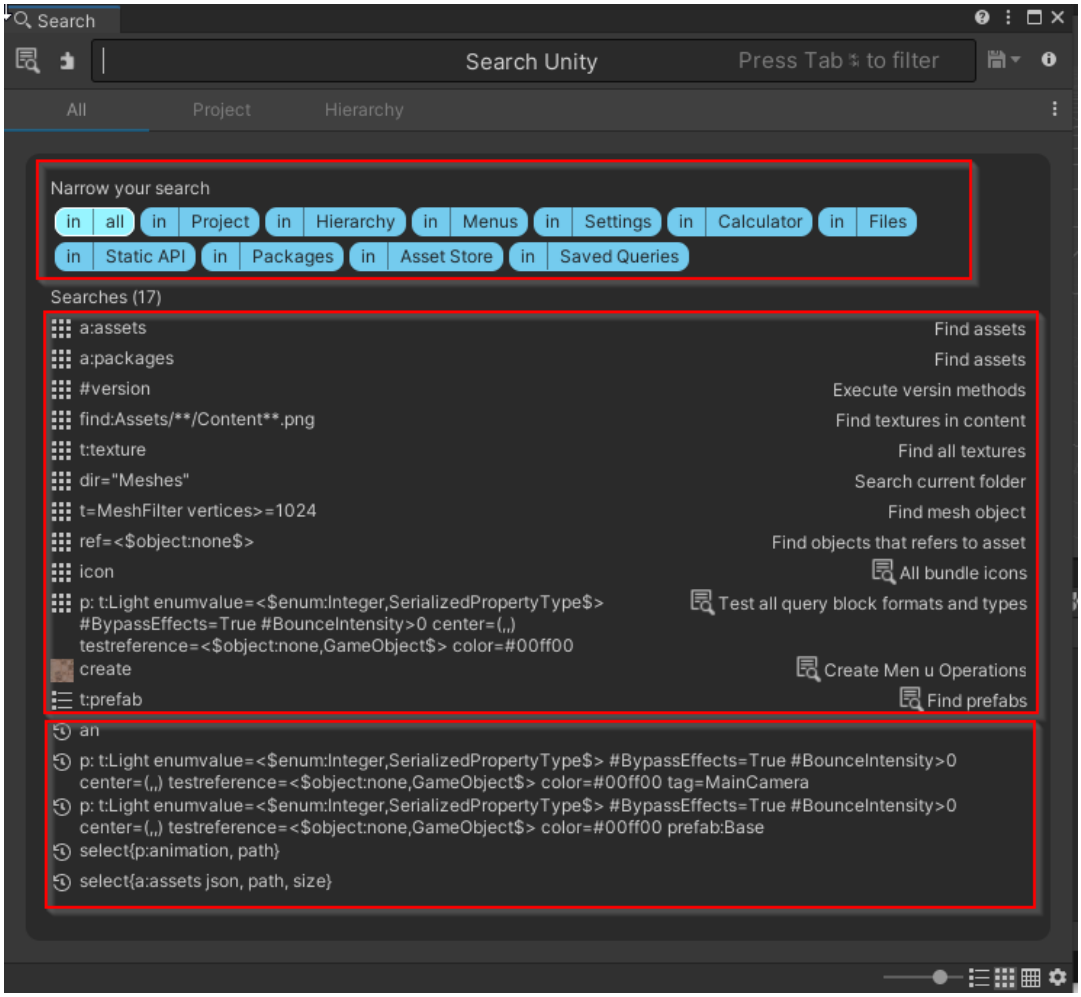


단축키 또는 도움말 메뉴를 사용하여 QuickSearch를 실행합니다.

그러면 검색 창이 열립니다. 여기에서 쉽게 검색을 필터링할 수 있습니다.



검색 창



가운데 섹션에 현재 선택한 검색 영역에서 사용할 수 있는 쿼리 목록이 표시되어 있습니다. 원하는 쿼리를 클릭하여 실행할 수 있습니다. 이름을 검색할 때는 유형을 기준으로 검색할 수도 있습니다. 드롭다운을 사용하여 **Type**을 선택하거나 **t:** 단축 구문을 입력합니다. 예를 들어 모든 씬을 검색하려면 t:scene을, 모든 텍스처를 검색하려면 t:texture를 입력합니다.

이에 더해 검색 창에서는 컴팩트 목록 뷰, 큰 목록 뷰, 서로 다른 크기의 그리드 아이콘 등 다양한 방식으로 오브젝트를 시각화할 수 있습니다. 오브젝트를 테이블에 표시할 수도 있습니다.

Unity 내부 및 외부 검색에 대한 자세한 내용은 QuickSearch 가이드를 참조하시기 바랍니다.

쿼리 빌더를 사용하여 커스텀 검색 필터 만들기

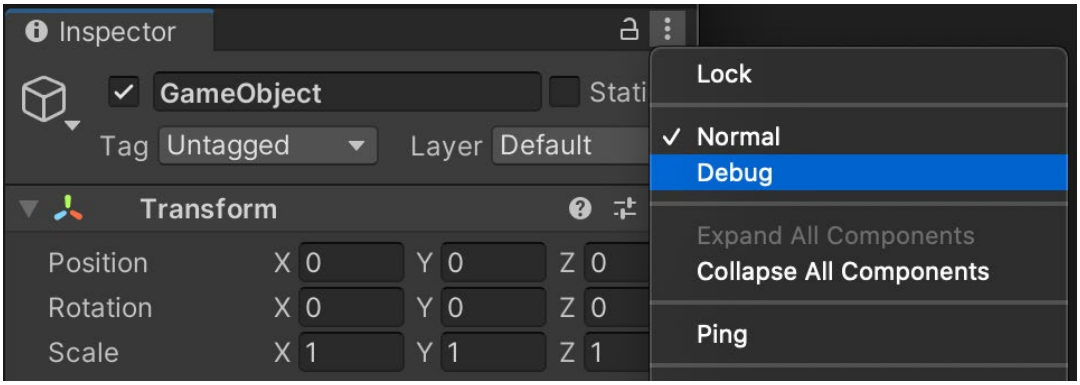
복잡한 검색 쿼리를 사용할 때 프로젝트를 탐색하는 데 도움이 필요하면 쿼리 빌더 워크플로를 사용하면 됩니다. 쿼리 빌더는 빌더 토글(검색 필드 옆의 퍼즐 버튼)로 활성화할 수 있습니다.

쿼리 빌더

일반적인 작업에 대한 커스텀 쿼리를 만들어서 저장해 두면 시간을 절약할 수 있습니다. 예를 들어 특정 태그를 기준으로 프리팹을 찾거나 사용되지 않는 에셋을 찾아야 하는 경우가 많다면 재사용 가능한 쿼리를 정의하여 이 프로세스를 자동화합니다. 쿼리 빌더는 더 이상 패키지로 제공되지 않고 에디터에 통합되어 있으므로 개발 팀의 워크플로를 간소화할 수 있습니다.

인스펙터에서 디버그 데이터 노출

각 게임 오브젝트의 인스펙터를 일반 모드와 디버그 모드로 바꿀 수 있습니다. **더 보기**( ) 버튼을 클릭하여 컨텍스트 메뉴를 열고 원하는 모드를 선택하면 됩니다. 디버그 모드를 사용할 때는 프라이빗 필드, 숨겨진 레퍼런스, 내부 Unity 데이터를 비롯한 모든 직렬화된 필드가 인스펙터에 노출됩니다. 이는 표준 퍼블릭 인터페이스를 넘어서서 컴포넌트를 살펴보거나 문제를 해결하는 데 유용합니다.

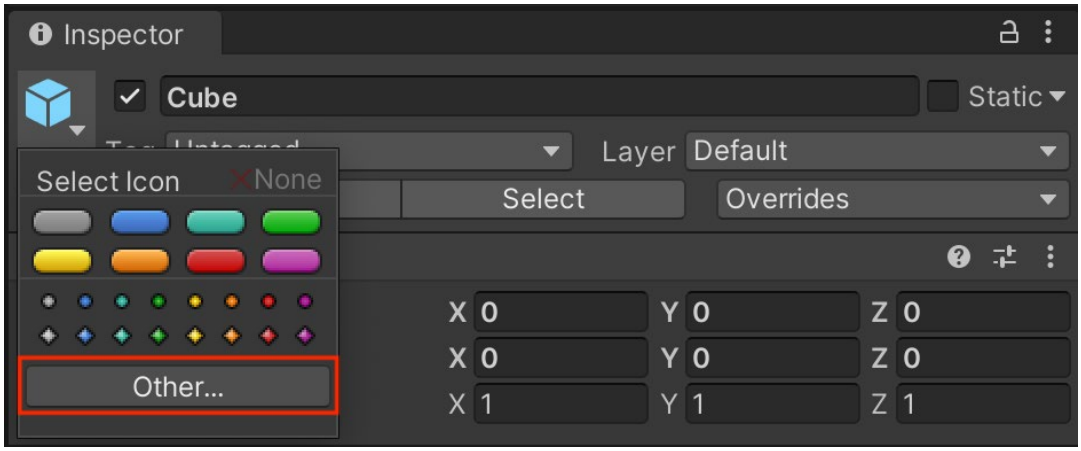


인스펙터 디버그 모드

비주얼 디버깅에 커스텀 기즈모와 아이콘 사용

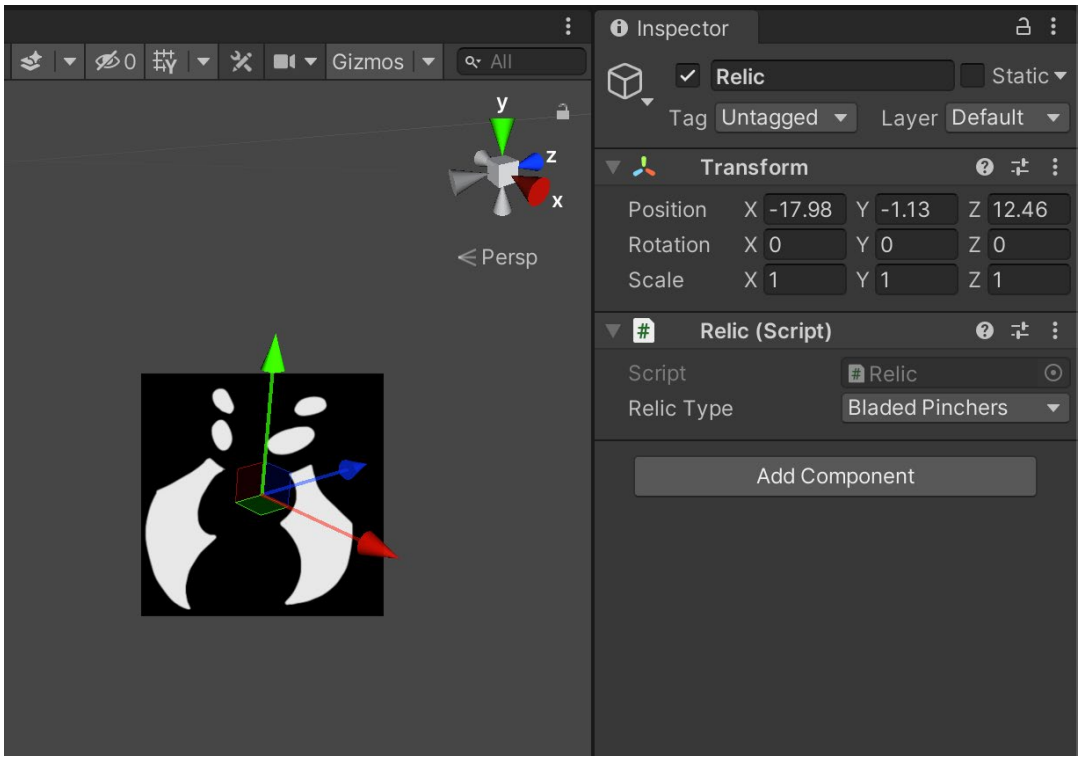
기즈모는 개발 중에 게임 오브젝트를 찾고 식별하는 데 도움이 되도록 씬 뷰에 표시되는 비주얼 오버레이입니다. 트리거, 생성 지점, 스크립트 같은 렌더링되지 않는 요소를 시각화하는 데 특히 유용합니다.

Select Icon 메뉴를 사용하여 게임 오브젝트의 아이콘을 원하는 대로 수정할 수 있습니다. 나만의 아이콘을 정의하려면 **Other**를 선택합니다.



인스펙터에서 드롭다운을 사용하여 기즈모를 전환합니다.

스크립트를 사용하면 인터랙티브 기즈모를 만들 수도 있습니다. 일례로 기즈모를 통해 커스텀 컴포넌트의 영향 범위 또는 영역을 정의할 수 있습니다.



이 예시에서는 사용자의 선택에 따라 스크립트가 기즈모를 변경합니다.

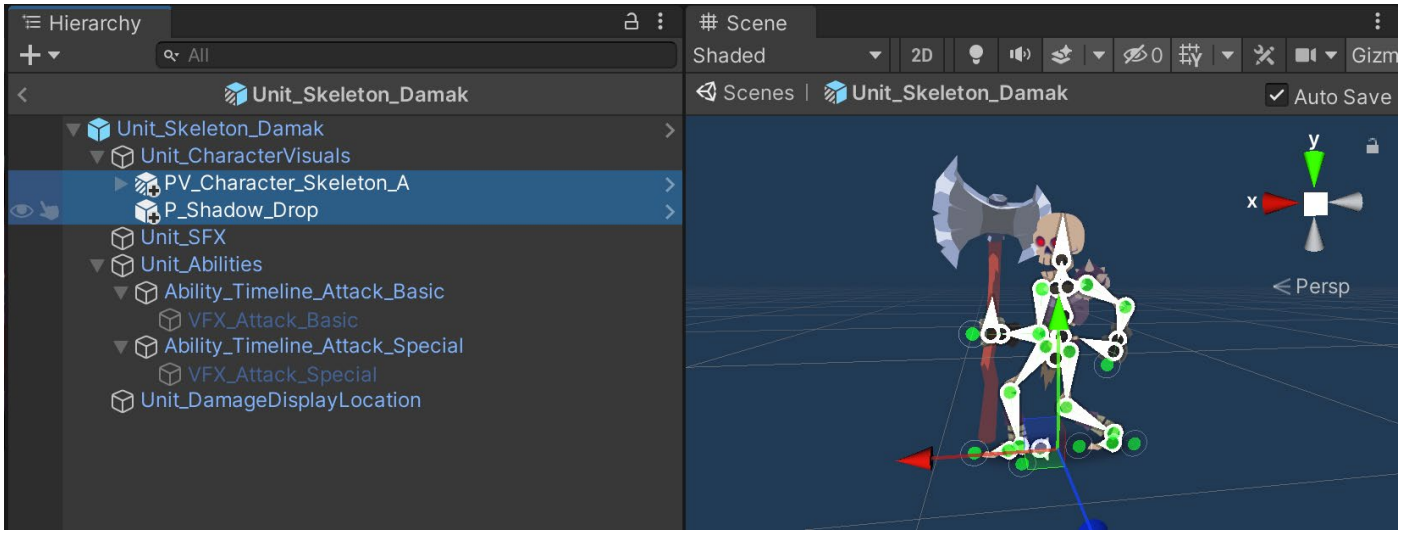
씬 컨트롤 바에서 **Gizmos** 다이얼로그를 사용하여 특정 기즈모를 켜고 끄거나, 모든 기즈모를 전체적으로 활성화/비활성화할 수 있습니다.

사용 예시는 개발용 커스텀 기즈모 만들기를 참조하시기 바랍니다. 기즈모 및 핸들에 대한 API도 살펴보시기 바랍니다.

네스티드 프리팹 및 프리팹 배리언트 워크플로 사용

프리팹을 사용하면 완전히 구성된 게임 오브젝트를 저장 및 재사용하는 동시에 씬을 유연하고 효율적으로 제작할 수 있습니다. 하지만 프리팹에는 이외에도 다양한 용도가 있습니다. 프리팹을 중첩하거나, 프리팹 배리언트를 만들거나, 프리팹을 사용하여 런타임에 게임 오브젝트를 인스턴스화할 수 있습니다.

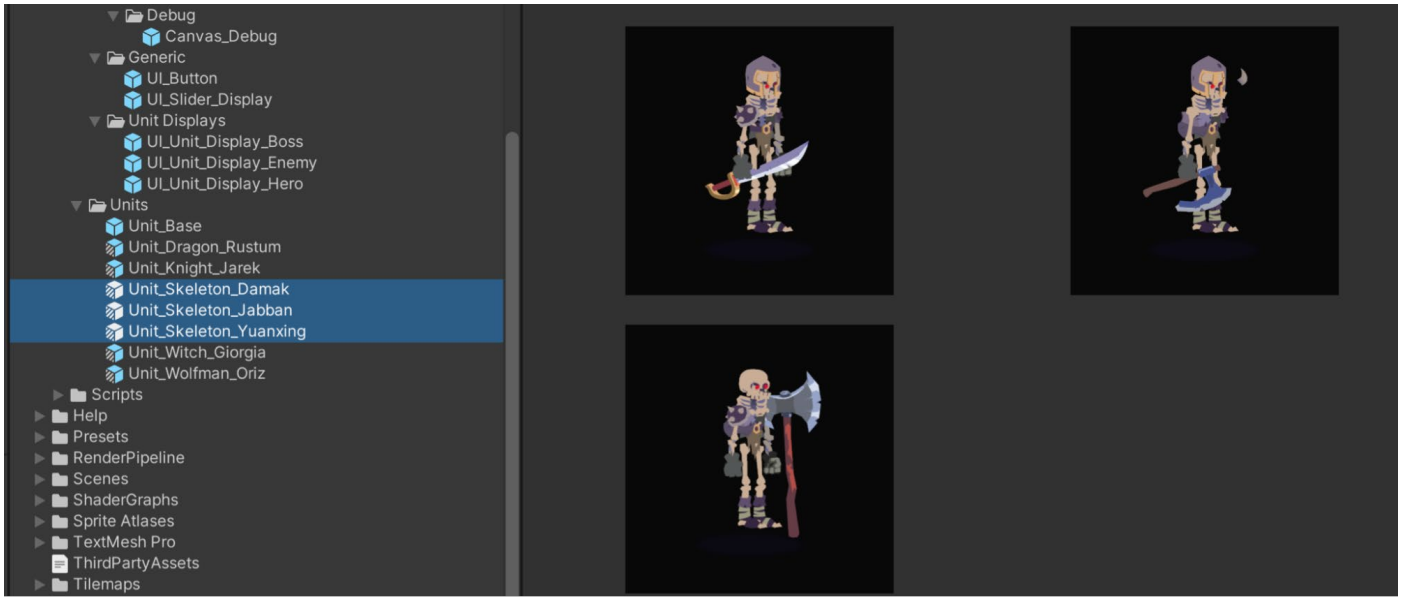
네스티드 프리팹을 사용하면 프리팹 간 부모 지정이 가능하며, 소형 프리팹(예: 방, 가구)으로 구성된 대형 프리팹(예: 건물)을 만들 수 있습니다. 따라서 여러 아티스트와 개발자로 구성된 팀에서 에셋 개발을 효율적으로 분담하여 콘텐츠의 다양한 부분을 동시에 작업할 수 있는 것입니다.



드래곤 크래셔(Dragon Crashers) 샘플 프로젝트의 네스티드 프리팹 예시

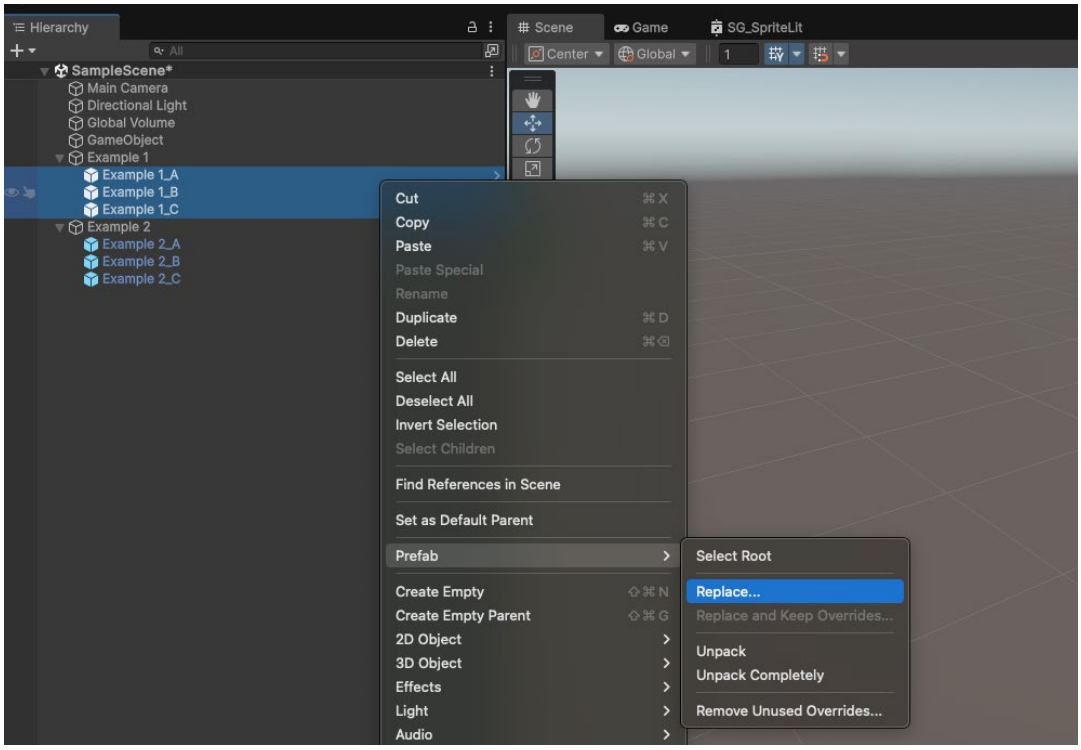
프리팹 배리언트를 사용하면 객체 지향 프로그래밍의 상속과 유사한 방식으로 프리팹에서 다른 프리팹을 파생할 수 있습니다. 배리언트를 변경하려면 특정 부분을 오버라이드하면 됩니다. 이때 원본 프리팹은 영향받지 않습니다. 또한 언제든지 모든 변경 사항을 삭제하고 기본 프리팹으로 되돌릴 수 있습니다.

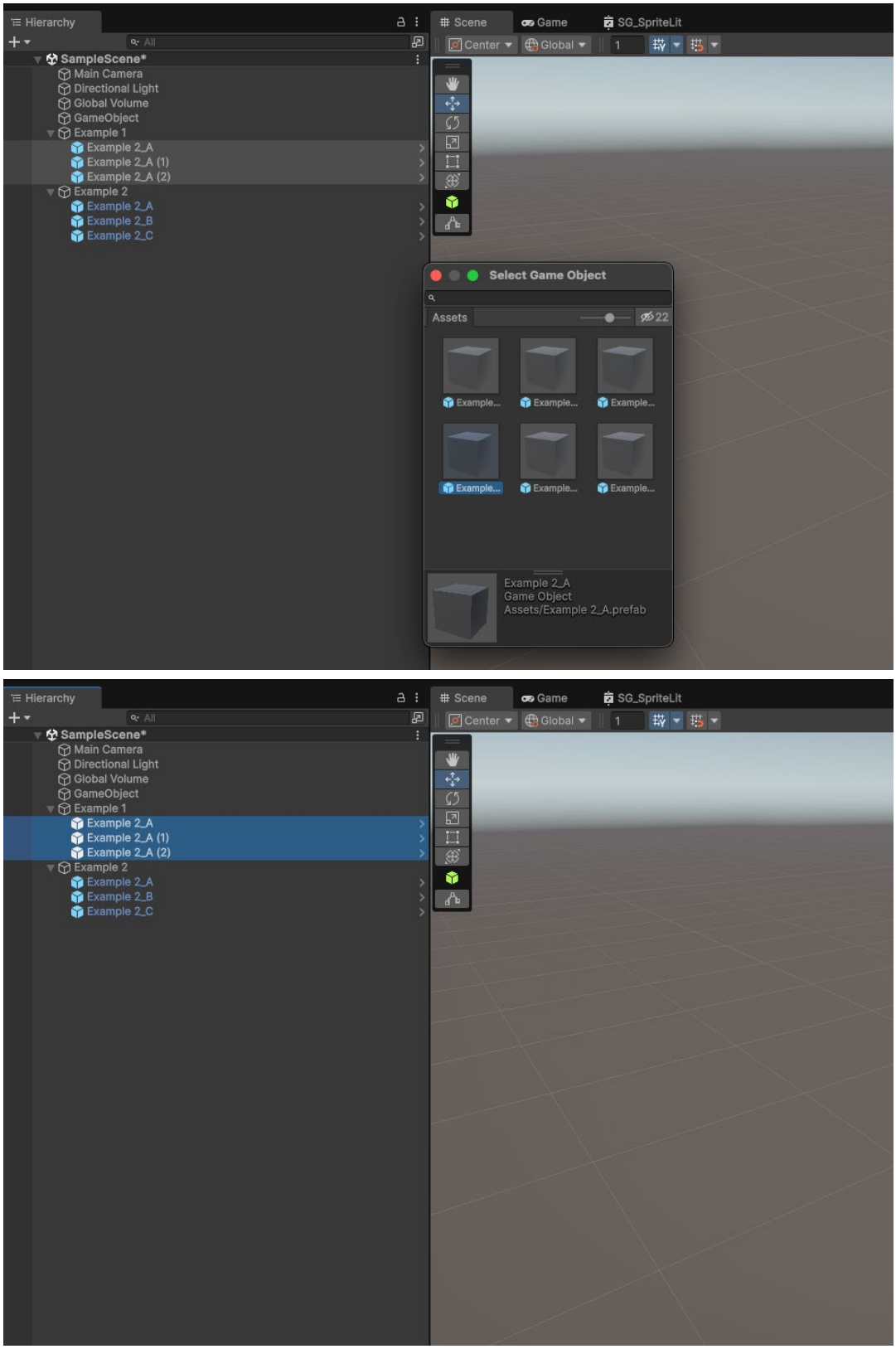
모든 배리언트를 한 번에 변경하려면 기본 프리팹에 직접 변경 사항을 적용하면 됩니다.



드래곤 크래셔 샘플 프로젝트의 프리팹 배리언트

씬에서 하나 이상의 프리팹을 다른 프리팹으로 대체하려면 Ctrl 키(Windows) 또는 Cmd 키(macOS)를 누른 채로 새 프리팹을 기존 프리팹 위로 드래그합니다.

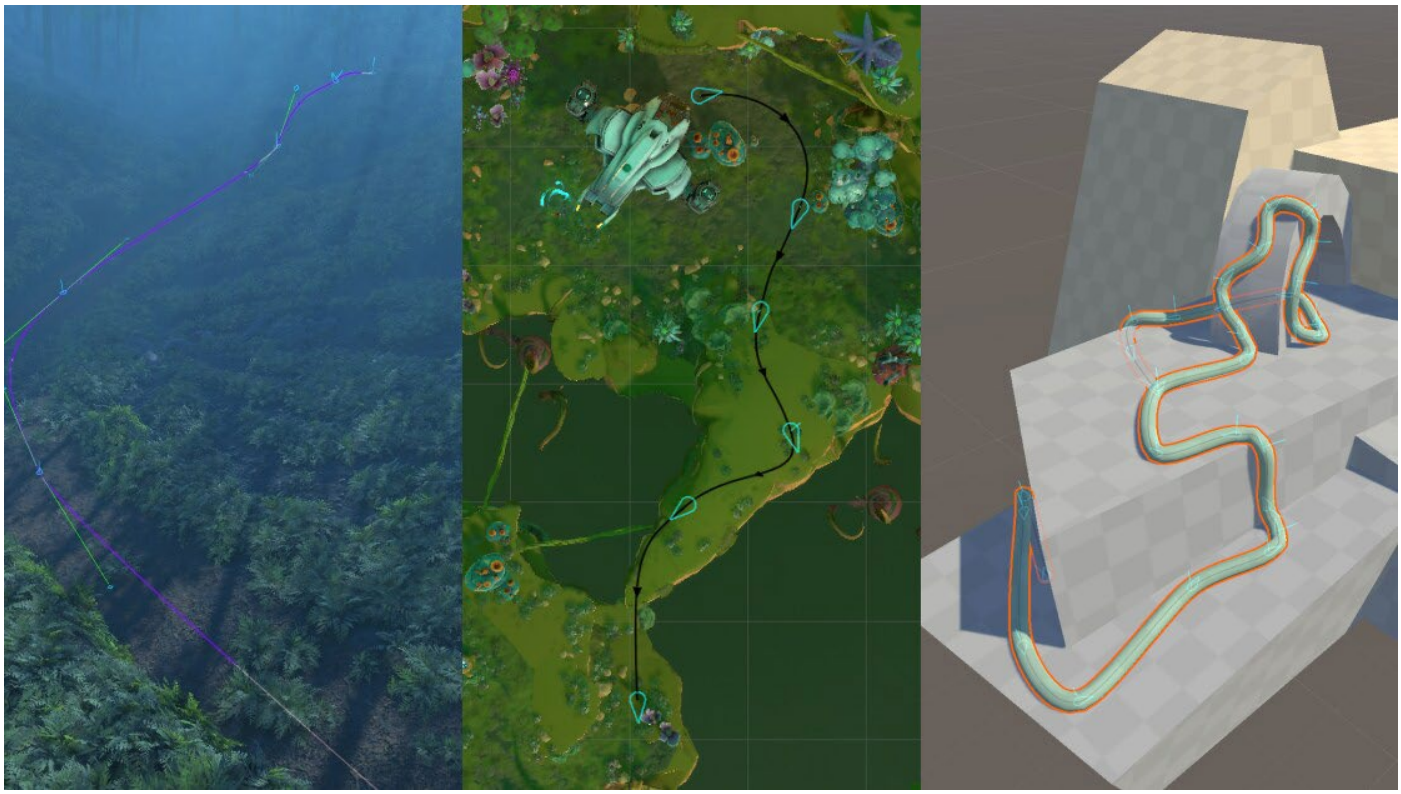



Hierarchy
All
SampleScene*
Main Camera
Directional Light
Global Volume
GameObject
Example 1
Example 2_A
Example 2_A (1)
Example 2_A (2)
Example 2
Example 2_A
Example 2_B
Example 2_C
Scene
Game
SG_SpriteLit
Center
Global
Select Game Object
Assets
22
Example...
Example 2_A
Game Object
Assets/Example 2_A.prefab

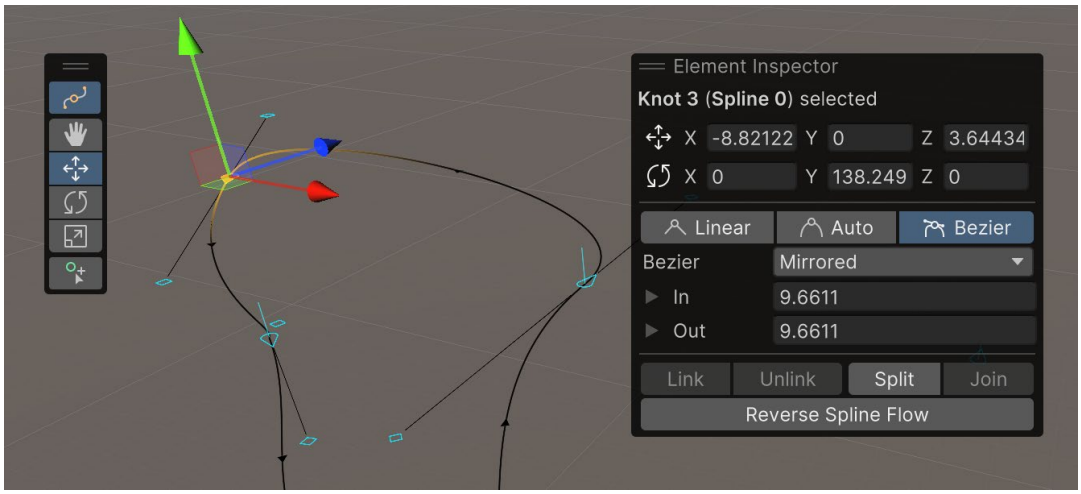
계층 창에서 프리팹 대체

Splines 패키지를 사용하여 손쉽게 곡선 경로 만들기

Splines 패키지를 사용하면 에디터에서 바로 스플라인 곡선 기반 경로를 만들고 편집할 수 있습니다. 스플라인은 도로, 하천, 철도, 카메라 트랙이나 기타 경로 기반 비주얼 또는 게임플레이 요소를 구성하는 부드러운 곡선 경로를 정의하는 데 유용합니다.



스플라인 사용 사례(왼쪽부터): 숲을 통과하는 도로, 애니메이션 경로, 튜브 메시. Splines 패키지 관리자 페이지에서 패키지 설치 후 구현 가능한 모든 사례를 살펴볼 수 있습니다.



Unity의 스플라인 핸들 및 컨트롤은 잘 알려진 DCC 애플리케이션의 벡터 또는 3D 드로잉 툴과 유사합니다.

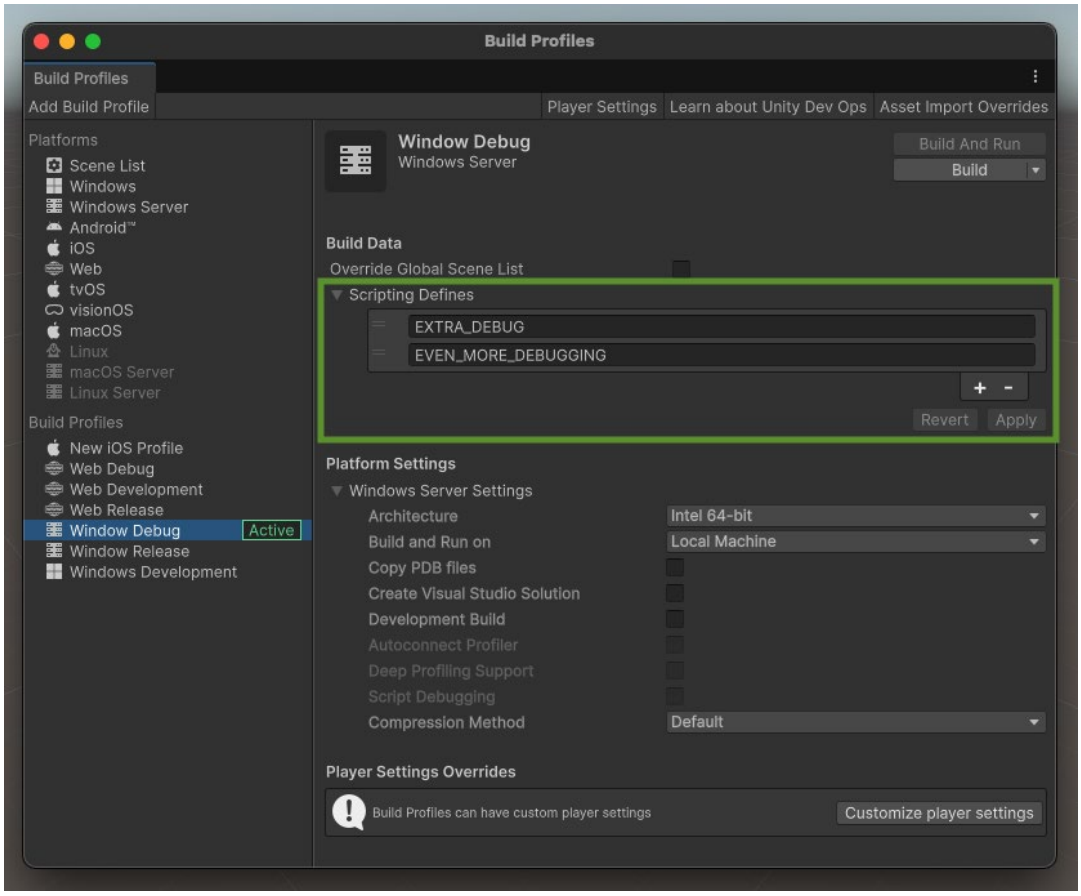
스플라인을 만들면 프로그래머와 아티스트 모두 흥미로운 방식으로 스플라인을 사용할 수 있습니다. 예를 들어 프로그래머는 스플라인의 포인트를 읽고 API를 통해 게임 로직에 사용할 수 있습니다.

스플라인 패키지를 시작하는 방법(영어) 동영상 튜토리얼에서 Splines 패키지를 사용하여 일반적인 스플라인 사용 사례를 살펴보고, 스플라인을 따라 게임 오브젝트의 위치와 회전을 애니메이션화하고, 스플라인을 따라 프리팹을 인스턴스화하여 환경을 만드는 방법을 알아보세요.

빌드 프로파일을 사용하여 빌드 커스터마이즈

빌드 프로파일 워크플로는 Unity 6에서 빌드를 구성하는 새로운 방법입니다. 이 새로운 워크플로에서는 플랫폼별로 서로 다른 설정을 갖는 다양한 구성을 활용할 수 있습니다.

Unity 6에서 커스텀 빌드 프로파일을 만들면 특정 플랫폼 또는 용도(디버깅, 테스트, 정식 제작 등)를 위해 서로 다른 빌드 설정을 효율적으로 구성할 수 있습니다. 이렇게 하면 여러 빌드 구성 간에 빠르게 전환하여 워크플로를 최적화할 수 있게 됩니다.



Build Profiles 창에서 Scripting Defines 옵션과 Player Settings Overrides 옵션을 사용하여 빌드 전용 구성(예: 디버그 툴 활성화, 스플래시 이미지 커스터마이징 등)을 설정하고 버전 관리를 통해 팀원들에게 손쉽게 공유하세요.

더 많은 리소스:

— Unity 6의 빌드 프로파일에 대해 알아야 할 사항

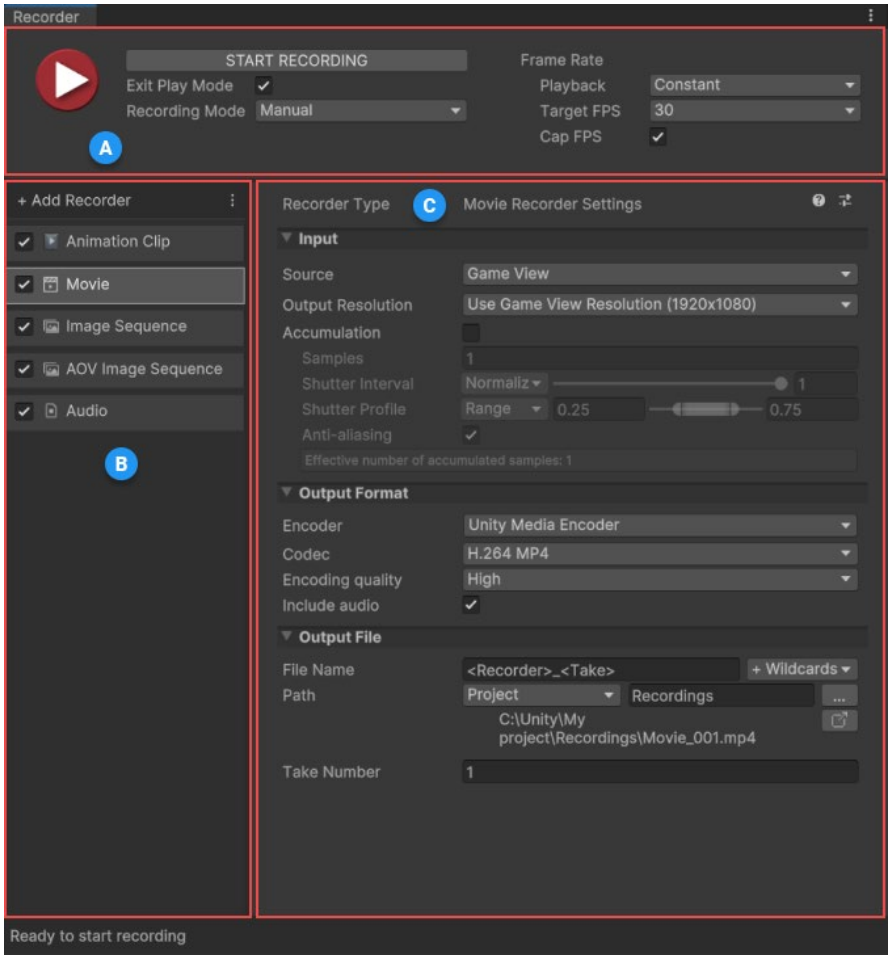
— Unity 6의 빌드 프로파일에 대해 알아야 할 모든 것 | 유나이트 2024(영어)

Recorder 패키지를 사용하여 부드러운 게임 영상 캡처

Unity Recorder 패키지를 사용하면 플레이 모드 중에 프로젝트에서 데이터를 캡처하여 익스포트할 수 있습니다. 이는 게임플레이, 시네마틱 또는 기타 런타임 시퀀스를 동영상 파일, 이미지 시퀀스, 애니메이션 데이터로 레코딩하는 데 유용합니다.

