



아티스트를 위한 2D 게임 아트, 애니메이션 및 조명 (Unity 6.3 LTS 에디션)



목차

들어가는 말.....	8
도움을 주신 분들.....	10
작성자	10
도움을 주신 유니티 직원 분들	10
Unity 2D: 필수 사항.....	11
새 2D 프로젝트를 여는 방법.....	11
2D 게임을 위한 URP.....	13
URP 설정	13
2D 렌더러 설정	15
2D 아트와 에셋 준비.....	16
기술 관련 고려 사항	16
에셋 해상도.....	17
Photoshop PSD 파일을 Unity로 가져오기	22
드로우 순서.....	24
정렬 레이어.....	24
너무 많은 정렬 레이어 지양	25
Transparency Sort Mode	26
정렬 그룹.....	27
2D에서의 카메라와 원근	29
직교 또는 원근 카메라	29
카메라 스택킹	30
2D용 Cinemachine	31
스프라이트 작업	35
스프라이트 에셋	35

Sprite Editor	36
Sprite Editor	37
Custom Outline	38
Custom Physics Shape	38
보조 텍스처	39
Skinning Editor	40
Sprite Renderer 컴포넌트	40
스프라이트 마스크	42
Sprite Library Asset.	44
Sprite Resolver 컴포넌트	45
2D 타일맵	46
타일맵 생성	47
Rule Tile	53
Unity 6.3의 AutoTile	54
Unity 2D 샘플의 타일맵	55
효율적인 환경 디자인	55
타일맵의 보조 텍스처	56
타일맵을 이용한 레벨 및 게임플레이 디자인	57
팁: 타일맵에서 텍스처 번짐 피하기	58
게임플레이를 위한 Tilemap API	59
해피 하비스트의 Tilemap API	59
젼 헌터 매치의 Tilemap API	61
2D Sprite Shape	62
스프라이트 셰이프 레벨 디자인	65
2D 애니메이션: 골격 애니메이션	67
디자인, 임포트 및 릭	68

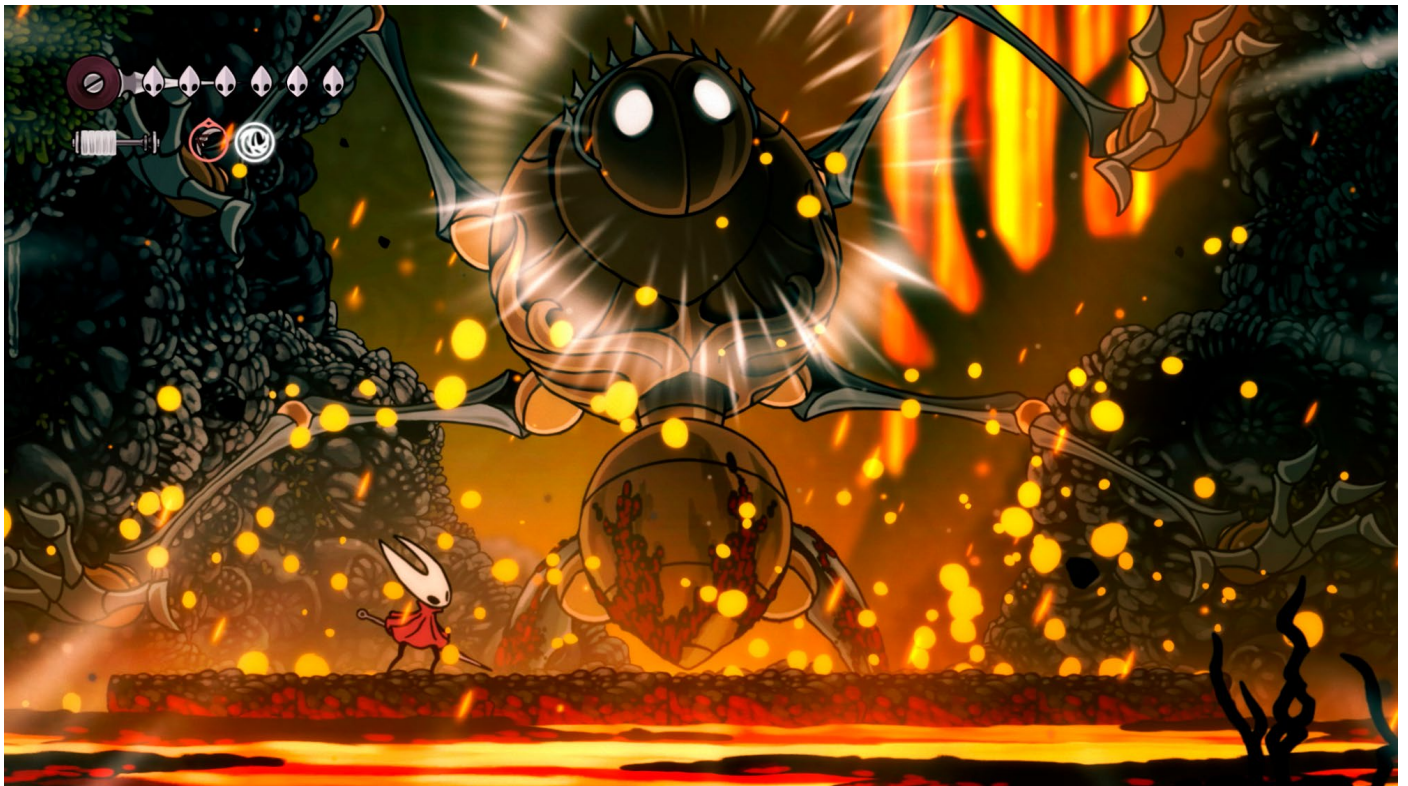
시점	68
캐릭터 스킨 변경.....	70
일반적인 2D 애니메이션 규칙	70
Unity로 캐릭터 임포트.....	71
스프라이트 에디터에서의 리깅.....	73
골격 생성.....	74
스프라이트 지오메트리.....	76
가중치	77
2D IK(역운동학)	79
스프라이트 스왑 및 스킨.....	81
Sprite Swap 오버레이.....	84
스킨	84
애니메이션 기초	86
2D 물리	88
Is Trigger	91
타일맵 콜라이더 2D 만들기	91
충돌 매트릭스	93
리지드바디: 오브젝트가 물리에 따라 움직이게 만들기	93
코드에서 리지드바디 2D 옮기기.....	94
2D 광원 및 그림자.....	96
2D 광원 유형 및 설정	97
보조 텍스처.....	101
노멀 맵	103
노멀 맵에 사용하는 페인팅 기법.....	104
조명 방향을 수동으로 페인팅하기	104
노멀 맵 제너레이터.....	104
Unity에서 그레이스케일로 노멀 맵 만들기.....	105

컬러 샘플링	106
2D 조명용 스프라이트 처리	109
노멀 맵 활성화	109
마스크 맵.....	110
프레넬 효과를 만드는 방법.....	111
2D 그림자	113
2D 광원 및 그림자를 위한 시각적 기법	114
무한 그림자.....	114
블롭 그림자.....	115
낮-밤 사이클	116
프리폼 광원 조작.....	118
Sprite Custom Lit 셰이더.....	120
2D 렌더러에서 2D 광원 사용	122
2D 타일맵	122
2D Sprite Shape.....	123
보조 텍스처가 있는 2D 애니메이션 캐릭터	124
3D를 2D로 렌더링.....	126
호환성	127
3D 및 2D 에셋 정렬.....	128
3D 및 2D 마스킹	129
2D 최적화 팁.....	130
2D 광원 일반 최적화.....	131
2D Light Batching Debugger 사용.....	132
2D 렌더러와 Render Graph를 함께 사용	133
SRP Batcher 사용	134
Sprite Atlas Analyzer 사용.....	135
타일맵 최적화	136

애니메이션 최적화	136
VFX 및 포스트 프로세싱 효과	137
Unity에서 성능 툴을 사용하는 방법 알아보기	137
맺는말.....	139
부록: 2D 시각 효과	140
2D VFX를 위한 Unity 툴셋.....	142
프레임 바이 프레임 애니메이션	142
2D용 VFX Graph	144
VFX Graph 창	145
그래프 로직	147
빌트인 파티클 시스템	149
Shader Graph.....	150
Unity 샘플의 VFX.....	156
해피 하비스트의 비 효과	156
카메라 타일링을 통한 최적화	157
아늑한 모닥불	158
불길 애니메이션	159
물 타일 애니메이션.....	159
굴절 효과.....	160
게임플레이에서의 VFX	160
아이템 사용 시 효과	161
타일로 반복 작업할 때의 효과	161
작은 디테일이 만들어 내는 차이.....	161
젼 헌터 매치의 VFX	162
Camera Sorting Layer Texture	163
드래곤 크래셔의 Shader Graph 예시	164
버텍스 변위.....	164

플로 맵.....	167
유체 프랍 애니메이션용 임계값 애니메이션	168
광원에 글로우 추가.....	170
포스트 프로세싱	171
로컬 볼륨.....	174

들어가는 말



Team Cherry에서 개발한 Hollow Knight: Silksong은 Unity로 제작되었으며 PC와 콘솔에서 플레이할 수 있는 2D 게임입니다.



게이밍 플랫폼, Unity의 2D 툴셋, 기타 그래픽스 개발 소프트웨어의 발전 덕분에 실시간 광원, 고해상도 텍스처, 거의 무한에 가까운 수의 스프라이트를 사용하여 2D 게임을 만들 수 있게 되었습니다. 아티스트는 평면이라는 2D 그래픽스의 특징을 살려, 어떤 기기에서든 멋지게 보일 수 있는 아트 스타일로 게임을 자유롭게 제작할 수 있습니다.

이 전자책은 혼자서, 또는 팀 단위로 상용 2D 게임을 개발하려는 중급 수준의 Unity 경험을 보유한 개발자와 아티스트를 위한 가이드입니다.

유니티 전문가들의 중점적인 도움을 받아 작성된 가장 포괄적인 2D 게임 개발 가이드로, 크리에이터가 Unity 2D 툴셋을 활용하여 최대한의 성과를 내는 데 도움이 될 것입니다.

Unity 6.3 LTS는 현재 정식으로 제작에 사용 가능한 Unity 릴리스이며, 이 2D 가이드에서는 2D 기능 및 시스템의 워크플로에 대한 설명이 업데이트되었습니다. 이 전자책을 통해 학습한 내용은 이후 출시되는 버전의 Unity에도 적용할 수 있습니다.

2D 프로젝트 설정을 위한 기술과 워크플로, DCC 소프트웨어에서 Unity로 스프라이트 가져오기, Unity에서 스프라이트 생성, 정교한 조명과 그림자 및 시각 효과 생성, 타일맵 2D 및 기타 기능으로 레벨 디자인, 애니메이션, 최적화 팁이 포함된 유용한 내용을 살펴보세요.

전자책의 많은 기능에 대한 설명은 Unity의 최근 2D 샘플에서 발췌한 에셋과 씬을 사용하여 제공됩니다. [드래곤 크래셔\(Dragon Crashers\)](#), [해피 하비스트\(Happy Harvest\)](#), [젬 헌터 매치\(Gem Hunter Match\)](#)와 같은 샘플이 있으며, 버니 블리츠(Bunny Blitz)는 2D와 3D가 혼합된 게임으로 2026년 초에 공개 예정입니다. 이러한 게임은 Unity 에셋 스토어에서 무료로 제공되며, 이 전자책과 함께 다운로드하고 학습 참고 자료로 사용하면 좋습니다.



이 가이드가 도움이 되기를 바라며 게임 개발에 좋은 결과가 있기를 기원합니다. 2D 크리에이터의 미래는 밝습니다.

도움을 주신 분들

작성자

야렉 마예프스키는 전문 2D 아티스트 겸 테크 아티스트이며 독학으로 광범위한 C# 스크립팅 기술을 습득한 Unity 개발자입니다. 야렉은 모바일 및 콘솔 게임용 아트를 제작했습니다. 게임 스튜디오를 위한 전문 아트 작업 외에도 현재 자체 게임 [Ultimate Action Hero](#)를 개발 중이며, 이 게임은 [2019 Unity 2D Challenge](#)에서 2위를 차지한 바 있습니다.

야렉은 Unity 2D 샘플의 아트 디렉터이자 아티스트로서 [드래곤 크래셔](#), [해피 하비스트](#), [젼 헌터 매치](#) 및 버니 블리츠의 작업을 담당했습니다.

도움을 주신 유니티 직원 분들

[에두아르도 오리즈](#)가 이 가이드의 제작을 이끌었습니다. 에두아르도는 유니티의 시니어 콘텐츠 마케팅 매니저로, 게임 개발 팀과 함께 다년간 협력한 경험이 있습니다. 2D 툴 팀과 긴밀하게 협력하며, 유니티가 게임 개발자와 스튜디오에 제공하는 서비스를 폭넓게 이해하고 있습니다.

[러스 스캐럴](#)은 유니티 2D 개발 팀의 프로덕트 매니저입니다. 게임 및 소프트웨어 개발 분야에서 15년 이상의 경력을 보유하고 있으며 게임 기술에 대한 폭넓은 지식을 바탕으로 아티스트, 프로그래머, 게임 디자이너가 Unity 2D 툴과 워크플로를 사용할 수 있도록 지원하고 있습니다.

멜빈 메이는 Unity 2D 물리 시스템의 주요 개발자이며, 버니 블리츠 및 2D 물리 샘플 제작에도 핵심적인 역할을 했습니다. 멜빈은 Unity 커뮤니티에서 활발히 활동하며, Discussions 스레드에서 사용자들의 질문에 답변하는 모습을 자주 볼 수 있습니다.

유니티의 다른 2D 개발 팀원들도 이 가이드를 검토하고 검증하는 데 도움을 제공했으며, 2026년뿐만 아니라 그 이후에도 Unity에서 많은 사용자들이 만들어 나갈 2D 게임을 기대하고 있습니다.

Unity 2D: 필수 사항

새 2D 프로젝트를 여는 방법

이 전자책을 따라 진행하려면 먼저 설치 단계를 진행하세요.

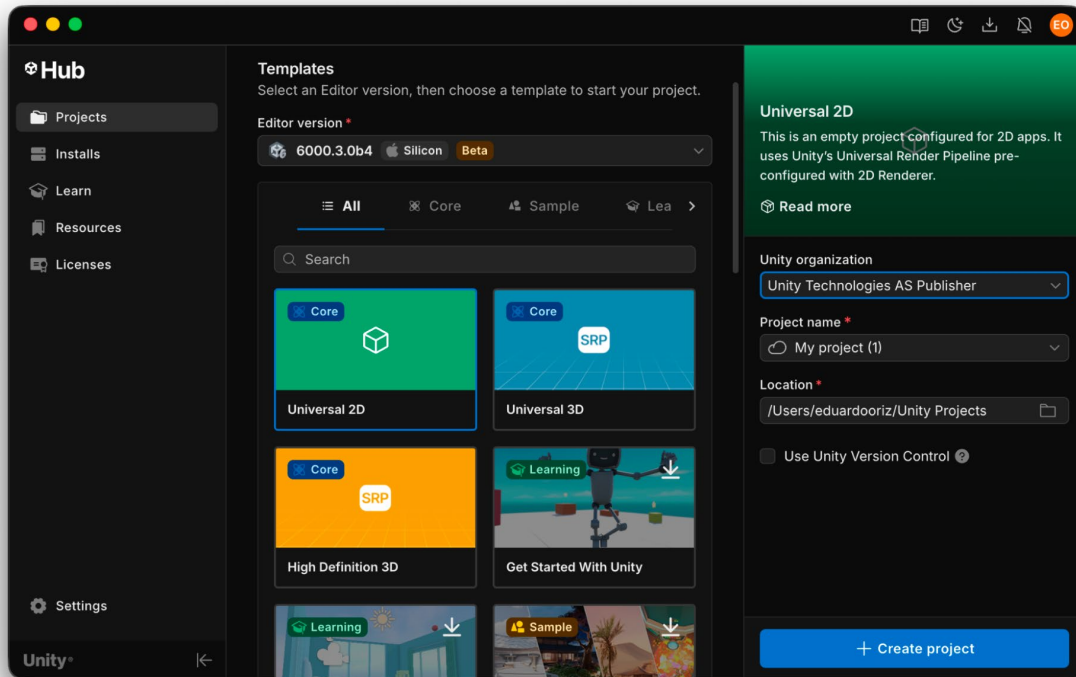
1. Unity Hub에서 최신 버전의 Unity 6를 설치합니다. 이 전자책의 작성 시점 현재 최신 공식 릴리스 버전은 Unity 6.3입니다.
2. Hub에서 유니버설 2D 템플릿으로 새 프로젝트를 생성합니다. 이 템플릿은 URP(유니버설 렌더 파이프라인)를 사용합니다. 2021년 이후 모든 버전의 Unity에서 2D 렌더러는 URP와 연동되도록 구성되어 있습니다.

2D 템플릿에 포함된 여러 패키지는 다음과 같습니다.

- **2D Animation:** 런타임 시 네이티브 골격 애니메이션을 구현하는 데 사용됩니다.
- **2D Aseprite Importer:** .aseprite 파일을 픽셀 아트 생성 툴 Aseprite에서 임포트할 수 있게 합니다.
- **2D Common:** 여러 2D 패키지에서 사용되는 코드를 포함합니다.
- **2D Pixel Perfect:** 다양한 해상도와 비율로 선명한 픽셀 아트를 완성할 수 있는 카메라 컴포넌트를 제공하며, 스네핑, 회전 기반 업스케일링 등의 기능이 포함되어 있습니다.
- **2D PSD Importer:** 여러 레이어로 구성된 .psd 및 .psb 파일을 지원하며, 멀티 스프라이트 캐릭터 애니메이션 작업을 손쉽게 수행하는데 도움이 됩니다.
- **2D Sprite:** 스프라이트 프로퍼티를 구성하는 데 사용되는 Sprite Editor 창입니다.
- **2D SpriteShape:** 셰이프의 외곽선을 따라 스프라이트 타일링을 만들되, 외곽선의 각도에 맞게 스프라이트가 자동으로 변형되고 다른 스프라이트로 교체되도록 합니다.



- **2D Tilemap Editor**: 대규모 그리드 기반 월드를 채우기 위해 타일맵을 생성하고 편집하는 데 사용됩니다.
- **2D Tilemap Extras**: 커스텀 타일 및 브러시와 같은 2D 타일맵과 함께 사용할 추가 스크립트를 포함합니다.



Unity Hub 창

유니버설 2D 템플릿의 여러 프로젝트 설정은 다음과 같이 2D 게임에 최적화되어 있습니다.

- 이미지는 스프라이트로 импорт되어 **Sprite** 모드로 설정됩니다.
- **씬(Scene) 뷰**는 2D로 설정된 상태입니다.
- 기본 씬에는 Global Light 2D가 포함됩니다.
- 기본 Sprite Renderer 머티리얼은 Sprites-Lit-Default입니다.
- 카메라의 기본 위치는 0, 0, -10입니다.
- 카메라는 **Orthographic**으로 설정된 상태입니다.
- **Lighting** 창 설정은 다음과 같습니다.
 - 모든 전역 조명은 비활성화된 상태입니다.
 - Skybox Material은 none으로 설정된 상태입니다.
 - Ambient Source는 Color로 설정된 상태입니다.



2D 게임을 위한 URP

URP는 2D 및 3D 게임을 위한 Unity의 기본 렌더러입니다. Unity 빌트인 렌더 파이프라인의 뒤를 잇는 URP는 Unity가 지원되는 모든 플랫폼에서의 학습과 커스터마이징, 스케일링 등을 효율적으로 진행할 수 있도록 디자인되었습니다. Unity 6.x 사이클에서는 빌트인 렌더 파이프라인과 같은 기능을 제공하며 여러 영역에서 품질 수준과 성능이 향상되었습니다.

URP는 [2D 조명](#)을 위한 그래픽스 파이프라인을 제공하므로 2D 광원 및 조명 효과를 생성할 수 있습니다. 이는 [Sprite Renderer](#), [Tilemap Renderer](#), [Sprite Shape Renderer](#)와 같은 2D 렌더러 및 [MeshRenderer](#) 또는 [SkinnedMeshRenderer](#)와 같은 3D 렌더러와 연동됩니다. 또한 [Shader Graph](#), [VFX Graph](#), [포스트 프로세싱 효과](#) 및 [카메라 스택킹](#)과도 호환됩니다.

URP 설정

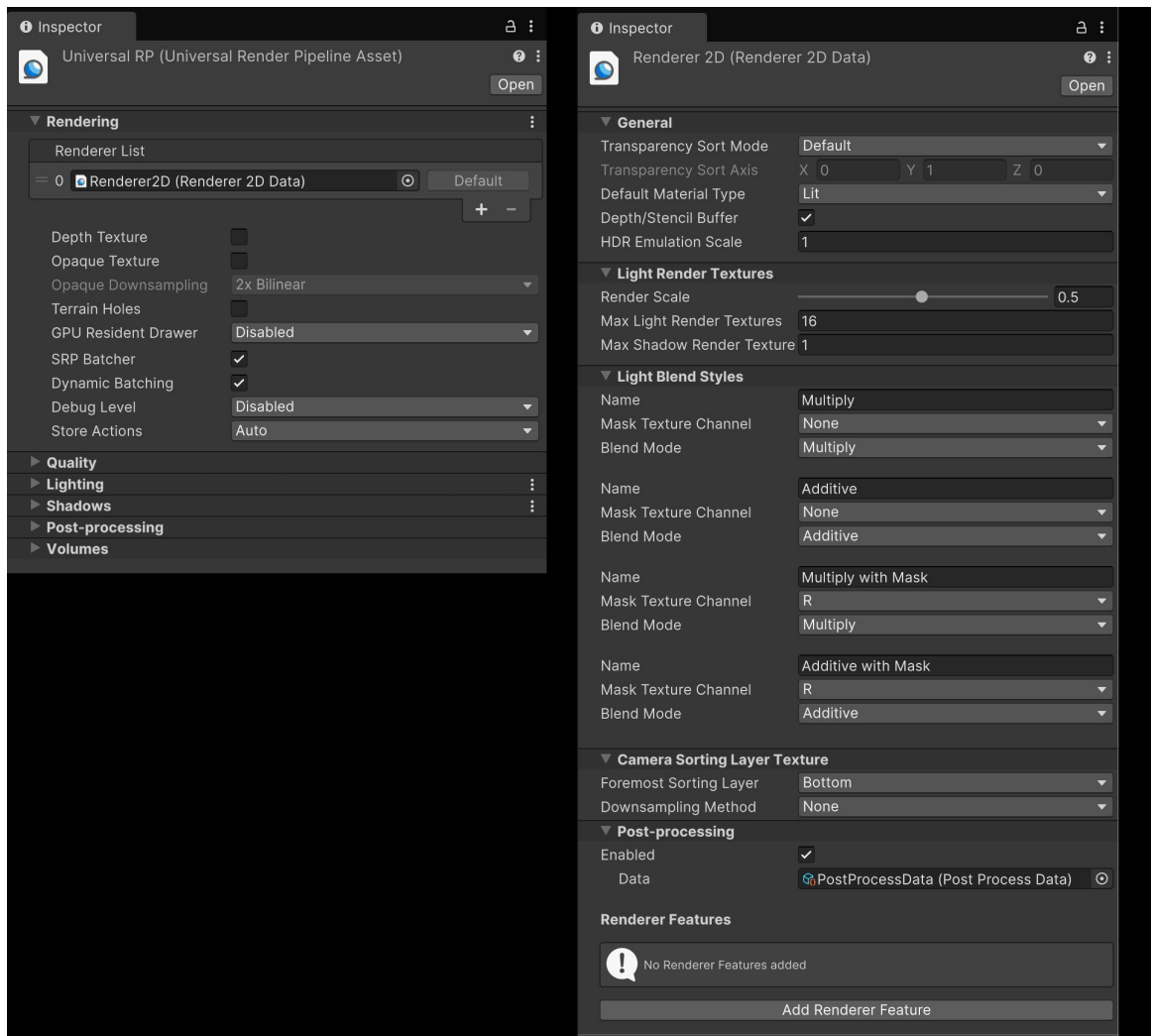
URP의 기본 구성을 사용하게 되는 프로젝트의 두 에셋은 다음과 같습니다.

- [유니버설 렌더 파이프라인 에셋](#)은 품질, 조명(3D), 그림자(3D), 포스트 프로세싱을 위한 설정을 제공합니다.

다른 하드웨어에 서로 다른 설정을 사용하려면 여러 URP 에셋에 걸쳐 품질 설정을 구성하고 필요에 따라 전환할 수 있습니다.

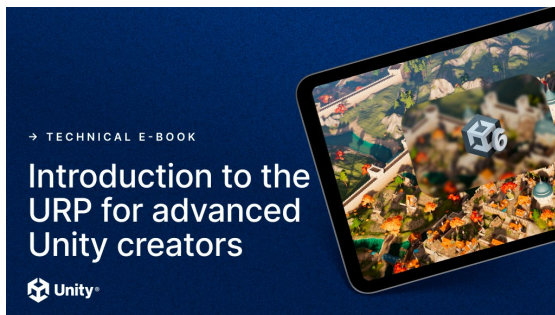
URP 에셋은 **Renderer List**를 통해 **Renderer Data Asset**에 연결됩니다. 새 URP 에셋을 만들면 단일 항목이 포함된 Renderer List가 추가되며, Renderer Data Asset이 이와 동시에 생성되어 기본으로 설정됩니다. 다른 Renderer Data Asset을 이 목록에 추가할 수 있습니다.

- **Renderer Data Asset**은 렌더러가 작동하는 레이어를 필터링하고 렌더링 파이프라인 중간에서 씬이 렌더링되는 방식을 커스터마이징하는 데 사용됩니다. 또한 URP를 위한 상위 렌더링 로직 및 패스를 제어합니다. 포워드 및 디퍼드 경로와 2D 렌더러가 지원되어 2D 광원, 2D 그림자, 광원 블렌드 스타일 등의 기능을 사용할 수 있습니다. URP를 확장하여 자체 렌더러를 만들 수도 있습니다.

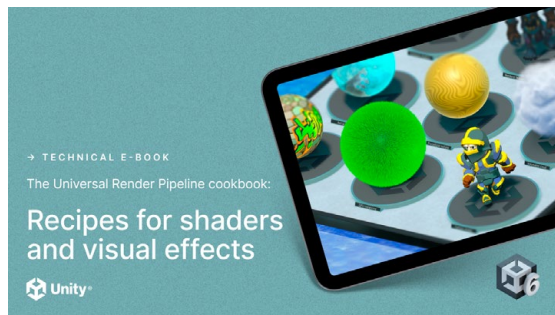


Unity 6 2D 프로젝트의 URP 구성 예셋

이 전자책에서는 URP로 구현 가능한 모든 기술적 및 시각적 기능을 다루지는 않습니다. URP의 전체 기능을 이해하려면, 유니티에서 제공하는 다음과 같은 전자책도 함께 살펴보세요.



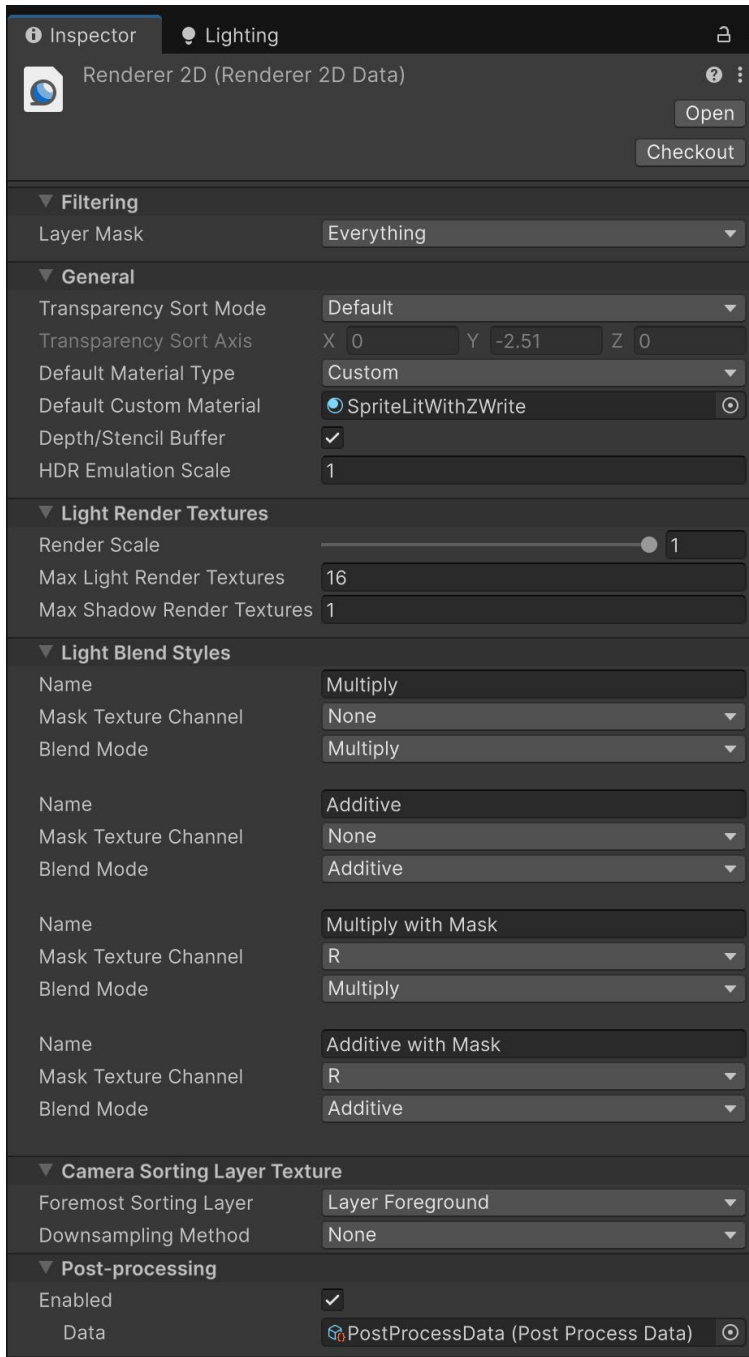
[전자책 다운로드](#)



[전자책 다운로드](#)

2D 렌더러 설정

Unity 6.3의 2D 렌더러 설정에 대한 개요입니다. 자세한 내용은 [기술 자료](#)를 참조하세요.



- **Layer Mask:** 렌더링할 레이어로, 기본적으로 All로 설정되어야 합니다.
- **Transparency Sort Mode:** 사용하는 카메라 유형에 따라 [스프라이트 정렬](#)을 수행합니다.
- **Default Material Type:** 새로운 2D 오브젝트에 추가할 머티리얼을 정의하는 데 유용하며, 2D 광원을 사용하는 경우 Lit, 사용하지 않는 경우 Unlit, 자체 머티리얼의 경우 Custom으로 설정할 수 있습니다.
- **Depth/Stencil Buffer:** 기본적으로 활성화되어 있으며, z 뎀스와 원근을 활용하여 뎀스 효과를 사용할 수 있습니다.
- **HDR Emulation Scale:** 동적 범위를 사용할 때 리매핑에 사용되는 비율입니다.
- **Light Render Textures:** 2D 광원 텍스처를 생성할 때의 비율과 한계입니다. 2D 광원을 사용할 때 성능을 높이려면 값을 낮추세요.
- **Light Blend Modes:** 2D 광원에 사용할 수 있는 네 가지 블렌드 스타일입니다. RGB 값을 더하고, 곱하고, 빼며 마스크 맵 채널에만 영향을 미칩니다.
- **Camera Sorting Layer Texture:** Shader Graph에서 2D 렌더러를 통해 생성된 이미지에 액세스하여 물 반사와 같은 효과를 적용할 수 있습니다.
- **Post-Processing:** 처리한 프레임에 효과를 추가합니다.

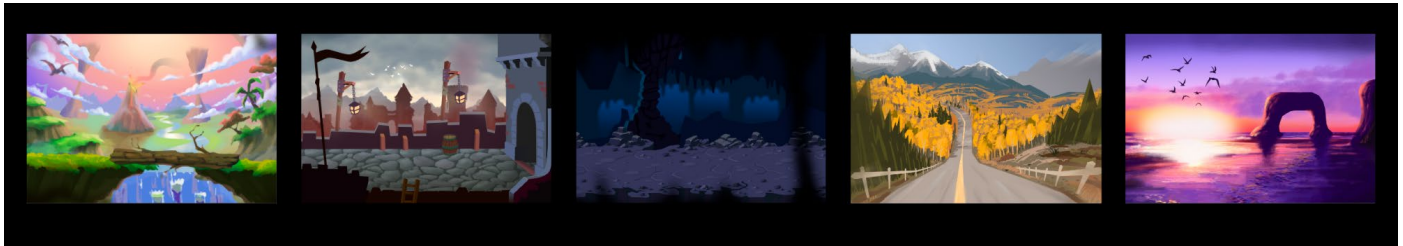


2D 아트와 에셋 준비

아트 구상 단계에서는 프로젝트의 기술적 측면에 영향을 주는 다양한 결정을 내려야 합니다.

게임 디자인에서는 일반적으로 게임의 아트 스타일, 액션, UI를 스크린샷이나 여러 이미지 샘플 형태로 마련합니다. 컨셉을 구현할 수 있을지 빠르게 가능성을 판단하고 게임의 최종 모습을 대략적으로 파악하려면 샘플을 활용하는 것이 유용합니다.

간단한 썸네일 샘플을 여럿 준비해 다양한 아트 방향성을 살펴볼 수도 있습니다. 샘플 단계에서는 간단한 셰이프를 사용하고, 보다 전체적인 모습과 카메라 각도, 오브젝트 크기, 컬러 팔레트, 콘트라스트에 더 중점을 둡니다. 썸네일을 사용하면 테스트 및 반복 작업(iteration)을 위한 다양한 접근 방식을 팀에 제시할 수 있습니다.



간단한 썸네일 제작을 통한 아이디어 테스트

샘플을 만들 때 고려해야 할 사항은 다음과 같습니다.

- 게임의 카메라 각도와 시점은 어떻게 설정하려 하나요?
- 타깃 플랫폼의 화면 크기를 기준으로 플레이어 캐릭터의 크기는 어느 정도인가요?
- 아트 스타일이 타깃 플랫폼, 테마, 타깃층에 적합한가요? 예를 들어 아트 스타일이 캐주얼 게임 플레이어, 젊은 플레이어, 전략 게임 팬층 등의 마음을 사로잡을 수 있을까요?
- 아트 스타일이 전반적인 그래픽스 접근 방식에 부합하나요?
 - 빠른 반응 시간이 중요한 게임플레이를 제작하는 경우 플레이어 캐릭터, 적, 발사체 등의 요소를 배경에서 한눈에 알아보도록 만드는 것이 좋습니다.
 - 모바일 게임은 햇빛 아래에서 작은 화면으로도 잘 보이도록 밝고 콘트라스트가 높아야 합니다.
- UI 요소의 크기, 위치, 가시성은 어떤가요?

기술 관련 고려 사항

샘플 단계에서 중요하게 고려해야 하는 기술 관련 문제도 있습니다. 여기에 포함되는 내용은 다음과 같습니다.

- **애니메이션:** 골격 애니메이션으로 애니메이션화하려는 요소는 무엇이며, 프레임 바이 프레임 애니메이션을 적용하거나 셰이더로 애니메이션화할 요소는 무엇인가요?



- **환경:** 타일맵이나 스프라이트 셰이프로 제작하려고 하나요? 아니면 씬에 플랫폼 스프라이트를 직접 배치할 예정인가요? 샘플을 페인팅해서 해당 툴을 사용한 것처럼 보이게 만들면 게임에 샘플을 바로 활용할 수 있습니다.
- **정렬:** 예상되는 Unity의 정렬 방식과 유사하게 샘플 레이어를 그룹화하여 스프라이트 정렬을 계획할 수 있습니다. 이때 시간이 조금 걸릴 수 있지만 제작 후반에 수천 개에 달할 스프라이트를 정렬하는 수고를 덜 수 있습니다.
- **조명:** 스프라이트에 직접 조명과 그림자를 칠하나요, 아니면 실시간 조명을 적용하나요? 한 가지 팁을 드리자면 이미지를 칠할 때는 조명을 적용하지 않고, 이미지 편집 소프트웨어에서 별도의 레이어에 그림자와 조명을 추가하는 방식이 바람직합니다. 그러면 제작 후반에 조명의 모습을 필요에 따라 언제든지 변경할 수 있습니다.

이 단계에서 시간을 투자하여 게임의 최종 모습과 최대한 비슷해 보이는 에셋을 제작하세요. 그러면 컨셉 구상 단계에서 정식 제작 단계로 더 빠르게 넘어갈 수 있습니다.



드래곤 크래셔 원본 컨셉 아트와 최종 Unity 샘플 버전 비교

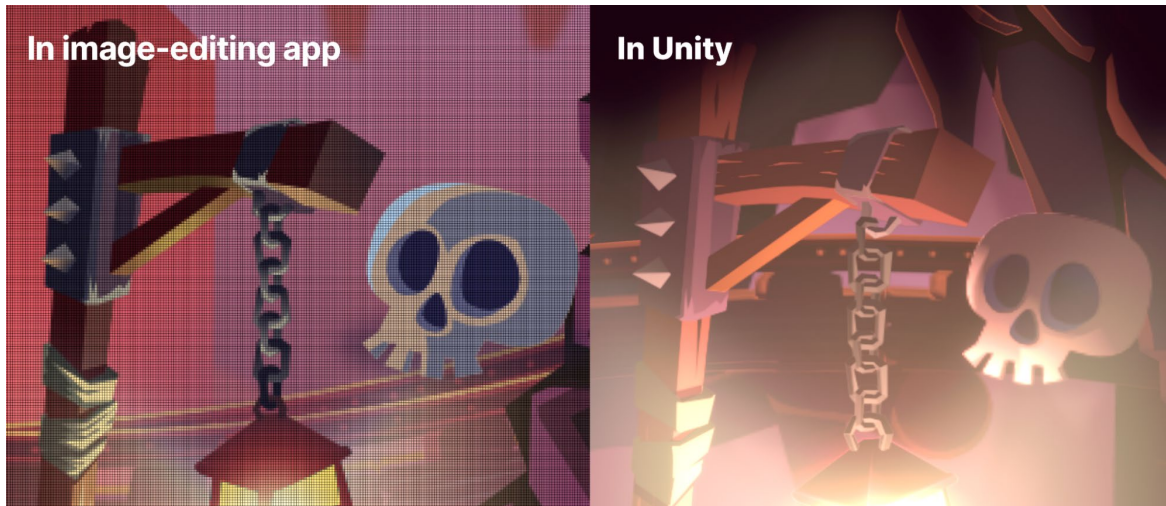
에셋 해상도

Unity 2D 툴은 3D 게임을 위해 제작된 에디터에서 진화했고, 그 결과 몇 가지 독특한 기능을 지니게 되었습니다. 예를 들어 씬 뷰의 2D 스프라이트는 화면 해상도에 완전히 바인딩되지 않습니다. Unity에서 스프라이트는 메시 위에 드로우되는 텍스처이며, 간편하게 크기를 조절할 수 있습니다. 2D 게임의 카메라도 확장할 수 있으며 줌 인/줌 아웃도 자유롭습니다.

따라서 Unity에서 2D 콘텐츠를 제작하려면 Adobe Photoshop, Affinity Photo by Serif, GIMP, Krita 등의 기존 래스터 그래픽스 소프트웨어와는 다른 접근 방식을 사용해야 합니다. 이러한 기존 앱에서는 해상도가



설정된 특정 문서 캔버스 크기를 사용할 수 있으며, 모든 레이어는 이 해상도에 바인딩된 상태입니다. 각 레이어의 픽셀 크기와 문서 픽셀 크기는 1:1 비율입니다.



게임 에셋을 그리는 경우, 픽셀 크기가 일관되고 이미지는 항상 픽셀 그리드로 제한되므로 근사하게 보입니다. 단 Unity에서는 스프라이트 해상도가 달라질 수 있으며, 카메라 줌 레벨도 최종 결과에 영향을 미칩니다.

하지만 Unity에서는 화면 해상도와 에셋 해상도가 서로 독립적이므로 스프라이트 해상도를 계산해야 합니다.

하드웨어의 기능에 따라 설정할 수 있는 최대 해상도가 달라지기 때문에 타깃 플랫폼부터 고려해야 합니다.

모바일 기기의 경우 해상도 범위가 넓지만, 저사양에서 고사양까지 다양한 기기를 타기팅할 수 있다는 점에서 1920x1080을 사용하는 것이 가장 안전합니다.

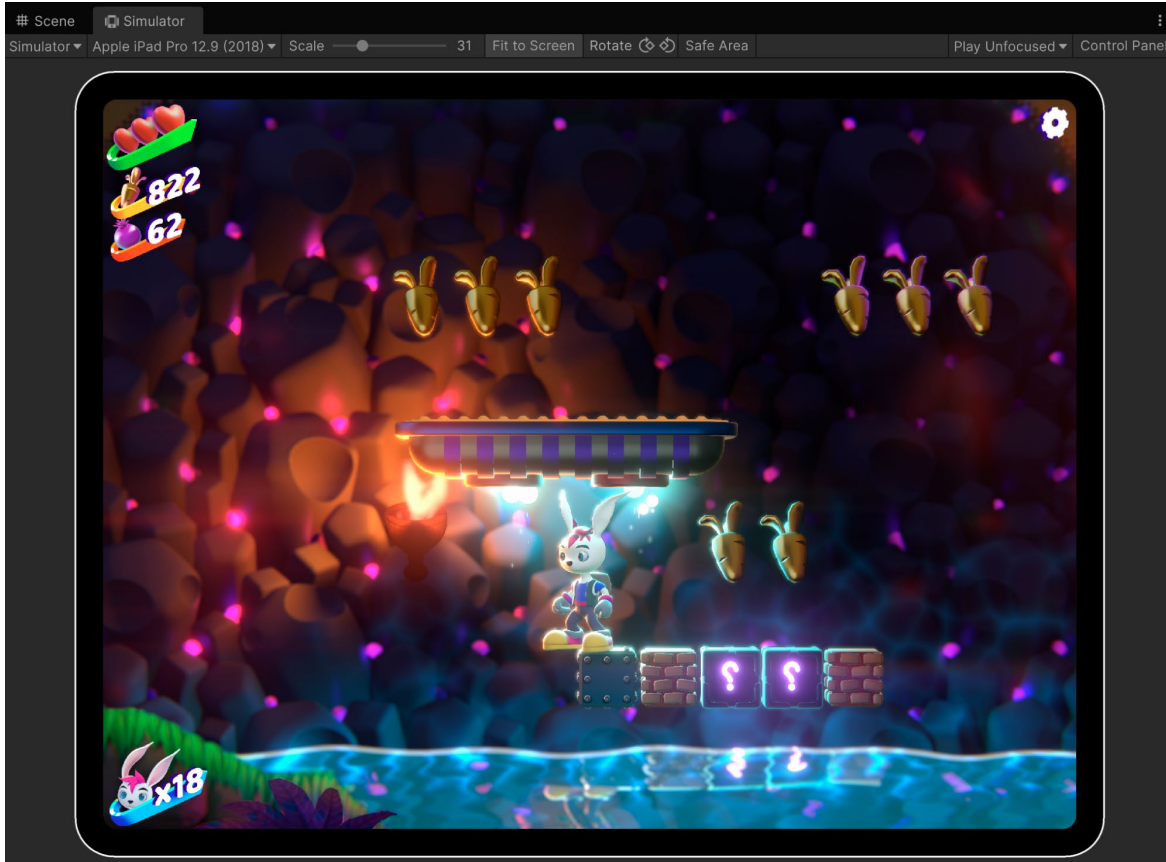
PC의 경우 [Steam의 설문조사](#)에 따르면 데스크톱 게이머의 대다수가 Full HD(1920x1080)를 사용합니다. 울트라와이드 화면을 지원하려는 경우, 21:9 같은 화면 비율을 테스트하여 카메라가 레벨에서 의도치 않은 영역을 표시하지 않는지 확인해야 합니다.

Full HD 또는 4K를 타기팅할 때 적용할 만한 베스트 프랙티스는 다음과 같습니다. 단, 픽셀 아트에는 적용되지 않습니다.

- 타깃 기기에 맞는 최고 해상도로 아트를 제작합니다. 픽셀화가 일어나거나 아트가 흐릿해질 수 있으므로 래스터 아트의 크기를 키워서는 안 됩니다.
- 모든 에셋에 하나의 해상도만 사용하고, 저사양 기기 지원에 필요한 경우 나중에 해상도를 하향 조정합니다.

한 가지 유용한 팁으로, 필요한 크기보다 두 배 큰 아트를 제작하고 Unity로 익스포트할 때 크기를 50%로 줄이는 것이 좋습니다. 그러면 스프라이트가 깔끔하고 선명해 보이며 브러시 선이 흔들리지 않습니다. 아트의 크기를 줄일 것이기 때문에 디테일을 필요 이상으로 살릴 필요는 없습니다. 이렇게 하면 손으로 그린 아트에서 발생하는 사소한 결함을 숨길 수 있습니다. 물론 게임에 직접 그린 비주얼 스타일을 활용하고자 한다면 이 방법을 사용하지 마세요.

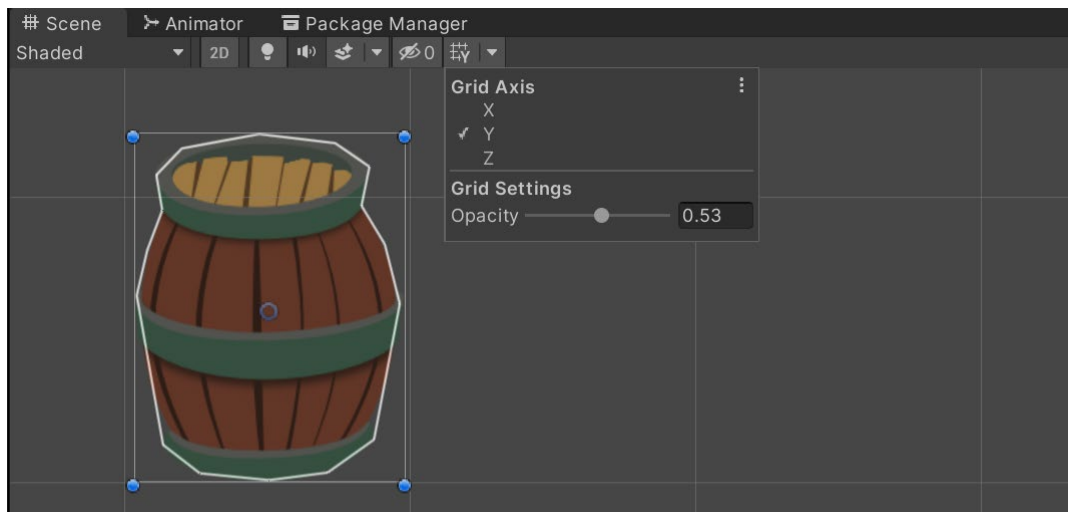
해상도를 선택했다면 게임(Game) 뷰에서 아트를 테스트하며 타깃 기기에서 표시되는 아트의 모습을 확인합니다.



에디터의 게임 뷰와 Simulator 뷰에서 다양한 비율과 해상도로 게임이 어떻게 보이는지 쉽게 미리 확인할 수 있습니다.

간단하게 테스트하려면 원근 뷰 대신 직교 뷰에서 스프라이트 해상도를 계산하세요. 스케일을 1, 1, 1로 설정하고 Z 뎀스를 0으로 설정하여 쉽게 시각화할 수 있습니다.

Unity 그리드와 유닛을 사용하면 스프라이트의 일관된 배치와 형상을 유지하고 카메라 줌과 오브젝트 크기를 계산할 수 있습니다.



씬 뷰의 Unity 그리드

Unity 유닛은 씬 뷰에서 그리드로 시각화됩니다. 1 Unity 유닛이 1미터라고 생각하면 됩니다. 기본 크기를 먼저 설정하고 게임 전체에서 일관성을 유지해야 합니다.

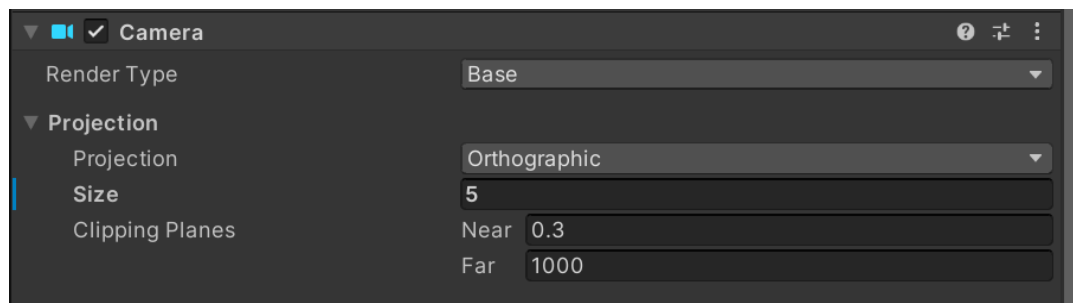
플레이어 캐릭터의 크기부터 설정하는 것이 좋습니다. 플레이어 캐릭터의 키를 0.5~2유닛 사이로 유지합니다. **타일맵**을 사용하는 경우 타일 크기를 1유닛으로 설정합니다.

캐릭터와 오브젝트가 게임의 다른 시각적 요소보다 너무 작거나 크면 트랜스폼 값이 이상해지고 물리 계산에 문제가 발생합니다.

메인 씬에서 기본 오브젝트(플레이어 캐릭터, 적, 수집형 오브젝트, 레벨 위험 요소)의 크기를 설정했다면, 직교 카메라의 Size 프로퍼티를 설정하여 줌 레벨을 선택합니다. 그런 다음 카메라 크기 값을 확인합니다. 카메라 크기 값에 2를 곱하면 카메라의 세로 크기를 Unity 단위로 구할 수 있습니다.



화면에 표시될 게임플레이 영역의 크기를 미리 결정하세요. 예를 들어, 공간이 제한적인 플랫폼어 게임을 개발 중이라면 게임 디자인과 아트 준비를 위해 게임플레이 영역의 크기를 조기에 고정해야 합니다.



직교 카메라의 크기는 중앙에서 맨 위까지 카메라의 수직 크기를 나타내는 단위로 표시됩니다.

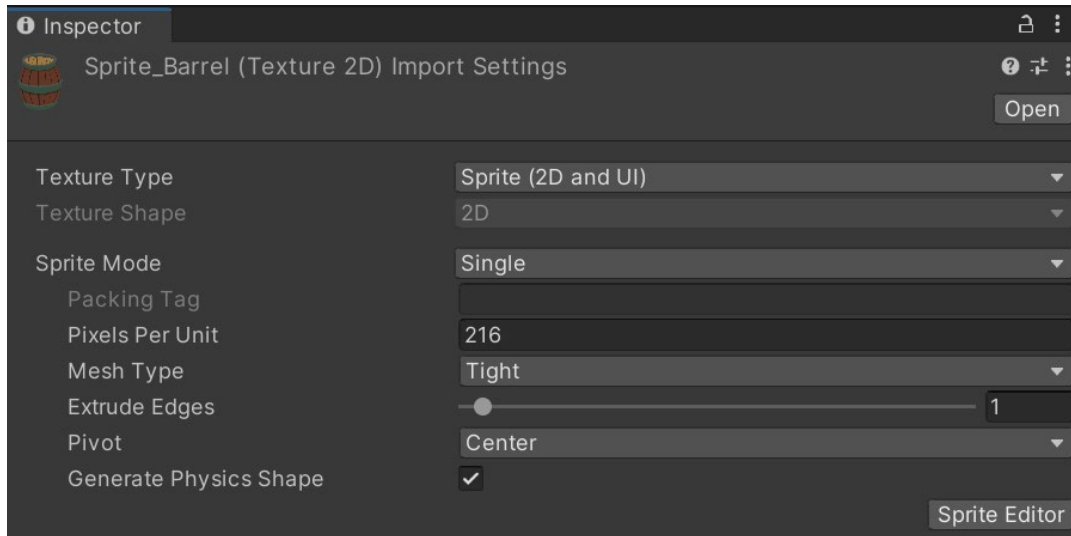


카메라 크기가 5면 화면 중앙에서 위쪽으로 5유닛을 표시하는 것이며, 총 높이는 10유닛이 됩니다. 예를 들어 4K 해상도를 타기팅하는 경우 화면(카메라)의 높이는 2160픽셀입니다. 다음과 같이 간단한 계산을 통해 아트에 필요한 PPU(단위당 픽셀)를 구할 수 있습니다.

$$2160 : 10 = 216$$

$$\text{최대 세로 해상도} : (\text{직교 카메라 크기} * 2) = \text{스프라이트 PPU}$$

따라서 게임의 모든 스프라이트가 네이티브 4K 해상도에서 제대로 표시되려면 약 216PPU를 사용해야 합니다.



스프라이트의 PPU 설정

여기서는 간단한 예시를 들었지만 카메라 줌을 활용하려는 경우, 이 부분도 고려해야 합니다. 직교 뷰에서 최대 줌 비율을 3유닛 크기로 설정했다면 PPU는 계산식($2160 : (3 * 2)$)에 따라 360이 되어야 합니다.

골격 애니메이션을 사용하는 경우 스프라이트의 해상도를 권장 PPU보다 조금 더 높게 설정해야 합니다. 골격 메시가 급격하게 회전하거나, 늘어나거나, 왜곡되거나 할 때 이상하거나 조잡해 보이는 결과를 초래할 수 있기 때문입니다.

기기에서 게임을 테스트해서 어떤 모습인지 확인해 보세요. 고해상도 에셋을 사용한 장점이 뚜렷하게 눈에 띄지 않는 경우가 많습니다. 대신 아트 제작에 걸리는 시간과 기기 메모리를 주인공 캐릭터, 시각 효과, UI 등 더 필수적인 게임 요소에 할당할 수 있습니다. 모바일 기기에서는 게임 용량이 매우 중요하기 때문에 해상도를 낮출 수 있는 에셋을 확인하고 메모리 제한을 염두에 두어야 합니다. 2D 에셋 해상도에 관한 자세한 내용은 이 [블로그 게시물](#)을 확인하세요.



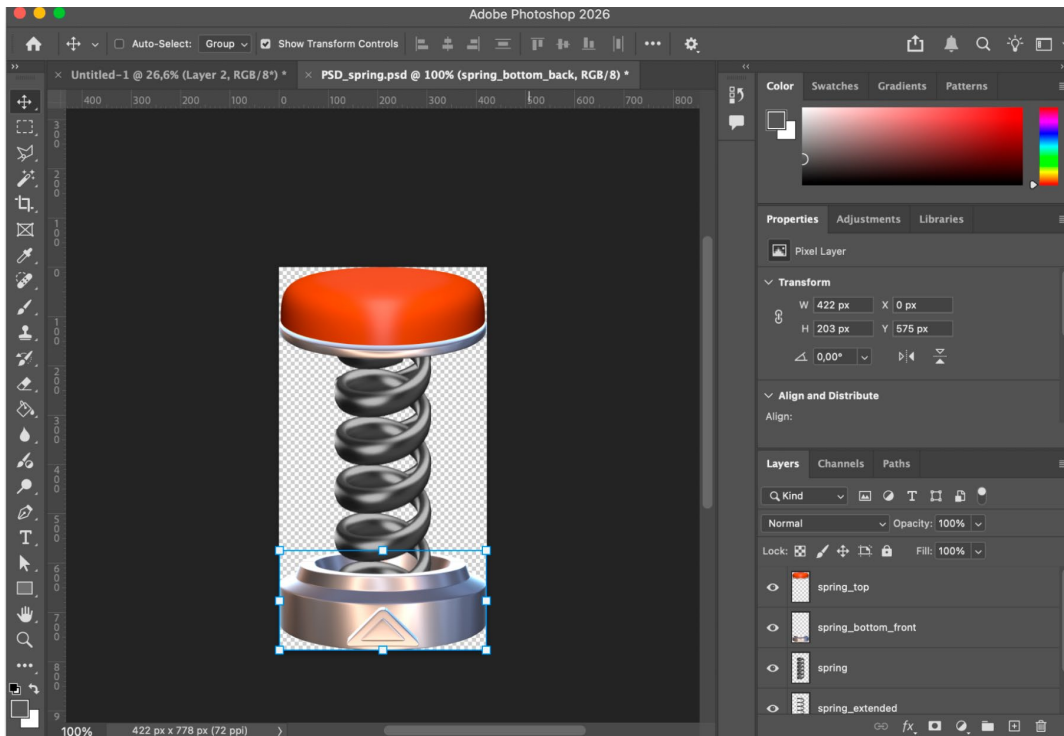
Photoshop PSD 파일을 Unity로 가져오기



PNG 파일 형식으로 익스포트하는 방법이 가장 일반적인 스프라이트 워크플로이지만, PSD 파일을 Unity로 바로 임포트할 수도 있습니다. 기본적으로 Unity는 2D 템플릿에서 프로젝트를 시작하는 경우 2D PSD Importer 패키지가 설치되어 있지 않으면 PSD 레이어를 하나의 이미지로 평면화합니다.

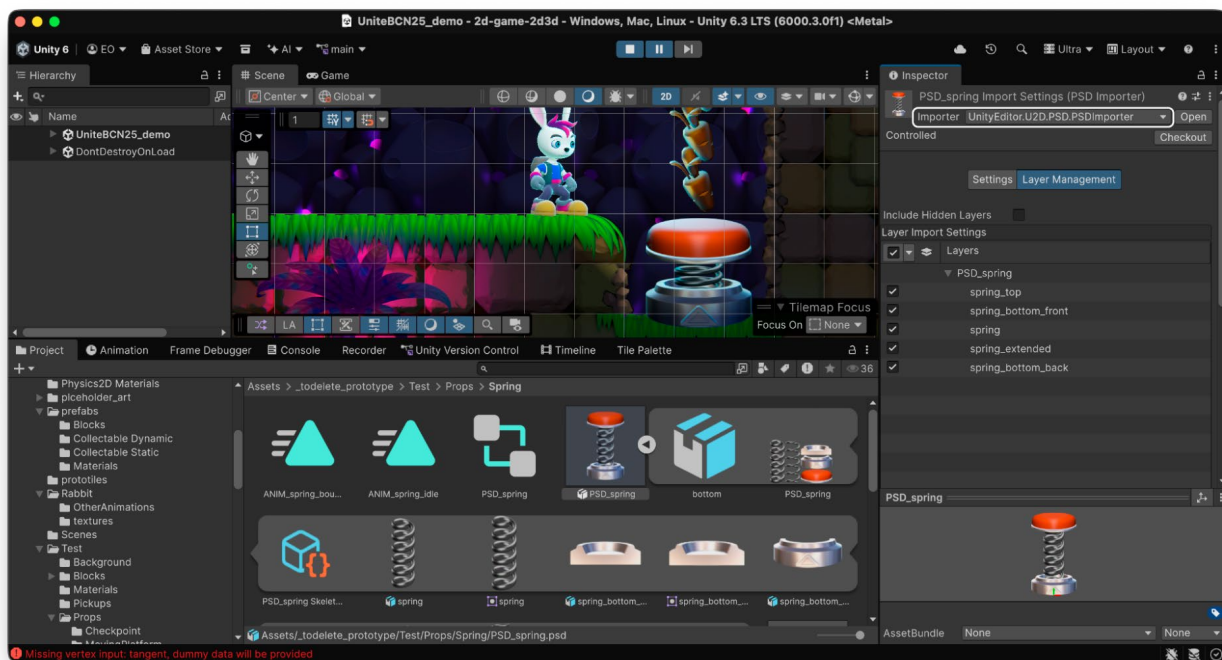
Unity [2D PSD Importer](#) 패키지에는 레이어를 별도의 스프라이트로 임포트하는 옵션이 있습니다. 이 패키지는 2D 애니메이션에서 사용하도록 제작된 것이지만, 하나의 PSD 파일에 포함된 여러 개의 일반 스프라이트를 임포트하는 데 사용할 수도 있습니다.

PSD 레이어는 서로 다른 블렌드 모드, 불투명도, 효과를 가질 수 있으며, Unity에서 적절하게 표시되는 스마트 오브젝트일 수도 있습니다.



Photoshop에서 프랍(prop) 만들기

PSD 워크플로는 게임 스프라이트나 UI 요소에 매우 편리하게 사용할 수 있습니다. 레이어를 편집하고 페인팅하거나 생성한 후에 파일을 저장하면 변경 사항을 에디터에서 즉시 확인할 수 있습니다. Unity 프로젝트에 PSD 소스 파일이 직접 포함되면 이 파일은 [Unity Version Control](#)과 같은 프로젝트의 소스 관리에 포함될 수 있습니다.



중간 스탠드얼론 PNG 파일 없이 Unity에서 사용할 수 있는 레이어와 함께 임포트된 PSD 프랍



드로우 순서

2D 게임에서 모든 스프라이트와 오브젝트는 덤스가 같습니다. 그렇다면 모든 요소를 표시순으로 정렬하려면 어떻게 해야 할까요?

Unity는 렌더러의 유형과 용도를 기반으로 정해지는 우선순위에 따라 렌더러를 정렬합니다. [렌더 대기열](#)을 사용하면 렌더러의 렌더링 순서를 지정할 수 있습니다. 일반적으로는 [불투명 대기열](#)과 [투명 대기열](#)이라는 두 가지 주요 대기열이 있습니다. 2D 렌더러는 주로 투명 대기열에 있으며 [Sprite Renderer](#), [Tilemap Renderer](#), [Sprite Shape Renderer](#) 유형을 포함합니다.

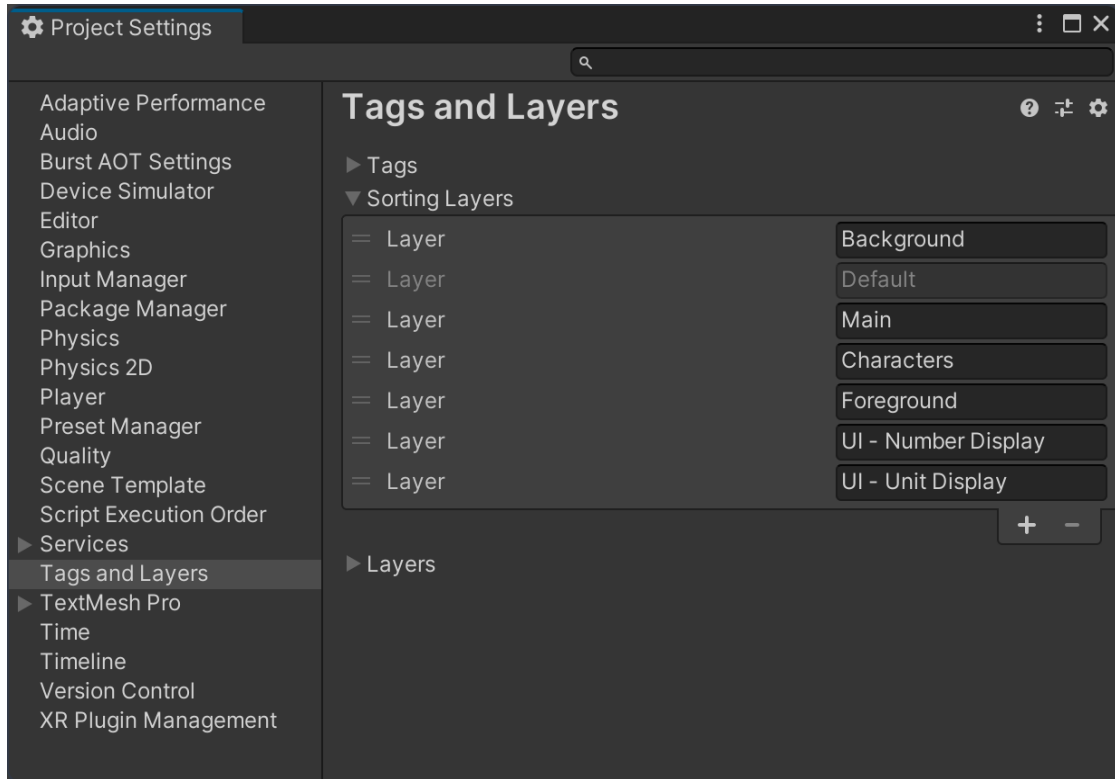
투명 대기열의 2D 렌더러는 보통 우선순위를 따릅니다. 두 개 이상의 오브젝트가 같은 공간을 차지하는 경우 Unity는 이 목록을 확인하고 어떤 오브젝트를 위에 드로우해야 하는지 확인합니다. 우선순위가 같고 두 오브젝트의 값이 같으면 아래 나열된 기준을 평가합니다.

1. 카메라 또는 커스텀 축까지의 거리
2. 레이어 내 순서
3. 정렬 레이어
4. 정렬 그룹
5. 렌더 대기열 지정
6. 머티리얼/셰이더

두 오브젝트가 위의 모든 값이 같아서 순위를 정할 다른 기준이 필요한 경우, 이 과정에서 어떤 오브젝트를 위에 렌더링할지 선택해야 합니다. 이 방법은 이상적인 해결법이 아니므로, 정렬 레이어와 정렬 그룹을 사용하여 정렬 순서를 명확하게 설정해야 합니다.