Recorder 창은 Unity 에디터 메인 메뉴의 **Window**에 있습니다. **Recorder** 창을 열면 마지막 레코딩 세션의 값이 복원되어 빠르게 다시 시작하거나 조정할 수 있습니다.

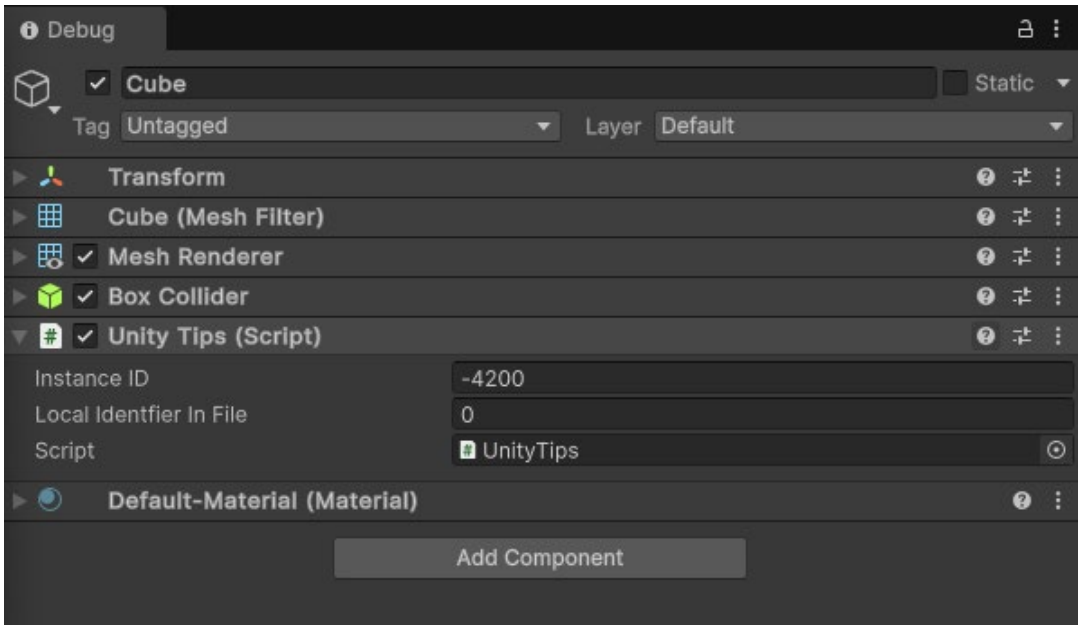
이미지 또는 동영상을 투명 배경으로 캡처할 수 있습니다. Unity Recorder는 특정 조건이 충족되면 알파 채널을 캡처하여 투명도를 출력할 수 있습니다. 해당 조건은 사용 중인 렌더 파이프라인과 Recorder 설정에 좌우됩니다.



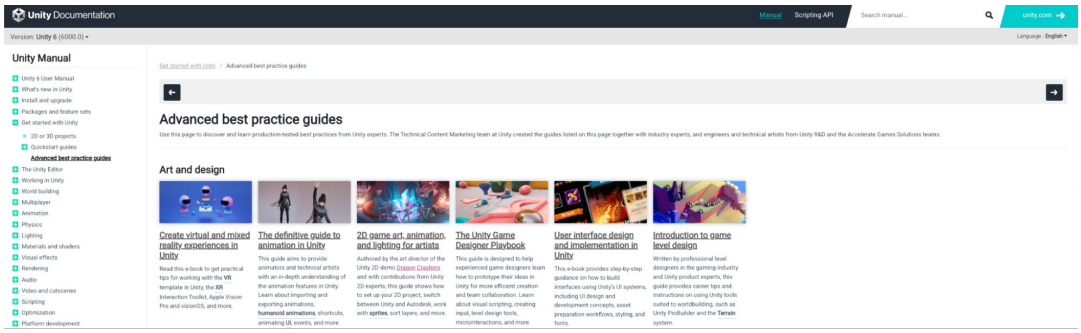
Unity Recorder 창

에디터 워크플로 팁

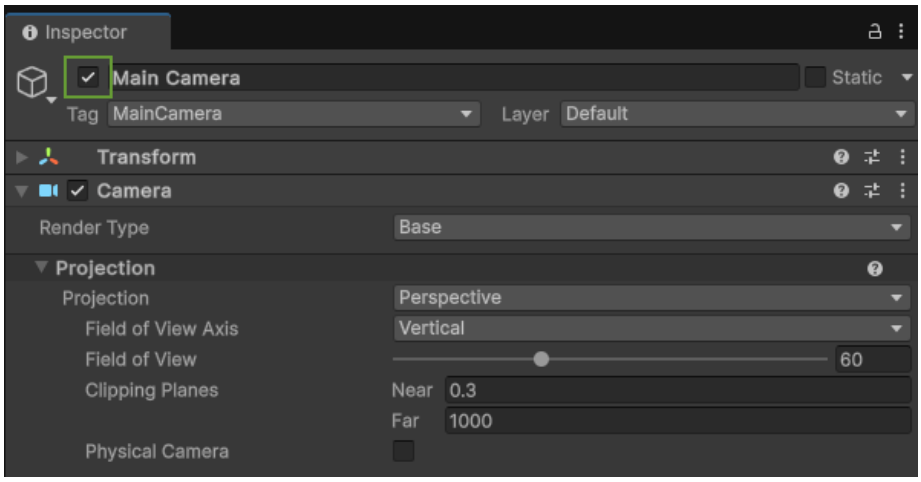
1. **[HelpURL]** 속성을 사용하여 **인스펙터**의 물음표 아이콘을 기술 자료나 기타 리소스에 연결하세요. 인스펙터에서 클릭하면 지정된 URL이 사용자의 기본 브라우저로 열립니다.




```
using UnityEngine;

[HelpURL ("https://docs.unity3d.com/Manual/best-practice-guides.html")]
public class UnityTips : MonoBehaviour
{    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {

    }

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

    }
}

```

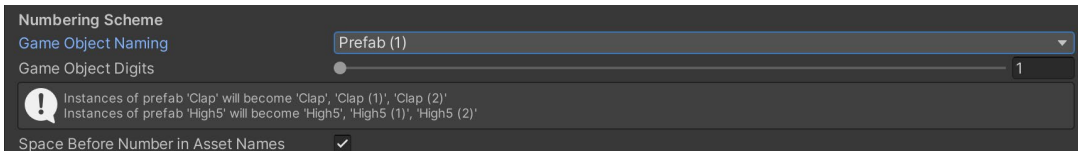


2. 마우스 클릭 수를 줄여 보세요. **Shift+Alt+A**(Windows) 또는 **Shift+Option+A**(macOS)를 길게 눌러 현재 선택된 게임 오브젝트를 활성화 또는 비활성화할 수 있습니다.



선택된 게임 오브젝트를 활성화 또는 비활성화합니다.

3. Unity에서는 게임 오브젝트의 번호 매기기 방식을 커스터마이즈할 수 있습니다. 에디터 탭의 **Project Settings**에서 Numbering Scheme을 찾으세요. 여기에서 명명 규칙의 옵션과 인스턴스 번호의 패딩(padding) 및 간격을 정의할 수 있습니다.



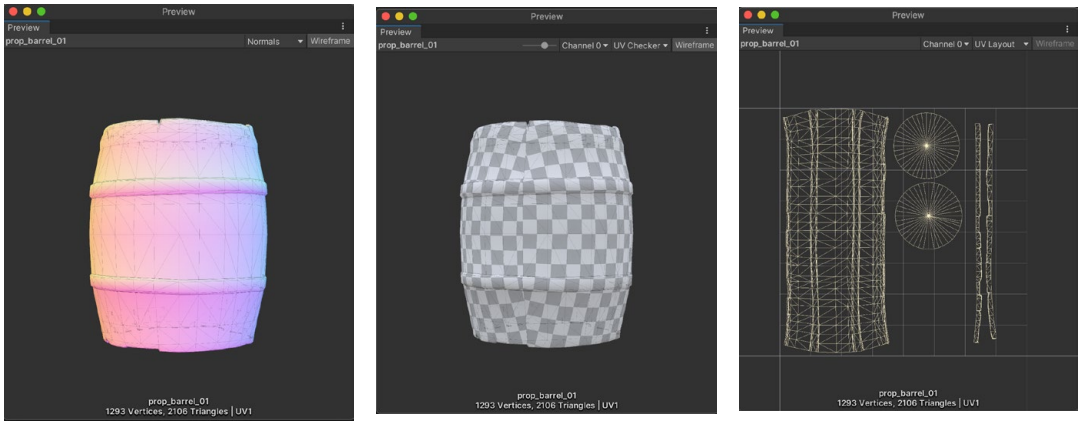
번호 매기기 방식

4. 계층 창에서 게임 오브젝트를 잘라내고 붙여 넣을 수 있습니다. 컨텍스트 메뉴에서 **Paste As Child**를 활용할 수도 있습니다.

Paste As Child 메뉴 항목

5. **F** 단축키를 사용하면 씬 뷰에서 선택한 오브젝트를 프레이밍할 수 있습니다. 플레이 모드에서 **Shift+F**를 누르면 현재 선택된 움직이는 게임 오브젝트를 잠글 수 있습니다.

6. 인스펙터 미리 보기에는 UV, 노멀, 탄젠트뿐만 아니라 기타 메시 정보가 표시됩니다.

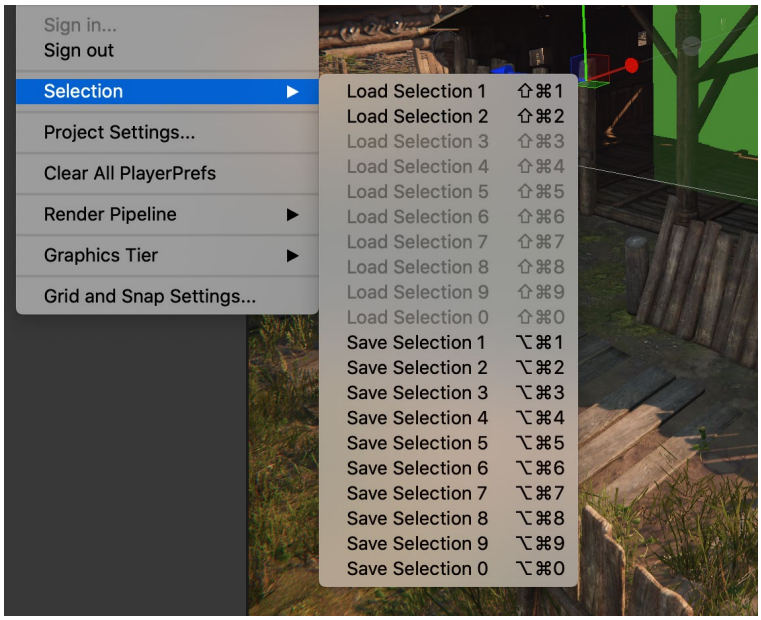


인스펙터 미리 보기

7. Layers 메뉴를 사용하여 씬 뷰를 가리는 레이어(UI 등)를 보이지 않게 할 수 있습니다. 이때 레이어를 잠그면 실수로 레이어의 상태를 변경하는 상황을 방지할 수 있습니다.

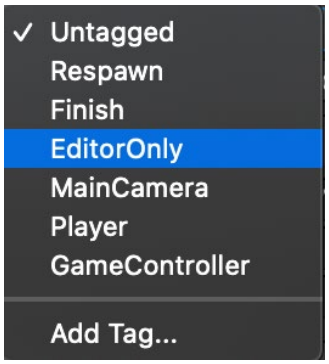
레이어 토글 및 편집

8. 씬에서 같은 오브젝트를 자주 선택한다면 **Edit** > **Selection**에 있는 단축키 조합을 사용하여 선택 사항을 빠르게 저장하거나 로드할 수 있습니다.



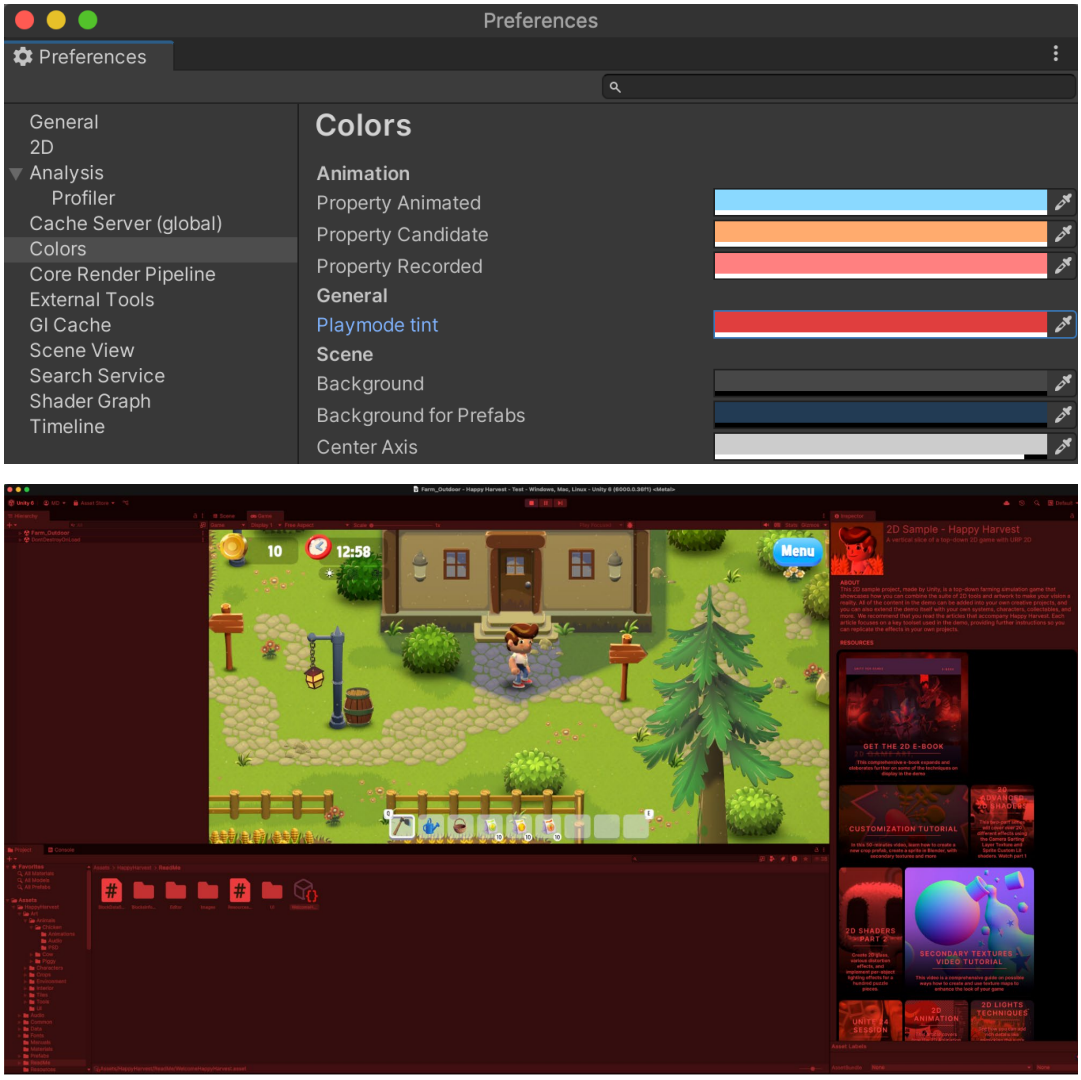
선택 사항 로드 및 저장

9. **EditorOnly** 태그를 사용하면 애플리케이션 빌드에 표시되지 않을 게임 오브젝트를 지정할 수 있습니다.



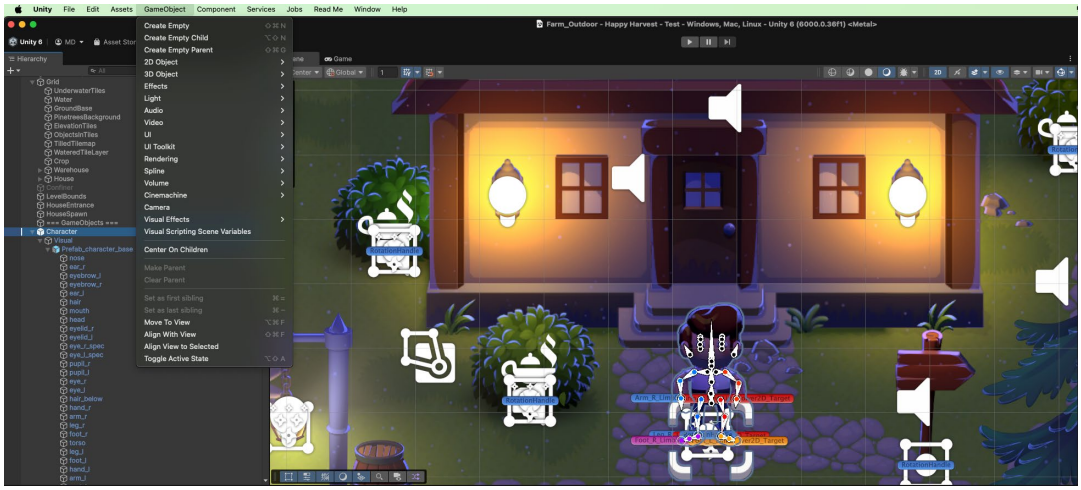
EditorOnly 태그

10. **Unity > Preferences > Colors**에서 에디터의 컬러를 변경하여 특정 UI 요소나 오브젝트를 더 빠르게 찾을 수 있습니다. 플레이 모드 종료 시 저장하려는 변경 사항이 유실되지 않도록 플레이 모드가 활성화 중임을 표시하는 플레이 모드 틴트를 조정할 수 있습니다.



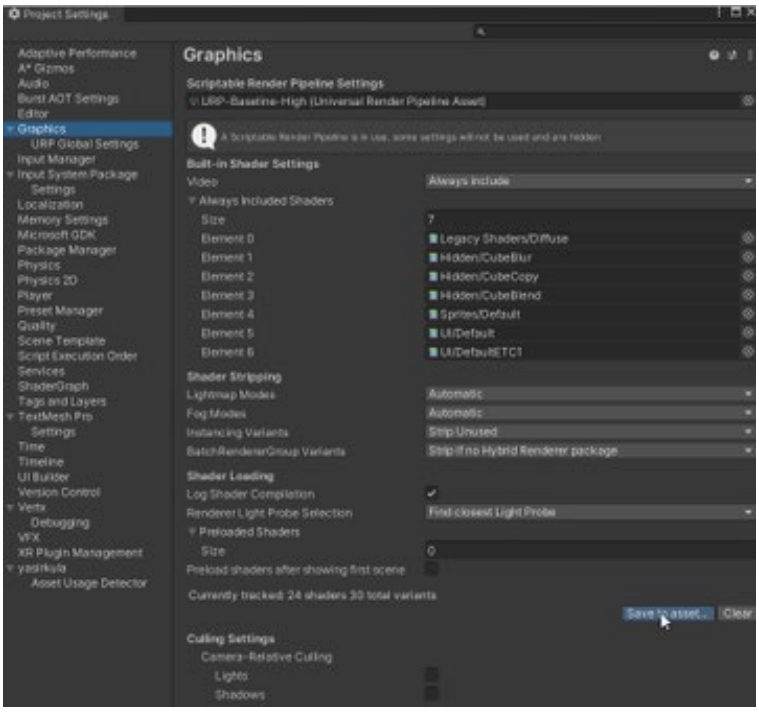
플레이 모드 틴트

11. 카메라를 설정할 때 **GameObject > Align With View**를 사용하면 게임 카메라를 씬 카메라와 정렬할 수 있습니다. 씬 카메라를 게임 카메라와 정렬하려는 경우, **Align View to Selected**를 사용하면 씬 카메라를 계층 구조의 다른 카메라와 정렬할 수 있습니다.



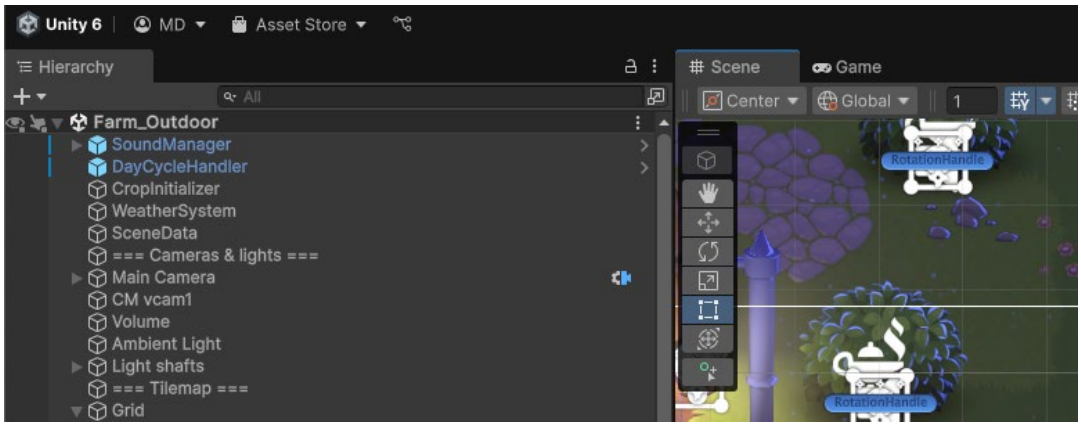
Align With View 옵션

12. 에디터가 씬에서 사용하는 셰이더 배리언트 목록을 생성할 수 있습니다. **Edit** > **Project Settings** > **Graphics**로 이동합니다. 'Shader Loading' 아래에 몇 개의 셰이더 및 배리언트가 있는지 봅니다. **Save to asset...**을 선택하여 셰이더 배리언트 컬렉션 에셋을 만듭니다.



Graphics 창

13. 아무 탭(**Project**, **Scene**, **Game** 등)을 더블 클릭하여 에디터에 전체 화면으로 표시합니다.



씬 탭 더블 클릭

씬 탭을 더블 클릭한 후

더 많은 자료

더 많은 생산성 팁을 보려면 다음 자료를 참고하세요.

- 키보드 단축키로 Unity의 워크플로 속도 높이기(영어)
- 확장성 있는 아트 에셋 - 워크플로 개선을 위한 6가지 팁(영어)
- Unity 2022 LTS 스크립팅을 위한 최고의 팁(영어)

2D

스프라이트

2D 프로젝트는 스프라이트를 사용하여 비주얼을 제작합니다. 스프라이트는 여러 텍스처 에셋이 포함되어 있기 때문에 배칭이 올바르게 이루어지지 않으면 드로우 콜이 많이 발생할 수 있습니다.

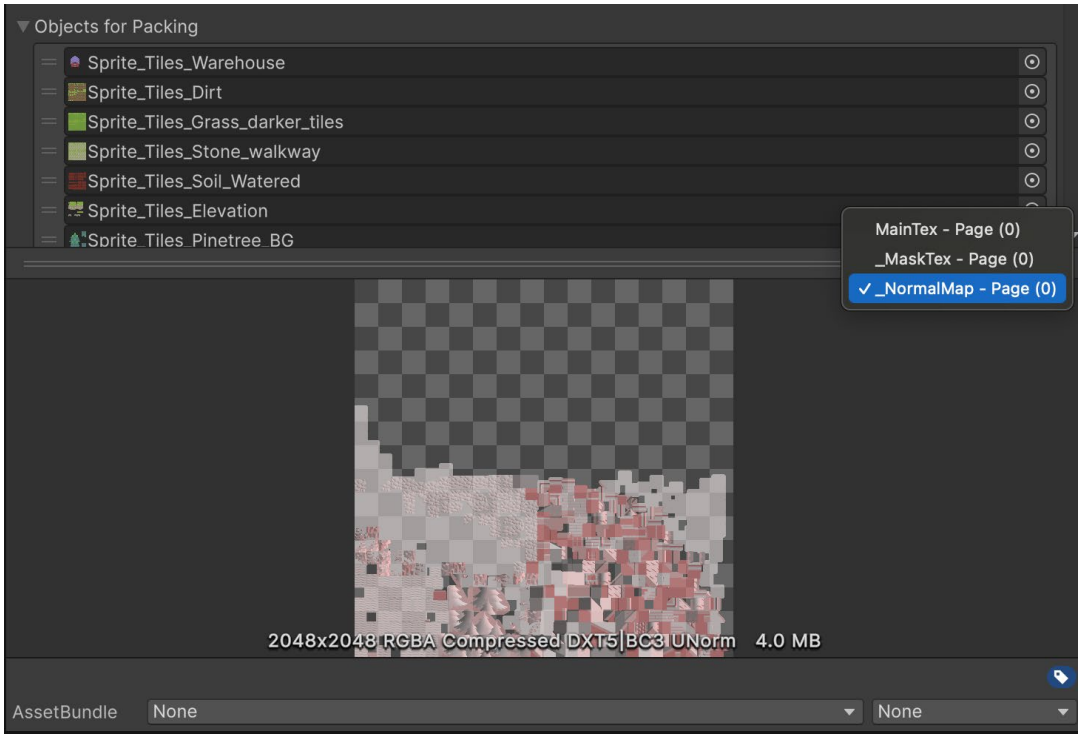
리소스를 최적화하려면 동일한 머티리얼과 패킹된 텍스처를 사용하여 스프라이트를 배칭하는 **스프라이트 아틀라스**(**Asset** > **Create** > **Sprite Atlas**)를 사용하는 것이 좋습니다.

스프라이트를 패킹할 때 중복된 에셋 방지

2D URP에서는 스프라이트가 노멀 맵 또는 마스크 맵을 포함할 수 있으며, 이를 스프라이트 아틀라스에 함께 패킹해야 할 수 있습니다. 메모리 최적화와 드로우 콜 감소, 로딩 효율 같은 성능 및 품질 측면에서 중요한 부분입니다.

스프라이트 아틀라스에 폴더를 추가할 경우, 해당 폴더에 노멀 스프라이트 또는 알베도 텍스처만 포함해야 합니다. 노멀 맵과 마스크 맵은 드롭다운 목록에서 확인할 수 있는 각각의 아틀라스에 자동으로 추가됩니다. 그렇지 않으면 노멀 팩 아틀라스에 불필요하게 추가되어 공간을 차지하게 됩니다.

노멀 맵 또는 마스크 맵의 아틀라스를 깔끔하게 분리해 두면 각 아틀라스의 압축 설정을 세밀하게 조정할 수 있습니다. 예를 들어 노멀 맵은 해상도를 절반으로 낮추고 노멀 스프라이트는 전체 해상도를 유지하는 등 개발 대상 플랫폼에 적합하게 조정할 수 있습니다.



2D 스프라이트 패킹에 사용되는 세 가지 아틀라스 유형

일관된 폴더 구조

2D 스프라이트를 폴더 구조로 만들면 에셋을 쉽게 찾아서 관리하고, 로드 시간을 기준으로 에셋을 정리하여 메모리를 관리하고, 여러 게임 요소를 명확히 구별하는 데 도움이 됩니다.

다음은 Unity 프로젝트에서 폴더를 정리할 때 유용하게 사용할 수 있는 폴더 구조입니다.

- Sprites/Characters/[CharacterName]/Animations
- Sprites/UI/[MenuType]/Elements
- Sprites/Environment/[LevelName]/Background
- Sprites/Effects/[EffectType]
- Sprites/Common/SharedElements

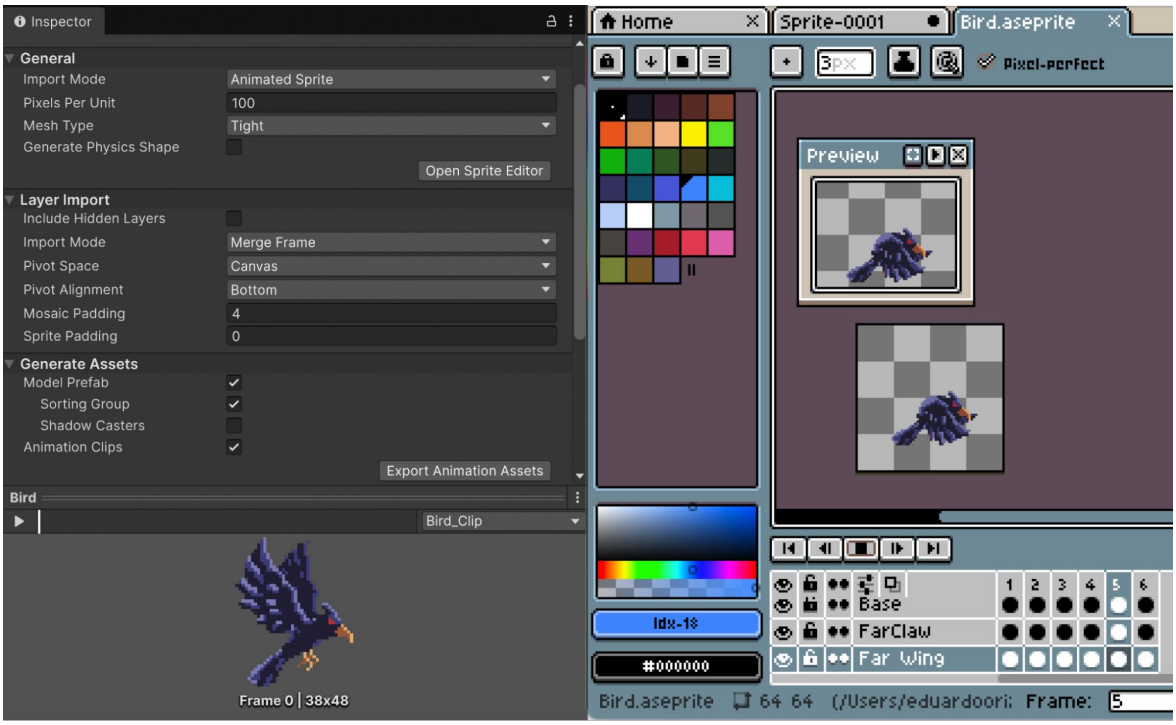
Aseprite Importer를 사용하여 Unity로 픽셀 아트 가져오기

스프라이트 에디터이자 픽셀 아트 툴인 Aseprite는 픽셀 아티스트를 위한 업계 표준 DCC 앱으로 자리 잡았습니다.

Unity Aseprite Importer를 사용하면 더 이상 수백 개의 .PNG 스프라이트를 개별적으로 익스포트할 필요 없이 **.aseprite** 파일을 프로젝트에 바로 추가할 수 있습니다.

Aseprite에서 작업하고 **Save**를 클릭하면 즉시 Unity에서 변경 사항을 확인할 수 있습니다. Aseprite Importer는 .aseprite 파일의 드래그 앤 드롭을 지원하고, 게임 오브젝트 및 애니메이션 클립을 곧바로 사용할 수 있도록 자동 설정을 포함하며, 방대한 2D 기능 세트가 통합되어 있습니다.

Unity 6에서는 타일맵을 포함하는 **.aseprite** 파일에서 자동으로 타일맵 에셋을 생성할 수 있습니다. 소스에 적용된 변경 사항은 하위 수준으로 자동 전파됩니다.

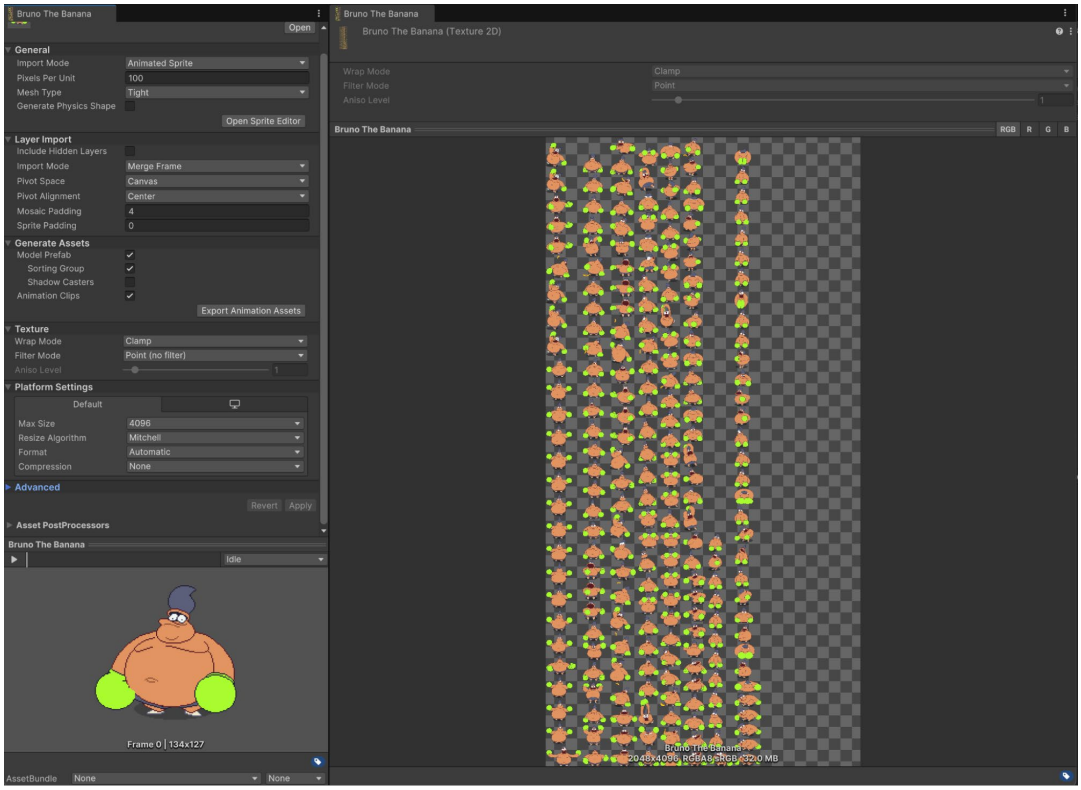


Unity 프로젝트 창의 Aseprite Importer: 유나이트 2024 발표 '새로운 2D 워크플로 및 AI 개선 소개(영어)'의 이 섹션에서 Aseprite Importer를 실제로 사용하는 모습을 볼 수 있습니다.

Aseprite에서 Unity로 프레임 바이 프레임 애니메이션 가져오기

Unity에서 Aseprite Importer를 사용할 때 유용한 팁 중 하나는 바로 임포트에 앞서 **태그**와 **슬라이스**를 사용하여 Aseprite 파일을 체계적으로 정리해 두는 것입니다. 그러면 에디터에서 애니메이션 및 스프라이트 슬라이싱이 자동으로 매끄럽게 처리됩니다.

예를 들어 'Idle', 'Walk' 등의 태그를 사용하여 애니메이션을 정의 및 정리하고, 슬라이스를 사용하여 스프라이트의 특정 영역이나 머리, 무기 같은 개별 파트를 표시하세요. 이렇게 하면 임포트 프로세스를 간소화하고 시간을 절약할 수 있습니다.



Aseprite에서 Unity로 임포트된 애니메이션

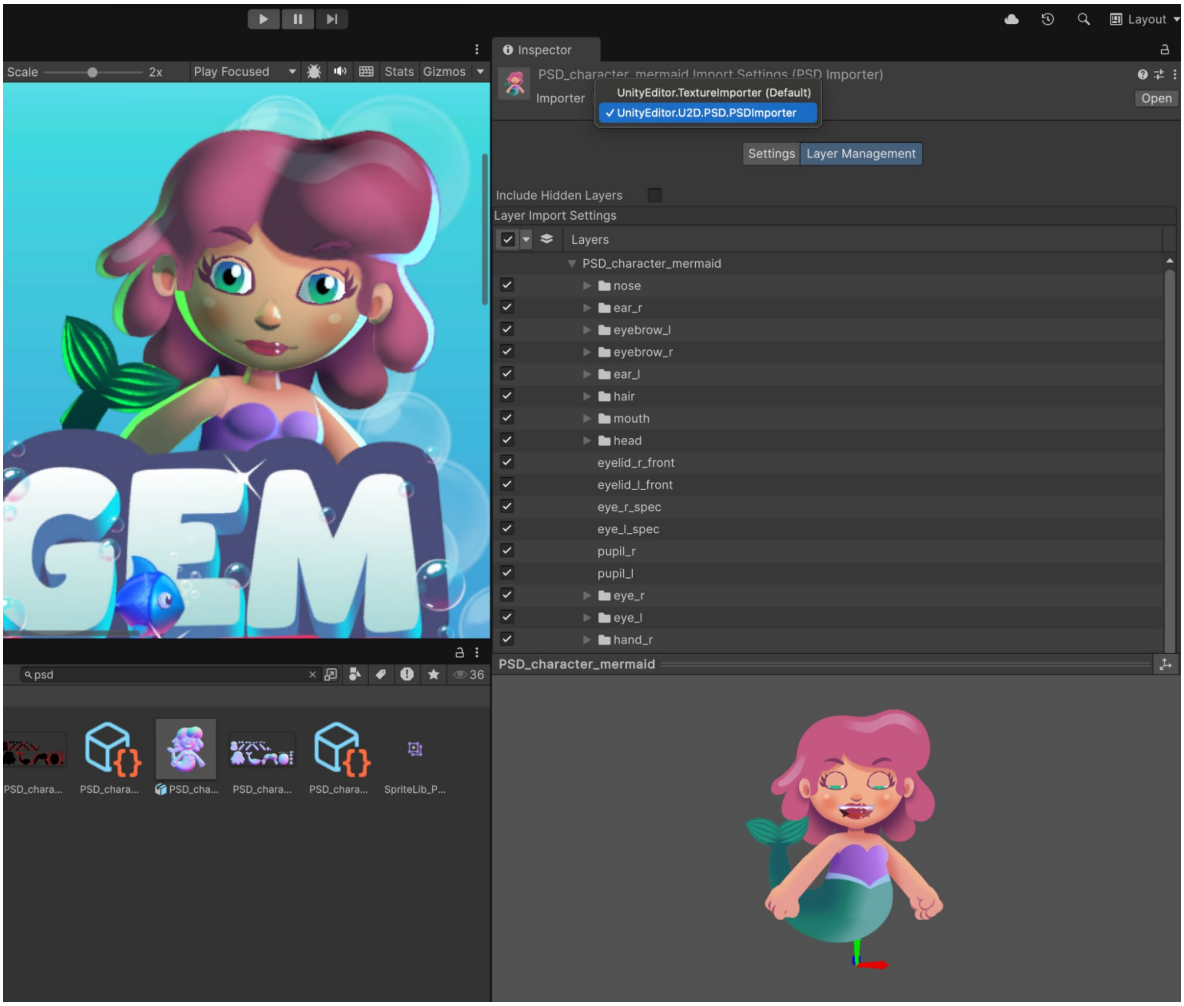
PSD Importer를 사용하여 라운드트립 임포트 프로세스 속도 높이기

PSD Importer는 Adobe Photoshop 파일을 Unity로 임포트하여 임포트된 소스 파일을 기반으로 스프라이트 프리팹을 생성합니다.

PSD Importer를 사용하여 2D 스프라이트, UI 요소 또는 Photoshop으로 제작한 에셋의 임포트 프로세스 속도를 높일 수 있습니다. 기존에는 아티스트가 Photoshop 파일의 각 레이어를 .png 파일로 익스포트하여 Unity에서 사용했으며, 변경 사항을 적용할 때마다 .png 파일을 다시 익스포트해야 했습니다. 이제 Unity에서 레어어가 적용된 .psd 포맷을 지원하므로 더 이상 이처럼 번거로운 작업이 필요하지 않습니다.

각 팔다리나 부위가 서로 다른 레이어에 있는 2D용 애니메이션 캐릭터도 Unity로 임포트하여 2D 애니메이션에 사용할 수 있습니다. 그러면 나중에 캐릭터를 리깅하고 애니메이션화할 수 있도록 Unity가 캐릭터를 설정해 줍니다.

모든 PSD 레이어를 스탠드얼론 스프라이트로 사용해야 하는 경우에는 **Character Rig** 옵션을 선택 해제하여 프리팹 오브젝트를 제거하면 모든 레이어를 스탠드얼론 스프라이트로 사용할 수 있는 단순한 슬라이스 스프라이트를 얻을 수 있습니다.