정렬 레이어

가장 중요한 정렬 기준은 [정렬 레이어](#)입니다. 모든 2D 렌더러에는 이 옵션이 있으며 가장 먼저 설정해야 하는 옵션이기도 합니다. Project Settings를 열고 Tags and Layers 옵션을 지정하면 기본 정렬 레이어를 편집할 수 있습니다.



정렬 레이어 편집

레이어를 추가, 삭제하거나 레이어 왼쪽에 있는 핸들을 드래그하여 순서를 변경할 수 있습니다. 목록에서 더 위쪽에 있는 레이어가 먼저 렌더링되며 카메라에서 더 멀리 표시됩니다.

샘플 디자인 단계에서부터 정렬 레이어 구조를 계획하여 씬을 초기에 정리해야 합니다. 그리고 스프라이트를 씬에 배치할 때 바로 정렬 그룹을 설정하는 것이 좋습니다. 그러면 프로젝트 개발 후반에 수천 개에 달하는 스프라이트의 정렬 설정을 갑자기 변경해야 하는 상황을 방지할 수 있습니다.

너무 많은 정렬 레이어 지양

2D 광원은 정렬 레이어에 의존하므로 정렬 레이어 구조를 설정할 때 조명을 고려해야 합니다. 게임에서 광원이 어떻게 작동하게 할 것인지와 어떤 그룹이 해당 광원의 영향을 받을 것인지 미리 파악해야 합니다.

예를 들어 아이소메트릭 게임을 만들 때 2D 광원을 사용하는 햇불이 있다면 이 햇불은 주변 환경 전체에 영향을 줄까요, 아니면 벽에만 영향을 줄까요? 캐릭터에도 영향을 줄까요, 아니면 캐릭터를 광원 앞에 드로우할 것인가요? 광원의 영향을 받는 오브젝트만을 위한 정렬 레이어가 필요할까요? 정렬 레이어를 편집할 때 이와 같은 질문을 염두에 두고, 2D 광원 섹션에서 조명 계획에 대해 자세히 알아보세요.

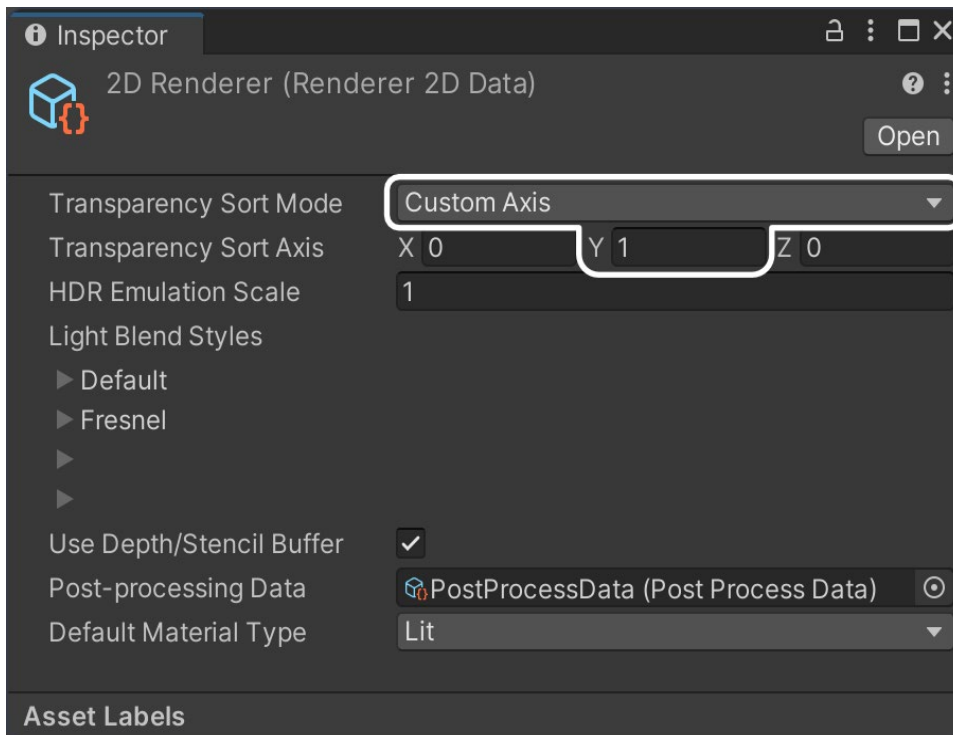


정렬 레이어를 너무 많이 사용하면 세부적인 부분을 모두 관리하지 못할 수 있으니 주의해야 합니다. 렌더러를 더 세부적으로 정렬해야 하는 경우 레이어 내 순서를 활용하는 것이 좋습니다. 그러면 같은 레이어에 있는 오브젝트를 더 쉽게 정렬할 수 있습니다.

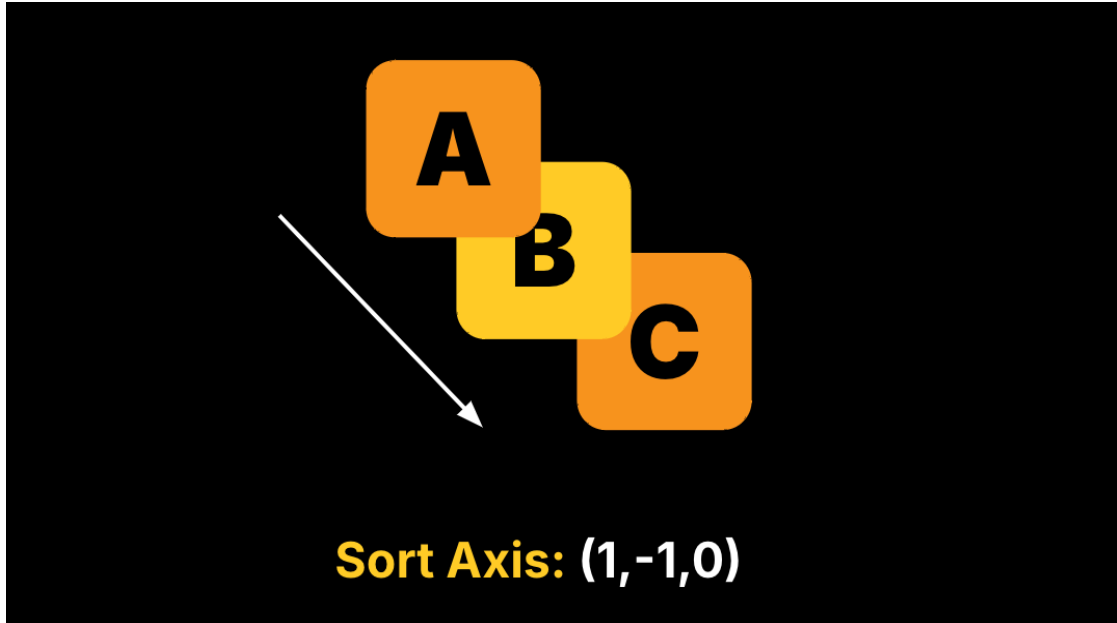
Transparency Sort Mode

직교 카메라를 사용하는 경우 각도를 조금 조정한 아이소메트릭 탑 다운 게임의 축축 투영이나 경사 투영, 배틀 게임의 캐비닛 투영 등 게임에서 유사 3D 뷰를 구현해야 할 때가 있습니다. 그럴 경우에는 카메라까지의 거리를 기준으로 렌더러를 커스텀 정렬해야 합니다.

먼저 오브젝트를 정렬할 커스텀 축을 선택합니다. 탑 다운 게임이나 배틀 게임의 경우, Y축에서 오브젝트를 정렬하면 더 높은 곳에 있는 캐릭터가 다른 캐릭터보다 먼저 렌더링되어 더 멀리 있는 것처럼 보입니다. 이 옵션을 수정하려면 URP를 설정할 때 생성된 2D 렌더러 에셋을 찾아서 Transparency Sort Mode 옵션을 Custom Axis로 변경하고 Transparency Sort Axis 값을 0, 1, 0으로 설정합니다.



Y축에 스프라이트를 정렬하는 과정에서, 값(X:0, Y:1) 1은 정렬 기준을 기반으로 아래쪽 벡터를 따르도록([Vector2.down](#)으로 나타냄) Unity에 전달합니다. 위에 있는 스프라이트를 먼저 드로우하고, 아래에 있는 스프라이트를 나중에 드로우하여 앞쪽에 표시합니다.



스프라이트가 위에서 아래로 렌더링될 방향이 되는 정렬 축으로 모든 벡터를 표현할 수 있습니다.

정렬 그룹

이전 단계를 따르고 Y축에서 오브젝트를 정렬해도 캐릭터의 일부 또는 자식 오브젝트의 형상이 올바르게 표시되지 않습니다.



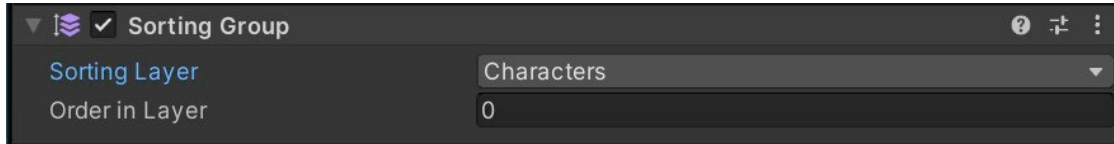
동일한 정렬 레이어에 있는 캐릭터의 부분이 두 캐릭터 간에 섞여 있습니다.

이는 버그가 아닌 정렬의 작동 방식 때문입니다. 정렬 기능에서는 오브젝트가 같은 정렬 레이어에 있다면 어떤 부분이 어떤 캐릭터에 속해 있는지 파악하지 못합니다. 다행히 정렬 우선순위의 네 번째 기준이 되는 정렬 그룹을 활용할 수 있습니다.



Sorting Group은 정렬을 위해 공통의 루트를 공유하는 렌더러를 그룹화하는 컴포넌트입니다. 같은 Sorting Group에 있는 모든 렌더러는 Sorting Layer, Order in Layer, Distance to Camera를 동일하게 공유합니다.

여러 스프라이트로 구성된 캐릭터나 기타 오브젝트를 사용할 때는 이 컴포넌트를 부모 오브젝트(계층 구조(Hierarchy)의 가장 위쪽)에 두고, 다른 2D 렌더러와 마찬가지로 Sorting Layer와 Order in Layer를 설정합니다. 이 작업의 예시는 [Unite Now 기술 세션](#)에서 확인할 수 있습니다.



부모 게임 오브젝트의 정렬 그룹을 생성하면 해당 오브젝트와 자식 오브젝트가 단일 요소로 정렬되어 해당 부분과 게임 내 다른 스프라이트 간의 충돌을 방지할 수 있습니다.



정렬 그룹을 통해 의도한 대로 캐릭터를 표시할 수 있습니다.

정렬 그룹을 중첩할 수도 있습니다. 여러 스프라이트로 구성된 무기를 무작위로 생성하려는 경우 정렬 그룹에 배치해야 캐릭터의 손에 무기를 올바르게 표시할 수 있습니다.

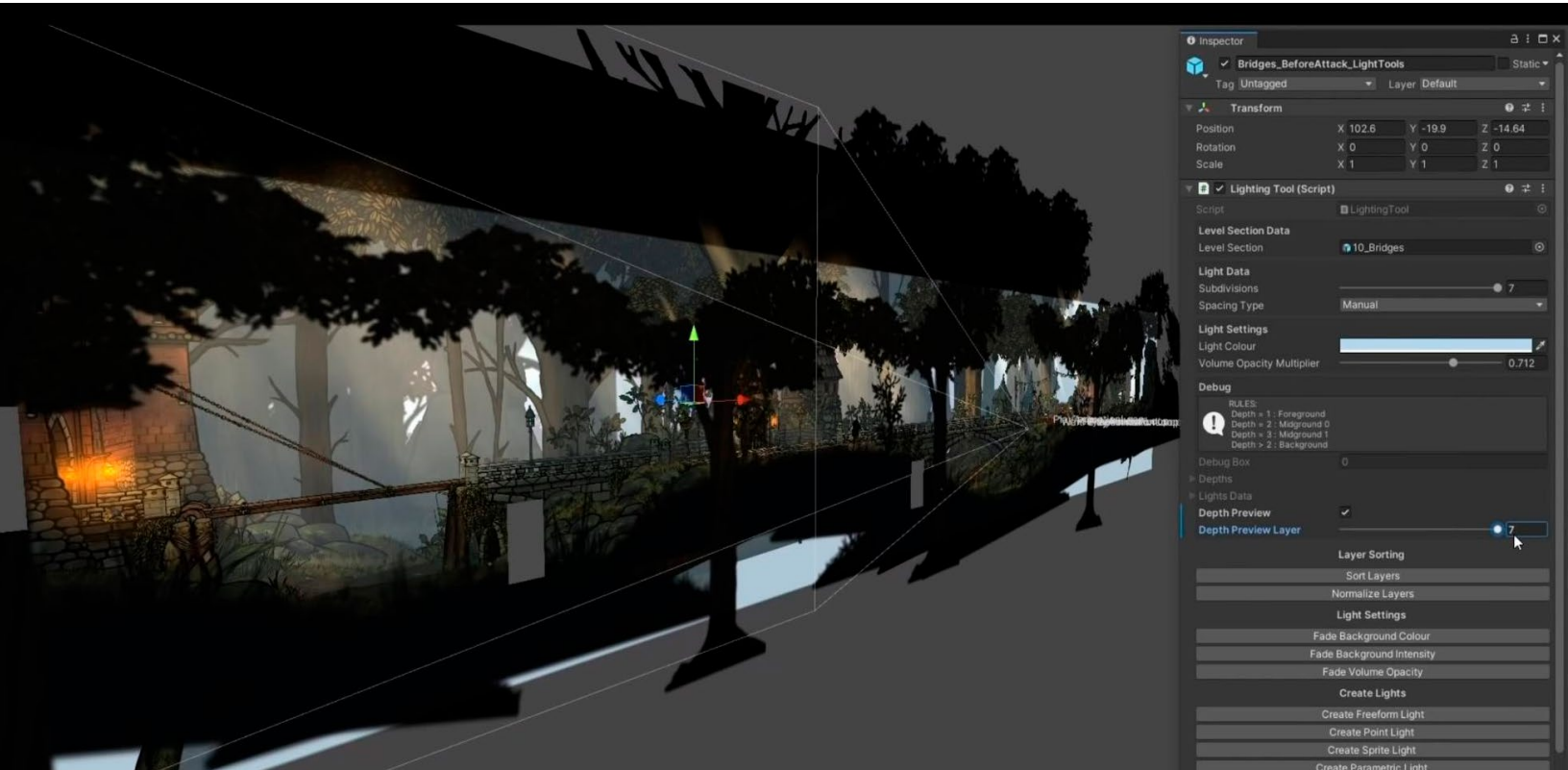


2D에서의 카메라와 원근

직교 또는 원근 카메라

대부분의 2D 게임에서는 카메라를 직교 모드로 설정하는 것이 좋습니다. 2D 프로젝트에서 직교 카메라를 주로 사용하는 이유는 1차원의 선이 평행하고 2차원에서만 작업하기 때문입니다. 직교 모드는 픽셀 아트 게임, 퍼즐, 2D 아이소메트릭, 탑 다운, 플랫폼 등 모든 스타일에 사용할 수 있습니다.

2D 게임에는 실질적인 뎁스가 없지만 패럴랙스 효과를 사용하여 뎁스가 있는 것처럼 만들 수 있습니다. 패럴랙스 효과는 카메라가 움직일 때 배경과 전경에 있는 여러 레이어를 다른 속도로 스크롤합니다. 멀리 있는 오브젝트가 가까이 있는 오브젝트보다 더 느리게 움직이는 것처럼 보이는 사람의 깊이 지각력을 모방하는 것입니다.



Oddbug Studio의 Tails of Iron에 적용된 원근 카메라 설정입니다. [Unity 2D 광원](#)으로 게임에 조명을 어떻게 구현했는지 알아보세요.

직교 모드가 모든 2D 게임 스타일에 적합하다면 원근 모드를 사용하는 이유는 무엇일까요?

이름에서도 알 수 있듯이, 첫 번째로 원근 카메라는 원근 시점을 사용하기 때문에 패럴랙스 효과를 바로 사용할 수 있습니다. 패럴랙스 레이어처럼 스크롤을 처리하기 위한 스크립트를 사용할 필요가 없습니다. 대신 카메라에서 더 멀리 떨어진 요소를 Z축에 배치하고, 카메라와의 상대적 거리에 따라 원근을 고려하여 스케일을 조정하면 됩니다.

카메라 선택이 아트 제작 과정에 영향을 주지는 않지만 레벨 설정 방식과 게임의 디자인에는 영향을 줍니다.

요약하자면, 직교 모드는 고급 스크롤링과 패럴랙스를 사용하지 않는 한 가장 일반적인 선택이 됩니다. 고급 스크롤링과 패럴랙스를 사용하는 경우에는 원근 모드를 선택하는 것이 좋습니다.

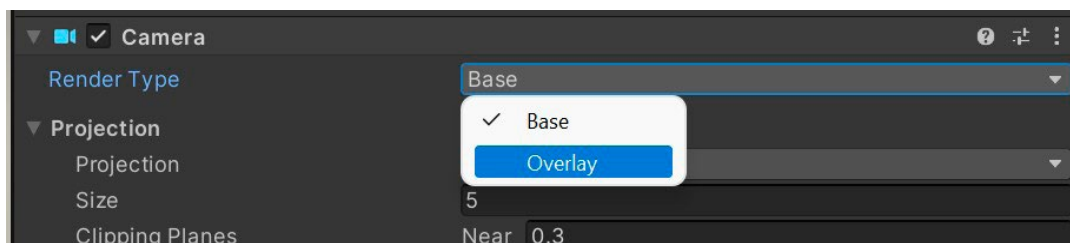
카메라 스테킹

URP에서 **카메라 스테킹**을 사용하면 카메라 출력을 하나로 레이어링하여 하나의 최종 이미지로 결합할 수 있습니다. 카메라 스테킹을 통해 2D, 3D, UI 오브젝트를 결합할 수 있습니다.



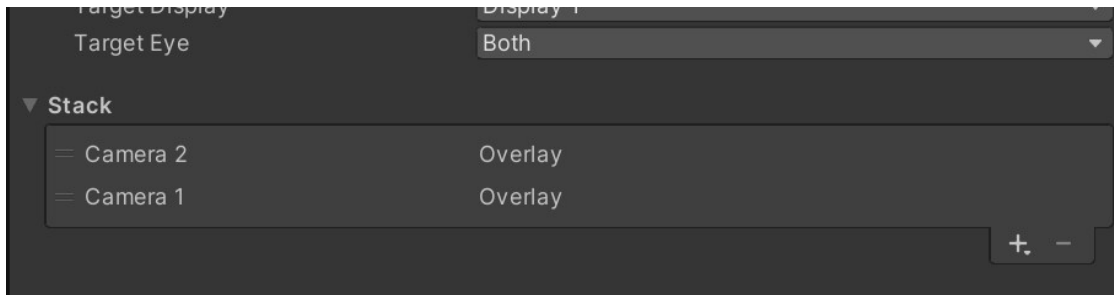
카메라 스테킹의 사용 사례로는 메인 카메라 위에 맵의 다른 영역을 표시하는 것이 있습니다. 예를 들어, 두 번째 플레이어가 맵의 다른 부분을 분할 화면처럼 둘러볼 수 있도록 하거나, 이 경우 썸의 조감도를 제공하는 것입니다.

스테킹을 위해서는 씬에 최소 2개 이상의 카메라가 필요합니다. 먼저 렌더링할 이미지를 위해 한 개의 카메라를 Base로 선택하고 나머지 카메라는 Overlay로 설정합니다.



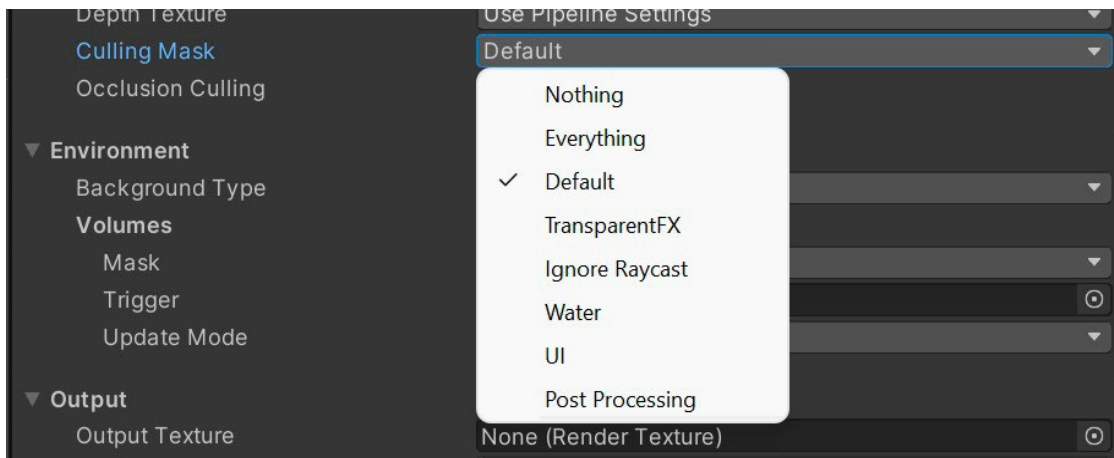
카메라 스테킹 설정

Base 카메라는 어떤 카메라가 어떤 순서에 따라 위에 오버레이되는지 알아야 합니다. 순서를 정하려면 Stack 옵션으로 이동해 + 버튼을 눌러서 오버레이할 카메라를 선택합니다. Overlay 카메라를 목록에서 위아래로 드래그해서 원하는 대로 렌더링 순서를 변경할 수 있습니다.



오버레이로 렌더링할 카메라 선택

레이어를 사용하여 각 카메라에서 렌더링할 오브젝트를 설정합니다. 기본적으로 카메라는 모든 레이어를 렌더링합니다. 하지만 Camera 컴포넌트의 Culling Mask 옵션에서 카메라가 렌더링할 레이어를 선택해서 이 설정을 변경할 수 있습니다.



카메라의 Culling Mask 옵션에서 렌더링할 레이어 선택

카메라가 렌더링할 게임 오브젝트를 선택하고 게임 오브젝트의 레이어를 앞서 설정한 레이어로 변경해 보세요. 그러면 오브젝트를 서로 다른 카메라에서 렌더링하도록 3D 배경 레이어, 2D 플랫폼 레이어 등으로 분류할 수 있습니다.

2D용 Cinemachine

카메라 유형을 선택했으면 카메라가 게임플레이를 따라가도록 설정해야 합니다. Unity의 [Cinemachine](#) 시스템은 이를 구현하기 위한 기능 외에도 카메라 레벨 경계 제한, 카메라 전환, 노이즈 설정 등 다양한 기능을 제공합니다. 이 섹션에서는 2D 게임을 위한 몇 가지 핵심 Cinemachine 기능을 중점적으로 살펴봅니다.



우선 패키지 관리자에서 Cinemachine을 설치합니다. 그리고 프로젝트의 씬 뷰에 Cinemachine 카메라를 추가하는 방법은 다음과 같습니다.

1. 씬 뷰에서 Cinemachine 카메라로 프레임을 맞추고자 하는 지점을 찾습니다.
2. Unity 메뉴에서 **GameObject > Cinemachine > Cinemachine Camera**를 선택합니다.
다음과 같은 요소가 포함된 새로운 게임 오브젝트가 추가됩니다.

- Cinemachine Camera 컴포넌트
- 씬 뷰 카메라의 최신 위치 및 방향과 일치하는 트랜스폼

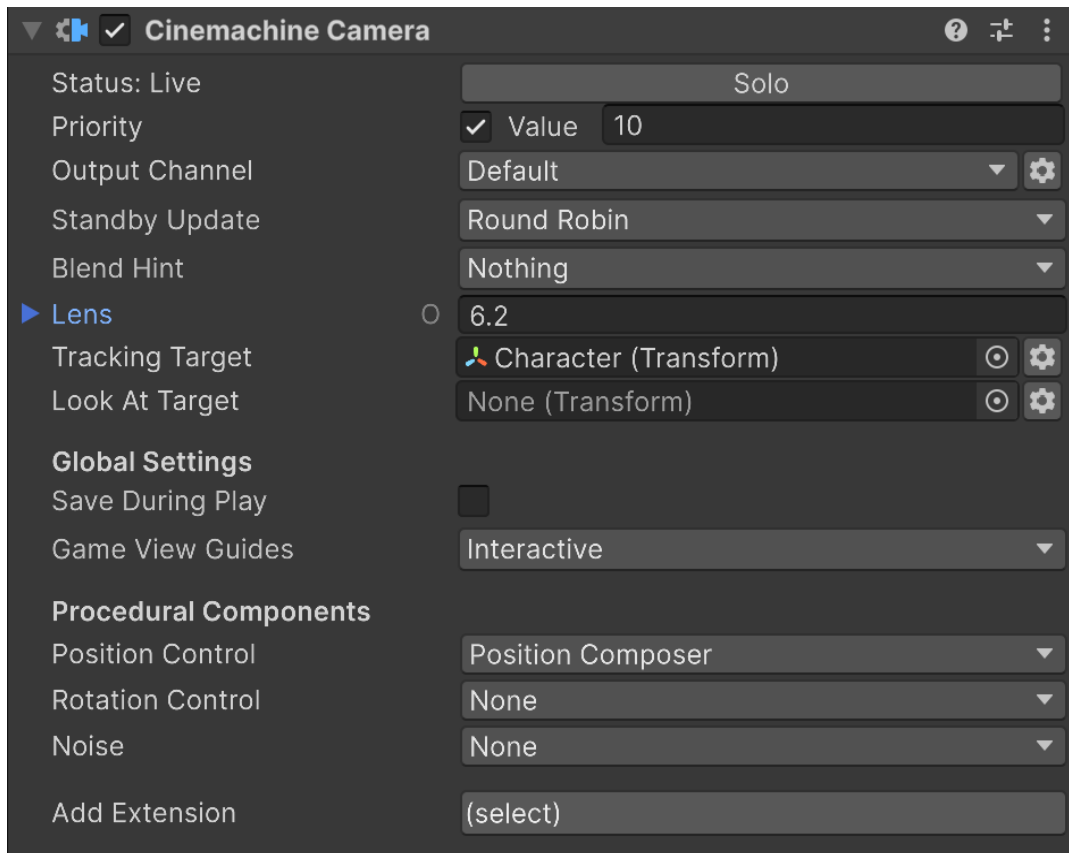
씬에서 첫 번째 Cinemachine Camera를 생성할 때, Unity는 Unity 카메라에 Cinemachine Brain 컴포넌트를 자동으로 추가합니다(Unity 카메라에 이미 Cinemachine Brain 컴포넌트가 포함되어 있지 않은 경우에 한함). 이를 확인하는 방법은 다음과 같습니다.

1. 계층 구조에서 Unity 카메라를 선택합니다(Camera 컴포넌트를 포함하는 게임 오브젝트).
2. 인스펙터(Inspector)에서 게임 오브젝트가 Cinemachine Brain 컴포넌트를 포함하는지 확인합니다.

Cinemachine Brain은 씬에서 작동 중인 모든 가상 카메라를 모니터링합니다. 키프레임을 통해 가상 카메라에 애니메이션을 추가하거나, 카메라를 블렌딩하거나, 카메라를 자연스럽게 전환하거나, 두 카메라를 조합하여 한 카메라에 애니메이션을 추가하면서 다른 카메라로 전환할 수 있습니다. 모든 애니메이션은 Cinemachine Brain에서 처리하여 메인 카메라에 적용됩니다. 메인 카메라를 움직이는 강력한 애니메이션 시스템이라고 생각하면 됩니다.



2D 가상 카메라를 생성하려면 **Cinemachine > Create 2D Camera**를 클릭합니다. 그러면 2D 환경에 맞게 설정된 가상 카메라가 생성됩니다. 씬에서 처음으로 가상 카메라를 생성한 경우 메인 카메라에 Cinemachine Brain 컴포넌트가 추가됩니다.



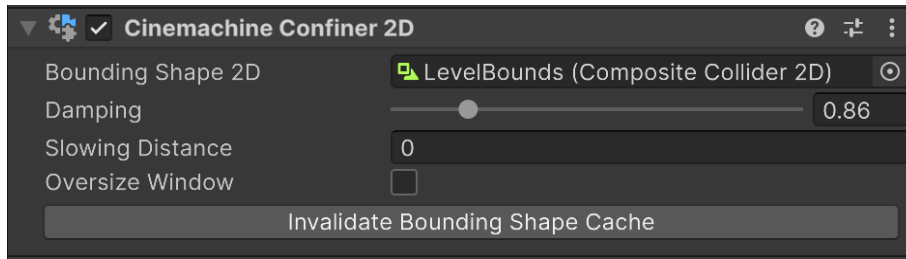
Cinemachine 가상 카메라 생성

Cinemachine에는 따라갈 오브젝트가 필요하기 때문에 Tracking Target 필드에 플레이어 캐릭터를 할당합니다. 2D 평면에서 작업하고 있고 카메라가 이동하거나 기울어지지 않았는지 확인하려면 Look At Target 필드가 비어 있고 Position Control이 Follow 또는 Position Composer로 설정되어 있는지 확인하세요. 마지막으로 프로젝트에 맞게 Lens 프로퍼티를 설정합니다. 일부 옵션은 메인 카메라의 프로퍼티에서 상속된다는 점을 유의해야 합니다.

Position Composer 컴포넌트에서는 Offset, Damping, Dead, Soft Zones 등 가상 카메라가 타겟을 따라가는 방식을 수정하는 데 유용한 옵션을 찾을 수 있습니다. 플레이 모드에서 다양한 옵션을 테스트할 수 있으며 Save During Play 옵션을 선택하면 테스트하면서 변경 사항을 저장할 수 있습니다.

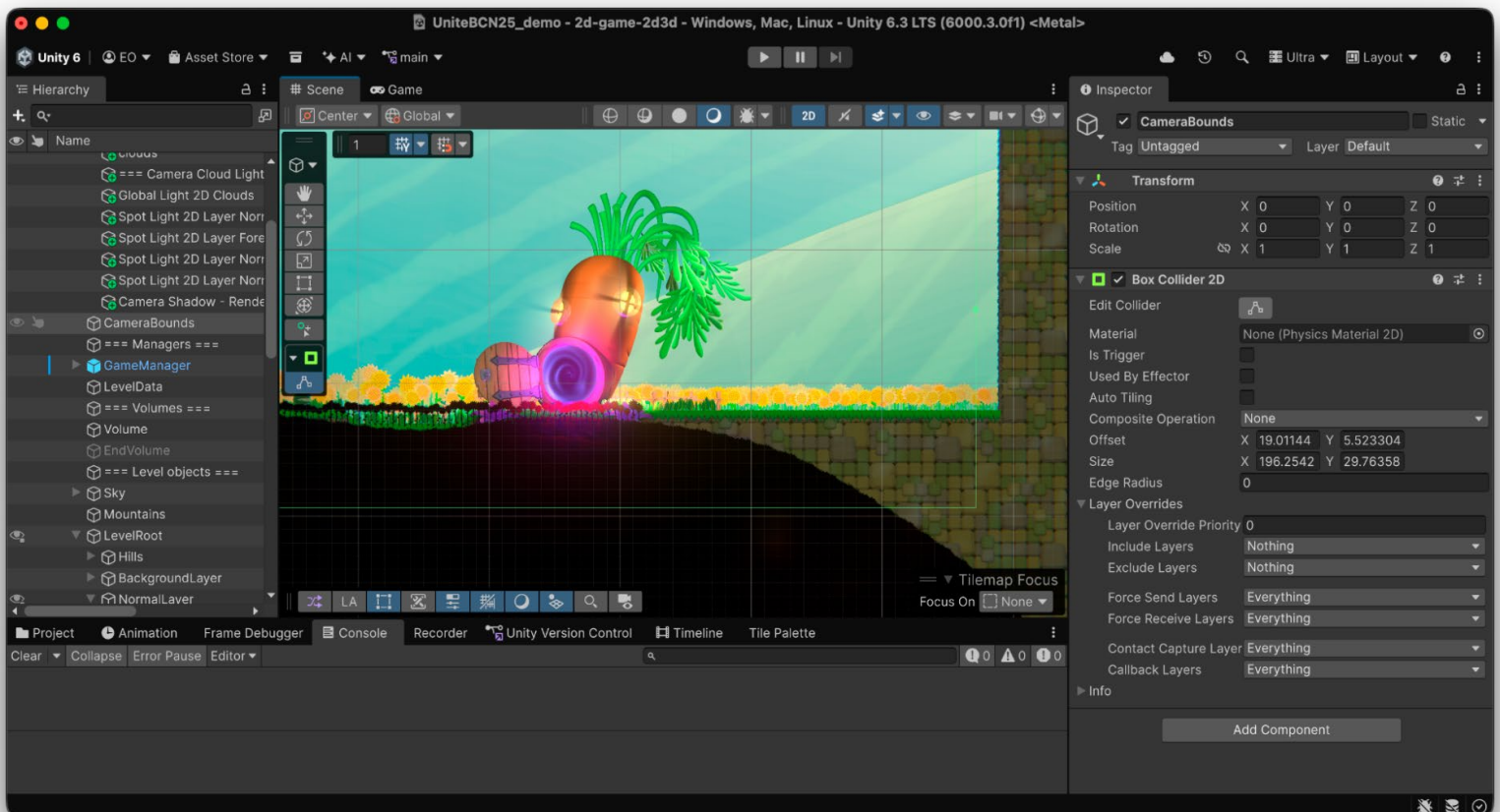
가상 카메라의 확장 기능인 Cinemachine Confiner 2D를 활성화하면 카메라가 레벨 경계 밖으로 나가지 못하도록 제한하여 플레이어에게 보여 주고 싶은 영역만 표시할 수도 있습니다. 이 기능을 활용하면 레벨에서 불필요한 부분은 디자인하지 않아도 됩니다.

Cinemachine Confiner 2D를 추가하려면 Add Extension 드롭다운 메뉴에서 선택하세요.



Cinemachine Confiner 2D 확장 기능 추가

Cinemachine Confiner 2D에는 Bounding Shape 2D로 Collider 2D(Composite 또는 Polygon)를 사용해야 합니다. 빈 게임 오브젝트를 생성한 후 Composite Collider 2D와 Box Collider 2D를 추가합니다. 자동으로 추가된 Rigidbody 2D의 Body Type을 Static으로 설정합니다. Box Collider 2D의 Used by Composite 옵션도 선택합니다. 그리고 이 게임 오브젝트를 Cinemachine Confiner 2D 스크립트의 Bounding Shape 2D 필드로 드래그합니다. Box Collider 2D가 카메라보다 커지도록 크기를 수정해야 합니다.



CameraBounds라는 Collider 2D는 카메라가 경계를 넘어 표시되지 않게 합니다.

이제 카메라의 절두체가 콜라이더의 바운딩 박스를 벗어나지 않습니다. 더 자세한 내용은 Cinemachine Confiner 2D [기술 자료](#)를 확인하세요.

스프라이트 작업

스프라이트는 모든 2D 게임의 핵심 요소입니다. 2D 템플릿을 사용하는 프로젝트에서는 시각적 에셋이 기본적으로 스프라이트로 임포트됩니다. 스프라이트는 아트를 포함하며, 씬에 추가하고 Unity의 전체 2D 툴셋과 함께 사용할 수 있습니다.

스프라이트 에셋

스프라이트는 평평한 3D 메시에 매핑된 2D 텍스처입니다. 스프라이트 에셋에서 사용할 수 있는 스프라이트 모드는 다음과 같습니다.

- **Single:** 기본 모드로, 이미지에 하나의 이미지 요소만 포함됩니다.
- **Multiple:** 텍스처 소스 파일에서 같은 이미지 안에 요소가 여러 개 있는 경우 이 값을 선택합니다. 그런 다음 이미지를 여러 개의 하위 에셋으로 분할할 방법을 Unity에서 인지할 수 있도록 스프라이트 에디터에서 요소의 위치를 정의합니다. 분할된 각 그래픽은 UI 툴킷에서 따로 사용할 수 있는 개별 스프라이트가 됩니다. PSD Importer로 가져온 PSD 에셋은 기본적으로 Photoshop 파일의 레이어마다 하나의 스프라이트를 표시합니다.
 - 타일러블 이미지나 9슬라이스 스프라이트와 같이 슬라이스가 필요한 이미지는 Multiple 유형이어야 합니다.
- **Polygon:** 원형 또는 정다각형 폴리곤 이미지에 가장 적합한 모드로, 이미지 셰이프와 가장 일치하는 외곽선을 설정하여 깔끔한 최적의 외곽선을 얻을 수 있습니다. 이는 파티클 효과 등에 유용하며 오버드로우를 줄여 줍니다.



다양한 스프라이트 모드를 사용하는 예셋의 예시

보조 텍스처(보조 텍스처 작업에 대한 자세한 내용은 조명 섹션을 참조)로 사용할 스프라이트는 **Normal map**(노멀 맵)과 **Default** 또는 **Single Channel**(마스크 맵)로 임포트해야 합니다.

추가 스프라이트 전용 설정은 인스펙터에서 찾을 수 있습니다.

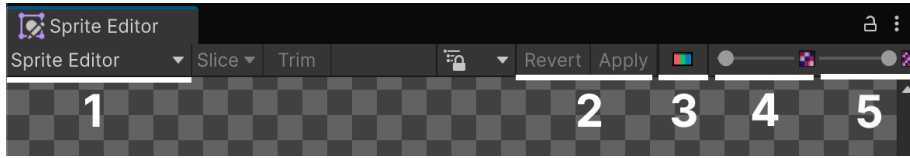
- **Pixels Per Unit:** 그리드 유닛당 예상 픽셀 밀도를 나타냅니다(에셋의 해상도 설정에 대한 이전 섹션에서 설명).
- **Mesh Type:** 스프라이트가 **Full Rect** 옵션으로 사각형 3D 메시가 될지, 아니면 **Tight** 옵션으로 텍스처의 불투명 픽셀을 래핑하여 렌더링을 최적화하는 정밀한 3D 메시가 될지를 나타냅니다. 메시는 스프라이트 에디터에서 외곽선으로 표시되며 조정이 가능합니다.
- **Extrude Edges:** 텍스처의 불투명 픽셀과 메시 사이에 여백을 추가하는 옵션입니다.
- **Pivot:** 스프라이트를 확대/축소, 이동 및 회전할 때 메시의 기준점을 나타냅니다. 값은 정규화되며 0은 하단 또는 왼쪽, 1은 상단 또는 오른쪽, 0.5는 중앙입니다. 이는 Sprite Editor 창에서 조정할 수도 있습니다.
- **Generate Physics Shape:** Sprite Editor에서 특별히 정의된 다른 셰이프가 없으면 물리에 스프라이트의 외곽선을 사용합니다.

Sprite Editor

스프라이트에는 Unity의 2D 툴셋 전반에 걸쳐 공유되는 많은 프로퍼티가 있습니다. 간단한 스프라이트는 Unity에 임포트하는 즉시 사용할 수 있습니다. 그러나 용도에 따라 추가 조정이 필요한 스프라이트도 있습니다.

[Sprite Editor](#)는 스프라이트 에셋을 선택할 때 인스펙터 창을 통해 액세스할 수 있으며, 상단 메뉴의 **Windows > 2D > Sprite Editor**를 통해서도 액세스할 수 있습니다.

Sprite Editor의 메뉴 옵션을 간단히 소개하면 다음과 같습니다.



1. Sprite Editor 메뉴를 클릭하여 스프라이트 편집을 위한 추가 옵션을 제공하는 드롭다운 목록을 사용합니다.
2. 변경 사항을 적용하거나 되돌려야 할 경우 이 옵션 중 하나를 선택합니다.
3. 컬러 및 알파 사전 시각화 모드를 전환합니다.
4. 확대 및 축소가 가능하며, 마우스 휠을 사용하여 확대/축소할 수도 있습니다.
5. 이 옵션을 활성화하면 스프라이트의 다양한 맵 레벨을 미리 볼 수 있습니다.

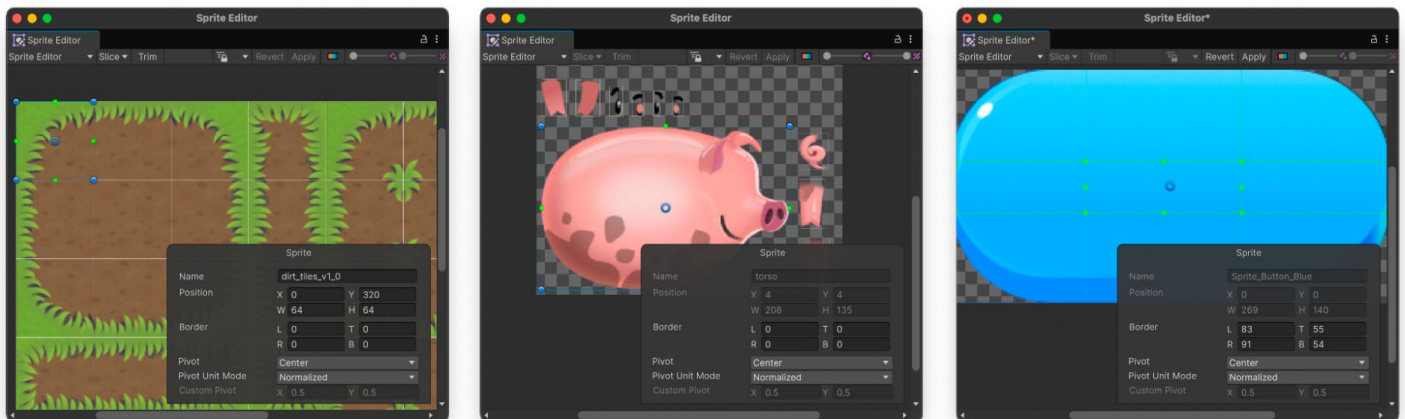
각 옵션을 좀더 자세히 살펴보겠습니다.

Sprite Editor

이 메뉴에서는 각 스프라이트를 슬라이스하고, 경계를 조정하며, 피벗을 설정하는 툴을 찾을 수 있습니다. 슬라이스 옵션은 스프라이트 에셋에서 **Multiple** 모드를 선택한 경우에 사용할 수 있습니다(예: 타일 시트를 임포트하는 경우).

멀티 파트 캐릭터의 여러 부분은 모두 같은 시트에 있을 경우 수동으로 슬라이스할 수 있지만, PSD Importer를 사용할 때 권장되는 워크플로는 Photoshop 파일의 레이어별로 스프라이트를 생성하는 것이므로 슬라이스가 필요하지 않습니다.

마지막으로 스프라이트의 크기를 조정하되 경계를 유지해야 하는 경우(예: 플랫폼 스프라이트나 UI 버튼), [9슬라이스](#) 기능을 사용하여 초록색 핸들로 경계와 반복 작업이 가능한 내부 영역을 정의할 수 있습니다. 그리고 피벗 핸들을 옮겨 각 스프라이트의 피벗을 조정할 수 있습니다.

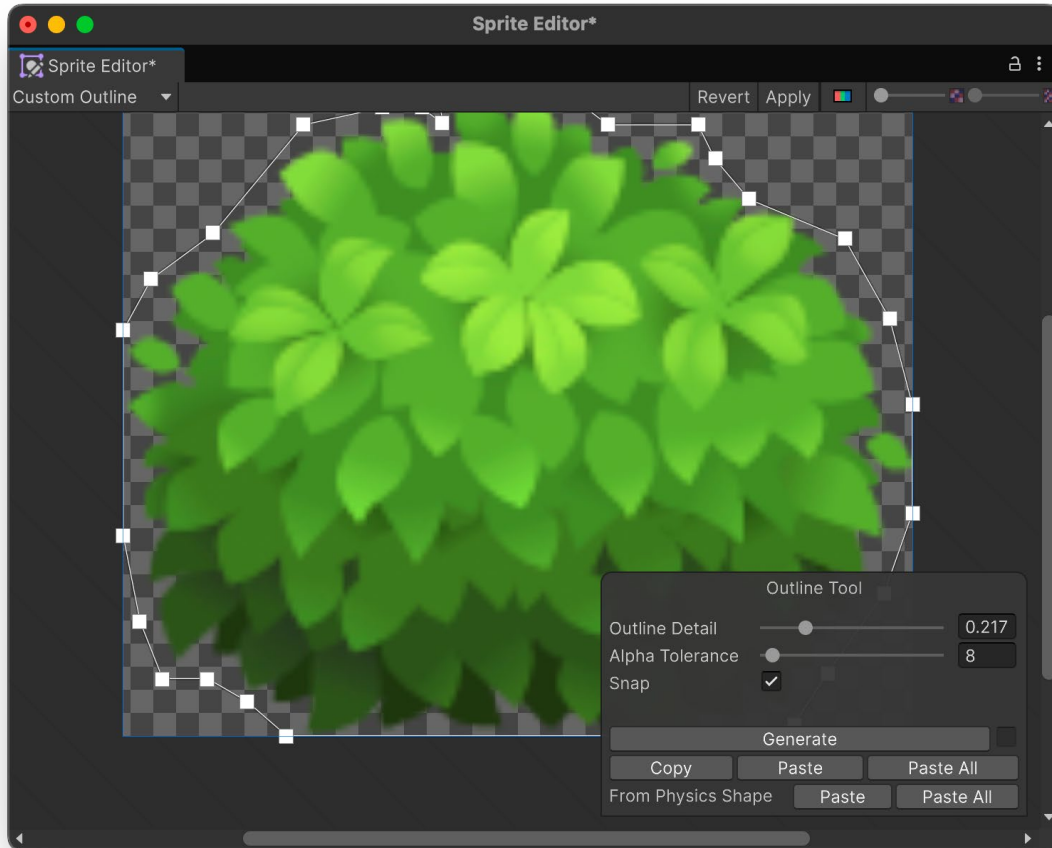


왼쪽부터: 슬라이스된 타일 시트, 애니메이션을 위한 멀티 파트 캐릭터, 9슬라이스 스프라이트



Custom Outline

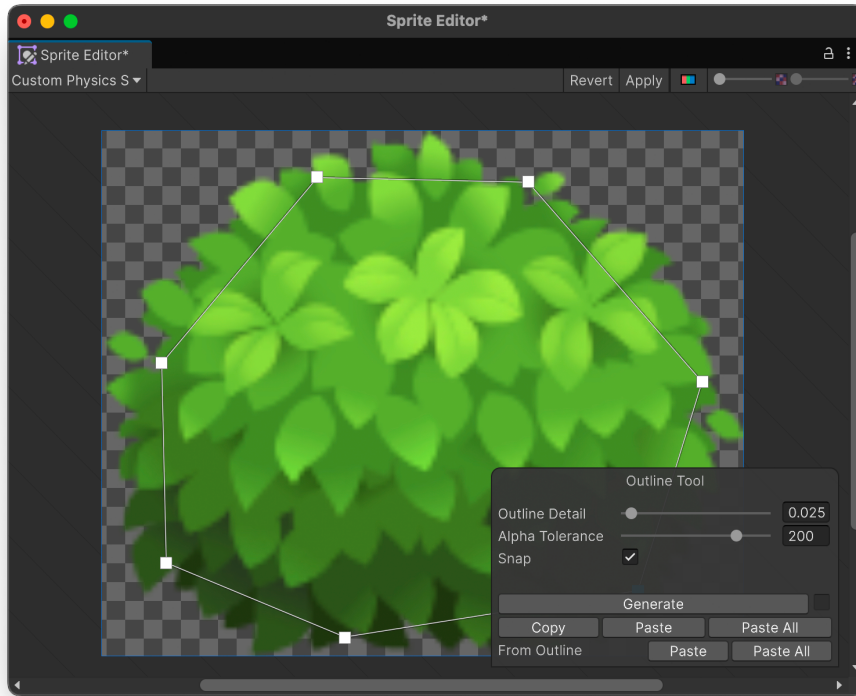
Custom Outline 옵션을 사용하여 스프라이트의 불투명 픽셀에 맞춰 조정된 정밀한 메시지를 생성함으로써 성능을 향상시킬 수 있습니다(외곽선 외부의 픽셀은 렌더링되지 않고 무시됨). 경로 포인트를 수동으로 옮겨 메시 또는 외곽선을 조정해서 만들면 외곽선 셰이프를 너무 복잡하게 만들지 않고 셰이프에 더 적합하게 맞출 수 있습니다.



Custom Outline 옵션으로 생성 및 조정된 외곽선

Custom Physics Shape

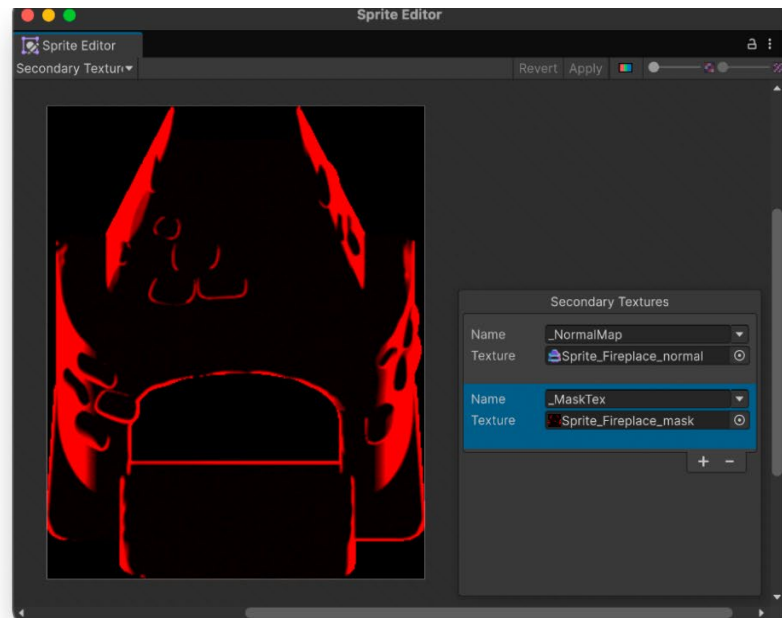
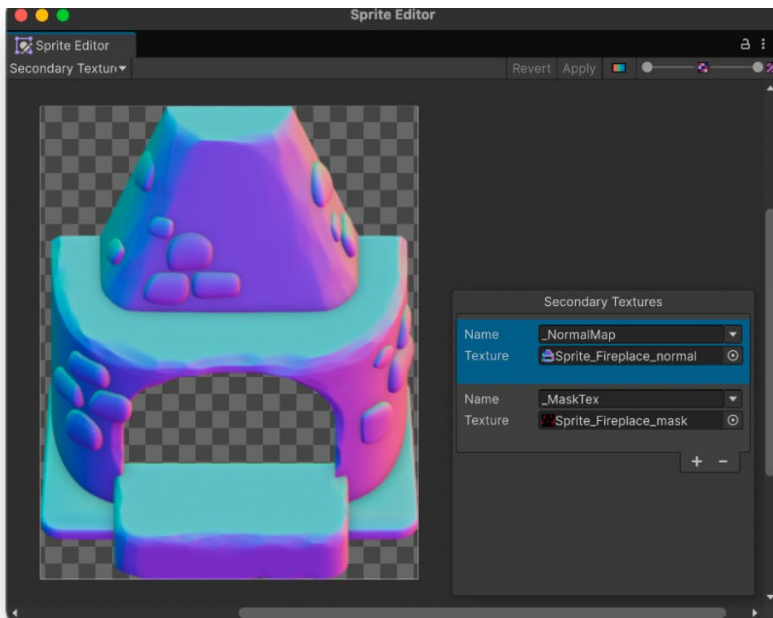
Custom Physics Shape 모듈은 Custom Outline 기능과 동일한 생성 옵션을 공유하지만, 그 용도는 다릅니다. 이 기능을 사용하면 스프라이트의 시각적 실루엣을 밀접하게 따르지 않는 다른 물리 오브젝트와 충돌할 스프라이트 영역을 정의할 수 있습니다. 복잡한 물리 셰이프는 성능 비용이 많이 들게 되므로 최대한 간단하게 유지하세요.



UI 빌더에서 생성된 커스텀 물리 셰이프

Secondary Textures

이 모듈을 사용하면 2D 광원 시스템이나 Shader Graph에서 사용할 노멀 맵 또는 마스크 맵 에셋을 선택할 수 있습니다. 보조 텍스처를 추가하는 것이 필수는 아니지만, 2D 광원과 함께 사용하면 게임의 시각적 품질을 높일 수 있습니다. 아래의 2D 광원 및 그림자 챕터에서 보조 텍스처에 대해 더 자세히 소개합니다.

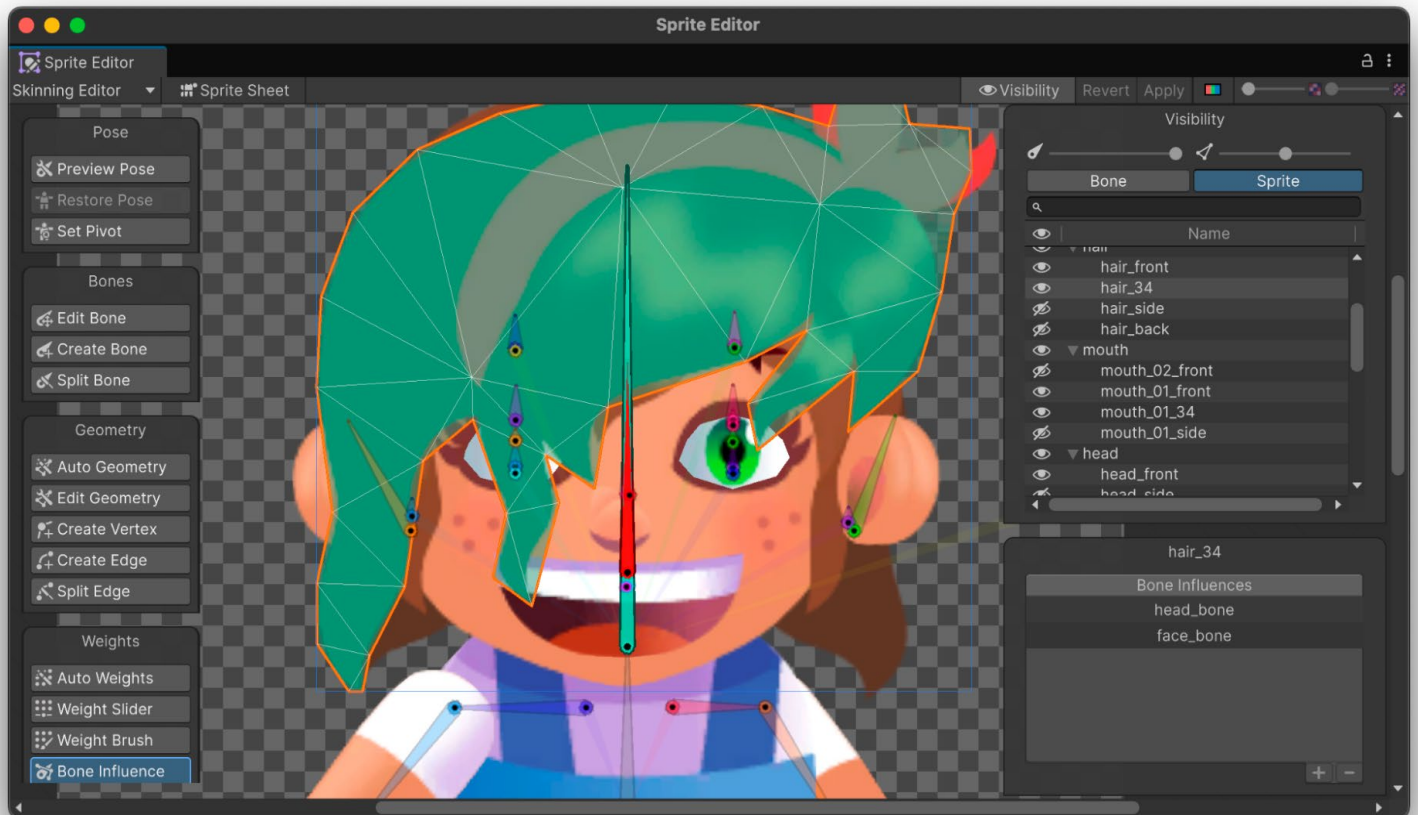


스프라이트에 추가된 노멀 맵과 마스크 맵으로 흥미로운 조명 효과를 연출할 수 있습니다.



Skinning Editor

Unity에는 골격 애니메이션을 위한 완벽한 솔루션이 있습니다. 바로 **2D Animation**입니다. 이 섹션에서는 각 스프라이트에 대한 지오메트리를 생성하고, 뼈대를 연결하며, 스프라이트 지오메트리의 각 버텍스에 대한 뼈대의 영향(가중치)을 조정할 수 있습니다. 설정이 완료되면 Unity의 애니메이션 시스템을 사용하여 스프라이트가 구부러지는 애니메이션을 만들 수 있으며, 이는 3D 캐릭터의 스킨드 메시에서처럼 작동합니다. 2D Animation 챕터에서 더 자세히 알아보세요.

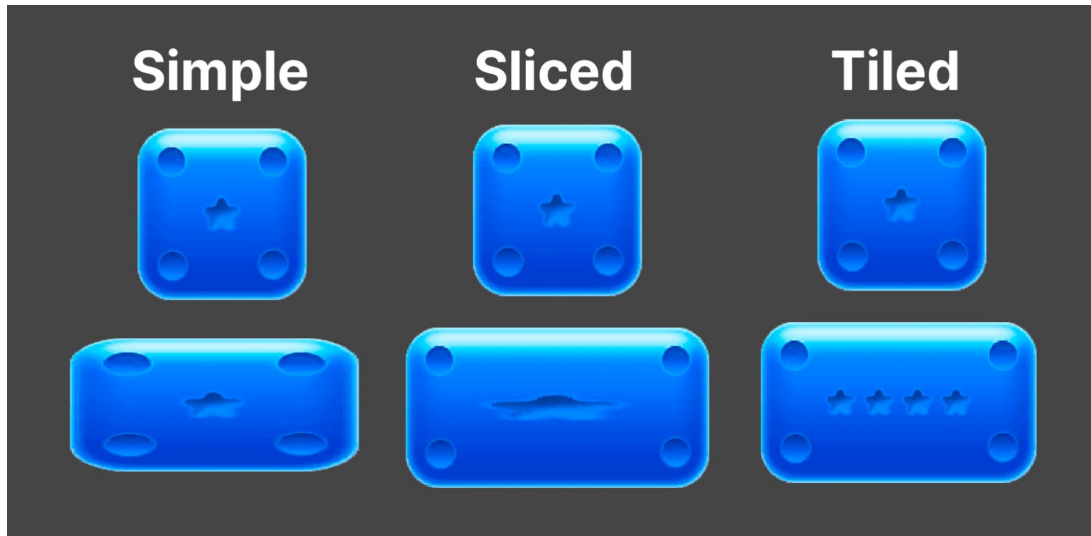


여러 부분으로 구성된 캐릭터. Sprite Editor로 리깅을 처리했으며, 애니메이션으로 만들 수 있습니다.

Sprite Renderer 컴포넌트

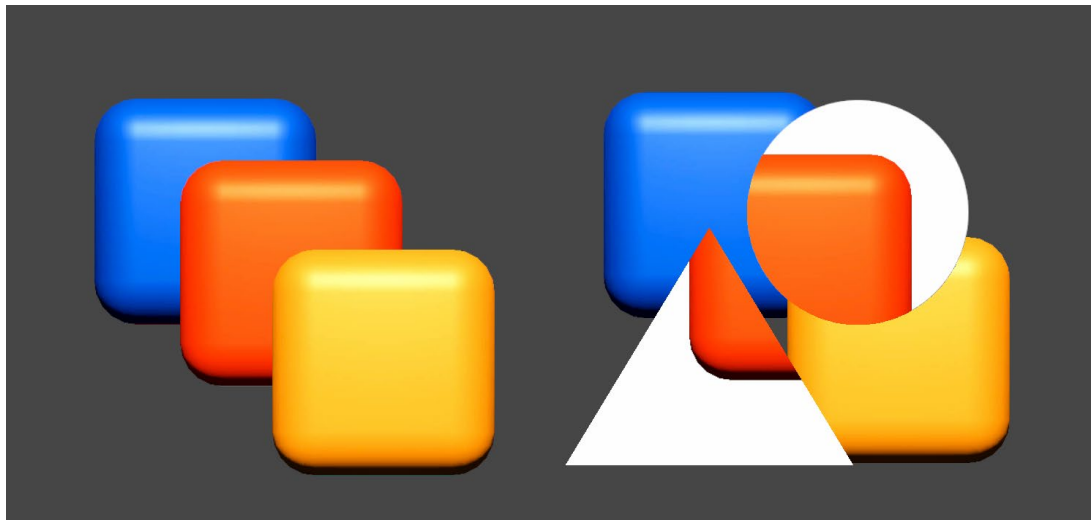
사용할 스프라이트 에셋이 준비되면 에셋을 간단하게 씬이나 계층 뷰로 드래그하여 사용할 수 있습니다. 일반적인 Transform 컴포넌트와 Sprite Renderer가 포함된 새 게임 오브젝트가 생성됩니다. **Sprite Renderer**에서는 다음과 같은 옵션을 사용할 수 있습니다.

- **Color:** 게임 오브젝트의 빠른 배리레이션을 만들기 위해 스프라이트에 컬러를 적용하거나 게임플레이 중에 발생할 효과를 추가할 수 있습니다. **API**를 사용하면 코드를 통한 조정이 가능합니다.
- **Flip:** 스프라이트의 방향을 반전시켜 스프라이트가 빠르게 다른 쪽을 향하게 만듭니다.
- **Draw Mode:** 스프라이트의 크기가 변경될 때 스케일이 어떻게 조정되는지를 정의합니다. Sliced, Tiled, Simple과 같은 옵션 중에서 선택할 수 있습니다.



다양한 Draw Mode로 스케일이 조정된 동일한 스프라이트

- **Mask Interaction:** 스프라이트는 가려질 수도 있고, 다른 것을 가릴 수도 있습니다. 스프라이트가 마스크 역할을 하는 다른 스프라이트 안팎에서만 보이도록 하려면 Mask Interaction 옵션을 사용하세요. 스프라이트가 [스프라이트 마스크](#)처럼 동작하도록 하려면 Sprite Mask 컴포넌트를 추가해야 합니다. 이 옵션 사용에 대한 자세한 내용은 다음 섹션을 참조하세요.



Mask Interaction을 사용하기 전후의 스프라이트 3개. 삼각형과 원 스프라이트는 스프라이트 마스크 역할을 하고, 주황색 큐브는 마스크 안에서만 보이며, 노란색 큐브는 마스크 밖에서만 보입니다.

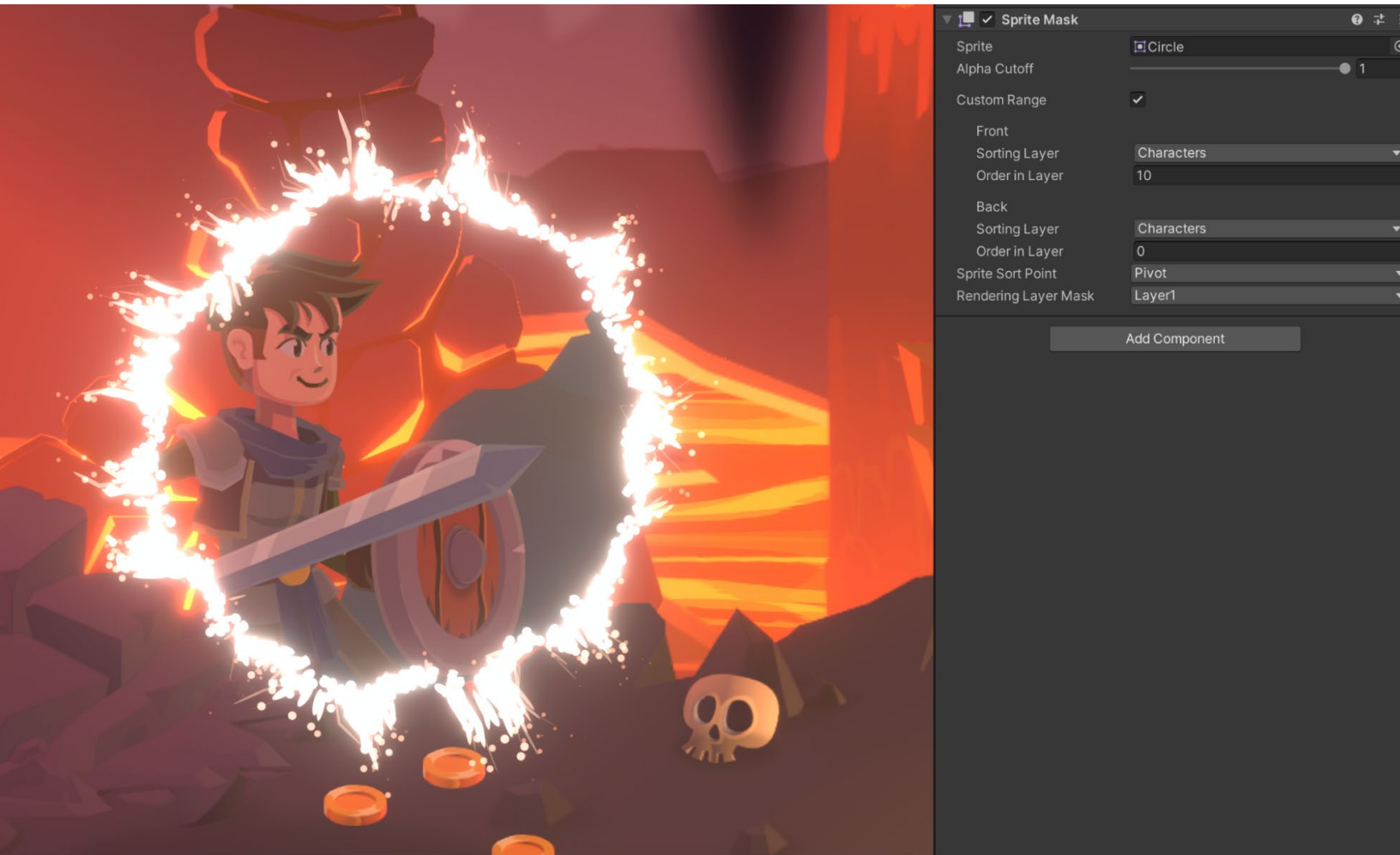
- **Sprite Sort Point:** 스프라이트와 카메라 간의 거리를 계산할 때 스프라이트의 중심 또는 피벗 포인트 중에서 선택합니다.
- **Material:** 새로 생성된 스프라이트의 머티리얼을 선택합니다. 기본 머티리얼은 **sprites - Default**로, 정렬 시스템 및 광원 시스템과 같은 2D 툴과 호환됩니다. 원형 아이콘을 클릭하여 오브젝트 피커 창을 열고, 2D 광원을 사용하지 않을 경우 **sprite - Unlit-Material**과 같은 다른 머티리얼을 선택할 수 있습니다. 스프라이트 렌더러와 호환되는 다른 셰이더는 Shader Graph로 생성할 수 있습니다.



스프라이트 마스크

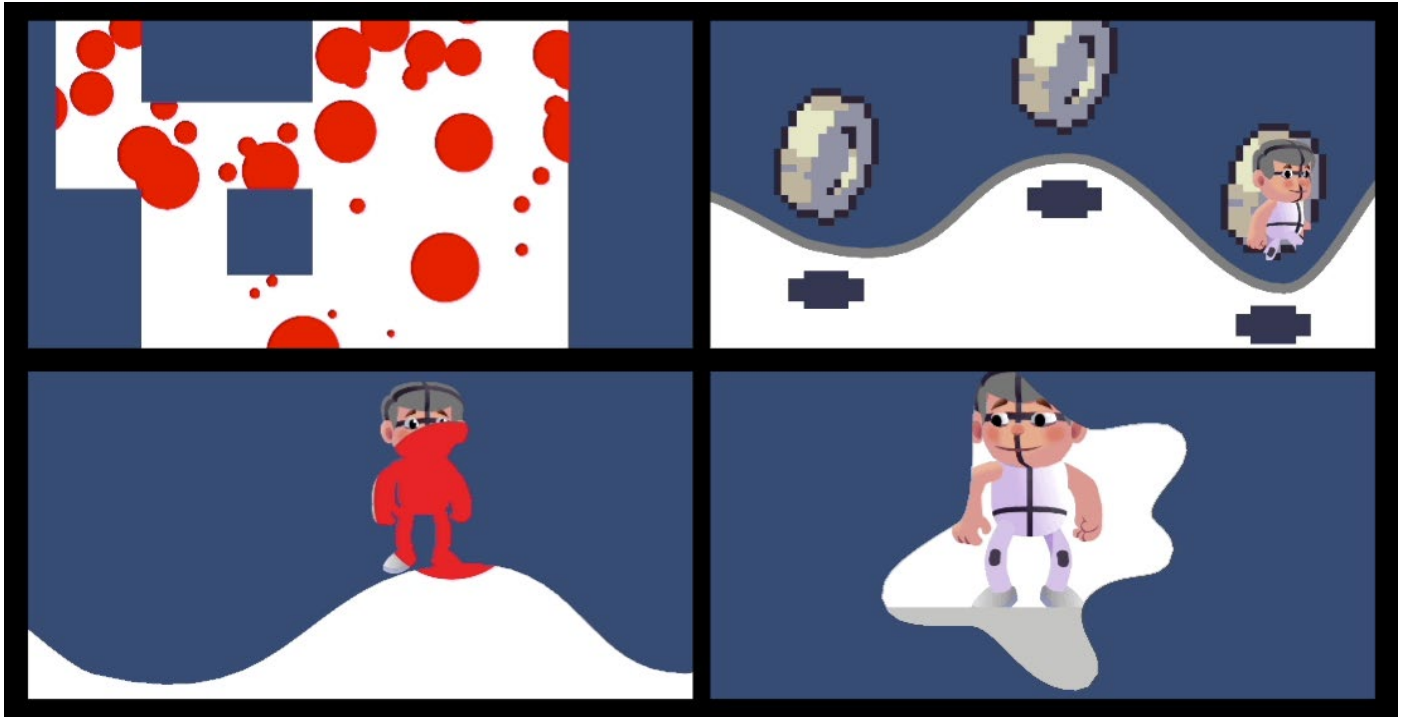
스프라이트 마스크를 사용하여 흥미로운 효과를 연출할 수 있습니다. 예를 들어 이미지의 일부분을 숨겨 포털 효과를 생성하거나 3D 효과가 있는 수집형 카드를 만들 수 있습니다.

마스크 이미지로 사용되는 스프라이트 자체는 애니메이션 톨로 애니메이션화할 수 있습니다. 이 방법으로 흥미로운 효과를 구현할 수 있습니다. 예를 들어 포털을 나타나게 한 다음 크기를 키우거나 셰이프를 변경하는 것이 가능합니다.



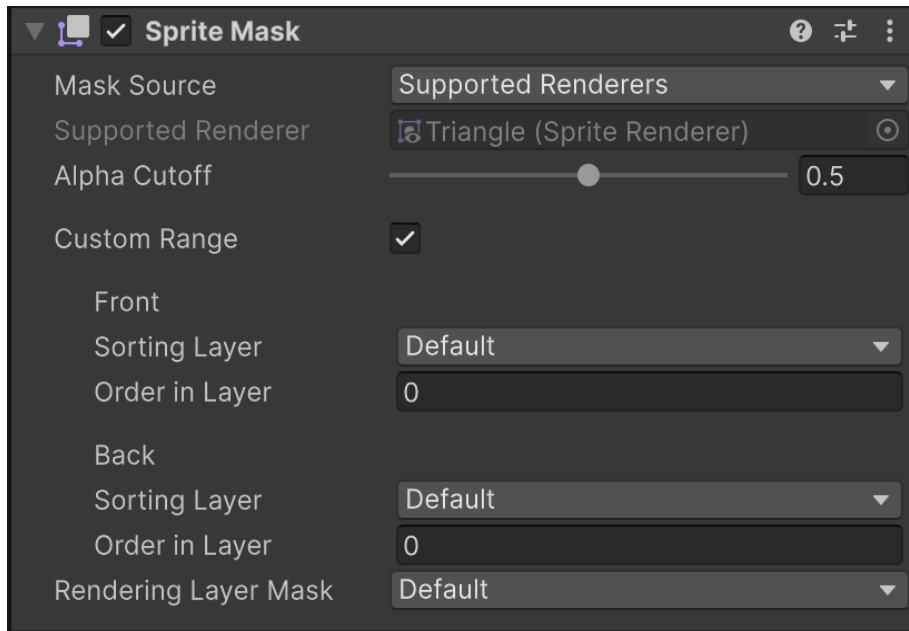
스프라이트 마스크와 빌트인 파티클 시스템으로 생성된 포털 효과입니다.

이전 버전의 Unity에서는 스프라이트 마스크가 마스크로 사용될 특정 스프라이트를 통해 정의되었습니다. 이를 통해서도 이미 많은 효과를 구현할 수 있지만, Unity 6에서는 마스크 시스템이 크게 개선되었습니다.



2024년 유나이티드 바르셀로나의 2D 워크숍 강연에서 선보인 몇 가지 예시입니다.

Sprite Mask 컴포넌트는 모든 2D 렌더러를 마스크 소스로 사용할 수 있습니다. 여기에는 스프라이트, 타일맵, 스프라이트 셰이프 및 뼈대에 리깅된 캐릭터도 포함됩니다. 모든 2D 오브젝트는 애니메이션화된 경우에도 마스크가 될 수 있으며, 이를 통해 새로운 창의적 연출을 펼칠 수 있습니다.



더욱 세밀한 제어가 필요한 경우 마스크의 영향을 받을 스프라이트를 정렬 레이어 또는 레이어 내 순서로 선택할 수 있습니다.

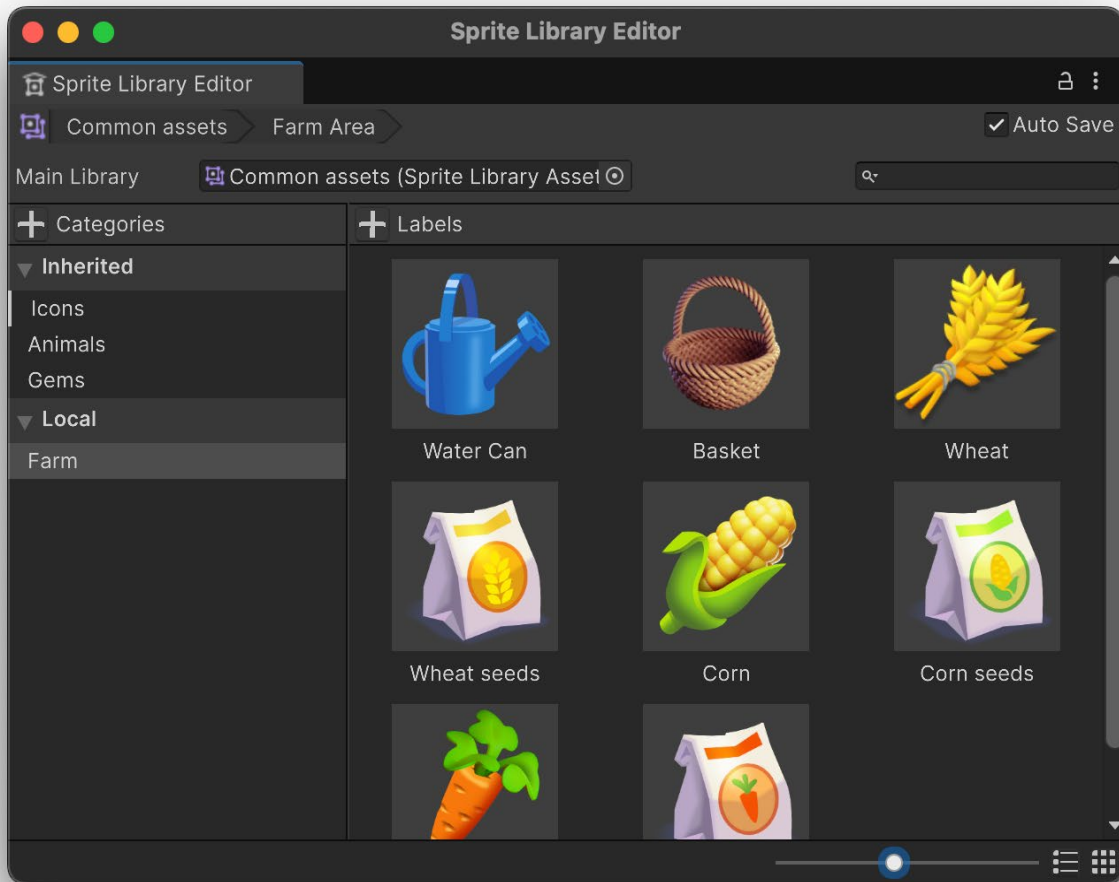


Sprite Library Asset

Sprite Library Asset을 통해 카테고리별로 그룹화된 스프라이트 컬렉션을 만들 수 있습니다. 스프라이트 라이브러리를 활용해 카테고리에 따라 에디터에서 또는 런타임 시 스프라이트를 간편하고 직관적으로 교체할 수 있습니다. 이는 프로젝트의 스프라이트 수가 너무 많아질 경우 매우 유용할 수 있는데, 예를 들어 직접 그린 애니메이션이 많은 2D 인디 게임의 경우 스프라이트 수가 수만 개에 이를 수 있습니다.

스프라이트의 유형을 다음과 같은 카테고리로 그룹화할 수 있습니다.

- **게임플레이 목적:** 예를 들어 무기 카테고리는 검, 도끼 또는 철퇴로 레이블이 지정된 스프라이트를 포함할 수 있습니다.
- **애니메이션:** 캐릭터의 왼손 카테고리에는 다양한 위치에서 그려진 손이나, 모자 카테고리나 같은 다양한 시각적 액세서리가 포함될 수 있습니다.
- **레벨 디자인:** 이 카테고리에는 아파트, 마천루, 사무실 등 배경 건물과 같은 요소가 포함될 수 있습니다.



Sprite Library Editor 창



Sprite Resolver 컴포넌트

스프라이트 라이브러리와 그 컴패니언 컴포넌트인 Sprite Resolver의 핵심적인 용도로는 멀티 파트 캐릭터 애니메이션의 설정 및 스프라이트 스왑이 있습니다.

Unity 6에서는 씬 뷰에 새로운 **Sprite Swap 오버레이**가 추가되었습니다. 이 오버레이는 특정 카테고리에서 선택한 스프라이트가 교체될 수 있는 모든 관련 레이블을 보여 줍니다.

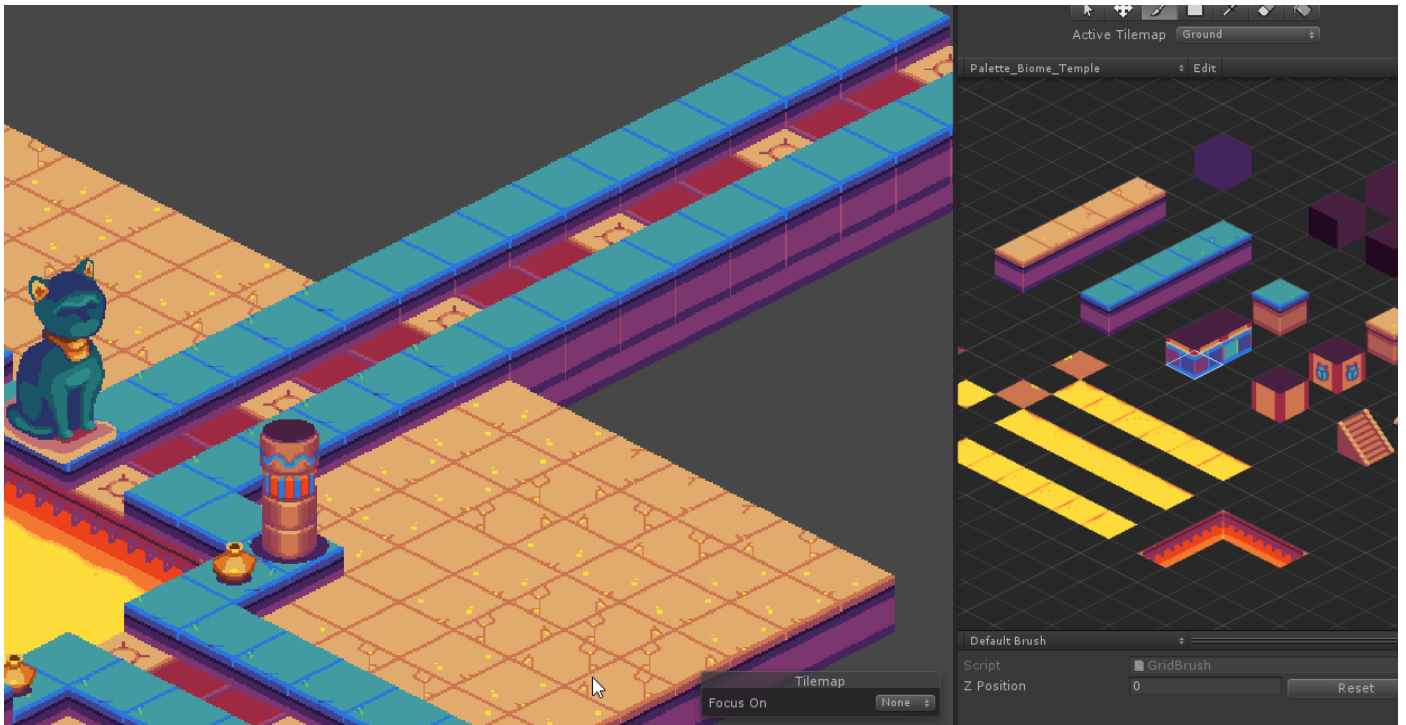


애니메이션 또는 레벨 디자인 단계에서 유용하며, 프로젝트(Project) 창에서 검색하지 않고도 동일한 카테고리의 스프라이트를 선택할 수 있습니다.

[기술 자료](#)에서 Sprite Swap 및 Sprite Library Asset에 대한 내용을 자세히 알아보고, 개발 사이클 초기부터 사용해 프로젝트 확장 규모에 따라 라이브러리를 적절하게 정리하세요.

2D 타일맵

타일맵을 사용하면 그리드에 배치된 작은 스프라이트, 즉 타일로 게임 월드를 제작할 수 있습니다. 하나의 큰 이미지로 게임 월드를 만드는 대신, 마치 벽돌처럼 전체 레벨에 걸쳐 반복되는 일정한 단위로 게임 월드를 분할할 수 있습니다.



Unity는 세 가지 유형의 타일맵, 즉 아이소메트릭, 육각형 또는 직사각형을 지원합니다.



타일맵 생성

타일맵을 사용하면 화면에 표시되지 않는 타일을 비활성화할 수 있으므로 메모리 사용량과 CPU 리소스를 절약할 수 있습니다. Unity에서 브러시 툴을 사용하면 그리드에 효율적으로 타일을 페인팅할 수 있으며, 몇 가지 페인팅 규칙을 따르도록 스크립팅할 수도 있습니다. 효율적인 테스트와 편집을 위한 자동 충돌 생성 기능도 제공됩니다.



버니 블리츠의 타일맵

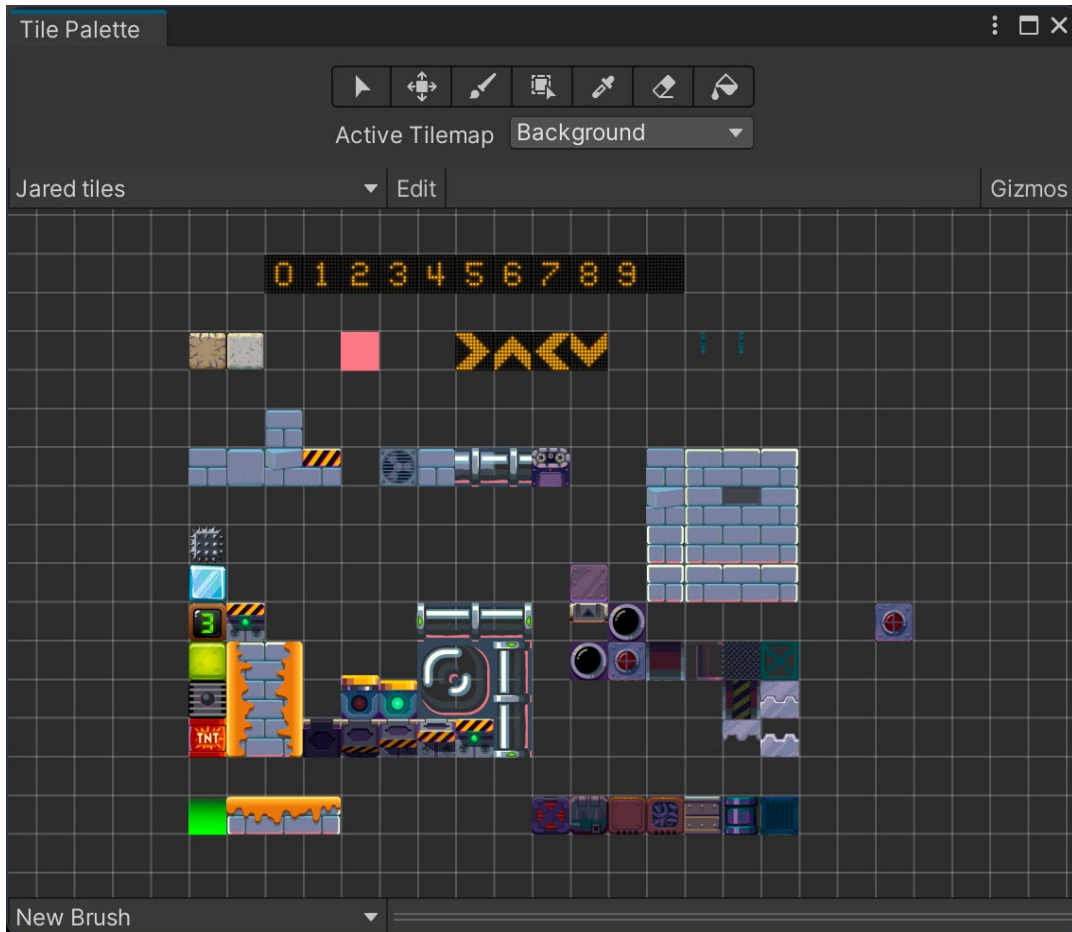
기본적으로 Tilemap 패키지는 에디터에 포함되어 있지 않으므로, [2D Tilemap](#) 패키지를 [패키지 관리자](#)에서 다운로드하거나 2D 템플릿으로 프로젝트를 시작해야 합니다.

Tilemap 패키지와 함께 [2D Tilemap Extras](#)도 설치해야 합니다. 이렇게 하면 재사용 가능한 2D 및 타일맵 에디터 스크립트를 프로젝트에 사용할 수 있으며, 제공되는 기능으로 커스텀 브러시와 타일을 만들 수도 있습니다.

패키지에 포함된 [스크립터블 브러시](#)와 [스크립터블 타일](#)에 관해 자세히 알아보세요.

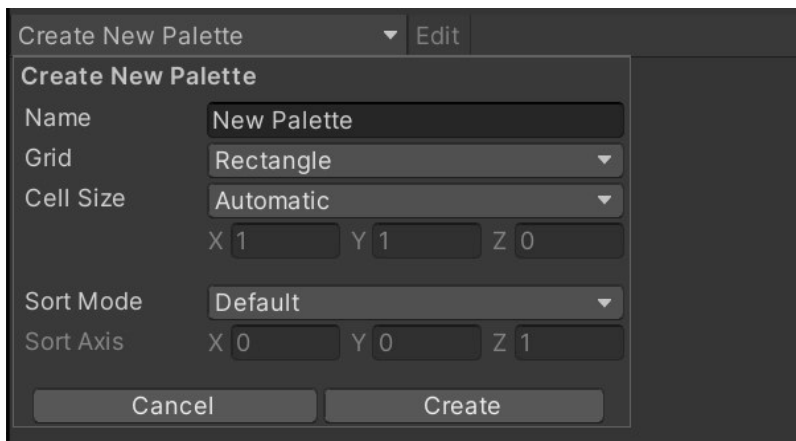
두 패키지를 모두 설치했다면 **Window > 2D > Tile Palette**에서 [Tile Palette](#) 창을 엽니다.

이 창에는 타일맵을 페인팅하거나 편집할 수 있는 모든 타일과 툴이 있습니다.



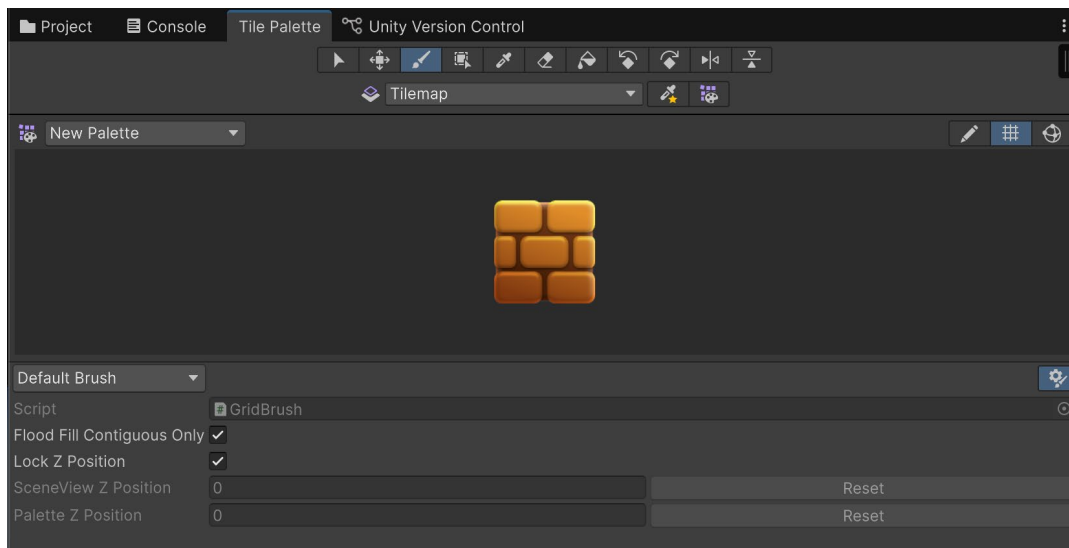
팔레트가 로드된 Tile Palette 창

Create New Palette 버튼을 클릭하여 새로운 팔레트를 생성합니다. 그러면 여러 옵션이 있는 드롭다운 창이 표시됩니다. 팔레트 이름을 정하고 옵션을 설정한 다음 **Create**를 클릭하고 선택한 폴더에 저장합니다.



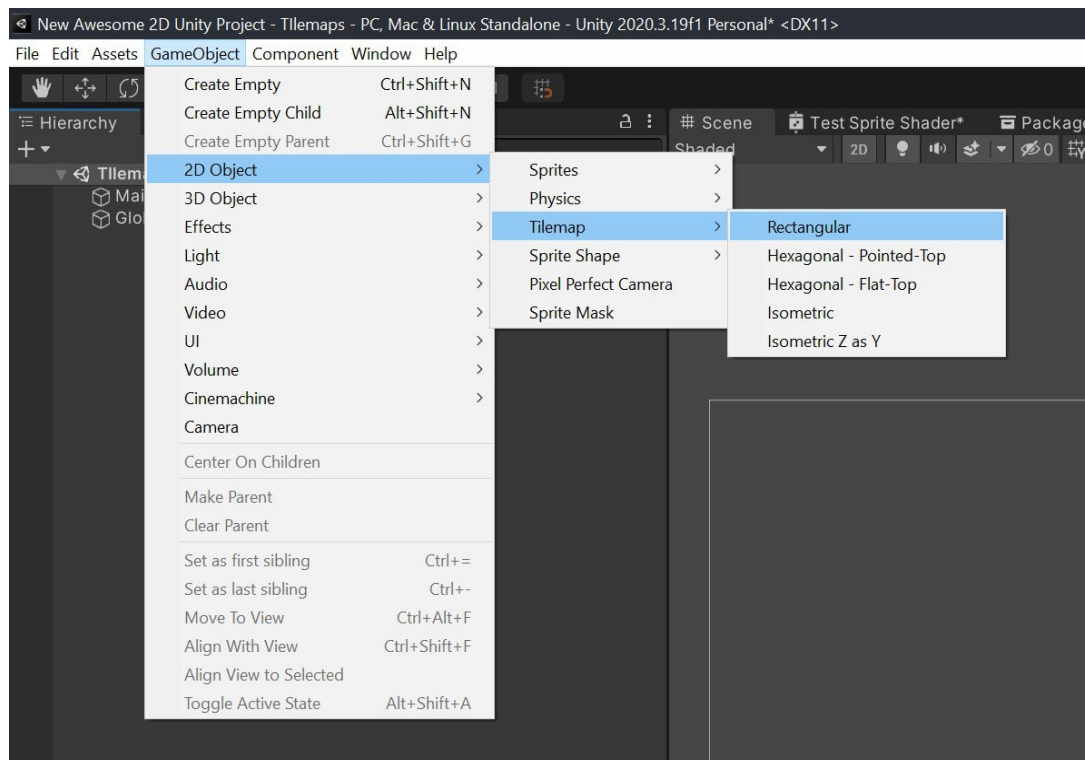
새로운 팔레트 생성

이제 팔레트에 타일을 추가하면 됩니다. **Palette** 창으로 스프라이트, 텍스처 또는 타일 에셋을 드래그합니다. 스프라이트를 드래그한 경우 먼저 타일 에셋을 저장해야 할 수 있습니다. 이제 팔레트의 그리드에서 스프라이트를 확인할 수 있습니다.



스프라이트를 팔레트로 드래그하면 새 타일이 생성됩니다.

이제 타일을 페인팅할 타일맵을 생성해야 합니다. **GameObject > 2D Object > Tilemap > Rectangular** 메뉴 항목을 클릭합니다.

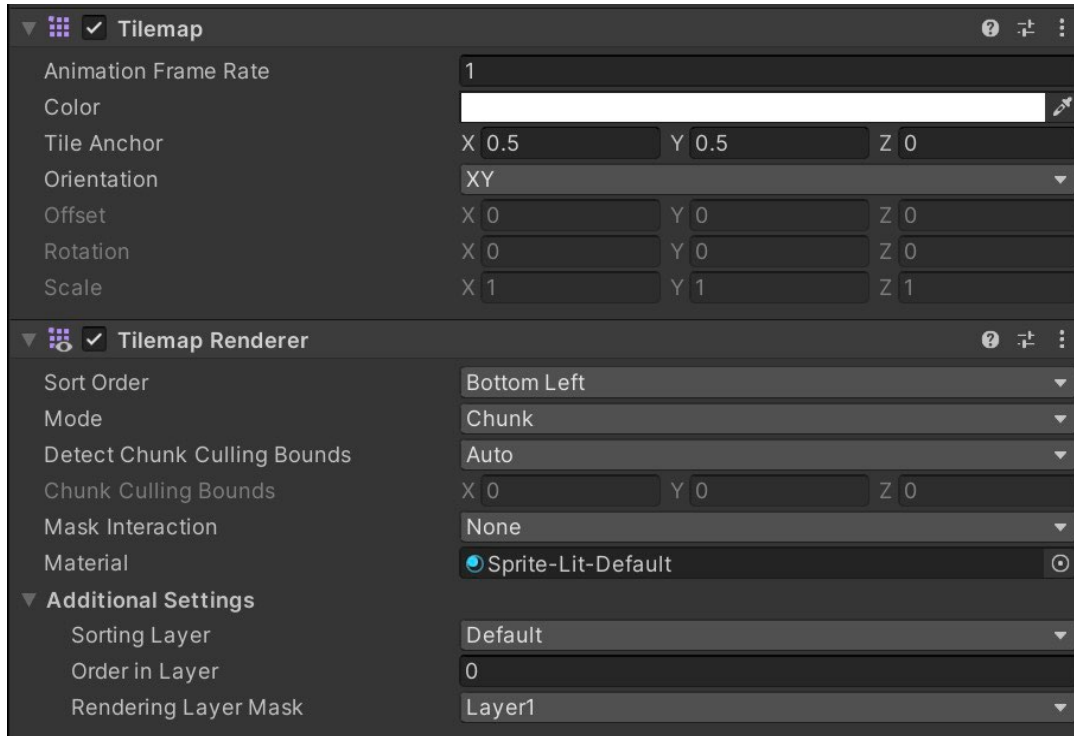


타일맵 생성



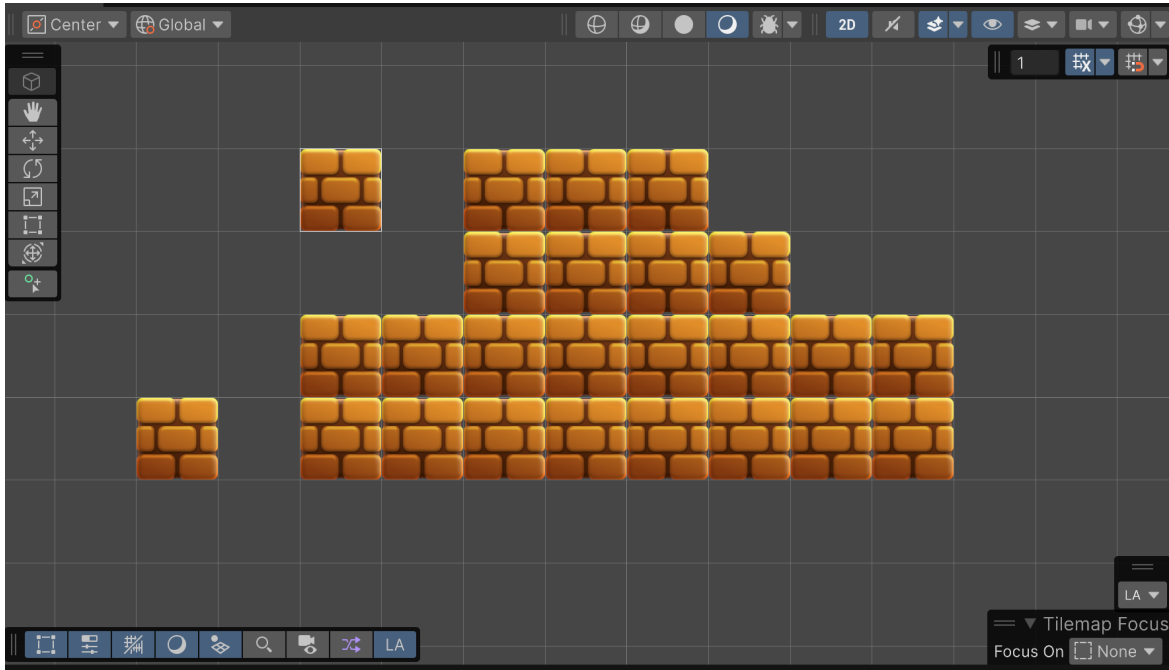
이렇게 하면 **Grid** 게임 오브젝트와 그 안의 **Tilemap** 게임 오브젝트가 생성됩니다. **Tilemap** 게임 오브젝트의 이름을 좀더 알아보기 쉽게 변경합니다. **Grid** 게임 오브젝트에는 여러 개의 타일맵이 있을 수 있으며, 씬마다 셀 크기가 다른 여러 그리드가 있을 수도 있습니다. 지금은 **Grid** 컴포넌트의 모든 설정을 기본값 그대로 사용합니다.

Tilemap 게임 오브젝트에는 **Tilemap** 및 **Tilemap Renderer**라는 두 개의 컴포넌트가 있습니다. 두 컴포넌트의 설정은 그대로 사용합니다. 필요한 경우 레이어 구조에 맞게 **Sorting Layer** 설정을 변경합니다.

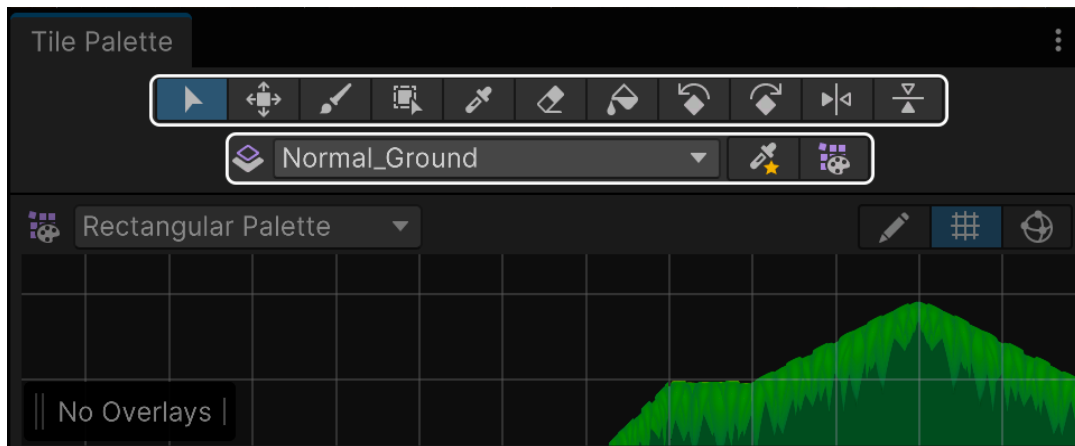


Tilemap 및 Tilemap Renderer 컴포넌트

타일맵과 팔레트의 타일이 있으면 이제 페인팅 작업을 할 수 있습니다. **Palette** 창 툴바에서 브러시 툴을 클릭하고 페인팅할 타일을 선택합니다.



타일맵에 페인팅된 간단한 벽돌 타일



Tile Palette 툴

위의 이미지에서 상단 바 메뉴를 시작으로 다양한 팔레트 툴을 살펴보겠습니다(키보드 단축키는 괄호 안에 나와 있음).

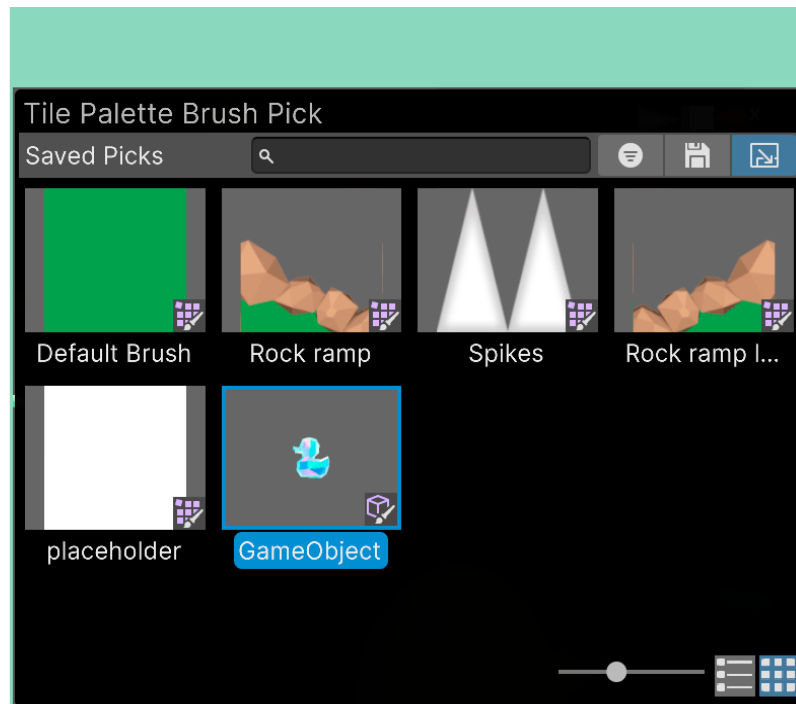
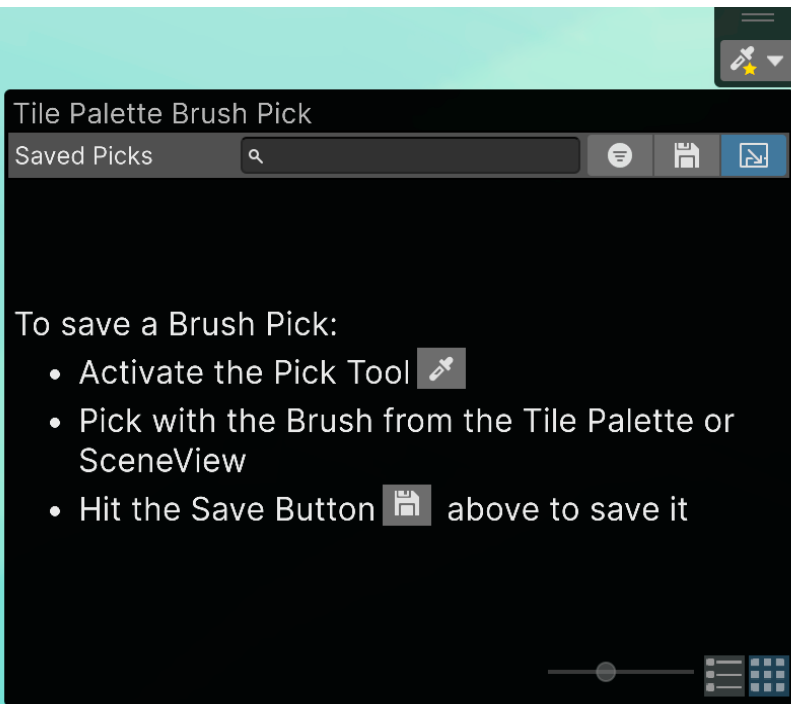
1. **선택(S)**: 타일 하나를 클릭하여 선택하거나, 직사각형 영역 내에 있는 여러 타일을 드래그해서 선택합니다.
2. **이동(M)**: 선택한 타일을 옮깁니다.
3. **브러시(B)**: Active Tilemap 드롭다운에서 선택한 액티브 타일맵을 원하는 타일 및 브러시로 페인팅합니다.
4. **선택 영역 채우기(U)**: 드래그하여 직사각형 영역을 선택한 타일로 채웁니다.
5. **타일 샘플러(I)**: 타일맵에서 타일을 선택하여 페인팅에 사용할 타일로 설정합니다.



6. **지우개(D)**: 타일맵에서 타일을 삭제합니다.
7. **채우기(G)**: 영역을 타일로 채웁니다. 해당 영역에는 다른 타일과 구분되는 경계가 있어야 하며, 그렇지 않으면 큰 직사각형이 그려집니다.
8. **왼쪽으로 회전(I)** 및 **오른쪽으로 회전(J)**: 브러시에서 활성 타일을 시계 방향 또는 반시계 방향으로 90도 회전합니다. 이는 방향별 배리어이션이 없으며 가끔 사용하는 타일에 유용합니다.
9. **수평 반전(shift + [)** 및 **수직 반전(shift +)]**: 회전 툴과 유사한 사용 사례로, 반전 툴을 사용하면 타일을 뒤집어 브러시에서 활성 타일의 빠른 배리어이션을 만들 수 있습니다.

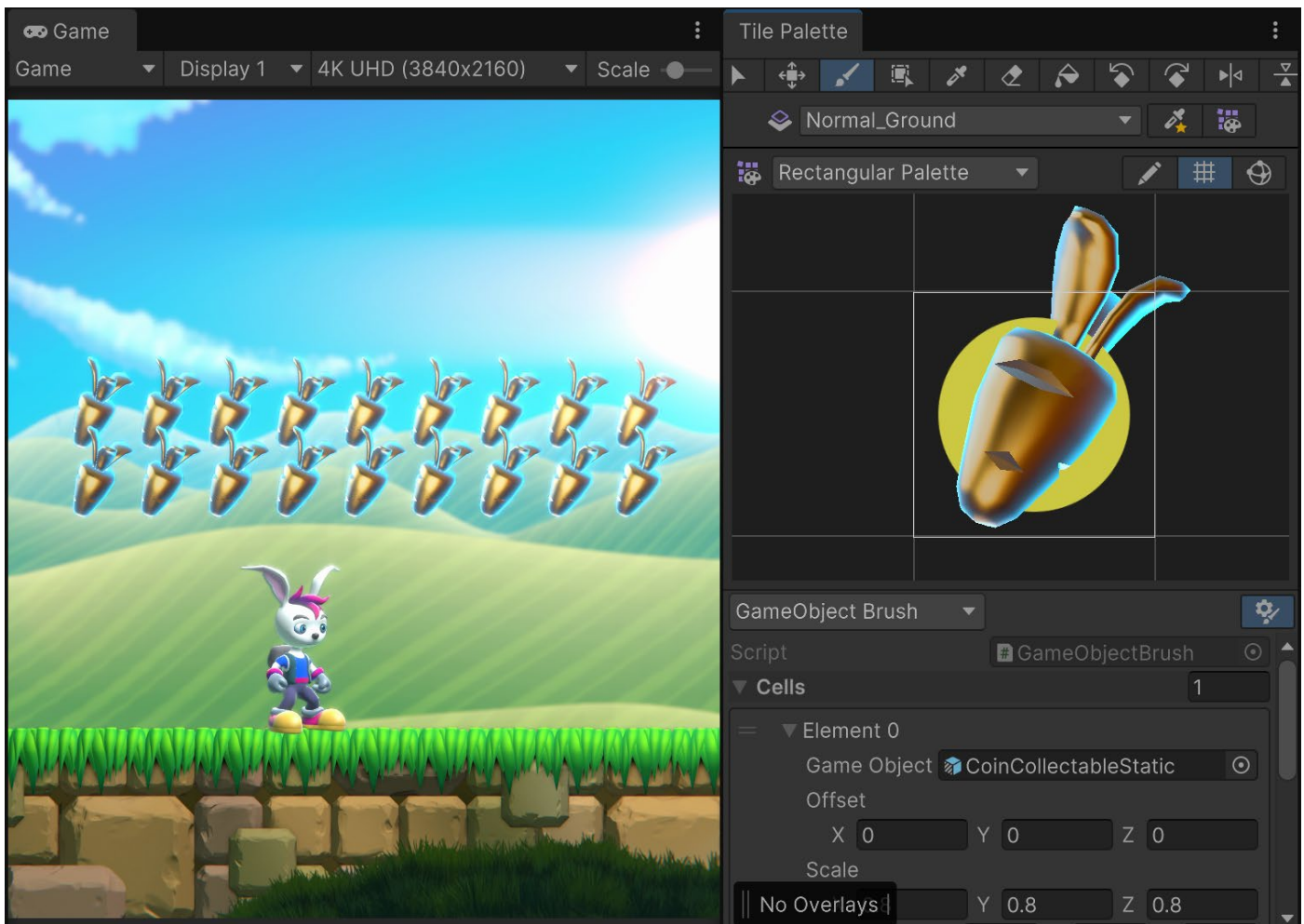
하단 바 메뉴에 포함된 툴은 다음과 같습니다(왼쪽에서 오른쪽으로).

1. **Active target**: 페인팅할 타일맵을 선택합니다.
2. **Tile Palette Brush Pick(')**: 씬 뷰에서 툴 오버레이를 활성화하거나 비활성화합니다. 이를 통해 회전 및 반전된 타일과 게임 오브젝트를 포함하여 타일을 저장하고 빠르게 선택할 수 있습니다.
3. **Tile Palette Overlay(;**: 씬 뷰에서 타일 팔레트 오버레이를 활성화하거나 비활성화합니다.



Tile Palette Brush Pick 오버레이는 가장 많이 사용하는 타일을 빠르게 사용할 수 있도록 북마크하는 유용한 방법입니다.

위의 툴만 있으면 타일을 효율적으로 페인팅하고 편집할 수 있습니다. 또한 Tilemap Extras Asset은 유용한 스크립트를 제공합니다. 예를 들어 **Rule Tile**(다음 섹션에서 Rule Tile의 실제 작동 예시 참고)이나 **GameObject Brush**가 있으며, 이를 통해 타일 팔레트에 게임 오브젝트를 배치하고 씬에서 페인팅할 수 있습니다.



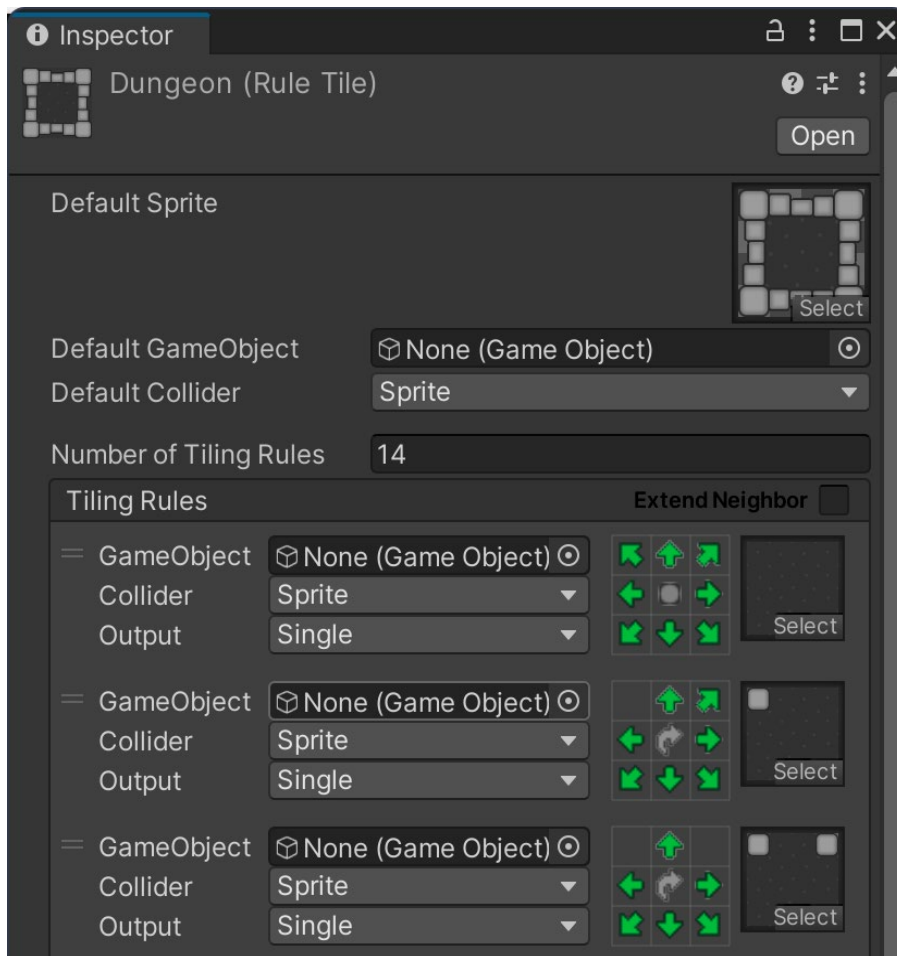
GameObject Brush는 타일맵 그리드를 사용하여 레벨에 수집 가능한 오브젝트를 배치하는 데 사용됩니다.

Rule Tile

Tile Palette 툴은 각 타일맵의 그리드를 타일로 효율적으로 채우는 데 사용됩니다. 하지만 일부 경우, 경로를 통해 형성된 세이프의 가장자리나 코너를 수동으로 배치하면 시간이 많이 걸리고 오류가 발생하기 쉽습니다. 경로 또는 기타 타일 기반 세이프에 변경 사항이 있을 때마다 수동 작업을 반복해야 합니다. 이를 피하기 위해 [Rule Tile](#) 기능을 사용할 수 있습니다. 이 기능은 세이프가 인접한 타일에 따라 올바른 경계 타일을 페인팅하는 작업을 수행합니다.

2D Tilemap Extras 패키지에 포함된 Rule Tile은 주변 환경을 인식하여 적절한 이미지(예: 위에 잔디가 자라고 아래에 그림자가 드리워진 지면 타일)를 선택하도록 스크립팅된 타일입니다.

Rule Tile은 인접한 타일의 위치와 경계의 적절한 위치를 자동으로 인식합니다. Rule Tile이 적절한 스프라이트를 선택해 주기 때문에 경계를 페인팅하기 위해 다른 타일을 선택할 필요가 없습니다.



인스펙터의 Rule Tile 에셋

Rule Tile 인스펙터에는 인접한 타일을 기반으로 어떤 스프라이트를 선택할지에 대한 규칙 목록이 있습니다. 각 규칙의 오른쪽에는 매트릭스와 스프라이트가 있습니다. 스프라이트는 초록색 화살표가 가리키는 모든 방향에 인접 타일이 있을 때 사용됩니다. Rule Tile 기능에 관한 자세한 내용은 [여기](#)에서 확인할 수 있습니다.

Unity 6.3의 AutoTile

Unity 6.1에서 도입된 [AutoTile](#)은 AutoTile 에셋 설정에서 정의한 마스크에 따라 올바른 인접 타일을 표시하는 타일 에셋입니다.

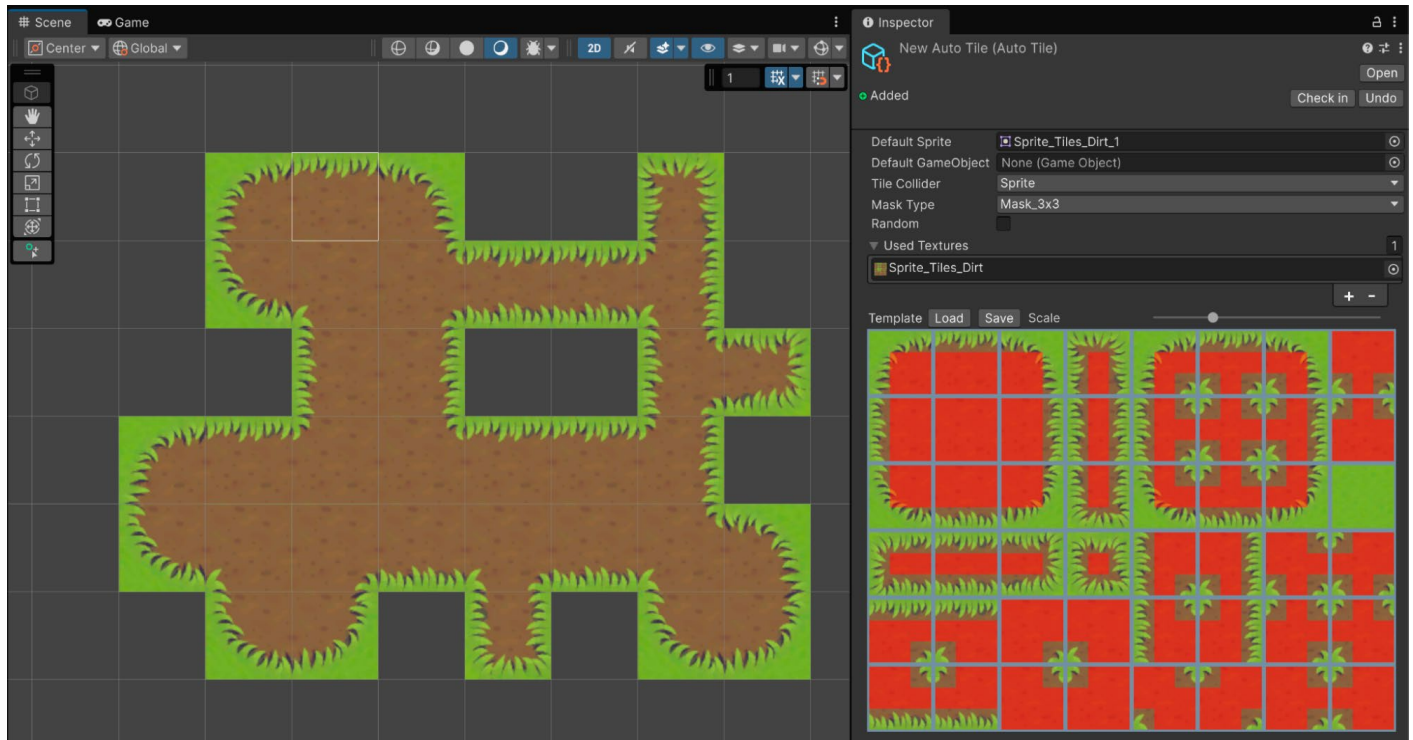
Rule Tile과 비교할 때 AutoTile은 설정이 보다 직관적이며, 마스크를 다시 설정하지 않고도 타일 시트의 템플릿 배리어이션을 생성할 수 있게 합니다.

AutoTile을 생성하는 단계는 다음과 같습니다.

1. 프로젝트에 타일 시트를 임포트합니다.
2. 스프라이트 에디터에서 타일 시트를 슬라이스합니다. 아래에 표시된 예시에서 타일 시트는 8x6, 즉 8열과 6행으로 구성되어 있습니다. 변경 사항을 적용합니다.



3. 프로젝트 창에서 새 AutoTile을 만듭니다.
4. 인스펙터에서 **Used Textures** 필드에 슬라이스된 타일 시트를 추가합니다.
5. 타일의 빈 부분을 채우며 페인트 브러시로 마스킹을 시작합니다. 타일의 마스킹이 다른 타일과 동일한 경우 강조 표시되므로 마스킹이 올바른지 다시 확인할 수 있습니다.
6. 타일맵에 페인팅하여 AutoTile을 테스트합니다.



해피 하비스트의 AutoTile 예시

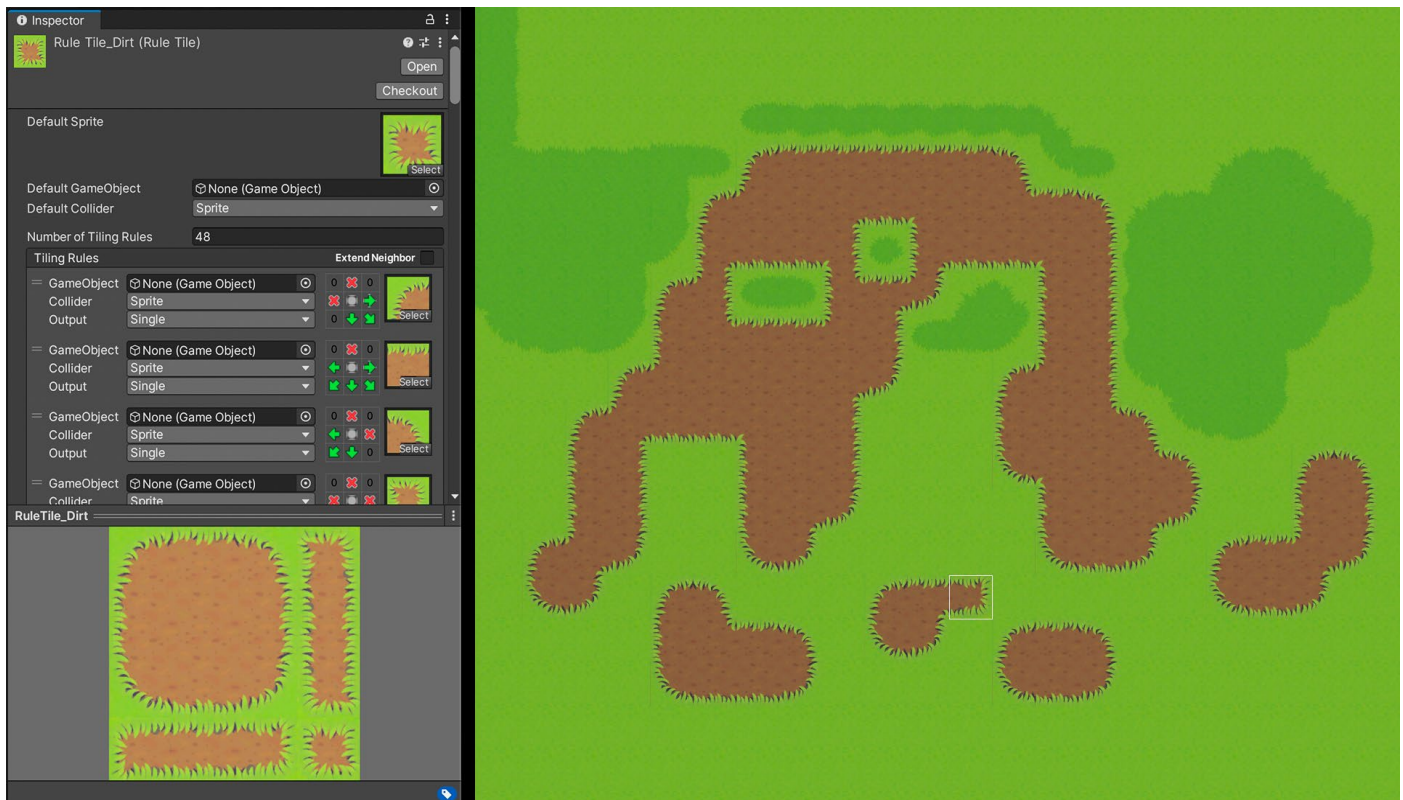
Used Textures 필드에 추가 텍스처를 로드할 수 있습니다. 예를 들어, 동일한 타일 시트의 배리언트를 로드한 다음 **Random** 옵션을 선택하여 동일한 인접 기준에 맞는 타일 중에서 무작위로 선택할 수 있습니다. 이는 큰 씬에서 반복되는 패턴처럼 보이는 현상을 줄이는 데 도움이 됩니다.

Unity 2D 샘플의 타일맵

효율적인 환경 디자인

Rule Tile은 해피 하비스트 전반에서 광범위하게 사용됩니다(예: 풀밭이 있는 토양 패치 생성). 샘플의 프로젝트 창에는 **Palette_Tiles**라는 타일 팔레트가 있으며, 팔레트의 첫 번째 행에는 Rule Tile이 있습니다. 프로젝트를 다운로드하고 연 뒤 인스펙터에서 게임 오브젝트를 선택하여 에셋이 프로젝트 창에 표시되도록 한 다음, 이 에셋을 선택하여 Rule Tile 설정을 확인할 수 있습니다.

참고: 프로젝트에서 가능한 경우 이러한 타일 에셋을 재사용하거나 텍스처 위에 페인팅하여 시간을 단축할 수 있습니다. 이 샘플과 함께 제공되는 스프라이트를 변경할 수도 있고, 게임의 필요에 맞게 새로운 Rule Tile을 생성할 수도 있습니다.



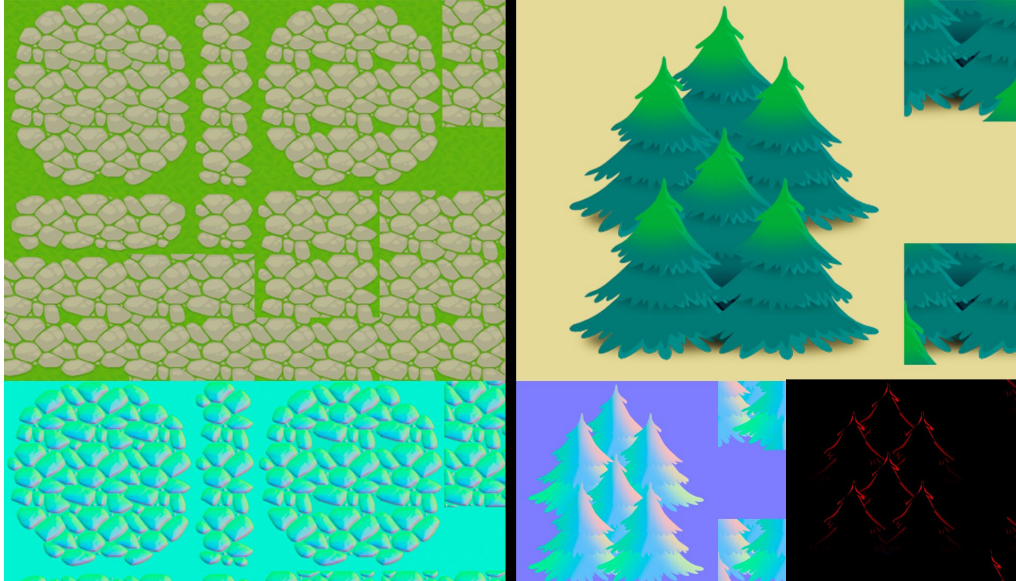
Rule Tile이나 AutoTile을 설정한 후에는 연속 경로를 페인팅하는 것이 훨씬 더 쉽고 시간 면에서도 효율적입니다.

이 샘플은 AutoTile 기능이 도입되기 전에 만들어졌지만, 이전 섹션에서 설명한 대로 AutoTile을 통해 동일한 기능을 복제할 수 있습니다.

타일맵의 보조 텍스처

해피 하비스트에서 Tilemaps 폴더 아래에 있는 모든 타일맵에는 각각 대응되는 [노멀 맵](#)과 [마스크 맵 텍스처](#)가 있으며, 이 텍스처들은 같은 크기와 동일한 레이아웃을 공유하되 별도로 페인팅되어 조명 효과를 표현합니다.

노멀 맵과 마스크 맵 텍스처는 스프라이트 에디터에서 타일 세트의 메인 텍스처에 추가됩니다. 샘플에서는 마스크 맵을 사용하여 캐릭터와 프랍의 림 라이트 실루엣 효과를 만듭니다. 그러나 대부분의 타일은 지면에 사용되므로 림 라이트 효과가 필요하지 않습니다. 때문에 이 타일 세트의 텍스처는 대부분 검은색이며, 림 효과를 위해 생성된 빛을 반사하지 않습니다. 다만 건물의 지붕에 사용되는 건물 타일에 대해서는 예외가 적용되었는데, 이 경우 이러한 타일의 가장자리를 강조해야 했습니다.