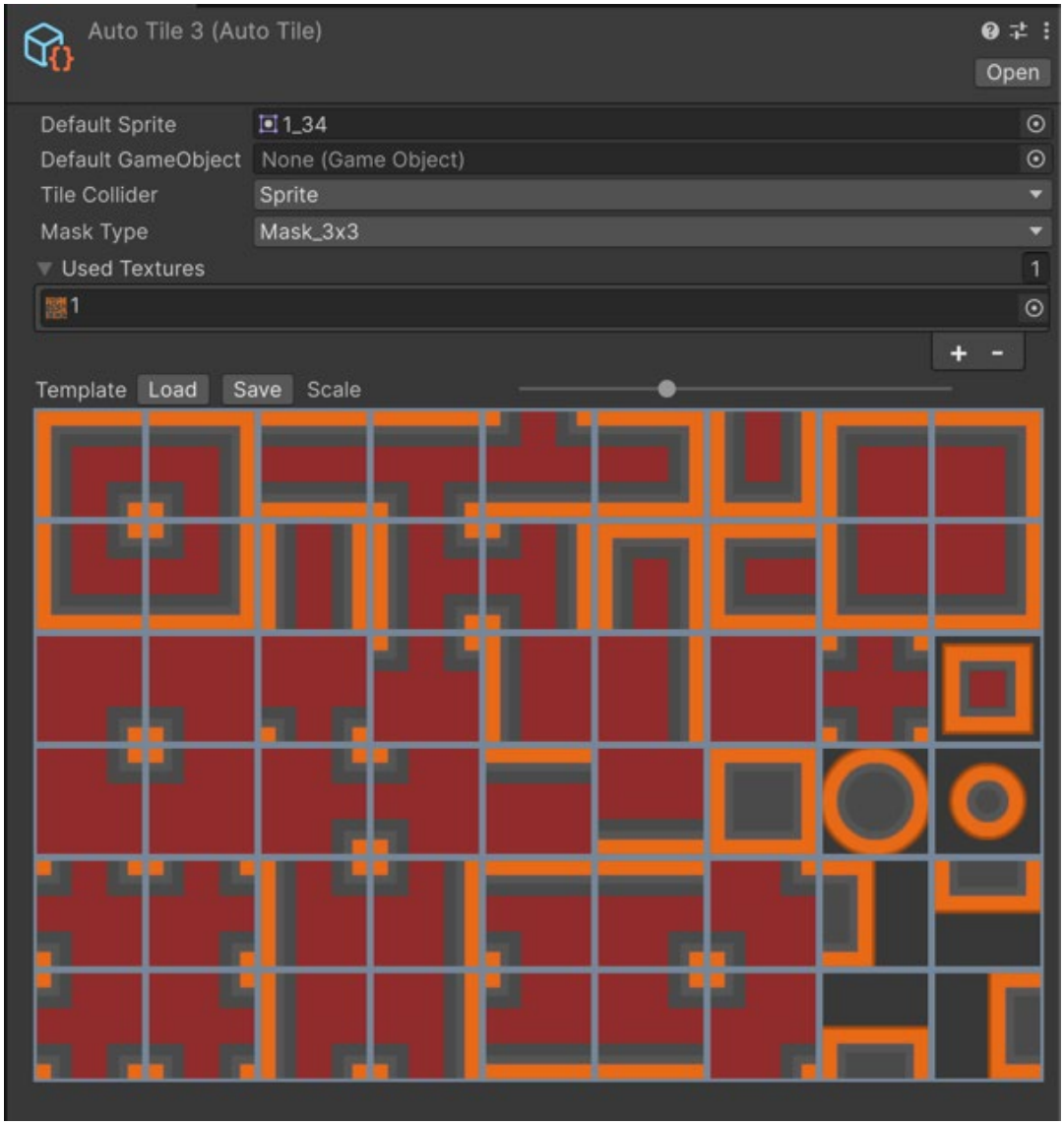
모든 임포터 기능에 액세스할 수 있도록 기본 텍스처 임포터가 아니라 PSD Importer를 사용하여 레이어가 적용된 PSD 파일을 임포트해야 합니다.

Photoshop에서 Unity로, PSD Importer 사용법 (영어) 동영상 튜토리얼에서 더 많은 팁을 확인하세요.

Unity 6.1의 Tilemap Editor를 사용한 간편한 규칙 타일

Unity의 타일맵 시스템은 그리드 기반 2D 레벨 생성을 위한 타일 에셋을 저장하고 처리합니다. 따라서 Unity에서는 쉽게 레벨 디자인 사이클을 만들고 반복 작업(iteration)할 수 있습니다.

타일을 사용할 때 번거로운 점 중 하나로 모든 인접 타일 에셋을 만들고 이를 규칙 타일 에셋에 할당하는 것을 꼽을 수 있습니다. Unity 6.1에서는 AutoTile이라는 새로운 옵션이 이 프로세스를 간소화합니다. 타일 시트를 로드한 다음 내부 영역을 선택하고 저장하면 타일 팔레트에서 사용할 수 있는 타일 에셋이 생성됩니다.

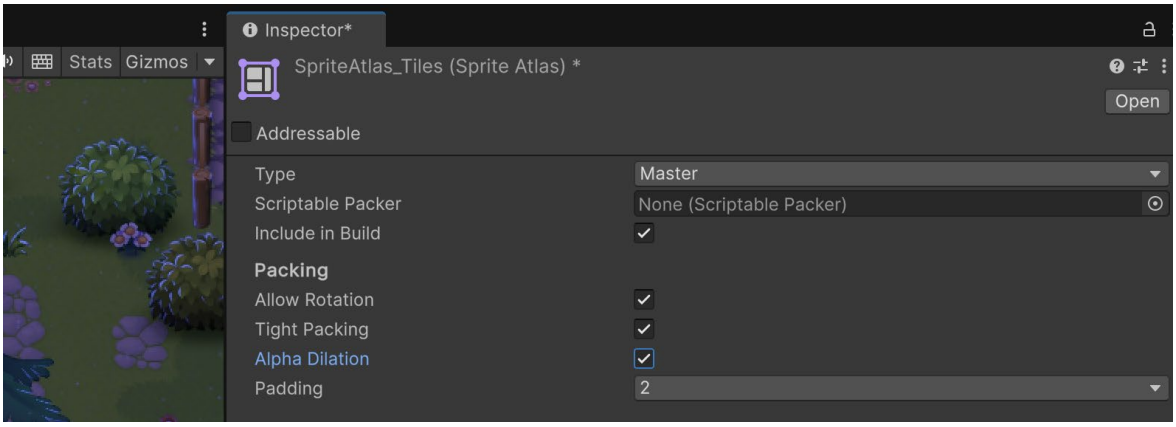


AutoTile 예시

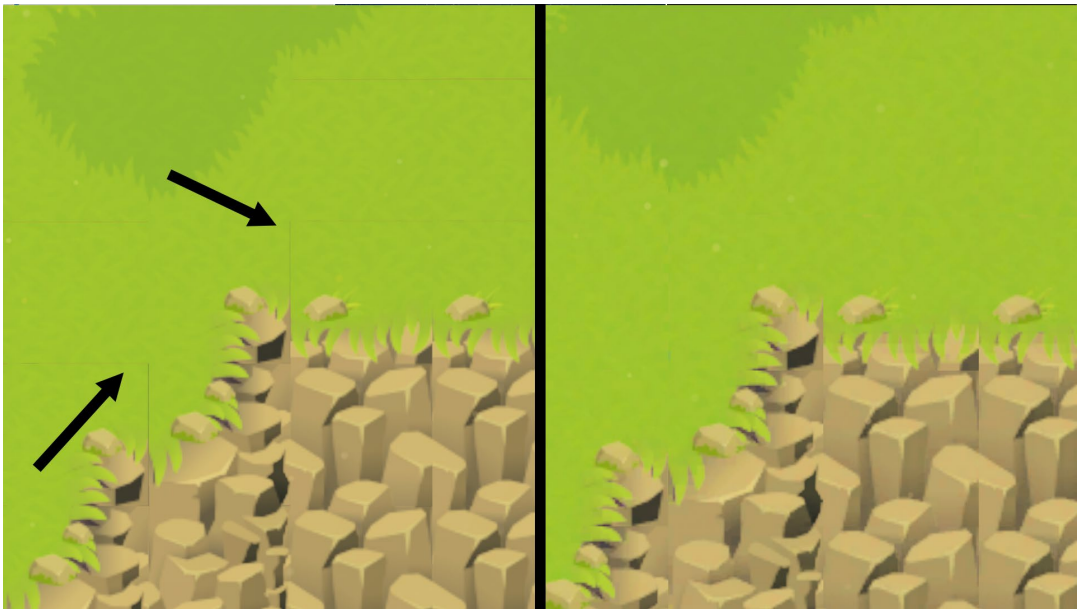
타일 사이의 텍스처 번짐이나 작은 틈 방지

아트와 타일맵에 큰 노력을 들였다면, 보간 처리와 가장자리 평탄화 때문에 타일 사이에 번짐이나 작은 틈이 보이는 현상을 방지하고 싶을 것입니다(픽셀 아트 게임에서는 스프라이트에 평탄화가 적용되지 않기 때문에 이러한 문제가 발생하지 않습니다).

타일 시트를 사용하지 않는다면 스프라이트 아틀라스를 사용하여 경계를 평탄화하고 TilemapRenderer의 내부 정렬을 사용하여 스프라이트를 패킹할 것을 권장합니다. 이는 프로젝트 구성, 성능, 소재 제어 측면에서 더 나은 결과를 제공합니다. 대부분의 스프라이트는 기본 설정을 사용해도 충분하지만, 타일맵의 경우 기본 설정을 수정해야 할 수 있습니다. 스프라이트 패킹 시 회전 방지나 Alpha Dilation 같은 기능은 타일의 가장자리를 선명하게 유지하는 데 도움이 됩니다.



스프라이트 아틀라스의 Alpha Dilation 옵션



왼쪽 이미지는 타일의 가장자리가 번지는 것을 확인할 수 있는데, 오른쪽 이미지는 스프라이트 아틀라스의 설정을 조정하여 이 현상이 사라졌습니다.

2D 역운동학(IK) 시스템으로 자연스러운 움직임 연출

Unity에는 골격 애니메이션을 위한 완벽한 솔루션이 있습니다. 바로 2D Animation입니다. 동작을 리깅할 때는 Unity의 2D IK(역운동학) 시스템을 사용하세요. IK를 사용하면 뼈대를 하나하나 수동으로 조정하지 않고도 무릎 굽히기, 손 뻗기 같은 자연스럽고 역동적인 움직임을 연출할 수 있습니다.

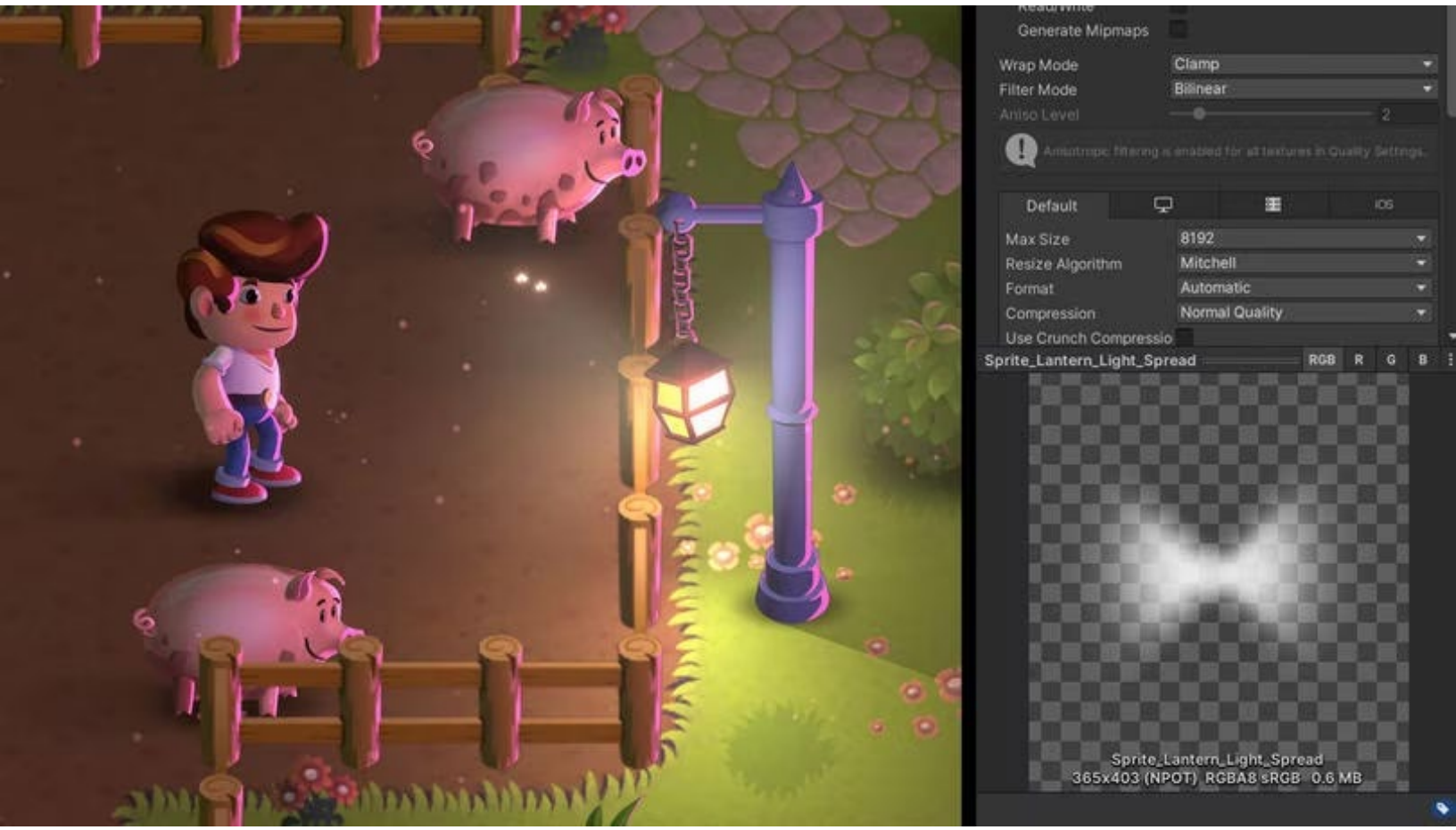
해피 하비스트 샘플 프로젝트에서 메인 캐릭터의 애니메이션 프로세스 중에 2D IK를 사용하는 예시

뼈대를 올바르게 제어할 수 있도록 스프라이트 에디터에서 뼈대 계층을 신중하게 설계하세요. 손 뼈대를 팔 뼈대의 자식으로 설정하는 것을 예로 들 수 있습니다. 간단한 애니메이션으로 계층을 테스트하여 상위 레벨 뼈대가 조정되면 자식 뼈대도 조정되는지 확인하세요.

속도나 입력 방향 같은 변수에 따라 걷기, 달리기, 대기 상태 등의 애니메이션을 부드럽게 전환하려면 애니메이션 블렌드 트리를 사용하세요. 이렇게 하면 상태 전환이 갑작스럽지 않고 자연스러운 애니메이션을 구현할 수 있습니다.

2D 광원을 사용한 IES 프로파일 시뮬레이션

3D 소프트웨어는 HDRP 같은 환경에서 IES 프로파일을 사용하여 광원의 감쇠를 시뮬레이션하는 것이 일반적입니다. Unity의 2D 광원 시스템은 광원 컬러, 강도, 감쇠, 블렌딩 효과 등 간편하게 구성할 수 있는 파라미터를 사용하여 유연하게 설정할 수 있습니다. 광원의 감쇠 효과를 더 세밀하게 구현하려면 스프라이트 유형 광원과 커스텀 광원 효과를 사용하세요.

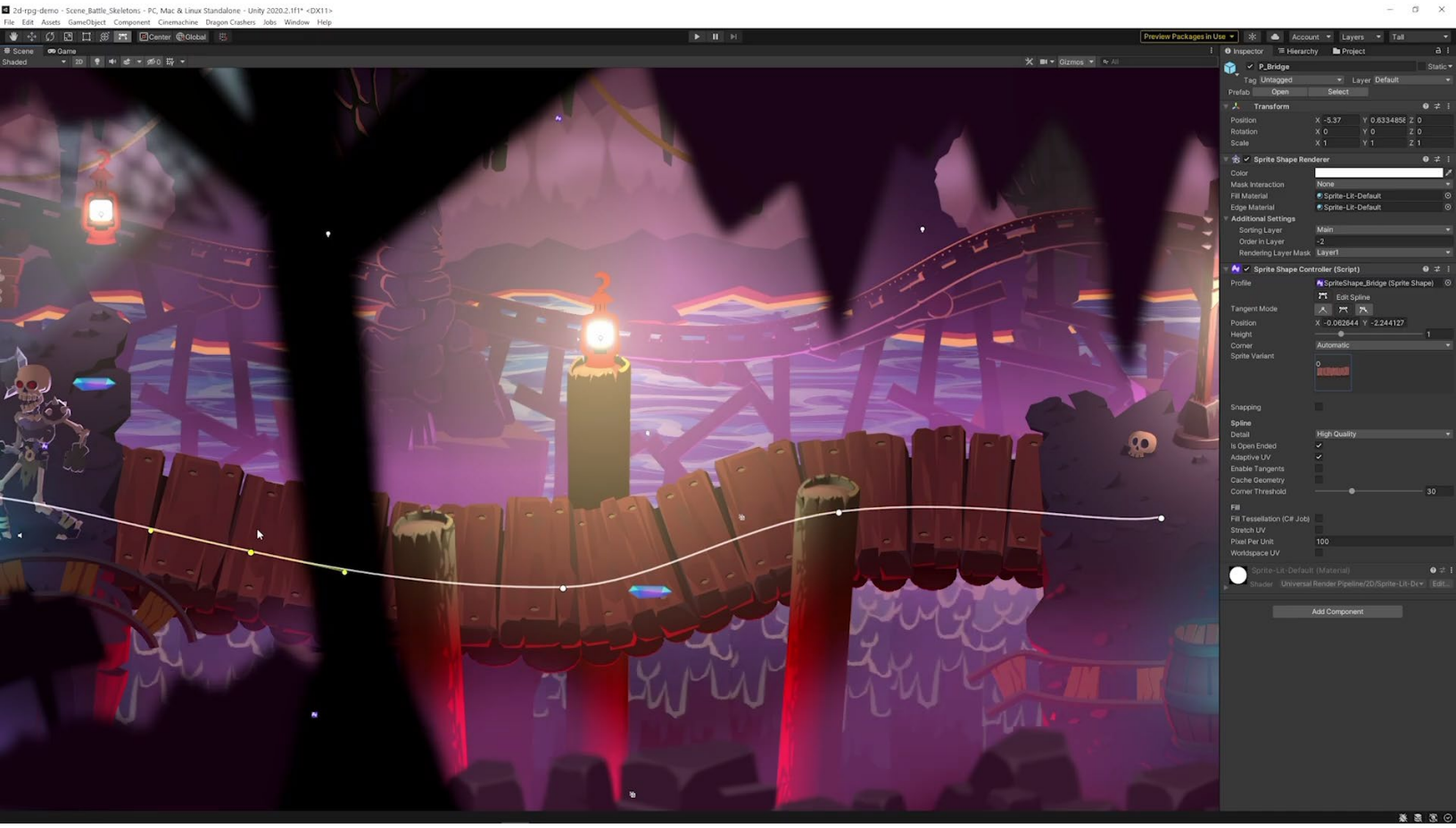


해피 하비스트 샘플 프로젝트에서 매달려 있는 램프 주변 헤일로에 스프라이트를 사용하는 예시

Odd Bug Studio의 마틴 라인만이 작성한 이 문서와 유니버설 렌더 파이프라인의 2D 광원 및 그림자 기법에 관한 이 페이지에서 2D 조명에 대한 팁을 추가로 확인할 수 있습니다.

2D 스프라이트 셰이프를 사용하여 풍부한 프리폼 2D 환경 만들기

2D 스프라이트 셰이프를 사용하면 시각적이고 직관적인 워크플로로 풍부한 프리폼 2D 환경을 자유롭게 제작할 수 있습니다. 2D 스프라이트 셰이프는 셰이프의 윤곽선을 따라 스프라이트를 타일링하여 윤곽선 각도에 따라 자동으로 스프라이트를 변형하고 교체합니다.



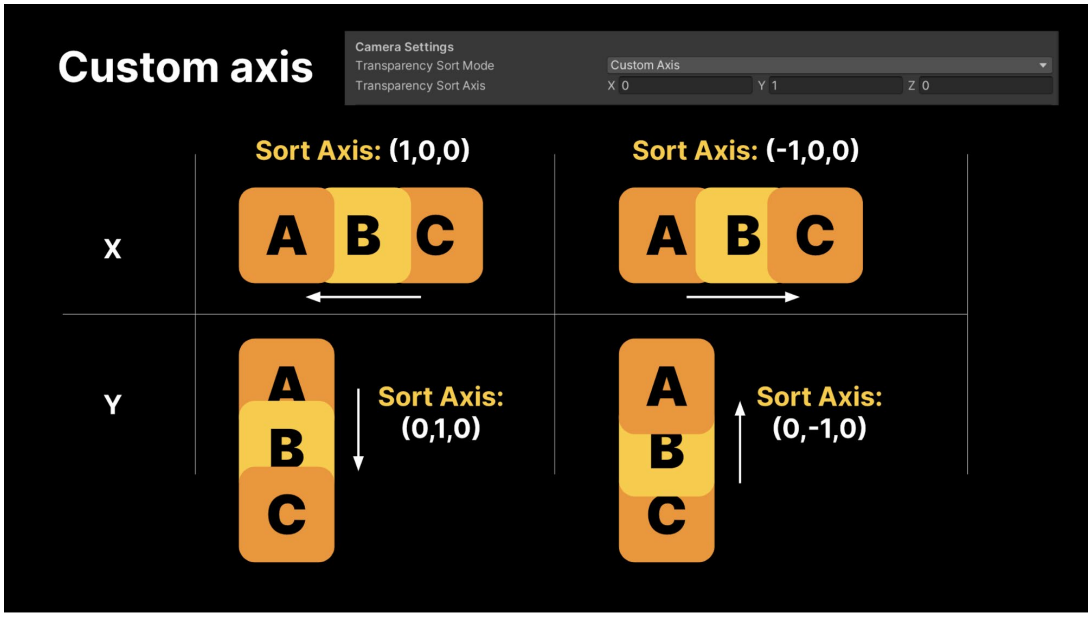
드래곤 크래셔 2D 샘플 프로젝트에서 2D 스프라이트 셰이프가 사용되는 예시

물리 2D 시스템으로 경로나 셰이프를 사용하려는 경우, 스프라이트 셰이프는 폴리곤 콜라이더 2D를 자동으로 생성합니다.

커스텀 스프라이트 정렬 축

원하는 방향으로 스프라이트를 정렬할 수 있습니다. 이는 같은 레이어에 동일한 순서로 정렬된 여러 개의 스프라이트를 특정 방식으로 정렬하려는 경우에 유용합니다. 각 카드가 조금씩 겹쳐져 있는 카드 게임에서 화면 하단에 있는 카드가 맨 위에 표시되도록 렌더링하려는 경우를 예로 들어 보겠습니다.

URP의 2D 렌더러 에셋에서 **General > Transparency Sort Mode**를 선택하고 **Custom Axis**를 선택합니다. 예를 들어 Y축을 따라 위에서 아래로 정렬하려면 **Transparency Sort Axis**에 (0, 1, 0)을 사용하면 됩니다.



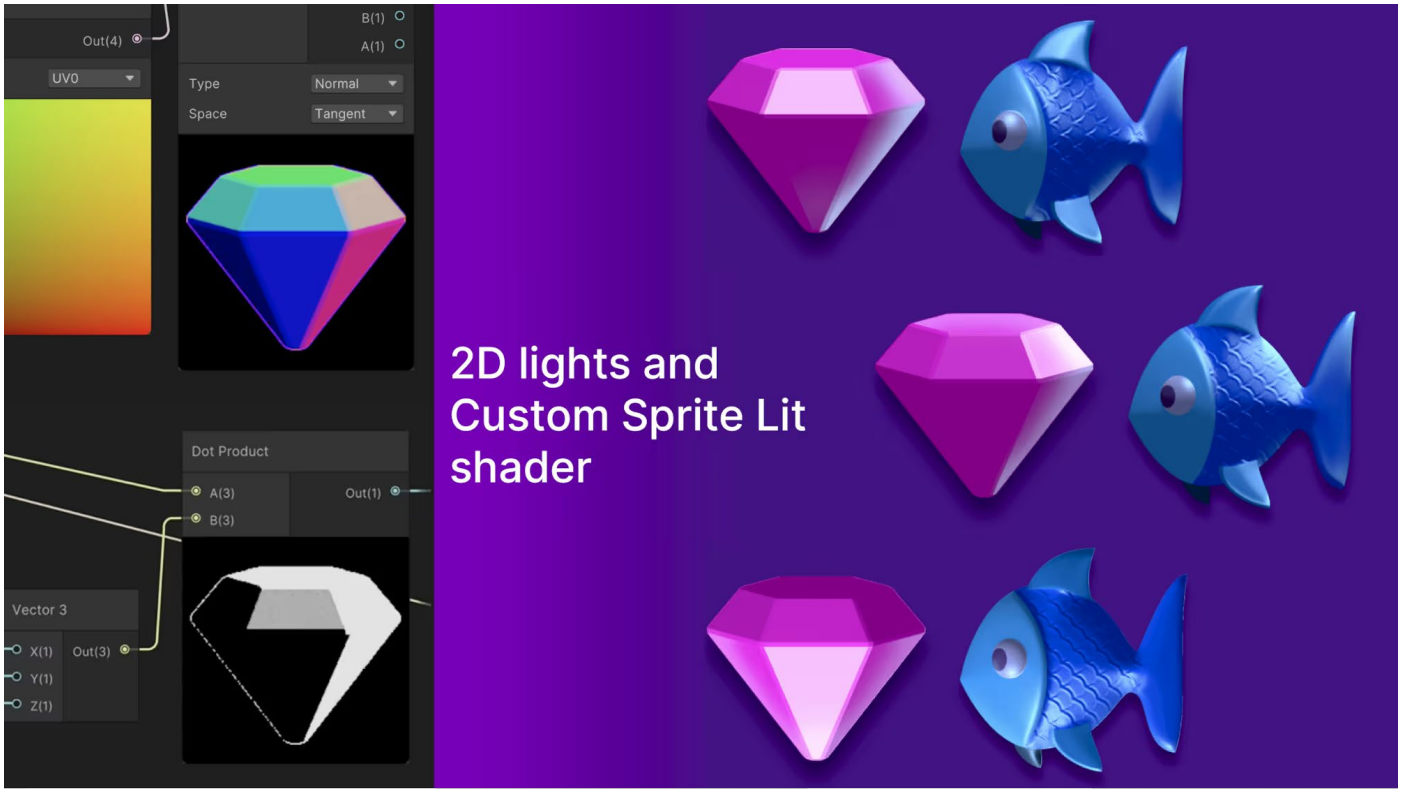
Transparency Sort Mode 및 Transparency Sort Axis 예시. 이 동영상 튜토리얼에서 더 많은 팁을 확인하세요(이전 Unity 버전 기반이나 여전히 활용 가능).

스프라이트 커스텀 릿 셰이더를 사용하여 커스텀 조명 만들기

전역 씬 조명과 별도로 작용하는 조명 효과를 만들려면 스프라이트 커스텀 릿 셰이더를 사용해 보세요.

젬 헌터 매치(Gem Hunter Match) 샘플에서 비주얼 효과를 생성하는 데 이 기법이 사용되었습니다. 이 셰이더는 씬 조명을 대체하기 때문에 2D 광원 텍스처 정보를 수정하고 요소별로 조명을 제어할 수 있습니다. 그 결과 여러 요소 위를 움직이며 반짝이는 효과 같은 창의적인 스프라이트 조명을 구현할 수 있습니다.

광원 위치 데이터가 셰이더 내부로 이동하므로 씬에 실제 광원 오브젝트를 배치할 필요가 없어지는데, 이에 따라 씬을 깔끔하게 유지하는 데 도움이 됩니다. 셰이더 내에 캡슐화된 오브젝트별 조명으로 대규모 분리 및 편집 작업이 용이해지며, 배칭이 가능한 경우 성능 향상도 이룰 수 있습니다.

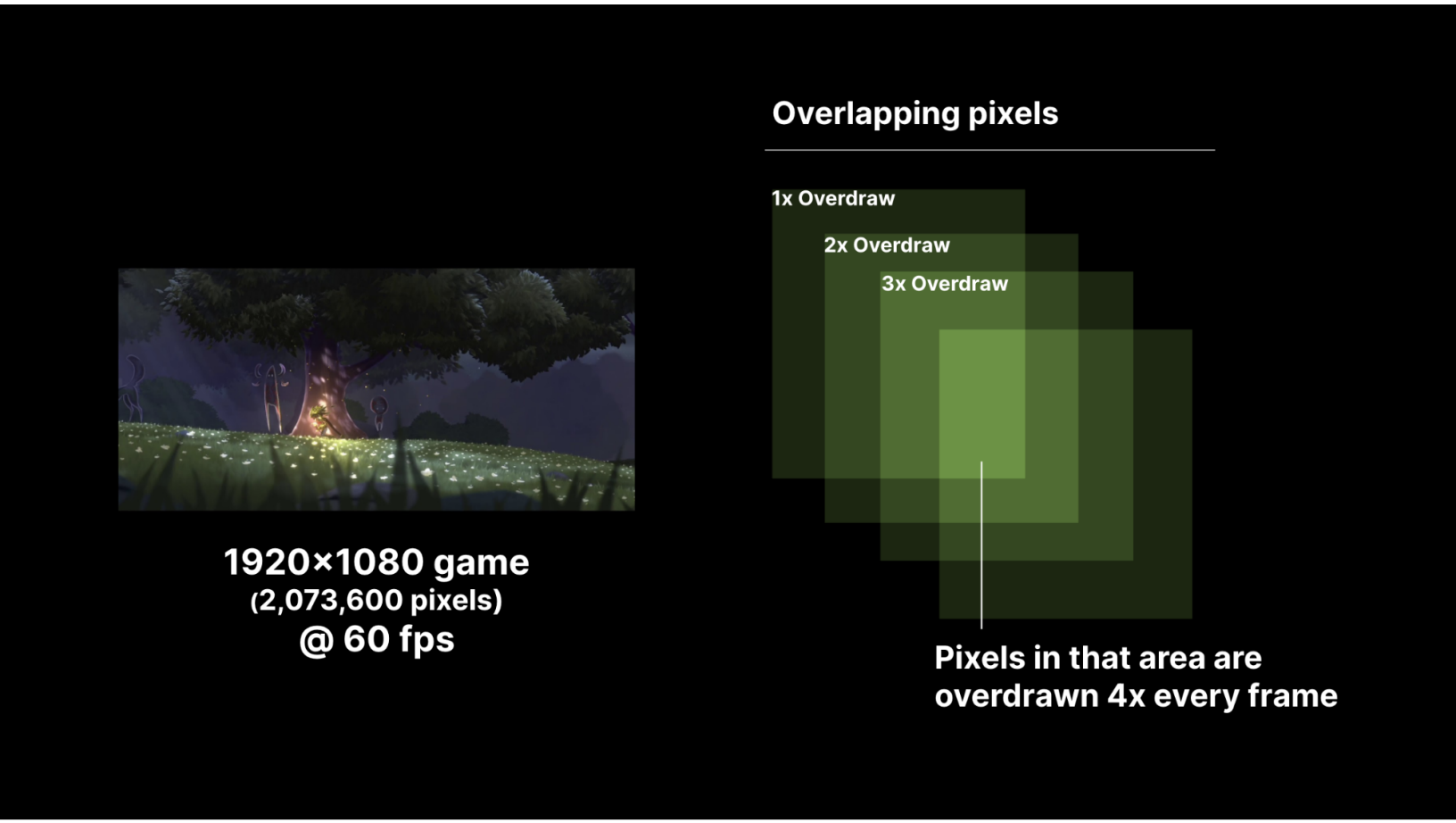


젬 헌터 매치의 스프라이트 커스텀 릿 셰이더

커스텀 스프라이트 릿 셰이더를 사용하여 2D 스프라이트의 오브젝트별 조명으로 소재 제어를 개선한 방식을 유니티의 새로운 매치3 샘플 젬 헌터 매치가 선보이는 다양한 조명 및 시각 효과 블로그 게시물에서 알아보세요.

투명도 픽셀의 오버드로우 줄이기

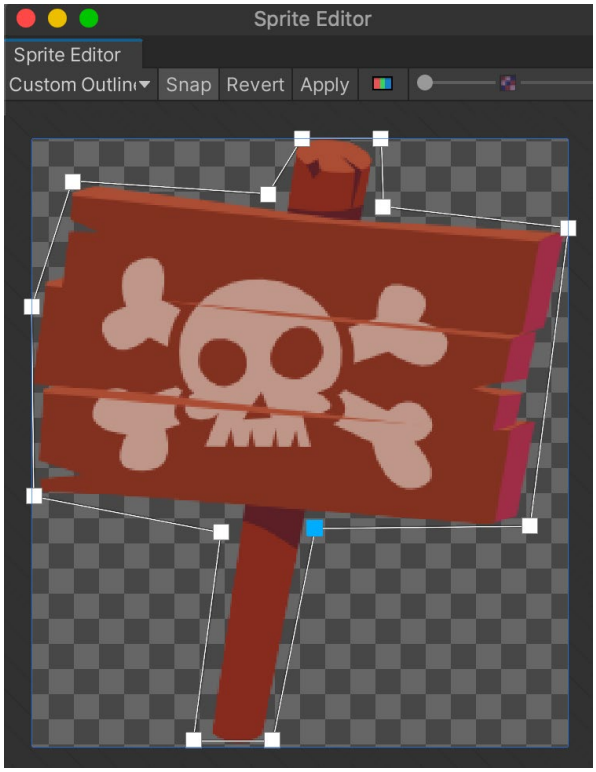
픽셀 오버드로우를 방지하여 성능을 개선하세요. Import Settings에서 **Mesh Type**을 **Tight**(기본 옵션)로 전환합니다. 가능하면 하나의 스프라이트에서 겹치는 그래픽스를 병합하고 게임에서 사용하지 않는 배경 레이어에 있는 스프라이트를 비활성화합니다. 그러면 오버드로우 영역 및 인접한 스프라이트와 겹칠 가능성을 줄일 수 있습니다.



스프라이트 사이의 오버드로우를 줄입니다.

스프라이트 에디터를 사용하여 사용되지 않는 영역 최소화

2D 스프라이트 에디터로 각 스프라이트 주변에 커스텀 윤곽선을 정의하면 스프라이트를 패킹할 때 사용되지 않는 영역을 최소화하고 오버드로우를 방지할 수 있습니다.

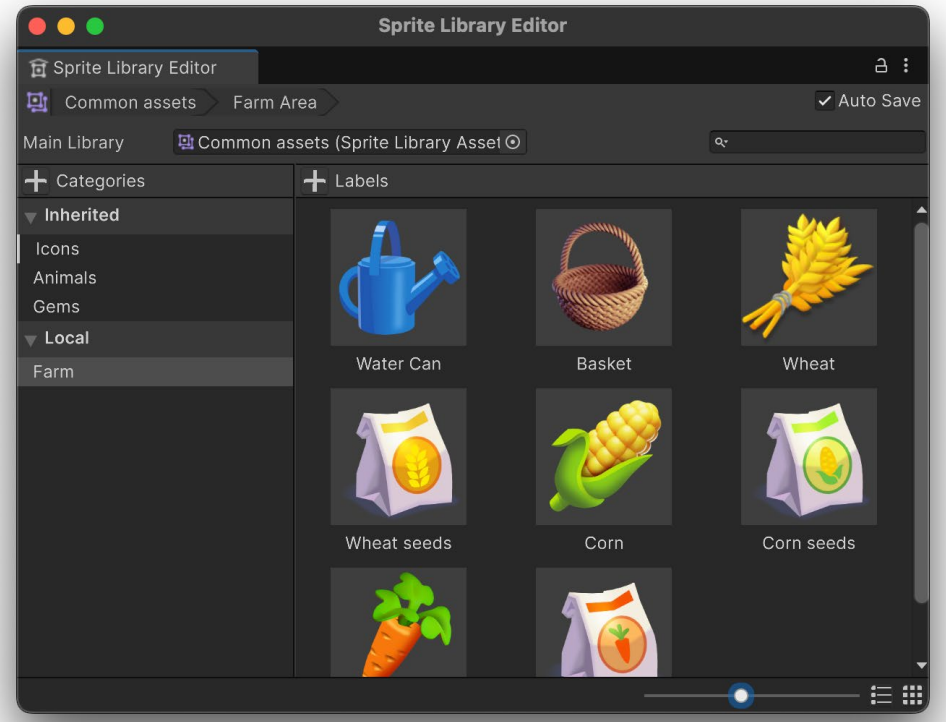


커스텀 윤곽선이 있는 스프라이트 에디터

스프라이트 라이브러리로 프로젝트 스프라이트 및 애니메이션 정리

스프라이트 라이브러리를 사용하여 레벨 디자인과 여러 파트로 구성된 캐릭터의 스프라이트를 손쉽게 정리하세요.

스프라이트 라이브러리는 렌더링 성능을 위해 스프라이트를 그룹화하도록 설계된 스프라이트 아틀라스와 다릅니다. 단 스프라이트 라이브러리, 스프라이트 아틀라스, 어드레서블은 서로 호환되므로 조합을 통해 대규모 프로젝트를 체계적으로 관리할 수 있습니다.



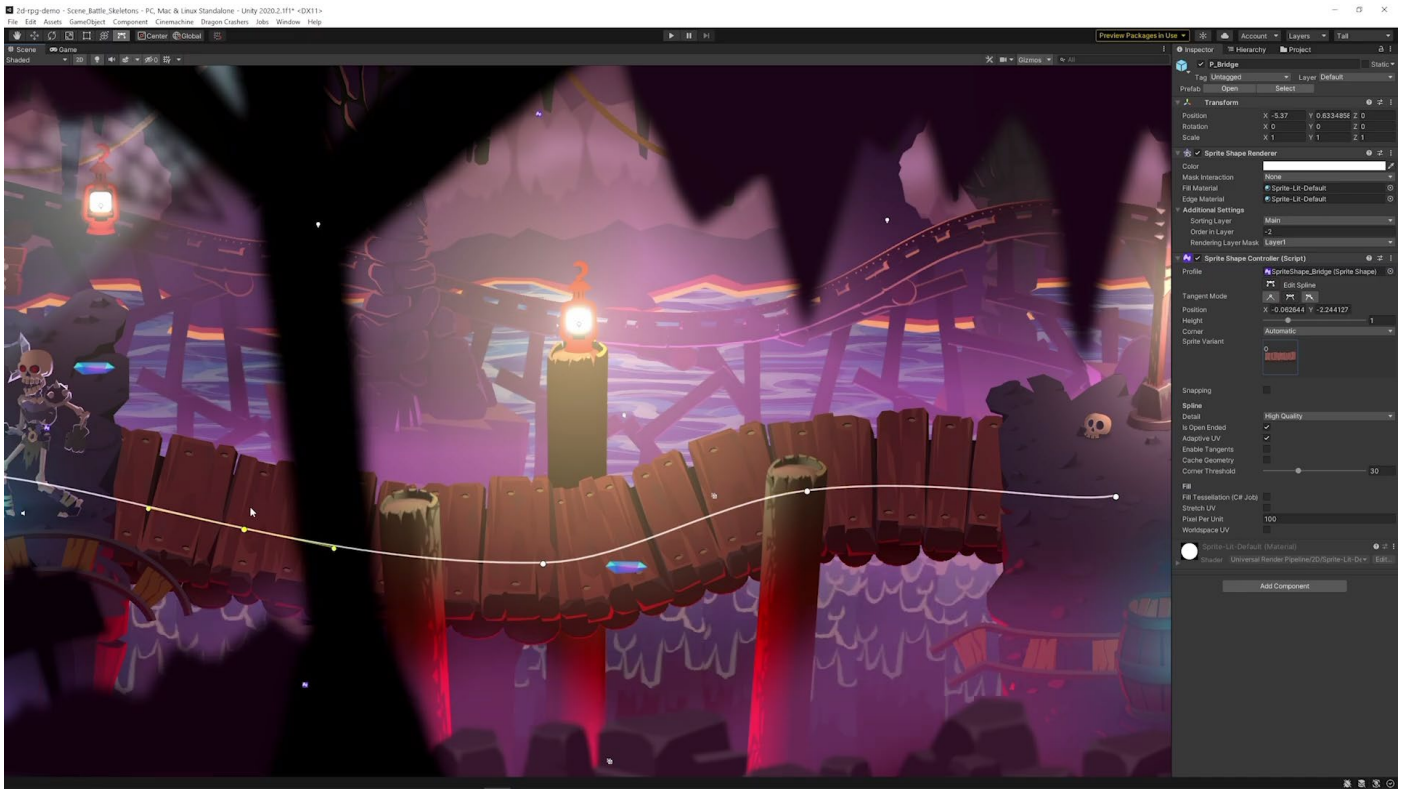
스프라이트 라이브러리 에디터 창

스프라이트 셰이프 컨트롤 포인트 수정

스프라이트 셰이프는 언덕, 역동적인 도로, 트레일, 레이스 트랙, 수역, 장식 요소를 만들어서 유기적인 2D 터레인을 디자인하는 데 사용할 수 있습니다.

Unity 2D 물리 시스템과 결합하면 실시간으로 변화하고 형태가 바뀌는 인터랙티브 영역(예: 파괴 가능한 터레인이나 변형 가능한 플랫폼) 같은 게임플레이 기능을 구현할 수도 있습니다.