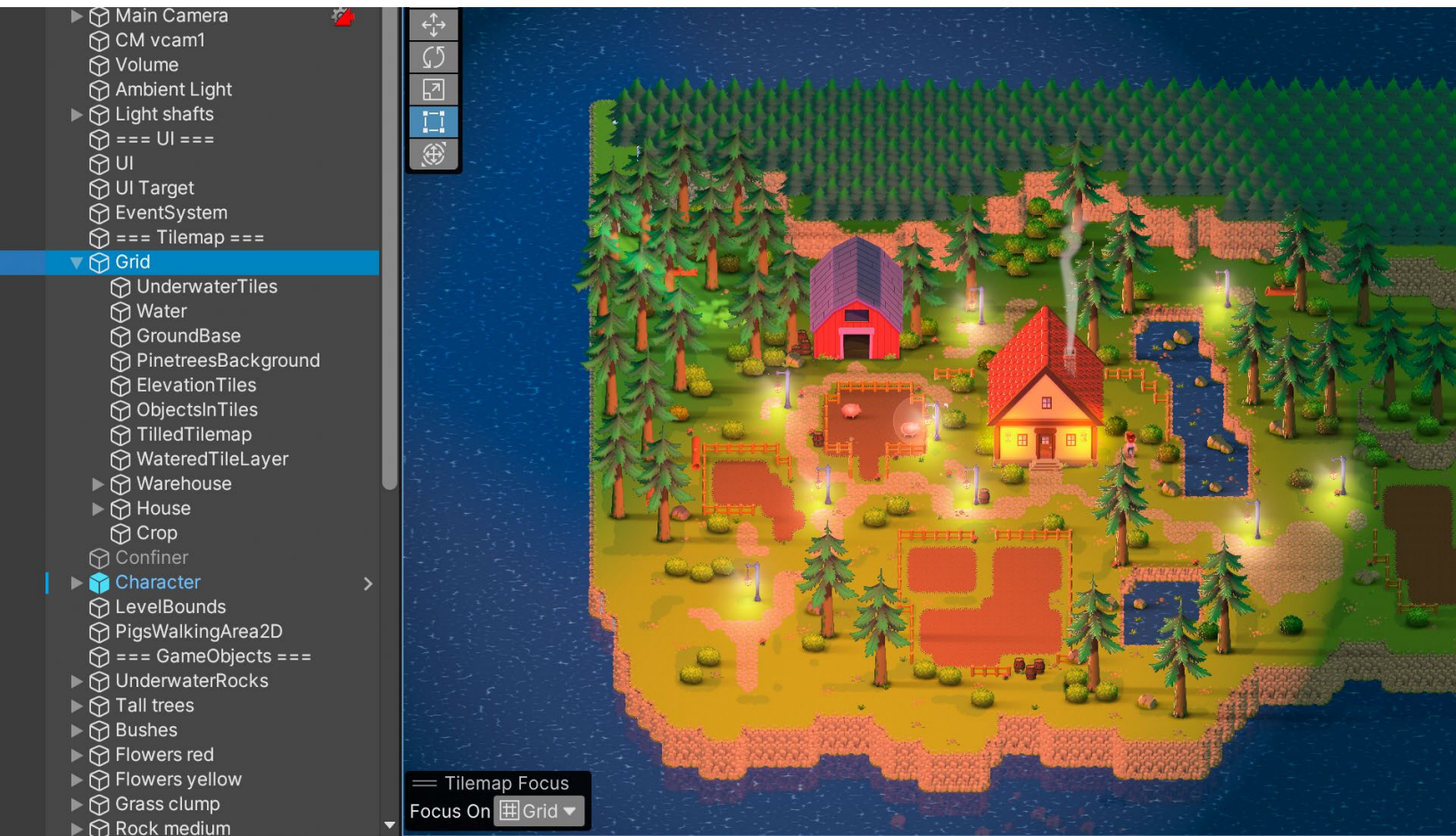
돌길에 사용된 비교적 복잡한 타일 시트(왼쪽)와 배경 프랍에 사용된 비교적 단순한 타일 시트(오른쪽)

타일맵을 이용한 레벨 및 게임플레이 디자인

해피 하비스트의 모든 타일은 크기와 셰이프가 동일하며 같은 Grid 게임 오브젝트에 포함되어 있습니다. 이렇게 하면 게임 내 타일맵의 수를 적게 유지할 수 있습니다. 또한 **Terrain Manager**라는 컴포넌트가 Grid 게임 오브젝트에 연결되어 있으며 일부 게임플레이 기능을 제공합니다(다음 섹션 참조).

다음은 Grid 게임 오브젝트 아래의 일부 타일맵입니다.

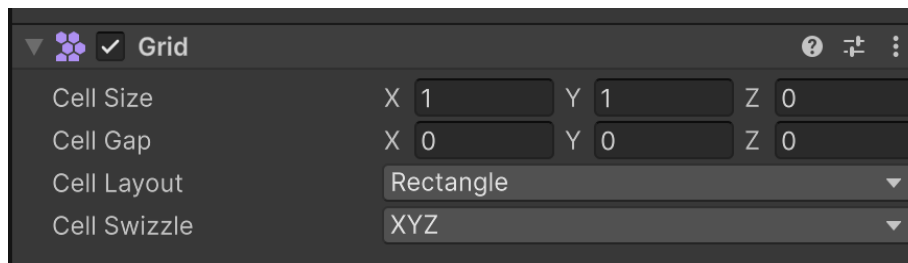
- **UnderwaterTiles:** 이 타일은 물 아래에서 보이는 타일, 절벽, 연못 영역의 초록색 지면에 사용됩니다.
- **Water:** 이 타일은 물 애니메이션을 시뮬레이션하기 위해 ShaderGraph에서 만들어진 Spite-Lit-Material 셰이더를 사용합니다.
- **GroundBase:** 길, 잔디, 진흙 타일과 함께, 높낮이가 서로 다른 타일 사이의 빈 공간을 채우기 위한 돌 타일로 구성된 지면 아트웍입니다.
- **ObjectsInTiles:** Tile Palette의 [GameObject Brush](#)를 사용하여 타일맵에 페인팅된 게임 오브젝트에 사용됩니다(예: 울타리).
- **TilledTilemap:** 이 타일은 코드에서 사용되어 씨앗을 심을 수 있는 타일을 감지합니다.
- **WateredTileLayer:** 이전 타일맵과 유사하게, 게임 로직에 의해 사용되어 타일이 젖어 보이도록 합니다. 타일이 마르면 '젖은' 타일이 제거됩니다.
- **Warehouse, House:** 이 타일맵은 창고와 집을 만드는 데 사용됩니다. 타일맵으로 건물을 만들면 탁월한 유연성으로 더 효율적인 재구성과 크기 조정이 가능하며, 건물의 일부가 동일한 텍스처를 사용하기 때문에 텍스처 공간을 줄일 수 있습니다.
- **Crop:** 모든 식물을 하나의 게임 오브젝트 아래에 유지하는 데 사용되며, 터레인 관리자에서 사용합니다.



타일맵이 포함된 Grid 게임 오브젝트. Tilemap Focus 툴을 사용하면 선택한 타일맵의 콘텐츠만 씬 뷰에 그려서 현재 작업 중인 타일맵에 집중할 수 있습니다.

팁: 타일맵에서 텍스처 번짐 피하기

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 않음). 타일 시트를 사용하지 않는 경우 스프라이트를 [Sprite Atlas](#)에 패킹하는 것이 좋습니다. 이렇게 하면 타일의 가장자리에 투명한 끝부분이 없도록 하여 경계를 부드럽게 만들 수 있습니다. 또한 타일맵 렌더러의 내부 정렬 기능을 활용하면 타일을 약간 크게 표시하여 타일 사이에 생기는 틈을 방지하고 약간 겹치도록 만들 수도 있습니다.



발생 가능한 틈을 완화하는 데 도움이 되는 Grid 컴포넌트의 몇 가지 옵션



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.



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

픽셀 아트로 작업하는 경우, 스프라이트의 필터 모드를 Point(필터 없음)로 변경하면 타일이 완벽하게 일치하여 최대한 틈이 남지 않습니다.

게임플레이를 위한 Tilemap API

타일맵은 그리드 기반 게임에 적합합니다. API는 저작 및 게임 로직을 위한 강력한 레벨 디자인 툴로, 이를 활용해 시각적 그리드에서 간편하게 타일을 읽고 쓸 수 있습니다. 일반적으로 그리드 기반 게임 플레이는 코드 내에서만 존재하는 다차원 리스트나 딕셔너리를 사용하며, 일종의 시각적 구현 및 저작 툴을 필요로 합니다. 타일맵은 이러한 모든 내용을 기본적으로 제공합니다.

해피 하비스트의 Tilemap API

Grid 게임 오브젝트 내부에 있는 **Terrain Manager** 스크립트는 타일의 변화를 처리하고 타일을 사용하여 농작물을 추적하는 주요 관리자입니다. 이는 그리드 기반 게임 플레이를 설정하는 데 유용한 **Tilemap API**를 사용하며, 이를 활용해 2차원 위치에서 항목의 위치를 쉽게 식별할 수 있습니다.

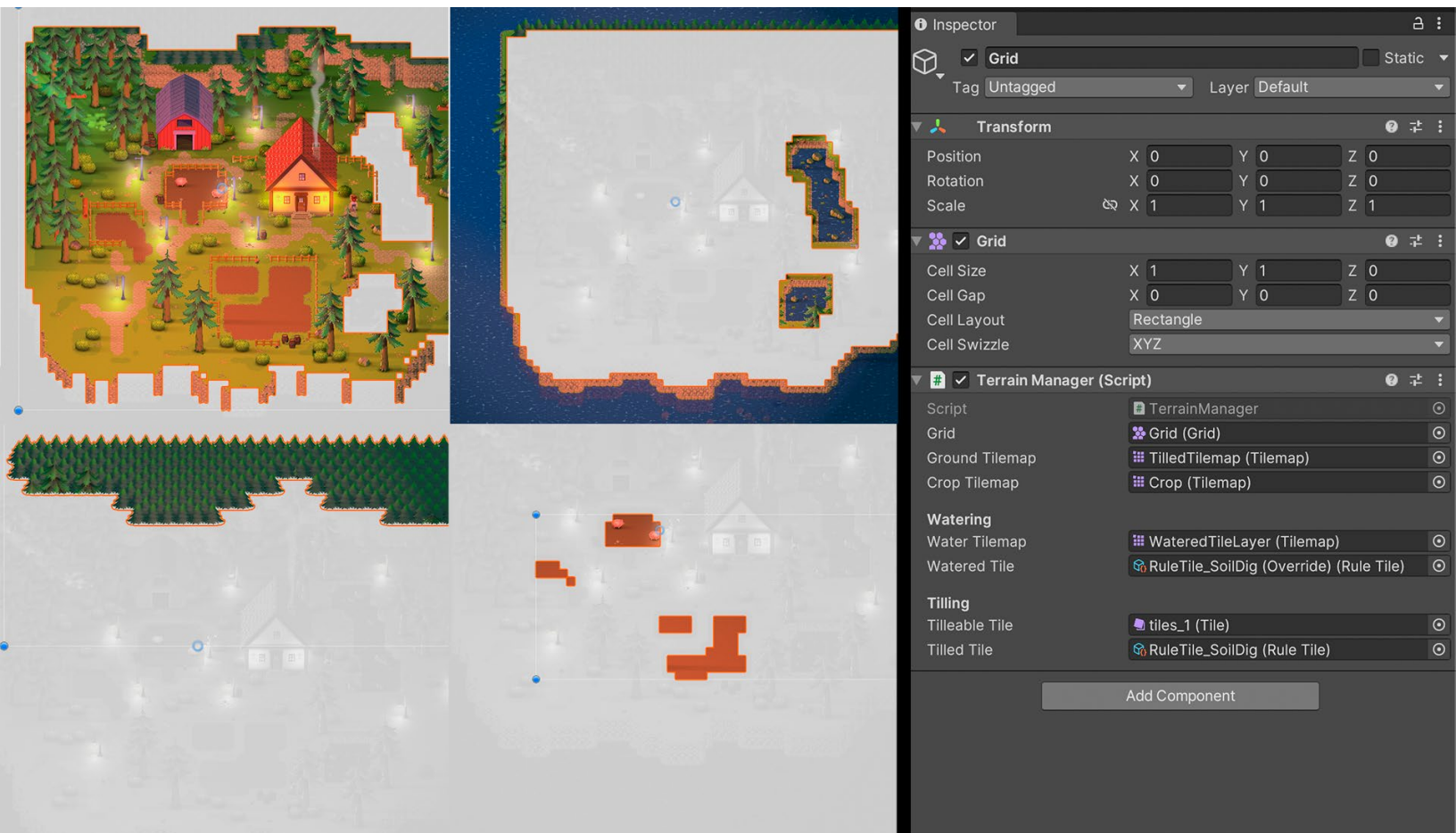
이 MonoBehaviour 클래스는 **GroundData**와 **CropData**라는 두 개의 제네릭 클래스를 생성합니다. 여기에는 타일이 젖은 것처럼 보이는 시간, 식물이 자라는 속도, 수확하지 않은 작물이 얼마나 오래 지속되는지와 같은 게임플레이 관련 변수가 포함됩니다.

Terrain Manager 스크립트는 다음과 같이 게임플레이 용도로 사용되는 타일맵과 타일을 참조합니다.



- **Ground Tilemap:** 플레이어가 땅을 파고 씨앗을 심을 수 있는 위치를 나타내는 토양 타일이 포함된 타일맵을 참조합니다. 이 타일맵 외부의 타일에서 땅을 파면 아무 일도 일어나지 않습니다.
- **Crop Tilemap:** 작물의 부모 오브젝트로 사용될 Tilemap 게임 오브젝트를 참조합니다. Tilemap API를 사용하면 런타임 시 이 타일맵의 작물 타일을 배치, 업데이트 및 제거할 수 있습니다.
- **Water Tilemap:** 물을 준 상태를 시뮬레이션하기 위해 경작된 토양 타일을 페인팅하는 데 사용되며, 해당 타일이 심기와 재배가 가능한 상태임을 플레이어에게 시각적으로 전달합니다.
- **Watered tile:** 물 그래픽이 있는 [규칙 오버라이드 타일](#)로, 새로운 규칙을 설정하지 않고 규칙 타일의 배리에이션을 제공하는 타일을 생성할 수 있게 합니다. 이는 경작된 토양 타일의 규칙을 오버라이드합니다.
- **Tilleable tile:** 경작할 수 있는 타일을 식별하는 데 사용됩니다.
- **Tiled tile:** 경작된 토양의 시각적 요소를 페인팅하는 데 사용되는 Rule Tile입니다.

Terrain Manager 스크립트는 Tilemap API를 사용하여 VectorInt 위치 형식으로 타일 정보를 읽고 타일을 적절히 업데이트하는 함수를 포함합니다. 이러한 함수는 PlayerController가 이들에 대한 함수를 트리거할 때 다양한 툴에서 호출됩니다.

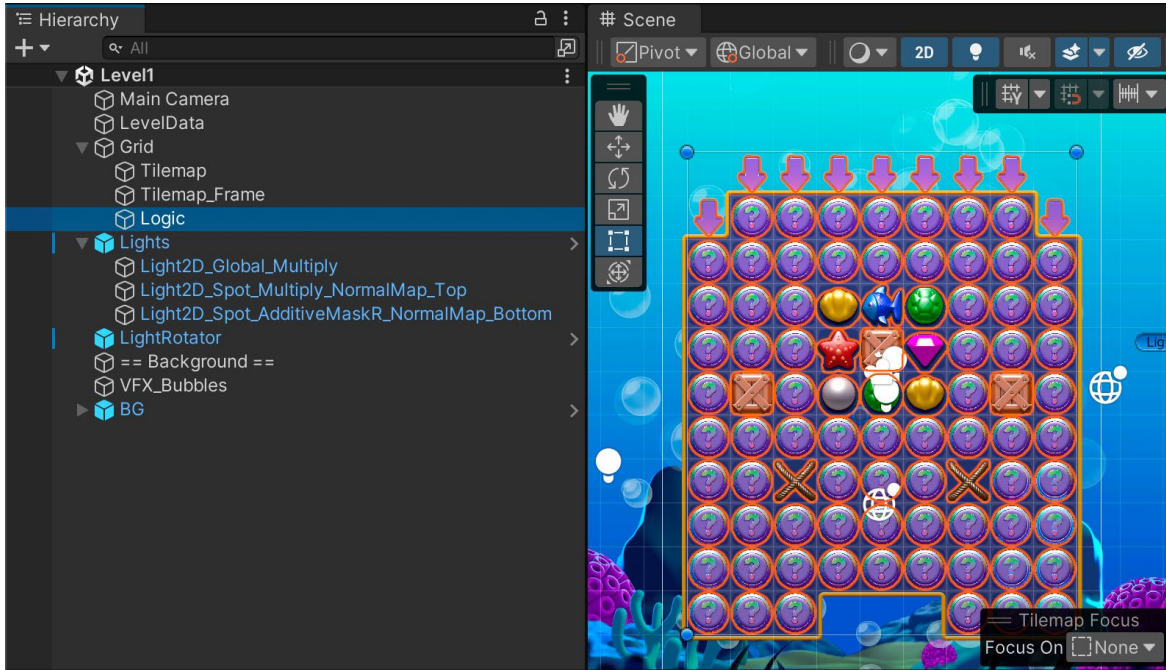


게임플레이 로직을 위해 일부 타일맵을 사용하는 Terrain Manager 시스템



젬 헌터 매치의 Tilemap API

젬 헌터 매치는 공식 Unity 샘플로, 모바일용으로 설계된 매치3 퍼즐 게임이지만 여러 플랫폼에서 실행할 수 있습니다.



젬 헌터 매치의 레벨 씬 구조는 단순합니다.

레벨 구조를 살펴보면 시각적 에셋뿐만 아니라 게임 로직에도 사용되는 타일맵을 찾을 수 있습니다. 이러한 타일맵은 다음과 같습니다.

- **Grid 게임 오브젝트:** 이 게임 오브젝트는 **Board**라는 스크립트를 포함하고 있으며, 이는 레벨에서 사용 가능한 보석의 유형과 VFX Graph 기반의 시각 효과에 대한 레퍼런스를 결정합니다.
- **자식 게임 오브젝트:** Grid 게임 오브젝트에는 몇 가지 자식 게임 오브젝트가 포함되어 있습니다. 자식 게임 오브젝트는 다음과 같습니다.
 - **Tilemap:** 배경 타일과 장식을 생성하는 데 사용되며, **Tilemap API**에는 많은 요소의 위치를 추적하는 편리한 기능이 있습니다.
 - **Tilemap_Frame:** 배경 타일 주위의 프레임 장식을 만드는 데 사용됩니다.
 - **Logic:** 레벨 디자인을 위한 플레이스홀더 타일을 포함하며, 여기에서 모든 플레이스홀더 '무작위 보석' 타일(물음표로 표시됨)이 실제 게임플레이 게임 오브젝트(코드를 통해 무작위로 생성됨)로 교체됩니다. 그러나 팔레트의 다른 타일을 사용하여 특정 타일 위치에 생성할 보석 또는 아이템의 유형을 선택할 수도 있습니다.

[이 동영상](#)에서 타일맵 베스트 프랙티스를 자세히 알아보세요.

2D Sprite Shape

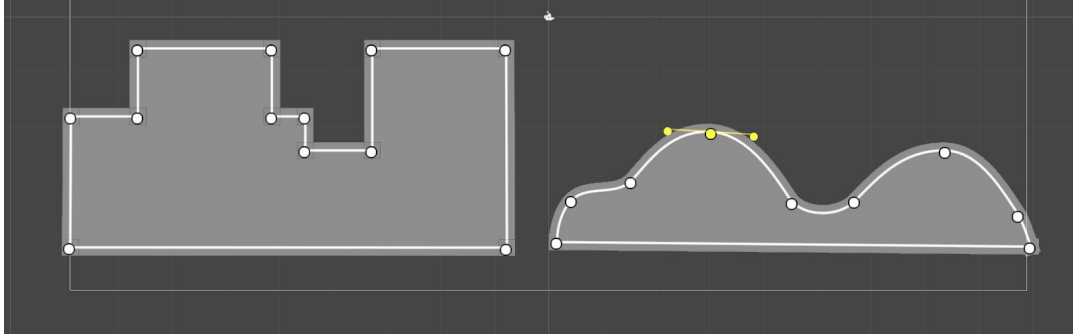
2D Sprite Shape 톨은 셰이프의 외곽선을 따라 스프라이트 타일링을 만들되, 외곽선의 각도에 맞게 스프라이트가 자동으로 변형되고 다른 스프라이트로 교체되도록 합니다. 또한 2D Sprite Shape 게임 오브젝트에 Fill 텍스처를 할당하면, 타일 텍스처로 채워진 셰이프를 생성하여 배경이나 다른 대형 레벨 구성 프랍으로 활용할 수 있습니다.



버니 블리츠는 새로운 Unity 2D 샘플 프로젝트로, 배경의 덩굴 및 언덕과 같이 2D Sprite Shape로 만들어진 장식 요소가 있습니다.

2D Sprite Shape를 사용하여 생성한 경로는 그래픽스 소프트웨어에서 사용하는 주요 펜 툴과 비슷하게 작동합니다. 이 경로는 씬에서 직접 편집할 수 있고, 필요에 따라 닫거나 타일링 텍스처로 채울 수 있는 베지어 커브입니다.

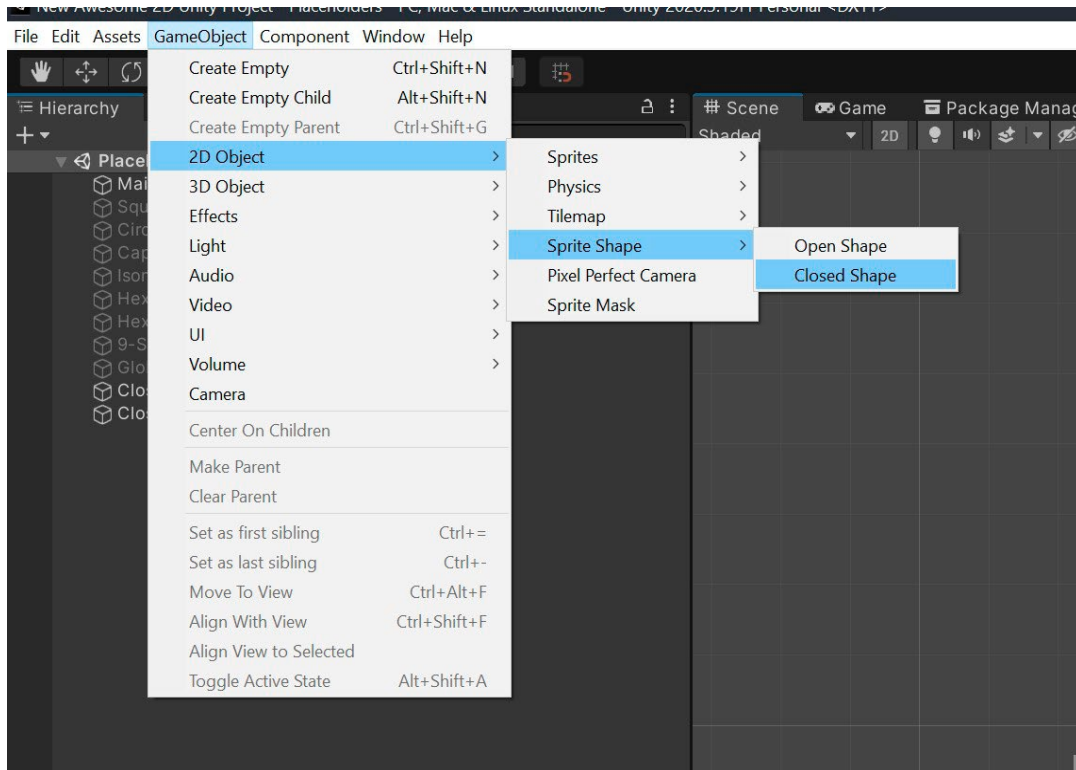
Sprite Shape는 새로운 스프라이트 셰이프를 효율적으로 생성하고 편집할 수 있는 Unity의 편리한 프로토타이핑 기능입니다.



Sprite Shape에 사용할 수 있는 두 가지 탄젠트 모드: 선형 및 연속

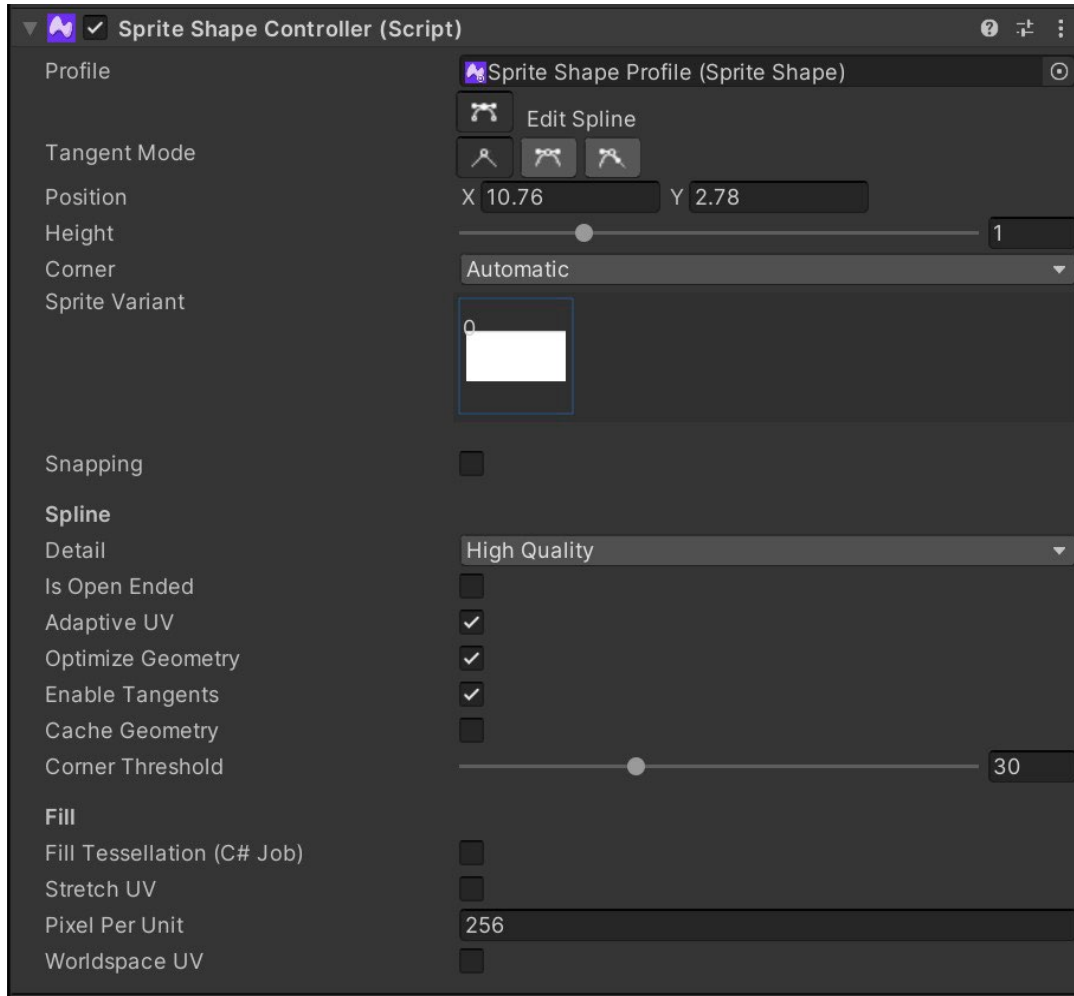
2D 템플릿을 사용하여 2D 프로젝트를 만드는 경우 Sprite Shape가 기본적으로 설치되어 있습니다. 설치되어 있지 않다면 패키지 관리자를 통해 설치할 수 있습니다.

새로운 스프라이트 셰이프를 생성하려면 GameObject 메뉴를 클릭하고 **2D Object > Sprite Shape**에서 Open Shape 또는 Closed Shape를 선택합니다.



스프라이트 셰이프 생성

스프라이트 셰이프를 수정하려면 셰이프를 선택하고 인스펙터에서 Edit 버튼을 클릭합니다.



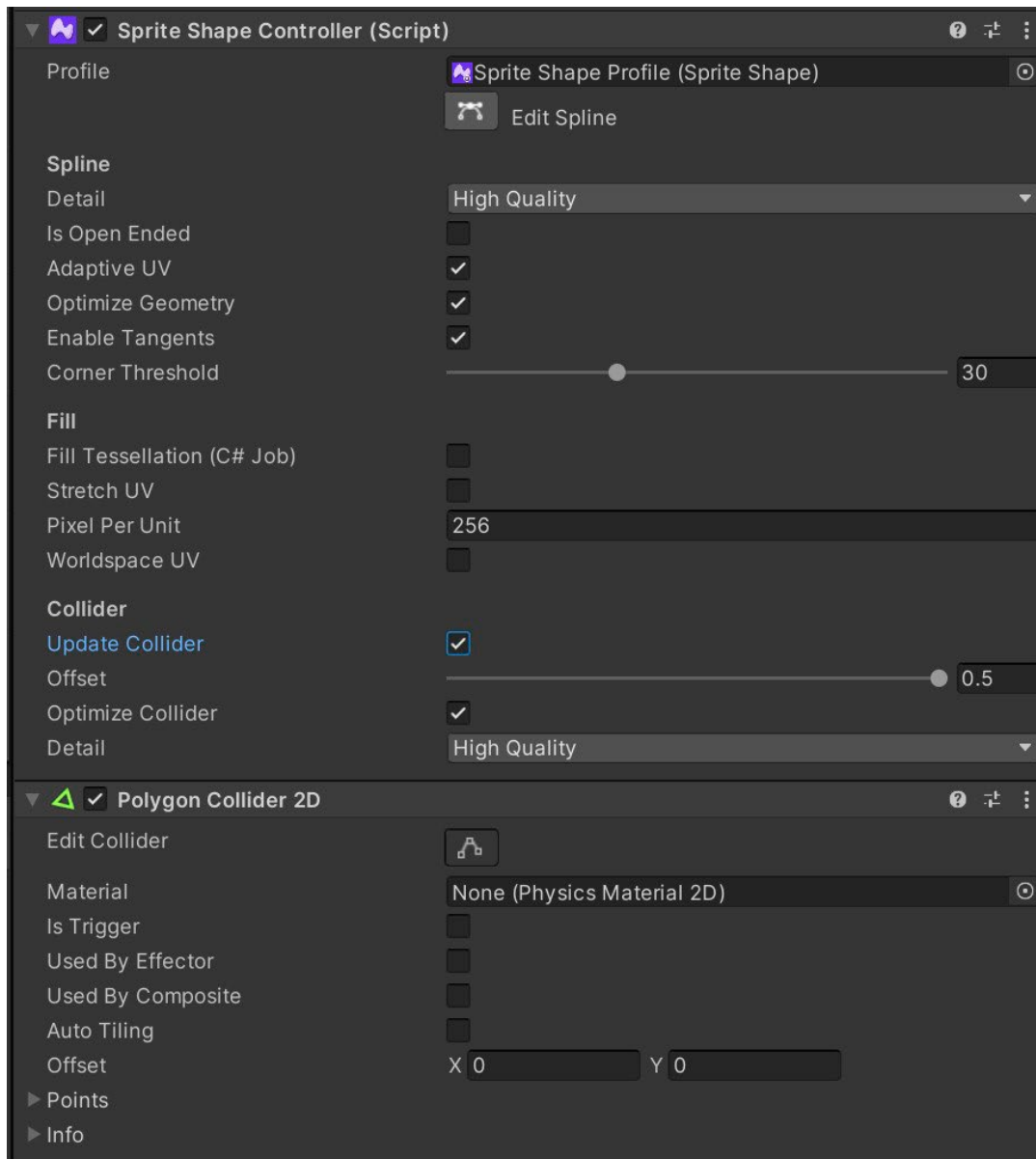
인스펙터 창에서 스프라이트 셰이프 수정

스프라이트 셰이프의 아무 부분이나 클릭하고 포인트를 추가하거나 Delete 키로 삭제합니다. 포인트를 선택하면 세 개의 **Tangent Mode** 버튼 중 하나를 선택하여 모드를 변경할 수 있습니다.

- **선형**: 커브를 만들지 않고 포인트 양쪽을 직선으로 연결합니다.
- **연속**: 핸들이 서로 반대 방향을 향하도록 포인트 주변에 커브를 생성합니다.
- **불연속**: 핸들이 독립적으로 움직이도록 포인트 주변에 커브를 생성합니다.

포인트 주변의 코너 모양을 선택할 수도 있습니다. 스네핑 톨은 그리드에 포인트를 스냅하는 또 다른 유용한 옵션입니다.

물리용 **Polygon Collider 2D** 컴포넌트를 추가합니다. 프로토타이핑이 목적이라면 기본 프로퍼티로 충분합니다.



Sprite Shape 게임 오브젝트에 Polygon Collider 2D 컴포넌트를 추가합니다.

이 단계에 따라 생성한 스프라이트 셰이프는 상호 작용이 가능하며 인스펙터에서 수정할 수 있습니다.

스프라이트 셰이프 레벨 디자인

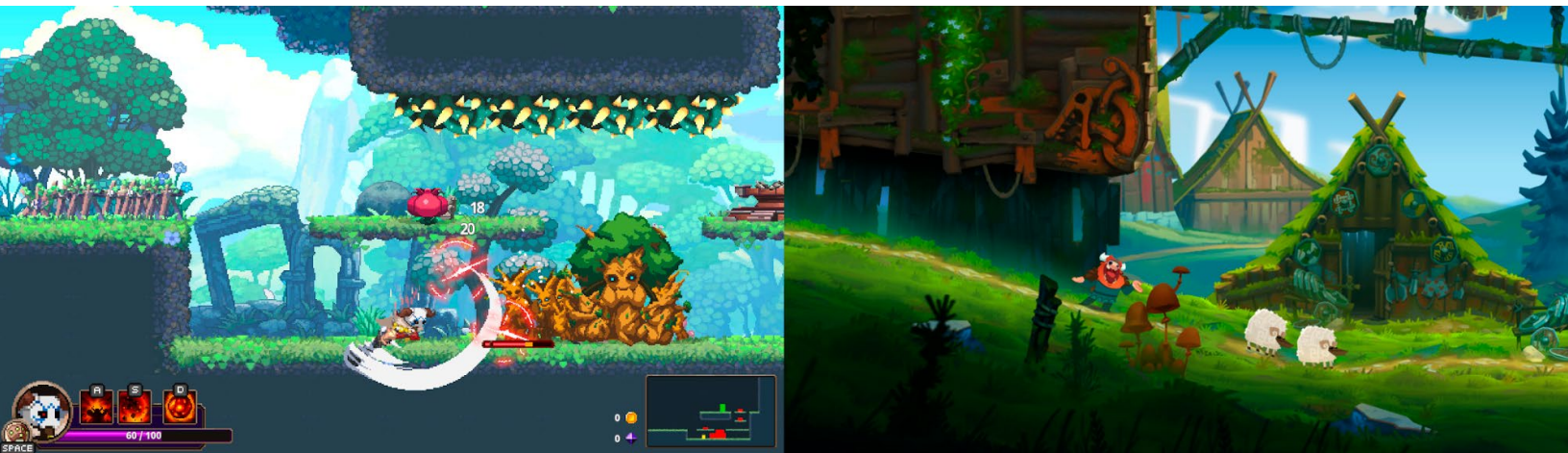
그리드 또는 타일 기반 디자인은 특히 픽셀 아트를 사용할 때 레트로 2D 비주얼 스타일에 잘 어울립니다. 또한 횡스크롤 수평 플랫폼 게임이나 탑 다운 RPG에서 활용하기 좋으며, 건물이나 성처럼 규칙적인 셰이프의 구조물을 만드는 데에도 유용합니다.



그리드 기반 디자인을 사용하면 더 간편하게 경로를 탐색하거나 레벨을 생성할 수 있습니다. 그리드로 경로를 제한하면 더 간단하게 경로를 탐색할 수 있습니다. 레벨을 디자인할 때도 일정한 거리를 유지하기 더 쉽습니다. 예를 들어 캐릭터의 점프 높이가 3유닛인 경우 플랫폼 위치를 쉽게 계획할 수 있으며, 플레이어는 점프할 수 있는 높이인지 즉각적으로 알 수 있습니다.

블록 기반 스타일을 벗어나 유기적인 형태의 레벨을 만들고 싶다면 2D Sprite Shape를 사용하면 됩니다. 커브를 사용하여 원하는 셰이프를 만들 수 있습니다. 터레인, 언덕, 잔디밭, 매끄러운 표면 등은 모두 스프라이트 셰이프를 사용하는 것이 좋으며 선명한 픽셀 아트 스타일에 비해 씬에 더 현대적인 느낌을 더할 수 있습니다.

스내핑 툴을 활성화하면 각도나 커브를 쉽게 설정하고 요소를 정확한 위치에 배치할 수 있습니다.



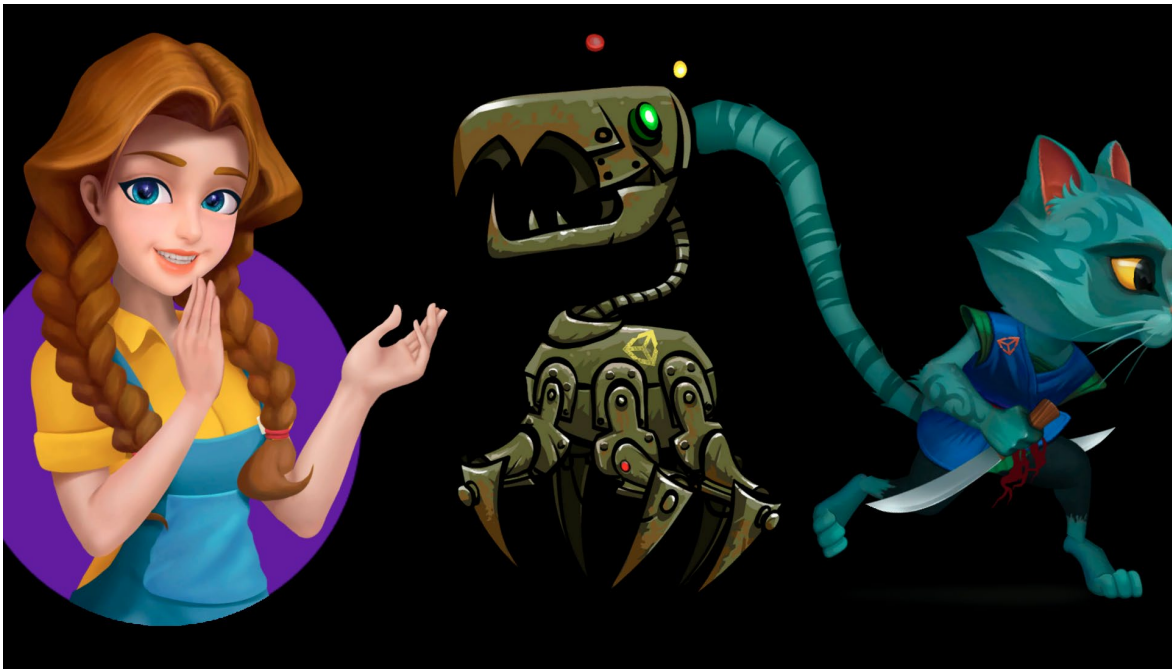
왼쪽에 있는 SouthPAW Games의 Skul: The Hero Slayer는 타일맵 기반 게임인 반면, 오른쪽에 있는 Mobge LTD의 오드마는 스플라인 곡선 기반 게임입니다.

스프라이트 셰이프로 만든 레벨과 요소는 나중에 쉽게 편집할 수 있습니다. 시각적 스타일에 따라 게임에서 두 시스템을 모두 사용할 수도 있습니다.

스프라이트 셰이프의 Fill 또는 Border 머티리얼에 Shader Graph 스프라이트 기반 셰이더를 사용하면 원하는 스타일을 적용할 수 있으며, 더 나아가 생성된 셰이프를 수정하여 더욱 다양하게 변경하는 것도 가능합니다. Unity 6부터는 [Custom Geometry](#) 기능이 [Sprite Shape Controller](#)에서 제공되어, 커스텀 스크립트를 사용하여 Sprite Shape 지오메트리를 생성하거나 수정할 수 있습니다.

타일맵과 스프라이트 셰이프는 모두 런타임 중에 수정할 수 있으므로 흥미롭고 새로운 게임플레이를 만드는 데 도움이 됩니다. 예를 들어 타일맵을 파괴하거나 스프라이트 셰이프를 변형하는 등 많은 가능성을 발휘할 수 있습니다. 두 옵션에 대한 자세한 내용은 [Tilemap API](#) 및 [SpriteShape API](#) 기술 자료에서 확인해 보세요.

2D 애니메이션: 골격 애니메이션



Unity의 2D 애니메이션 기능을 사용하여 다양한 장르와 스타일의 캐릭터를 애니메이션화할 수 있습니다. Unity 에셋 스토어에서 무료 2D Animation Samples 패키지를 다운로드하고 더 많은 정보를 알아보세요.

2D 애니메이션은 게임 아트 개발 과정에서 가장 까다롭고 시간이 많이 걸리는 부분일 수 있습니다. 애니메이션 캐릭터를 제작하려면 애니메이션 타이밍, 모멘텀, 운동학 등 애니메이션에 관한 지식이 필요합니다. 모든 프레임을 다듬는 과정에서 오랜 시간이 걸리기도 하며 프레임을 저장하고 표시하려면 많은 메모리를 사용해야 합니다. 타이밍이나 캐릭터의 일부분을 변경하면 모든 프레임을 다시 드로우해야 할 수도 있습니다.

원래 3D 애니메이션이 2D 애니메이션보다 더 간단한 편입니다. 3D 모델을 만들고, 골격을 추가하여 리깅하고, 뼈대 가중치를 설정하고, 키프레임을 설정하여 애니메이션을 적용하면 소프트웨어 보간이 이루어집니다. 이후 키프레임을 편집하여 애니메이션을 수정하게 되는 식입니다.



스프라이트 에디터에서 해피 하비스트 캐릭터의 표정 리깅 테스트

Unity는 여러 면에서 3D 애니메이션 프로세스를 모방하는 효율적인 **2D 애니메이션** 툴셋을 제공합니다. 예를 들면 Photoshop에서 직접 Unity로 캐릭터 아트워크를 임포트할 수 있습니다.

디자인, 임포트 및 릿

게임에서 가장 중요한 요소는 바로 플레이어의 캐릭터입니다. 따라서 캐릭터를 디자인할 때는 충분히 고민해야 합니다. 캐릭터 디자인에 앞서 몇 가지 중요한 포인트를 고려해야 합니다.

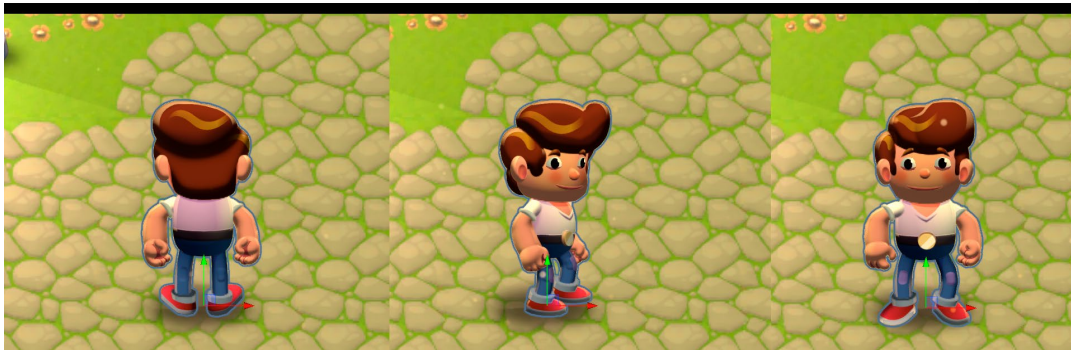
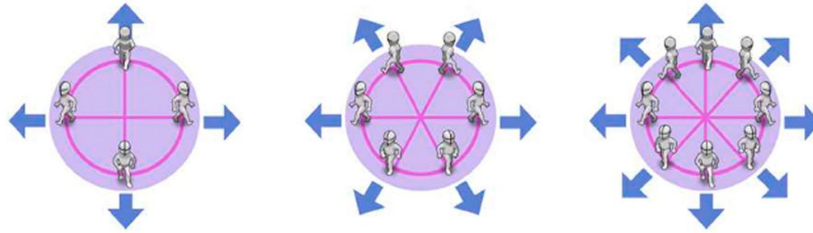
시점

게임에 활용하는 시점은 캐릭터의 모습과 애니메이션 적용 방식에 영향을 줍니다. 대부분의 게임 뷰에서 캐릭터를 옆모습으로 나타낼 수 있는데, $\frac{3}{4}$ 뷰로 살짝 회전하면 얼굴의 특징을 더 많이 표시할 수 있습니다.



아이소메트릭이나 ¾ 탑 다운 시점을 사용하는 경우 비슷한 뷰를 만들 수 있지만 얼굴을 세부적으로 보여 주기 위해 카메라를 위에서 약간 기울여도 좋습니다. 이 시점에서는 캐릭터가 여러 방향을 바라보게끔 나타냅니다. 원하는 결과물과 예산에 따라 일반적으로 사용되는 다음 세 가지 옵션 중 하나를 선택하면 됩니다.

- 4방향
- 6방향
- 8방향



해피 하비스트: 2D 탑 다운 또는 아이소메트릭 게임을 만들 때 구현할 수 있는 다양한 캐릭터 회전

어떤 옵션을 선택하든 왼쪽 또는 오른쪽 중 어느 방향을 뒤집어서 애니메이션이 반대쪽을 향하게 할 것인지 결정해야 합니다. 애니메이션을 반전시킬 때 드로우해야 할 부분은 다음과 같습니다.

- 3방향: 오른쪽, 위, 아래
- 3방향: 오른쪽, 오른쪽 위, 오른쪽 아래
- 5방향: 오른쪽, 오른쪽 위, 위, 오른쪽 아래, 아래

애니메이션을 반전시키면 캐릭터의 손도 그에 맞춰 반전됩니다. 캐릭터가 오른쪽을 바라보고 오른손에 검을, 왼손에 방패를 들고 있는 경우, 왼쪽을 바라보도록 캐릭터를 반전시켰을 때 검과 방패의 손 위치가 바뀌게 됩니다. 이 부분을 감안하고 수용할 것인지, 아니면 시간을 더 들여서 모든 애니메이션이 올바른 방향을 향하도록 만들지 정해야 합니다.

기존의 횡스크롤 격투 게임을 디자인하는 경우 한 방향만 사용하면 됩니다. 캐릭터가 위아래로 움직이면 옆으로 움직이는 것처럼 보입니다.

테스트를 거친 후 게임 환경에서 캐릭터가 자연스러워 보일지는 결국 여러분에게 달려 있습니다. 사실적인 시야각과 시점을 적용하는 것보다 중요한 것이 바로 개발자의 육안입니다.



캐릭터 스킨 변경

2D Animation 패키지의 이점은 여러 캐릭터가 동일한 골격과 애니메이션을 공유할 수 있다는 점입니다. 기본 캐릭터 하나의 디자인과 리깅, 애니메이션화를 마쳤다면 스킨만 교체하면 됩니다. **골격 공유**라는 기능으로 이 프로세스를 간소화할 수 있습니다.

미리 작업을 계획해서 시간을 단축하세요. 동일한 골격을 공유할 모든 캐릭터의 컨셉화를 그린 후 골격이 모든 캐릭터에 맞는지 확인하세요. 별도의 레이어에 골격을 그리고 모든 캐릭터에 오버레이하면 됩니다. 모든 캐릭터의 레이어 개수가 동일해야 한다는 점을 기억해 두세요.

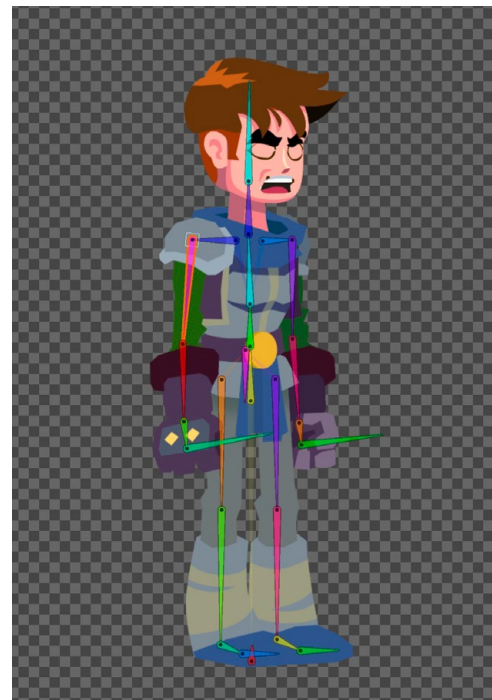


이미지 편집 애플리케이션의 골격 그림 오버레이

일반적인 2D 애니메이션 규칙

리깅과 애니메이션화가 예정된 캐릭터를 디자인할 때 참고할 만한 규칙은 다음과 같습니다.

- 팔과 다리를 곧게 뻗어 중립적인 포즈를 취하고 있는 캐릭터를 그리세요. 신체 부위를 구부정하게 그리면 애니메이션화할 때 문제가 발생할 수 있습니다.
- 게임의 PPU에서 권장하는 것보다 조금 더 높은 해상도로 제작하세요. 정지 상태에서는 해상도가 좋아 보일 수 있지만 이미지를 회전하거나 늘리면 픽셀화 현상이 나타날 수 있습니다.
- 나중에 2D 광원 챗터에서 소개할 팁이지만, 노멀 맵을 사용할 계획이라면 스프라이트에 명암을 칠하지 않는 것이 좋습니다. 대신 방향이 없는 그림자를 칠해 두세요. 이 기술을 앰비언트 오클루전이라고 합니다. 방향 광원으로 태양광 같은 연출을 넣으면 스프라이트가 더 멋져 보일 수는 있으나 사용하지 않기를 권장합니다.



중립 캐릭터 포즈의 예시: 팔다리는 어떤 위치에도 있을 수 있지만 구부러지면 안 됩니다. 팔다리가 구부러지면 픽셀 스트레칭을 유발합니다.

- **스프라이트 스왑** 기능을 사용하여 교체되는 신체 부위 레이어는 적절한 그룹으로 묶어야 합니다. 예를 들어 입 위치와 관련된 모든 레이어는 이미지 편집 앱에서 ‘입’이라는 그룹에 배치되어야 합니다.

이렇게 디자인 시 고려할 내용에 대해 알아보았으니, 선호하는 그래픽 앱을 열고 아트를 제작해 보세요.

Unity로 캐릭터 импорт

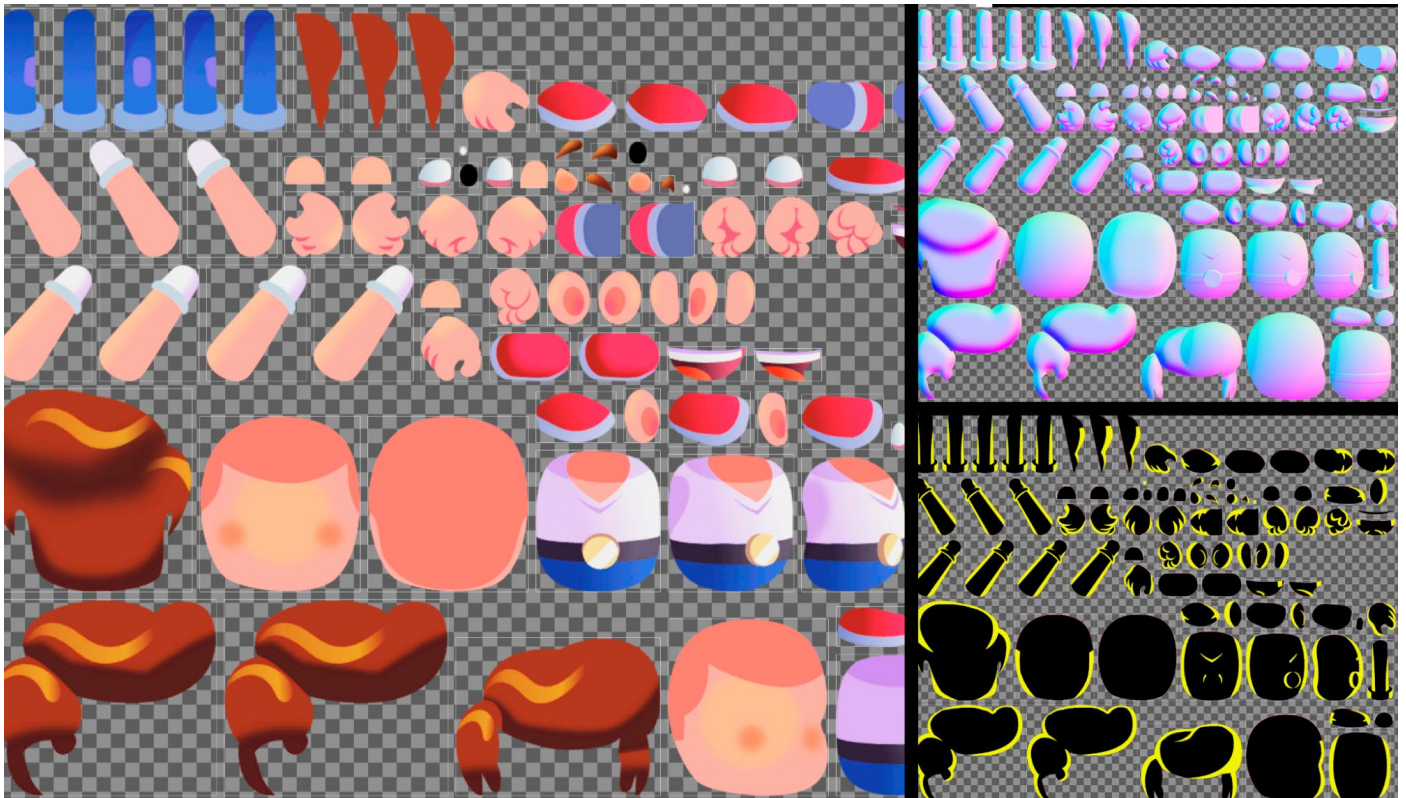
2D Animation 패키지를 사용하면 캐릭터 아트웍을 Photoshop에서 Unity로 바로 импорт할 수 있습니다.

PSD Importer 패키지는 이 목적으로 레이어가 있는 PSD 및 PSB 파일을 처리하는 기능을 제공합니다.

이 방법을 사용하면 캐릭터의 모든 레이어를 스프라이트로 импорт하여 앱에서 페인팅한 대로 배치해 줍니다.

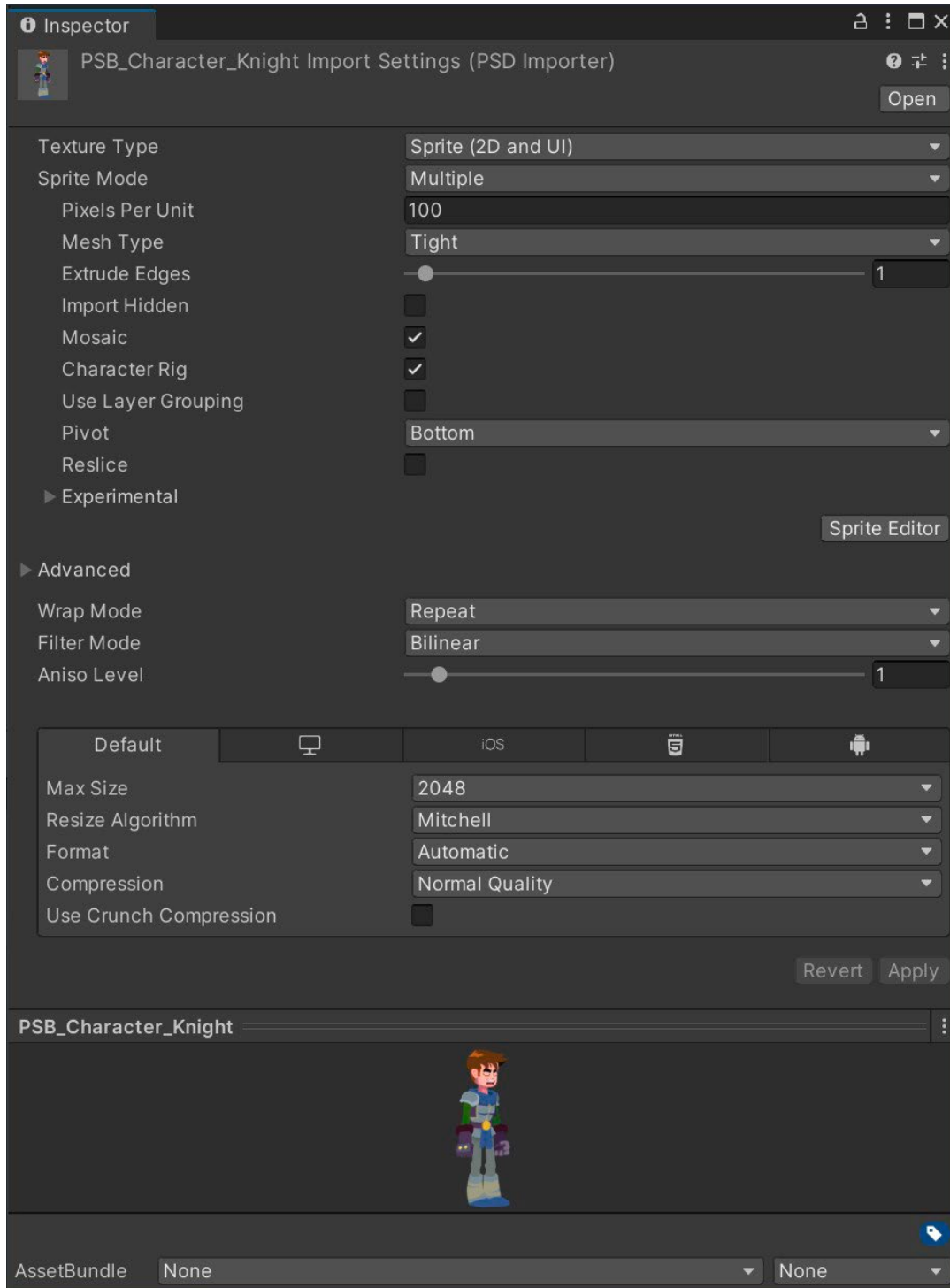
Unity에서 2D 프로젝트 템플릿을 사용하는 경우 이 임포터가 이미 설치되어 있습니다. 설치되어 있지 않다면 패키지 관리자를 사용하여 설치하세요.

익스포트할 때는 파일을 PSD 또는 PSB 포맷으로 저장하면 됩니다. PSD는 Photoshop 및 기타 제작 툴에서 일반적으로 사용됩니다.



주요 캐릭터 부위가 레이어로 구성된 임포터 PSD의 스프라이트 시트: 오른쪽 이미지에는 조명 목적으로 보조 텍스처로 추가된 PSD 파일이 나와 있습니다.

Unity로 임포트하는 작업은 다른 에셋 작업과 동일합니다. 파일을 Assets 폴더에 저장하거나 프로젝트 창으로 드래그하세요.



캐릭터 인스펙터

임포트한 PSD 또는 PSB 파일을 선택하면 스프라이트 임포트 설정과 비슷한 옵션이 2D 애니메이션 및 리깅과 관련된 추가 설정과 함께 표시됩니다. 설정해야 할 주요 옵션은 다음과 같습니다.

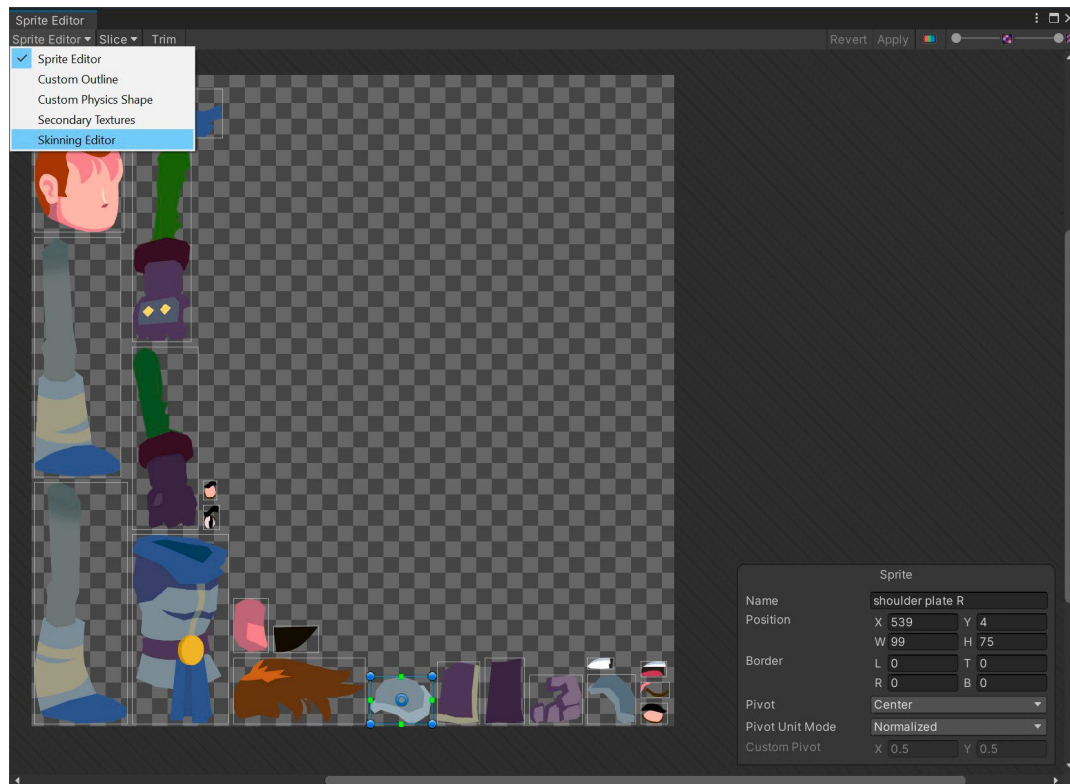
- **Import Hidden Layers:** 이 옵션은 숨겨진 레이어를 포함하여 PSD 또는 PSB 파일의 모든 레이어를 임포트합니다.
- **Individual sprites (Mosaic):** 이 설정은 Texture Type이 Multiple로 설정된 경우에만 사용할 수 있습니다. 임포트한 레이어에서 스프라이트를 만들고 텍스처 아틀라스에 배치하는 설정입니다. 캐릭터를 리깅하려면 이 옵션을 활성화하세요.
- **Use as Rig:** 이 옵션은 PSD 또는 PSB 파일과 동일한 레이어 계층 구조 및 위치를 유지하며 캐릭터 프리팹을 생성합니다. 반드시 활성화해야 합니다.
- **Use Layer Group:** PSD 또는 PSB 파일에서 레이어 그룹을 추가합니다. 이 옵션을 활성화하여 스프라이트가 교체된 부위 등 캐릭터의 부위를 그룹화할 수 있습니다.

Layer Management 섹션에서 레이어의 계층 구조를 확인하고 임포트할 레이어를 수동으로 추가하거나 제거할 수 있습니다. 옵션 설정이 끝나면 Apply를 클릭합니다. 그러면 캐릭터 프리팹이 완성되어 씬으로 드래그할 수 있습니다.

[Unity 에셋 스토어](#)에서 2D Animation Sample 패키지를 다운로드하세요.

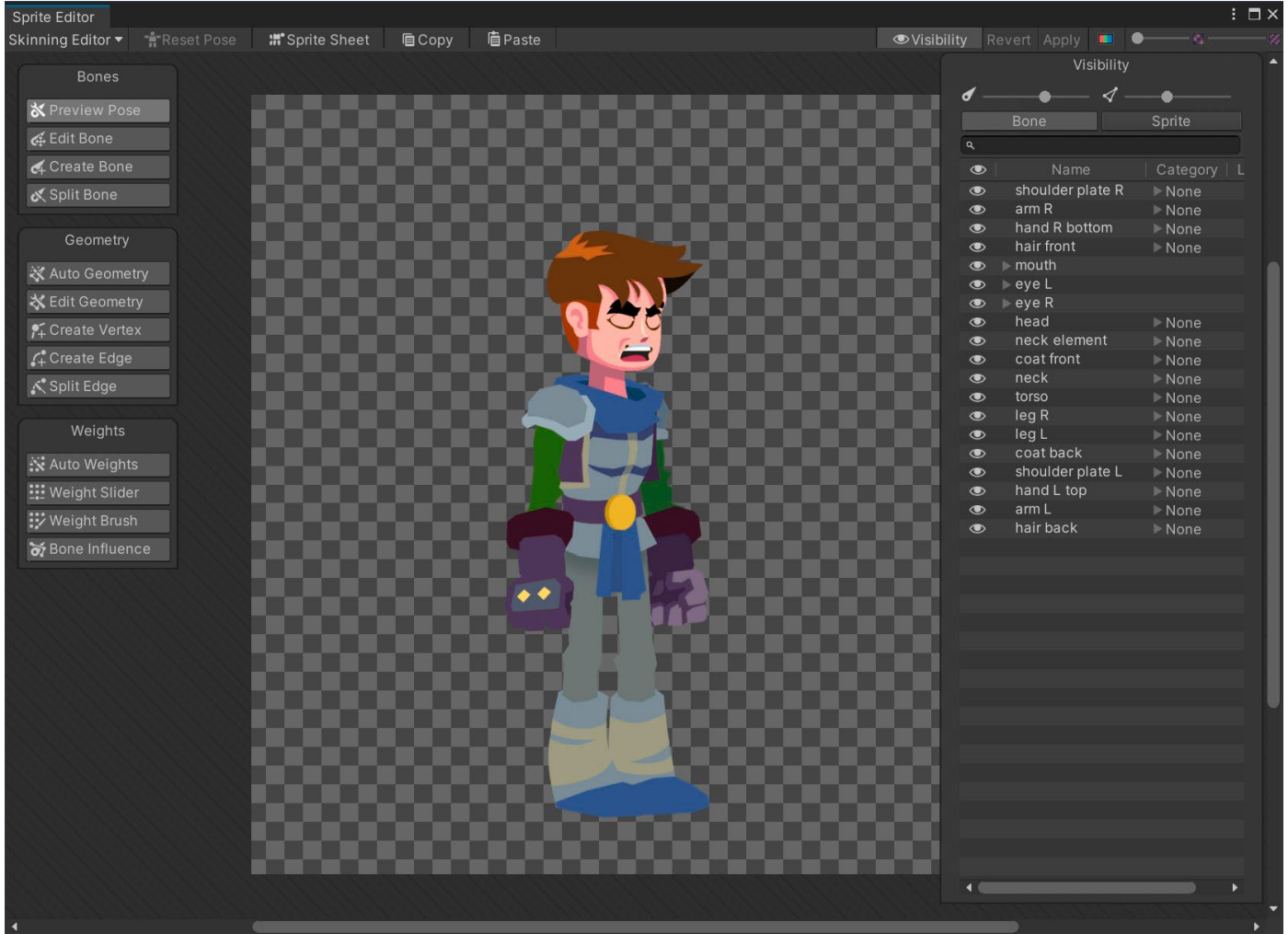
스프라이트 에디터에서의 리깅

캐릭터 리깅을 시작하려면 인스펙터에서 Sprite Editor 버튼을 클릭하여 스프라이트 임포트 설정에 액세스합니다.