스프라이트 셰이프의 컨트롤 포인트를 수정할 수 있습니다. API를 통해 런타임 또는 에디터 타임에 포인트에 액세스할 수 있습니다. 이때 성능 비용이 따르지만 신중하게 통합하면 게임에 차별화되는 요소를 더할 수 있습니다.



드래곤 크래셔 2D URP 샘플 프로젝트에서 스프라이트 셰이프 컨트롤 포인트를 수정하는 예시.

컨텍스트 메뉴에서 편리하게 스프라이트 교체

Sprite Resolver는 2D 스프라이트 라이브러리 시스템과 함께 작동하여 런타임에 여러 스프라이트 비주얼을 동적으로 바꿀 수 있는 기능을 제공하는 컴포넌트입니다.

인스펙터에서 수동으로 할당하지 않고도 게임 오브젝트의 스프라이트를 바꿀 수 있어 프로젝트에서 스프라이트를 더 유연하고 효과적으로 관리할 수 있습니다.

다음은 Sprite Resolver의 권장 사용 사례입니다.

— 캐릭터 커스터마이징 시스템

— NPC 또는 캐릭터의 역동적인 얼굴 표정과 기분 표현

— 시즌 또는 인게임 이벤트에 따른 환경 스프라이트 변화

— 오브젝트의 시각적 업그레이드 또는 디그레이드

— 애니메이션 스프라이트 스왑 간소화로 효율성과 정리 개선

Unity 6에서는 인스펙터 창과 에디터 레이아웃에서 Sprite Resolver를 사용할 수 있습니다.

스프라이트 라이브러리의 Sprite Resolver 컴포넌트

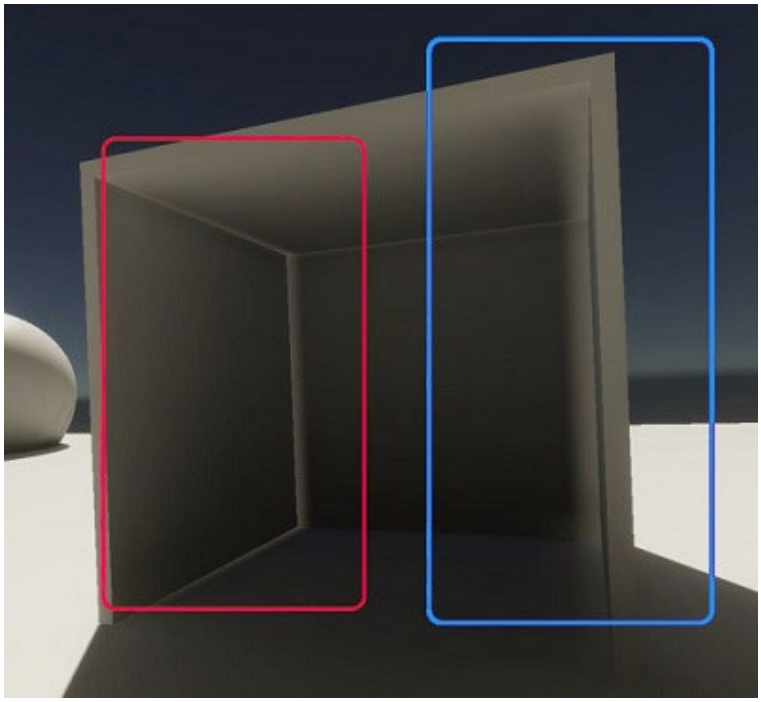
그래픽스 및 아트 에셋

라이트 프로브로 인한 빛 번짐 현상

빛 번짐 현상은 지오메트리가 해당 지오메트리에서 볼 수 없는 벽 반대편의 라이트 프로브에서 빛을 받을 때 주로 발생합니다.

간단히 다룰 수 있는 주제는 아니지만, 다음과 같은 기법을 사용하여 빛 번짐 현상을 줄일 수 있습니다.

- 더 두꺼운 벽 만들기
- 씬에 Adaptive Probe Volumes Options 오버라이드 추가하기
- 렌더링 레이어 활성화하기
- 베이킹 세트 프로퍼티 조정하기
- Probe Adjustment Volume 사용하기

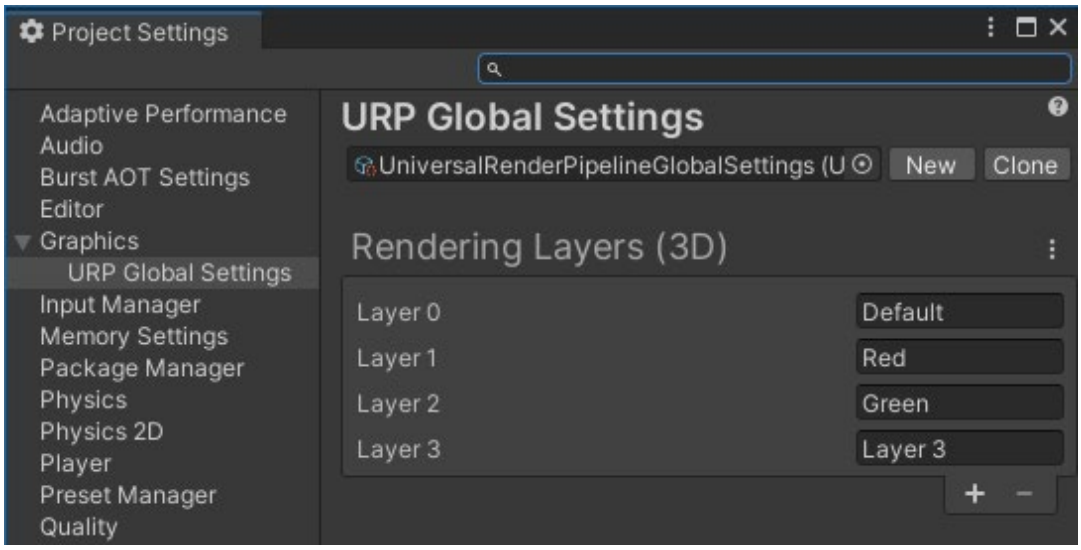


빛 번짐 현상 예시

렌더링 레이어를 사용하여 특정 게임 오브젝트에 특정 광원 구성

렌더링 레이어 기능을 사용하면 특정 게임 오브젝트에만 영향을 주도록 특정 광원을 구성할 수 있습니다.

Custom Shadow Layers 프로퍼티를 사용하여 특정 게임 오브젝트가 특정 광원에서만 그림자를 드리우도록 구성할 수 있습니다(해당 광원이 해당 게임 오브젝트에 영향을 주지 않을 때도 가능).

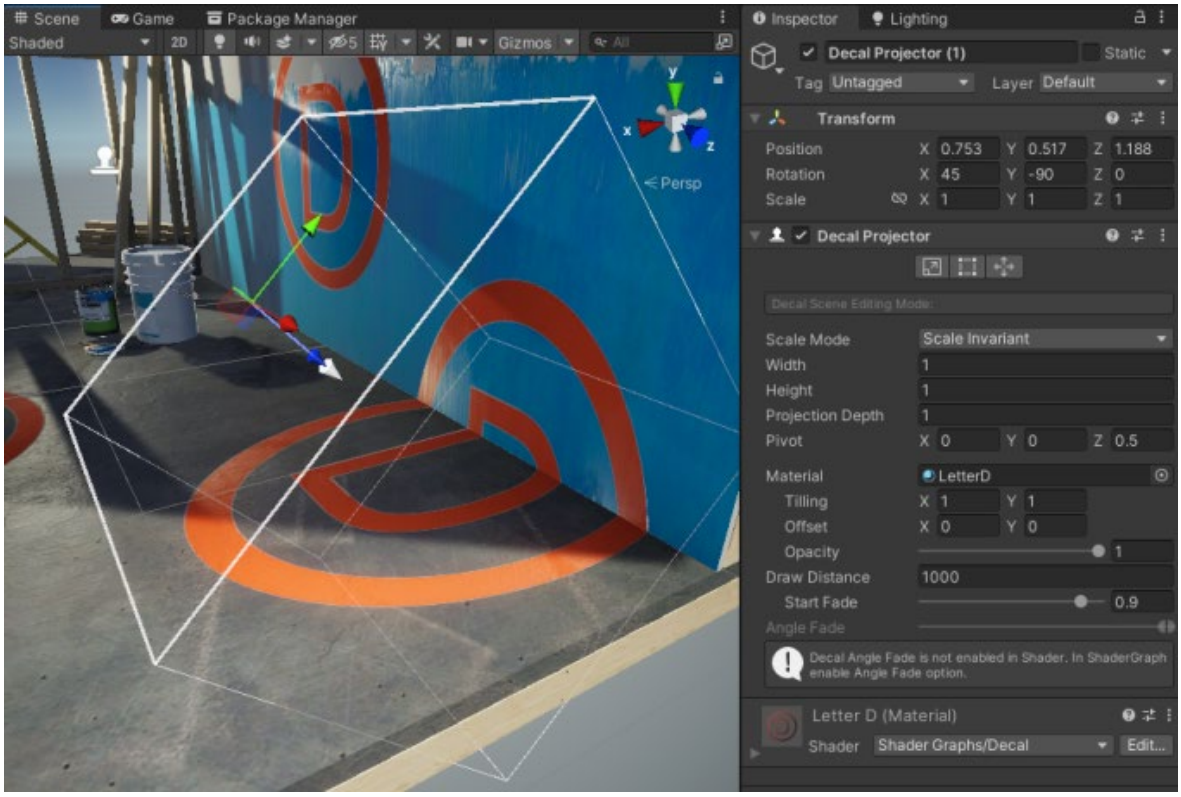


URP Global Settings의 렌더링 레이어

데칼 프로젝터를 사용하여 메시에 디테일 추가

Decal Projector 컴포넌트는 메시에 디테일을 추가하는 효과적인 방법입니다. 총알 구멍, 발자국, 간판, 균열 등의 요소에 적합합니다.

투영 프레임워크를 사용하므로 울퉁불퉁하거나 휘어진 표면에도 적용할 수 있습니다. URP에서 데칼 프로젝터를 사용하려면 렌더러 데이터 에셋에 **데칼 렌더러 기능**을 추가해야 합니다.



씬 뷰에 적용된 데칼 프로젝터

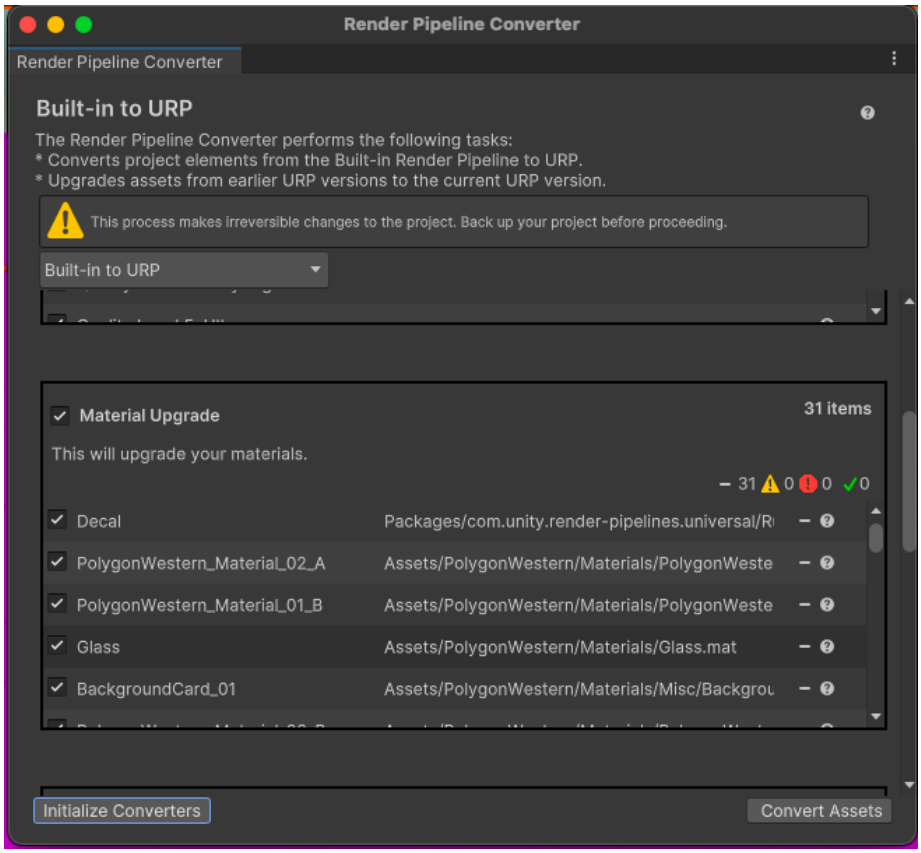
빌트인 렌더 파이프라인에서 URP로 커스텀 셰이더 전환

URP 기술 자료의 이 표를 참조하면 각 URP 셰이더가 해당하는 빌트인 렌더 파이프라인 셰이더에 어떻게 매핑되는지 확인할 수 있습니다.

하나 이상의 컨버터를 선택한 후 **Initialize Converters** 또는 **Initialize And Convert**를 클릭합니다. 어떤 옵션을 선택하든 프로젝트가 스캔되고 전환이 필요한 에셋이 각 컨버터 패널에 추가됩니다.

Initialize Converters를 선택한 경우, 각각에 대해 제공된 체크박스를 사용하여 항목을 선택 해지함으로써 전환을 제한할 수 있습니다. 이 단계에서 **Convert Assets**를 클릭하면 전환 프로세스가 시작됩니다.

Initialize And Convert를 선택하면 컨버터가 초기화된 후 자동으로 전환이 시작됩니다. 완료되고 나면 에디터에서 활성 상태인 씬을 다시 열라는 메시지가 표시될 수 있습니다.

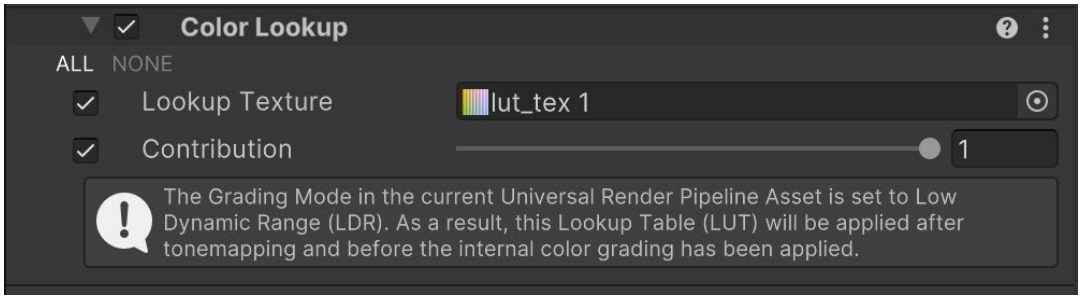


렌더 파이프라인 컨버터

LUT 텍스처를 사용하여 컬러 그레이딩 만들기

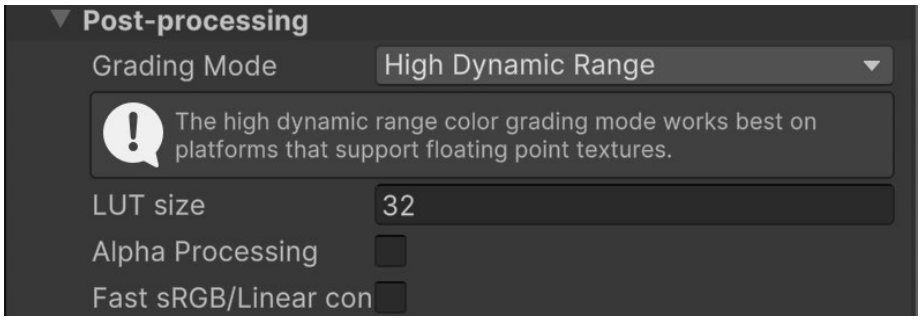
컬러 그레이딩 효과는 Unity가 생성하는 최종 이미지의 컬러와 휘도를 수정 또는 정정합니다. 이 효과를 사용하여 프로젝트의 디자인(look and feel)을 바꿀 수 있습니다.

Volume 설정에 추가해야 하는 포스트 프로세싱 효과에서 컬러 그레이딩 효과가 작동하는 방식을 **Lookup Texture 및 Contribution** 설정을 통해 제어하세요.

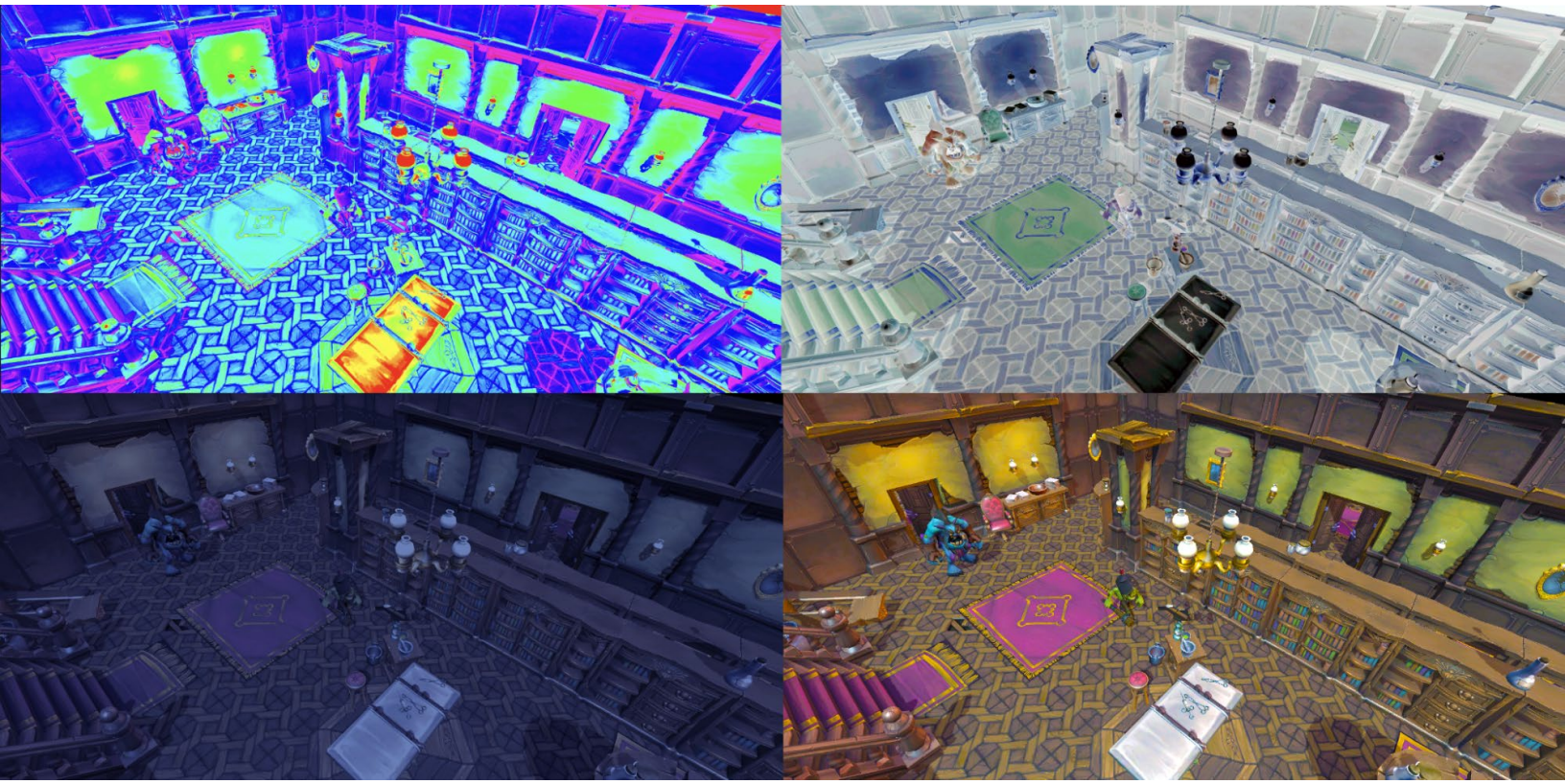


Color Lookup의 Lookup Texture 및 Contribution 설정

여러 LDR(Low Dynamic Range) 룩업 텍스처 간의 볼륨 블렌딩이 지원되긴 하나, 크기가 모두 동일한 경우에만 올바르게 작동합니다. 따라서 프로젝트 전체에서 단일 LUT 크기(256x16 또는 1024x32)를 사용하는 것이 권장됩니다.



URP 에셋 설정의 LUT 크기 설정



다양한 LUT 텍스처 활용

렌즈 플레어를 통한 사실적인 렌즈 효과와 스타일라이즈드 효과 연출

렌즈 플레어는 카메라 렌즈에 밝은 빛을 비출 때 나타나는 현상입니다. 하나의 밝은 빛으로 나타날 수도 있고, 카메라의 조리개 모양과 일치하는 여러 색상의 다각형 플레어로 나타날 수도 있습니다.

렌즈 플레어에 익숙해지는 가장 좋은 방법은 패키지 관리자를 통해 샘플을 설치하는 것입니다. 샘플을 설치하면 렌즈 플레어 효과가 포함된 사전 정의된 플레어 에셋 세트가 추가됩니다. 또한 샘플에는 렌즈 플레어를 살펴보고 직접 만들어 볼 수 있는 테스트 씬이 포함되어 있습니다.

렌즈 플레어 샘플은 사실적인 렌즈 효과부터 스타일라이즈드에 가까운 효과까지 다양한 프리셋을 제공합니다. 예시에서는 HDRP 샘플 씬이 사용되었습니다.

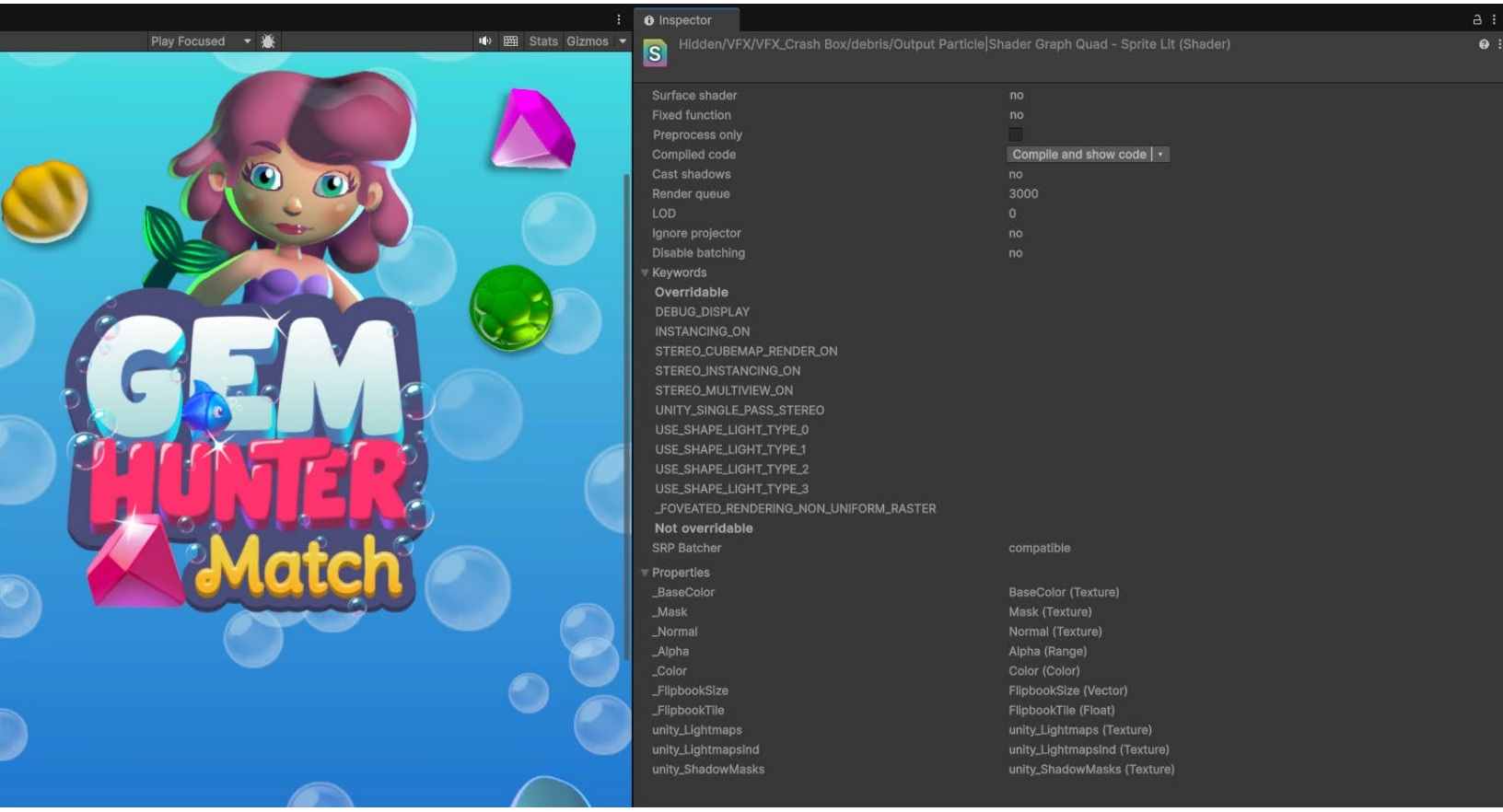
렌즈 플레어는 여러 Lens Flare Element로 구성되어 있으며, 각 요소는 플레어가 만드는 다양한 현상을 나타냅니다. 요소의 모양은 Polygon, Circle, 커스텀 이미지 중에서 선택할 수 있습니다.

Lens Flare Element 파라미터를 사용하면 **Color**, **Transform** 위치, 변형을 조정할 수 있으며, Lens Flare Element가 여러 요소가 있는 렌즈 플레어 효과의 일부인 경우 요소의 위치, 색상, 크기를 어떻게 설정할지 조정할 수 있습니다. 광원과 연결한 플레어 요소는 틴트 컬러를 사용할 수 있고 같은 플레어 에셋을 여러 광원에 다시 사용할 수 있습니다.

렌즈 플레어 작동 방식은 이 SIGGRAPH 발표에서 자세히 다루고 있습니다.

셰이더 배리언트 관리

빌드 크기와 시간을 줄이려면 셰이더 배리언트를 주의 깊게 관리해야 합니다. 특정 머티리얼에만 배리언트가 필요한 경우 #pragma multi_compile 대신 #pragma shader_feature를 사용하세요. shader_feature 배리언트는 머티리얼에서 사용되거나 런타임에 활성화되지 않으면 제거되지만, multi_compile 배리언트는 항상 포함됩니다. 셰이더의 **인스펙터**에서 키워드를 확인하세요.



젬 헌터 매치 샘플의 Universal Render Pipeline/Lit (Shader) 창의 Keywords 섹션

머티리얼의 배리언트 만들기

머티리얼 배리언트를 사용하면 템플릿이나 머티리얼 프리팹을 만들 수 있습니다. 템플릿 머티리얼과 공통 프로퍼티를 공유하고 다른 프로퍼티만 오버라이드하는 배리언트를 기본 템플릿을 기반으로 만들 수 있습니다. 템플릿 머티리얼에서 오버라이드되지 않은 공통 프로퍼티를 변경하면 배리언트 머티리얼에 자동으로 변경 사항이 반영됩니다. 특정 프로퍼티가 배리언트에서 오버라이드되지 않도록 머티리얼에서 잠글 수도 있습니다.

머티리얼 배리언트 예시. 모두 동일한 기본 머티리얼을 공유하며 컬러 프로퍼티만 다릅니다.

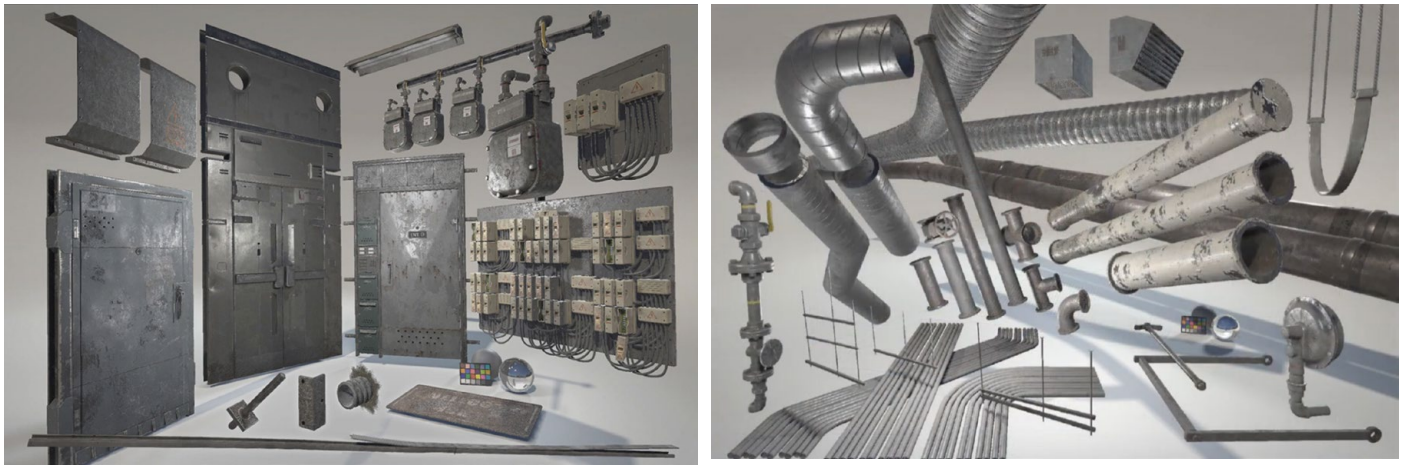
더 복잡한 설정에서는 머티리얼 배리언트의 배리에이션을 만들 수 있습니다. 머티리얼 상속 계층은 프로젝트에서 재사용 가능성을 높이고 머티리얼 저작(authoring)의 반복 속도와 확장성을 향상합니다.

에셋 그룹을 계획하여 에셋 워크플로 개선

사전 제작 계획 과정의 일환으로 팀은 만들어야 하는 필수 에셋 목록, 프로젝트의 기술적 요구 사항에 따른 예산, 개발 비용을 비롯한 에셋 워크플로의 다양한 측면을 합의해야 합니다.

필수적인 요소를 그룹으로 나눈 예를 들면 다음과 같습니다.

- **대형 요소**: 대형 요소는 고유한 텍스처로 저작할 수 없고 타일링이나 모듈식 요소를 사용해야 합니다. 예를 들면 벽, 바닥, 절벽, 건물 지붕 등이 있습니다.
- **중간 크기 프랍(prop)**: 통, 상자, 바위, 문 등 고유한 텍스처 시트를 적용하는 프랍입니다.
- **자주 반복되는 소형 요소**: 자갈, 나뭇잎, 나사, 볼트 등 지면에 활용하거나 스케일을 표현하는 데 사용되는 요소입니다.



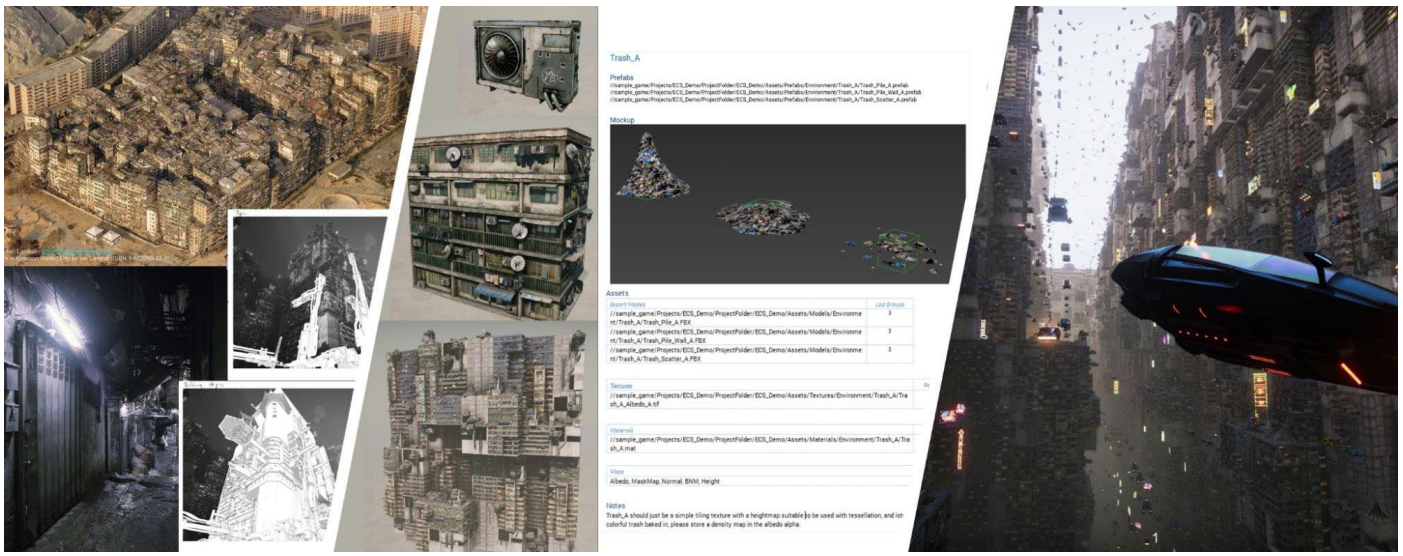
유형, 크기, 셰이프를 기준으로 에셋 제작 계획

각 에셋의 제작 시간, 폴리곤 수, 텍스처 해상도를 정하고 그에 따라 작업의 우선순위를 정해야 합니다. 그런 다음 필수적이지 않은 에셋을 만들고 필수 에셋의 수를 최소한으로 줄입니다.

AssetPostProcessor API를 통한 임포트 파이프라인 자동화 및 속도 개선

수천 개의 에셋이 포함된 제작 환경에서는 인스펙터에서 각 에셋의 사양을 직접 설정하지 않는 것이 좋습니다.

에셋 파일의 유효성 검사 프로세스를 자동화하려면 AssetPostProcessor API를 사용하면 됩니다. AssetPostProcessor를 사용하면 임포트 파이프라인에 연결하여 에셋을 임포트하기 전이나 후에 스크립트를 실행할 수 있습니다. 스크립트를 위한 에셋을 추가하여 필요한 부분을 변경하기만 하면 됩니다. 단일 스크립트에서 설정을 변경하면 프로젝트에서 해당 변경 사항을 에셋에 자동으로 다시 적용합니다.



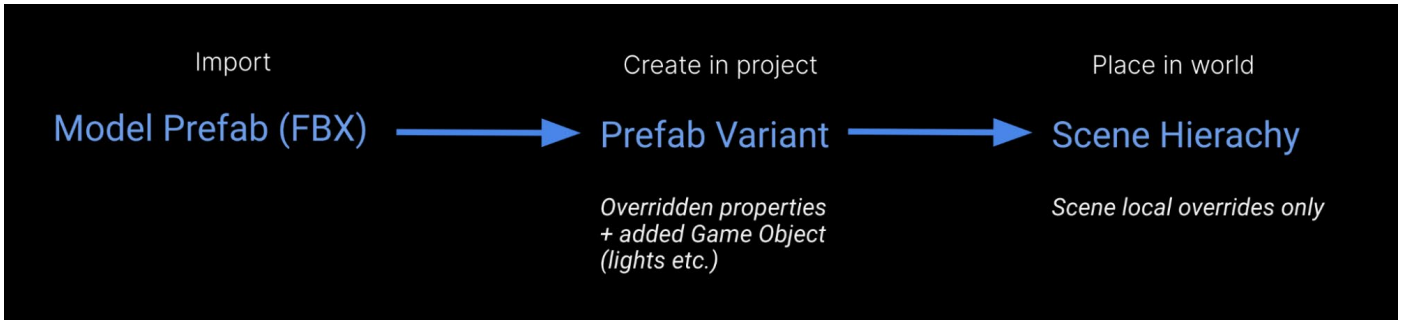
왼쪽부터 Unity에서 제작한 데모 메가시티(Megacity)의 무드 보드, 컨셉 아트, 프리팹 에셋, AssetPostProcessor를 사용한 디렉토리 및 명명 기준, 최종 비주얼 예시

프리팹 배리언트를 통한 팀워크 효율 향상

에셋 임포트 워크플로에서 중요한 또 다른 요소는 3D 모델에서 프리팹을 생성하는 것입니다. 제작 과정에서 FBX 모델을 프리팹으로 바로 전환하는 것은 좋지 않습니다. 지오메트리의 계층 구조가 바뀌는 등 FBX 파일이 변경되면 프리팹이 손상될 수 있기 때문입니다. 그렇게 되면 씬에 있는 모든 인스턴스를 업데이트된 버전으로 교체해야 합니다. 이 문제는 프리팹 배리언트로 해결할 수 있습니다.

프리팹 배리언트를 사용하면 씬에 종속되지 않는 아트 에셋을 제작할 수 있으며, 이렇게 만들어진 에셋은 프로토타이핑에서 사용하기 좋습니다. 게임 월드를 구축할 때 협업이 필요한 반복 작업에 특히 유용합니다. 프리팹 배리언트는 원본 모델의 배리에이션을 만드는 것과 같은 일반 프리팹의 장점과 동시에, FBX 모델 프리팹에 대한 '약한 참조'의 역할도 합니다. 따라서 프리팹을 다시 만들지 않아도 모델 요소의 추가, 분할, 제거 등을 간편하게 처리할 수 있습니다. 프로덕션 아티스트가 FBX 모델을 변경하면, 프리팹 배리언트가 변경된 사항에 따라 업데이트되도록 만들 수 있습니다. 또한 프리팹 배리언트를 사용하면 아티스트가 프로젝트 자체를 건드리지 않고도 아트가 변경되는 내용을 살펴보며 작업할 수 있습니다.

동시에 네스티드 프리팹을 사용하여 보다 원활하게 씬에 대한 협업을 진행하는 방안도 고려해 볼 만합니다.



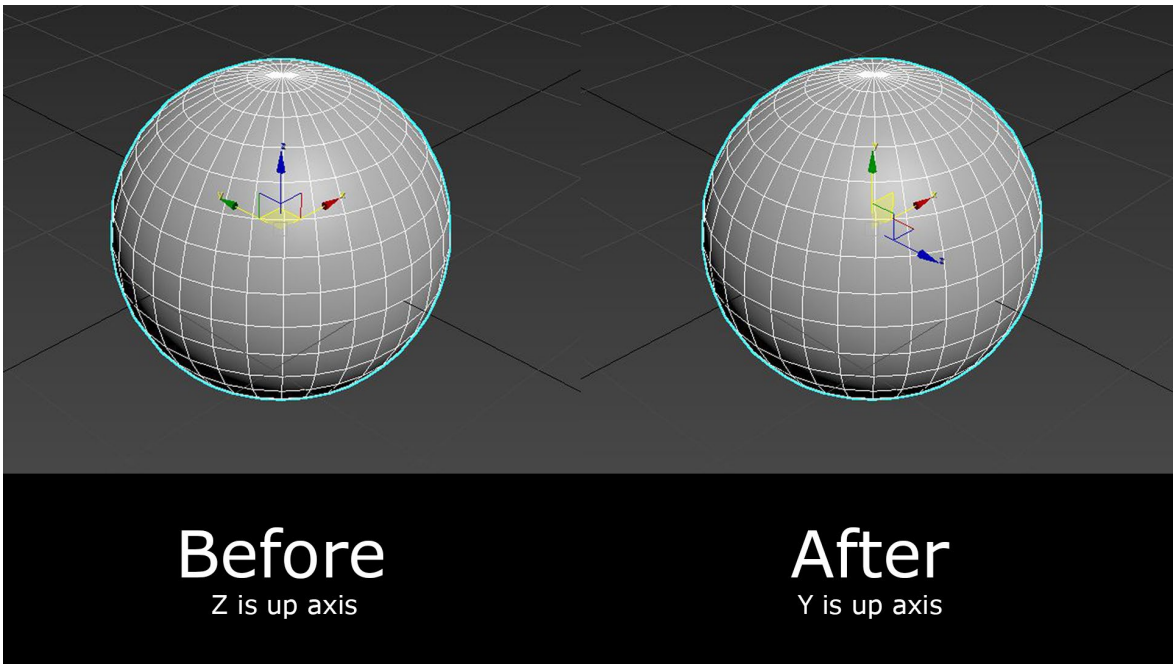
공동 작업자가 많은 제작 환경에서 사용할 수 있는 권장 프리팹 배리언트 워크플로

XR 또는 모바일 개발을 위한 에셋 고려 사항

XR 또는 모바일 게임의 아트 에셋 제작 또는 소싱 프로세스 초반에 내리는 결정에 따라 개발 사이클의 후반에 시간을 절약할 수 있습니다. 작은 프랍부터 캐릭터까지, 아래의 팁에 따라 아트 에셋을 효율적으로 제작하세요.

- 우선 제작하려는 콘텐츠에 대한 명확한 컨셉을 정하세요. 컨셉은 간단한 손 그림으로 작성해도 되며, 특히 초기 단계에는 그런 방법을 통해 상호 작용이나 비주얼을 빠르게 계획할 수 있습니다.
    - 대규모 스튜디오에는 아이디어를 구현하는 컨셉 아티스트가 있기도 하며, 개인이나 소규모 팀에서는 AI를 사용해 아이디어를 표현하기도 합니다.
- 3D 모델링 소프트웨어(Blender, Autodesk Maya, 3ds Max 등)를 사용하여 모델을 제작할 수 있습니다. 성능을 고려해 폴리 카운트(폴리곤 수)를 염두에 두어야 하며, 모바일 AR의 경우에는 더욱 그렇습니다. 처음에는 하이폴리곤 모델을 제작한 다음 나중에 최적화하면 됩니다.

— 3D 모델링 소프트웨어에서 Unity와 방향이 일치하도록 모델의 피벗 포인트를 설정하세요. 아래는 3ds Max에서 Z축을 위쪽 방향 축으로 사용하는 피벗 포인트를 설정하는 방법을 알려주는 예시입니다.



위쪽 방향 축 조정

— Unity에서는 다음 두 단계를 통해 모델의 방향을 수정할 수도 있습니다.

a. 계층 구조에서 빈 게임 오브젝트를 만듭니다.

b. 이 게임 오브젝트 내에 모델을 중첩하고 모든 축의 위치를 0,0,0으로 설정하여 정중앙에 피벗 포인트를 둡니다.

이 방법을 사용하면 모델의 모든 축이 0에서 중립적으로 회전하는 부모 게임 오브젝트에 고정됩니다. Z축이 정면을 향하도록 모델의 방향이 올바르게 설정되며 모든 축의 스케일도 1로 균일하게 유지됩니다. 또한 임포터에서 Bake Axis Conversion 옵션을 사용해 볼 수도 있습니다.

— 텍스처를 적용하면 모델에 사실적이거나 세련된 느낌을 줍니다. 디테일이 돋보이는 텍스처에는 UV 매핑을 사용하는 것이 좋습니다. 게임의 심미적인 측면에 따라 PBR 텍스처가 필요할 수도 있습니다.

— Unity의 PBR(물리 기반) 셰이더에는 일반적으로 두 가지 주요 컴포넌트가 포함됩니다. 메탈릭 거칠기 워크플로와 스페큘러 광택도 워크플로에서는 각각 다른 측면의 머티리얼 프로퍼티를 처리합니다. 그렇게 하면 더욱 생생하고 역동적인 시각적 형상을 구현하여 Unity 기반의 게임이나 시뮬레이션에서 3D 씬과 오브젝트의 사실감을 향상할 수 있습니다.

트림 시트를 사용하여 텍스처가 있는 넓은 영역을 효과적으로 채우기

환경에 있는 각종 메시에 트림 시트를 사용하세요.

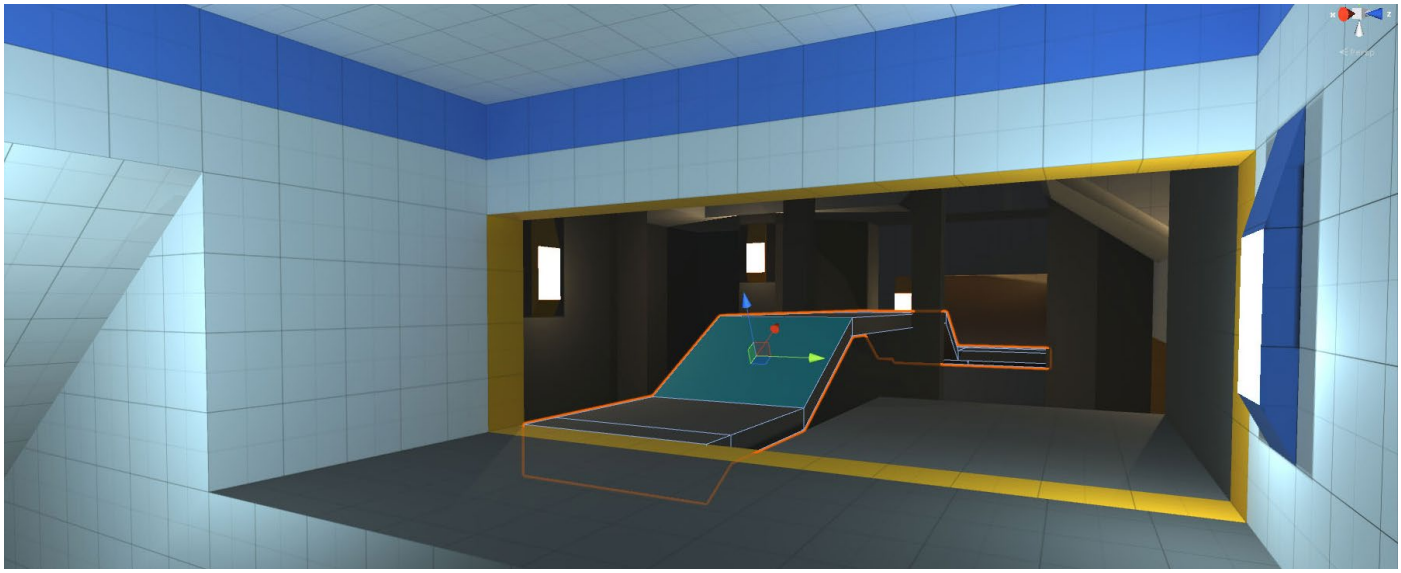
트림 시트는 3D 모델링과 게임 개발에 사용되는 텍스처링 툴입니다. 몰딩, 가장자리, 테두리 등의 다양한 트림 패턴과 디테일이 포함된 단일 텍스처로 구성됩니다.

트림 시트는 필요한 개별 텍스처의 수를 줄이면서도 시각적으로 복잡하고 다양한 디자인을 모델에 구현할 수 있으므로 게임 성능을 최적화하는 데 사용할 수 있습니다.

AR 및 VR 작업 시 3D 모델의 익스포트 및 스케일링

선호하는 DCC에서 또는 패키지 관리자에 포함된 ProBuilder를 사용하여 Unity에서 직접 3D 메시를 프로토타이핑 및 모델링할 수 있습니다. 레벨의 목업을 만들고, 오브젝트의 스케일을 조정하고, Unity에서 어떻게 보이는지 테스트하고, 스케일이 올바르면 익스포트한 다음 추가로 수정한 후 다시 Unity로 가져올 수 있습니다.

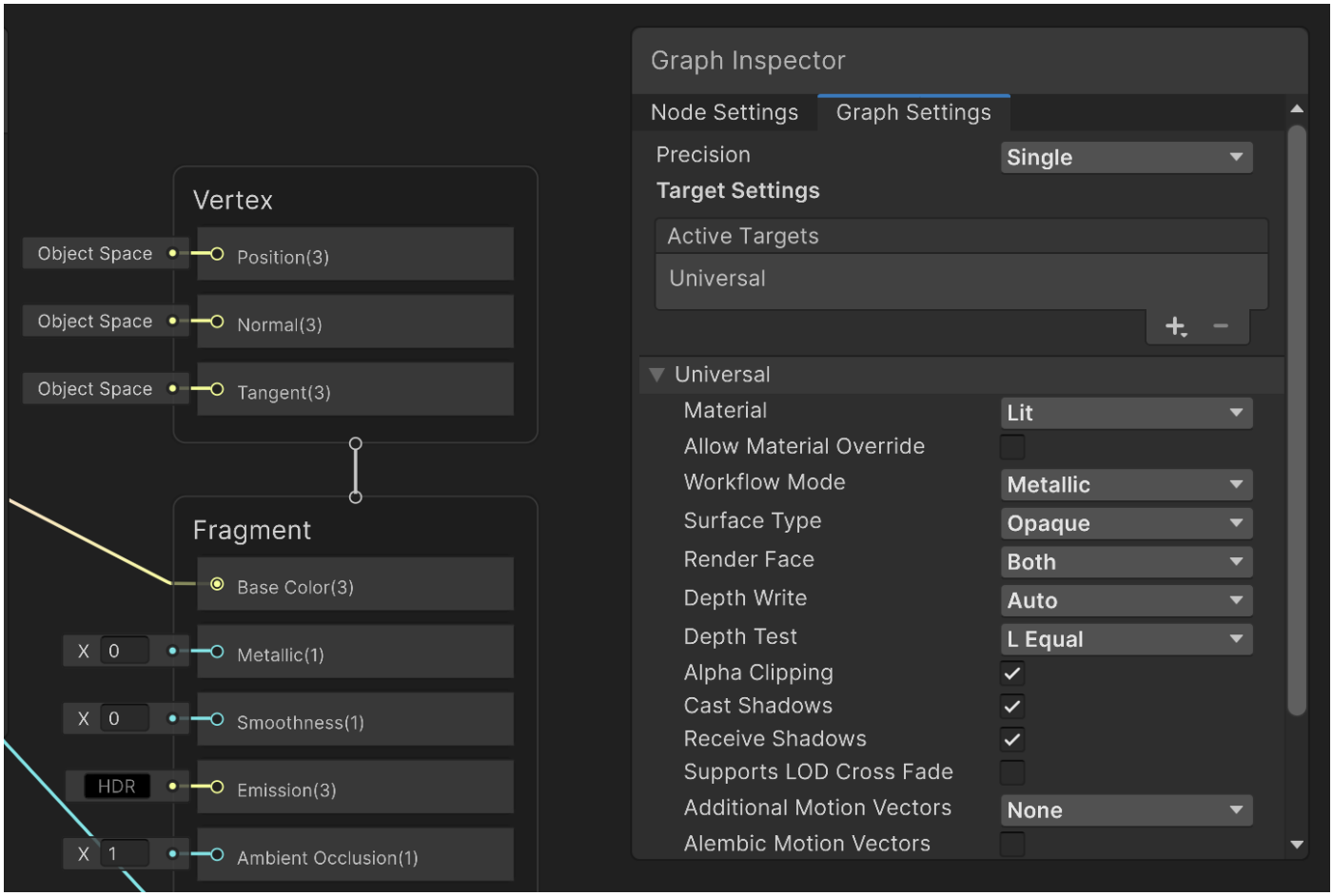
Unity의 측정 스케일은 1유닛은 정확히 1미터에 해당하도록 직관적으로 설계되어 있습니다. 임포트한 에셋의 스케일을 측정하려면 씬에 큐브 게임 오브젝트를 추가합니다. 1m x 1m x 1m 크기의 이 큐브는 에셋의 스케일을 평가할 때 즉각적이고 정확한 기준이 됩니다. 환경과 프랍을 제작할 때 핵심은 미관과 성능이 균형을 이루도록 하고, 부드러우면서도 흥미로운 AR/VR 경험을 구현하는 것입니다.



ProBuilder로 레벨을 프로토타이핑한 후, FBX Exporter를 사용해 3D 모델링 소프트웨어로 익스포트하여 미세 조정할 수 있습니다.

URP 및 HDRP 프로젝트용 Shader Graph

Shader Graph로 릿 또는 언릿 셰이더를 만들면 에셋을 URP 및 HDRP 프로젝트 양쪽에서 더 간편하게 사용할 수 있게 됩니다. Shader Graph는 마스터 노드에서 렌더 파이프라인을 정의하면서 로직은 그대로 유지할 수 있기 때문에 양쪽 파이프라인에서 작동하는 셰이더를 더 쉽게 만들 수 있습니다.



마스터 스택

UI 툴킷

비주얼 레퍼런스를 사용한 인터페이스 디자인

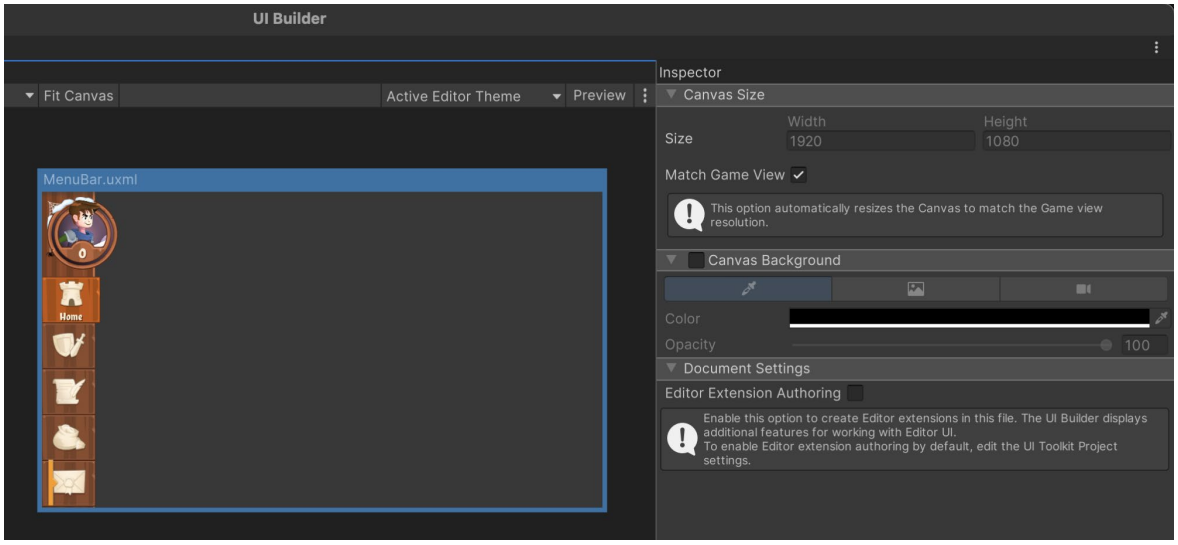
캔버스 배경을 사용하면 특정 컬러나 배경 이미지 위에서 요소의 스타일을 시각화할 수 있습니다. 계층 구조 창에서 UXML 파일을 선택하고 최종 UI 인터페이스와 비슷한 캔버스 배경을 선택하면 컨텍스트에 맞는 스타일 변화를 확인할 수 있습니다.

캔버스 배경에 사용할 수 있는 몇 가지 옵션은 다음과 같습니다.

— **Background Color**: 게임 환경의 특정 음영이나 색조를 나타냅니다.

— **Image**: 스프라이트나 텍스처를 배경으로 선택할 수 있으며, 목업 화면이나 참조 아트를 모사할 때 유용합니다.

— **Camera**: 배경에 현재 게임플레이가 표시됩니다. 실제 게임의 컨텍스트에서 UI를 볼 수 있습니다.



UXML 문서의 캔버스: Color 및 Image 옵션을 사용하여 외관을 조정합니다.

Photoshop 파일 작업 시 PSD Importer로 더 빠르게 반복 수정

사용자가 다중 레이어 PSD 파일을 저장할 때마다 Unity는 프로젝트에 해당 파일을 임포트하는 데 사용된 PSD Importer를 통해 파일에 포함된 스프라이트를 자동으로 새로 고침합니다. 따라서 플레이스홀더를 빠르게 만들고 게임(Game) 뷰에서 변경 사항을 확인하면서 이를 반복 수정할 수 있습니다. 이렇게 하면 파일을 바꾸거나 개발자 팀원의 지원을 받지 않고도 컨텍스트 내에서 작업물을 확인하며 시간을 단축하고 품질을 높일 수 있습니다.

게임의 이모티콘 사용

리치 텍스트 태그를 통해 텍스트에 이모티콘과 같은 스프라이트를 추가할 수 있습니다. 이모티콘을 사용하려면 그레디언트 에셋과 유사한 스프라이트 에셋을 사용해야 합니다.

여러 스프라이트를 임포트할 때 하나의 아틀라스로 패킹하면 드로우 콜을 줄일 수 있습니다. 스프라이트 아틀라스의 해상도는 반드시 타겟 플랫폼에 적합해야 합니다.

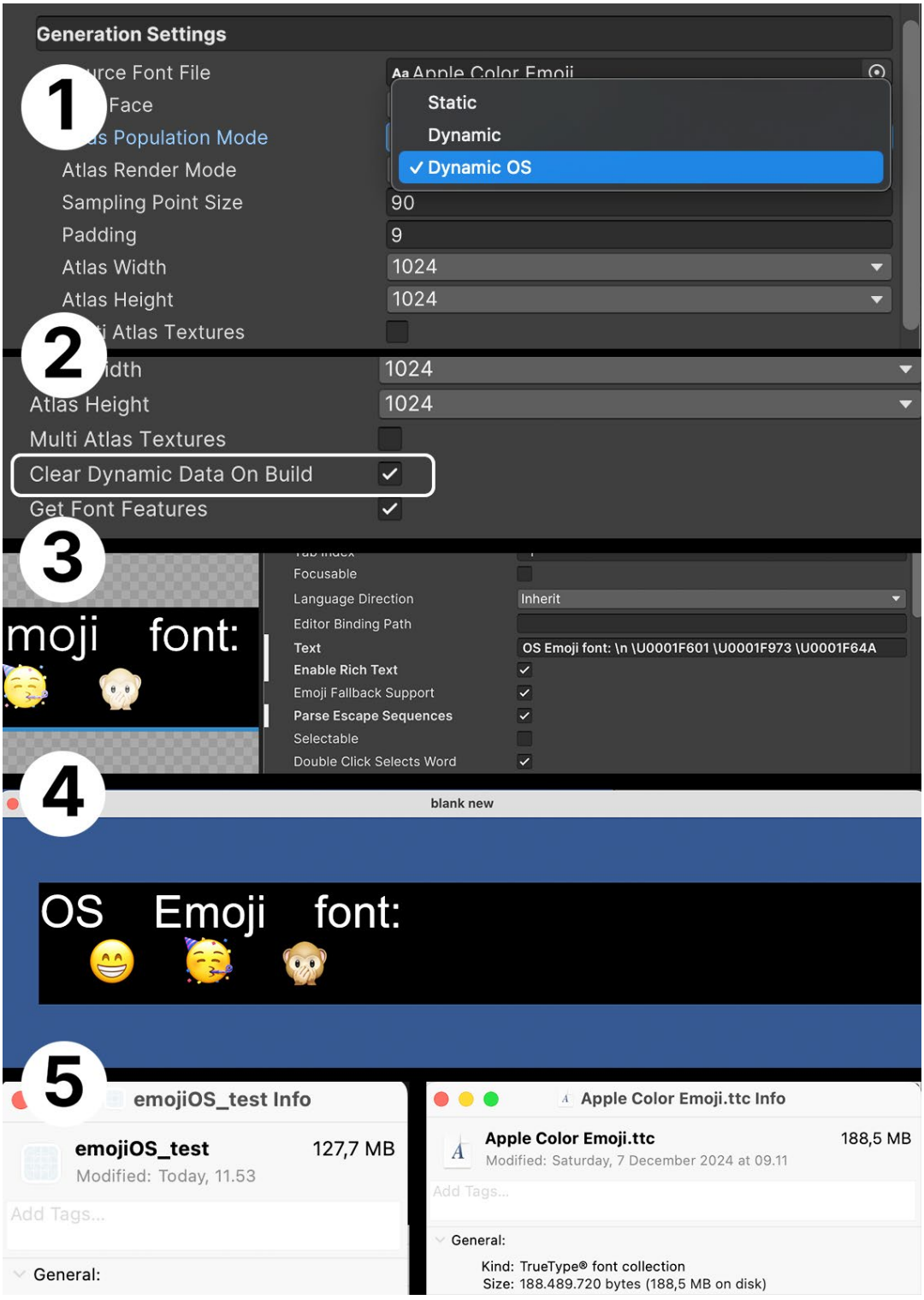
스프라이트를 임포트하고 Assets/Resources에 스프라이트 에셋을 만든 다음 필요에 따라 각 글리프의 정보를 수정합니다.

기기 OS의 빌트인 이모티콘 사용

iOS나 Android와 같은 특정 런타임 플랫폼을 타게팅한다면 프로젝트에 소스 폰트를 포함하는 대신 시스템의 빌트인 이모티콘 폰트를 사용할 수 있습니다. 그러면 메모리를 절감할 수 있으며 대용량 이모티콘 컬렉션을 애플리케이션과 함께 패킹하지 않아도 됩니다.

프로젝트에서 OS 이모티콘을 사용하려면 다음 단계를 따르세요.

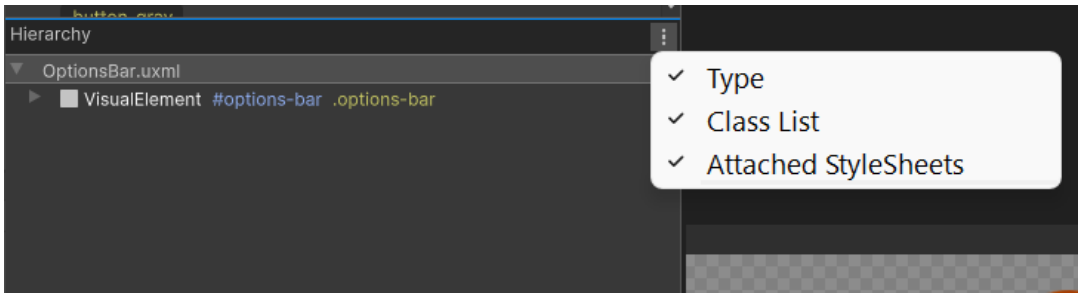
1. 타겟 시스템이 사용하는 폰트에서 폰트 에셋을 생성합니다. iOS의 폰트는 Apple Emoji고(이 예시에서 사용), Android의 폰트는 Noto Color Emoji입니다(현재 COLRv0만 지원됨). 폰트 에셋의 유형을 **Color**로 설정하고 Atlas Population Mode를 **Dynamic OS**로 설정해야 합니다. 그러면 에셋에 소스 폰트를 포함할 필요가 없어 공간이 더 확보됩니다.
2. 폰트 에셋에서 **Clean Dynamic Data On Build**를 선택합니다.
3. UI 빌더에서 **Parse Escape Sequences**를 활성화한 다음, macOS 또는 Windows의 이모티콘 키보드를 사용하거나 UTF 포맷으로 원하는 이모티콘을 입력합니다. 예를 들어 스마일 이모티콘은 \U0001F601입니다. 폰트 에셋의 Character Table에서 각 이모티콘의 UTF를 확인할 수 있습니다.
4. OS 폰트에 따라 macOS에서 실행 중인 빌드에 이모티콘이 표시됩니다.
5. 테스트에서 보면 스탠드얼론 이모티콘 폰트에 비해 **빌드 크기**가 작은 것을 확인할 수 있습니다. 따라서 프로젝트에 이모티콘 폰트가 포함되지 않았음에도 적합한 이모티콘을 렌더링하는 데 사용되고 있음을 알 수 있습니다.



UI 빌더에서 시각 요소의 추가 정보 표시

계층 구조 헤더에서 세로 줄임표( )를 클릭하여 UI 요소를 추가로 시각화할 수 있습니다.

계층 구조 패널에서 요소 유형 옆에 추가 정보가 표시됩니다. 요소를 선택하면 **#options-bar** 이름 선택자와 **.options-bar** 스타일 클래스 선택자가 표시됩니다.



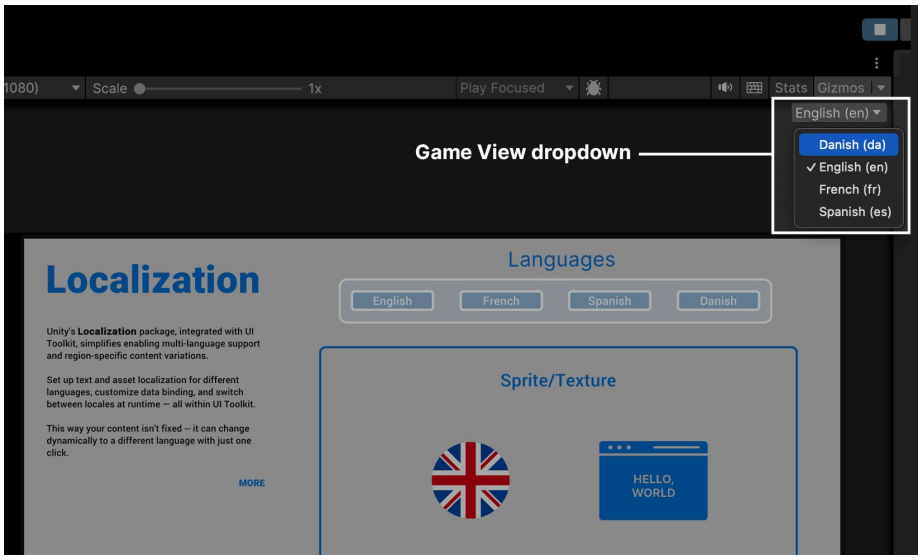
계층 구조에서 다양한 선택자를 필터링할 수 있습니다.

.unity-로 시작하는 선택자도 볼 수 있으며, 이러한 선택자는 모든 요소에 적용되는 기본 스타일을 나타냅니다. 정의된 선택자가 있으면 이 값이 오버라이드됩니다.

초반에 현지화를 통합하여 더 많은 시장 도달

UI 툴킷에 Localization 패키지를 통합하여 현지화 프로세스를 간소화할 수 있습니다. 이를 통해 개발자는 플레이어가 어디에 있든 상관없이 각자에게 맞는 지역별 콘텐츠를 제공할 수 있습니다.

게임 뷰 로케일 드롭다운을 사용하여 여러 언어로 UI를 미리 보며 각 로케일에서 요소들이 올바르게 표시되는지 확인합니다.



Game View 로케일 드롭다운 목록을 사용하여 현지화 옵션을 미리 봅니다.

그레디언트를 사용하여 인터페이스 전반에 스타일 추가

UI 툴킷에서 <gradient> 태그를 사용하여 그레디언트를 적용할 수 있습니다. **리치 텍스트**가 활성화된 상태면 UI 빌더 또는 게임 뷰에서 변경 사항이 적용된 것을 볼 수 있습니다.

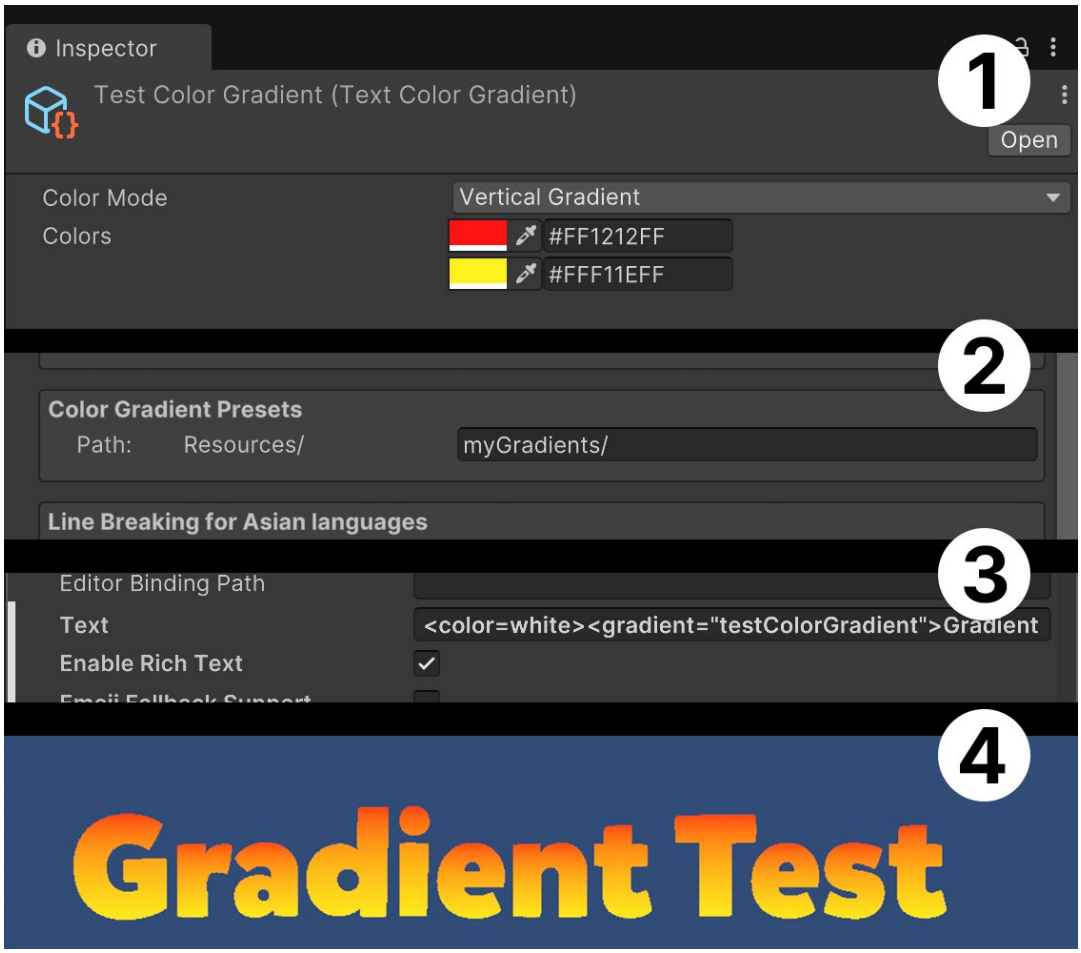
1. **Create > Text Core > Gradient Color**를 통해 그레디언트 컬러 에셋을 생성합니다. 이 파일은 **Assets/Resources** 나 Resources 아래의 하위 폴더에 넣어야 합니다.

2. Panel Settings에서 참조할 Text Settings 에셋을 생성합니다. 해당 에셋에서 Color Gradient Presets를 찾고 에셋이 있는 Resources 아래의 폴더나 하위 폴더를 지정합니다.

3. UI 빌더 내에서 <color=white><gradient="testColorGradient">Gradient Test</gradient></color> 리치 텍스트 태그를 추가합니다.

   이 컬러 태그는 그레디언트가 의도한 대로 보이도록 폰트 컬러를 흰색으로 복원하며, 이때 참조되는 그레디언트는 1단계에서 생성한 에셋 이름과 일치해야 합니다. **Rich Text**가 활성화되어야 합니다.

4. UI 빌더 또는 게임 뷰에서 변경 사항이 적용된 것을 볼 수 있습니다.



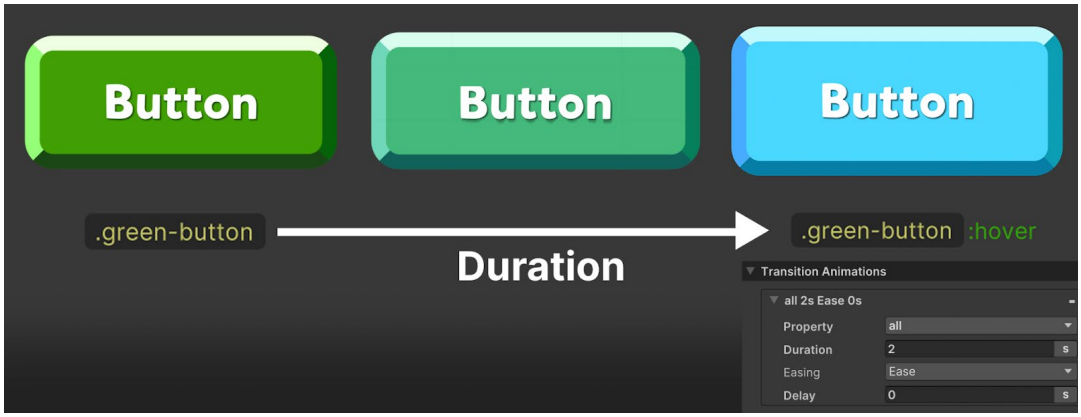
UI 툴킷에서 <gradient> 태그 사용

USS 전환을 사용하여 UI 애니메이션화

시각 요소의 경우, 유사 클래스(:active, :inactive, :hover 등)가 자체 선택자를 가질 수 있으므로 추가 코딩 없이 애니메이션을 적용할 수 있습니다. 유사 클래스가 스타일 변경을 트리거할 때마다, 정의된 전환이 자동으로 해당 변경 사항에 애니메이션을 적용합니다.

예를 들면 다음과 같습니다. 마우스 커서를 올리거나(:hover) 클릭했을 때(:active) 버튼이 커지거나 작아질 수 있고, 사용자 상호 작용이나 그 밖의 이벤트로 인해 요소가 페이드아웃되거나 보이지 않게 될 수 있습니다.

이러한 상태 변화는 보통 사용자 동작으로 트리거되지만, 유사 클래스 :enabled 및 :disabled를 임의로 변경하여 해당 유사 클래스에 연결된 USS 애니메이션을 수동으로 트리거할 수도 있습니다. 즉, 코드에서 원하는 대로 애니메이션을 트리거할 수 있는 것입니다.

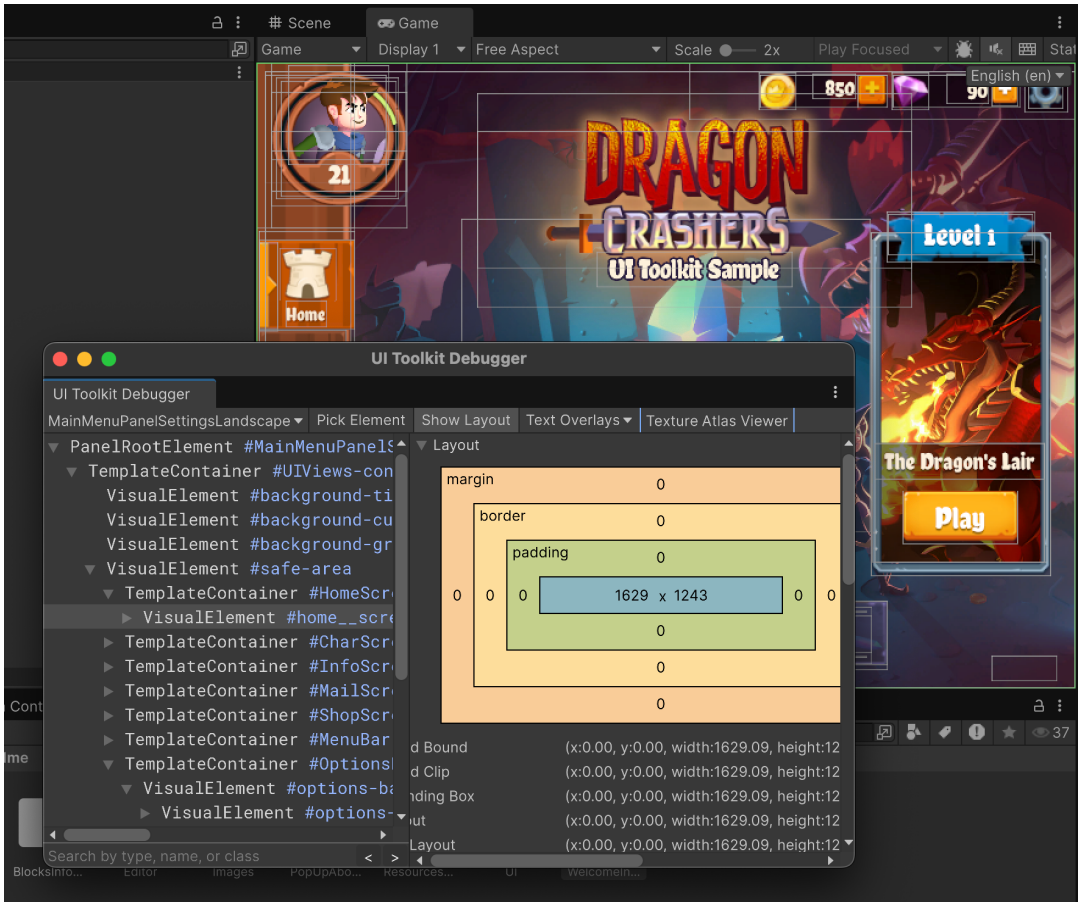


USS 전환 애니메이션 예시

에디터에서 최종 스타일의 바운딩 박스 시각화

바운딩 박스를 사용하여 레이아웃 문제와 얼라인먼트 문제를 식별하고 에디터에서 상호 작용을 통해 UI 구조를 디버깅할 수 있습니다.

UI 툴킷을 사용하여 최종 바운딩 박스를 시각화하려면 **Window** > **UI Toolkit** > **Debugger**로 이동하여 **UI Toolkit Debugger** 를 연 다음, **Pick Element** 툴을 사용하여 UI 요소를 선택하고 **Show Layout** 기능을 활성화합니다. 이 옵션은 게임 뷰에서 UI 바로 위에 바운딩 박스와 레이아웃 정보를 표시합니다.

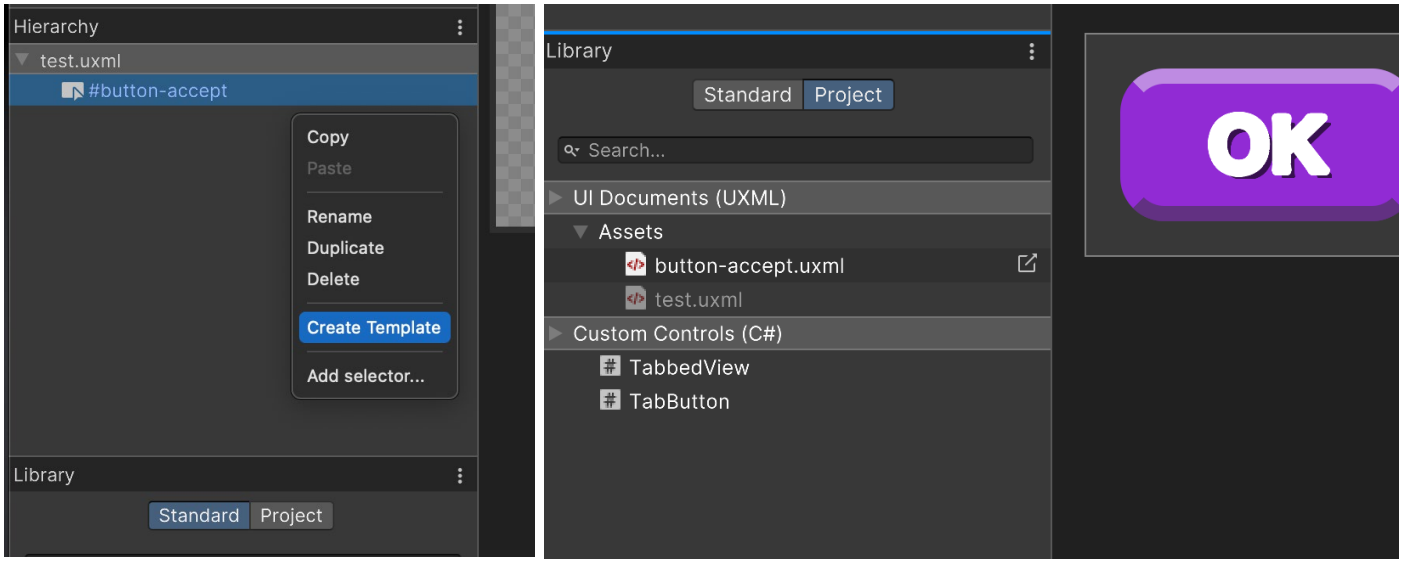


UI 툴킷 디버거에는 UI를 디버깅하는 데 도움이 되는 여러 유용한 기능이 있습니다.

UXML 파일을 템플릿으로 재사용하여 워크플로 속도 향상

UXML 파일은 프리팹처럼 사용할 수 있습니다. 예를 들어 프로젝트에 하나의 항목 아이콘과 카운트 수를 갖는 UXML 레이아웃이 있고, 인벤토리 내에서 여러 번 생성해야 한다고 가정해 보겠습니다.

UXML을 오른쪽 클릭하면 Create Template 옵션이 나타납니다. 이렇게 생성한 템플릿은 계층 구조 패널에 있는 시각적 요소에 추가하거나 코드를 통해 인스턴스화할 수 있습니다. 생성한 템플릿은 Library의 Project 탭에서 볼 수 있습니다.



템플릿은 재사용 가능한 UXML로, Library 패널의 Project 탭에서 볼 수 있습니다.

더 많은 자료

Unity 6에서 UI 툴킷을 사용해 확장 가능한 고성능 UI 만들기 전자책을 다운로드하여 다양한 기기에서 UI 툴킷으로 UI를 만드는 방법을 자세히 알아보세요.

전자책과 함께 샘플 프로젝트도 제공됩니다. Unity 에셋 스토어에서 UI 툴킷 샘플 – 드래곤 크래셔를 찾아보세요. UI 툴킷 샘플에서는 애플리케이션에서 UI 툴킷을 어떻게 활용할 수 있는지 보여 줍니다. 이 데모는 런타임에 Unity 6 UI 툴킷 워크플로를 사용하며, 미니 2D RPG 프로젝트 드래곤 크래셔의 일부를 통해 완전한 기능을 갖춘 인터페이스를 선보입니다.

Unity 에셋 스토어의 QuizU 프로젝트와 관련 문서를 확인하는 것도 잊지 마세요.