Sprite Editor 창에서 툴을 선택하는 방법

Sprite Editor 창의 왼쪽 상단에 있는 드롭다운 메뉴에서 **Skining Editor**를 선택합니다.



캐릭터 애니메이션화 옵션이 포함된 Skining Editor

이 창에서 처리할 수 있는 작업은 다음과 같습니다.

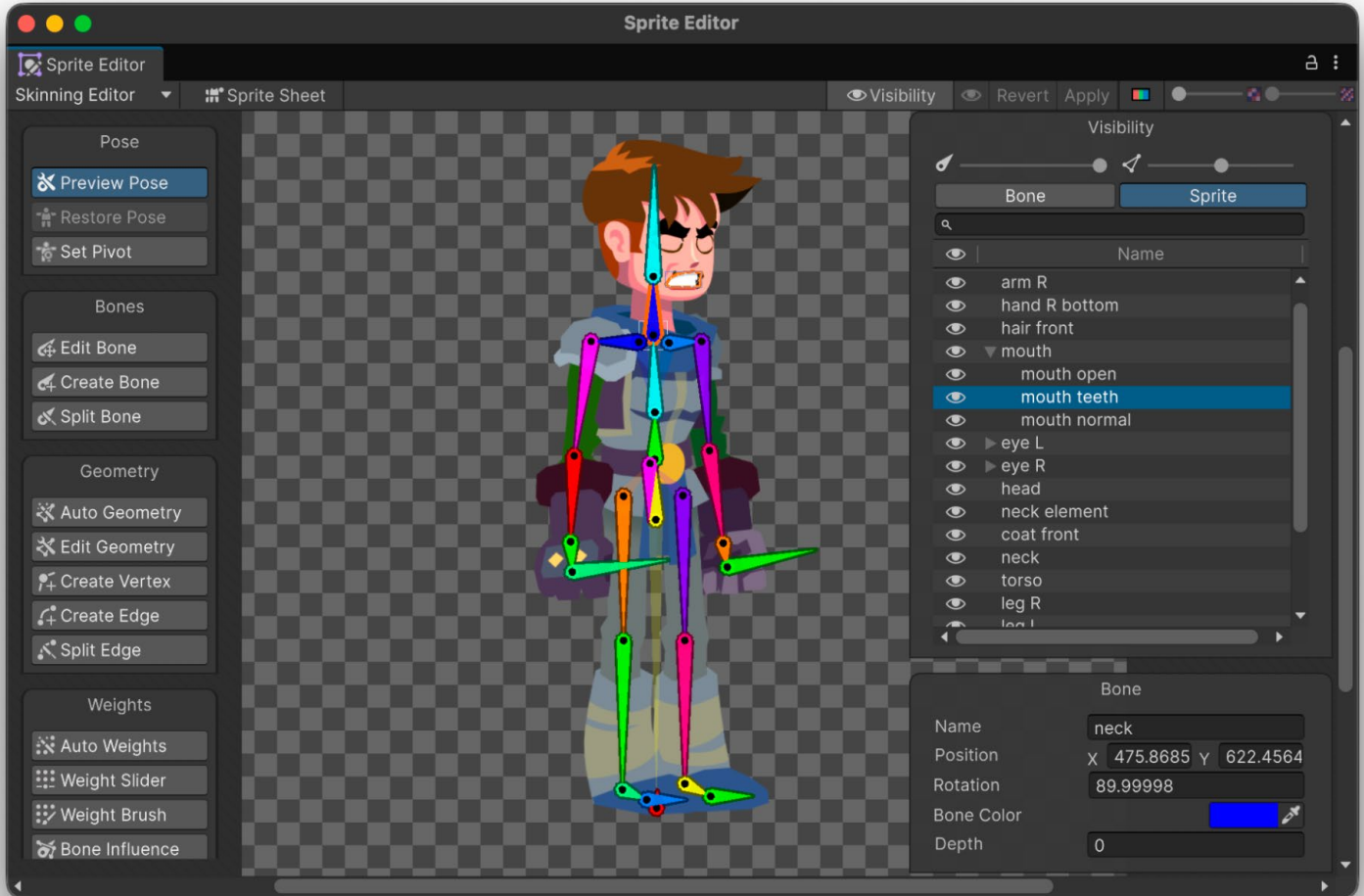
- 뼈대 생성 및 편집
- 스프라이트 지오메트리 생성 및 편집
- 뼈대가 스프라이트에 미치는 영향 편집
- 스킨과 스프라이트 스왑에 사용할 카테고리 및 레이블 생성

골격 생성

캐릭터의 골격을 먼저 만들어야 합니다. **Create Bone** 버튼을 선택하고 메인 창 영역을 클릭합니다. 첫 번째 클릭으로 뼈대의 중심점이 생성되며 두 번째 클릭으로 뼈대의 끝 위치가 표시됩니다. 이 톨은 뼈대를 서로 연결하고 중첩시킵니다.

뼈대 배치를 변경하려면 오른쪽 클릭 후 새 뼈대를 배치할 위치를 클릭합니다. 기존 뼈대를 앞으로 생성할 뼈대의 부모로 지정하려면 기존 뼈대를 클릭해서 선택한 상태로 새 뼈대를 생성합니다.

Edit Bone 버튼을 사용하여 뼈대를 조정합니다. **Split Bone** 버튼을 사용하면 뼈대를 둘로 나눌 수 있습니다. 팔다리를 만들 때 유용한 옵션입니다. 다리 뼈대를 하나 만들고 무릎이 될 곳을 클릭하면 뼈대가 허벅지와 종아리로 나뉩니다.



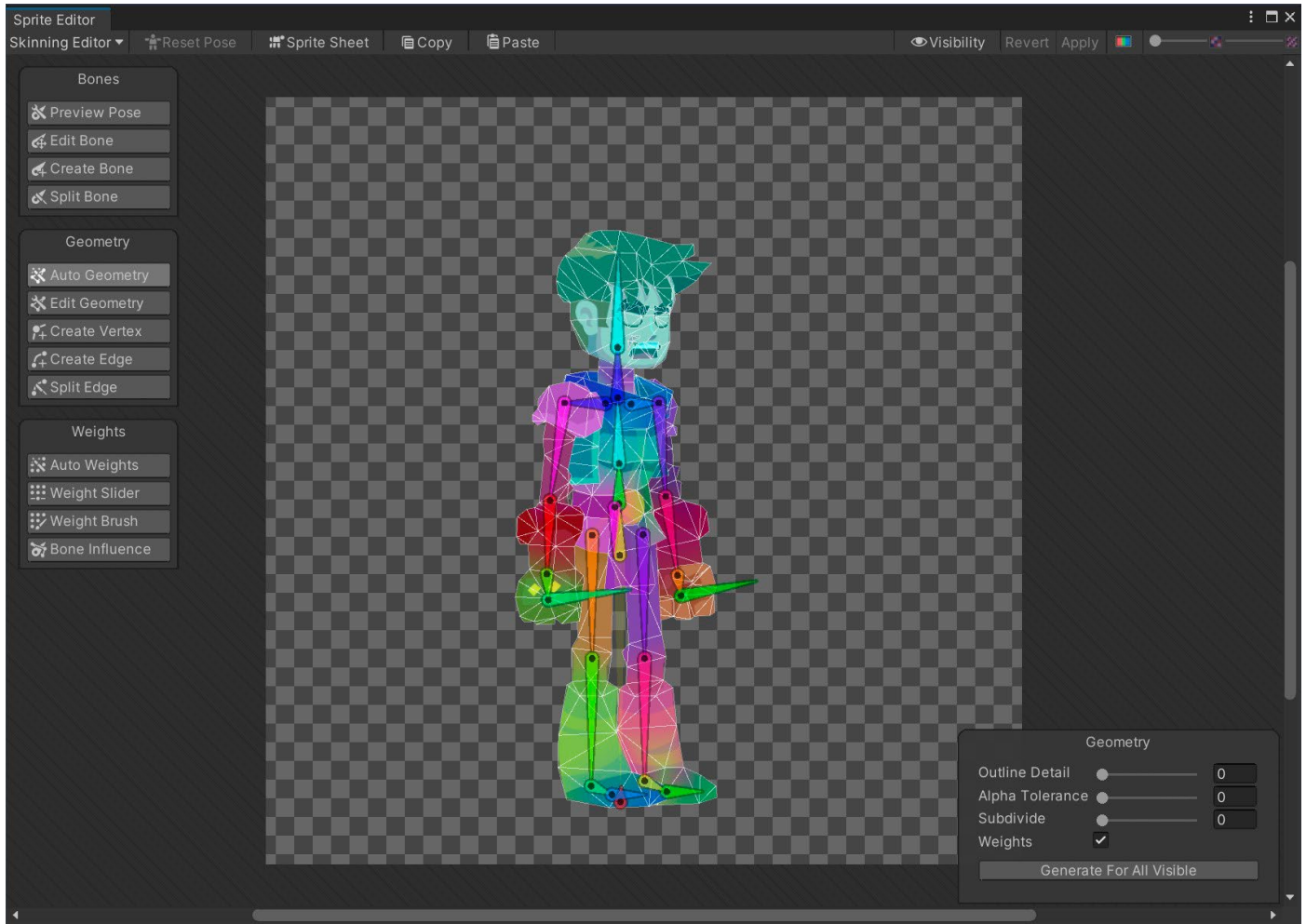
부모 및 자식 뼈대를 구성하기 위해 열려 있는 Bone 탭

뼈대 목록 뷰에서 뼈대의 부모와 이름을 다시 지정할 수도 있습니다. 이 뷰를 열려면 상단 바 오른쪽의 Visibility 버튼을 클릭한 다음 Bone 탭을 선택합니다. 뼈대의 부모를 변경하려면 목록 뷰에서 뼈대를 드래그합니다. 활성화된 뼈대의 이름을 클릭하여 다른 이름으로 변경할 수 있습니다. 뼈대에 이름을 부여하면 필요할 때 더 쉽게 찾을 수 있습니다. 계층 구조가 올바른지 확인하려면 Preview Pose 버튼을 선택하고 몇 가지 포즈를 테스트합니다. 뼈대의 위치를 재설정하려면 툴바에서 Reset Pose 버튼을 누릅니다.

스프라이트 지오메트리

뼈대에 스프라이트를 할당하려면 지오메트리가 생성된 상태여야 합니다. 먼저 Auto Geometry 버튼을 누릅니다. 작은 팝업 창이 열리면 지오메트리 생성 방식을 정의할 수 있습니다.

모든 슬라이더를 0으로 설정하여 지오메트리를 최대한 간단히 유지하는 것이 좋습니다. Weights 옵션을 활성화하여 뼈대를 스프라이트에 자동으로 바인딩합니다. Generate For All Visible 버튼을 클릭하면 모든 스프라이트에 대한 뼈대의 가중치가 생성 및 설정됩니다. 이를 개별 스프라이트에 적용하려면 스프라이트를 더블 클릭합니다. 특정 스프라이트의 지오메트리를 조정하는 데 유용한 방법입니다.



스프라이트 지오메트리 자동 생성

이렇게 지오메트리를 생성하는 수준을 넘어, 다양한 버텍스와 지오메트리 굴절 방식까지 전부 제어하려면 메시를 수동으로 편집해야 합니다. 다음 [툴](#)을 사용해 보세요.

- **Create Vertex:** 새로운 버텍스(또는 점)를 생성하거나 해당 점을 이동 또는 삭제합니다.
- **Create Edge:** 기존의 두 점 사이에 모서리를 만들거나 클릭으로 새 점을 생성하여 모서리를 만듭니다. 현재 선택한 점을 오른쪽 클릭하여 선택 해제하고 다른 버텍스 사이에 모서리를 만들 수 있습니다.



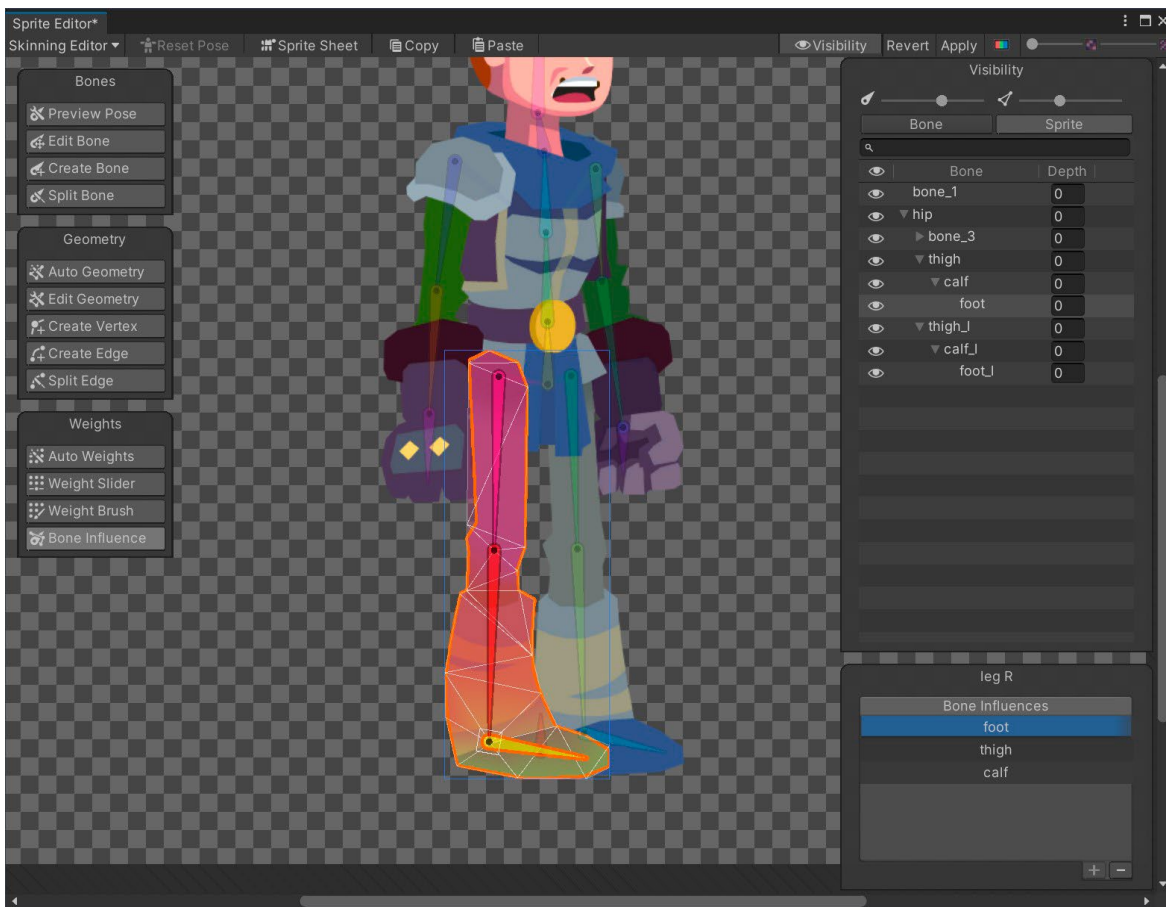
- **Edit Geometry:** 버텍스를 옮깁니다.
- **Split Edge:** 가운데에 버텍스를 생성하여 모서리를 분할합니다.

가급적 적은 수의 버텍스를 사용하세요. 뼈대의 회전에 따라 모든 버텍스 위치를 계산해야 하므로 버텍스 수가 적으면 리소스를 절약할 수 있습니다. 점의 개수가 적을수록 가중치 설정이 더 수월하므로 버텍스 수가 적으면 메시의 굴절도 더 개선됩니다. 또한 게임의 버텍스 수를 타겟 플랫폼마다 알맞게 조정하는 수단이기도 하니, 지오메트리는 항상 최적화하는 것이 좋습니다.

가중치

모든 스프라이트의 지오메트리를 잘 설정했다면 이제 **가중치를 설정**할 차례입니다. 가중치는 뼈대가 각 버텍스에 미치는 영향을 0에서 1로 정의합니다. 0은 뼈대가 버텍스에 미치는 영향이 없다는 것을, 1은 버텍스가 뼈대에 붙은 듯이 움직이는 것을 의미합니다. 적절한 가중치가 설정되면 메시의 굴절을 자연스럽게 표현할 수 있습니다. 가중치를 잘못 설정하면 게임의 몰입감을 깨뜨리고 스프라이트를 왜곡시킬 수 있습니다.

가중치 설정을 시작하려면 특정 스프라이트에 영향을 미칠 뼈대를 정의해야 합니다. Bone Influence 버튼을 클릭하고 스프라이트를 선택합니다.



뼈대가 스프라이트에 미치는 영향 설정



작은 팝업 창이 표시되면 선택한 스프라이트에 영향을 미치는 뼈대를 설정할 수 있습니다. 스프라이트에 영향을 주기 위해 필요한 뼈대만 설정하고 나머지는 제거합니다.

여기서 **Weight Brush** 및 **Weight Slider**를 사용하여 가중치를 설정합니다. Weight Brush를 사용하면 선택한 뼈대의 영향을 마우스로 칠하여 버텍스에 빠르게 추가할 수 있습니다. Weight Slider는 하나 이상의 점을 선택한 후 슬라이더로 각 뼈대의 영향을 정확히 결정할 수 있어 더욱 정밀한 방법입니다. 브러시는 빠른 가중치 설정에 유용하며 슬라이더는 팔꿈치와 무릎처럼 뼈대가 구부러지는 영역을 조정하는 데 유용합니다.



팔다리 굴절을 가장 잘 표현할 수 있도록 버텍스를 배치하는 방법

팁: 팔꿈치와 무릎을 작업할 때는 두 뼈가 만나 구부러지는 지점의 가운데를 지나는 선에 안쪽 버텍스와 바깥쪽 버텍스를 맞추면 가장 보기 좋은 결과를 얻을 수 있습니다. 이 선은 45도 각도로 뼈대를 지나가야 하며, 이때 버텍스는 위쪽 뼈대의 영향만 받아야 합니다. 그러나 캐릭터가 가지는 특성은 서로 다르므로 자유롭게 가중치를 조작하고 가장 자연스러운 결과물을 만들 수 있도록 커스터마이징하세요.

더 빠르게 작업하려면 [드래곤 크래쉬](#) 프로젝트를 열고 프리팹 및 프리팹 배리언트 섹션에서 다음 캐릭터 프리팹을 찾아 보세요.

- Prefab_Character_Base
 - PV_Character_Witch(자동 리바인딩에 다른 스프라이트 라이브러리 사용)
 - PV_Character_Knight
 - PV_Character_Wolfman
 - PV_Character_Skeleton

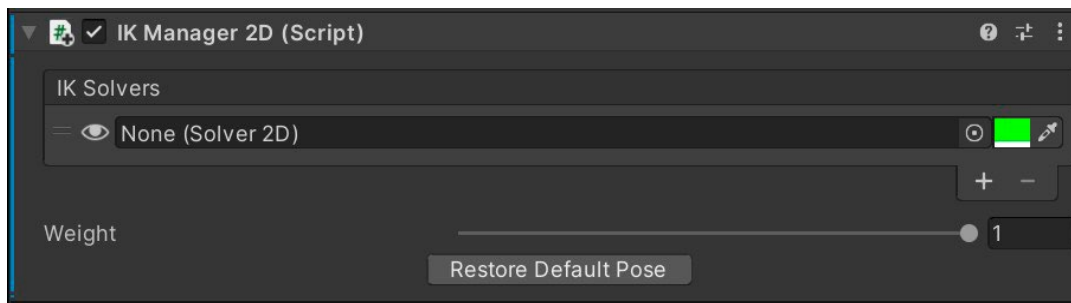
이 프리팹은 구조가 동일한 캐릭터 간에 공유할 수 있는 애니메이션을 보여 주는 훌륭한 예입니다.

2D IK(역운동학)

평소에는 잘 인식하지 못하겠지만 사실 인체의 움직임은 매우 복잡합니다. 테이블에 놓인 유리컵을 집으려면 유리가 차지하는 공간의 지점으로 손이 이동해야 합니다. 이러한 계산이 뇌에서 무의식적으로 처리되므로 사람은 위팔과 아래팔을 돌린다는 생각조차 하지 않고 이러한 동작을 수행하게 됩니다.

그런데 게임에서 동일한 움직임을 애니메이션화하려면 위팔과 아래팔의 회전을 동시에 애니메이션화해야 합니다. 손의 움직임을 그럴듯하게 구현하면서 두 회전을 일치시키는 것은 까다로운 작업입니다. 2D Animation 패키지에 포함된 **2D IK(역운동학)** 툴은 회전을 계산하고 연결된 뼈대를 목표한 위치로 이동할 수 있도록 지원합니다.

우선 계층 구조의 최상단에 있는 오브젝트에 IK Manager 2D를 추가해야 합니다. 이 컴포넌트는 캐릭터의 모든 **IK 솔버**를 관리합니다.

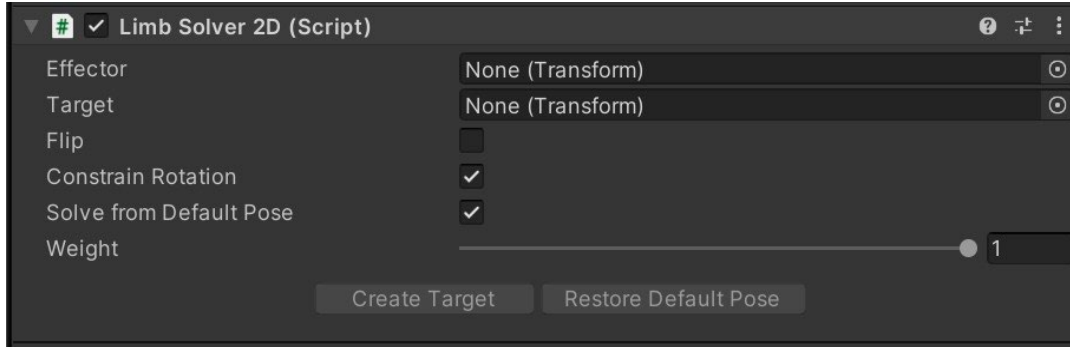


IK Manager 2D 컴포넌트

+ 버튼을 클릭하면 새로운 솔버가 추가됩니다. 솔버는 타겟 트랜스폼을 일치시킬 수 있는 뼈대의 회전을 계산합니다. 여기에 사용할 수 있는 세 가지 솔버는 다음과 같습니다.

- **Limb:** 다리와 팔에 사용되는 표준 솔버로 최대 두 개의 뼈대와 이펙터를 계산할 수 있습니다.
- **Chain (CCD) – Cyclic Coordinate Descent:** 알고리즘 실행 횟수가 늘어날수록 점점 더 정확해져 긴 뼈대 체인에 적합한 솔버입니다.
- **Chain (FABRIK) – Forward And Backward Reaching Inverse Kinematics:** Chain (CCD)와 마찬가지로, 이 솔루션은 알고리즘이 실행될수록 더 정확해집니다. 뼈대가 실시간으로 여러 위치에 영향을 미치는 경우 빠르게 적응합니다.

Limb 솔버는 가장 빠르며 두 개의 뼈로 이루어지는 팔다리에 최적화되어 있으므로 휴머노이드 캐릭터에 가장 적합합니다. 우선 Limb 솔버를 IK Manager 2D 목록에 추가하면 Limb Solver 2D 컴포넌트가 포함된 새 게임 오브젝트가 생성됩니다. 이 게임 오브젝트의 이름을 'Leg R LimbSolver2D' 또는 'Arm L LimbSolver2D'처럼 구체적으로 변경합니다.



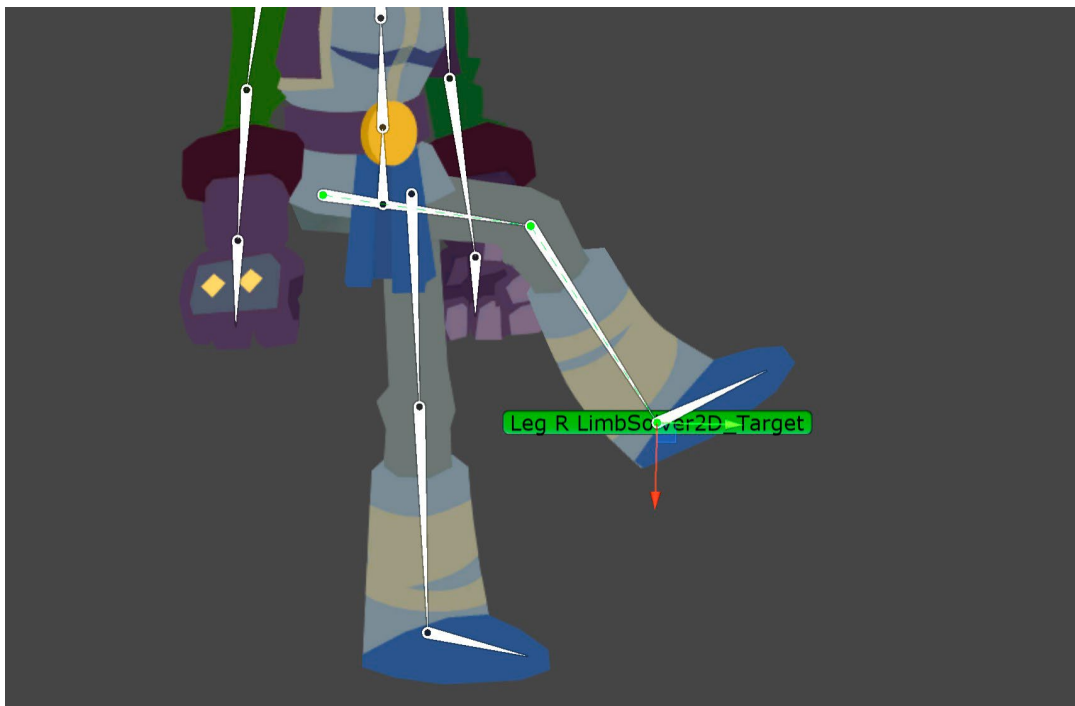
Limb Solver 2D 컴포넌트

솔버가 작동하려면 두 가지 게임 오브젝트 트랜스폼, 즉 **Effector** 및 **Target**이 필요합니다. Effector는 손가락 끝에 있는 뼈처럼 체인의 마지막 뼈대 내부에 배치됩니다. 그리고 그 위치부터 Target의 위치까지 도달하려고 시도합니다.

먼저 Effector를 생성합니다. 다리에 필요한 IK를 생성하는 경우 두 가지 뼈대, 즉 허벅지와 종아리(또는 정강이)를 고려해야 합니다. 종아리 뼈대를 선택하고 새로운 빈 게임 오브젝트를 생성하여 Effector를 종아리 뼈대의 자식으로 배치합니다. 게임 오브젝트의 이름을 Leg Effector로 변경하고 Effector를 뼈대 끝으로 옮깁니다.

새롭게 생성된 Effector를 Limb Solver 2D의 Effector 필드에 추가하고 Create Target 버튼을 누릅니다. 타겟 게임 오브젝트가 Limb Solver 오브젝트의 내부에 생성됩니다.

이제 타겟 오브젝트가 움직이면 다리가 따라갑니다.



2D IK 활용 사례: 타겟 게임 오브젝트를 움직이면 허벅지와 종아리의 회전이 타겟 위치에 맞게 자동으로 계산됩니다.



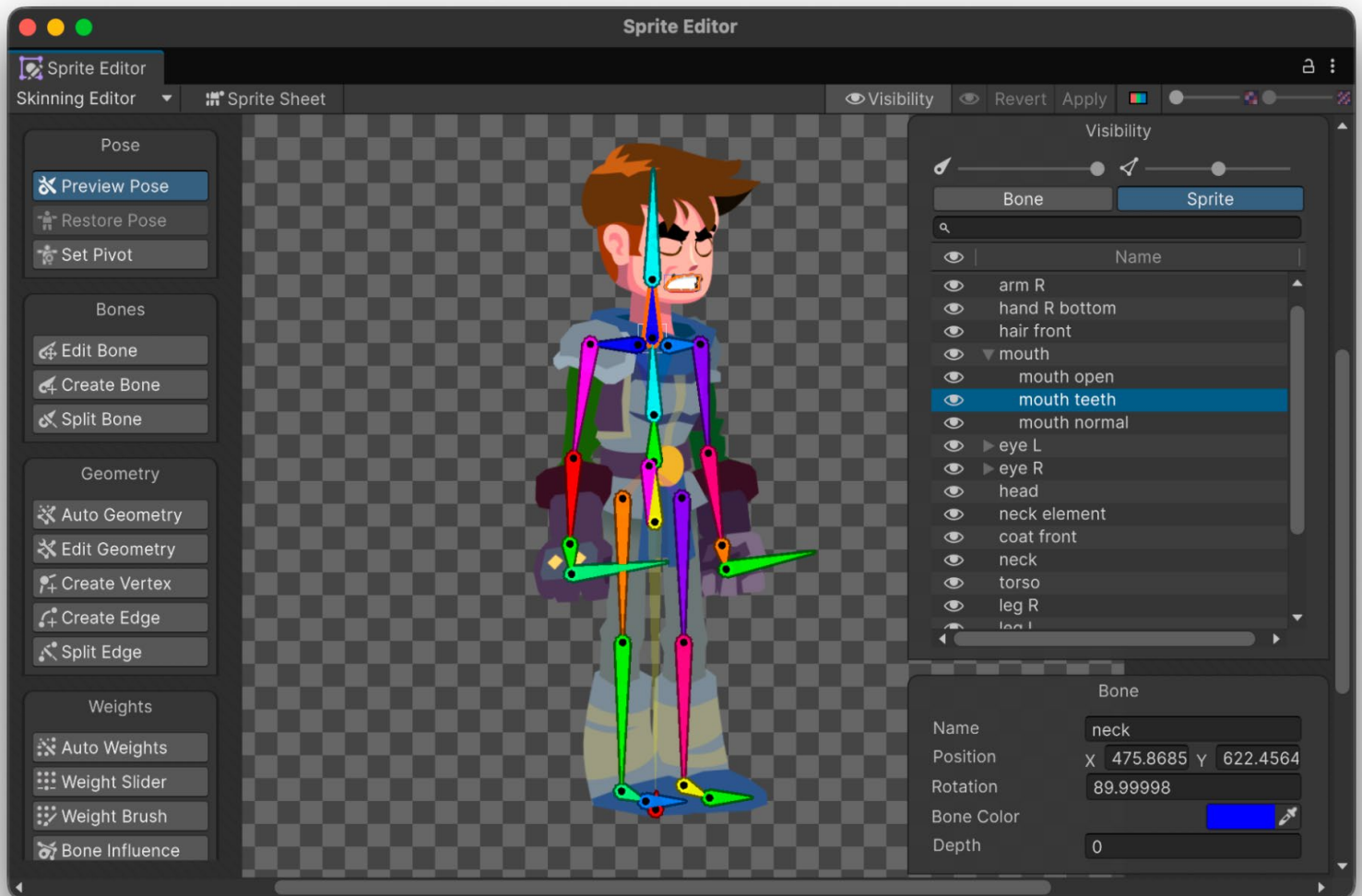
다른 팔다리 부위에도 IK를 추가하는 전체 프로세스를 반복합니다. 팔다리를 제외한 뼈대에도 IK를 추가할 수 있습니다. 예를 들어 머리와 목에 IK를 추가하면 캐릭터가 물체를 바라보도록 만들 수 있습니다.

이 내용을 숙지하면 한 단계 수준을 높여 응용해 볼 수 있습니다. IK를 설정하여 캐릭터가 총을 겨누게 하거나 절차적 걷기 애니메이션을 만드는 것 등을 예로 들 수 있습니다.

스프라이트 스왑 및 스킨

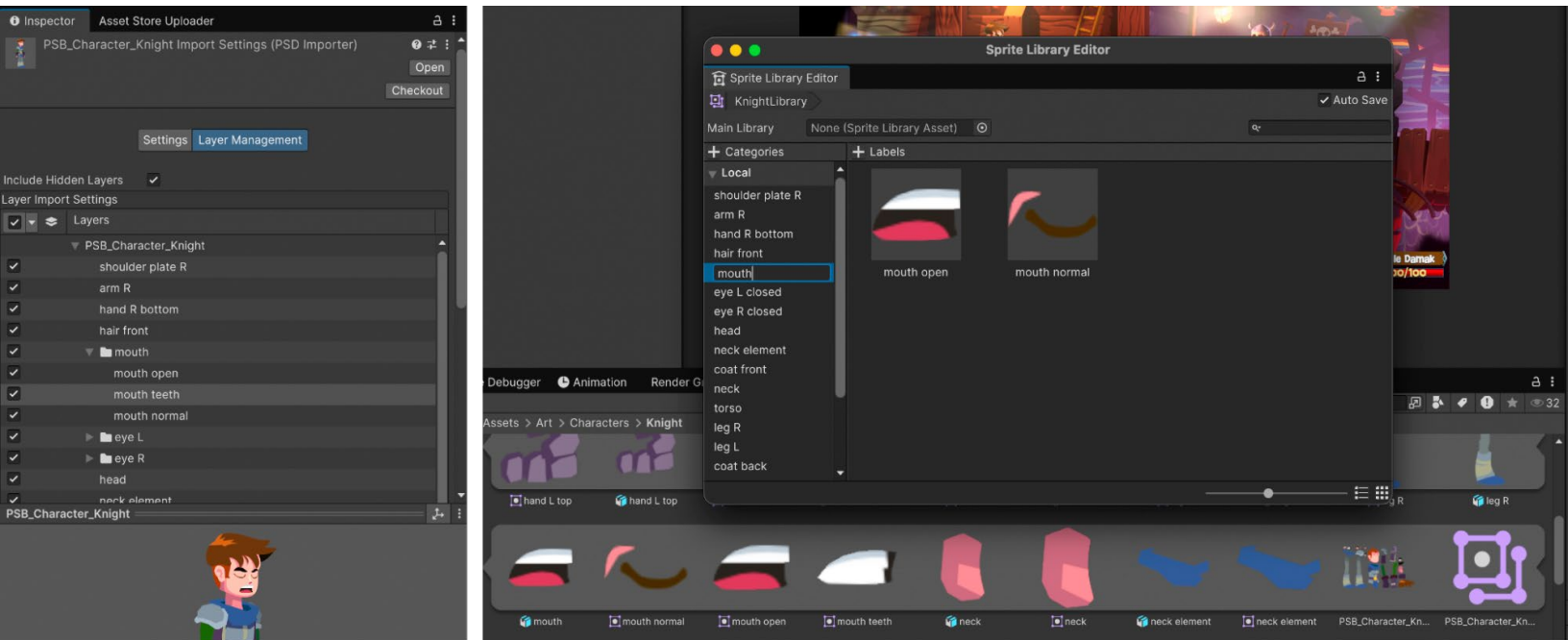
뼈대를 회전하여 애니메이션화할 수 없는 것도 있습니다. 다른 얼굴 표정이나 손 포즈가 필요한 경우도 있습니다. 이럴 때는 [스프라이트 스왑](#)을 사용하여 스프라이트를 다른 것으로 교체할 수 있습니다.

이전에는 스프라이트 스왑이 Skinning Editor에서 각 캐릭터의 스프라이트에 카테고리 and 레이블을 할당하여 작동했습니다. 하지만 이제 Unity 6에서는 카테고리 and 레이블이 스프라이트 라이브러리의 일부입니다. 47페이지의 스프라이트 스왑 섹션에서 자세한 설명을 확인해 보세요.



Skinning Editor에서 Visibility 설정을 활성화하여 캐릭터의 스프라이트와 뼈대를 볼 수 있습니다.

입 스프라이트를 드래곤 크래서의 기사 캐릭터에서 살펴보겠습니다. 배리에이션이 입 카테고리의 자식임을 볼 수 있으며, 이는 원본 PSD 또는 PSB 파일의 그룹화에도 일치합니다.



왼쪽 이미지에서는 PSD 파일의 레이어를 2D PSD Importer 인스펙터에서 볼 수 있습니다. 오른쪽에는 기사 캐릭터를 구성하는 스프라이트를 위한 스프라이트 라이브러리가 생성됩니다.

생성 메뉴를 통해 새 스프라이트 라이브러리를 만들 수 있습니다. 캐릭터와 관련된 이름을 지정하세요. PSD 파일을 Sprite Library 창의 Categories 열로 드래그 앤 드롭하세요.

카테고리는 PSD 파일에서 보이는 스프라이트를 기준으로 자동 생성됩니다.

기사 캐릭터의 예시에서 입 배리에이션의 카테고리 이름이 mouth로 설정되어 있는데, 이 카테고리 이름을 바꿀 수 있습니다.

모든 입 배리에이션이 추가된 것은 아닌데, 이는 PSD 파일에서 숨겨진 레이어였기 때문입니다. 하지만 드래그 앤 드롭으로 mouth 카테고리에 옮겨 놓으면 해당 입 배리에이션을 추가할 수 있습니다. 각 입의 배리에이션에는 레이블이나 이름이 있습니다.

교체될 스프라이트가 Sprite Resolver에 추가되었으며 이제 스프라이트를 클릭해 스프라이트 스위칭을 구현할 수 있습니다. 예를 들어 애니메이션화를 진행하는 동안 Sprite Resolver를 사용하여 스프라이트를 편리하게 변경할 수 있습니다.



드래곤 크래서의 기사 캐릭터에서 입 스프라이트 교체

이제 캐릭터에서 입을 선택하세요. Sprite Resolver에서 카테고리가 선택되지 않은 경우 하나를 선택하면 쉽게 교체할 수 있는 배리에이션이 표시됩니다. 스프라이트 스왑은 Overlay 메뉴에서도 사용할 수 있습니다.

스프라이트 스왑을 사용하여 손쉽게 얼굴 표정, 눈, 입을 애니메이션화하고, 립싱크 애니메이션을 만들고, 손짓을 변경하는 등의 작업이 가능합니다. 스프라이트 스왑을 통해 모자와 갑옷 등의 캐릭터 장비를 교체할 수도 있습니다.

Sprite Swap 오버레이



2D 애니메이션 캐릭터는 카테고리 및 레이블이 있는 스프라이트 라이브러리를 통해 인스펙터 또는 씬 오버레이에서 스프라이트를 쉽게 교체할 수 있습니다.

이제 Unity 6의 씬 뷰에서 새로운 **Sprite Swap 오버레이**를 사용할 수 있습니다. 이 오버레이는 선택한 스프라이트로 교체할 수 있는 모든 관련 레이블을 표시합니다.

또한 계층 구조의 모든 교체 가능한 부분에 맞게 확대/축소됩니다. 멀티 파트 캐릭터의 특정 부분이 선택되면, 교체 가능한 스프라이트가 있는 모든 자식 오브젝트가 오버레이에 함께 표시됩니다. Animation 창과 함께 사용하여 스프라이트 스왑 애니메이션 워크플로를 더 효율적으로 만드세요.

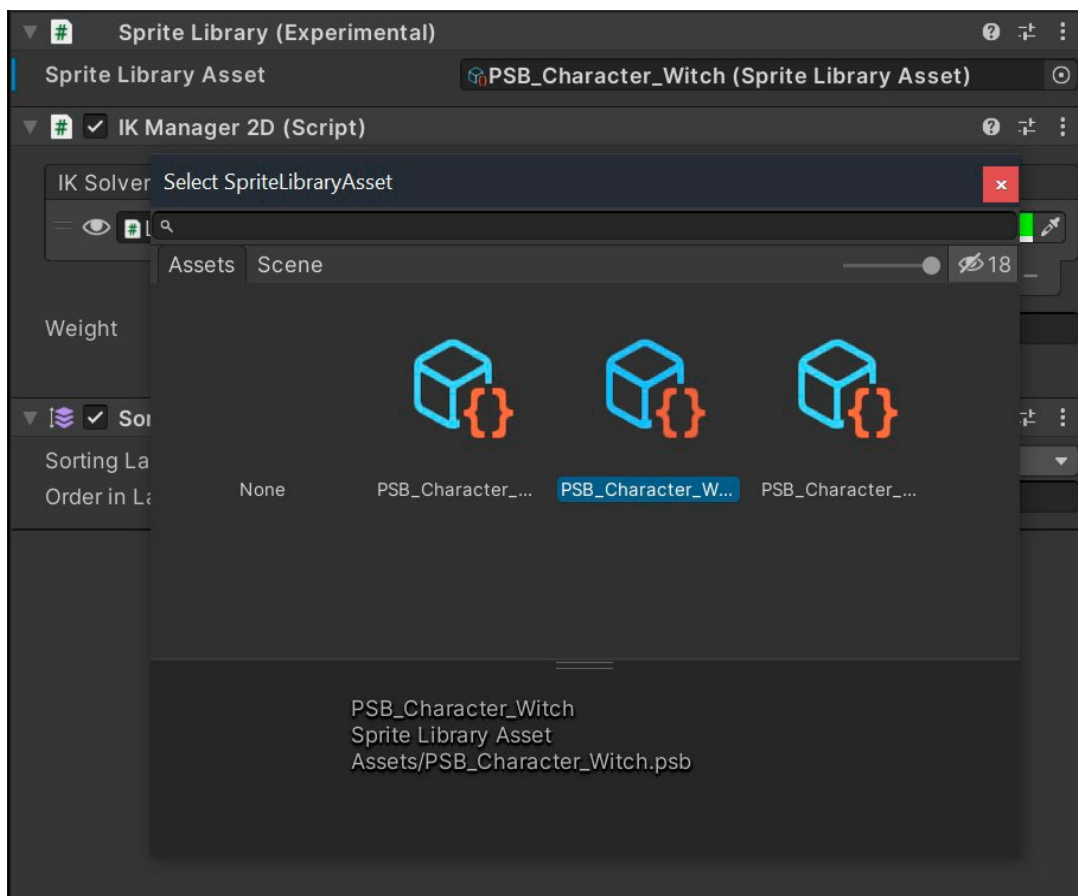
스킨

스프라이트 라이브러리 에셋을 사용하여 캐릭터용 스킨을 만들 수 있습니다. 스킨을 활용하면 애니메이션은 유지한 채 캐릭터의 외형을 변경할 수 있으므로 시간이 대폭 단축됩니다. 하나의 기본 캐릭터에 모든 스크립트가 포함될 수 있으며 변경 사항은 다른 캐릭터에도 적용됩니다.

드래곤 크래셔 및 해피 하비스트 샘플에서 캐릭터에 스킨 지원을 설정한 워크플로는 다음과 같습니다.

1. 기본 캐릭터를 완성합니다. 리깅하고 IK와 스프라이트 라이브러리를 설정합니다. 해당 캐릭터를 프리팹으로 만듭니다. 언제든지 캐릭터를 애니메이션화할 수 있습니다.
2. 기본 캐릭터의 Skinning Editor로 이동하여 툴바에서 **Copy** 버튼을 클릭합니다. 그러면 가중치를 포함하여 뼈대와 메시가 복사됩니다.

3. 이 캐릭터의 프리팹 배리언트를 생성하면 새로운 모습을 갖춘 새 캐릭터가 만들어집니다.
4. 새 캐릭터의 PSD 또는 PSB 파일을 임포트하고 해당 Skinning Editor를 엽니다.
5. 툴바에서 **Paste** 버튼을 클릭하여 2단계에서 복사한 골격을 붙여 넣습니다. 팝업 창에서 **Bones** 및 **Mesh** 옵션을 선택하고 Paste 버튼을 클릭하여 확인합니다.
6. 새 스프라이트에 맞게 지오메트리를 수정합니다. 가중치를 다시 확인합니다.
7. 3단계에서 생성한 Prefab Variant로 이동합니다. Sprite Library 컴포넌트로 이동하여 새 캐릭터의 Sprite Library Asset을 교체합니다.
8. 새 캐릭터를 생성하려면 2~8단계를 따릅니다.



Sprite Library 컴포넌트에서 Sprite Library Asset을 선택합니다.

이 워크플로를 따르면 기본 캐릭터 프리팹 하나가 생깁니다. 다른 캐릭터는 이 프리팹의 배리언트가 되므로 기본 프리팹과 동일한 컴포넌트를 갖습니다. IK 또는 정렬 그룹을 추가하거나 변경하는 등 기본 캐릭터를 변경하면 다른 캐릭터에도 변경한 내용이 상속됩니다.

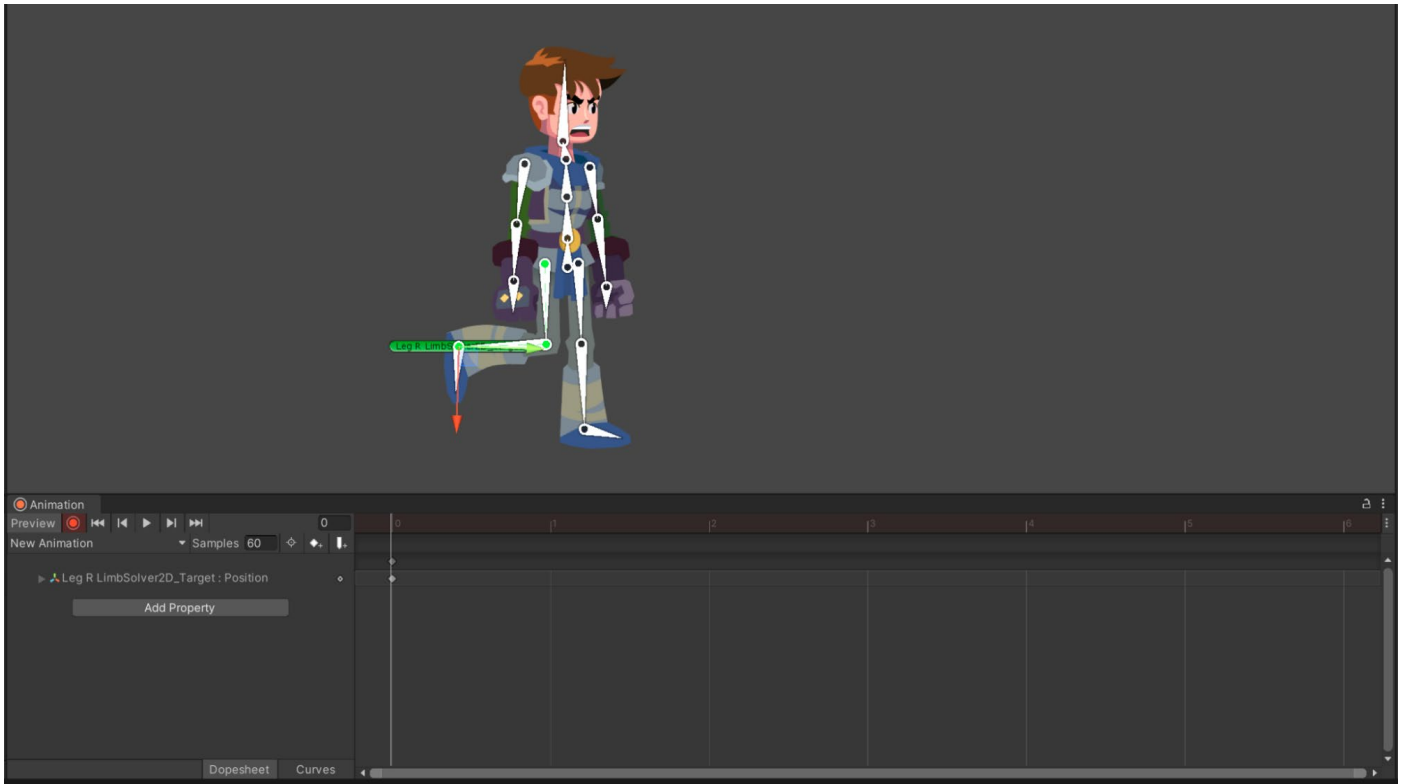


애니메이션 기초

캐릭터를 자연스럽게 구현하려면 애니메이션이 매우 중요합니다.

근사한 애니메이션을 제작하려면 애니메이션 원칙과 그 원칙을 적용할 툴을 학습해야 합니다. 기초적인 애니메이션 원칙을 배우지 않고 넘어가는 경우가 많지만 약간의 시간을 들여 캐릭터 움직임을 다듬으면 게임의 품질을 크게 높일 수 있습니다.

첫 번째 애니메이션을 만들기 위해 **Window > Animation > Animation**으로 이동하여 **Animation 창**을 엽니다.



애니메이션화할 수 있는 캐릭터가 표시된 Animation 창

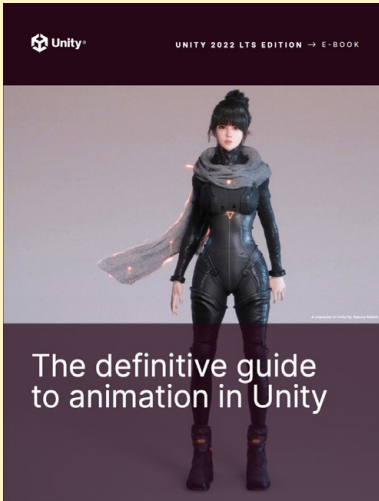
캐릭터를 선택합니다. **Create** 버튼을 클릭하면 첫 번째 **애니메이션 클립**이 생성되고 클립에 이름을 부여하여 작업을 시작할 수 있습니다. 애니메이션 클립은 오브젝트의 위치, 회전, 스케일 및 기타 프로퍼티가 시간 경과에 따라 변화하는 과정을 선형적으로 녹화한 것입니다. 이때 **Animator 컨트롤러**도 생성됩니다. 모든 애니메이션 클립을 관리하고, 어떤 클립을 재생하고 언제 애니메이션이 변경되거나 함께 블렌딩되는지 기록하는 기능입니다.

빨간 **레코드** 버튼을 클릭하여 애니메이션 작업을 시작합니다. 이제 캐릭터에 적용되는 모든 변경 사항이 애니메이션 클립에 기록됩니다. 만족스러운 애니메이션 클립을 얻을 때까지 빼대, IK 솔버를 옮기거나 Sprite Resolver로 스프라이트를 교체하고 재생하기를 반복할 있습니다.

애니메이션은 사용할 툴에 대한 기술적 이해가 필요한 아트 영역입니다. 이 전자책 외에도 Unity는 3D 또는 2D 캐릭터를 생성하게 만들 수 있는 애니메이션에만 집중한 심층 가이드를 제공합니다. 아래에서 더 많은 링크를 확인하고 Unity 에셋 스토어에서 **2D Animation Sample 패키지**를 확인하세요.



다음과 같은 리소스를 통해 Unity의 3D 및 2D 애니메이션에 대해 더 자세히 알아보세요.



전자책: [Unity 애니메이션 집중 탐구 가이드](#)



[Unity에서 휴머노이드 애니메이션 사용하기](#)

2D 물리

이 전자책은 주로 Unity를 처음 사용하는 아티스트와 테크 아티스트를 위해 작성되었으며, 2D 툴을 최대한 활용하여 시각적 충실도를 높이하고자 합니다.

그러나 소규모 팀이든 대규모 팀이든 게임플레이 로직의 일부로 2D 물리를 다루게 될 가능성이 높습니다. 그러면 Unity 6.3 기준으로 [2D 물리](#)의 기본을 살펴보겠습니다.

3D 프로젝트를 다루는 경우, 3D 물리와 2D 물리는 구현의 유사성에도 불구하고 완전히 다른 시스템이라는 것을 알아 두어야 합니다. 3D 오브젝트의 물리는 2D 오브젝트와 상호 작용하지 않습니다.

2D 물리를 활용하면 시스템이 물리에 현실적으로든 비현실적으로든 반응하는 생동감 넘치는 월드를 만들 수 있습니다. 캐릭터, 차량, 오브젝트, 체인, 플랫폼 등은 단단한 터레인, 물, 바람의 힘 등으로 이루어진 환경이나 여러분이 만들고 싶은 어떤 요소와도 상호 작용할 수 있습니다.

2D 물리의 모든 가능성을 한눈에 파악할 수 있도록, 샌드박스 2D 물리 샘플을 통해 모든 시스템이 작동하는 모습을 확인해 보세요. Unity에서 물리 시스템을 포함하여 게임 디자인에 대한 일반적인 내용을 더 알아보려면 무료로 게임 디자인 전자책을 받아 보세요. 이 전자책은 이전 Unity 버전을 기반으로 하지만, 대부분의 내용은 Unity 6.3에 계속해서 적용할 수 있습니다.



2D 물리 샘플



전자책: Unity 게임 디자이너
플레이북

콜라이더: 오브젝트의 상호 작용 가능한 영역

오브젝트가 물리 월드에서 상호 작용하려면 정의된 영역이 필요합니다. Unity로 2D 콘텐츠를 개발할 때, 게임 오브젝트에 연결된 2D Collider 컴포넌트를 통해 이 영역을 추가합니다. 셰이프가 단순할수록 성능이 더 좋아집니다. 사용할 수 있는 Collider 컴포넌트는 다음과 같습니다.

Edge Collider 2D	— 스프라이트 또는 다른 셰이프에 맞게 조정할 수 있는 선분으로 구성된 가장자리 콜라이더
Box Collider 2D	— 스프라이트의 로컬 좌표 공간에서 위치, 너비 및 높이가 정의된 사각형
Circle Collider 2D	— 스프라이트의 로컬 좌표 공간에서 위치 및 반지름이 정의된 원형 콜라이더
Capsule Collider 2D	— 둘레가 부드럽고 둥근 모양이며 버텍스 모서리가 없는 캡슐 모양의 콜라이더. 이를 통해 다른 콜라이더를 따라 더 부드럽게 이동할 수 있으며 날카로운 모서리에 걸리는 현상을 방지할 수 있습니다.
Polygon Collider 2D	— 물리를 통해 여러 개의 컨벡스 폴리곤 셰이프를 생성할 영역을 정의하는 오목/볼록 외곽선
Custom Collider 2D	— PhysicsShapeGroup2D API를 통해 PhysicsShape2D 지오메트리를 할당하여 콜라이더 셰이프를 구성
Tilemap Collider 2D	— 같은 게임 오브젝트의 Tilemap 컴포넌트에 대한 타일의 콜라이더 셰이프를 생성. Tilemap 컴포넌트에서 타일을 추가하거나 제거하면 Tilemap Collider 2D가 LateUpdate 중에 콜라이더 셰이프를 업데이트합니다.
Composite Collider 2D	— 고유한 셰이프가 없으며, 대신 사용하도록 설정된 모든 Box Collider 2D, Polygon Collider 2D, Circle Collider 2D 또는 Capsule Collider 2D의 셰이프를 병합

리지드바디와 다른 리지드바디 또는 콜라이더 간의 충돌을 감지하려면 API [OnCollisionEnter2D](#), [OnCollisionStay2D](#) 및 [OnCollisionExit2D](#)를 사용하여 충돌 이벤트가 발생할 때 이를 감지할 수 있습니다.



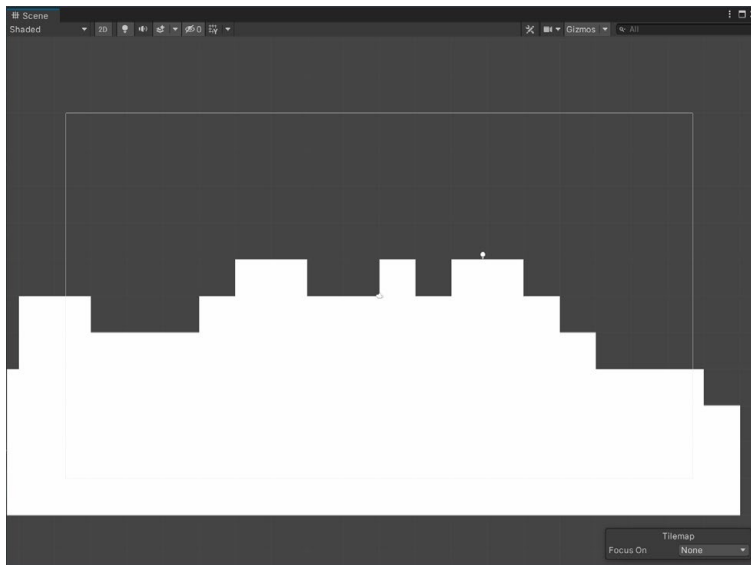
Is Trigger

콜라이더 2D에서 **Is Trigger** 상자를 체크하면, 이 콜라이더가 물리적 이동을 발생시키지 않고 다른 콜라이더를 감지하게 됩니다(오브젝트에 연결된 Rigidbody 2D 컴포넌트를 무시). 스텔스 게임의 센서를 예로 들면, 적들은 특정 영역에서 플레이어의 존재를 감지하기 위해 **IsTrigger**가 있는 콜라이더 2D를 사용할 수 있습니다. 콜라이더가 플레이어의 존재를 감지하면 적들이 플레이어를 쫓기 시작할 수 있습니다.

그 경우, C#에서는 트리거와의 충돌이 발생할 때 [OnTriggerEnter2D](#), [OnTriggerStay2D](#) 또는 [OnTriggerExit2D](#) 이벤트를 사용하여 로직을 작성합니다.

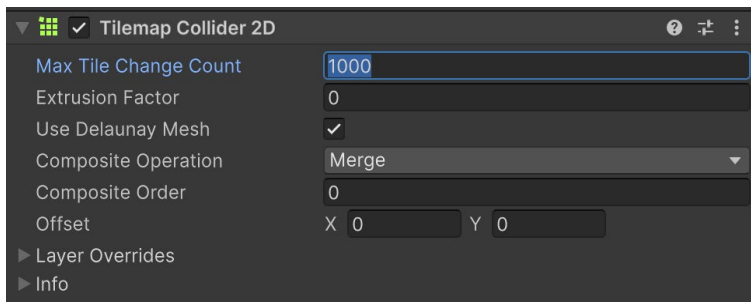
타일맵 콜라이더 2D 만들기

간단한 사각형 타일로 이루어진 타일맵 지면을 만들어서 2D 물리를 빠르게 테스트할 수 있습니다.



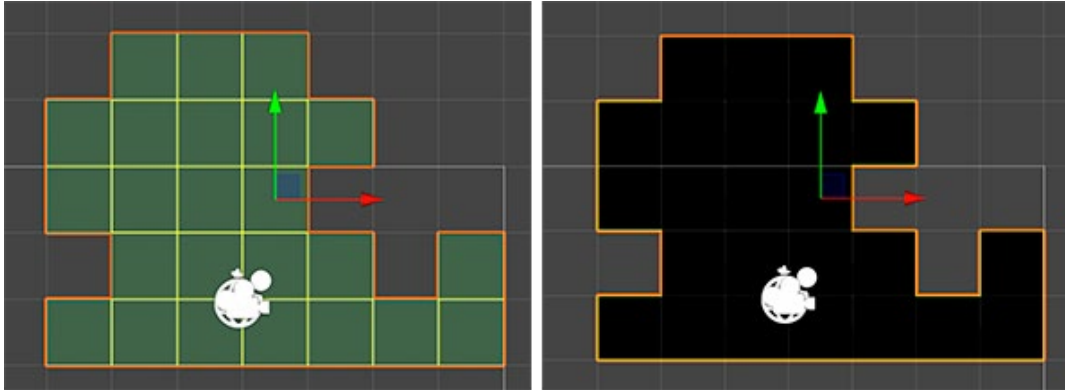
플레이스홀더 타일이 페인팅된 타일맵

다음으로 물리를 적용할 수 있도록 인스펙터 창에서 [Tilemap Collider 2D](#) 컴포넌트를 타일맵에 추가하세요. 그러면 타일 에셋에 설정된 콜라이더 유형에 따라 각 타일에 콜라이더가 추가됩니다.



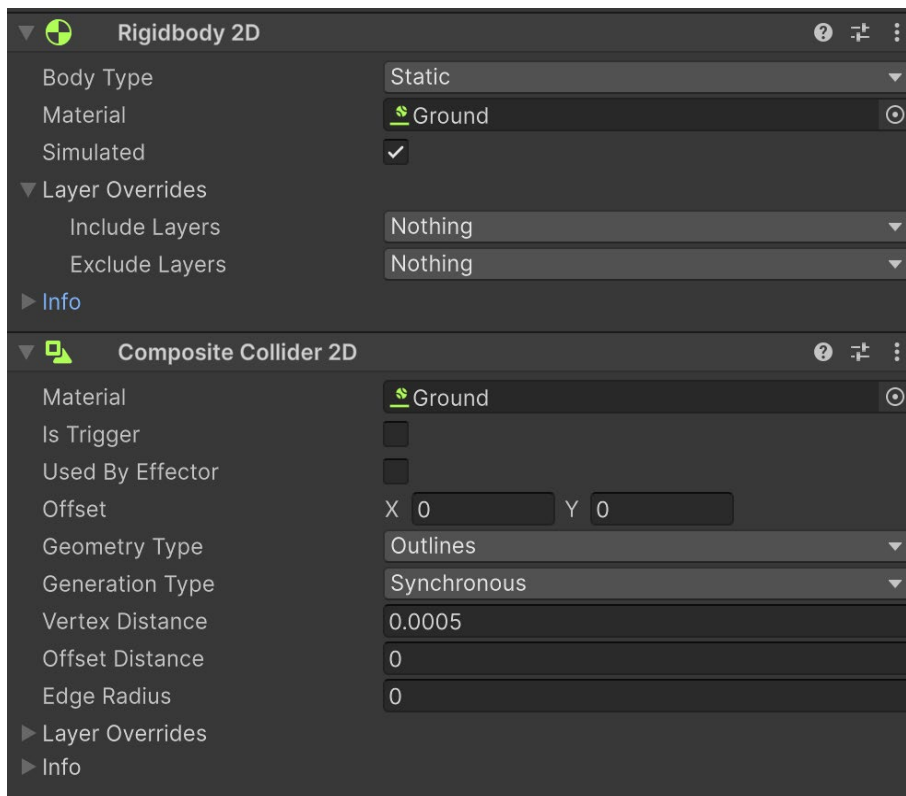
이 예시에서는 **Use Delaunay Mesh** 옵션이 활성화되어 있으며, 이는 콜라이더 메시에서 생성되는 셰이프와 작은 삼각형의 수를 줄여 전체적인 물리 성능을 향상시키는 데 도움을 줍니다.

같은 게임 오브젝트 또는 부모 그리드에 **Composite Collider 2D** 컴포넌트를 추가하여 콜라이더를 하나로 조합하면 지오메트리를 Outline으로 설정했을 때 충돌 동작이 훨씬 부드러워집니다. 타일은 개별 타일이 아닌 연속된 터레인처럼 동작합니다. 이 설정을 사용하면 성능을 약간 개선할 수도 있습니다. 그러나 런타임에 게임에 타일을 추가하거나 제거할 계획이라면 각 타일에 콜라이더를 유지하는 것이 좋습니다.



타일을 그룹화하여 Composite Collider 2D 및 Merge 설정을 통해 단일 표면을 형성할 수 있습니다. 왼쪽 이미지에서는 타일이 병합되지 않았고 오른쪽 이미지에서는 병합되었습니다.

Tilemap Collider 2D 컴포넌트에서 Composite Operation을 Merge로 설정하고, 자동으로 추가된 Rigidbody 2D Type을 Static으로 설정하여 타일맵이 지면 역할을 하고 떨어지지 않게 하세요.