- UI 툴킷 샘플 프로젝트 QuizU
- QuizU: 게임 플로를 위한 상태 패턴
- QuizU: UI 툴킷에서 메뉴 화면 관리
- QuizU: 모델 뷰 프리젠터 패턴
- QuizU: UI 툴킷의 이벤트 처리
- QuizU: UI 툴킷 성능 팁

개발자 워크플로

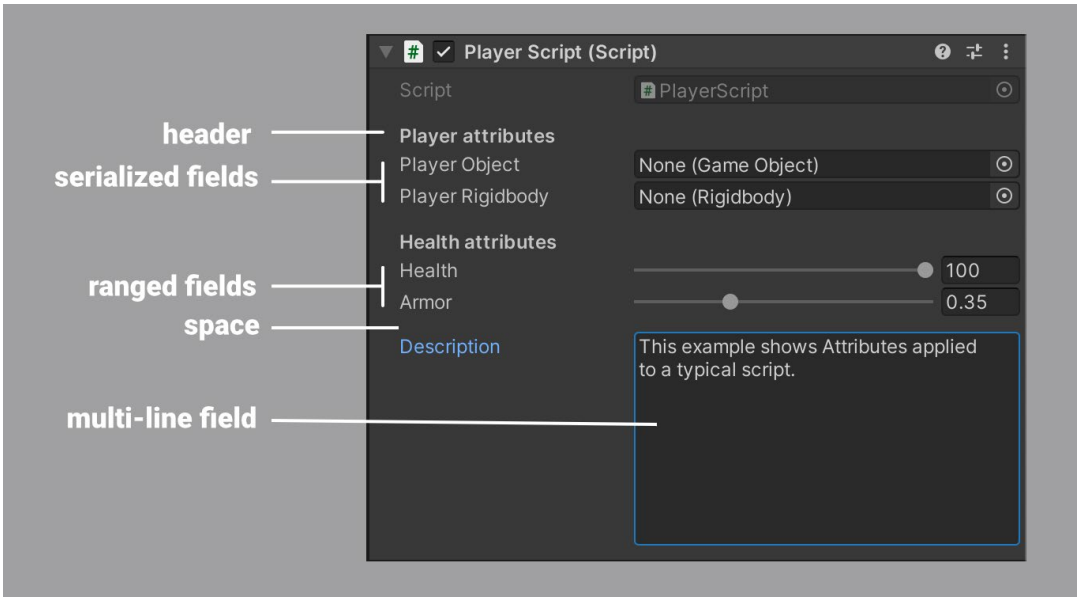
Awaitable 클래스

Unity 6에는 Unity에서 C# async/await 워크플로를 지원하도록 설계된 가볍고 메모리 할당이 없는 Awaitable 클래스가 도입되었습니다. 이 클래스는 코루틴의 고성능 대안으로 사용할 수 있습니다. 이 클래스를 사용하면 Unity의 프레임 기반 업데이트 사이클과 원활하게 통합되는 비동기 코드를 더 쉽게 작성할 수 있습니다.

```
using UnityEngine;
using System.Threading.Tasks;

public class LogWithDelay : MonoBehaviour
{
    private async void Start()
    {
        Debug.Log(“Message 1”);
        await Task.Delay(1000); // 1초 대기

        Debug.Log(“Message 2”);
        await Task.Delay(1000); // 다시 1초 대기
    }
}
```

속성을 사용하여 인스펙터 창 개선

Unity에는 클래스, 프로퍼티, 함수 위에 배치하여 특수한 동작(인스펙터에서 헤더, 간격, 범위 필드 만들기 등)을 나타낼 수 있는 다양한 속성이 있습니다.



인스펙터 필드에 영향을 주는 속성

C#에서는 속성 이름을 대괄호로 묶습니다. 스크립트에 추가할 수 있는 몇 가지 일반적인 속성은 다음과 같습니다.

속성	설명	예시
SerializeField	Unity가 비공개 필드를 직렬화하고 이를 인스펙터에서 볼 수 있게 합니다.	`[SerializeField]` `private GameObject m_myObject;`

Range	특정 범위로 제한된 float 또는 int 변수를 사용합니다. 필드는 인스펙터에 슬라이더로 표시됩니다.	`[Range(1,6)]`
		`public int IntegerRange;`
		
		`[Range(0.2f, 0.8f)]`
		`public float m_floatRange;`

HideInInspector	변수가 인스펙터에서 숨겨지고 직렬화됩니다.	`[HideInInspector]` `public Int p = 5;`

RequireComponent	자동으로 필요한 컴포넌트를 종속성으로 추가해 설정 오류를 방지합니다.
	
	참고: 컴포넌트가 게임 오브젝트에 추가될 때만 확인합니다.
		`// PlayerScript는 게임 오브젝트가 Rigidbody를 갖도록 요구`
		`[RequireComponent(typeof(Rigidbody))]`
		`public class PlayerScript: Monobehaviour`
		`{`
		`  private Rigidbody m_rBody;`
		
		`  void Start()`
		`  {`
		`  m_rBody =`
		`  GetComponent<Rigidbody>();`
		`  }`
		`}`

Tooltip	사용자가 인스펙터의 필드 위에 마우스 커서를 올려놓으면 툴팁을 표시합니다.	`public class PlayerScript: Monobehaviour`
		`{`
		`  [Tooltip("Health value between 0 and 100.")]`
		`  int m_health = 0;`
		`}`

Space	추가 텍스트 없이 필드 사이에 공백을 추가하여 필드 사이를 시각적으로 구분할 수 있습니다.	`[Space(10)] // 10픽셀 공백 추가됨` `int p = 5;`

Header	하여 인스펙터에서 변수를 구성할 때 편하도록 굵은 텍스트와 공백을 추가합니다. 그룹에 넣으려는 첫 번째 필드에만 이 속성을 추가하는 것이 좋습니다.	`public class PlayerScript: Monobehaviour`
		`{`
		`  [Header("Health Settings")]`
		`  private int m_health = 0;`
		`  private int m_maxHealth = 100;`
		
		`  [Header("Shield Settings")]`
		`  private int m_shield = 0;`
		`  private int m_maxShield = 0;`
		`}`

Multiline	여러 줄의 텍스트 필드로 문자열을 편집할 수 있습니다. 옵션으로 int를 전달하여 줄 수를 지정합니다.
	
	팁: 자신이나 다른 사용자에게 알리는 메모가 있는 스크립트에 주석을 달 때 사용하면 좋습니다.
		`[Multiline]`
		`public string textToEdit;`
		
		`[Multiline(20)]`
		`public string m_moreTextToEdit;`

SelectionBase	자식에 메시가 있을 수 있는 빈 게임 오브젝트를 선택할 때 유용합니다. 원본 오브젝트의 컴포넌트에 속성을 추가합니다. 에디터에서 오브젝트를 선택할 때 [SelectionBase] 속성이 포함된 게임 오브젝트는 자식이 아닌 게임 오브젝트 자체가 선택됩니다.	`// 기본 게임 오브젝트에 아래 추가`
		`[SelectionBase]`
		`public class PlayerScript: Monobehaviour`
		`{`
		
		`}`

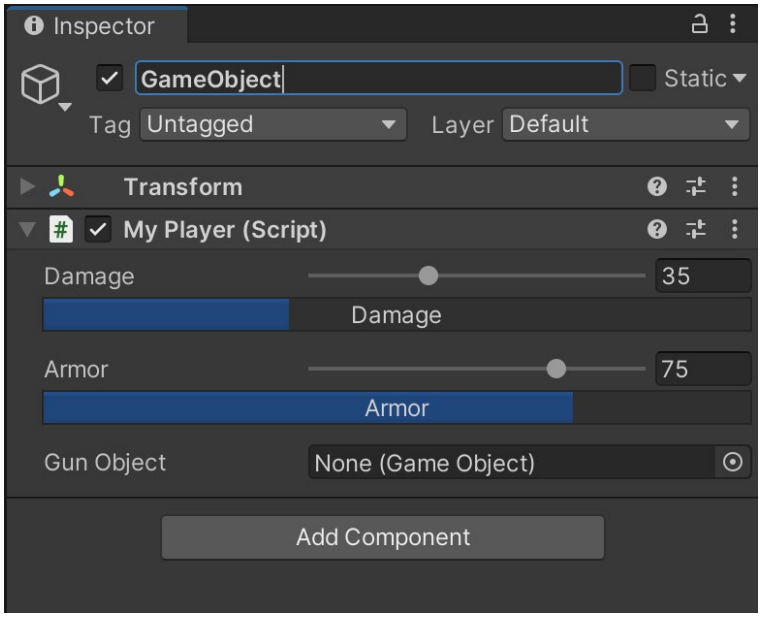
ColorUsage	[ColorUsage] 속성을 사용하면 컬러 필드에서 선택 가능한 컬러를 제어할 수 있습니다. 파라미터에 따라 HDR을 활성화하거나 알파 채널을 비활성화할 수 있습니다.	`public ColorUsageAttribute(bool showAlpha, bool hdr, float minBrightness, float maxBrightness, float minExposureValue, float maxExposureValue);`

RunOnce	프로젝트가 시작할 때 함수가 자동으로 한 번만 실행되어야 한다면 이 정적 메서드를 [RuntimeInitialization] 속성과 함께 사용하면 됩니다. QuizU 샘플의 부트로더처럼 일회성 프로젝트 설정 로직을 실행하는 데 사용하세요.	`public RuntimeInitializeOnLoadMethodAttribute(RuntimeInitializeLoadType loadType);`

위의 예시는 사용 가능한 수많은 속성 중 일부입니다. 변수의 값을 유지하면서 이름만 변경하거나, 빈 게임 오브젝트를 만들지 않고 로직을 호출하거나, 자체 PropertyAttribute를 만들어 스크립트 변수에 커스텀 속성을 정의할 수도 있습니다. 스크립팅 API에서 모든 속성 목록을 확인해 보세요.

커스텀 창 및 인스펙터 제작

확장 가능한 에디터는 Unity의 강점 중 하나입니다. 커스텀 창과 커스텀 인스펙터 같은 에디터 UI를 만들려면 **UI Toolkit** 패키지를 사용할 것을 권장합니다.



커스텀 에디터는 MyPlayer 스크립트가 인스펙터에 표시되는 방식을 수정합니다.

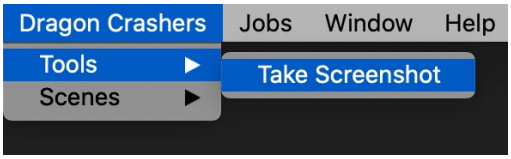
UI(사용자 인터페이스) 만들기를 참조하여 UI 툴킷이나 IMGUI로 커스텀 에디터 스크립트를 구현하는 방법을 자세히 알아보세요. **UI 툴킷**에 대한 간단한 소개는 에디터 스크립팅 시작하기 튜토리얼을 참조하세요.

커스텀 메뉴 제작

Unity에서는 **MenuItem** 속성으로 에디터 메뉴와 메뉴 항목을 간단하게 커스터마이즈할 수 있습니다. MenuItem 속성은 스크립트의 어느 정적 메서드에나 적용할 수 있습니다.

프로젝트에서 자주 사용하는 함수가 있다면 메뉴 항목으로 구성할 수 있습니다. 그러면 하나의 PropertyAttribute 조정값으로 기본 사용자 인터페이스를 구축할 수 있습니다.

```
using UnityEditor;
using UnityEngine;
using System;
using System.IO;

public class ScreenshotTaker
{
    [MenuItem("Dragon Crashers/Tools/Take Screenshot")]
    public static void TakeScreenshot()
    {
        if (!Directory.Exists("Screenshots"))
            Directory.CreateDirectory("Screenshots");

        ScreenCapture.CaptureScreenshot(string.Format("Screenshots/{0}.png",
            DateTime.Now.ToString("yyyy-MM-dd HH.mm.ss")));
    }
}
```



정적 메서드(스크린샷 찍기)를 연결하는 간단한 인터페이스를 만드는 MenuItem 속성

플레이 모드 진입 시간 단축

플레이 모드에 진입하면 빌드와 동일하게 프로젝트가 실행됩니다. 플레이 모드일 때 에디터에서 변경한 사항은 플레이 모드를 종료하면 초기화됩니다.

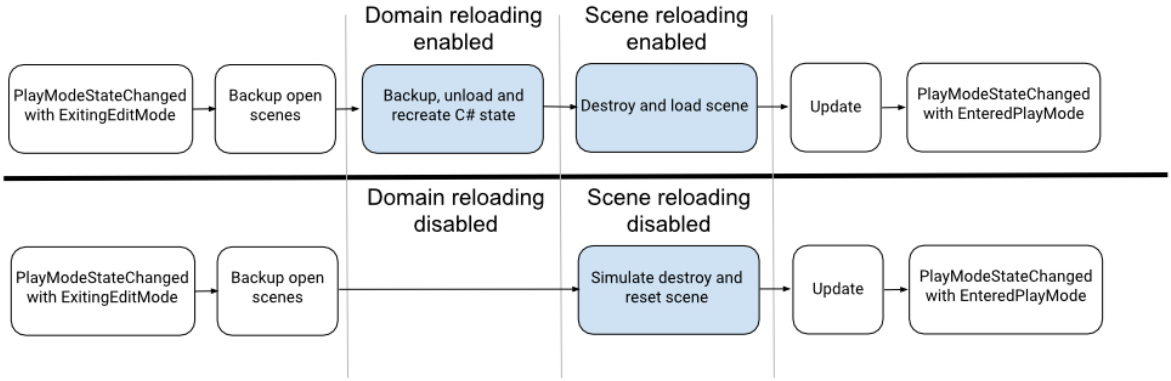
플레이 모드에 진입할 때마다 Unity는 다음과 같은 두 가지 중요한 작업을 처리합니다.

- **도메인 리로드:** Unity가 스크립팅 상태를 백업, 언로드, 재작성합니다.
- **씬 리로드:** Unity가 씬을 없애고 다시 로드합니다.

스크립트와 씬이 복잡할수록 이 두 가지 동작에 더 많은 시간이 소요됩니다.

스크립트를 더 변경할 계획이 없는 경우 **Enter Play Mode Settings(Edit > Project Settings > Editor)**를 활용하여 컴파일 시간을 단축하는 것이 좋습니다. Unity는 도메인 리로드와 씬 리로드 중 하나 또는 두 가지 모두를 비활성화하는 옵션을 제공합니다. 이를 통해 플레이 모드 진입 및 종료 속도를 높일 수 있습니다.

다만 스크립트를 추가로 변경할 계획이라면 도메인 리로드를 재활성화해야 합니다. 마찬가지로 씬 계층 구조를 수정할 때는 씬 리로드를 재활성화해야 합니다. 그렇지 않으면 예기치 않은 동작이 발생할 수 있습니다.



도메인 리로드와 씬 리로드 설정 비활성이 미치는 결과

기본 스크립트 템플릿 커스터마이즈

새 스크립트를 작성할 때 매번 동일한 변경 사항을 적용하시나요? 무의식적으로 네임스페이스를 추가하거나 업데이트 이벤트 함수를 삭제하시나요? 원하는 시작점에 대한 스크립트 템플릿을 설정하면 코드를 추가 또는 제거하지 않고도 팀 전체의 일관성을 유지할 수 있습니다.

새 스크립트나 셰이더가 생성될 때마다 Unity는 **%EDITOR_PATH%\Data\Resources\ScriptTemplates**에 저장된 템플릿을 사용합니다.

- Windows: C:\Program Files\Unity\Editor\Data\Resources\ScriptTemplates
- Mac: /Applications/Hub/Editor/[version]/Unity/Unity.app/Contents/Resources/ScriptTemplates

셰이더, 기타 동작 스크립트, 어셈블리 정의를 위한 템플릿도 있습니다.

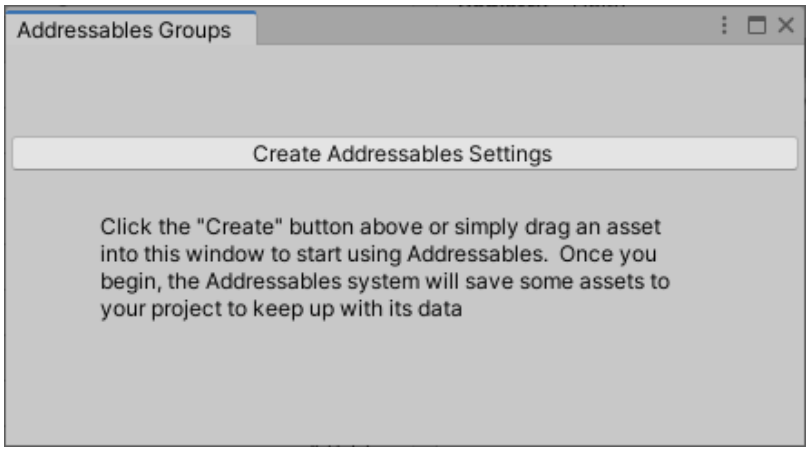
특정 프로젝트 전용 스크립트 템플릿을 정하려면 Assets/ScriptTemplates 폴더를 만든 다음, 해당 폴더에 스크립트 템플릿을 복사하여 기본 템플릿을 오버라이드하면 됩니다.

모든 프로젝트를 대상으로 하는 기본 스크립트 템플릿도 직접 수정이 가능하지만, 수정 전에 원본을 백업해야 합니다.

어드레서블을 사용하여 플레이어에게 온디맨드로 콘텐츠 배포

어드레서블과 에셋 번들은 게임을 논리적 블록으로 구성하는 강력한 툴입니다. 이렇게 구성한 블록은 개별적으로 익스포트하여 필요할 때 메인 실행 파일에 추가할 수 있습니다. 어드레서블을 사용하여 에셋을

로드 및 언로드하고 에셋 번들을 구성, 제작, 로드한 후 플레이어에게 온디맨드로 배포할 수 있습니다. 어드레서블 시스템은 에셋 번들을 기반으로 만들어졌기 때문에 종속성 확인과 번들 로드가 자동으로 처리됩니다.



Unity 프로젝트에서 어드레서블 시스템을 초기화하기 전

어드레서블을 처음 사용한다면 Unity 기술 자료의 시작하기 페이지를 참조하세요.

효과적인 에셋 관리 팁:

- 처음부터 어드레서블을 사용하고 모든 신규 에셋을 어드레서블로 등록합니다.
- 에셋을 유형별로 정리하기보다 함께 로드되고 사용되는 빈도를 기준으로 그룹화하는 것을 목표로 합니다. 이렇게 하면 런타임 메모리 사용량을 개선하고 부트 시간을 단축하여 결과적으로 게임 리텐션을 향상할 수 있습니다.
- 번들을 가능한 한 작게 만듭니다. 그러면 종속성 체인을 단축하고 런타임 메모리 사용량을 줄일 수 있습니다.

자세한 내용은 Unity에서 어드레서블을 사용하여 효과적으로 에셋 관리하기 문서를 참조하세요.

프리 프로세서 지시문을 사용하여 조건부로 컴파일되는 코드 생성

플랫폼별 컴파일 기능을 사용하면 타겟 플랫폼, Unity 버전, 스크립팅 백엔드에 따라 코드를 조건부로 컴파일하고 실행할 수 있습니다.

이는 크로스 플랫폼 코드를 작성하거나, 기기별 행동에 최적화하거나, 버전별 API를 관리할 때 유용합니다.

에디터에서 테스트할 때 고유의 커스텀 #define 지시문을 제공할 수도 있습니다. Player 설정의 **Other Settings** 패널을 열고 **Scripting Define Symbols**로 이동하세요.

Script Compilation의 Scripting Define Symbols

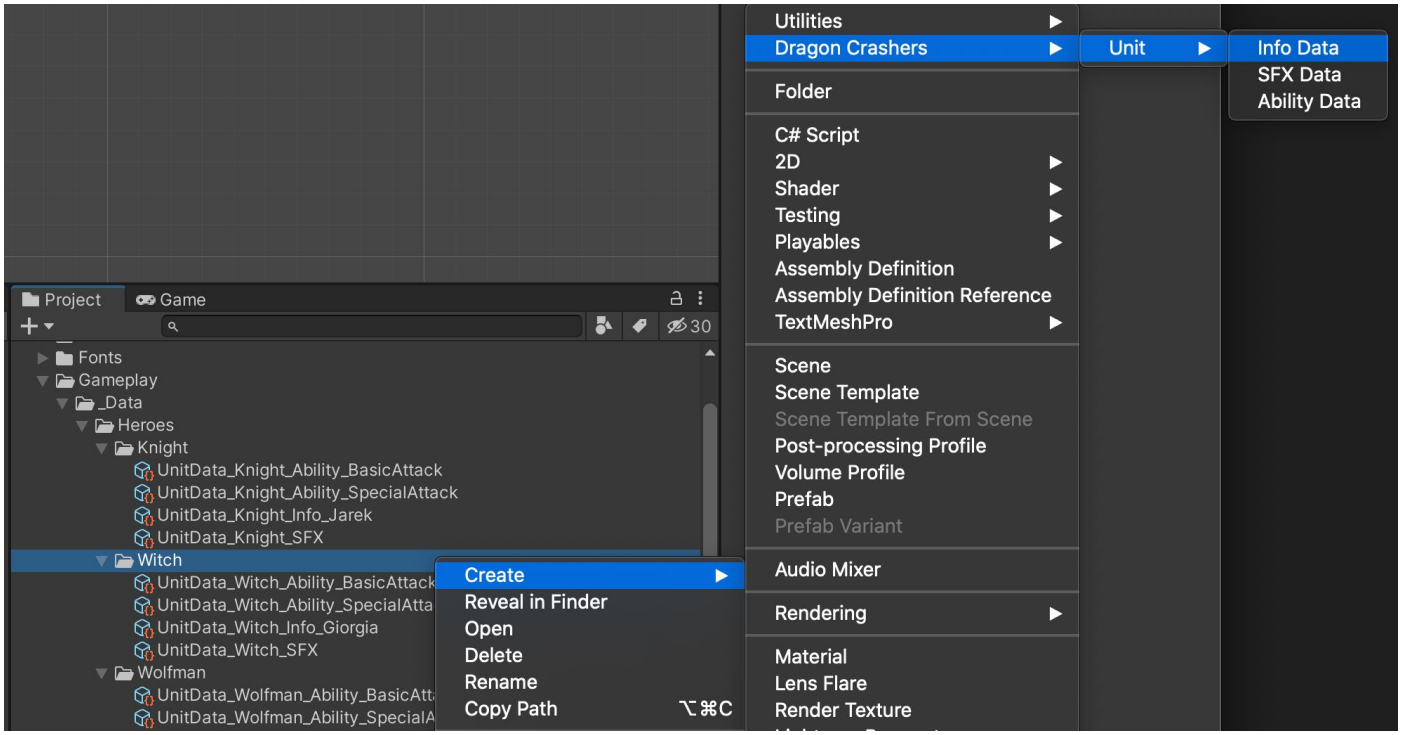
스크립터블 오브젝트를 사용하여 데이터를 로직에서 분리

스크립터블 오브젝트를 사용하면 데이터를 로직에서 분리하여 코드를 깔끔하게 작성할 수 있습니다. 의도치 않은 부작용을 발생시키지 않으면서 변경 사항을 쉽게 적용할 수 있어 테스트와 모듈식 구성이 용이해집니다. 코드를 수정하지 않고도 게임 데이터를 변경할 수 있어 아티스트, 디자이너 등 프로그래머가 아닌 사람들과 협업할 때도 유용합니다.

Dragon Crashers는 전형적인 사용 사례를 보여줍니다. **UnitInfoData** 클래스는 **ScriptableObject**에서 상속됩니다. 각 인스턴스에는 유닛의 이름, 스프라이트, 체력 설정이 포함되어 있습니다. 이 데이터는 게임플레이 중에 일정하게 유지되기 때문에 특히 스크립터블 오브젝트 내부에 저장하기 좋습니다.

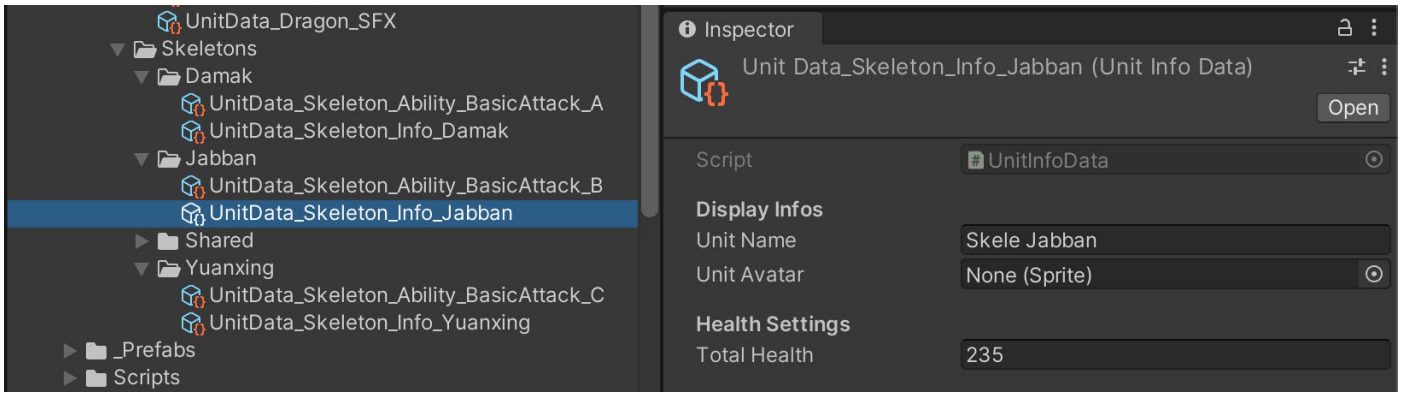
```
using UnityEngine;

namespace DragonCrashers
{
    [CreateAssetMenu(fileName = "Data_Unit_", menuName = "Dragon Crashers/Unit/Info Data", order = 1)]
    public class UnitInfoData : ScriptableObject
    {
        [Header("Display Infos")]
        public string unitName;
        public Sprite unitAvatar;

        [Header("Health Settings")]
        public int totalHealth;

    }
}
```

스크립터블 오브젝트는 데이터 컨테이너 오브젝트를 정의합니다.



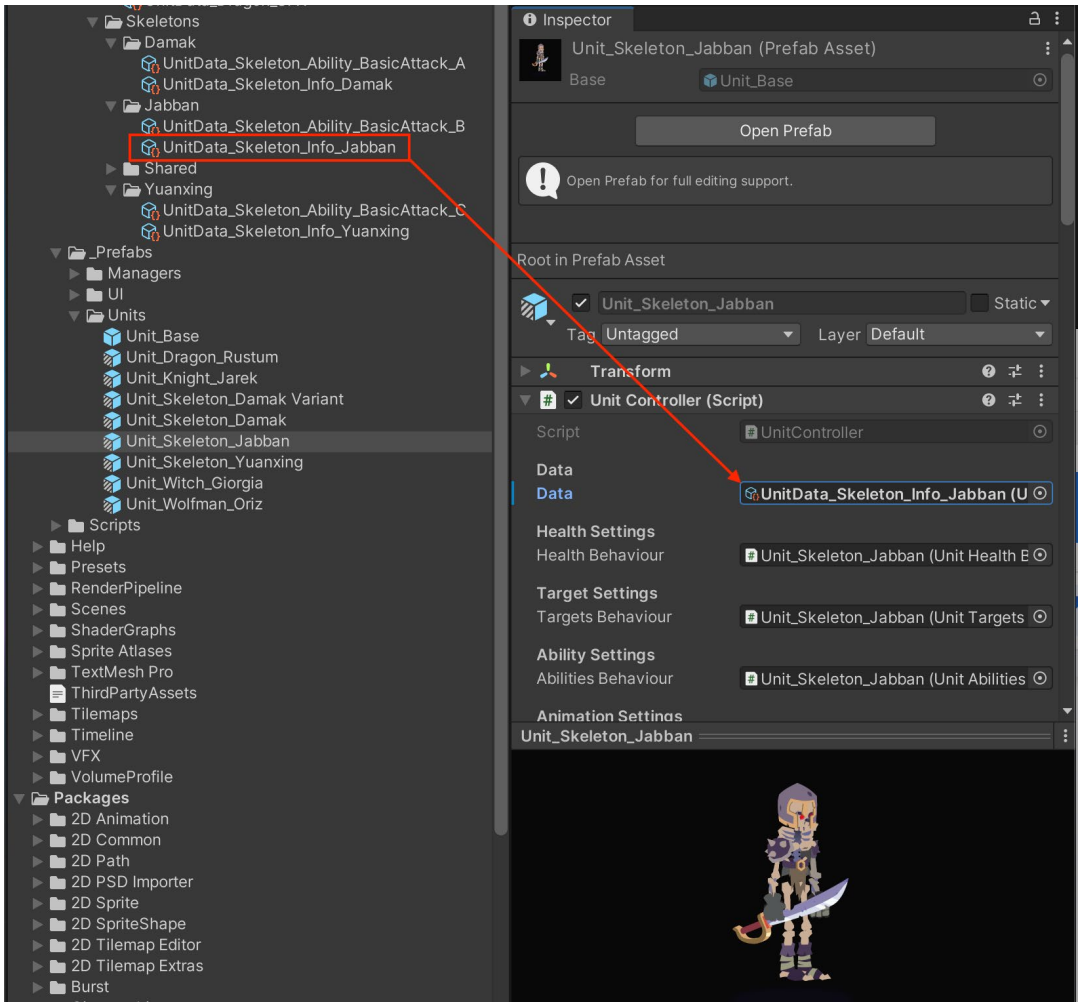
CreateAssetMenu 속성은 스크립터블 오브젝트 에셋을 생성하는 데 도움이 되는 컨텍스트 메뉴 항목을 생성합니다. 각 유닛에는 음향 효과와 특수 능력에 대한 추가 스크립터블 오브젝트가 있습니다.

프로젝트 창에서 생성한 에셋으로 인스펙터를 사용하여 Unit Name, Unit Avatar(스프라이트), Total Health 필드에 올바른 값을 입력할 수 있습니다.

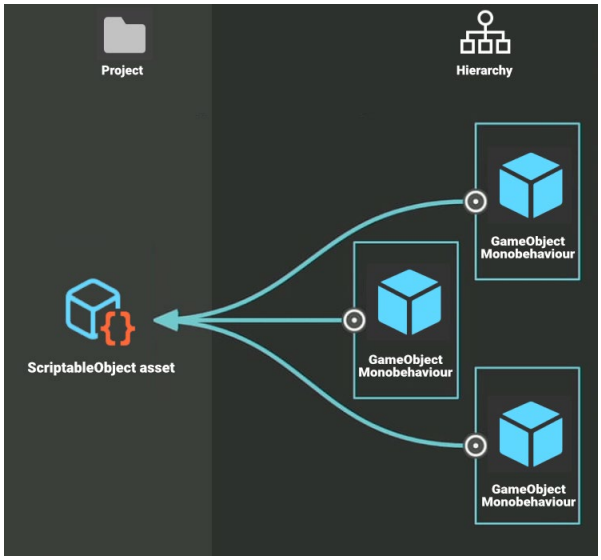


인스펙터를 사용하여 스크립터블 오브젝트 에셋에 대한 값을 입력합니다. 이 값은 게임플레이 중에 변하지 않습니다.

그러면 게임 오브젝트(여기서는 UnitController)는 스크립터블 오브젝트 에셋을 참조할 수 있습니다. 씬이 갑자기 유닛으로 채워지는 경우, 스크립터블 오브젝트 에셋의 데이터가 복제되지 않으므로 메모리를 절약할 수 있습니다.



프로젝트의 스크립터블 오브젝트 데이터 에셋을 참조하는 Monobehaviour 오브젝트(위의 UnitController)



스크립터블 오브젝트로 메모리를 절약하고 스크립트를 정돈된 상태로 유지할 수 있습니다. 게임 오브젝트가 많은 경우에도 프로젝트의 에셋에 정적 데이터와 설정을 한 번만 설정하면 됩니다.

씬에 수천 개의 프리팹 인스턴스를 추가하는 경우에도 여전히 에셋에 저장된 동일한 데이터를 참조합니다. 일련의 값을 한 번만 설정하면 일관성을 유지할 수 있습니다.

게임에 더 많은 유닛 유형이 추가될 때마다 스크립터블 오브젝트 에셋을 더 만들고 적절하게 교체하기만 하면 됩니다. 중앙에 저장된 에셋을 미세 조정하기만 하면 게임플레이 데이터를 유지할 수 있습니다.

스크립터블 오브젝트는 게임플레이 중에 데이터가 변경될 수 있는 애플리케이션의 다른 저장 파일에 대한 영구 데이터 유지를 대체하지 않습니다. 스크립터블 오브젝트는 정적 게임플레이 설정과 기본값을 저장하는 데 더 적합한 워크플로입니다. JSON 또는 XML의 데이터 파싱과 달리, 스크립터블 오브젝트 에셋을 읽을 때 가비지가 생성되지 않으며 더 빠릅니다.

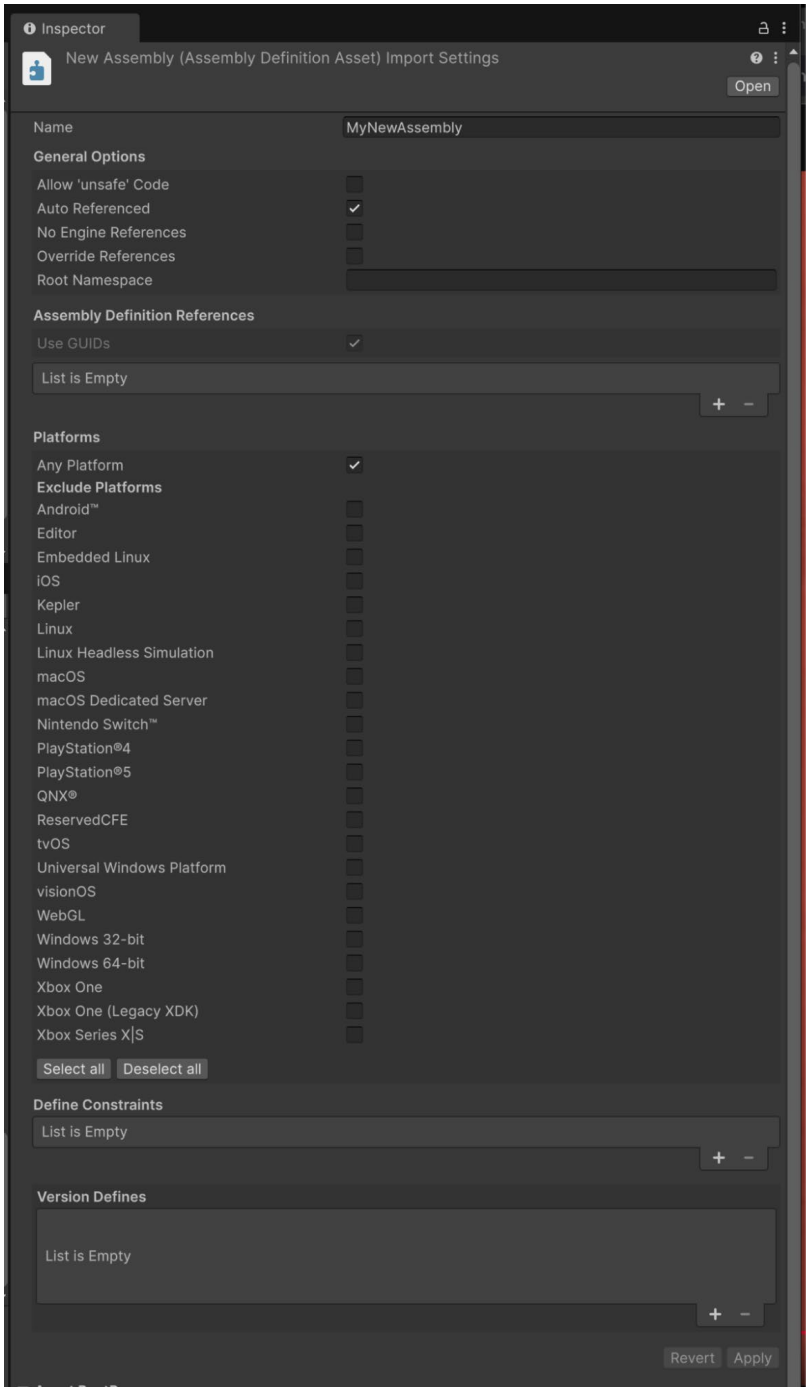
스크립터블 오브젝트에 관한 추가 리소스:

- 스크립터블 오브젝트를 사용하여 Unity에서 모듈식 게임 아키텍처 만들기
- 스크립터블 오브젝트 패들 볼 데모 프로젝트
- 스크립터블 오브젝트 기술 자료

어셈블리 정의를 사용하여 스크립트의 모듈성 향상

어셈블리는 관련 유형과 리소스를 하나의 논리적 단위로 그룹화하는 컴파일된 C# 코드 라이브러리입니다. Unity에서는 어셈블리 정의 파일(.asmdef)을 사용하여 어셈블리를 관리할 수 있습니다. 스크립트를 커스텀 어셈블리로 구성하면 모듈성과 재사용성이 향상되고 컴파일 시간이 단축됩니다. 또한 스크립트가 기본 어셈블리에 자동으로 추가되지 않으며, 액세스할 수 있는 스크립트를 제한합니다.

어셈블리 정의를 사용하여 프로젝트를 정리하는 과정에서 빌드에 에디터 스크립트를 포함하는 경우, Editor 폴더 안에 어셈블리 정의를 만든 다음 Editor 플랫폼만 포함하도록 설정하세요.



인스펙터의 어셈블리 정의 설정

Input System으로 업그레이드

아직 Input System 패키지로 업그레이드하지 않았다면 업그레이드를 고려해 보세요. Input Manager보다 유연한 이 최신 패키지를 사용하면 모든 유형의 입력 기기를 사용하여 Unity 콘텐츠를 제어할 수 있습니다. 이 시스템은 'Input System 패키지' 또는 'Input System'이라고 칭합니다. 또한 리바인드 가능한 컨트롤과 입력 동작 에셋을 지원하고, 입력과 게임플레이 로직을 명확하게 구분할 수 있으므로 레거시 시스템에 비해 상당한 이점을 얻을 수 있습니다.

다음 자료를 참조하여 시작해 보세요.

— Unity 6의 Input System을 사용하여 더 빠르게 모바일 게임 프로토타이핑하기 | 유나이트 2024(영어)

— Input System 시작하기

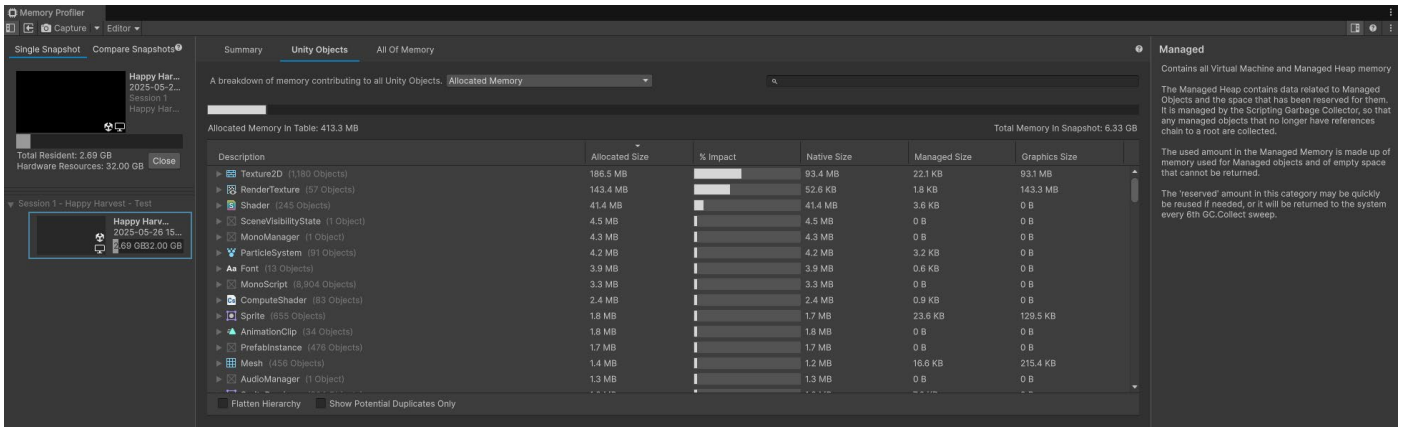
— Unity Input System 관련 7개 동영상 튜토리얼 시리즈(영어)

프로파일링 툴

Memory Profiler를 사용하여 메모리 성능 최적화

Memory Profiler를 사용하면 프로젝트의 메모리 사용량을 캡처하고 분석하여 메모리 누수를 식별하고, 메모리 급증을 줄이고, 런타임 성능을 최적화할 수 있습니다. 중요한 순간(예: 씬 로드, 긴 플레이 세션 후 등)에 메모리 스냅샷을 만들고 비교하여, 올바르게 해제되지 않는 오브젝트를 찾아내는 데 사용하세요.

코드에서 리소스를 만들 때는 Memory Profiler에서 해당 리소스의 이름을 지정하는 습관을 들이는 것이 좋습니다. 또한 메모리 누수를 방지하려면 메모리를 할당한 모든 오브젝트를 해제해야 합니다.



Memory Profiler의 스냅샷

ProfilerMarker를 사용하여 성능에 중요한 코드 섹션 파악

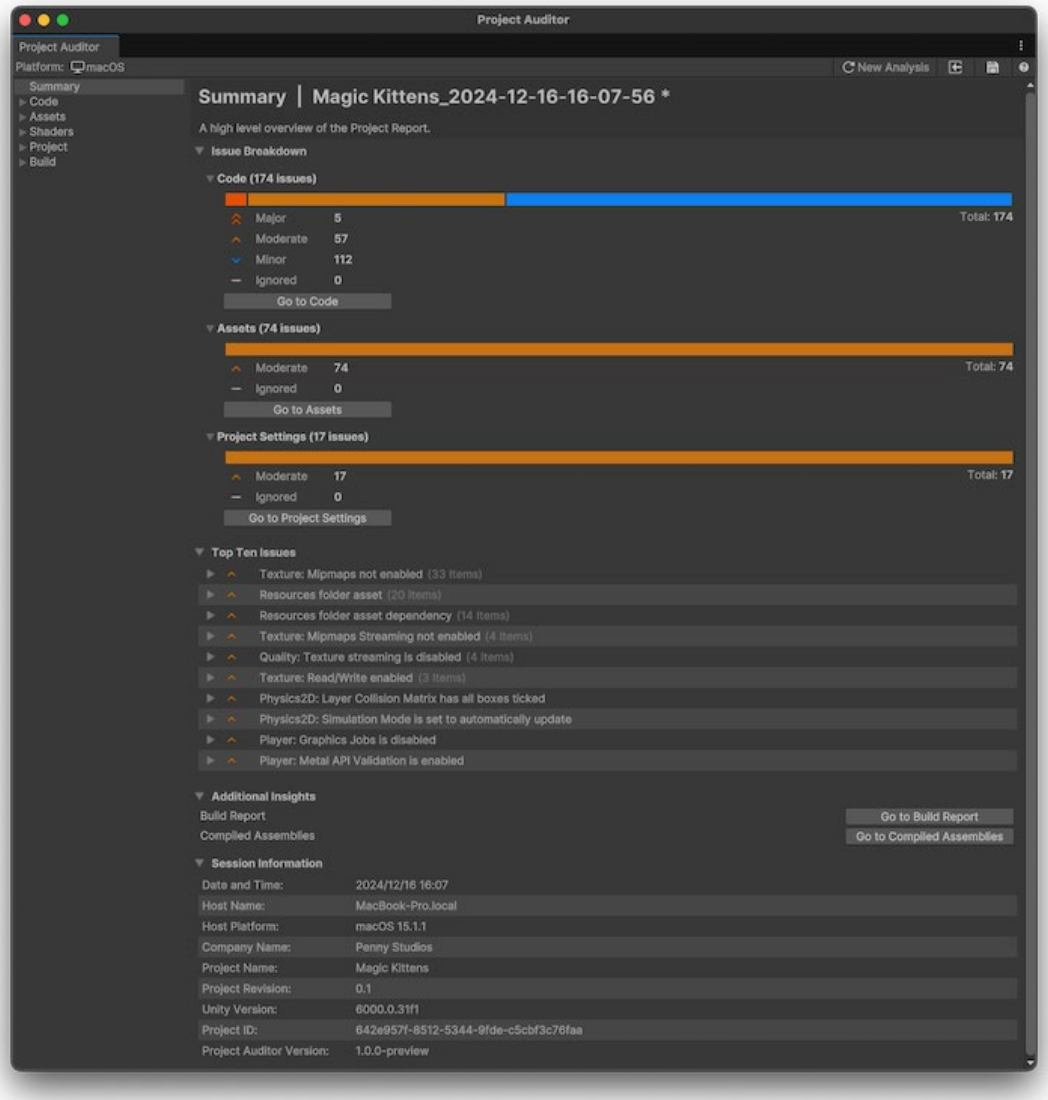
BehaviourUpdate 같은 일반적인 마커로 집계된 퍼포먼스 데이터만 확인하지 않고, 특정 함수의 정확한 실행 시간을 분리하여 측정할 수 있습니다. ProfilerMarker로 스크립트 코드 블록을 마크업하면 프로파일링을 더 디테일하게 실행할 수 있습니다. 수집된 정보는 CPU Profiler에 표시되며 Unity Recorder로도 캡처할 수 있습니다. 이를 통해 시간이 소모되는 특정 코드 섹션에 대한 상세한 분석을 바탕으로 더 쉽게 성능 병목 현상을 식별하고 코드를 최적화할 수 있습니다.

```
using UnityEngine;
using Unity.Profiling;
public class UnityTips : MonoBehaviour {
   private static readonly ProfileMarker SetupProfileMarker = new Pro-
fileMarker("Setup");
   private static readonly ProfileMarker ExpensiveProfileMarker = new
ProfileMarker("Expensive");

   public void UpdateLogic() {
       SetupProfileMarker.Begin();
       // 성능에 부담을 주는 요소, 이니셜라이저 등 설정
       SetupProfileMarker.End();
       using (ExpensiveProfileMarker.Auto()) { // 자동으로 시작되고 종료됨
       // 성능에 부담이 되는 요소 추가
```

프로젝트의 성능 감사 받기

Unity 6.1에서 패키지로 도입된 Project Auditor를 사용하여 프로젝트 성능을 분석하고, 베스트 프랙티스를 적용하고, 잠재적인 문제와 병목 현상을 찾아낼 수 있습니다.

클릭 몇 번으로 프로젝트 전체를 스캔한 다음, 과도한 스크립팅 호출, 사용되지 않는 에셋, 지나치게 많은 엔티티 수 같은 비효율적인 지점에 대한 상세한 보고서를 받을 수 있습니다.



Project Auditor Summary 뷰

생성된 보고서는 오류, 경고, 정보 등의 심각도로 분류되어 메모리 초과 할당, 과잉 가비지 컬렉션 같은 오류와 경고를 해결하는 데 우선적으로 집중할 수 있습니다.

개발의 주요 단계(예: 중요한 마일스톤 전, 베타 릴리스, 최종 빌드 등)에서 Project Auditor를 실행하는 것이 권장됩니다. 성능 병목 현상, 사용되지 않는 에셋, 오래된 코드를 초기에 발견하여 프로젝트의 규모가 커짐에 따라 문제가 덩달아 커지는 일을 방지할 수 있습니다.

커스텀 규칙과 필터를 사용하여 Project Auditor를 커스터마이즈할 수 있습니다. 예를 들어 사용하지 않아야 하거나 실험 기능인 스크립트와 에셋은 분석에서 제외하거나, 프로젝트 '예산'에 최적화되도록 빌드 타겟, 해상도, 텍스트 압축, 그 밖의 프로젝트 설정을 위한 규칙을 만들 수 있습니다.

애니메이션 커브

커스텀 lerp 함수를 사용하여 보간 제어

Mathf.Lerp(a, b, t)는 기본적으로 보간 계수 t를 0과 1 사이로 클램프합니다. 즉, a와 b 사이의 범위를 벗어나는 값은 반환하지 않습니다. 오버슈트(t>1) 값이나 언더슈트(t<0) 값이 필요하다면 대신 Mathf.LerpUnclamped(a, b, t)를 사용하세요. 이렇게 하면 보간을 원하는 대로 제어하고 외삽이나 모멘텀 기반 모션 같은 효과를 적용할 수 있습니다.

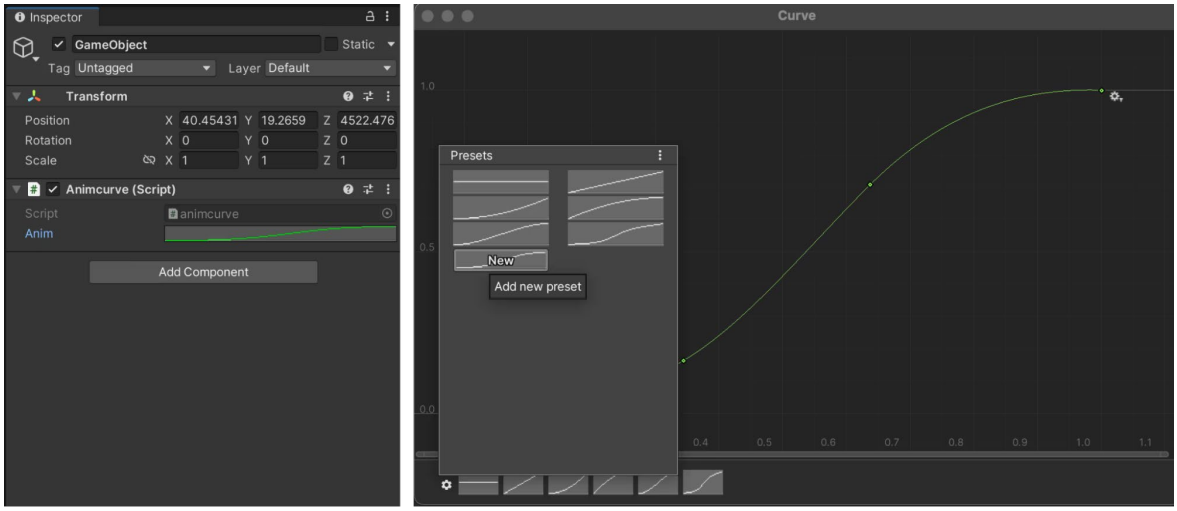
```
using UnityEngine;

public class LerpComparison : MonoBehaviour
{
    [SerializeField] private float start = 0f;
    [SerializeField] private float end = 10f;
    [SerializeField] private float t = 1.5f;

    private void Start()
    {
        // t를 [0, 1]로 클램프
        float clamped = Mathf.Lerp(start, end, t);
        // 전체 t 값 사용
        float unclamped = Mathf.LerpUnclamped(start, end, t);

        // 10 출력
        Debug.Log($"Mathf.Lerp: {clamped} (t = {t})");
        // 15 출력
        Debug.Log($"Mathf.LerpUnclamped: {unclamped} (t = {t})");
     }
}
```

Unity에서 커스텀 lerp를 사용하는 예시

애니메이션 외 다른 용도로 애니메이션 커브 사용

애니메이션 커브는 주로 AnimationClip에서 component 프로퍼티 값을 애니메이션화하는 데 사용되지만, float 값을 동적으로 제어하는 용도로 사용할 수도 있습니다.

애니메이션 커브는 퍼블릭 변수로 선언했을 때나 직렬화되었을 때 인스펙터에서 편집할 수 있습니다. 이 경우 편집 모드 또는 런타임에 저장, 익스포트, 로드할 수 있습니다. 편집 가능한 tangent를 사용하면 키 사이 구간에서 커브의 셰이프를 제어할 수 있습니다.



인스펙터의 애니메이션 커브 프로퍼티: 클릭하면 커브 에디터가 열립니다. 여기에서 톱니바퀴 아이콘을 클릭하여 커브를 조정하고 자체 라이브러리에 저장할 수 있습니다.

프로젝트에서 애니메이션 커브를 사용하는 실무 팁과 예시는 애니메이션 커브를 최고의 디자인 툴로 활용하는 방법 블로그 게시물을 참조하세요.

오브젝트 풀링으로 프로세싱 전력 감축

오브젝트 풀링은 CPU가 Create 및 Destroy 호출을 반복하는 데 필요한 프로세싱 전력을 줄여 성능을 최적화할 수 있는 디자인 패턴입니다. 오브젝트 풀링을 활용하면 호출을 반복하는 대신 기존 게임 오브젝트를 계속 재사용할 수 있습니다.

오브젝트 풀을 사용하는 방식은 애플리케이션에 따라 달라집니다. GC 스파이크가 발생할 수 있으므로 다수의 오브젝트를 인스턴스화할 때마다 코드를 프로파일링하는 것이 좋습니다.

게임플레이의 안정성을 해칠 정도의 막대한 스파이크가 감지된다면 오브젝트 풀을 사용해 보세요. 단, 오브젝트 풀링은 풀의 여러 라이프사이클을 관리해야 하므로 코드베이스가 더 복잡해질 수 있다는 점을 기억해야 합니다. 또한 너무 많은 예비 풀을 생성하여 게임플레이에 필요하지 않을 수 있는 메모리를 할당하게 될 수 있습니다.

디자인 패턴 및 SOLID 원칙으로 코딩 스킬 업그레이드 전자책과 Unity 에셋 스토어의 무료 샘플 프로젝트에서 오브젝트 풀링에 대해 자세히 알아보세요.

더 많은 자료

- C# 스타일 가이드로 깔끔하고 스케일링 가능한 게임 코드 작성하기(Unity 6 에디션)
- Unity 게임 디자이너 플레이북
- Unity에서 스크립터블 오브젝트로 모듈식 게임 아키텍처 만들기
- Unity에서 어드레서블을 사용하여 효과적으로 에셋 관리하기
- Unity 6의 빌드 프로파일에 대해 알아야 할 사항

IDE 및 디버깅

Debug.Break로 실행 일시 정지

애플리케이션을 수동으로 일시 중지하기 어려운 경우 인스펙터에서 특정 값을 확인하려면 Debug.Break를 사용하여 코드의 실행을 일시 중지할 수 있습니다.

Debug.Assert로 if 문 저장

Debug.Assert는 런타임에 조건을 검사하여 입력한 조건이 false를 반환하면 콘솔에 오류 메시지를 출력합니다. 항상 실행되는 **Debug.Log**와 달리 단언문은 예기치 않은 상태를 알리는 용도로 사용되며, 코드에 있는 가정의 유효성을 검사하는 데 더 효과적입니다.

```
// 릴리스에서 if 문을 저장하려면...
if (health > maxhealth)
{
  Debug.LogError("Current health is greater than maxhealth!", this);
}
//...단언문 사용하기
Debug.Assert(health < maxhealth, "Current health is greater than maxhealth!", this);
```

Debug.Log를 컨텍스트와 함께 사용

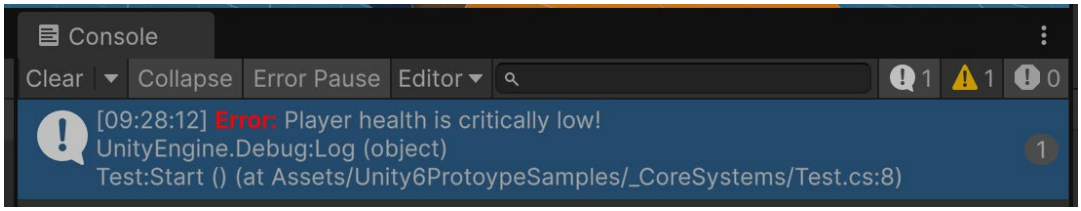
Debug.Log를 사용하면 오브젝트(주로 게임 오브젝트 또는 컴포넌트)를 두 번째 파라미터로 전달할 수 있습니다. 그러면 콘솔에서 로그 메시지가 해당 오브젝트에 연결됩니다. 메시지를 클릭하면 계층 창에서 연결된 오브젝트가 강조 표시됩니다.

Debug.Log("Enemy spawned", gameObject);

리치 텍스트를 사용하여 중요 메시지 강조

Unity 콘솔은 **Debug.Log** 메시지에서 일부 리치 텍스트(<b>, <i>, <color> 등)를 지원합니다. 리치 텍스트를 사용하여 로그 출력의 일부분을 강조 표시하거나, 컬러로 구분하거나, 강조하여 개발 중에 중요한 메시지를 시각적으로 구분할 수 있습니다.

```
Debug.Log("<b><color=red>Error:</color></b> Player health is critically low!");
```



빌드에서 디버그 로그 제거

Unity는 비개발 빌드에서 **디버그** 로깅 API를 자동으로 제거하지 않습니다. **디버그 로그** 호출을 커스텀 메서드로 래핑하고 **[Conditional]** 속성을 추가해야 합니다.

Player Settings에서 해당하는 **스크립팅 정의 기호**를 제거하면 디버그 로그가 한 번에 컴파일됩니다. 이는 디버그 로그를 **#if... #endif** 프리 프로세서 블록으로 래핑하는 것과 동일합니다.

예제는 일반 최적화 가이드를 참조하시기 바랍니다.

레이캐스팅을 시각화하여 물리 관련 문제 해결

물리와 관련된 문제를 해결 중이라면 **Debug.DrawLine**과 **Debug.DrawRay**로 지정된 시작점과 끝점 사이에 선을 그려 레이캐스팅을 시각화할 수 있습니다.

```
    void Start()
     {
         // 원점부터  5단위만큼  2.5초 동안 흰 선 그리기
         Debug.DrawLine(Vector3.zero, new Vector3(5, 0, 0), Color.white,
2.5f);
     }
```

개발 빌드에 Debug.isDebugBuild 사용

Debug.isDebugBuild를 사용하여 애플리케이션이 개발 빌드로 실행 중인지 확인할 수 있습니다. 이렇게 하면 릴리스 빌드에 영향을 주지 않으면서 로깅, 진단, 테스트 유틸리티 같은 디버그 전용 코드를 조건부로 실행할 수 있습니다.

```
if (Debug.isDebugBuild)
{
    Debug.Log("Running in Development Build mode.");
}
```

Application.SetStackTraceLogType 설정

Application.SetStackTraceLogType을 사용하거나 Player Settings에서 해당 옵션을 체크하여 스택 추적을 포함할 로그 메시지 유형을 결정합니다. 스택 추적은 유용할 수 있지만 속도가 느리고 가비지를 생성합니다.

에디터 창에 표시된 PlayerSettings의 Stack Trace

로그 커스터마이징

Logger API를 사용하면 고급 또는 모듈식 로깅을 위해 커스텀 로거 인스턴스를 만들고 구성할 수 있습니다. 빌트인 Debug.unityLogger를 사용해도 되지만, 로거를 직접 만들면 로그 포맷 지정, 필터링, 출력 채널을 세밀하게 제어할 수 있습니다.

```
var logger = new Logger(Debug.unityLogger.logHandler);
logger.Log(LogType.Log, "Custom log message");
```

Visual Code 단축키를 사용하여 워크플로 속도 향상

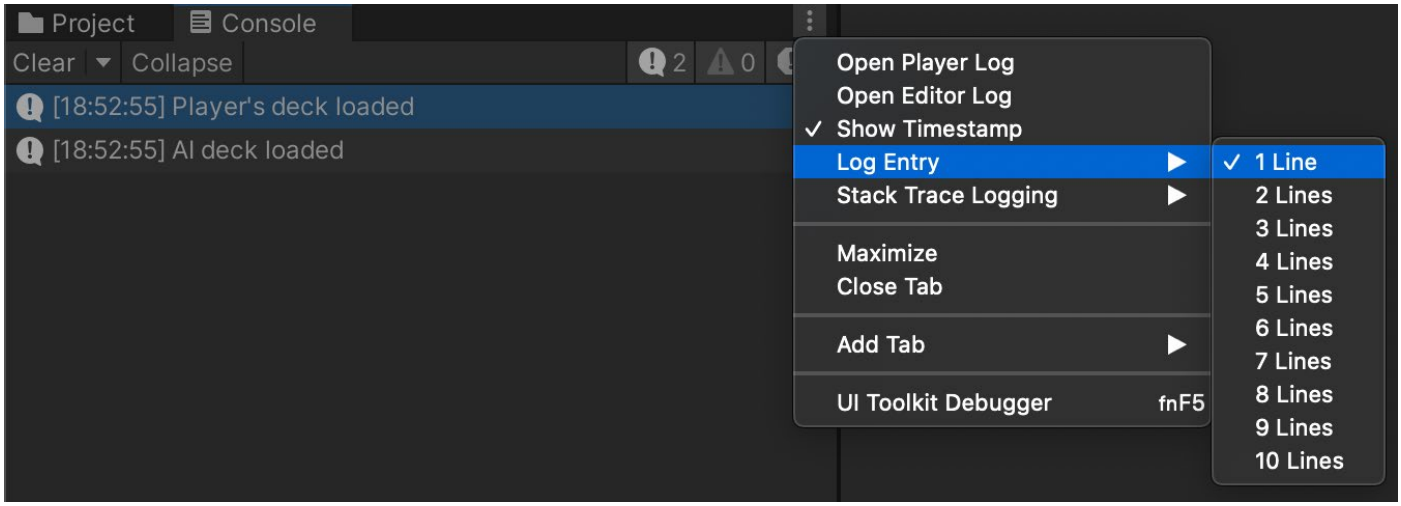
Visual Code를 IDE로 사용하는 경우 아래에서 소개하는 단축키를 유용하게 사용할 수 있습니다.

— Windows

— Mac

가독성 높은 Console Log Entry 구성

기본적으로 Console Log Entry는 두 줄로 표시됩니다. 개인 선호에 따라 한 줄 또는 여러 줄 형식으로 보다 간결하게 구성하여 가독성을 높일 수 있습니다(아래 이미지 참조).



Console Log Entry 옵션을 사용하여 로그의 줄 수를 설정할 수 있습니다.

더 많은 자료

- Roslyn 분석기로 코드 품질과 유지 관리성을 보장하는 방법
- Unity Test Framework를 통해 자동으로 게임 테스트를 실행하는 방법
- Microsoft Visual Studio 코드로 디버깅 워크플로 속도 향상
- Microsoft Visual Studio 2022로 코드를 디버깅하는 방법
- Unity 프로젝트를 위한 테스트 및 품질 보증 팁

DevOps 워크플로

Unity Version Control 팁

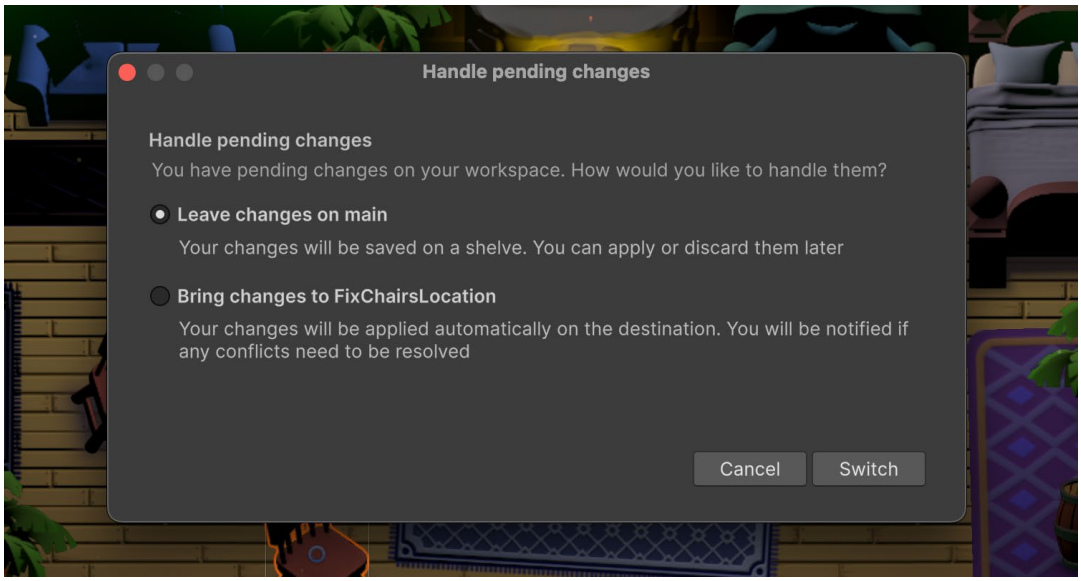
UVCS(Unity Version Control)는 모든 규모의 게임 개발 스튜디오에 맞게 개발된 확장성 있고 엔진에 구애받지 않는 버전 관리 및 소스 코드 관리 툴입니다.

Unity 6에서 Unity Version Control을 사용하고 있다면 아래에서 일상적인 작업에 유용하게 사용할 수 있는 최신 업데이트와 팁을 확인해 보세요.

Shelve와 Switch를 사용하여 작업 손실 없이 여러 브랜치를 병렬로 작업

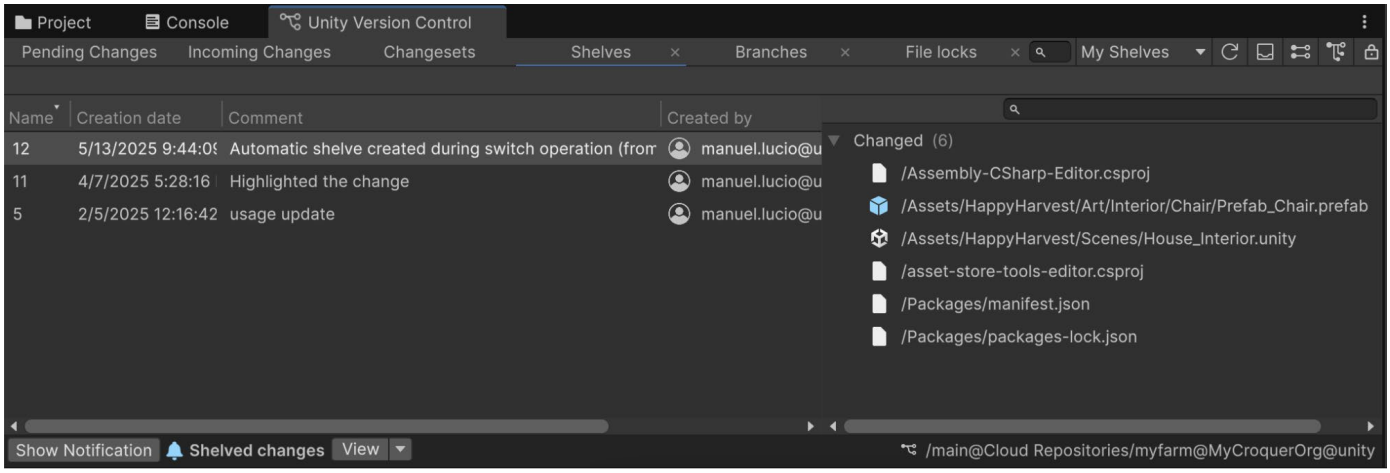
UVCS 패키지 버전 2.7.1부터 여러 브랜치 간에 전환할 때 shelvesets를 사용하여 현재 변경 사항을 임시로 저장할 수 있으며, 이에 따라 진행 중인 작업을 안전하게 저장하고 쉽게 액세스할 수 있습니다.

Unity Version Control 창에서 전환하려는 목표 브랜치를 오른쪽 클릭하고 **Switch workspace to this branch**를 선택합니다.



Handle pending changes 창

그런 다음 변경 사항을 적용할지 아니면 임시로 남겨 둘지 선택합니다. **Shelves** 옵션은 변경 사항을 다시 살펴보고, 재적용하고, 팀원들에게 공유할 수 있는 등 뛰어난 유연성을 제공합니다.

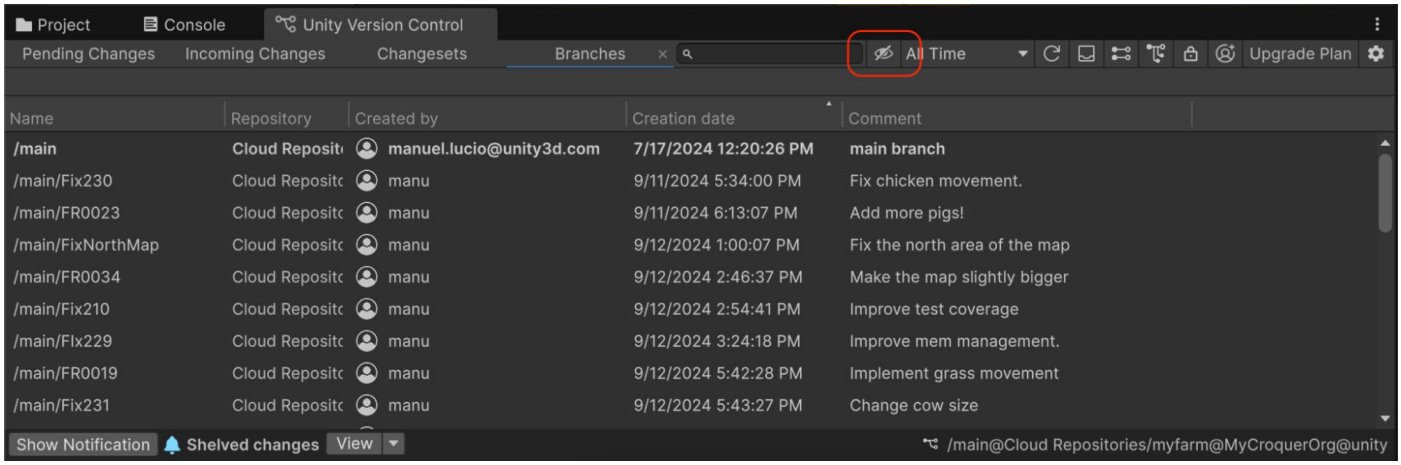


Unity Version Control 창의 Shelves 탭

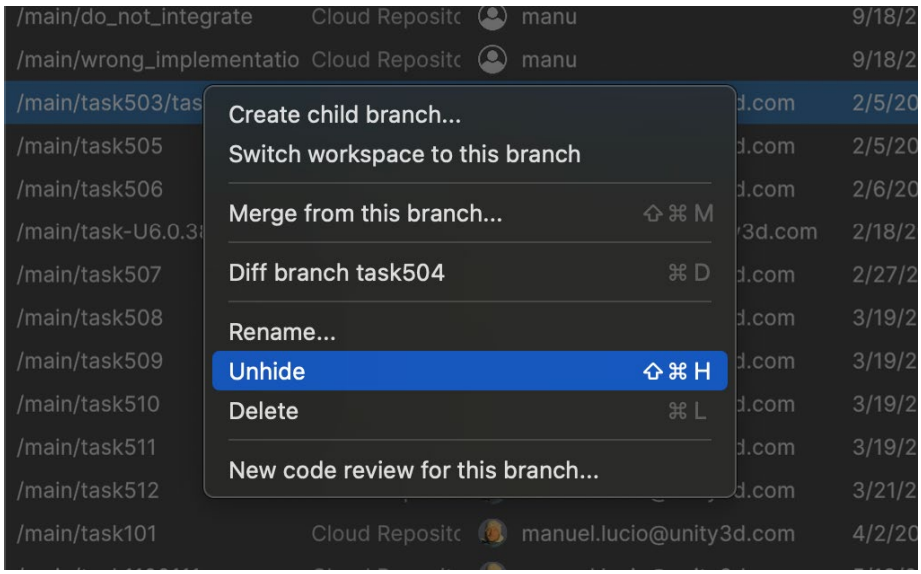
Hide/Unhide 기능으로 Branches 뷰 정리

프로젝트의 Branches 뷰가 복잡해지면 컨텍스트 메뉴에서 전역 **Hide/Unhide** 기능을 사용하여 브랜치를 숨겨 보세요. 그러면 전체 팀을 대상으로 모든 UVCS 플랫폼(Unity 플러그인, Unity Dashboard, 데스크톱 애플리케이션)에서 해당 브랜치를 숨길 수 있습니다.

숨겨진 브랜치를 다시 표시하려면 '사선이 그어진 눈' 아이콘이나 숨겨진 브랜치 목록의 컨텍스트 메뉴를 사용하면 됩니다(아래 이미지 참조).



Unity Version Control의 사선이 그어진 눈 아이콘

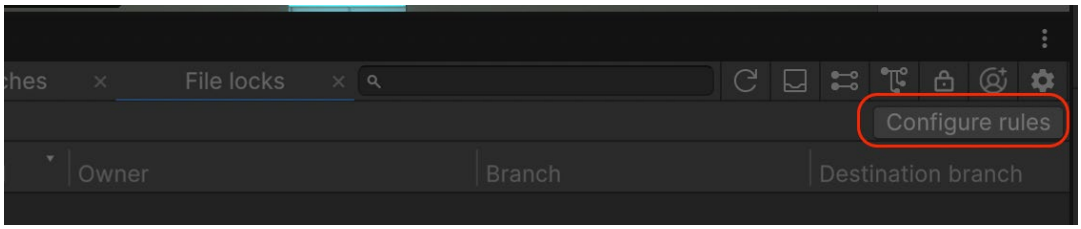


브랜치 컨텍스트 메뉴의 'Unhide' 옵션

배타적 잠금 옵션으로 에셋 데이터 손실 방지

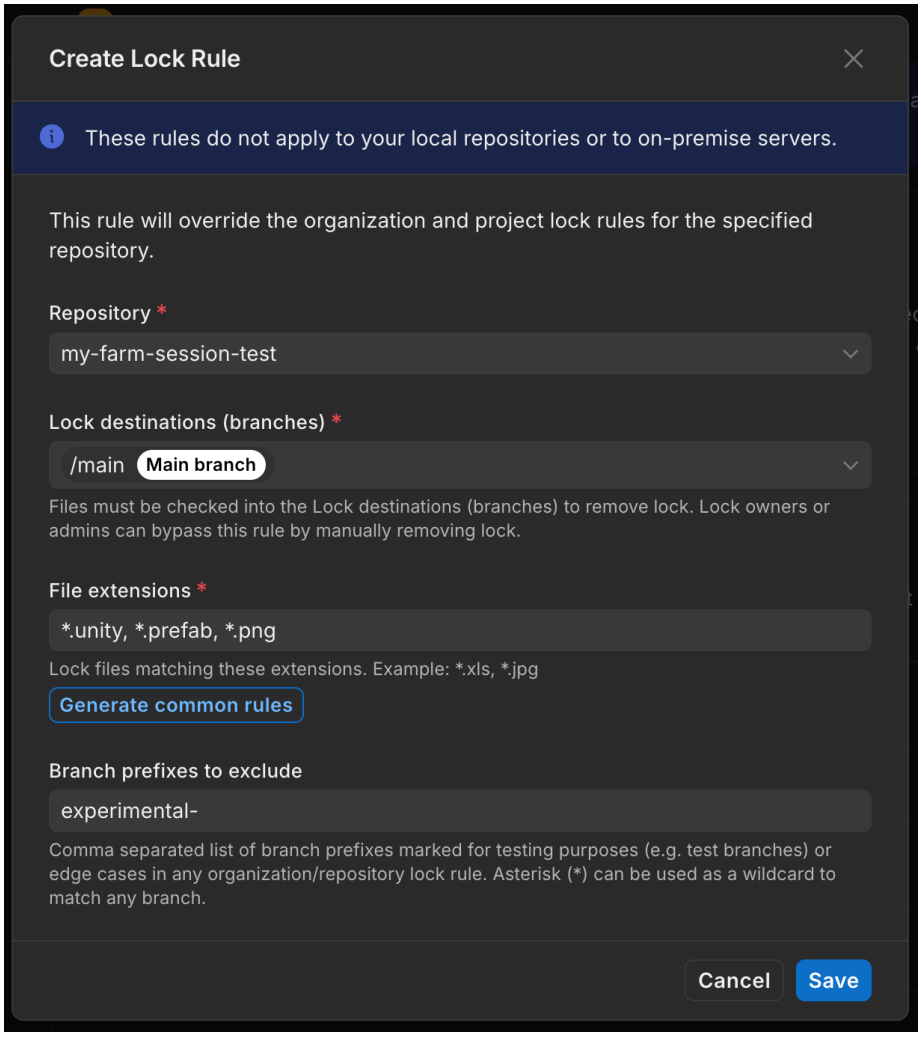
Unity 에디터의 UVCS 플러그인을 통해 다른 사람이 에셋을 수정하지 못하도록 잠글 수 있습니다. 이렇게 하면 작업이 진행 중인 데이터의 손실이 방지되고 병합할 수 없는 파일에 대해 병합 작업이 제거됩니다.

잠금을 설정하려면 **File locks** 탭의 **Configure rules**로 이동합니다.



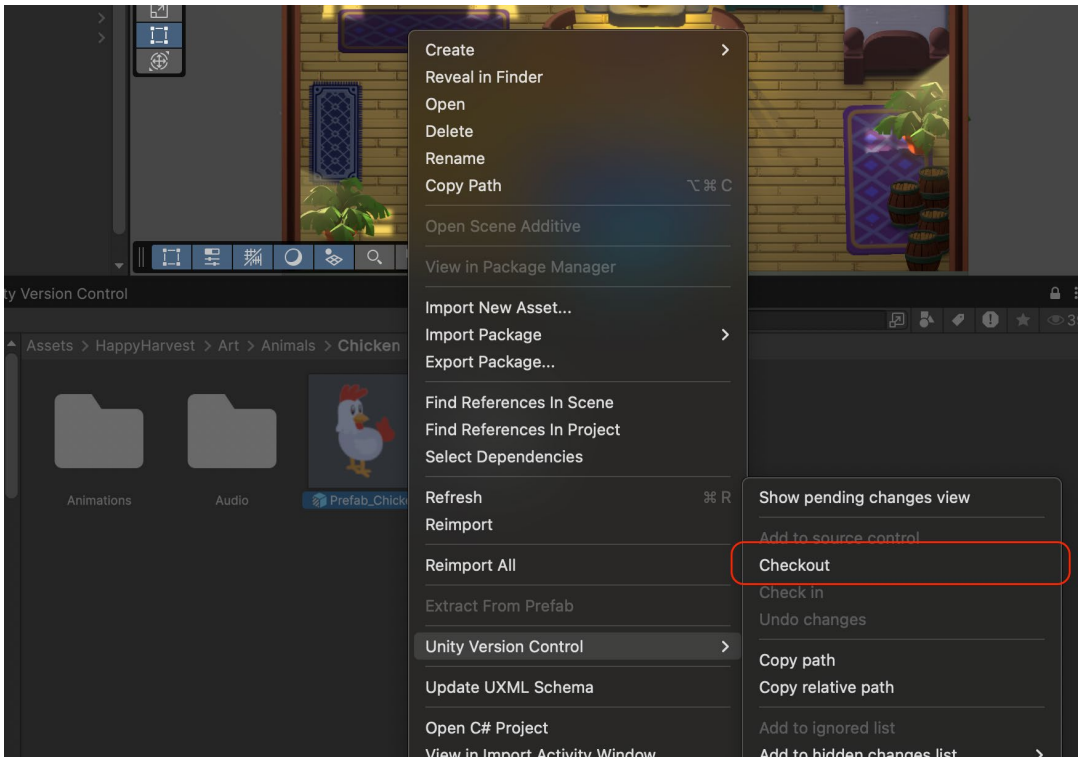
File locks 탭의 Configure rules 옵션

여기에서 저장소, 목표 브랜치(주로 메인 브랜치), 원하는 파일 확장자를 구성하여 잠금을 설정합니다.



Unity Version Control의 Create Lock Rule

충돌을 방지하려면 변경 사항을 적용하기에 앞서 에셋을 체크아웃하세요.



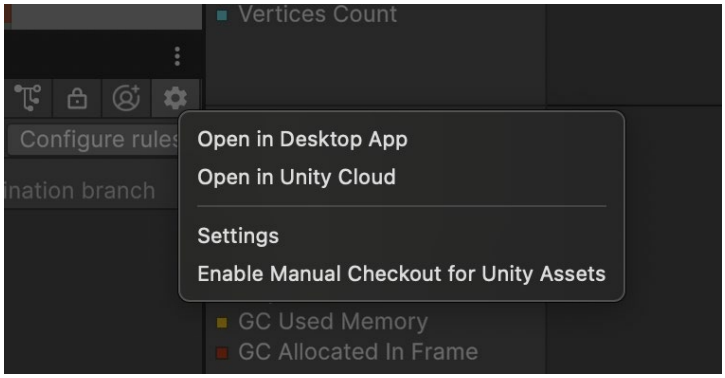
Unity Version Control 컨텍스트 메뉴의 Checkout 옵션

스마트 잠금 시스템에 대해 자세히 알아보려면 스마트 잠금 페이지를 참조하세요.

‘Enable manual checkout for Unity Assets’로 변경 사항 보호

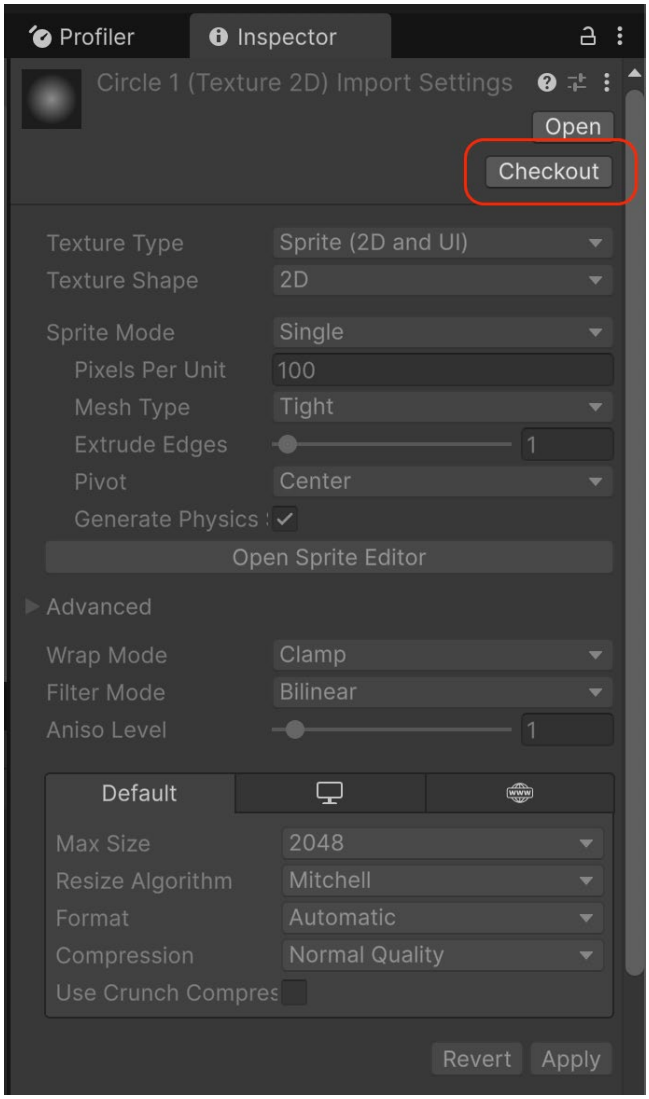
사용자가 병합 불가 에셋을 수정 및 저장하면 UVCS Unity 플러그인이 자동으로 파일을 체크아웃합니다. 이 워크플로로 인해 사용자의 변경 사항이 적용되지 않은 동일 파일을 다른 사람이 잠글 수 있는 여지가 발생합니다.