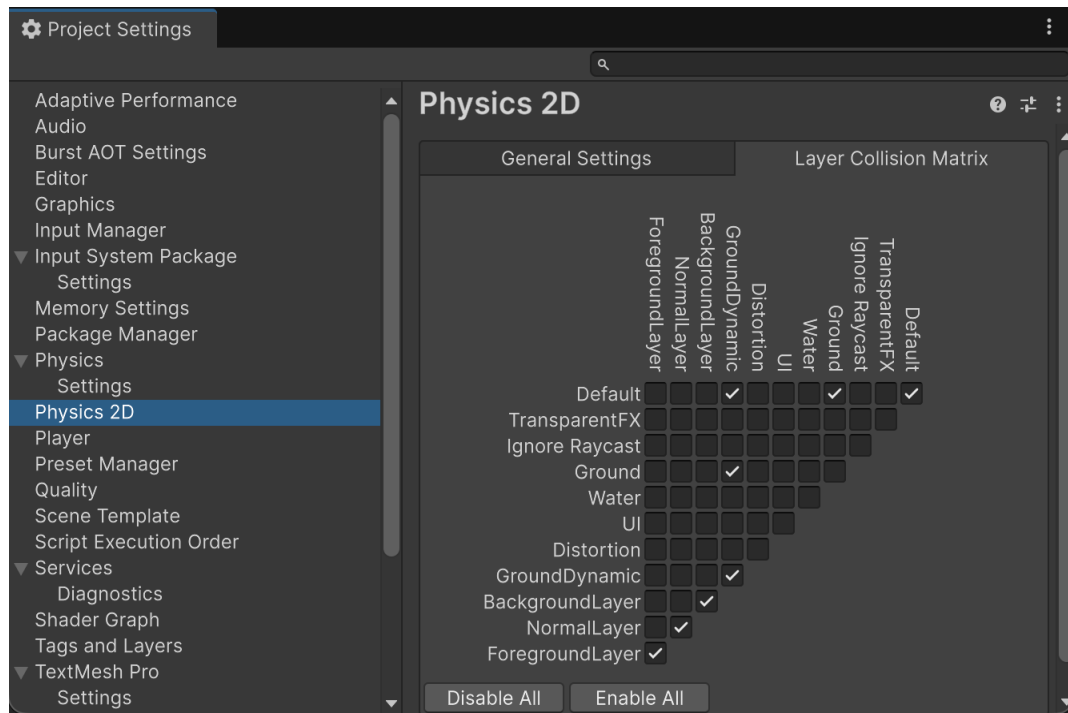


Tilemap 게임 오브젝트의 Collider 2D 컴포넌트

충돌 매트릭스

Physics 2D에서 게임 오브젝트 레이어를 사용하면 오브젝트가 서로 상호 작용할 수 있습니다. 하지만 모든 요소가 서로 상호 작용할 필요는 없습니다. 예를 들면 플랫폼 게임에서는 적이 지면과 충돌하도록 해야 하지만, 다른 적들과 동시에 충돌할 필요는 없습니다. 이 경우 적들이 지면과는 충돌하지만 동시에 적들끼리는 충돌하지 않게 설정할 수 있습니다.

이 설정은 **Project Settings > Physics 2D > Layer Collision Matrix**에서 찾을 수 있습니다.



Unity 6의 2D 콜라이더를 사용하면 인스펙터 창의 Rigidbody 2D 컴포넌트에서 오브젝트별로 특정 물리 레이어를 물리 계산에서 무시하거나 적용하도록 설정할 수 있습니다.

리지드바디: 오브젝트가 물리에 따라 움직이게 만들기

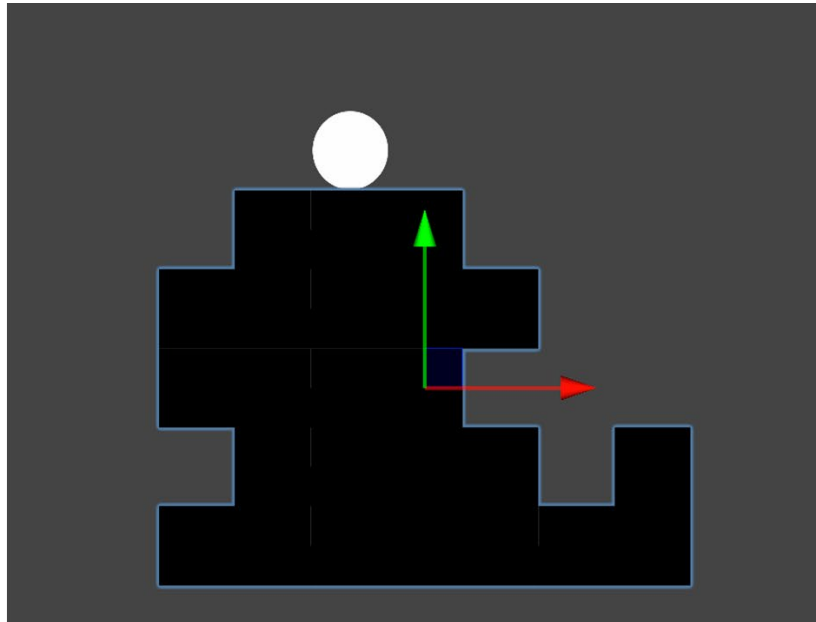
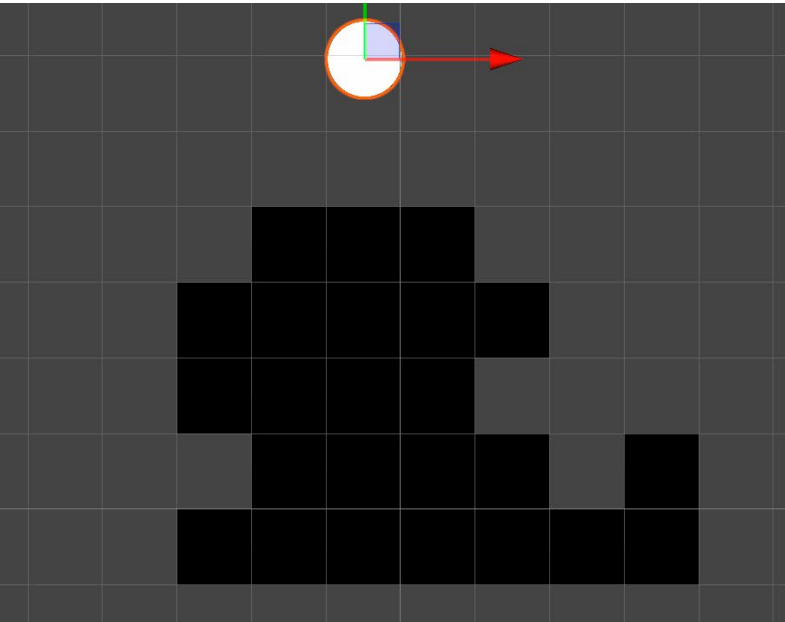
이제 오브젝트에 정의된 콜라이더 2D 영역이 있으므로 하나 이상의 **Rigidbody 2D** 컴포넌트와 물리적으로 반응할 수 있습니다.

리지드바디는 다음과 같이 설정할 수 있습니다.

- **Static:** 움직이지 않지만 다른 오브젝트가 그에 반응합니다. 예를 들어, 지면 오브젝트는 정적 리지드바디가 될 수 있습니다.
- **Kinematic:** 다른 충돌에 영향을 받지 않고 움직입니다. 예를 들어, 움직이는 플랫폼은 그 위에 몇 개의 오브젝트가 있든 상관없이 동일하게 움직입니다.
- **Dynamic:** 질량, 속도, 힘과 같은 물리적 프로퍼티에 따라 움직이며, 접촉하는 다른 리지드바디의 영향을 받습니다.



이전 타일맵 씬을 사용하여 빠르게 테스트해 보겠습니다. **GameObject > 2D Object > Physics > Dynamic Sprite**를 통해 씬에 새 오브젝트를 만듭니다. 이 원형 오브젝트를 타일맵 위에 놓고 플레이 모드를 실행합니다.



정적 타일맵의 표면에서 동적 오브젝트가 실제처럼 튀기는 것을 볼 수 있습니다.

코드에서 리지드바디 2D 옮기기

2D 물리 오브젝트는 씬에 인스턴스화되면 실제 물리 오브젝트처럼 동작하지만, 이동하는 플랫폼을 움직이거나 캐릭터가 다양한 방향으로 이동하거나 점프하도록 만들어야 하는 경우가 많습니다.

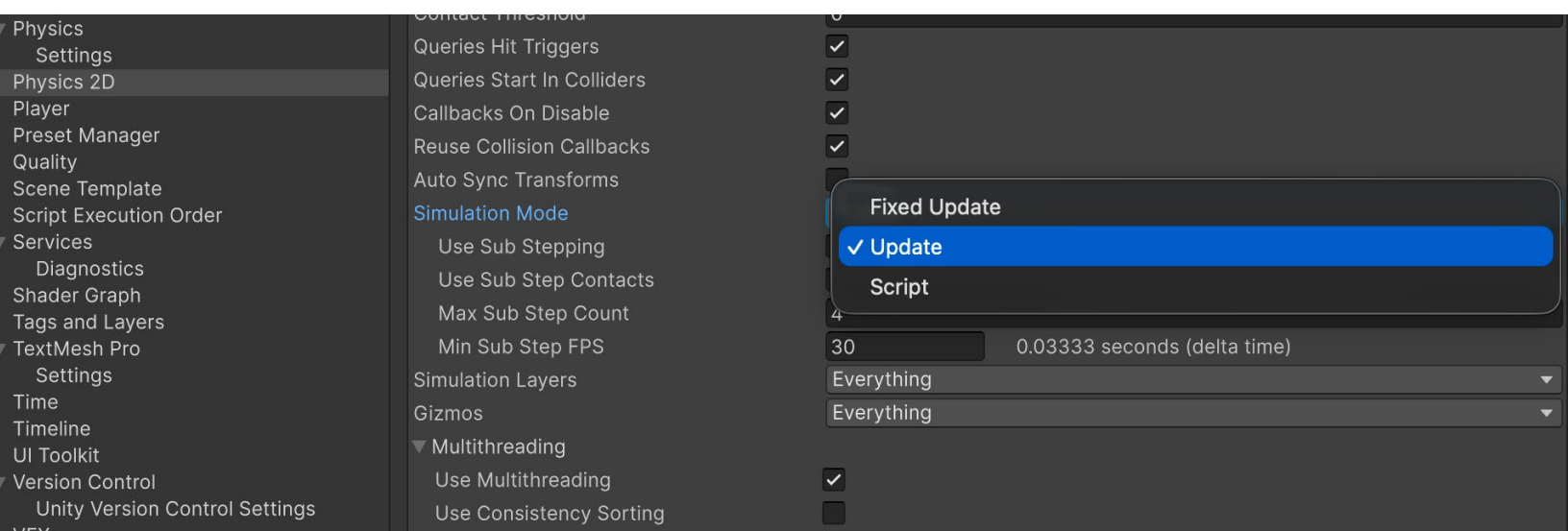
모든 물리 기반 오브젝트는 위치를 변경하려면 [Rigidbody2D API](#)를 사용해야 합니다. 게임 오브젝트 트랜스폼으로 이러한 오브젝트를 옮기지 마세요. 대신, 다음과 같은 경우에 함수를 사용할 수 있습니다.

- [AddForce](#) 또는 [Slide](#) 메서드를 사용하여 캐릭터의 움직임을 제어합니다.
- Impulse 모드로 [AddForce](#) 메서드를 사용하여 오브젝트를 발사하거나 캐릭터를 점프하게 만듭니다.
- 회전하는 플랫폼처럼 키네마틱 리지드바디가 계속 움직이도록 만들려면 [MoveRotation](#) 또는 [MovePosition](#)을 사용합니다.
- [position](#) 프로퍼티를 변경하여 리지드바디를 한 위치에서 다른 위치로 텔레포트합니다.
- 캐릭터의 이동 속도를 평가하려면 [linear velocity 프로퍼티](#)를 사용합니다.

코드에서는 Update 또는 FixedUpdate **사이클**에서 리지드바디 메서드를 사용해야 합니다. 프로젝트 설정에서 변경 가능합니다.

- Update는 매 프레임마다 실행되며 더 부드럽고 정밀한 시뮬레이션을 제공합니다.
- FixedUpdate는 기본적으로 0.2초마다 설정된 간격으로 발생합니다.

두 경우 모두 일정한 움직임을 위해 값에 **Time.deltaTime**을 곱해야 합니다. Time.deltaTime은 한 프레임을 처리하는 데 걸리는 시간(초)입니다. 시뮬레이션에서 일정한 결과를 얻으려면 이 값을 움직임 관련 값에 곱하세요.



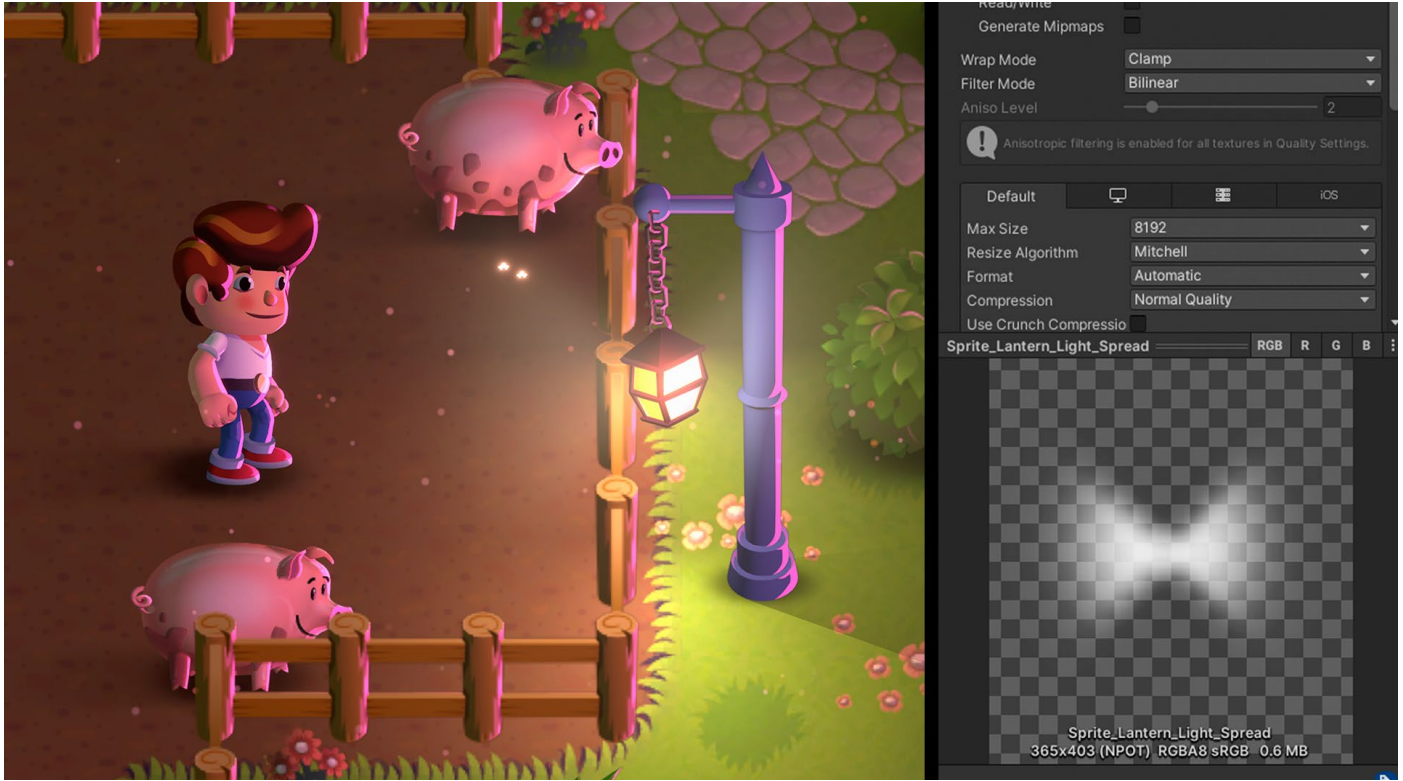
Simulation Mode 설정

2D 광원 및 그림자

2D 게임의 몰입도는 얼마든지 원하는 대로 만들 수 있습니다. 디지털 인형극처럼 조명을 포함해 다양한 요소와 레이어가 사용되며, 오브젝트가 환경이나 광원에 동적으로 반응하도록 만들 수 있습니다.

Unity 2D 광원 시스템은 게임에 풍부하고 다양한 조명 및 시각 효과를 생성하는 여러 기능을 제공하여, 원하는 유형의 분위기나 환경을 조성하는 데 도움이 됩니다.

2D 광원 유형 및 설정



해피 하비스트에서 매달린 램프에 광원 헤일로가 있는 스프라이트 사용

Unity의 고급 **2D 동적 조명 시스템**을 사용하고 스프라이트에 보조 텍스처와 같은 기술을 결합하면 레벨의 분위기를 극적으로 전달하고 게임플레이의 완성도를 높일 수 있습니다.



2D 게임에서는 2D 광원 시스템을 적절하게 사용하여 분위기와 몰입감을 크게 향상시킬 수 있습니다. 이미지는 Unity 샘플 프로젝트인 로스트 크립트(Lost Crypt)에서 가져온 것입니다.



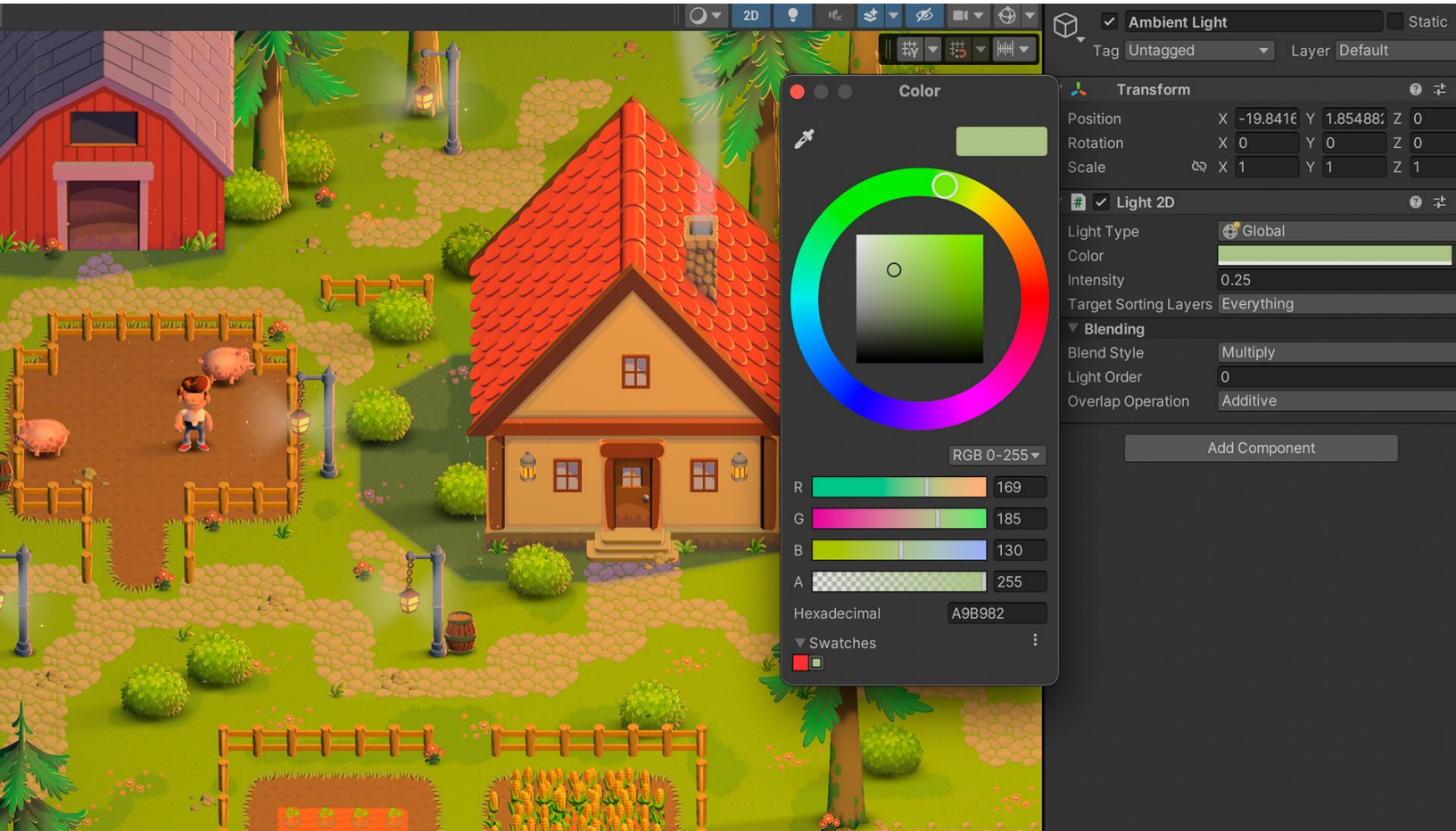
2D 광원은 Light 2D 컴포넌트가 연결된 게임 오브젝트입니다. [Sprite Renderer](#), [Sprite Shape Renderer](#), [Tilemap Renderer](#)와 함께 작동하며, Unity 6.3부터는 [MeshRenderer](#) 및 [SkinnedMeshRenderer](#)가 있는 3D 게임 오브젝트와 함께 작동합니다(자세한 내용은 3D 및 2D 믹스 섹션 참조). 또한 [정렬 레이어](#)를 사용하며, 각 광원은 하나 이상의 정렬 레이어에 영향을 줄 수 있습니다. Target Sorting Layers 드롭다운 목록에서 영향을 받을 레이어를 선택할 수 있습니다. 동일한 정렬 레이어에 있는 광원의 렌더링 순서를 제어하려면 Light Order 옵션을 사용합니다.



2D 프리폼 조명의 사용 사례 예시

4가지 유형의 2D 광원이 있습니다.

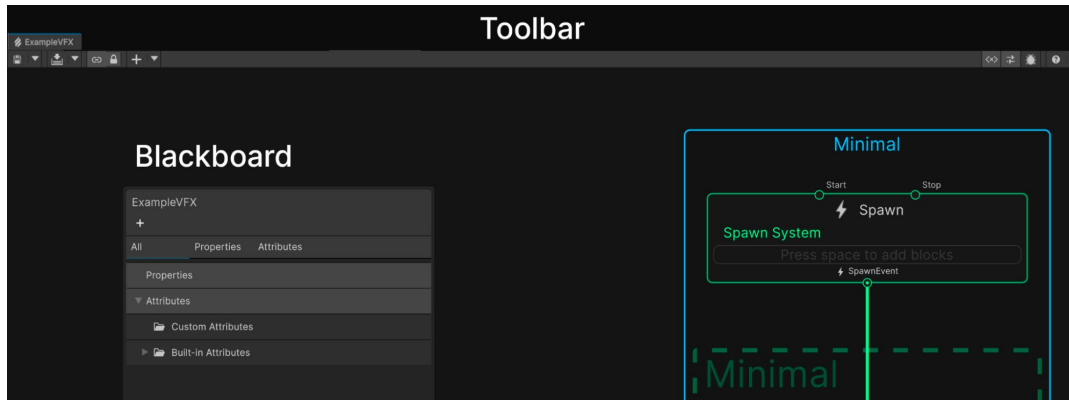
- **Freeform:** 이러한 광원은 면이 n 개인 폴리곤과 같은 모양이 될 수 있습니다. 비유기적 환경이나 스타일라이즈드 환경에 사용할 수 있는 광원 유형입니다. 환경의 큰 부분(예: 용암 웅덩이)을 효율적으로 비추거나, 광원의 형태를 시뮬레이션하거나(예: 천장의 구멍을 통해 들어오는 빛줄기), 창문에 광원이 비치는 형태를 따르는 경우에 좋은 선택입니다.
- **Sprite:** 이 셰이프를 사용하면 어떤 스프라이트든 광원의 텍스처로 사용할 수 있습니다. 다른 광원 유형으로 구현할 수 없는 특정 셰이프가 필요한 경우에 도움이 됩니다. 가능한 텍스처의 예로는 렌즈 플레어, 섬광, 광원 쿠키, 디스코 볼 광원 같은 광원 셰이프의 투사, 벽에 별을 투사하는 유아용 램프 등이 있습니다.
- **Spot:** 이 광원은 원형이나 부채꼴 형태를 띌 수 있습니다. 스포트라이트나 햇불, 촛불, 자동차 조명, 손전등, 볼류메트릭 광원 등으로 특정 지점을 비추는 경우에 좋은 옵션입니다.
- **Global:** 셰이프가 없는 대신 타기팅된 정렬 레이어의 모든 오브젝트를 비춥니다. 전역 광원은 블렌드 스타일당 하나(광원과 스프라이트 간의 상호 작용 방식), 정렬 레이어당 하나만 사용할 수 있습니다. 환경의 기본 조명을 적용할 때 전역 광원을 먼저 사용하면 좋습니다.



전역 광원은 전체 씬에 영향을 미쳐 게임 월드의 분위기를 쉽게 바꿀 수 있습니다. 씬에는 전역 광원이 하나만 있어야 하며, 파라미터를 조작하여 강도를 낮추고 씬에 보라색 색조를 적용하면 밤 시간과 같은 다양한 환경 조건을 손쉽게 시뮬레이션할 수 있습니다.

조명 유형별로 사용할 수 있는 **설정**은 다음과 같습니다.

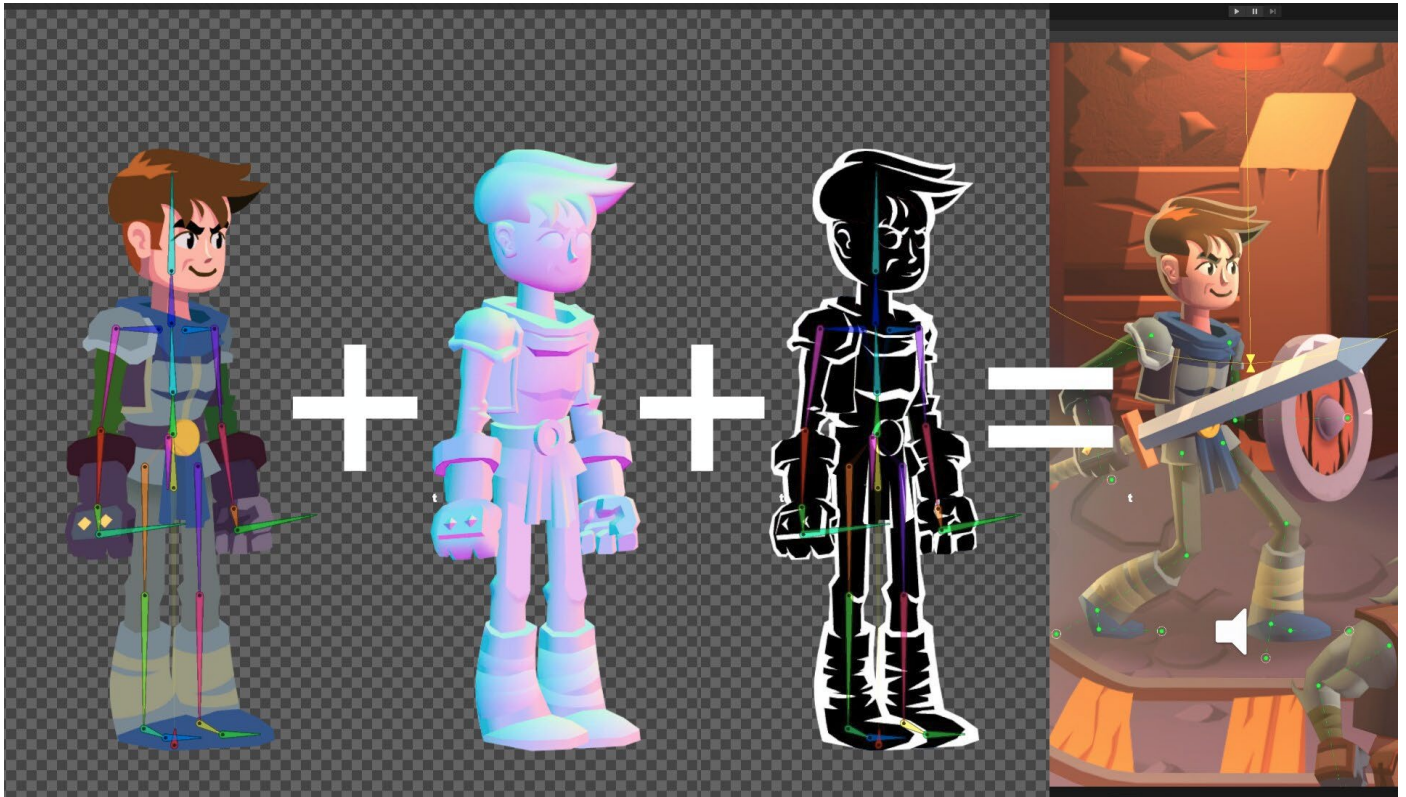
- **Blend Style:** Blend Style은 씬에서 특정 광원이 스프라이트와 상호 작용하는 방식을 결정합니다. 씬의 모든 광원은 2D Renderer에 정의된 네 가지 사용 가능한 Blend Style 중 하나를 선택해야 합니다. 사용할 수 있는 블렌드 모드는 세 가지이며, 각 모드에서는 스프라이트에 광원을 비추는 방식을 제어합니다.
 - **Additive:** 각 픽셀의 RGB 값을 기본 레이어의 픽셀 값에 추가합니다. 또한 밝기를 추가합니다.
 - **Multiply:** 각 픽셀의 RGB 값(정규화된)을 기본 레이어의 픽셀 값과 곱하여 색상을 블렌딩합니다.
 - **Subtractive:** 각 픽셀의 RGB 값을 기본 레이어의 픽셀 값에서 차감하여 더 어두운 톤을 만듭니다.



왼쪽에서 오른쪽 순으로 Additive, Multiply, Subtractive 블렌드 스타일

- **Light Order:** 이 값은 렌더 대기열에서 같은 정렬 레이어를 타기팅하는 다른 광원과 상대적인 광원 위치를 결정합니다.
- **Overlap Operation:** 이 프로퍼티는 선택한 광원이 다른 렌더링된 광원과 상호 작용하는 방식을 제어합니다. 이 프로퍼티에는 두 가지 모드가 있습니다.
 - **Additive:** 광원이 다른 광원과 가산적으로 블렌딩되어 교차하는 광원의 픽셀 값이 더해집니다. 이 모드가 기본 광원 블렌딩 동작입니다.
 - **Alpha Blend:** 광원이 알파 값에 따라 함께 블렌딩됩니다. 두 광원이 교차하는 부분에서 하나로 다른 광원을 덮어쓰는 데 사용할 수 있지만, 광원의 렌더 순서도 다양한 광원의 **Light Order**에 따라 달라집니다.
- **Shadows:** 이 옵션을 활성화하면 Shadow Caster 2D 컴포넌트가 있는 게임 오브젝트가 이 광원에서 그림자를 드리웁니다.
- **Volumetric:** 광원의 감쇠와 그림자가 투사되는 어둠을 제어할 수 있습니다.
- **Normal maps:** 활성화하면 스프라이트의 노멀 맵 정보가 픽셀 방향에 따라 광원의 양을 계산하는 데 사용됩니다(아래에서 보조 텍스처에 대한 자세한 내용 참조).

보조 텍스처



보조 텍스처를 스프라이트에 추가

보조 텍스처는 **노멀 맵**과 **마스크 맵**으로 생성됩니다. 2D 광원 사용 시 필수는 아니지만 이에 할당할 수 있는 예산과 시간, 아트 리소스가 있으면 게임 비주얼의 품질을 새로운 차원으로 높일 수 있습니다. 광원에 반응하는 캐릭터와 환경의 디테일이 향상되므로 보다 입체적으로 보이는 셰이프를 만들 수 있습니다.



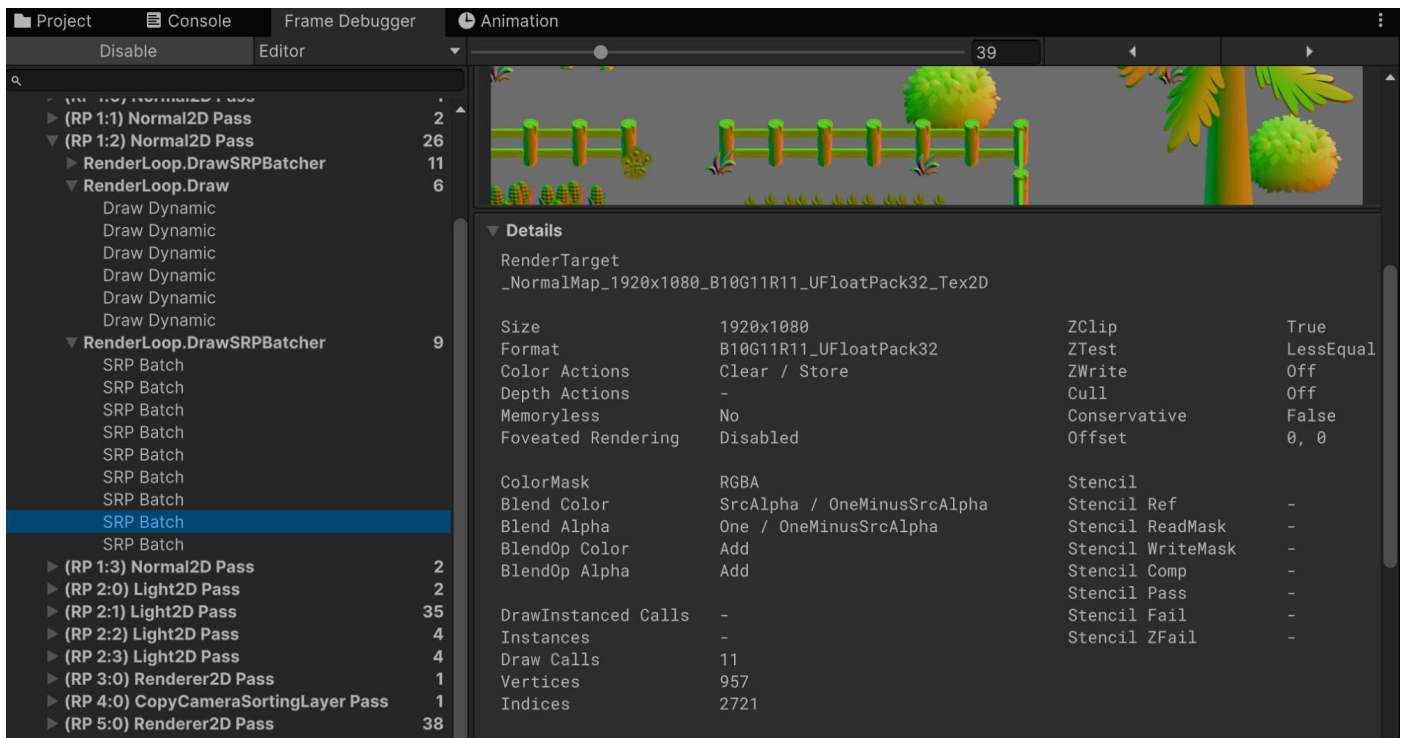
따뜻한 느낌의 랜턴 조명을 받고 있는 해피 하비스트의 행복한 돼지. 왼쪽은 노멀 맵을 사용하는 광원, 오른쪽은 노멀 맵을 사용하지 않는 광원입니다.

필요에 따라 스프라이트 에셋마다 머티리얼에서 사용할 수 있는 [보조 텍스처](#)를 할당할 수 있습니다. **Sprite Editor > Secondary Textures**로 이동하여 보조 텍스처를 변경합니다. 기본적으로 두 가지 텍스처 유형 (노멀 맵 및 마스크 맵)을 할당할 수 있습니다.

머티리얼이 아닌 에셋 수준에서 텍스처를 할당해야 하는 데는 몇 가지 이유가 있습니다.

스프라이트와 보조 텍스처가 다른 경우에도 스프라이트를 사용하는 렌더러가 머티리얼(Sprite-Lit-Material)을 공유할 수 있다는 것이 첫 번째 이유입니다. 즉 배치를 통해 더 효율적으로 렌더링할 수 있습니다.

스프라이트 셰이더를 사용하는 경우 노멀 맵과 마스크 맵을 머티리얼에 추가해야 합니다. 각 스프라이트에는 스프라이트, 노멀 맵, 마스크 맵이 포함된 자체 머티리얼이 필요합니다. 그 결과 프로젝트에는 수많은 머티리얼이 생기게 됩니다. 하지만 모든 스프라이트에 추가 텍스처를 설정하면 하나의 스프라이트 머티리얼만 사용해도 됩니다. 머티리얼이 하나이므로 드로우 콜도 한 번이면 충분합니다.



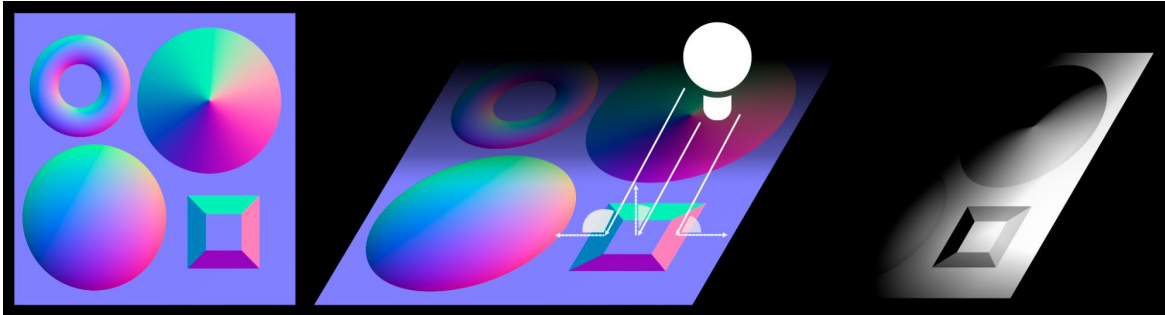
프레임 디버거를 사용하면 프레임 렌더링의 모든 단계를 관찰할 수 있습니다. 배치할 수 없는 머티리얼이 많으면 성능이 저하됩니다.

전자책의 끝부분에 있는 최적화 섹션에서 더 많은 팁을 얻어 보세요.

노멀 맵

제대로 구현된 노멀 맵이 있으면 필요에 따라 스프라이트를 3D처럼 보이게 할 수 있습니다. 노멀 맵의 각 픽셀은 메인 텍스처의 각도에 대한 데이터를 저장합니다. RGB(빨간색, 녹색, 파란색) 채널은 X, Y, Z 좌표로 각도 데이터를 저장합니다. 노멀 맵에 활용하는 모든 광원은 방향이 있으며, 노멀 맵이 있는 텍스처의 픽셀은 이 방향과 픽셀의 방향에 따라 음영 처리됩니다. 이 작업은 실제 환경과 동일한 방식으로 이루어집니다. 픽셀이 광원의 방향을 바라보면 빛이 비춰지고 바라보지 않으면 빛을 받지 않습니다.

RGB 값이 노멀 맵의 각도에 어떻게 영향을 미치는지 살펴보세요.



노멀 맵에서 받는 빛은 픽셀이 향하고 있는 벡터와 광원 사이의 각도가 얼마나 큰지에 따라 달라집니다(내적이라고도 함). 각도가 작을수록 두 벡터가 더 정렬되고 픽셀이 받는 빛이 더 많아집니다. 위 이미지의 오른쪽에 있는 노멀 맵은 Unity의 2D 광원을 통해 빛을 받습니다.

위 이미지는 보라색 픽셀이 카메라를 바라보는 노멀 맵입니다. RGB 값은 각각 127, 127, 255입니다. 각 컬러 채널은 0~255 사이의 값을 가질 수 있으므로 127은 거의 중간값입니다. 왼쪽 표면을 바라보려면(-90도) R 컬러 값을 0으로 설정해야 합니다. 오른쪽 표면을 바라보려면 R을 255로 설정합니다. 정면 아래 또는 위를 바라보려면 G 채널을 각각 0 또는 255로 설정합니다.



해피 하비스트에서 애니메이션 캐릭터, 프람 및 타일맵에 적용된 노멀 및 마스크 맵의 예시

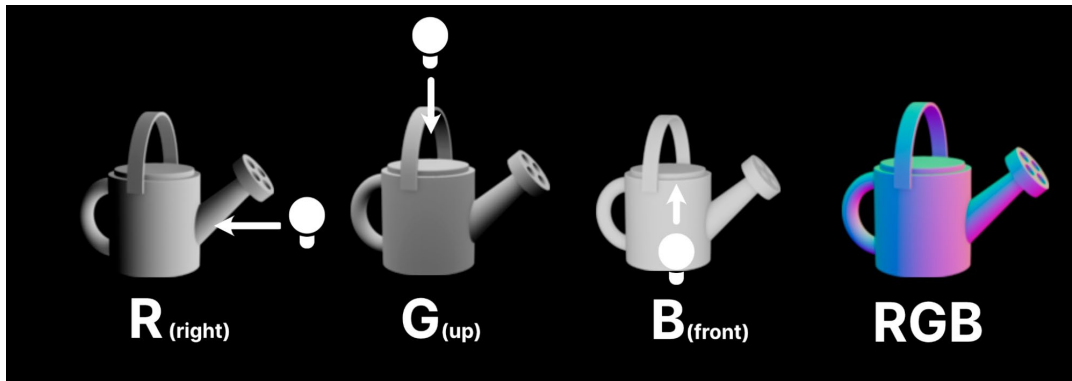
노멀 맵에 사용하는 페인팅 기법

노멀 맵을 만드는 방법은 사용하는 아트 스타일과 기술에 따라 크게 달라집니다. 3D 머티리얼을 위한 노멀 맵 생성 기법은 일반적으로 2D 스프라이트에도 적용됩니다. 다음 팁을 사용하여 외형을 완벽하게 제어해 보세요.

조명 방향을 수동으로 페인팅하기

노멀 맵을 페인팅하는 한 가지 방법은 다양한 각도에서 빛나는 스프라이트를 그린 다음 하나의 텍스처로 결합하는 것입니다. 스프라이트는 R 채널에서는 오른쪽에서 나오는 하나의 광원을 통해, G 채널에서는 위쪽에서 나오는 하나의 광원을 통해 빛을 받습니다. B 채널에서는 스프라이트가 정면으로부터 빛을 받지만, 2D 스프라이트에 노멀 맵을 사용하는 경우에는 작업 간소화를 위해 이 채널을 생략할 수 있습니다. 2D에서는 정면 조명이 전반적인 셰이딩 효과에 크게 기여하기 때문입니다.

그러나 이 접근 방식은 X축과 Y축에 두 번 이상 셰이딩을 칠해야 하므로 시간을 많이 소요할 수 있습니다.



셰이딩된 두 개의 그레이스케일(오른쪽과 위에서 빛을 비춤) 이미지를 노멀 맵으로 결합하는 방법. 앞쪽의 셰이딩된 그레이스케일은 선택적으로 사용할 수 있습니다.

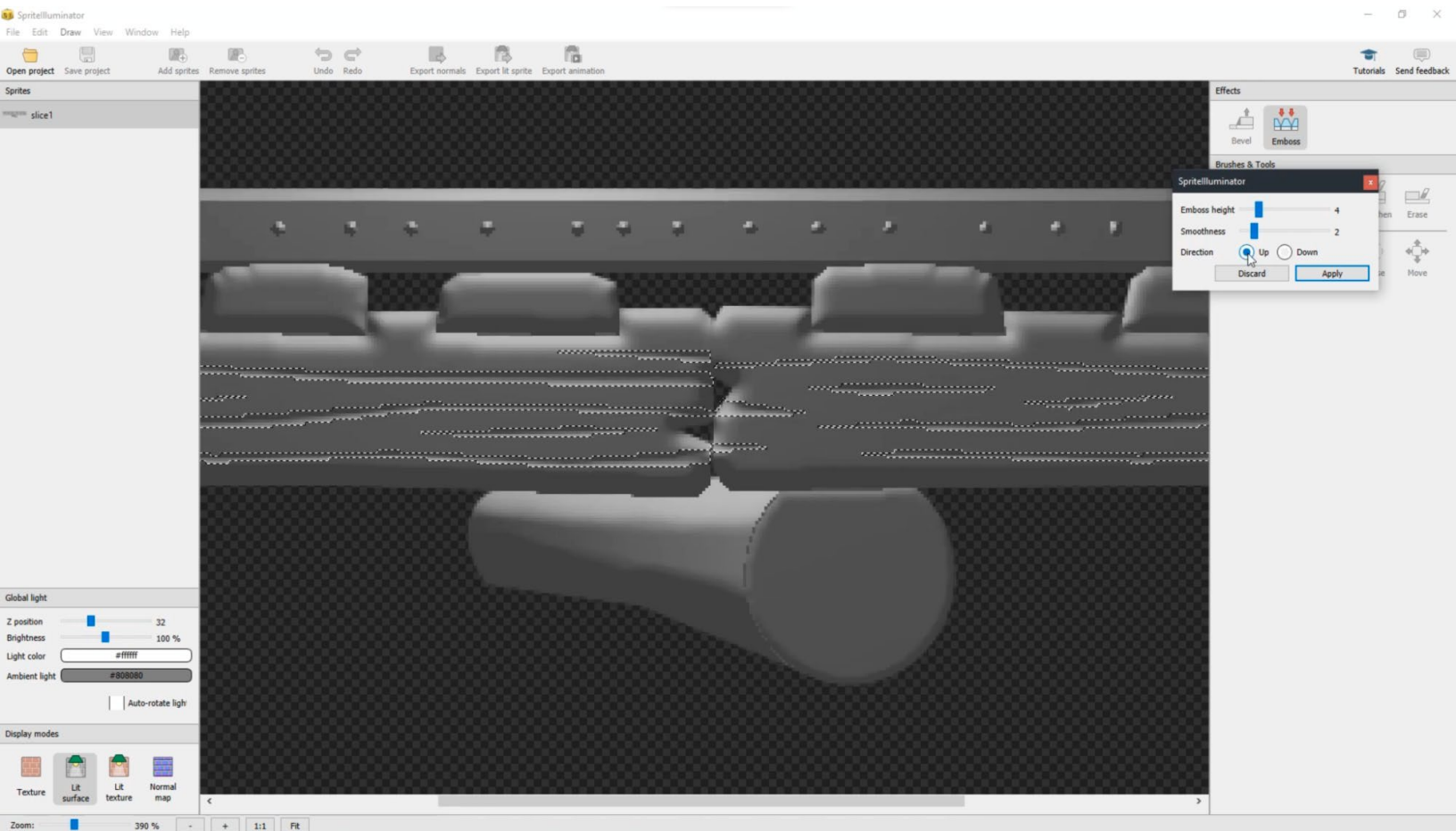
노멀 맵 제너레이터

또 다른 페인팅 접근 방식은 노멀 맵 제너레이터 앱을 사용하는 것입니다. 이러한 앱에서 스프라이트를 열면 몇 번의 클릭만으로 노멀 맵을 생성할 수 있습니다. 제너레이터 앱은 스프라이트의 각도를 고려하지 않으므로 전체 스프라이트에는 사용하지 마세요. 오브젝트를 인식하지도 못합니다. 그 대신 셰이프를 추정할 때는 스프라이트 컬러를 참고하거나 이미지 편집 앱 또는 부조 조각품의 굴곡이나 엠보싱과 유사한 일반 필터를 추가합니다. 예를 들어 면의 각도를 인식하지는 못하지만 각도에 변화가 있는지 추측하려고 시도합니다. 이러한 제약이 있어도 체인, 케이블, 용의 꼬리 같은 굴곡이 있는 스프라이트 섹션의 노멀 맵과 벽돌, 돌, 나무 같은 표면 노멀의 노멀 맵을 생성하는 데 유용합니다.

섹션을 노멀 맵 제너레이터로 임포트하고, 값을 조정하고, 익스포트한 후 필요한 부분과 디테일을 직접 추가하세요.

일부 노멀 맵 제너레이터에 포함되는 내용은 다음과 같습니다.

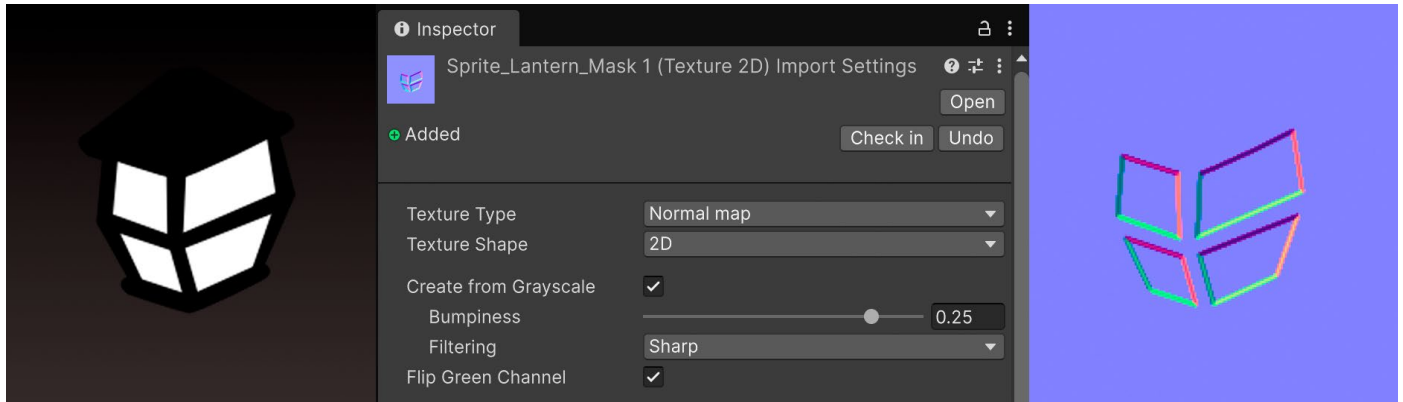
- [Spriteluminator](#)
- [NormalPainter](#)
- [Krita의 Tangent Normal Brush](#)
- [Laigter](#)
- [Sprite Lamp](#)



Spriteluminator에서 Emboss 툴을 사용하여 노멀 맵에 균열 생성

Unity에서 그레이스케일로 노멀 맵 만들기

마지막으로 Unity에서는 그레이스케일 하이트맵으로 노멀 맵을 생성할 수 있습니다. 그레이스케일 하이트맵에서 검은색은 표면의 최저 높이를, 흰색은 최고 높이를 나타냅니다. 이미지를 노멀 맵으로 임포트하고 **Create from Grayscale** 옵션을 선택하세요. 엔진에서 벗어날 필요 없이 빠르게 노멀 맵을 생성할 수 있어서 유용한 기법입니다.



Create from Grayscale 옵션을 사용하여 텍스처를 노멀 맵으로 변환. **Flip Green Channel** 옵션을 사용하면 엠보스 효과가 돌출되거나 움푹 들어간 것처럼 보일 수 있습니다.

이 방법을 사용하면 인스펙터에 **Bumpiness** 슬라이더가 나타납니다. Unity는 픽셀 밝기를 사용하고 높이 차이를 노멀 맵 각도로 전환하며, 이 슬라이더는 해당 각도의 경사도를 제어합니다. 범프 값이 낮으면 하이트맵에서는 콘트라스트가 높더라도 각도와 요철이 완만하게 전환됩니다.

컬러 샘플링

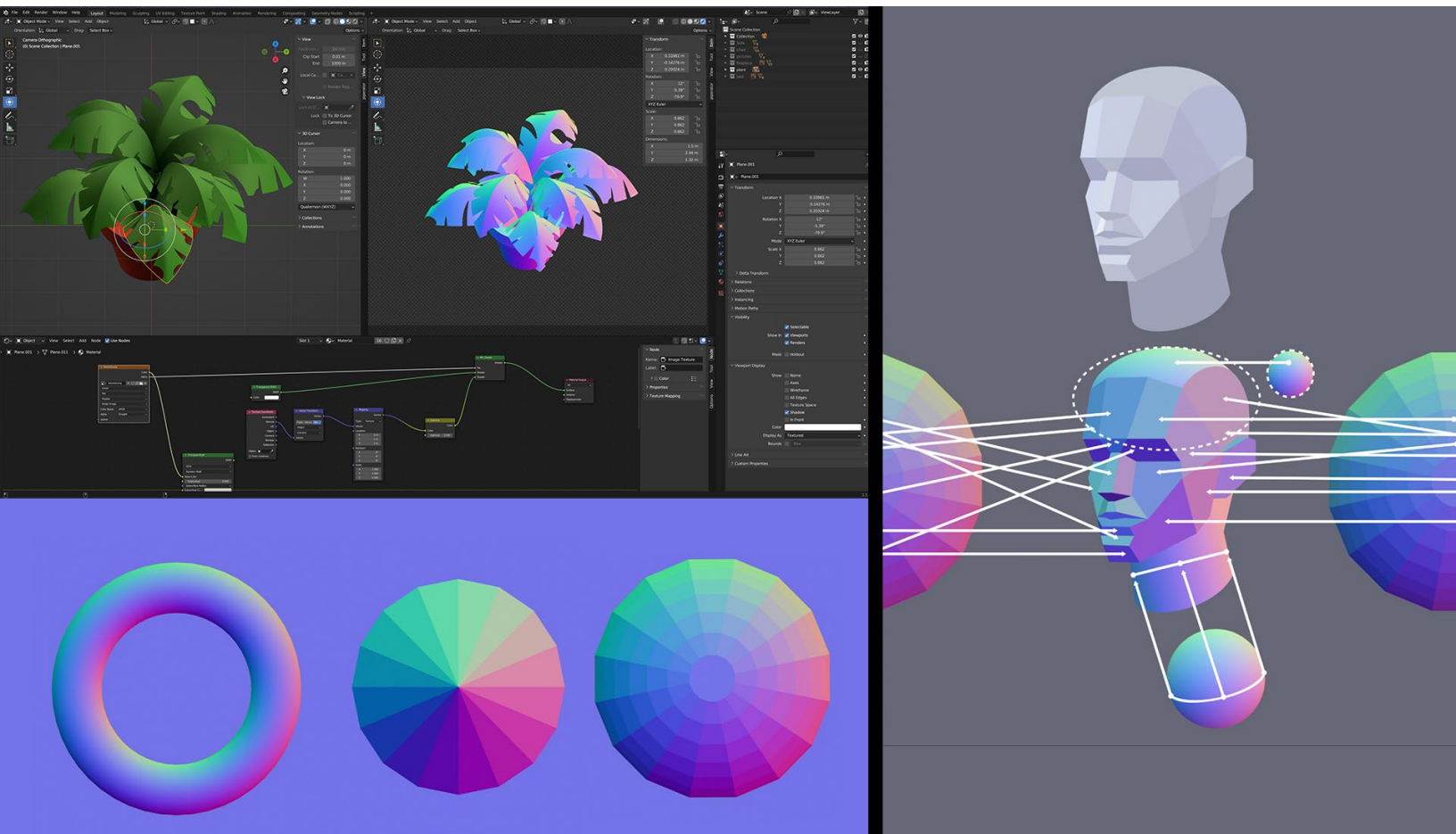
노멀 맵을 페인팅하려면 우선 여러 각도에 사용할 컬러를 파악해야 합니다.

먼저 온라인 템플릿에서 표면 각도를 나타내는 데 사용되는 컬러를 샘플링할 수 있도록 노멀 맵 팔레트를 확보합니다. 페인팅 워크플로가 어떻게 선호하는 페인팅 앱에 팔레트를 복사하고 컬러 피커를 사용하여 노멀 맵을 페인팅할 컬러를 선택하면 됩니다.

각도 컬러가 100% 정확할 필요는 없으며 몇 도 정도는 차이를 보이지 않습니다. 하지만 스프라이트는 전반적으로 그럴듯한 셰이프를 유지해야 합니다. 컨텍스트에 어울리지 않는 각도 컬러를 사용하면 빛을 받았을 때 전체 셰이프가 무너집니다.

공간적 상상력을 발휘할 수 있어야 하므로 노멀 맵을 페인팅하는 작업이 처음에는 어려울 수 있습니다. 머리의 기본 평면과 같이 간단한 것으로 시작하는 것이 좋습니다. 아래의 오른쪽 이미지에 있는 단순화된 사람 머리 모델은 로우 폴리 느낌을 줍니다. [이 zip 파일](#)을 다운로드해 준비된 예시를 확인할 수 있습니다.

노멀 맵을 페인팅할 때는 스프라이트의 부분을 구성하는 기본 3D 셰이프를 상상한 다음, 개별 부분의 각도를 시각화하세요. 각도를 알면 팔레트 스프라이트의 어느 부분에서 컬러를 샘플링할지 알 수 있습니다.



왼쪽 상단: 노멀 맵 생성을 위한 Blender 프랩 제작, 템플릿에서 컬러를 샘플링하여 머리 부분의 노멀 맵 페인팅, 셰이프 템플릿

이 사람 머리의 예시는 평면에서 작업하고 있지만 더 연한 브러시로 페인팅해도 프로세스는 비슷합니다. 하드 에지를 블렌딩하여 더욱 자연스러운 모습을 얻을 수 있습니다.

몇 가지 유용한 팁을 소개하자면, 구형 셰이프가 있는 경우에는 팔레트에서 노멀 구체를 붙여 넣을 수 있습니다. 원통형 셰이프가 있는 경우 구체의 일부분을 붙여 넣고 늘리거나 그레디언트를 만드세요.



노멀 맵의 일부분을 복사하여 붙여 넣고 회전하면 셰이딩이 깨집니다. 하지만 이 방법을 유용하게 활용할 수도 있습니다. 예를 들어 오목한 구형 세이프가 필요한 경우 구체를 180도 회전하면 구멍을 만들 수 있습니다.



엔진 내에서 흰 스프라이트의 노멀 맵은 이렇게 표시됩니다.

본인에게 가장 적합한 노멀 맵 생성 방법을 선택하세요. 여러분의 게임과 어울리는 기성 에셋이 다양하게 있을 것이므로 게임에서 가장 눈에 띄는 오브젝트에 집중하고 다른 부분은 단순화하세요.

다음 동영상에서는 다양한 페인팅 기법을 설명합니다.



보조 텍스처: 빛을 받는 스프라이트 및 2D VFX

2D 조명용 스프라이트 처리

기본 컬러 스프라이트를 기록해 두세요. 게임에 2D 조명을 많이 사용하고 노멀 맵을 최대한 활용하려는 경우에는 스프라이트에 명암을 칠하지 마세요.

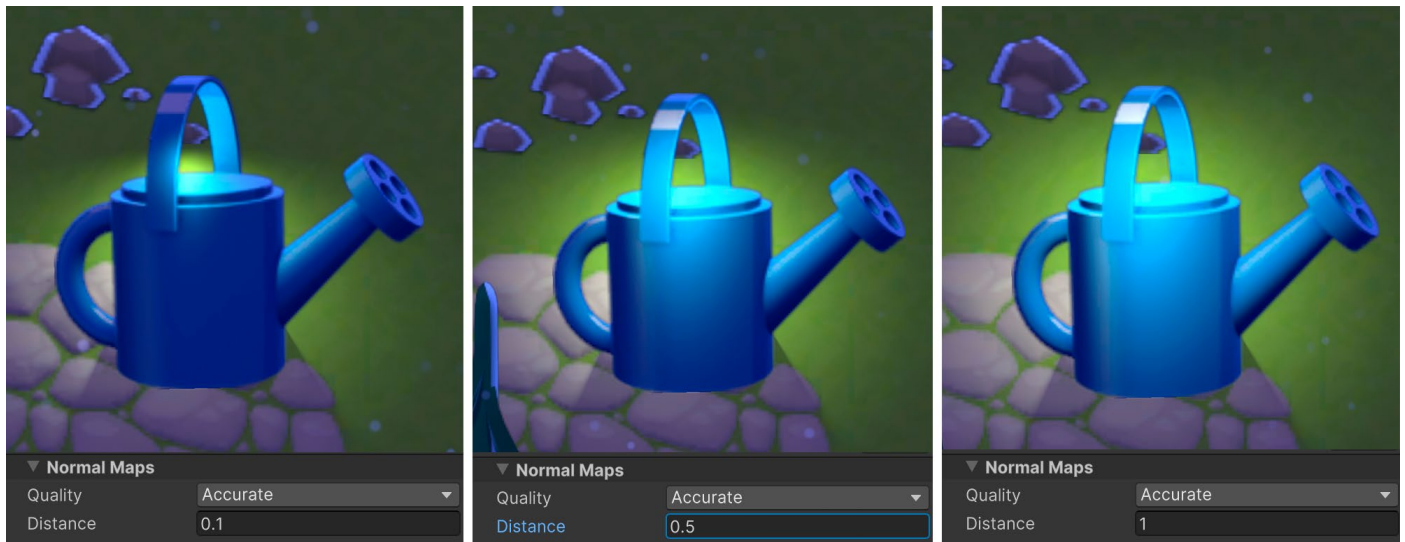
2D 조명은 그림자를 칠한 스프라이트에 어울리지 않습니다. 노멀 맵에서 조명을 칠하기 때문에 일이 두 배로 늘어나 버리기도 합니다. 그 대신 비방향광 그림자를 페인팅하면 태양광 같은 방향 광원을 사용하지 않는 한 스프라이트가 근사해 보일 것입니다.



2D 광원이 꺼진 상태의 스프라이트는 컬러 정보(알베도)가 있어도 광원 정보나 그림자 정보가 없기 때문에 평평하게 보입니다.

노멀 맵 활성화

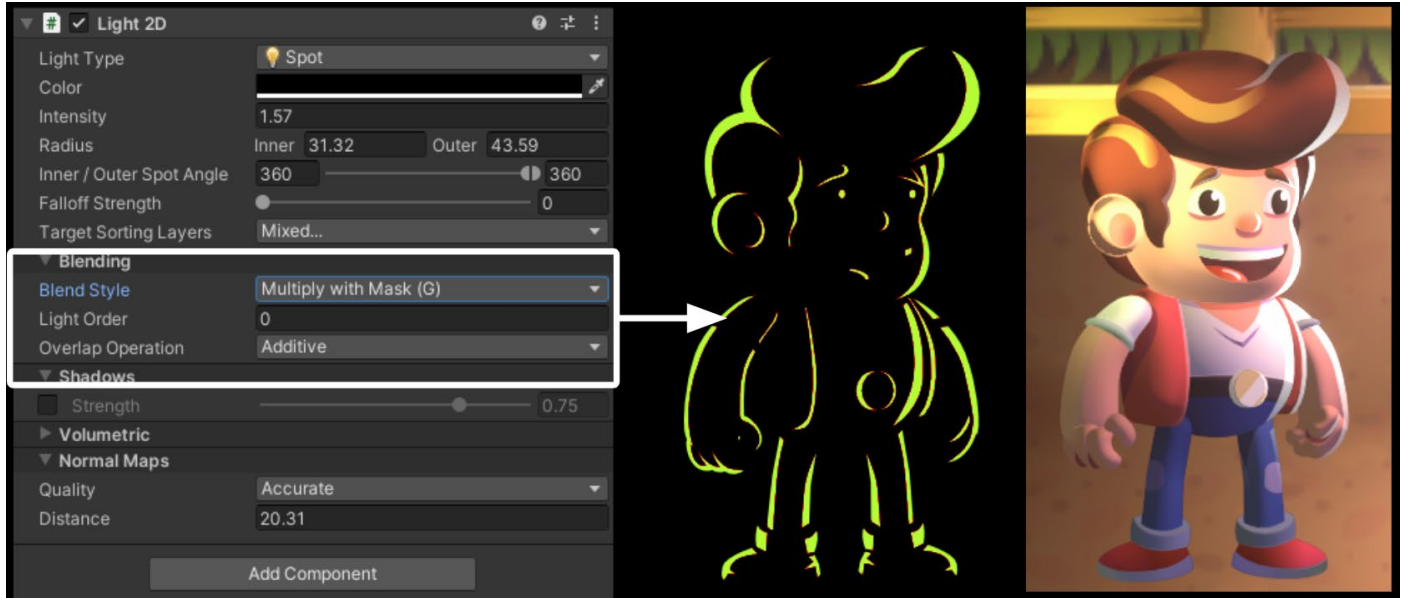
광원과 노멀 맵은 Unity 샘플 해피 하비스트와 드래곤 크래셔에서 입체감을 연출하기 위해 광범위하게 사용됩니다. 스폿, 점 및 프리폼 광원과 함께 노멀 맵을 사용할 수 있습니다. 스프라이트에서 노멀 맵을 사용하려면 [2D 광원](#) 오브젝트에서 노멀 맵을 활성화해야 합니다. 광원과 스프라이트 표면 간의 거리도 조절할 수 있습니다. 효과 크기를 늘리거나 줄여 보세요.



2D 광원에서의 노멀 맵 지원 설정 및 거리 설정에 따른 효과

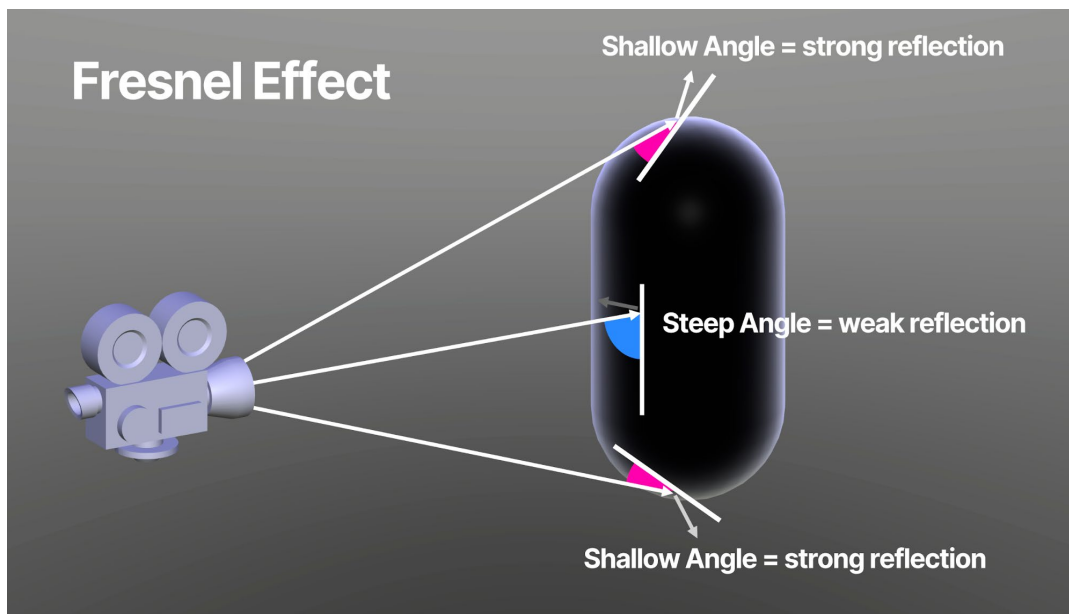
마스크 맵

마스크 맵은 림 라이팅과 같은 추가 조명에 사용할 수 있습니다. 자체 텍스처를 추가하고 [Shader Graph](#) 셰이더에서 이름으로 참조하여 기타 시각 효과를 추가할 수 있습니다.



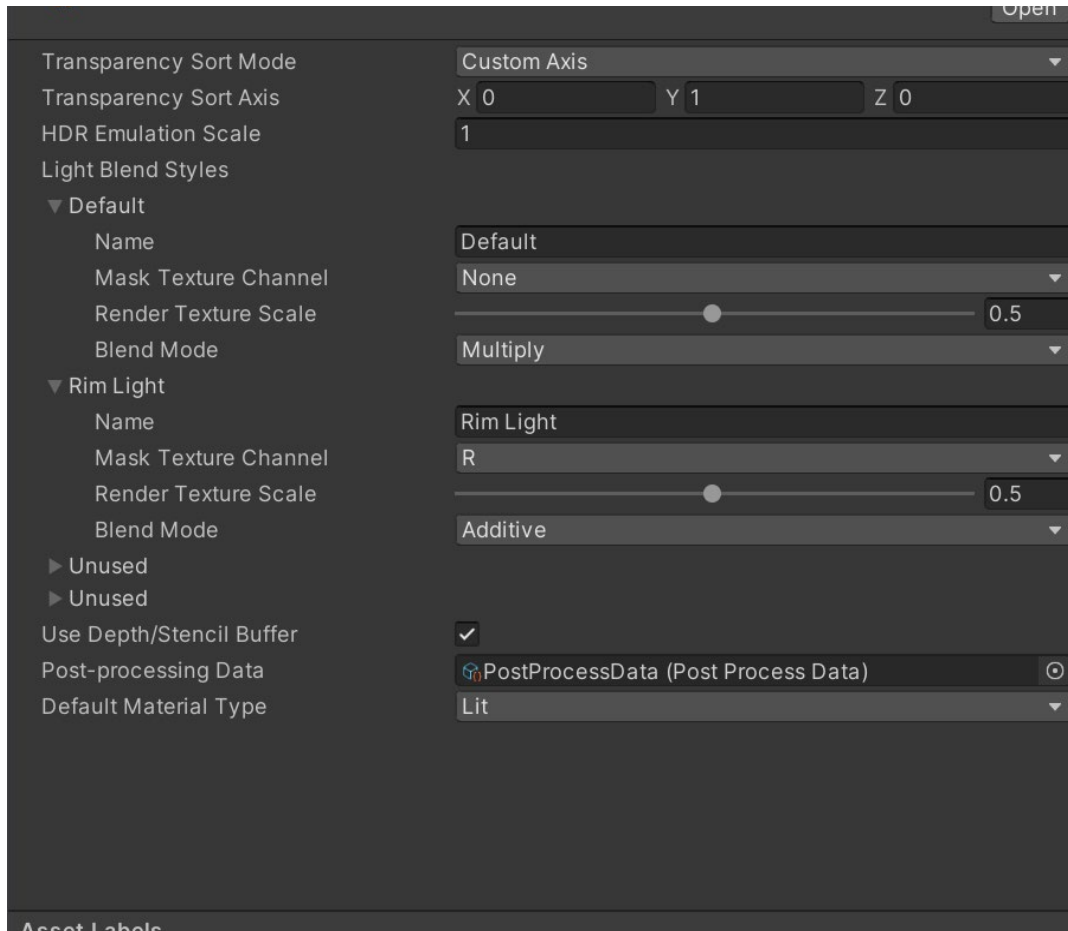
농부 캐릭터가 해피 하비스트에서 돋보이도록 하기 위해 '대표' 조명 스타일이 만들어졌습니다. 이 효과는 그린 채널에 있는 마스크 맵에만 영향을 미치는 블렌딩 방식을 사용하며, 이 채널은 캐릭터의 하이라이트를 페인팅하는 데 사용됩니다.

림 라이팅은 캐릭터의 윤곽을 강조할 때 사용하는 효과입니다. 오브젝트의 뒤쪽에서 비추는 광원과 빛 산란의 자연적인 특성을 시뮬레이션합니다. 이 기능을 프레넬 효과라고 합니다. 현실에서 빛이 표면에 닿는 각도가 넓어지면 오브젝트에서 더 많은 빛이 반사됩니다.



프레넬 효과 설명

따라서 멀리서 보았을 때는 호수에 있는 물이 더 많이 반사되지만 아래를 내려다보았을 때는 물 속에 있는 발이 보이게 됩니다. 오브젝트의 가장자리가 더 많이 반사되는 이유도 마찬가지입니다. 2D 그래픽스에서는 마스크 맵과 블렌드 스타일을 사용하여 이 효과를 시뮬레이션할 수 있습니다.



2D Renderer Data 에셋의 Light Blend Styles 옵션 설정

프레넬 효과를 만드는 방법

위에서 머리 샘플을 다운로드했다면, Unity에서 스프라이트를 열고 다음 단계를 따라 프레넬 효과를 만들어 볼 수 있습니다.

1. 2D Renderer Data 에셋의 **Light Blend Styles** 옵션 아래에는 서로 다른 블렌딩 스타일을 위한 네 가지 옵션이 있습니다. 기본 옵션인 첫 번째 옵션은 그대로 두고 두 번째 옵션을 활용합니다. 'Rim Light' 또는 'Fresnel' 등 적당한 이름을 지정합니다.
2. **Mask Texture Channel**을 R로 설정하면 광원이 마스크 맵 텍스처의 빨간색을 사용합니다.
3. **Blend Mode**를 **Additive**로 설정합니다. 그러면 기존 조명에 광원이 추가되어 밝기가 상승합니다.

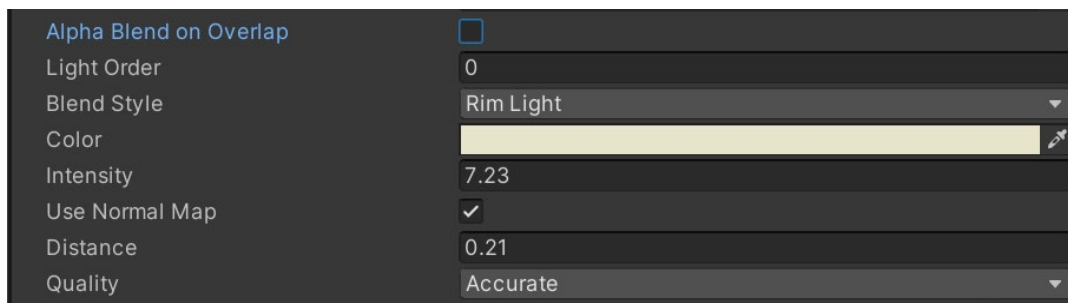


4. 프레넬 광원은 마스크 맵의 채널을 한 개만 사용하지만(이 경우 R), 알아보기 쉽게 만들기 위해 마스크 맵을 흑백으로 표현합니다.
 - 빛을 반사하는 부분은 흰색, 반사하지 않는 부분은 검정색입니다. 프레넬 효과는 오브젝트의 가장자리에 영향을 미치므로 기본 스프라이트를 복사하고 검정색으로 페인팅한 후 가장자리는 흰색으로 강조합니다. 그러면 뒤쪽이 밝게 빛나는 오브젝트와 비슷해지기 시작합니다.
5. 작업 속도를 높이려면 오브젝트에 이너 글로우 효과를 추가하고 디테일을 묘사합니다. 이 경우 이 프로세스를 단축할 수 있는 앱이 없으므로 본인의 그림 실력을 최대한 발휘해야 합니다.

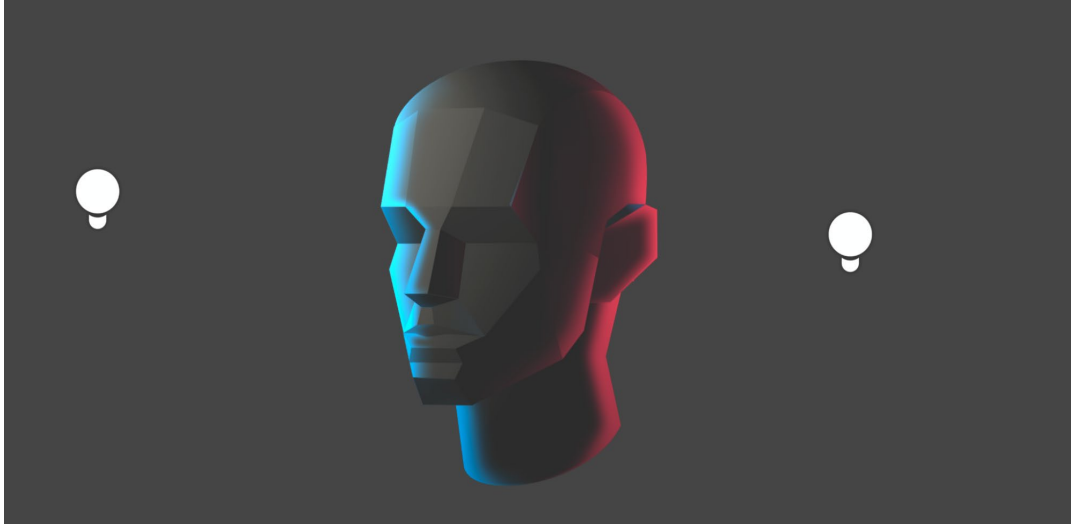


샘플의 기본 평면 머리 스프라이트에 대한 마스크 맵

6. 광원이 마스크 맵에 영향을 미치도록 하려면 Blend Style을 이전에 만든 것으로 변경합니다.



광원의 Blend Style을 Rim Light로 변경

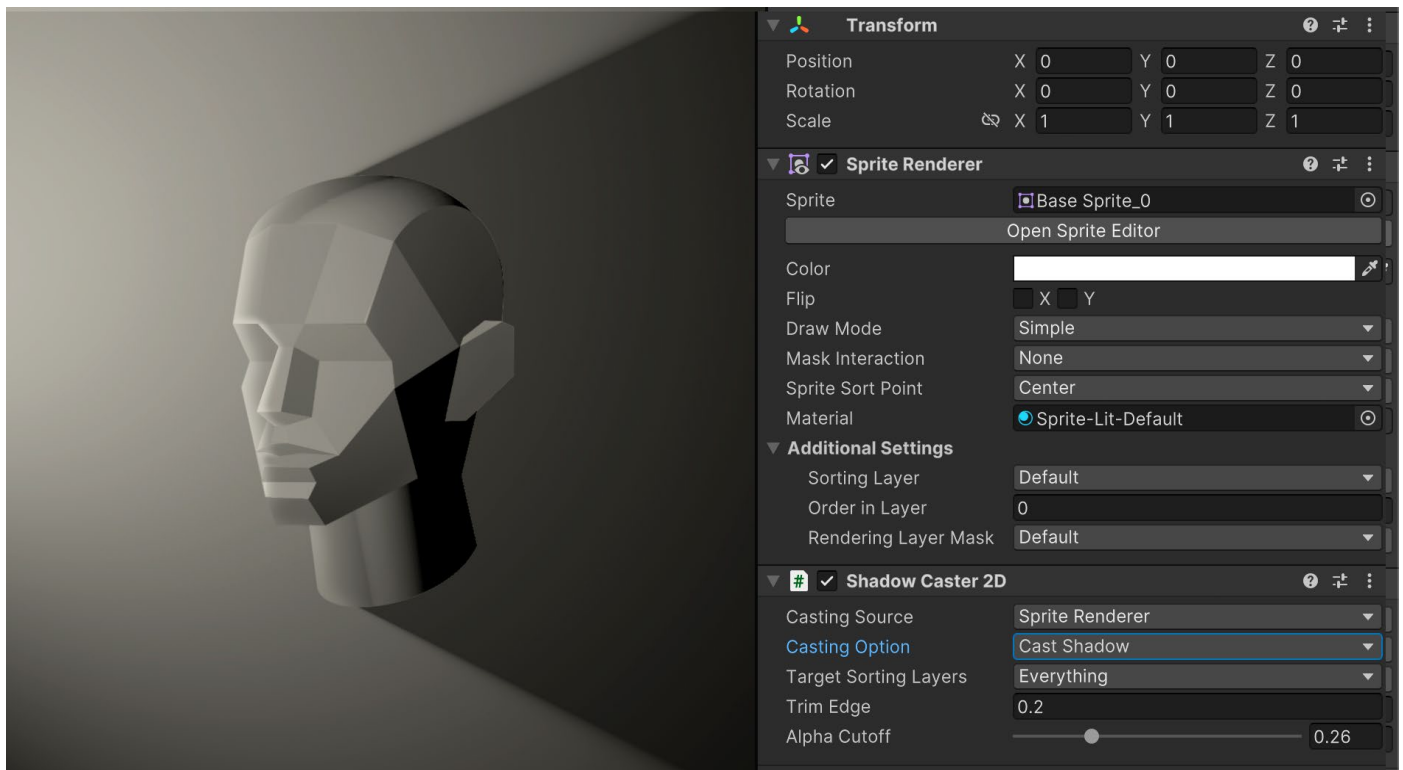


Unity 에디터에서 본 스프라이트의 림 라이팅 효과

7. 프레넬 광원은 Use Normal Maps 옵션이 활성화되어 있고 Distance가 낮은 값으로 설정되어 있는 경우에 가장 효과가 좋습니다. 이렇게 설정하면 오브젝트의 반대편이 강조되지 않습니다.

2D 그림자

게임 오브젝트가 2D 광원으로부터 그림자를 드리우려면 Shadow Caster 2D 컴포넌트를 오브젝트에 추가합니다.

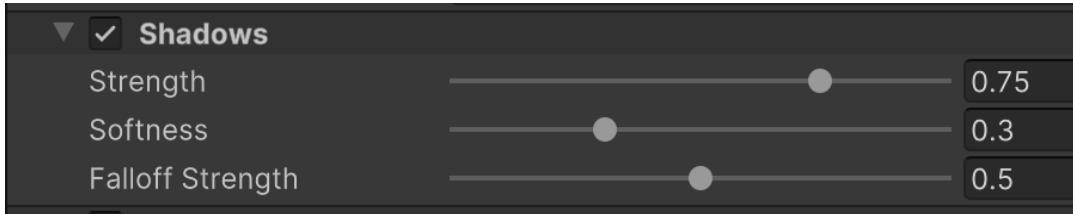


Shadow Caster 2D 컴포넌트



Unity 6에서는 기본적으로 그림자를 드리우는 소스가 스프라이트 렌더러가 되며, 스프라이트 에셋의 자동 생성된 외곽선을 사용합니다. 외곽선의 트림 및 알파 컷오프에 대한 추가 설정이 있습니다.

또한 Shadow Caster의 셰이프 에디터로 지오메트리를 편집하여, 오브젝트의 실루엣에 맞추거나 직접 생성한 다른 셰이프를 따라가도록 할 수도 있습니다. 그림자를 드리우려면 광원에 몇 가지 옵션도 설정해야 합니다.



그림자를 드리우기 위한 2D 광원 옵션

그림자 속에 있는 다른 오브젝트를 비추는 광원의 불투명도는 **Strength** 옵션에 따라 결정됩니다. 이 값을 1로 설정하면 광원이 그림자 영역의 그 어떤 것도 비추지 않습니다.

Softness 옵션은 새도우 캐스터로부터의 거리에 따라 그림자 가장자리를 블러 처리하여 보다 자연스러운 효과를 주며, **Falloff Strength**는 그림자 가장자리의 강도를 조절합니다.

2D 광원 및 그림자를 위한 시각적 기법

이 섹션에서는 게임에서 창의적으로 조명 시나리오를 모방하는 방법을 살펴보겠습니다. 이러한 예시는 해피 하비스트 프로젝트의 스크립트와 에셋에서 가져온 것입니다.

무한 그림자

그림자를 드리우는 오브젝트는 무한 그림자를 생성하며, 이는 광원의 영역이 제한되거나 가로등 또는 벽난로와 같은 강하고 집중된 광원을 사용할 때 적합합니다.

새도우 캐스터는 일반적으로 횡스크롤 게임의 전체 오브젝트나 캐릭터지만, 아이소메트릭 또는 탑 다운 시점에서는 이 효과가 비현실적으로 보일 것입니다. 뎀스를 더 효과적으로 표현하는 한 가지 방법은 지면과 접하는 오브젝트의 바닥에 맞는 새도우 캐스터를 만드는 것입니다. 다음 예시에서 해피 하비스트의 Shadow Caster 2D 컴포넌트는 캐릭터 게임 오브젝트 내부에 있는 발 뼈에 연결되어 발에만 영향을 미칩니다.



왼쪽 이미지에서 새도우 캐스터는 캐릭터의 발 뼈에 연결되어 잘 작동하지만, 오른쪽 이미지에서는 주변 일반 조명에 대해 올바르게 작동하지 않습니다.

이 기술은 범위가 제한적이며 강도가 높은 광원에는 잘 작동하지만, 열린 공간의 태양광에서 광원이 오는 경우 무한 그림자 효과가 위의 나무 이미지처럼 이상하고 부자연스럽게 보입니다.

Unity의 2D 광원 시스템은 다른 유형의 조명과 그림자를 모사하는 데 창의적으로 유연하게 활용할 수 있습니다.

블룸 그림자

움직이는 캐릭터가 환경과 자연스럽게 어우러지도록 하는 오래된 기법으로, 그림자를 드리우는 오브젝트에 자식 오브젝트로 간단한 블룸 그림자 스프라이트를 추가합니다. 여기서 요령은 Light 2D 컴포넌트에서 **Light Type**을 **Sprite**로 설정하고, **Blend Style**을 **Multiply**로 설정하는 것입니다. 광원의 컬러가 어두우면 그 아래 영역을 어둡게 하여 **네거티브 조명**이라는 효과를 만듭니다.



그림자를 구현할 때 비용이 낮으면서도 효과적인 방법은 캐릭터 아래에 블룸 그림자로 2D 광원을 사용하는 것입니다.

탑 다운 게임에서 태양광을 통해 무한 그림자가 투사될 경우 이상하게 보일 수 있으며, 나무와 같은 정적 오브젝트에 대한 표준 블룸 그림자도 마찬가지입니다. 해피 하비스트에서는 하루의 시간에 따라 회전하고 늘어나는 긴 블룸 그림자를 만들었습니다. 그 결과, 아트 디렉션에 더 부합하는 소프트 새도우가 만들어집니다.

UpdateShadow 함수는 DayCycleHandler 스크립트에서 이 그림자를 회전시킵니다. 이 스크립트는 관리자 역할을 하며, 함수를 사용해 그림자의 크기와 회전을 하나씩 순회하면서 업데이트합니다.

늘어난 블룸 그림자는 다른 블룸 그림자와 마찬가지로 스프라이트 기반의 광원입니다. 해피 하비스트를 다운로드하고 열면 **Trees**라는 부모 게임 오브젝트 내부의 자식 오브젝트를 점검하여 이를 확인할 수 있습니다.



해피 하비스트의 나무와 덩굴에 대해 늘어난 블룸 그림자

낮-밤 사이클

2D 씬에는 3D 씬처럼 직접광이 없습니다. 그러나 X 및 Y 위치에서 스프라이트를 비추는 광원을 사용할 수 있으며, 노멀 또는 마스크 맵으로 효과를 더욱 돋보이게 할 수 있습니다. 이 기능을 활용해 하루가 진행됨에 따라 태양이 움직이는 효과를 연출할 수 있으며, 탑 다운 및 시뮬레이션 게임에서 핵심적인 요소가 될 수 있습니다. 저사양 플랫폼을 타기팅할 경우, 큰 광원을 사용하면 비용이 발생한다는 점도 기억해 두세요.

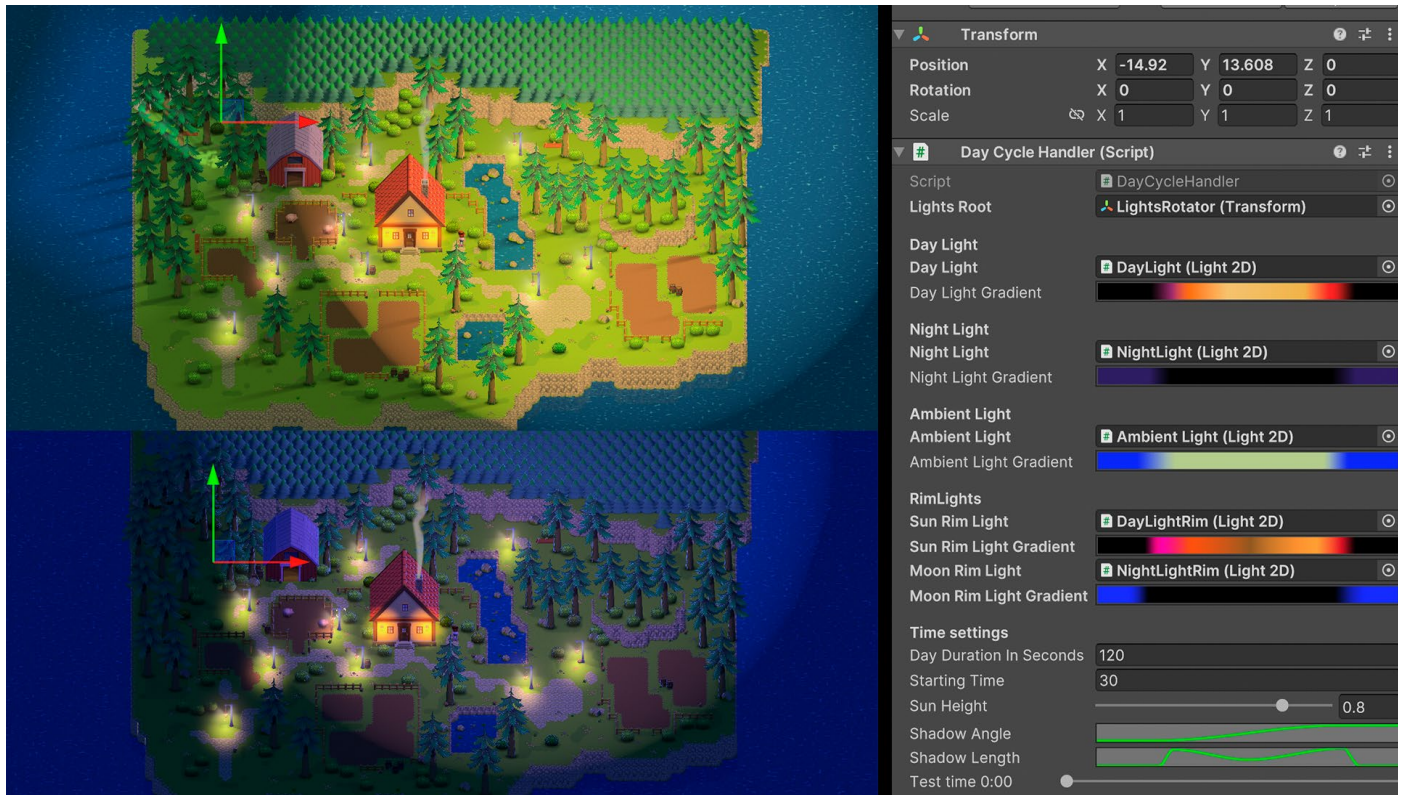
대형 스포트라이트는 해피 하비스트에서 주요 광원으로 사용됩니다. 메인 카메라에 연결되어 항상 화면에 표시되며 태양의 움직임을 시뮬레이션하는 스크립트를 사용하여 회전합니다.

프로젝트에 있는 **LightsRotator**라는 자식 게임 오브젝트에는 네 개의 광원이 연결되어 있습니다.

- **NightLight**: 달빛 방향을 시뮬레이션합니다.
- **DayLight**: 태양광 방향을 시뮬레이션합니다(NightLight과 반대 위치).
- **NightLightRim**: NightLight 달 방향 광원과 유사하지만 캐릭터와 프랍의 림 라이트에만 사용됩니다.
- **DayLightRim**: DayLight 태양 방향 광원과 유사하지만 캐릭터와 프랍의 림 라이트에만 사용됩니다.

이 광원의 움직임을 제어하는 스크립트는 하루 동안의 컬러 변화를 제어합니다. 그레디언트는 특정 시간에 각 광원의 컬러를 설정하는 데 사용되었습니다. **DayCycleHandle** 스크립트에서 이 그레디언트가 **DayCycleHandler** 게임 오브젝트에 연결되어 있는 것을 확인할 수 있습니다.

유한 그림자 효과를 제어하기 위한 추가 파라미터로는 **Shadow Angle**과 **Shadow Length**가 있습니다. 각 필드에서 애니메이션 커브는 하루 동안 그림자의 시계 방향 각도를 나타냅니다. 길이 파라미터는 특정 순간의 그림자 길이를 정의합니다. 예를 들어 태양이 지고 있을 때는 더 긴 그림자가 필요하고, 태양이 썬을 수직으로 비출 때는 더 짧은 그림자가 필요할 수 있습니다. Test time 슬라이더를 옮겨 Shadow Angle 및 Shadow Length 설정을 새로 고쳐야 할 수도 있습니다.



해피 하비스트의 대형 스폿 광원(각각 태양과 달에 사용)

프리폼 광원 조작

탑 다운 게임에서는 높고 큰 건물이 선명한 가장자리가 포함된 정밀한 그림자를 드리워야 합니다. 해피 하비스트에서는 건물의 그림자를 만들기 위해 프리폼 광원이 사용되었습니다. 이는 건물이 지면에 생성할 투영을 모방한 것으로, 2D에서는 텍스 정보가 없기 때문에 불가피하게 근사적으로 추정해야 합니다.



건물 그림자를 만들기 위해 사용되는 프리폼 광원은 네거티브 조명 기법을 사용하며, 여기에는 Multiply 블렌드 모드와 더 어두운 컬러가 사용됩니다.

샘플에서처럼 뚜렷한 그림자를 사용할 때의 과제는 이 그림자를 낮-밤 사이클에 맞게 작동시키는 방법입니다. 태양의 다양한 위치에 따라 그림자가 반응하도록 하기 위해, 서로 다른 레퍼런스 그림자 사이에서 프리폼 광원의 벡터 포인트를 보간하는 Light Interpolator 스크립트를 생성했습니다.

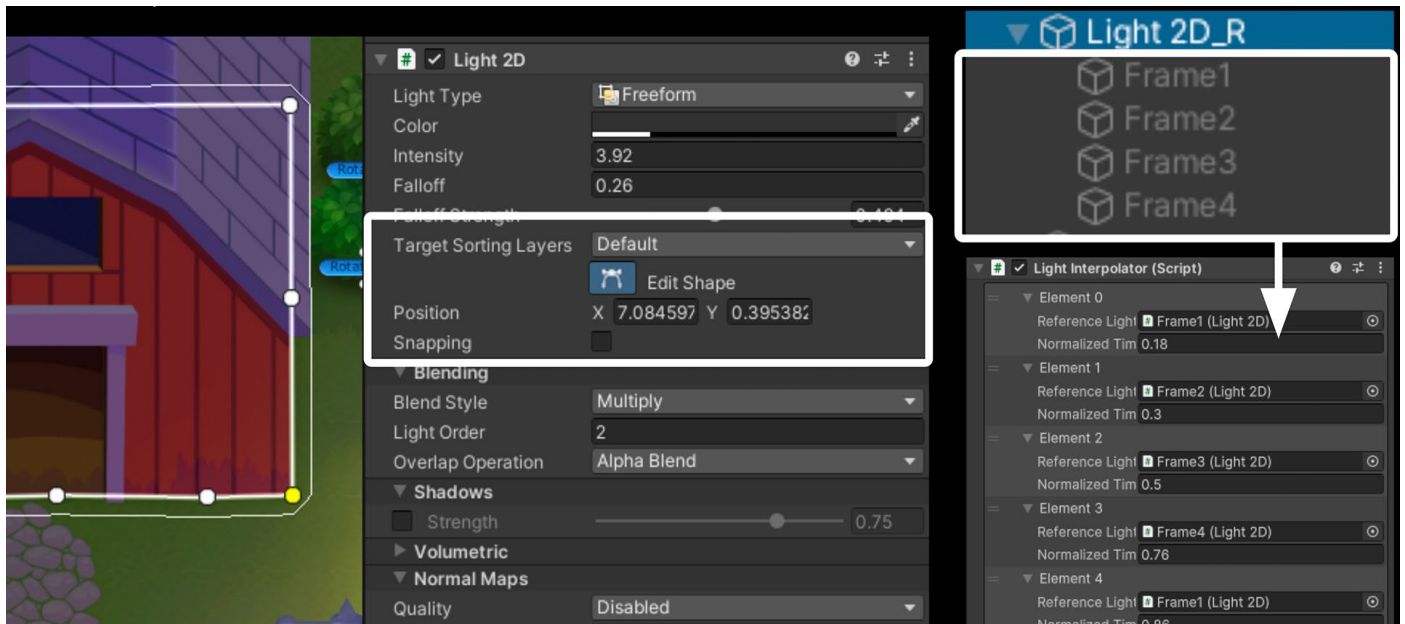
샘플의 계층 창에는 **Light_2D_Warehouse**라는 이름의 게임 오브젝트가 있습니다. 그에 연결된 네 개의 프리폼 광원이 있으며, 각각 태양이 위, 오른쪽, 아래, 왼쪽에 있을 때 건물이 투사할 그림자를 모방합니다. 이 스크립트는 API를 사용하여 서로 다른 벡터 포인트를 부드럽게 보간합니다.



하루 종일 움직이는 참고 아래의 더 뚜렷한 그림자

상단 그림자가 먼저 생성되고, 그 후 수정을 통해 하단 그림자와 건물의 양쪽 그림자가 만들어집니다. 각 그림자가 동일한 수의 포인트를 갖도록 하는 것이 중요하며, 그림자를 만들 때 포인트 사이의 전환이 자연스럽게 이루어지도록 해야 합니다.

그림자는 생성된 후 Light Interpolator 컴포넌트 스크립트에 추가되며, Normalized 파라미터는 낮 동안 각 그림자가 가지는 시간상의 가중치를 나타냅니다. 스크립트의 **Preview Time** 기능을 사용하면 에디터에서 보이는 모습을 미리 시각화할 수 있습니다.



샘플에서 사용된 Interpolator 스크립트는 프리폼 광원의 벡터 위치를 보강합니다.

Sprite Custom Lit 셰이더

오브젝트가 서로를 비추거나 환경을 비추도록 적용하는 것이 2D 광원 시스템의 표준 사용 방식이지만, 자체 조명 시스템을 만들려면 어떻게 해야 할까요? Shader Graph에서 **Sprite Custom Lit** 셰이더를 사용하면 2D 광원 시스템을 통해 생성된 라이트 맵을 읽고, 이 맵에 필요한 효과를 적용하여 스프라이트에서 사용할 수 있습니다.



젬 헌터 매치 샘플에서는 Sprite Custom Lit 셰이더를 사용하여 수백 개의 조각에 조명을 효율적으로 비춥니다.

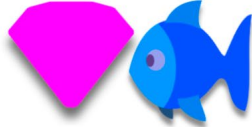
젬 헌터 매치 샘플에서는 게임 보드의 보석 조각에 조명이 각각 정밀하게 비춰집니다. 저희가 이러한 효과를 구현한 방법은 다음과 같습니다.

- 노멀 맵과 가상의 광원 위치를 사용한 조각별 커스텀 조명: Dot Product 노드를 사용하여 픽셀이 받아야 할 빛의 양을 계산할 수 있습니다.
- 2D Light Texture에 광원 정보 블렌딩: 림 라이트나 프레넬 효과를 위한 스폿 광원과, 월드 공간에서 실제 광원 역할을 하는 또 다른 스포트라이트를 함께 사용합니다.

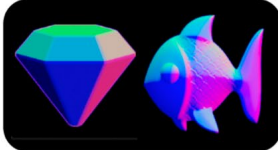
Sprites

Sprites and secondary textures

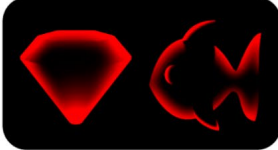
Sprite



Normal map



Mask map

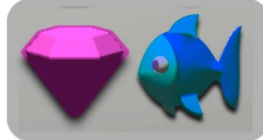


Lighting

Ambient light
Light2D_Global_Multiply



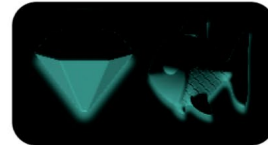
Light with normals
Light2D_Sport_Multiply...



Highlight (per piece)
TileShader (Custom Sprite Lit shader)

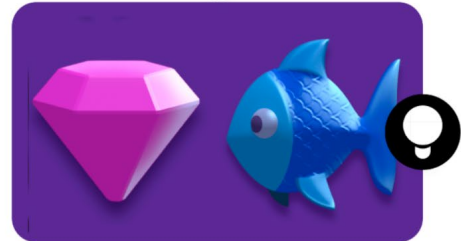


Light for rim lights
Light2D_SpotAdditiveMask...



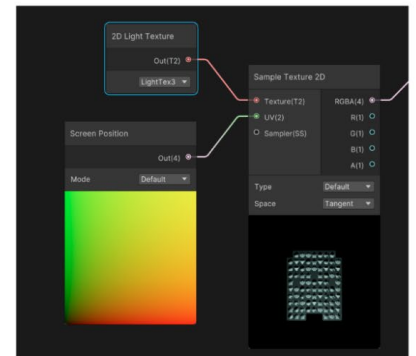
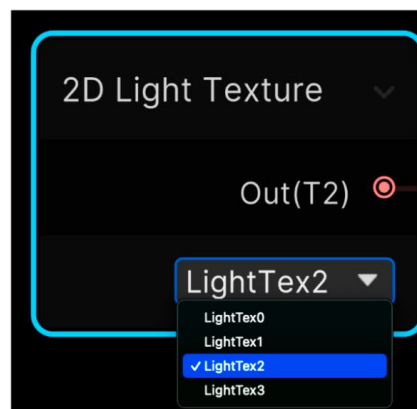
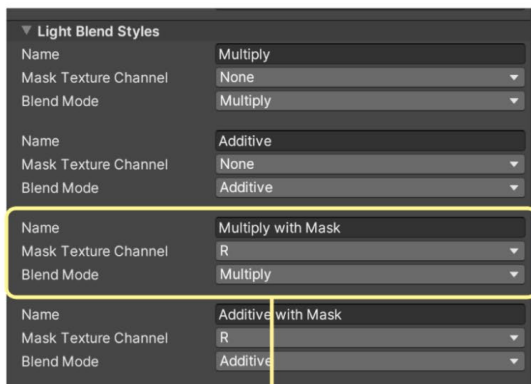
Result

Dynamic lighting that reacts to lights position and TileShader parameters



스프라이트 준비: 이 이미지에서는 썬 센터 매치에서 동적 조명 효과를 적용할 수 있도록 노멀 맵과 마스크 맵을 활용한 보드 조각 스프라이트를 어떻게 준비했는지를 확인할 수 있습니다.

2D Light Texture 노드에서 각 블렌딩 스타일에 해당하는 2D 라이트 텍스처를 자세히 살펴보고 이를 사용하여 오브젝트에 커스텀 2D 조명을 적용하세요.



다음 동영상에서 줌 렌터 매치의 커스텀 스프라이트 조명 및 기타 창의적인 아이디어를 자세히 알아보세요.



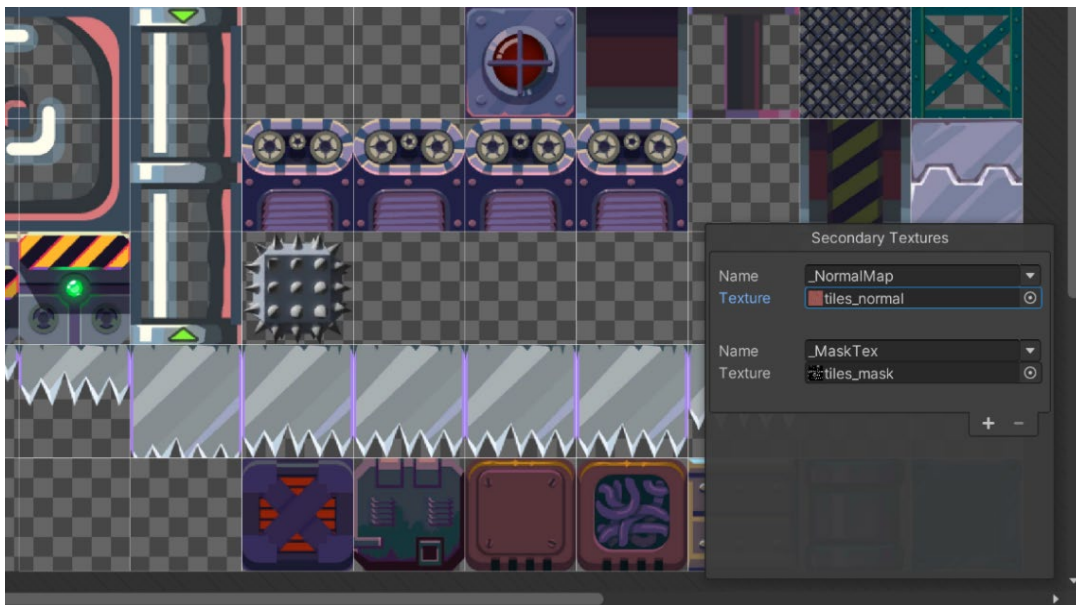
1부 및 2부

2D 렌더러에서 2D 광원 사용

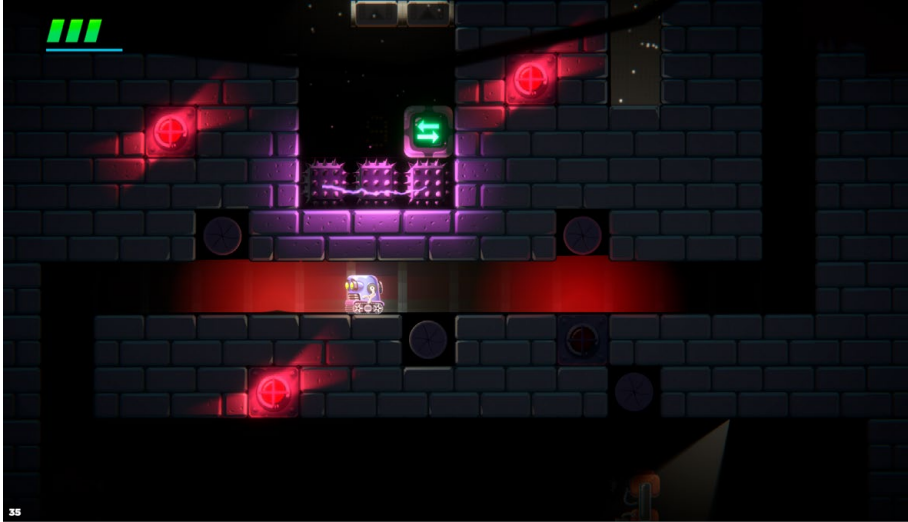
2D 광원은 스프라이트에서 노멀 맵 및 마스크 맵과 함께 사용할 수 있지만 여러 렌더러에서 사용할 수도 있습니다. 그 방법을 알아보겠습니다.

2D 타일맵

타일맵에 2D 광원을 사용하려면 스프라이트 에디터의 Secondary Textures 모듈을 사용하여 타일로 사용되는 스프라이트에 노멀 맵과 마스크 맵 텍스처를 할당합니다. 보조 텍스처는 2D 광원 시스템에서 자동으로 사용됩니다.



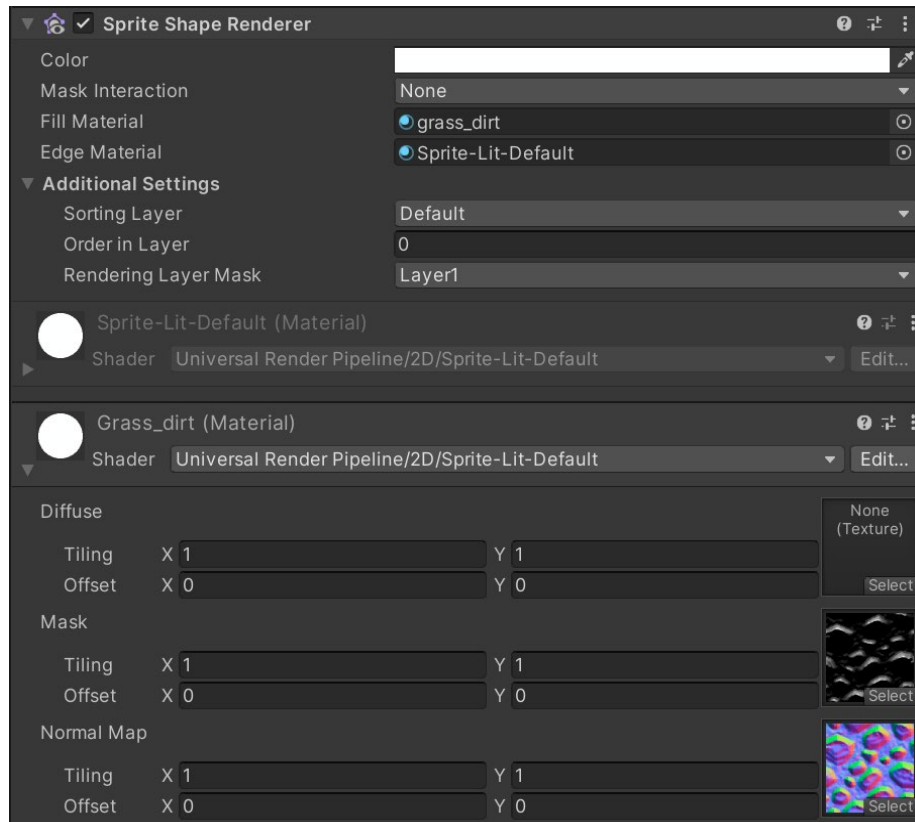
보조 맵을 포함한 타일맵



타일맵에 노멀 매핑 및 마스크 맵을 사용한 2D 광원

2D 스프라이트 셰이프

[Sprite Shape Renderer](#)는 오브젝트당 두 가지 다른 머티리얼을 사용합니다. 하나는 **Fill Material** 필드에 할당되고 다른 하나는 **Edge Material** 필드에 할당됩니다. Edge Material은 스프라이트 에셋의 보조 맵을 사용합니다. Fill Material의 경우 스프라이트 대신 **Wrap** 모드가 **Repeat**로 설정된 텍스처를 사용하여 타일링합니다. 보조 텍스처 설정을 위해 **Material Property Block**이 활성화된 새로운 Fill Material을 생성해야 합니다.





노멀 맵을 사용하는 스프라이트 셰이프 게임 오브젝트의 예

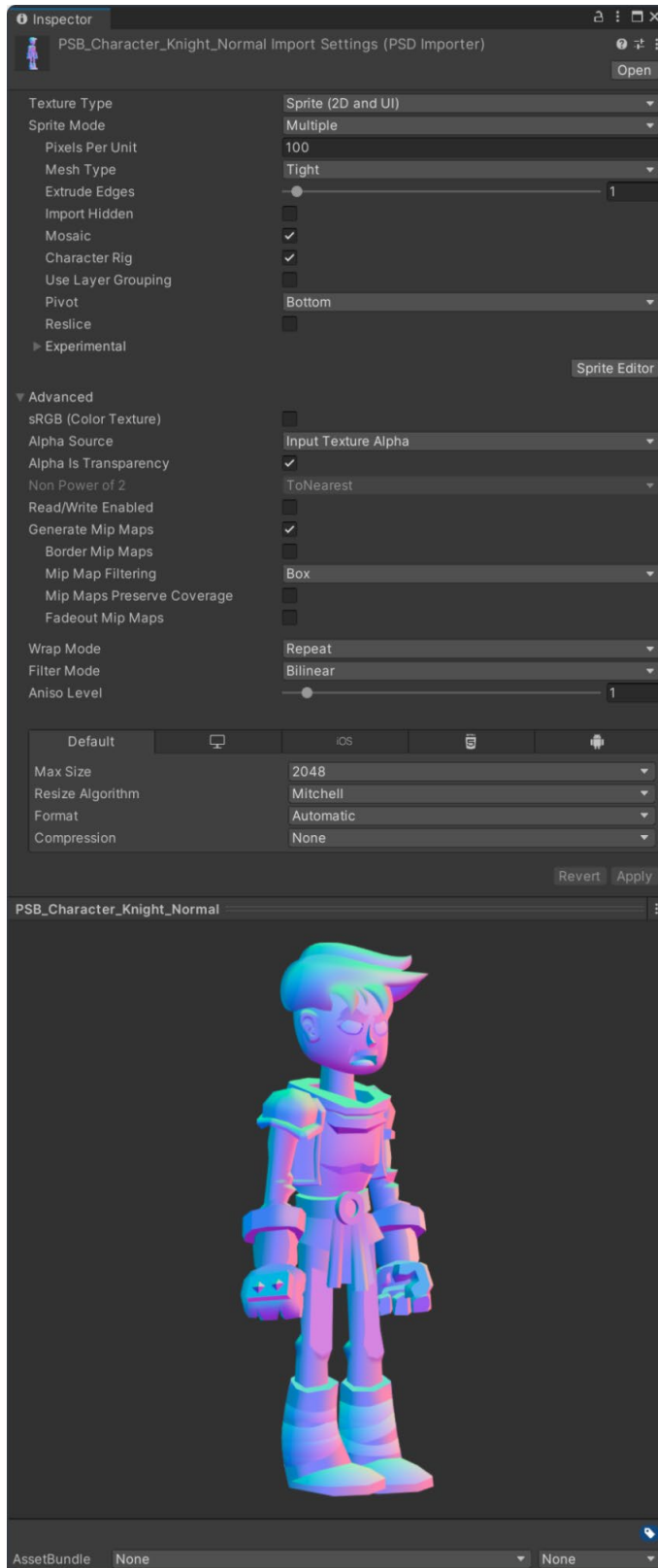
보조 텍스처가 있는 2D 애니메이션 캐릭터

2D PSD Importer에서 보조 텍스처를 설정하는 방식도 비슷하지만 노멀 맵과 관련하여 한 가지 작은 차이점이 있습니다.

2D PSD로 임포트된 캐릭터에 필요한 노멀 맵과 마스크 맵을 만드는 가장 빠른 방법은 기본 캐릭터의 .PSD 파일로 작업하는 것입니다.

애니메이션 캐릭터에 노멀 맵을 추가하는 방법은 다음과 같습니다.

1. 기본 캐릭터의 .PSD 파일을 복제합니다. 이름에 접미사(예: _normal)를 붙여 복제된 파일의 이름을 변경합니다.
2. 원하는 이미지 에디터에서 이 파일을 열고 각 레이어에 노멀 맵을 페인팅합니다. 파일을 저장합니다.
3. 노멀 맵이 있는 PSD를 Unity로 임포트하는 경우, Texture Type을 Sprite로 설정하고 Advanced 설정에서 sRGB(Color Texture) 옵션을 선택 해제해야 합니다. 노멀 맵에는 컬러 sRGB 데이터 없이 각도 값만 포함됩니다.
4. 이 PSD 파일을 기본 캐릭터의 보조 텍스처로 할당합니다. 마스크 맵을 만들려면 이 프로세스를 반복하여 복제된 파일에 또 다른 접미사를 부여하고 3단계를 건너뛸입니다.



애니메이션 캐릭터에 노멀 맵 추가

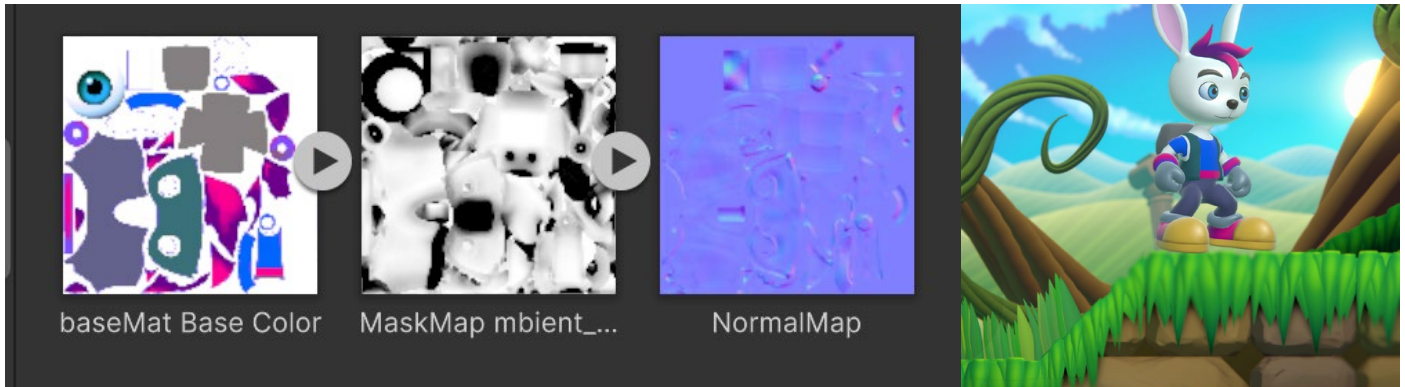
3D를 2D로 렌더링

Unity로 제작된 많은 게임은 게임플레이와 환경에서 2D 및 3D 요소를 믹스합니다. 예를 들면 3D 프로젝트에서 2D 그래픽스를 표현하는 시스템을 제작하거나 커스텀 렌더링으로 2D에 3D를 가져오는 등의 작업이 있습니다. 게임의 아트 주제, 제작 파이프라인 및 성능 목표에 맞게 2D와 3D를 믹스할 수 있습니다. 몇 가지 사용 사례는 다음과 같습니다.

- 아이소메트릭 게임을 예로 들면 다양한 각도에서 보이는 애니메이션 캐릭터는 3D로 만드는 것이 더 쉬울 수 있습니다.
- 애니메이션, 액세서리 또는 커스텀 요소의 재사용 또한 3D 캐릭터를 사용했을 때 더 쉬울 수 있습니다.
- 깊이감을 더 효과적으로 표현하기 위해 2D 캐릭터가 들어간 3D 배경을 만들 수 있습니다.

Unity 6.3 LTS는 3D 에셋을 2D 프로젝트로 가져오는 새로운 워크플로를 제공하며, 이 경우 다음과 같은 이점이 있습니다.

- 모든 2D 툴 활용 가능
- 3D 및 2D 에셋 전반에 2D 광원 시스템과 셰이더 머티리얼을 사용하여 더 일관적인 디자인 구현
- 2D 워크플로에 익숙한 개발자를 위한 자연스러운 3D 통합
- 스프라이트, 타일맵, 스프라이트 셰이프 및 3D 에셋 간의 손쉬운 정렬



버니 블리츠의 3D 캐릭터에 사용된 텍스처는 2D 스프라이트에 추가할 수 있는 보조 텍스처와 동일합니다.

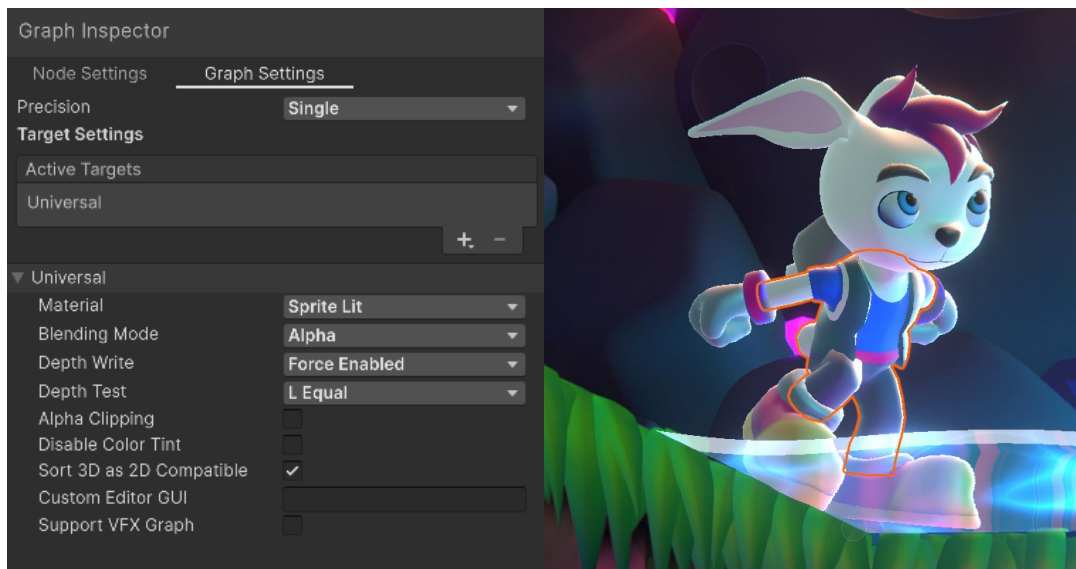
호환성

[MeshRenderer](#)와 [SkinnedMeshRenderer](#)는 URP의 2D 렌더러와 함께 2D 프로젝트에서 사용할 수 있는 호환 가능한 3D 렌더러입니다.

호환 가능한 머티리얼

Mesh2D-Lit-Default와 **Mesh2D-Unlit-Default**가 포함되어 있으며, 3D 에셋이 2D 광원을 통해 빛을 받도록 합니다. 이 머티리얼에 할당할 수 있는 텍스처는 기본 텍스처, 노멀 맵, 마스크 맵으로 2D 스프라이트의 텍스처와 동일합니다.

또한, **Sort 3D as 2D** 설정이 활성화된 Shader Graph 스프라이트 머티리얼은 3D 에셋에서 사용할 수 있으며 2D 조명을 받을 수도 있습니다.



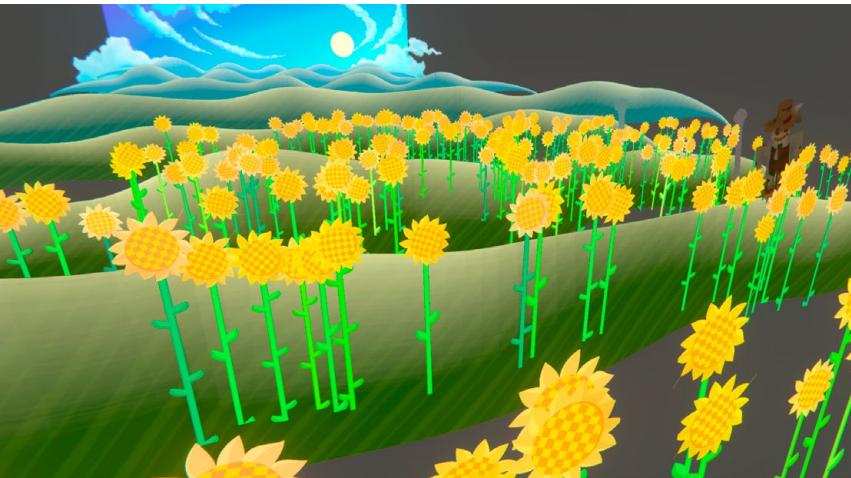
Sort 3D as 2D 모드가 활성화된 스프라이트 릿 셰이더



3D 및 2D 에셋 정렬

프로젝트에서 오브젝트 정렬을 계획하는 방법은 두 가지입니다.

1. Z축에서 요소를 정렬하고 원근 효과를 사용하려고 하시나요?
 - 3D 및 2D 오브젝트를 서로 다른 평면에 배치하기만 하면 올바르게 정렬됩니다.
 - **참고로**, 3D 오브젝트가 2D 광원을 통해 빛을 받으려면 어떤 정렬 레이어 광원에서 빛을 받을지를 나타내기 위해 **Sorting Group** 컴포넌트가 연결되어야 합니다.
 - 이 기법을 사용하면 덤스 버퍼에 값을 기록하여 덤스오브필드(피사계심도) 같은 포스트 프로세싱 효과를 사용할 수 있습니다.
2. 2D 프로젝트의 정렬 레이어 시스템을 사용하여 오브젝트를 정렬하려고 하시나요?
 - 3D 오브젝트를 2D 월드로 가져오고 Sorting Group 컴포넌트를 추가하세요. **3D as 2D** 모드에 체크하면 클리핑을 피하기 위해 오브젝트가 평면화되어 다른 스프라이트처럼 동작합니다.



왼쪽 이미지는 Z축에 배열된 요소가 있는 2D 씬에서 가져온 것입니다. 오른쪽 이미지는 해피 하비스트 샘플에서 가져온 것으로, 정렬 레이어를 사용해 버니 블리츠의 3D 토끼 캐릭터가 스프라이트 사이에 자연스럽게 블렌딩되도록 정렬하고 있습니다.



3D 및 2D 마스크

스프라이트 마스크는 3D와 2D 모두와 호환되며, 모든 오브젝트는 2D 렌더러를 통해 마스크될 수 있습니다.

Mesh Renderer 또는 Skinned Mesh Renderer 내의 **Mask Interaction** 옵션과 2D Tilemap, Sprite Shape, Sprite Renderer를 포함한 모든 2D 렌더러를 확인하세요. 이들은 마스크 내부, 마스크 외부 또는 마스크 역할을 하는 2D 렌더러에서만 보일 수 있습니다.

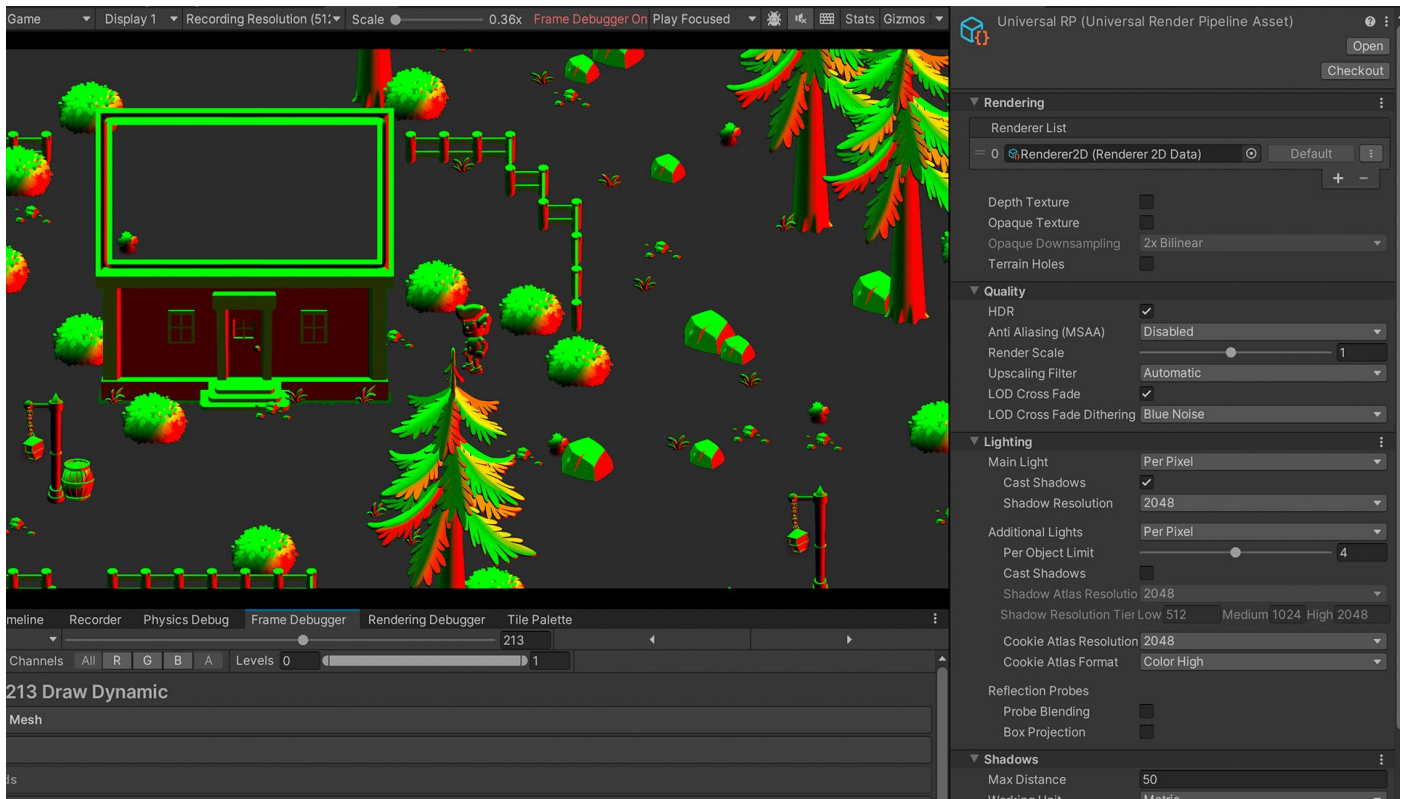
Unity 6.3 LTS의 새로운 API를 사용하여 런타임 시 모든 3D 또는 2D 오브젝트의 마스크 상호 작용을 변경할 수 있습니다.



토끼는 포털의 타원형 스프라이트 내부에서 마스크되어 스프라이트 내부에서만 보이게 됩니다.

유나이티드 바르셀로나 2025 [기조 연설](#) 및 [2D 세션](#)에서 새로운 2D/3D 워크플로에 대해 자세히 알아보세요.

2D 최적화 팁



프레임 디버거에서 볼 수 있는 해피 하비스트의 2D Renderer 설정. 프레임 렌더링의 각 단계별로 발생하는 상황을 알아보세요.

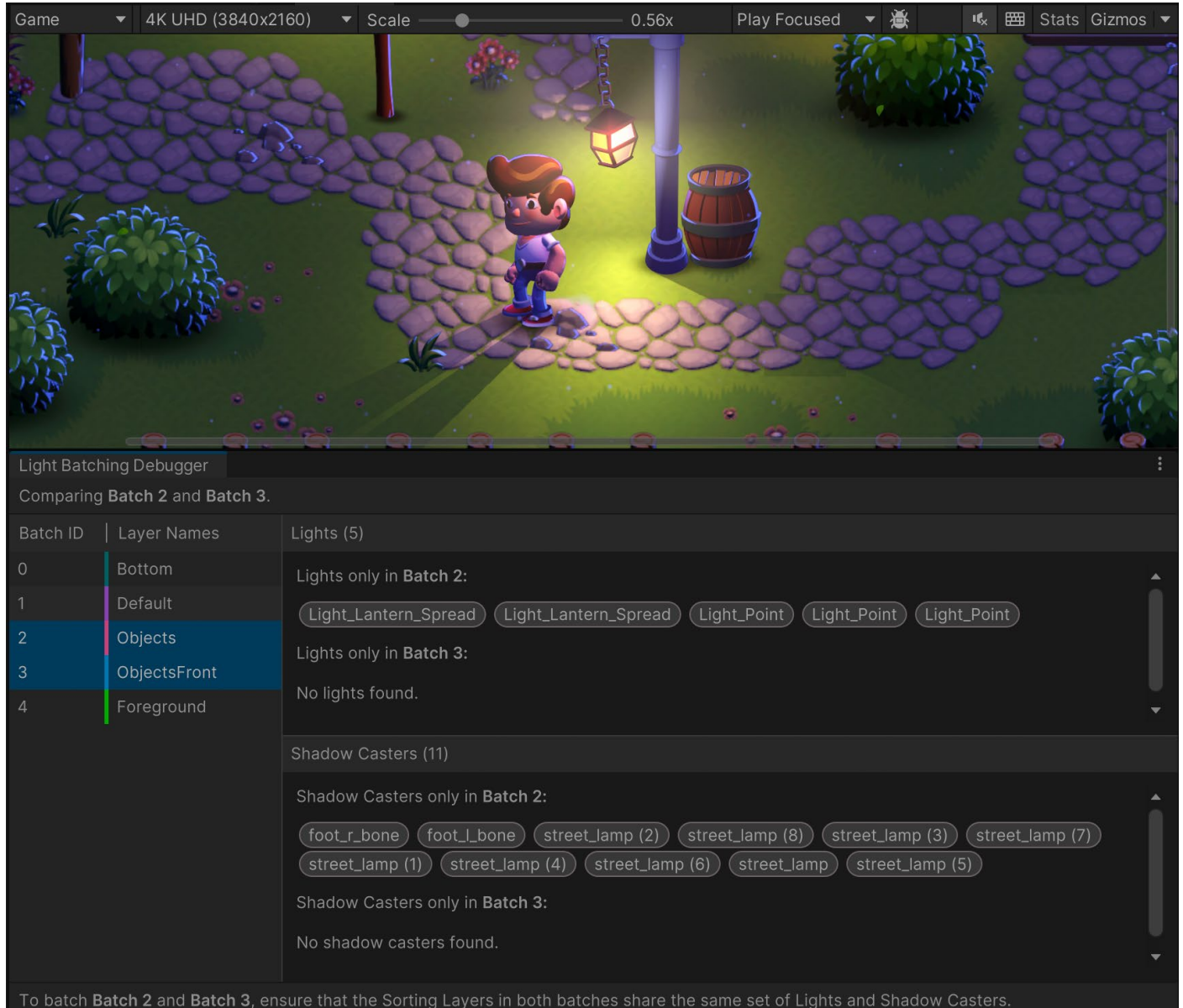


2D 광원 일반 최적화

특히 모바일 플랫폼을 타기팅할 경우 2D 조명을 사용할 때의 일반적인 우려는 게임에 광원을 추가하는 비용입니다. 항상 지원되는 최소 사양을 포함하는 실제 타겟 하드웨어에서 테스트하세요. 그리고 프로젝트 성능을 향상시킬 수 있도록 이러한 일반 최적화 팁을 시도해 보세요.