이를 방지하려면 **톱니바퀴** 아이콘을 클릭하여 UVCS 플러그인 설정으로 이동한 다음 **Enable manual checkout for Unity Assets** 옵션을 선택합니다.



Enable Manual Checkout for Unity Assets 옵션

이 옵션을 선택하면 에셋이 명시적으로 체크아웃될 때까지 읽기 전용으로 유지됩니다. 인스펙터에서 **Checkout** 옵션을 클릭하면 파일이 편집 가능해지고, 사용자가 배타적으로 잠금을 설정하여 파일을 변경할 수 있습니다.



인스펙터의 Checkout 옵션

Asset Manager

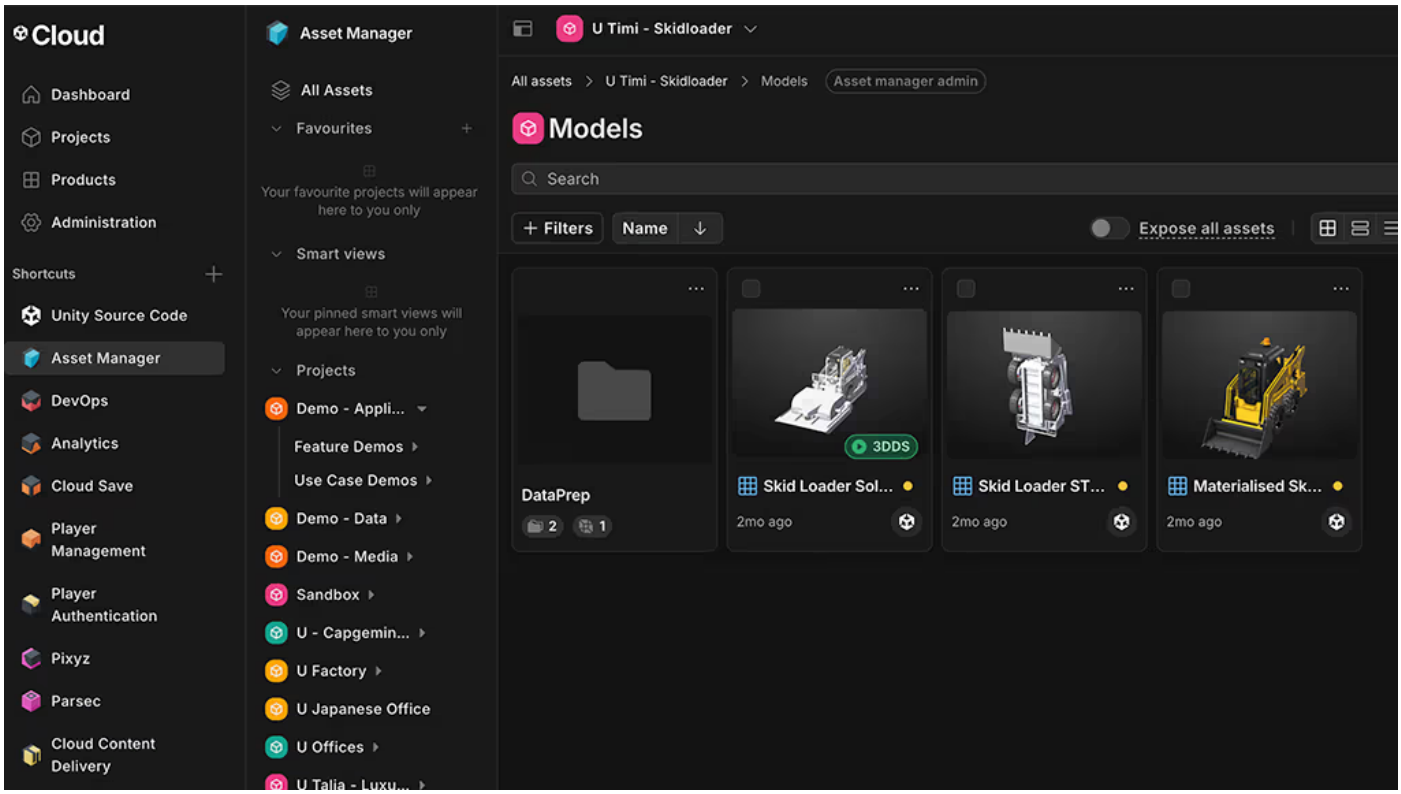
모델, 텍스처, 오디오 파일 등의 에셋을 관리할 디지털 에셋 관리 시스템이 필요하다면 Unity의 Asset Manager를 사용해 보세요. 여러 프로젝트의 에셋을 원활하게 중앙 집중화, 정리, 발견, 사용하는 데 도움이 되도록 70가지가 넘는 파일 포맷이 지원됩니다.

필요에 따라 웹 인터페이스 또는 에디터 통합을 통해 Asset Manager를 사용하세요.

선호하는 브라우저에서 에셋 관리, 조회, 정리

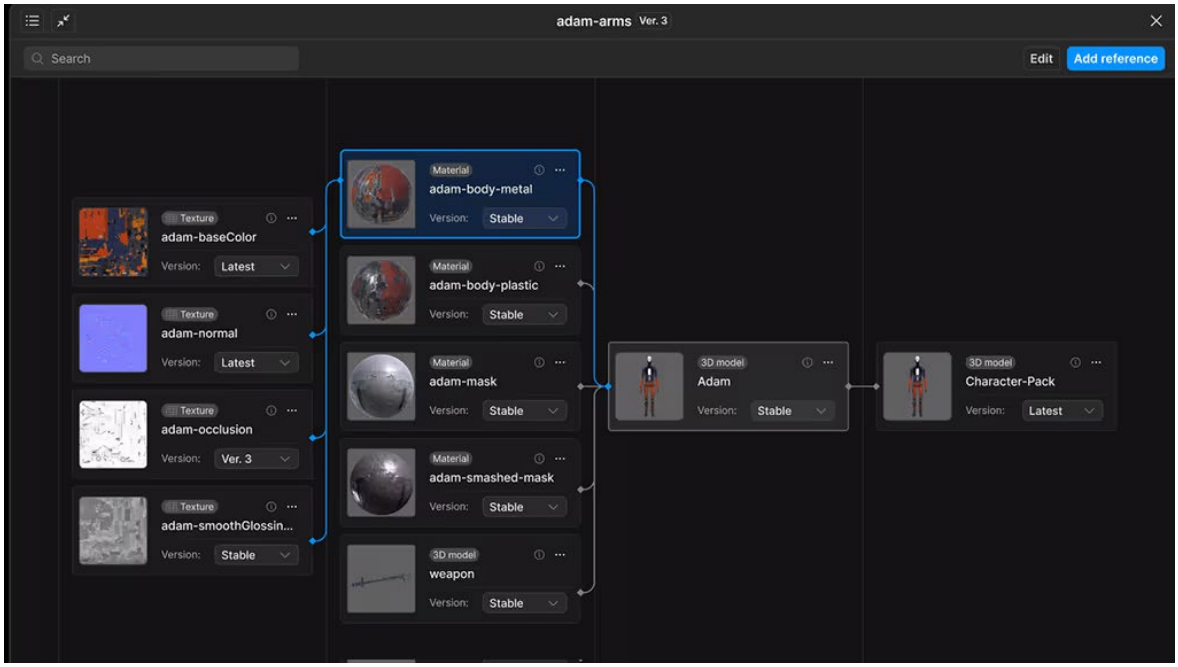
웹 인터페이스를 사용하면 프로젝트의 모든 팀원과 관계자가 빠르게 비동기식으로 협업할 수 있습니다. 웹 인터페이스를 통해 컬렉션을 생성 및 관리하고, 컬렉션 또는 프로젝트 간에 에셋을 이동하고, 버전 이력으로 변경 사항을 추적할 수 있습니다.

관련 컬렉션과 태그를 추가할 때 하나 이상의 파일을 손쉽게 업로드하여 간편하게 에셋을 구성하고 팀원들과 함께 검토 및 개선할 수 있습니다. 에셋에 커스텀 메타데이터를 추가할 수도 있습니다. 관리자, 소유자, 매니저는 텍스트, 부울, 숫자 필드 유형 같은 새 메타데이터 필드를 정의할 수 있고, 모든 사용자는 더 효과적인 관리를 위해 AI 추천 태그를 비롯한 태그를 추가할 수 있습니다.



Asset Manager의 웹 인터페이스에서 에셋 관리

Dependency Viewer는 에셋 관계를 한눈에 보고 각 에셋에 필요한 사항과 종속 항목을 확인할 수 있습니다. Dependency Viewer에서 업스트림 종속성과 다운스트림 종속성을 모두 확인할 수 있습니다.

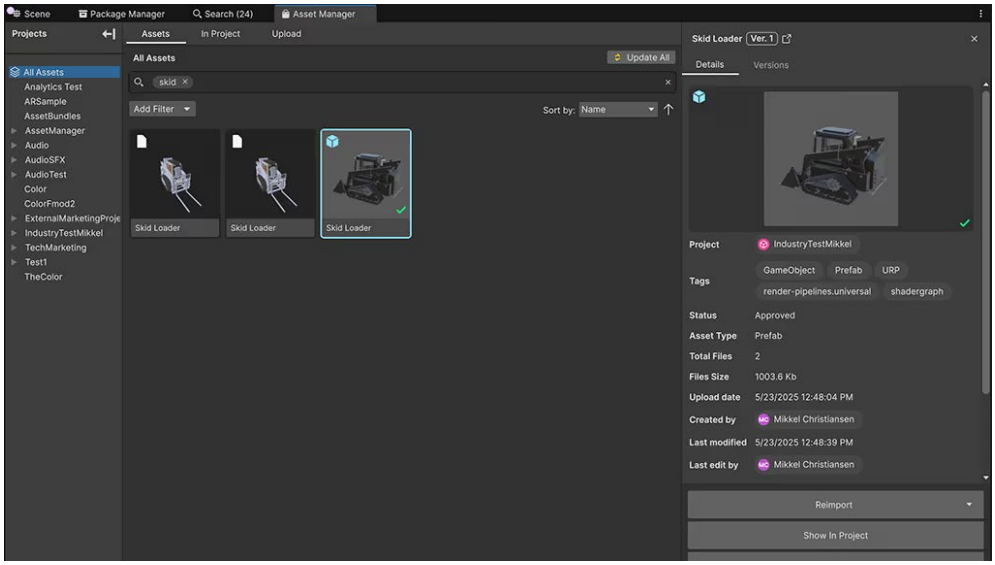


Dependency Viewer의 에셋 종속성 예시

에디터에서 중단 없이 에셋 관리

Unity Asset Manager 패키지는 Unity 에디터와 통합되어 있으므로 워크플로를 중단하지 않고 에셋을 관리할 수 있습니다. 설치한 후 Unity Cloud에 에셋을 업로드하거나 Unity Cloud에서 임포트할 수 있으며, 에디터의 검색 인터페이스를 통해 모든 프로젝트의 에셋에 액세스할 수 있습니다.

에디터 내 솔루션을 통해 프리팹, 스크립트 및 기타 크로스 프로젝트 컴포넌트를 간편하게 재사용하여, 반복 속도를 높이고 팀원들이 더 효과적인 워크플로와 최적화에 집중하도록 지원할 수 있습니다.



Unity 에디터에서 Asset Manager에 액세스

특정 에셋을 효율적으로 찾으려면 만든 사람, 상태, 유형, 업데이트 시간, 임포트 상태를 기준으로 필터링하세요. 플러그인이 모든 에셋에서 자동으로 종속성을 감지합니다. 예를 들어 여러 개의 프리팹이 동일한 머티리얼을 사용한다면 해당 머티리얼의 인스턴스 하나만 업로드됩니다. 따라서 스토리지가 최적화되고 불필요한 중복이 방지됩니다.

Unity Asset Manager에 대해 자세히 알아보려면 다음 자료를 참고하세요.

— 동영상 튜토리얼: Unity의 Asset Manager에 대한 간단한 가이드(영어)

— Unity Learn: Unity Asset Manager: 빠른 시작 가이드

— 문서: Unity Asset Manager를 활용한 전송 방법

Unity Build Automation

Unity DevOps Build Automation(이전 명칭: Cloud Build)은 클라우드에서 빌드를 실행하고 배포하는 턴키식 CI/CD(지속적 통합 및 배포) 솔루션입니다. 이 솔루션을 사용하면 더 자주 빌드하고 릴리스하여 높은 품질의 혁신적인 릴리스를 선보일 수 있습니다.

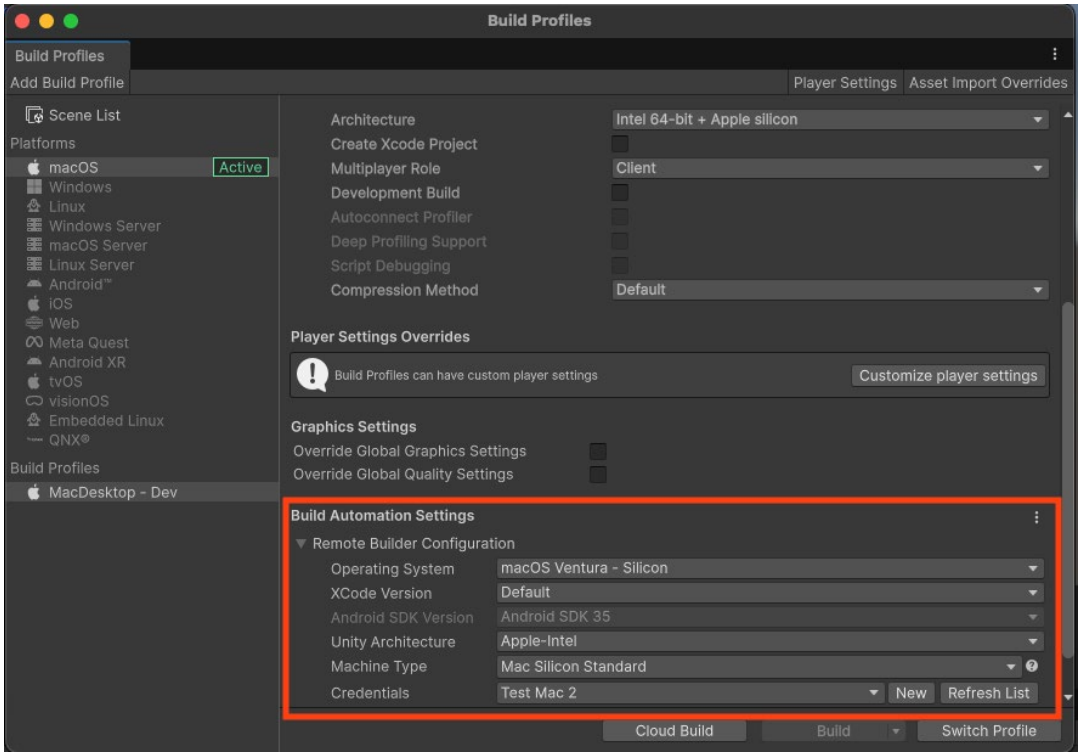
Build Automation은 원하는 소스 관리 저장소에 몇 분 만에 연결할 수 있으며, 버전 관리에 변경 사항이 커밋되면 빌드가 수동 또는 자동으로 실행되도록 설정할 수 있습니다. Build Automation은 iOS, Android, Windows, Unity Web을 비롯한 다양한 플랫폼을 지원하므로 플랫폼별로 빌드 인프라를 관리하는 수고를 덜 수 있습니다.

Unity 6.1은 Build Automation이 에디터에 완전히 통합되어 있어 Build Automation 설정에서 로컬 브랜치의 유효성 검사 빌드를 빠르게 실행할 수 있습니다.

팀에서 빌드 설정을 공유 및 재사용

Build Automation 패키지는 Build Profiles 창 또는 패키지 관리자에서 설치할 수 있습니다.

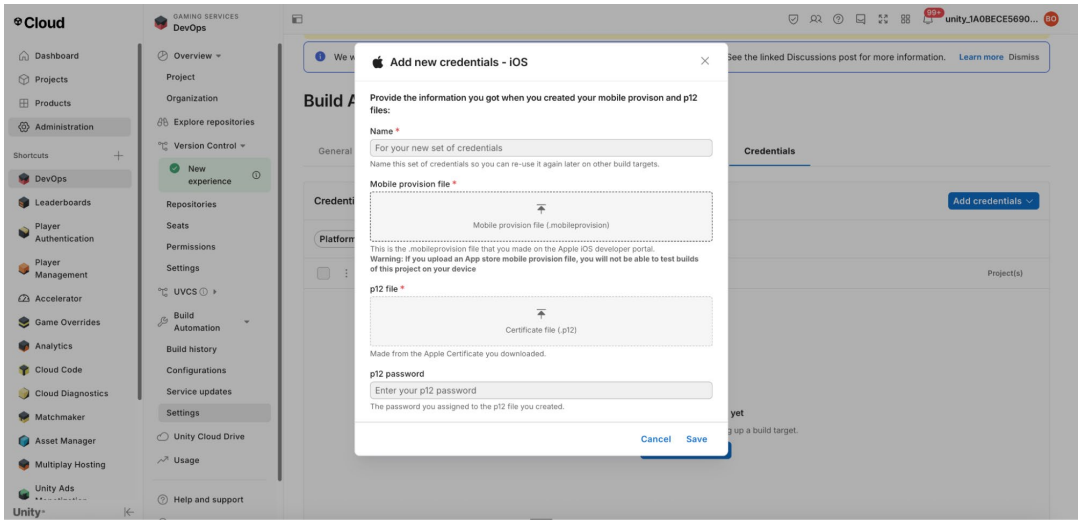
패키지를 설치하면 선택한 프로파일 아래에 **Build Automation Settings**라는 새로운 섹션이 나타납니다. 이 설정을 통해 Unity가 빌드를 자동화하는 방식을 정의할 수 있습니다.



Build Profiles 창의 Build Automation Settings 섹션

Android, iOS 및 자격 증명이 필요한 플랫폼의 경우 Unity Dashboard에서 자격 증명을 설정해야 합니다. 새로운 자격 증명을 만들려면 **Credentials** 드롭다운 옆의 **New** 버튼을 클릭합니다.

그러면 Build Automation 대시보드에 새로운 자격 증명 세트를 생성할 수 있는 창이 새로 열립니다. 자격 증명과 앱 서명에 대해 자세히 알아보려면 Unity Build Automation 기술 자료를 참조하세요.

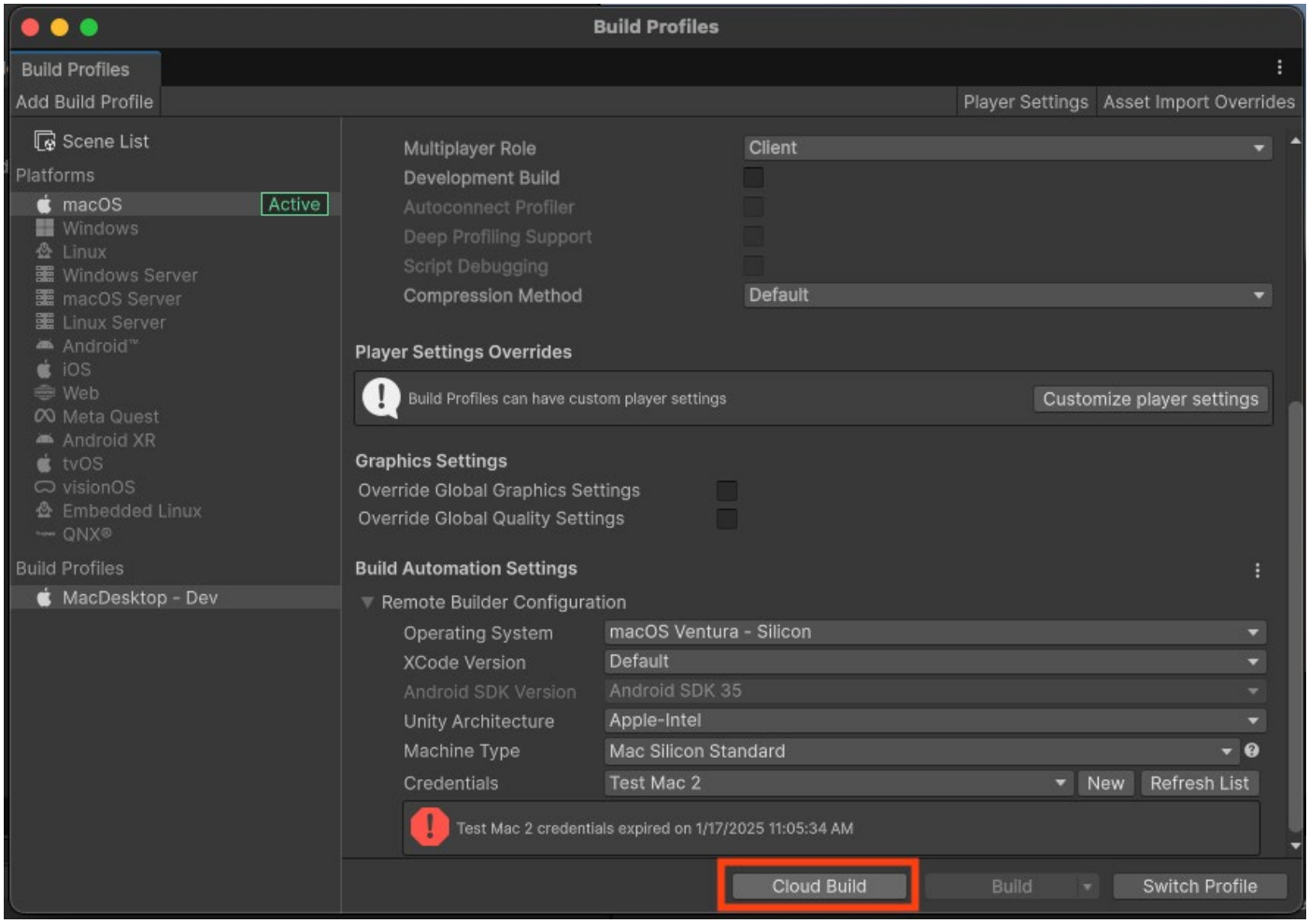


Unity Dashboard의 자격 증명 설정 창

자격 증명을 저장한 후에는 Build Profiles 창으로 돌아가서 **Refresh list**를 클릭합니다. 드롭다운 목록에 자격 증명이 표시될 것입니다. 빌드 설정은 자동으로 빌드 프로파일에 저장됩니다. 이제부터 팀에서 빌드 설정을 재사용하고 공유할 수 있습니다.

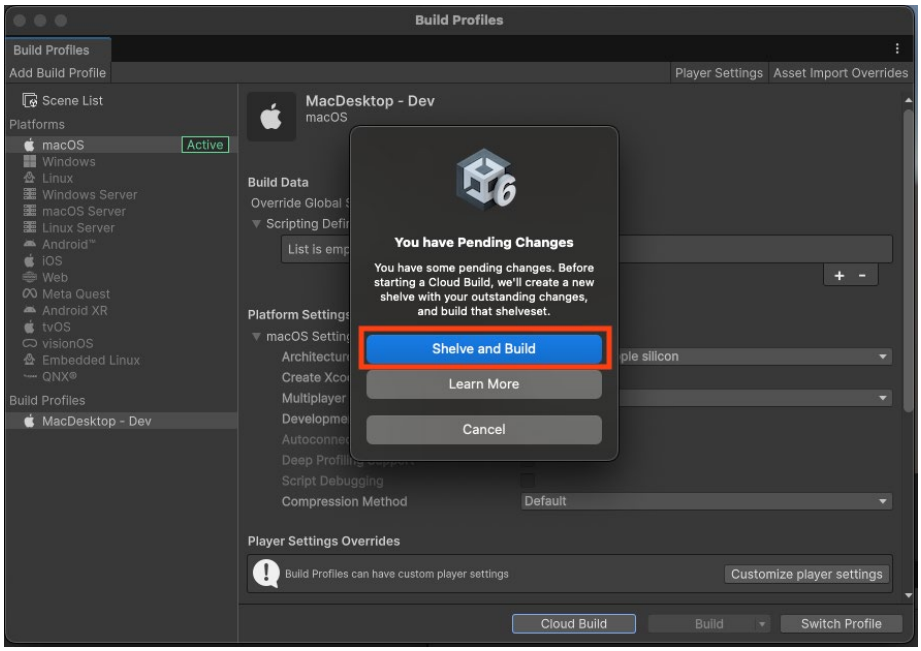
Shelve and Build를 사용하여 진행 중인 작업 테스트

설정을 구성했고 빌드를 실행할 준비가 되었다면 빌드 프로파일에서 **Cloud Build** 버튼을 클릭합니다. 그러면 Unity가 클라우드에서 빌드를 처리하기 시작합니다.



빌드 프로파일의 Cloud Build 설정

프로젝트에 대기 중인 변경 사항이 있으면 **Shelve and Build** 옵션이 포함된 메시지가 표시됩니다. 이 옵션을 사용하면 로컬 변경 사항을 메인 저장소에 푸시하지 않은 상태로 빌드하여 진행 중인 작업을 테스트할 수 있습니다.

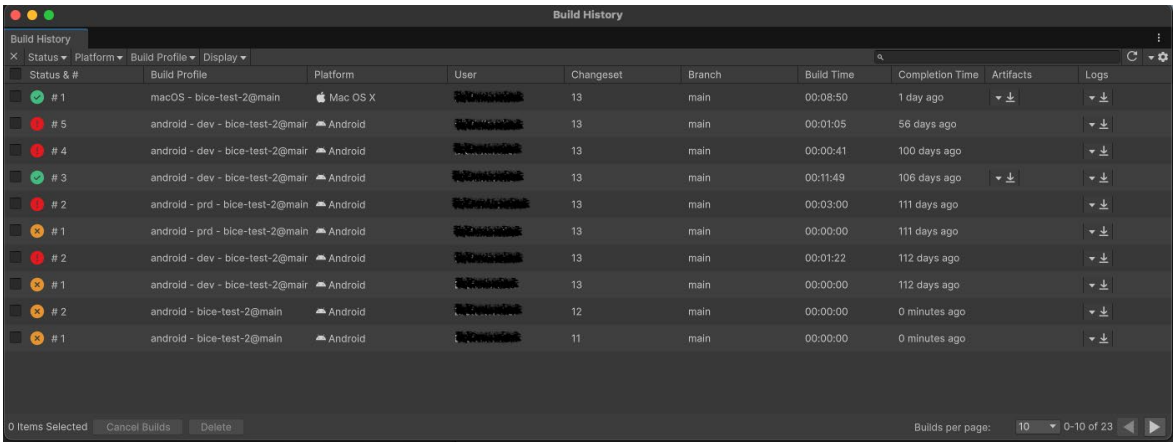


대기 중인 변경 사항의 Shelve and Build 옵션

대기 중인 변경 사항이 없으면 Unity가 현재 체크아웃되어 있는 브랜치의 최신 커밋을 사용하여 빌드를 만듭니다.

Build History를 사용하여 빌드 진행 상황을 추적하고, 문제를 해결하고, 결과 다운로드

빌드가 시작되면 에디터에 **Build History** 창이 열립니다. 여기에서 빌드 진행 상황과 상태를 모니터링할 수 있습니다. 이 뷰에서 로그를 확인하고 빌드 아티팩트에 액세스하여 손쉽게 문제를 해결하거나 결과를 다운로드할 수 있습니다.

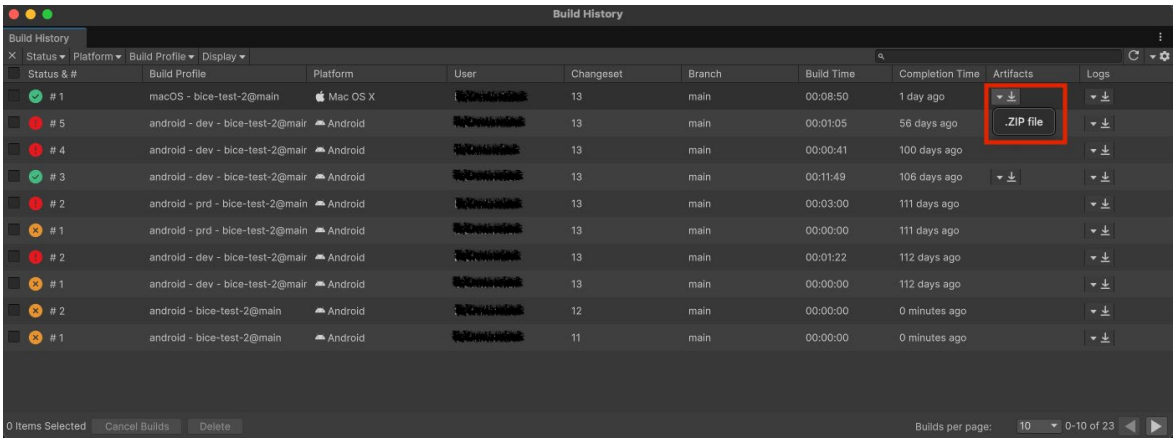


Build History 창

각 상태는 컬러로 구분되어 있어 빠르게 확인하고 필요한 동작(예: 실패한 빌드 재실행, 로그 다운로드)을 취할 수 있습니다. 각 상태 기호의 의미를 알아보려면 이 기술 자료 페이지를 참조하세요.

팀원들에게 빌드 아티팩트 링크 공유

빌드가 성공적으로 완료되면 빌드 아티팩트를 다운로드하거나 팀원들에게 링크를 공유할 수 있습니다.



성공적인 빌드의 빌드 아티팩트 다운로드

빌드가 실패하면 로그를 다운로드해 검토할 수 있습니다.

Unity에서 Build Automation 시작하기(영어) 동영상 튜토리얼과 Unity Learn의 Unity Build Automation: 빠른 시작 가이드 튜토리얼에서 Build Automation에 대해 자세히 알아보세요.

Unity Build Server

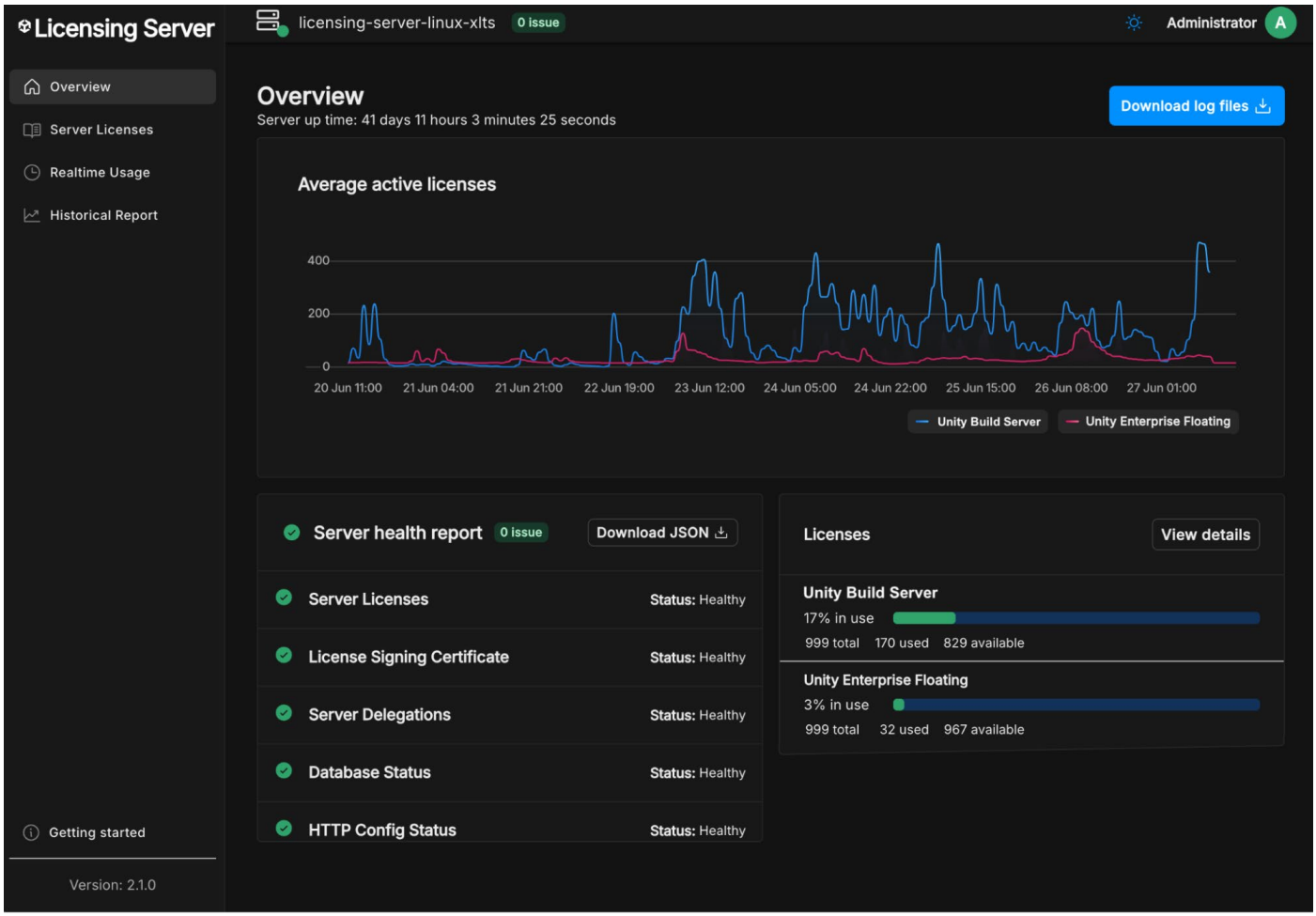
대규모 팀과 조직에서 Unity 프로젝트의 빌드, 컴파일, 패키징 프로세스를 간소화 및 자동화하려면 Unity Build Server를 사용해 보세요.

Unity Build Server는 각 머신별 정식 Unity 에디터 라이선스 없이도 여러 머신으로 빌드할 수 있도록 Unity 유동 빌드 라이선스를 활성화하는 라이선스 관리 툴입니다. 자동화된 빌드 파이프라인과 CI(지속적 통합) 환경에 특히 유용합니다.

Unity Build Server를 사용하려면 네트워크의 중앙 서버에 Unity Licensing Server 소프트웨어를 배포하고 이 서버에 Unity Build Server 라이선스를 추가합니다. 그런 다음 각 빌드 머신이 프로젝트를 빌드해야 하는 경우 Build Server에 빌드 라이선스를 요청하도록 설정합니다.

Build Server 솔루션에서 사용되는 Unity Licensing Server는 동일한 서버에 있는 두 가지 이상의 구독 유형을 지원할 수 있습니다. 고객은 동일한 서버에서 Unity Enterprise Floating 또는 Unity Industry Floating 구독을 사용하여 개발자들에게 유동 라이선스를 배포할 수 있습니다. 아래 이미지에서 예시를 살펴보세요.

Unity Build Server의 요구 사항과 설정 및 시작 방법을 알아보려면 Unity Licensing Server 기술 자료를 참조하세요.



두 개의 활성 구독: Unity Build Server(빌드 머신용) 및 Unity Enterprise Floating(개발자용).

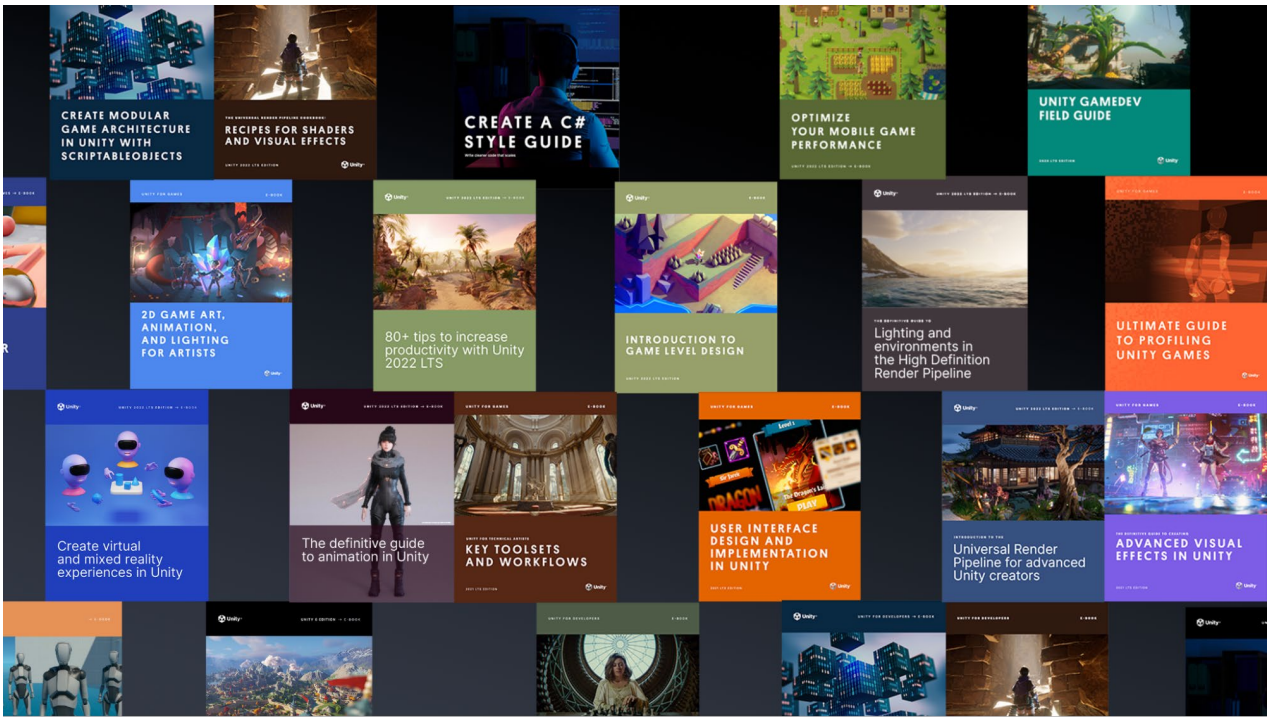
샘플 프로젝트

UI 툴킷 및 UI 빌더 워크플로를 보여 주는 게임 인터페이스가 포함된 샘플 프로젝트부터 각종 디자인 패턴과 프로젝트 아키텍처를 시연하는 샘플, Unity 6의 URP(유니버설 렌더 파이프라인)에서 2D 조명과 시각 효과의 기능을 보여 주는 샘플까지 전부 아래의 프로젝트에서 확인해 보세요.

- 해피 하비스트 - 2D 샘플 프로젝트
- 젬 헌터 매치 - 2D 샘플 프로젝트
- 드래곤 크래셔 - UI 툴킷 샘플 프로젝트
- QuizU - UI 툴킷 샘플
- 패들 게임 스크립터블 오브젝트

- 디자인 패턴 및 SOLID 원칙으로 코딩 스킬 업그레이드
- C# 코딩 스타일 가이드
- 판타지 킹덤(Fantasy Kingdom)
- Shader Graph 샘플
- VFX Learning Templates 샘플 콘텐츠
- URP 3D 샘플
- HDRP 샘플 씬
- HDRP 시간의 유령(Time Ghost)
- VR Multiplayer 템플릿
- MR Multiplayer 템플릿
- UGS 샘플

모든 Unity 사용자를 위한 리소스



Unity 베스트 프랙티스 허브에서 숙련된 Unity 개발자 및 크리에이터를 위한 더 많은 전자책을 다운로드하실 수 있습니다. 업계 전문가와 Unity 엔지니어 및 테크니컬 아티스트가 제작한 30개 이상의 가이드에서 Unity 툴셋과 시스템을 활용한 효율적인 게임 개발 베스트 프랙티스를 살펴보세요.

추가 팁, 베스트 프랙티스, 최신 소식은 Unity 블로그 및 Unity 커뮤니티 포럼에서 확인할 수 있으며, Unity Learn과 **#unitytips** 해시태그를 통해서도 확인할 수 있습니다.

unity.com/kr