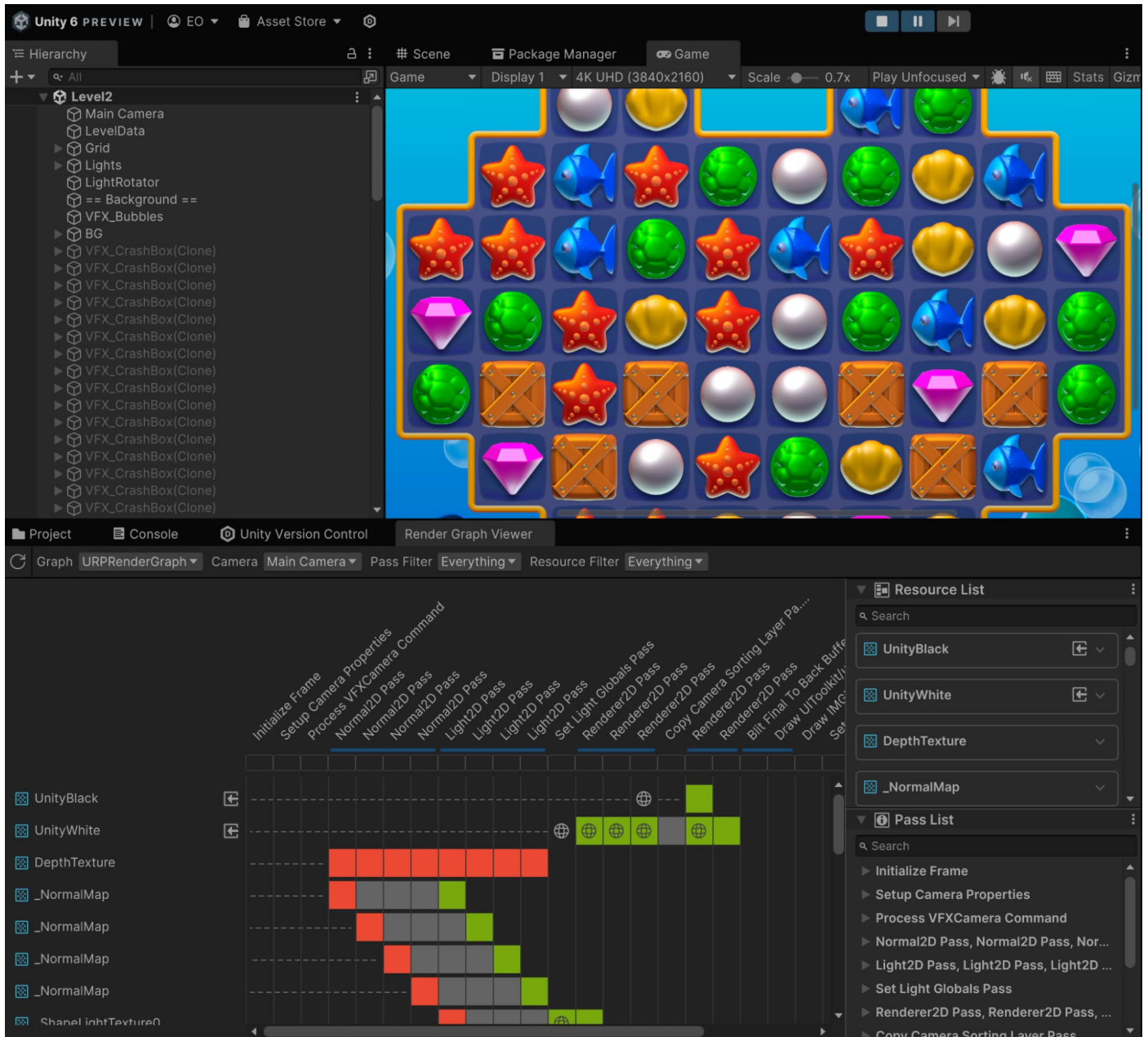
- 필 레이트를 최대한 낮게 유지하세요. 하나의 큰 광원을 사용할 경우 여러 개의 작은 광원보다 성능이 더 좋지 않을 수 있습니다.
- 광원은 배치 가능할 때 가장 성능이 좋습니다. 연속 레이어에서 조명 설정이 동일한 광원은 모두 함께 드로우될 수 있습니다.
- 렌더 스케일을 최대한 낮게 유지하세요. 렌더 스케일은 조명을 렌더링할 때 사용되는 텍스처 크기를 조정하며, 텍스처 크기가 작은 경우 렌더링할 픽셀이 적은 것입니다.
- 화면에서 그림자를 드리우는 광원의 수를 최소화하세요. 그림자를 드로우하도록 전환할 때 무시할 수 없는 수준의 성능 비용이 발생합니다.
- 화면에 표시되는 정렬 레이어와 다양한 블렌드 스타일의 수를 최소화하세요. 블렌드 스타일을 드로우하도록 전환할 때 성능 비용이 발생할 수 있습니다.
- 프로젝트의 요구 사항에 맞게 Max Light Render Textures 및 Max Shadow Render Textures의 수를 조정하세요. 숫자를 높이면 성능을 한계까지 향상시킬 수 있지만, 필요한 메모리도 증가합니다. 메모리와 렌더링 요구 사항에 맞는 적절한 수치를 찾아야 합니다.
- 노멀 맵을 사용하는 경우 광원의 노멀 조명 품질을 Accurate 대신 Fast로 설정하세요.
- 2D 그림자는 소수의 광원에만 사용하세요.

2D Light Batching Debugger 사용



- [2D Light Batching Debugger](#)를 사용하면 씬에서 타기팅하는 정렬 레이어에 따라 2D 광원과 Shadow Caster 2D 컴포넌트를 배치하는 방식을 시각화할 수 있습니다.
- 디버거는 인접한 배치를 비교하고 각 정렬 레이어를 타기팅하는 광원이나 캐스터를 하이라이트하며, Unity가 정렬 레이어를 배치할 수 있도록 추가하거나 제거해야 하는 항목을 표시합니다.
- 전반적으로 디버거는 광원과 그림자가 정렬 레이어에 미치는 영향을 더 원활하게 결정하여 게임 성능을 향상시키는 데 도움을 줄 수 있습니다.

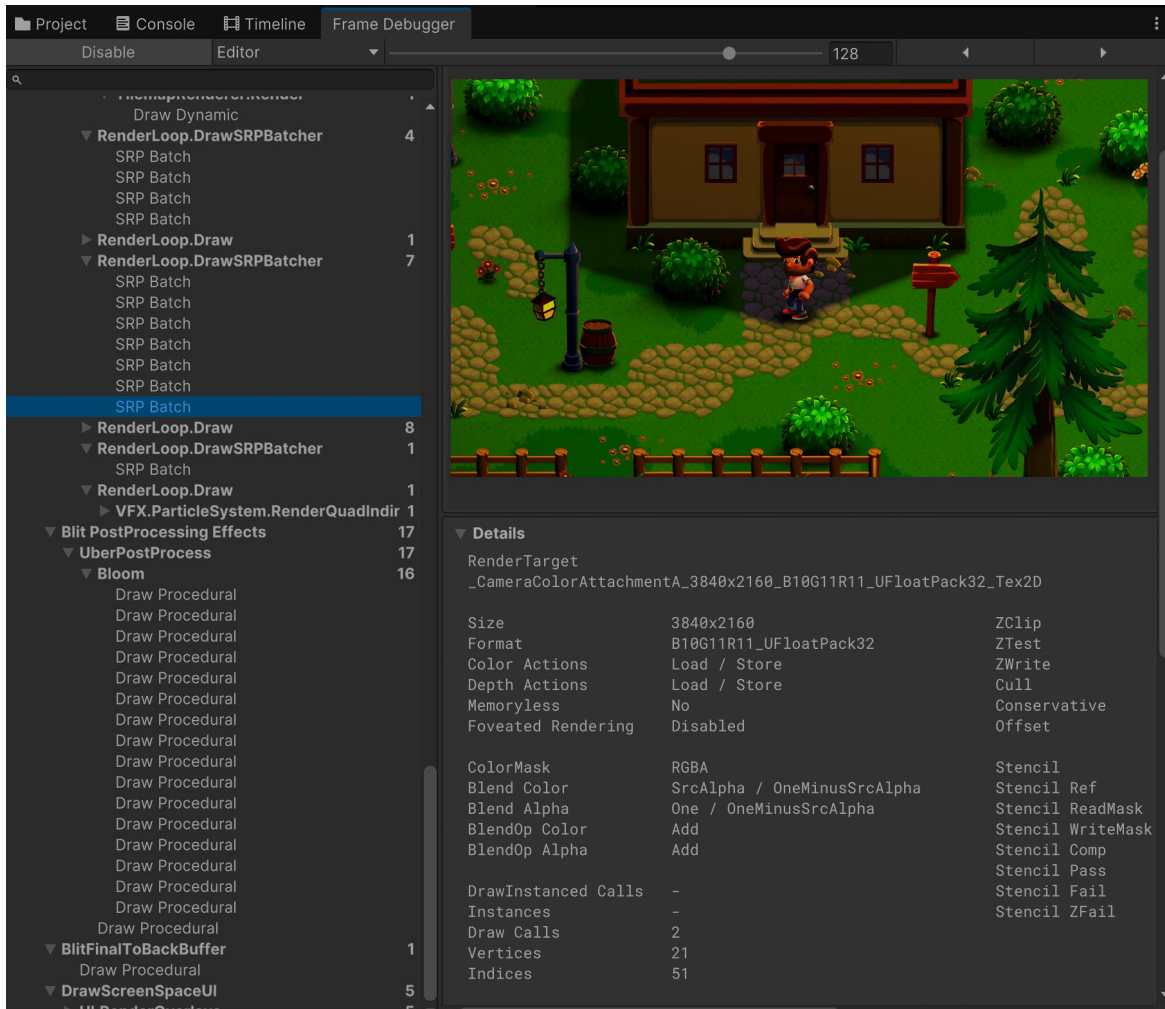
2D 렌더러와 Render Graph를 함께 사용



- Render Graph는 패스 수를 최소화하고 각 렌더 패스가 사용하는 메모리와 대역폭을 줄이도록 렌더 패스를 자동으로 최적화합니다.
- 프레임에서 사용하지 않는 추가 리소스를 할당하지 않습니다.
- 여러 렌더 패스를 단일 렌더 패스로 병합합니다.

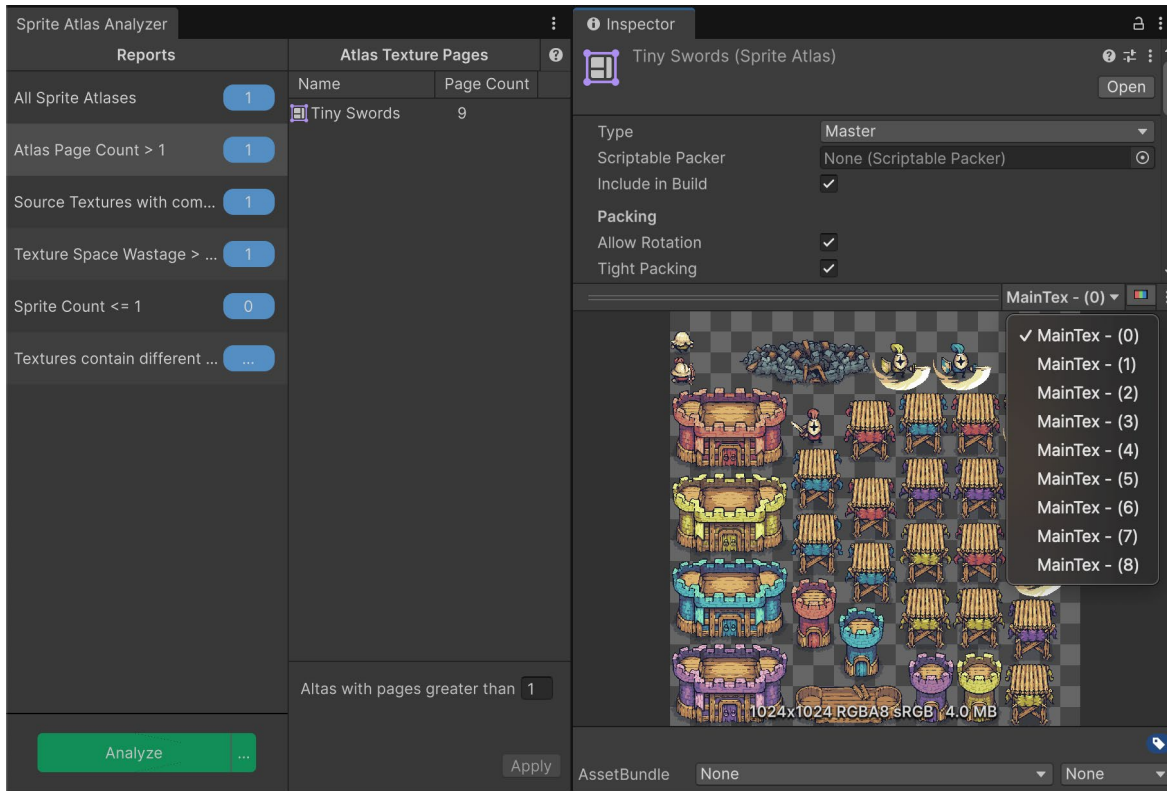


SRP Batcher 사용



- **SRP Batcher**는 이전에 CPU에 부하가 되었던 작업을 GPU 메모리로 옮겨 CPU 부하를 줄임으로써 더 원활한 게임플레이를 지원합니다.
- SRP Batcher는 조밀한 메시나 스킨이 있는 스프라이트의 경우 일반적으로 더 나은 성능을 제공합니다.
- **참고:** 전체 사각형 또는 버텍스가 매우 적은 스프라이트의 경우, 동적 배치(SRP Batcher 비활성화)를 사용하는 것이 더 좋습니다(API를 통해 오브젝트별로).

Sprite Atlas Analyzer 사용



Sprite Atlas Analyzer 창에서 Atlas Page Count 확인

Unity 6.3 LTS의 [Sprite Atlas Analyzer](#) 창은 스프라이트 아틀라스의 잠재적인 성능 문제를 파악하는 데 유용할 수 있습니다. 프로젝트 내 모든 스프라이트 아틀라스에 포함된 2D 텍스처 에셋의 자세한 정보를 살펴보세요.

사용하려면 패키지를 [설치](#)한 후 **Window > Analysis > Sprite Atlas Analyzer**로 이동하여 실행합니다. 창이 열리면 **Analyze** 버튼을 클릭하여 프로젝트의 스프라이트 아틀라스 내용을 분석할 수 있습니다.

Sprite Analyzer에는 6가지 [빌트인 보고서](#)가 포함되어 있습니다.

- All Sprite Atlases
- Atlas page count
- Source textures with compression in Sprite Atlas
- Texture Space Wastage
- Sprite count
- Textures contain different secondary texture count in Sprite Atlas

자세한 내용은 2D 톨에 대해 설명하는 유나이티 2025 세션의 [Sprite Atlas Analyzer 섹션](#)을 참조하세요.



타일맵 최적화



해피 하비스트: Rendering Debugger의 Overdraw Mode가 활성화되면 오른쪽에서 밝은 영역은 겹치는 픽셀을, 어두운 영역은 겹침이 적은 부분을 나타냅니다.

- 씬에 스프라이트가 많다면 타일맵을 사용하는 것이 좋습니다. 타일맵을 사용하면 오버헤드가 발생할 수 있는 많은 스프라이트 렌더러를 하나의 타일맵 렌더러로 맞바꿀 수 있습니다.
- 배치를 위해 레벨 콜라이더를 Static으로 표시하는 것이 좋습니다.
- 씬에서 함께 사용할 스프라이트는 스프라이트 아틀라스에 배치하는 것이 좋습니다.
- 스프라이트 에디터에서 Custom Outline 옵션을 사용하여 스프라이트의 메시지를 단순화할 수 있습니다.
- Cache Geometry 옵션을 활성화하여 스프라이트 셰이프 지오메트리를 캐시할 수 있습니다. 이 옵션은 Edit Spline 버튼이 켜져 있을 때 표시됩니다.

애니메이션 최적화

- 2D 애니메이션 성능을 향상하는 [Burst 패키지](#)를 설치하세요.
- 골격을 만들 때 뼈대와 스프라이트를 필요한 수보다 많이 만들지 마세요.
- 가급적 적은 수의 버텍스를 사용하여 골격 메시지를 단순화하세요.
- 애니메이터에서 Culling을 활성화하세요. 애니메이션 캐릭터가 화면에서 벗어나면 리소스를 절약하기 위해 이 옵션이 꺼집니다.
- 배경을 장식하는 새, 파리, 작은 동물처럼 작거나 때를 지어 등장하는 오브젝트나 캐릭터에는 골격 애니메이션을 사용하지 마세요. 대신 프레임 바이 프레임 애니메이션이나 셰이더 등의 다른 기법을 사용하세요.
- 프레임 바이 프레임 애니메이션을 사용할 때는 프레임 수와 텍스처 크기를 낮게 유지하세요.



VFX 및 포스트 프로세싱 효과

- 포스트 프로세싱 효과는 가급적 적게 사용하세요.
- 블룸, 색 수차, 컬러 그레이딩, 렌즈 왜곡, 비네트처럼 강렬한 효과는 가급적 적게 사용하세요.
- 필름 그레이인과 파니니 투영은 리소스를 더 많이 사용합니다.
- 블룸에는 고품질 필터링을 비활성화하세요.
- [오브젝트 풀링](#)을 사용하면 VFX 에셋의 인스턴스화와 파괴가 반복되는 대신, 게임 오브젝트의 활성화/비활성화가 교대로 이뤄집니다.
- Shader Graph의 정밀도는 가급적 Half로 설정하세요.

Unity에서 성능 툴을 사용하는 방법 알아보기

Unity 6 전자책은 Unity 프로젝트에 대한 고급 성능 최적화 기술을 개발하는 데 도움이 됩니다.



Unity에서의 애플리케이션 프로파일링, 메모리 관리, 전력 소비 최적화 등에 대한 팁 등 자세한 내용을 유니티 조직 내외부의 전문가들이 모여 70여 페이지가 넘는 가이드에 담았습니다.

[가이드 다운로드](#)



→ TECHNICAL E-BOOK

Optimize your game performance for mobile, XR, and the web in Unity

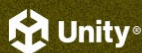


이 가이드를 참고하면 최대한 다양한 기기에서 성능이 보장된 애플리케이션을 제공하여 출시 후 성공의 기회를 극대화할 수 있습니다.

[가이드 다운로드](#)

→ TECHNICAL E-BOOK

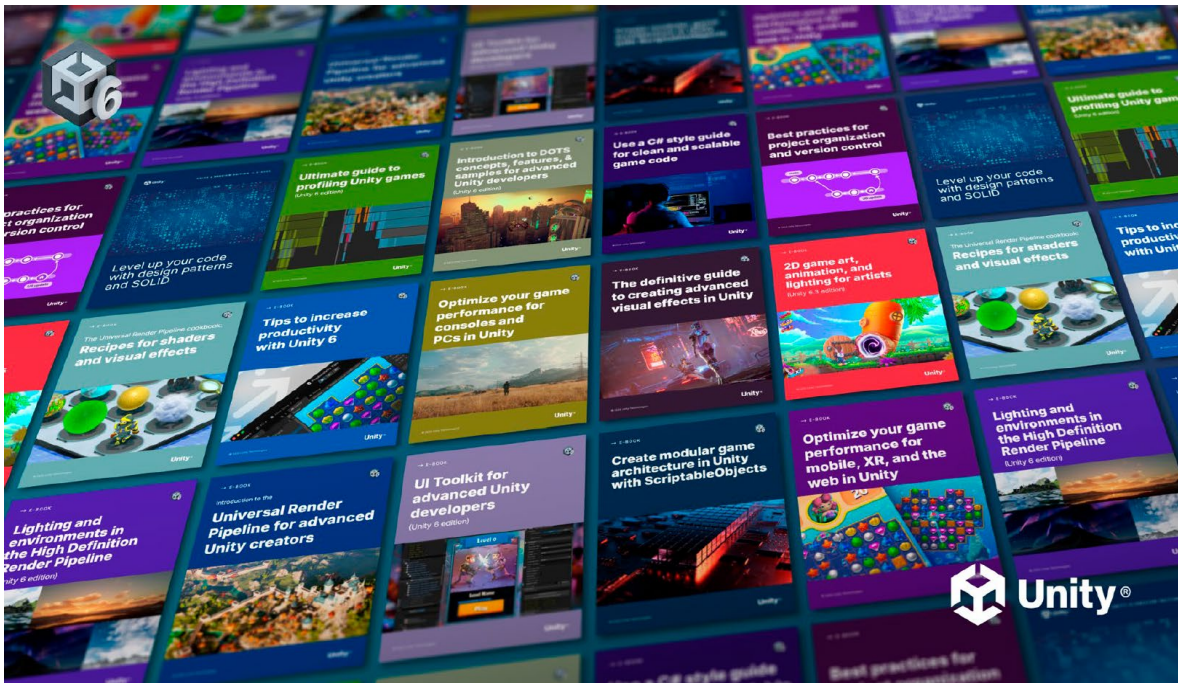
Optimize your game performance for consoles and PCs in Unity



코드 아키텍처와 아트 에셋을 최적화하는 방법을 알아보고, 타겟 하드웨어의 한계 내에서 원활하게 실행되는 게임을 출시하세요.

[가이드 다운로드](#)

맺는말



Unity 베스트 프랙티스 허브에서 숙련된 Unity 개발자 및 크리에이터를 위한 더 많은 전자책을 다운로드하실 수 있습니다. 업계 전문가, 유니티 엔지니어 및 테크니컬 아티스트가 제작한 30개 이상의 가이드를 읽어 보세요.

그리고 [Unity 블로그](#), [Unity Discussions](#), [Unity Learn](#) 및 [#unitytips](#) 해시태그를 통해 더 많은 팁과 뉴스를 찾아 보세요.

부록: 2D 시각 효과



왼쪽은 Unity 데모 [드래곤 크래셔 - 2D 샘플 프로젝트](#)의 이미지이고, 오른쪽은 Archanor VFX의 [Epic Toon FX](#) 이미지입니다. 두 패키지 모두 에셋 스토어를 통해 제공됩니다.

VFX(시각 효과)는 게임 비주얼의 대미를 장식하며 게이머의 플레이 경험에 지대한 영향을 미칩니다. VFX는 환경 오염 구역 및 회복 구역 같은 게임 이벤트를 전달하며, 플레이어가 훌륭한 액션을 보여 주면 크고 맹렬한 폭발로 절정에 이르는 액션 씬처럼 [그에 따른 시각적인 보상](#)을 제공합니다.



여러 가지 방법으로 Unity에서 2D 시각 효과를 만들 수 있습니다. 예를 들어, 불 효과에는 다음과 같은 방법을 사용할 수 있습니다.

- 프레임별로 불꽃을 애니메이션화합니다.
- [Shader Graph](#)에서 만든 셰이더로 스프라이트를 애니메이션화합니다.
- CPU에서 실행되는 [빌트인 파티클 시스템](#) 또는 GPU를 활용하여 수백만 개의 파티클을 생성할 수 있는 [VFX Graph](#)로 불꽃 파티클을 생성합니다. 두 시스템 모두 동일한 프로젝트에서 사용할 수 있으므로 특정 효과에 가장 적합한 것을 선택하면 됩니다.

Unity 2D 프로젝트에서 특수 효과를 만들기 전에 고려해야 할 몇 가지 핵심적인 내용이 있습니다.

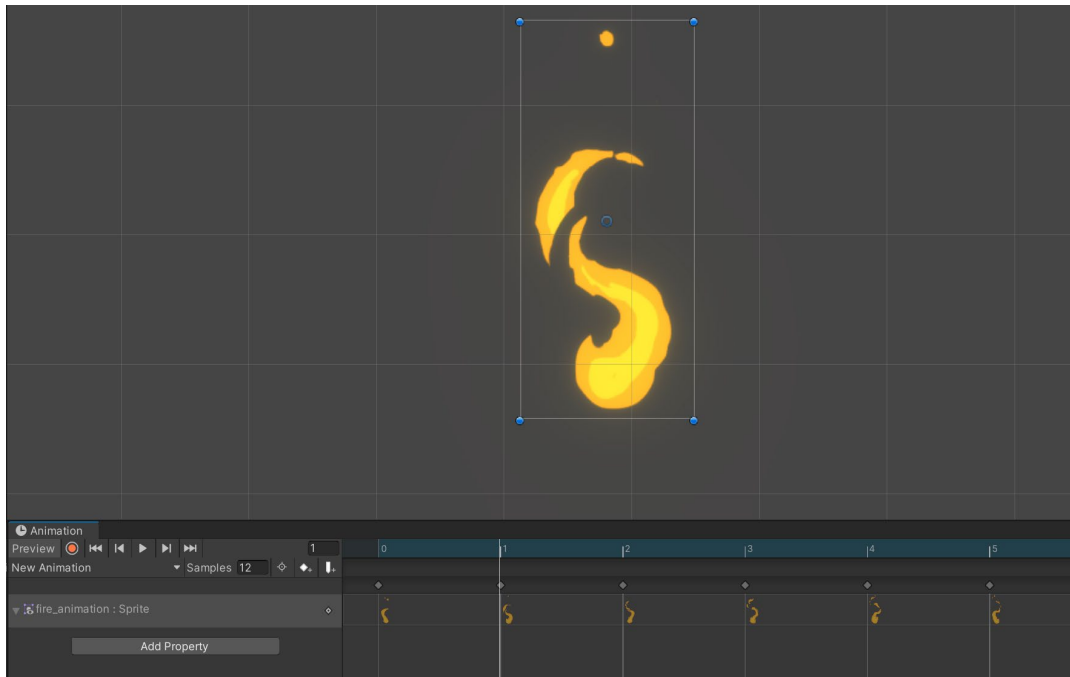
- **카메라:** 원근 카메라를 사용하면 효과가 3D 게임처럼 3차원 공간에서 발생할 수 있습니다. 직교 카메라를 사용하면 시스템의 렌더러를 설정하여 올바른 [정렬 레이어](#)를 사용해야 합니다. [여기](#)에서 카메라 시점에 대해 자세히 알아보세요.
- **포스트 프로세싱:** URP(유니버설 렌더 파이프라인)를 선택하면 2D 프로젝트에 [포스트 프로세싱 효과](#)를 적용할 수 있습니다. 예를 들면 URP [Volume](#) 프레임워크를 통해 파티클 효과에 블룸 효과를 추가할 수 있습니다.
- 효과는 다양한 방법으로 구현할 수 있습니다. 예를 들어, 효과를 나타내는 게임 오브젝트는 커스텀 셰이더가 있는 스프라이트나 여러 파티클 시스템처럼 다양한 시스템을 사용하는 여러 자식 게임 오브젝트를 가질 수 있습니다.
- **애니메이션 클립:** [애니메이션 클립](#)은 부드러운 스프라이트 기반 애니메이션 효과를 조정하는 데 도움이 될 수 있습니다. [Unity의 골격 애니메이션](#) 시스템을 필요한 경우에만 사용하면 스프라이트 애니메이션에서 스프라이트의 일부를 제어할 수도 있습니다.
- **Unity 에셋 스토어:** 프로젝트에 시각 효과를 추가할 때는 [Unity 에셋 스토어](#)를 먼저 살펴보세요. 미리 만들어진 다양한 2D 효과를 찾을 수 있습니다.

이 섹션에서는 VFX Graph에 중점을 두고 드래곤 크래셔, 해피 하비스트, 짐 헌터 매치 샘플의 특정 효과를 살펴보고 이러한 모든 방법을 소개합니다.



2D VFX를 위한 Unity 툴셋

프레임 바이 프레임 애니메이션



불꽃의 프레임 바이 프레임 애니메이션

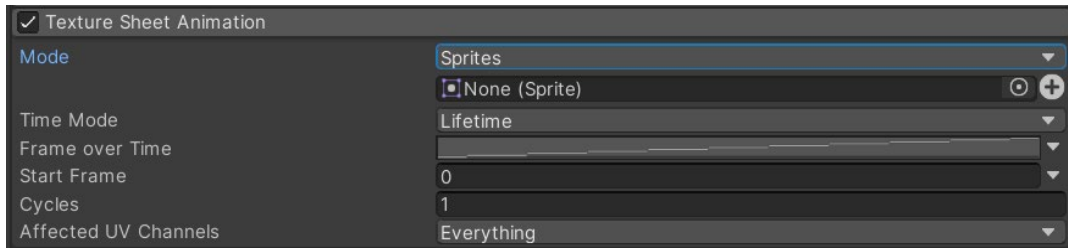
프레임 바이 프레임(또는 플립북) 애니메이션을 사용하면 Unity에서 효과를 빠르게 추가할 수 있습니다. 프레임을 스프라이트로 임포트한 후 Animation 창에서 Sprite 프로퍼티를 시간에 따라 애니메이션화할 수 있습니다. 모든 프레임 스프라이트를 선택하고 씬에 드래그하면 자동으로 애니메이션화되기도 합니다.

프레임 임포트 작업은 빠르게 진행되지만 애니메이션화할 각 프레임을 그리는 작업은 그렇지 않으며, 이 프로세스에는 주어진 것보다 훨씬 더 많은 시간과 기술이 필요할 수 있습니다. 그 대신 시간을 단축하기 위해 다른 앱에서 VFX를 프레임으로 익스포트할 수 있습니다.

또 한 가지 팁은 프로젝트 뷰에서 모든 애니메이션 프레임 에셋을 선택하고 계층 구조 뷰 또는 씬으로 드래그하는 것입니다. 그러면 이전에 선택한 이미지로 구성된 시퀀스를 사용하는 애니메이션이 포함된 새 게임 오브젝트가 생성됩니다.

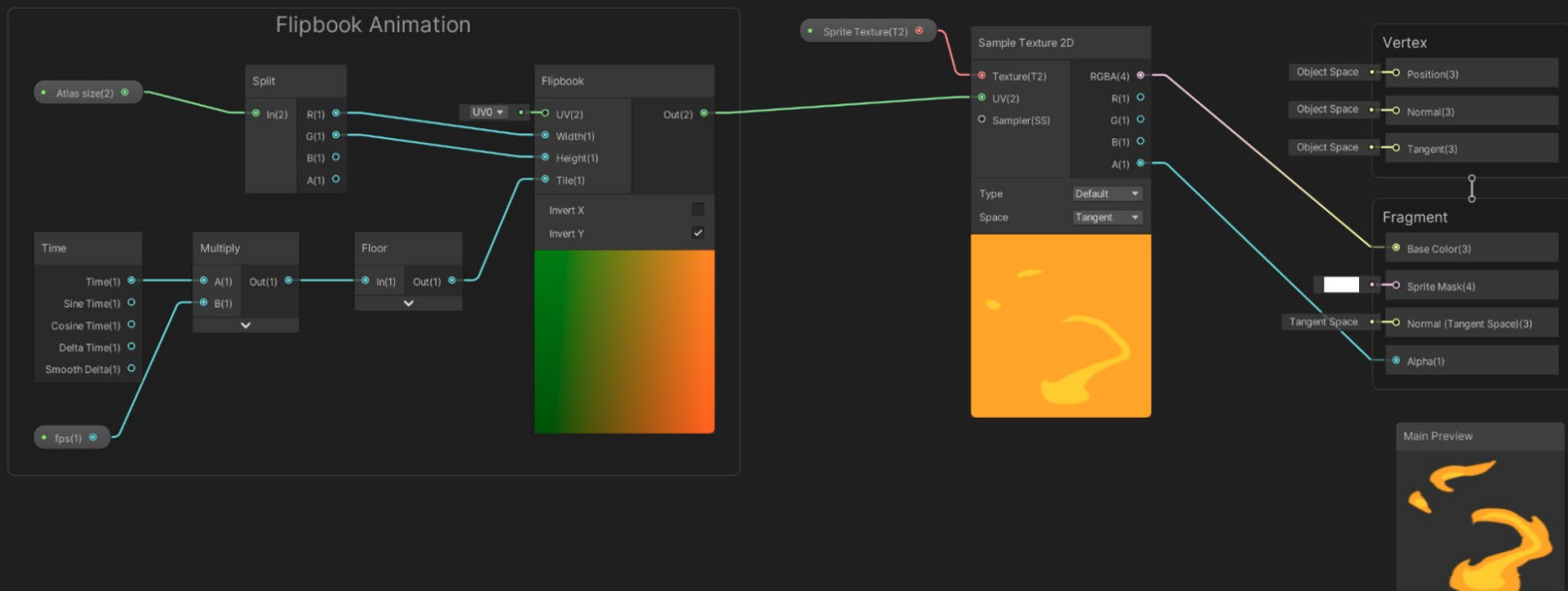
플립북 애니메이션 기법은 성능 면에서 매우 뛰어나므로 화면에서 효과 인스턴스를 많이 사용해야 하는 경우 활용하세요. 고해상도에서 많은 수의 프레임을 처리하려면 더 많은 메모리가 필요하다는 점에는 주의해야 합니다.

프레임 바이 프레임 애니메이션을 [파티클 시스템](#)에서 텍스처 시트 애니메이션으로 참조하여 사용할 수도 있습니다.



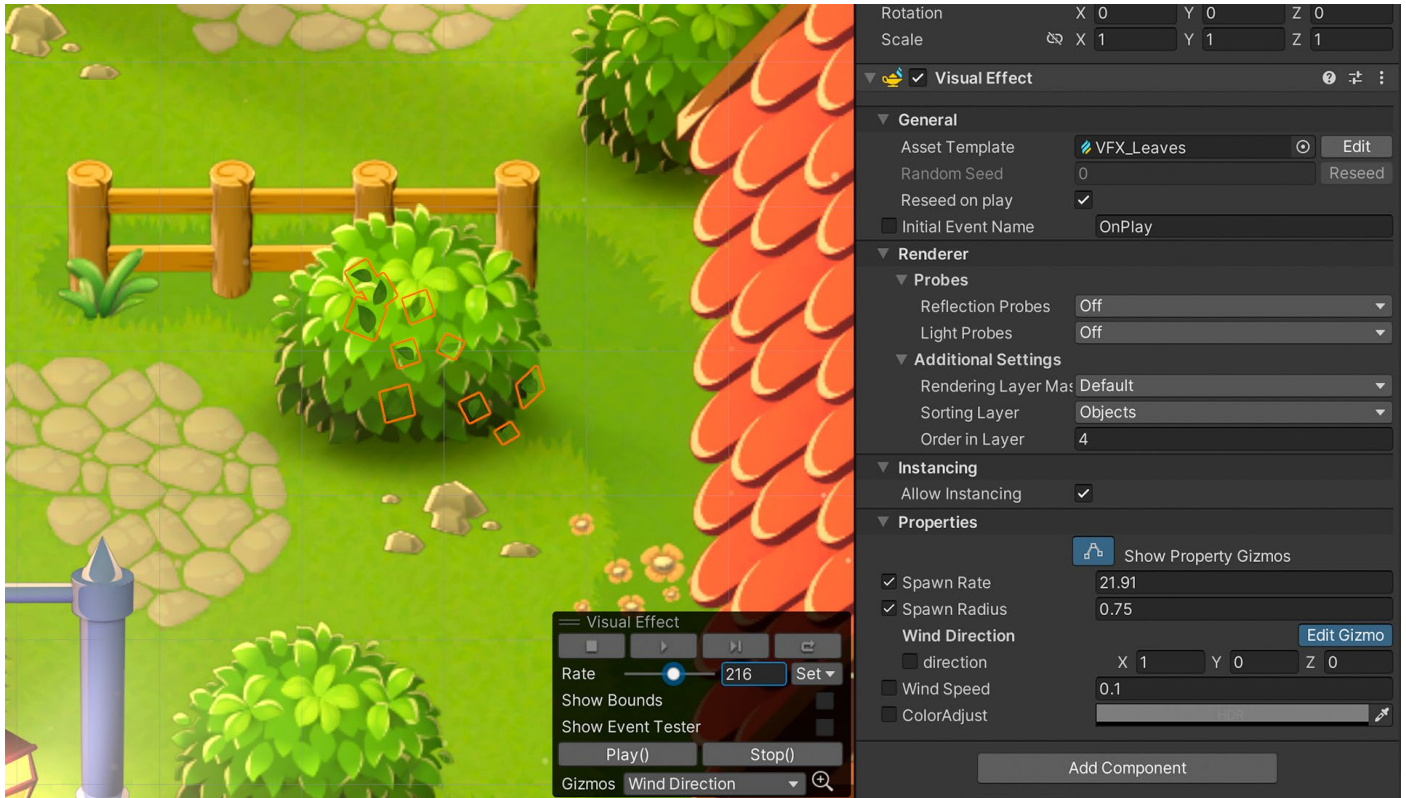
파티클 시스템에서 프레임 바이 프레임 애니메이션 설정

텍스처의 UV 위치를 시간 경과에 따라 애니메이션화하여 플립북 애니메이션을 Shader Graph에서 사용할 수도 있으며, 이렇게 하면 프레임이 이동하는 듯한 효과가 발생합니다.



Shader Graph의 플립북 애니메이션

2D용 VFX Graph



해피 하비스트에서는 캐릭터가 지나갈 때 덤불에서 잎이 떨어지는 효과를 만들기 위해 VFX Graph가 사용됩니다.

테크니컬 아티스트와 VFX 아티스트는 노드 기반 에디터인 **VFX Graph**를 사용하여 단순하고 평범한 파티클 동작부터 파티클, 선, 리본, 트레일 등이 포함된 복잡한 시뮬레이션까지, GPU 가속이 적용된 동적인 파티클 효과를 디자인할 수 있습니다.

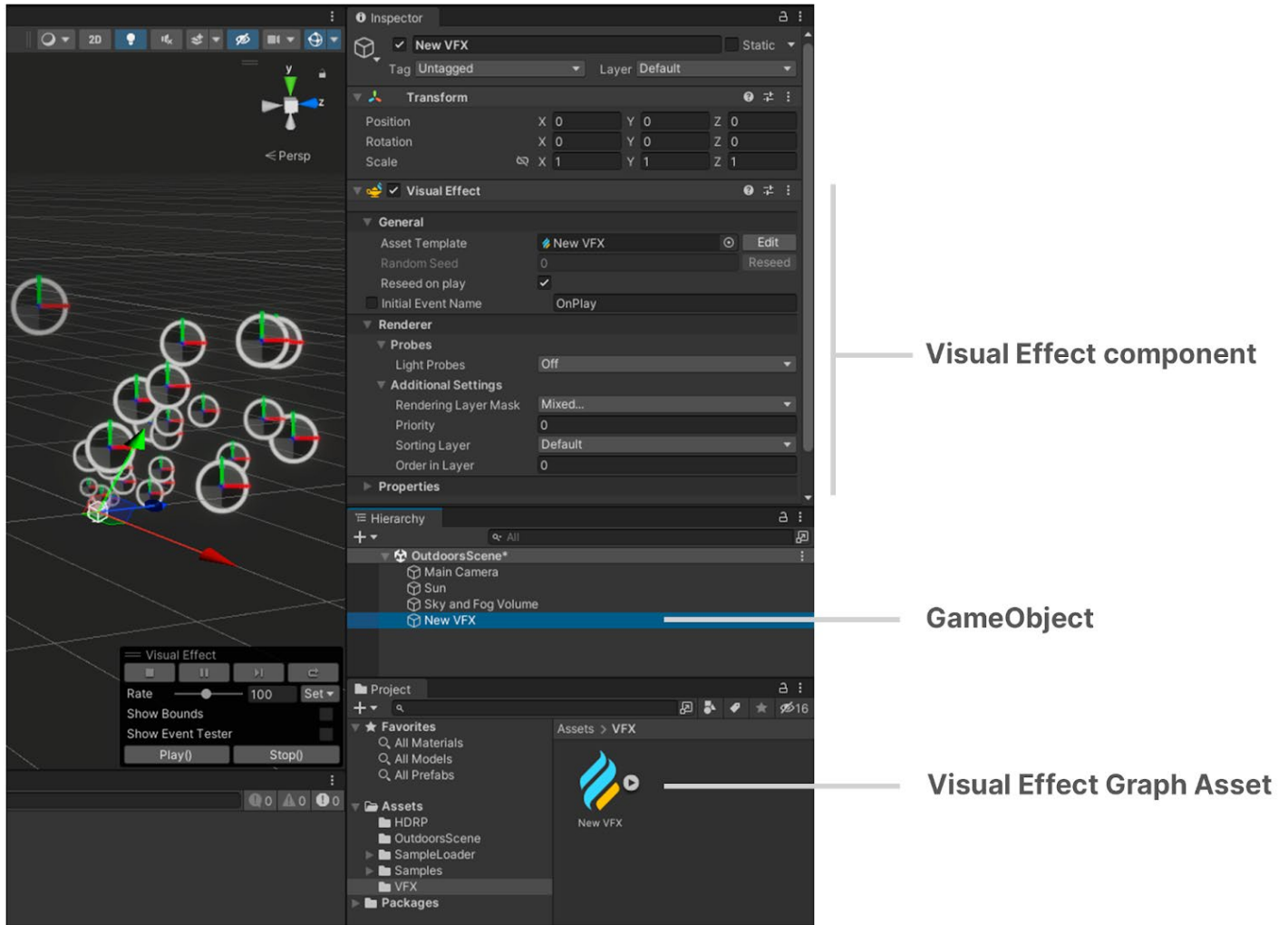
VFX Graph 시스템은 HDRP와 URP 모두에서 작동하며, 2022 LTS 버전 이상의 Unity에서 Shader Graph를 사용하는 2D 렌더러와 호환됩니다.

프로젝트에서 새 그래프를 만들려면 VFX Graph [기술 자료](#)를 참조할 것을 권장합니다. VFX Graph의 기본 개요를 살펴보면 2D 및 3D 게임에서 효과를 생성하는 데 VFX Graph가 어떤 역할을 하는지 더 손쉽게 이해할 수 있습니다.

VFX Graph의 모든 시각 효과는 다음 두 부분으로 구성됩니다.

- 씬의 게임 오브젝트에 연결된 **VFX(Visual Effect) 컴포넌트**
- 프로젝트 레벨에 존재하는 **VFX(Visual Effect) 그래프 에셋**

Unity는 각 VFX Graph를 Assets 폴더에 저장하므로 각 에셋을 씬의 Visual Effect 컴포넌트에 연결해야 합니다. 여러 게임 오브젝트가 런타임에서 같은 그래프 하나를 참조할 수 있다는 점을 기억하세요.

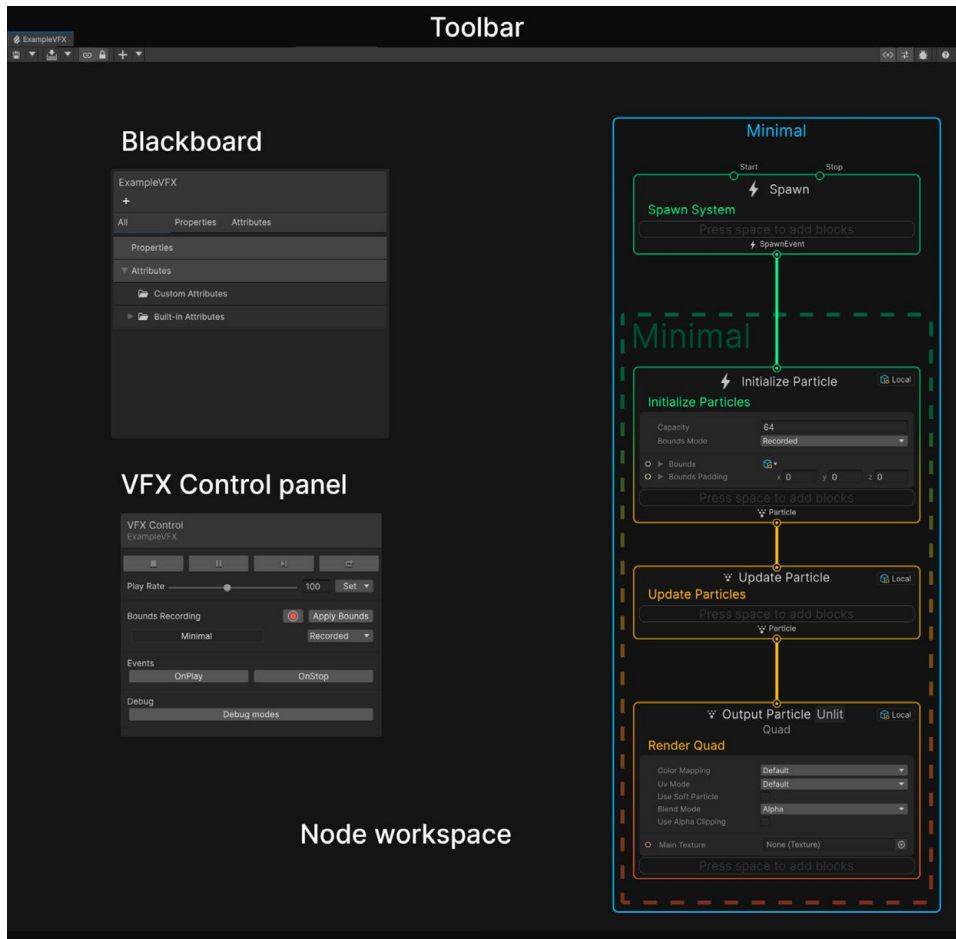


VFX Graph Asset과 Visual Effect 컴포넌트

VFX Graph 창

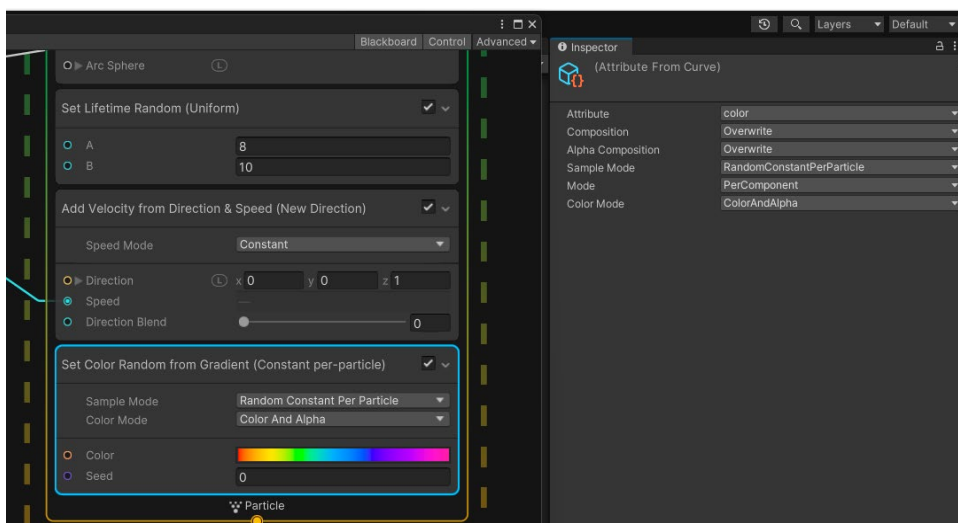
창의 레이아웃에 포함되는 내용은 다음과 같습니다.

- **Toolbar:** 전역 설정 액세스 및 여러 패널에 대한 토글 제공
- **Node workspace:** VFX Graph 구성 및 편집
- **Blackboard:** 그래프 전체에서 재사용할 수 있는 속성 및 프로퍼티 관리
- **VFX Control 패널:** 연결된 게임 오브젝트에서 재생 내용 수정



VFX Graph 창

에디터 레이아웃에 인스펙터를 위한 공간을 남겨 두어야 합니다. 그래프의 일부를 선택하면 파티션 옵션, 렌더 상태 같은 특정 파라미터가 나타날 수 있기 때문입니다.



인스펙터를 사용하여 특정 파라미터를 변경할 수 있습니다.

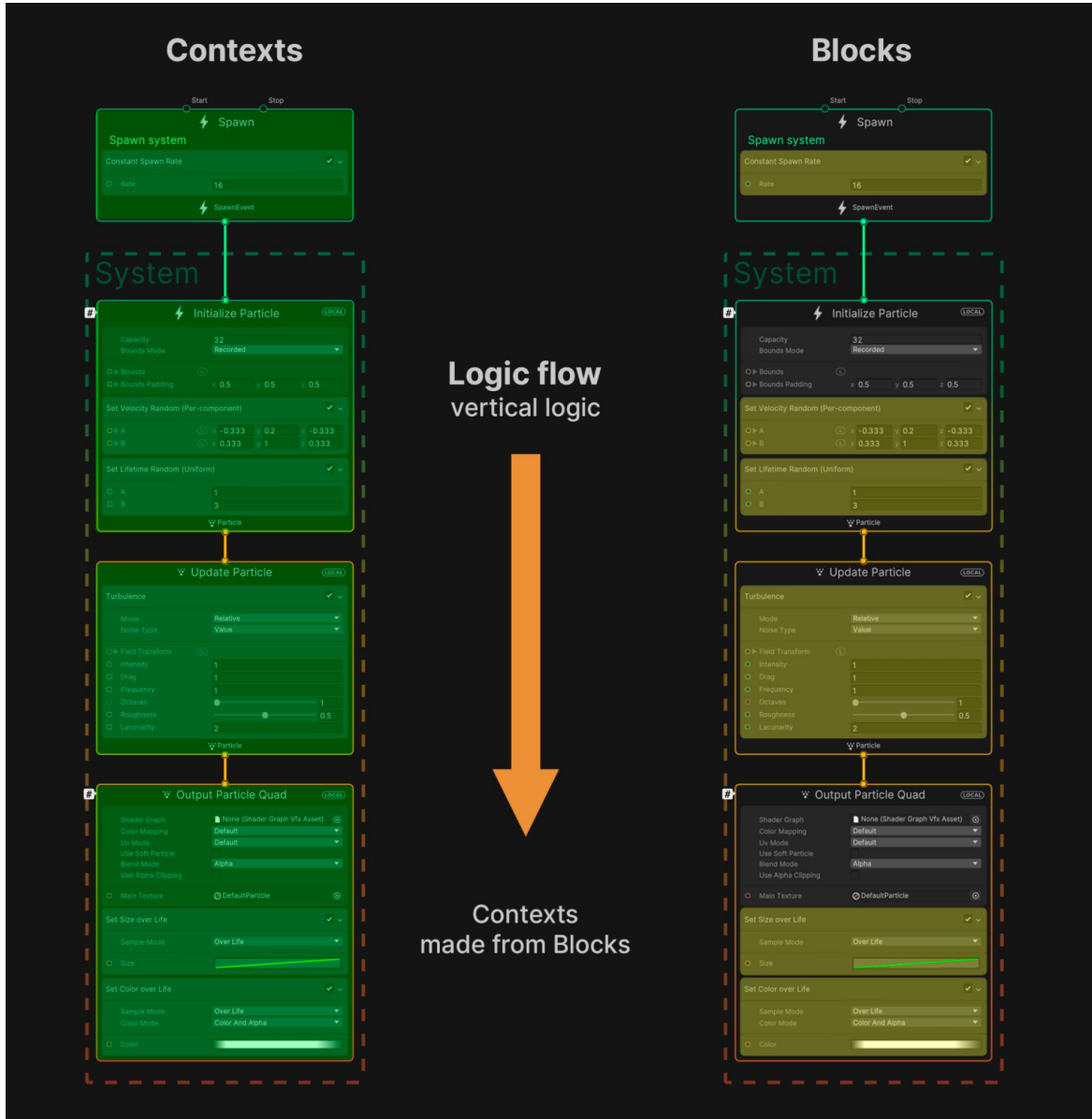


그래프 로직

시각 효과는 창의 작업 공간에 포함된 노드를 연결해 만들어야 합니다. VFX Graph는 Shader Graph 등의 다른 **노드 기반 툴**과 유사한 인터페이스를 사용합니다(이 챕터의 뒷부분에서 설명).

VFX Graph에는 **시스템**이라는 세로형 스택이 하나 이상 들어갑니다. 시스템은 그래프의 독립된 부분을 정의하고 여러 **컨텍스트**를 포함합니다. 시스템은 해당 시스템을 구성하는 컨텍스트를 묶는 점선으로 표시되어 있습니다.

각 컨텍스트는 개별 **블록**으로 구성되며, 블록에서 파티클과 메시의 **속성**(크기, 컬러, 속도 등)을 설정할 수 있습니다. 하나의 그래프 내에서 여러 시스템이 연동되며 최종적인 시각 효과를 만들어 냅니다.



그래프의 세로형 로직은 위에서 아래로 진행됩니다.



VFX Graph의 [Output 컨텍스트](#)에서는 효과가 어떻게 렌더링될지를 정의합니다. [Output Particle Quad](#)와 같은 메시용 Output 컨텍스트에서는 Shader Graph를 추가할 수 있는 옵션을 제공합니다. 조명 모델은 Shader Graph에 맞게 변경되므로 Lit이든 Unlit Output이든 상관없습니다. 아래에 나열된 기본 셰이더를 사용하거나 자체 Shader Graph 기반 셰이더를 사용할 수 있습니다.

2D에서 VFX Graph를 사용하기 위한 빠른 팁

URP의 2D 프로젝트에서는 [세 가지 스프라이트 기반](#) Shader Graph 중 하나를 사용할 수 있습니다.

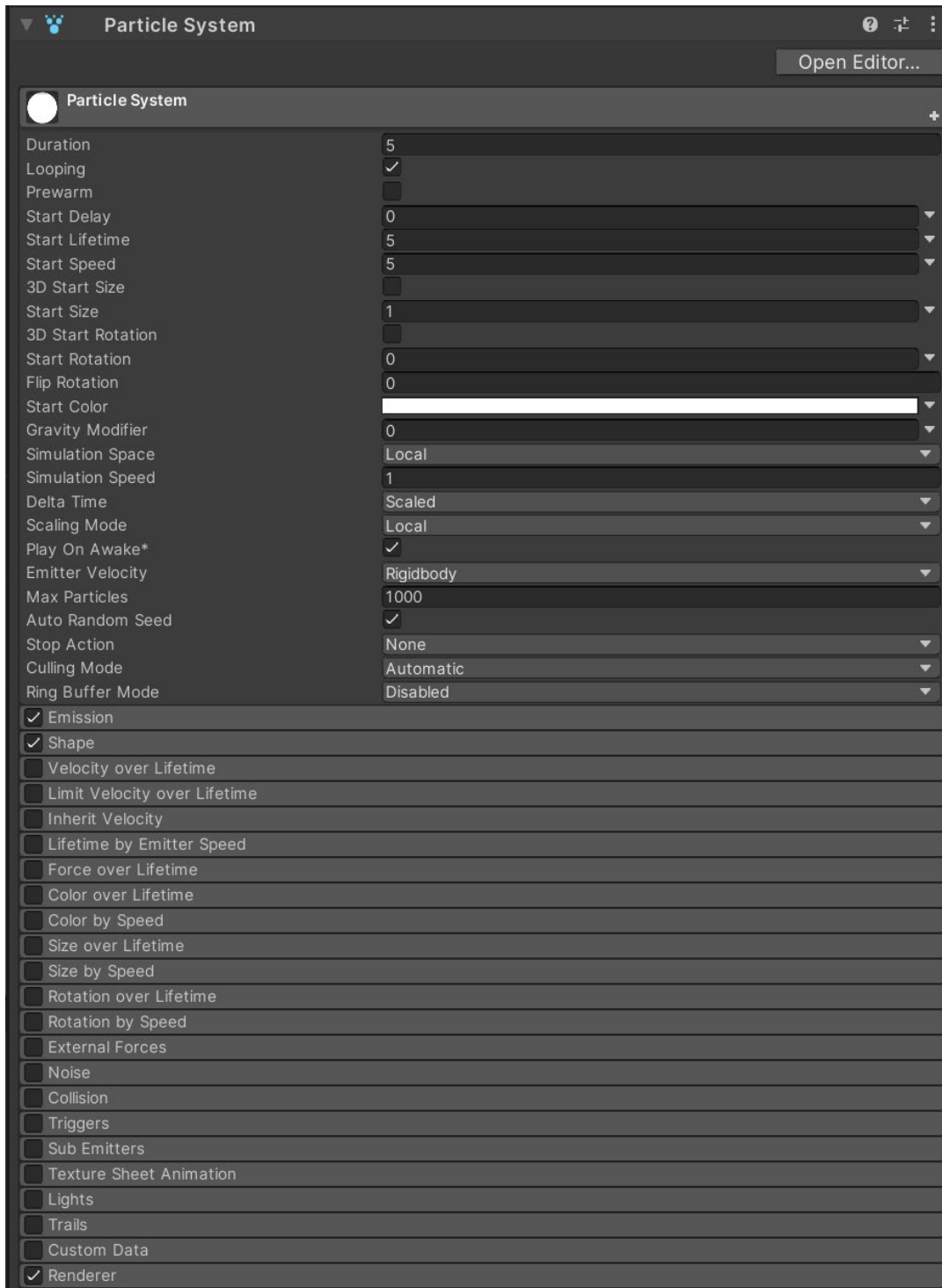
- Lit: 2D 광원 시스템에서 빛을 받음
- Unlit: 2D 조명 시스템의 영향을 받지 않음
- [Custom](#): 커스텀 2D 조명 모델을 구현하려는 경우

Shader Graph의 SRP 타겟에서 **Support VFX Graph** 옵션을 확인하세요.

Shader Graph에서 노출된 프로퍼티는 VFX Graph에 표시됩니다. 필요한 경우 이를 오버라이드할 수 있습니다. 예를 들어, VFX Graph 파티클 시스템이 시간에 따라 셰이더의 컬러를 변경하도록 제어하려는 경우가 있습니다.



빌트인 파티클 시스템



빌트인 파티클 시스템에 포함된 프로퍼티

빌트인 파티클 시스템을 사용하면 다수의 작은 이미지 또는 메시를 표시하고 애니메이션화하여 단일 시각 효과를 구현할 수 있습니다. 사전 정의된 특정 규칙과 무작위화를 사용하여 크기, 속도, 컬러, 회전과 같은



파티클 프로퍼티를 시간 경과에 따라 애니메이션화해 보세요. 이 작업을 통해 화재, 폭발, 연기, 마법 주문 등의 역동적인 효과를 구현할 수 있습니다.

새 파티클 시스템을 생성하려면 **GameObject > Effects > Particle System** 메뉴 옵션을 선택합니다.

파티클 시스템의 **Main 모듈**에는 전체 시스템에 영향을 미치는 글로벌 프로퍼티가 포함되어 있습니다. 이 프로퍼티의 대부분은 새로 생성된 파티클의 초기 상태를 제어합니다. 주요 프로퍼티는 다음과 같습니다.

- **Duration**: 시스템 실행 시간
- **Looping**: 전체 시스템의 무제한 반복 여부
- **Speed**: 파티클의 초기 속도
- **Start Size**: 파티클의 초기 크기
- **Color**: 파티클의 초기 컬러
- **Gravity**: 파티클에 적용되는 중력

빌트인 파티클 시스템에 포함된 모든 프로퍼티 또는 모듈에 대해 알아보려면 기술 자료의 **파티클 시스템 모듈 컴포넌트 레퍼런스** 섹션을 참조하세요.

파티클 시스템을 서로 중첩하는 것도 가능합니다. 중첩하면 시스템이 동시에 재생되므로 매우 복잡한 효과를 구현할 수 있습니다. 예를 들어 폭발 효과를 위해 연기용 시스템 하나, 작은 불꽃용 시스템 하나, 흩날리는 잔해용 시스템 하나를 조합하는 식으로 사용할 수 있습니다.

파티클 시스템은 게임에 필요한 거의 모든 VFX를 구현할 수 있는 매우 강력한 툴입니다. HDR 언릿 셰이더와 블룸 포스트 프로세싱을 사용하여 효과를 빛나게 할 수도 있습니다. 모든 프로퍼티를 테스트해 보세요. 간단한 파티클이라도 게임에 세련된 느낌을 더할 수 있습니다.

게임 제작 프로세스를 대폭 단축하려면 에셋 스토어에서 **사전 제작된 파티클 VFX**를 살펴보세요.

Shader Graph

VFX를 만드는 또 다른 방법은 셰이더를 사용하는 것입니다. 셰이더는 그래픽 프로세서용 명령 세트로 픽셀 컬러나 버텍스 위치를 계산할 수 있습니다.

Shader Graph를 통해 시각적으로 셰이더를 제작할 수 있습니다. 코드를 작성하지 않고 그래프 프레임워크에서 노드를 만들어 연결하세요. Shader Graph는 변경 사항이 반영된 피드백을 즉각 제공하며, 셰이더 제작이 처음인 사용자도 활용할 수 있을 만큼 간단합니다.

Shader Graph로 할 수 있는 일은 다음과 같습니다.

- UV 와핑 및 애니메이션화
- 표면 형상의 절차적 변경



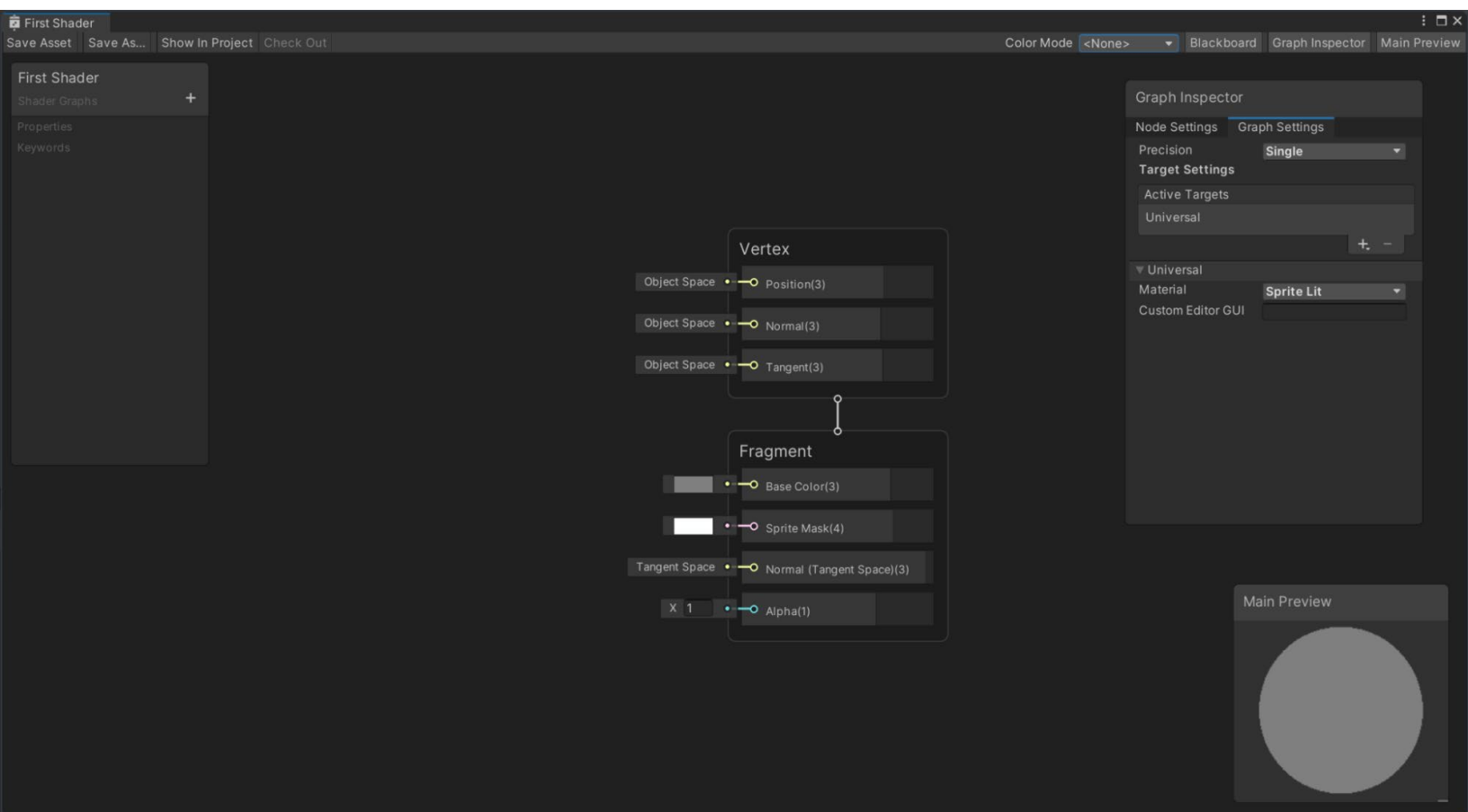
- 이미지 편집 앱과 유사한 이미지 필터 추가
- 월드 위치, 노멀, 카메라와의 거리 등의 정보를 기반으로 오브젝트 표면 변경
- 머티리얼 인스펙터에 프로퍼티를 노출하여 씬의 컨텍스트에서 셰이더 비주얼 조정

Shader Graph 사용

2D 프로젝트에 [Shader Graph](#)를 사용하려면 프로젝트 창에서 오른쪽 클릭하고 **Create > Shader > Universal Render Pipeline > Sprite Lit Shader Graph**(또는 **Sprite Unlit Shader Graph**)를 선택합니다.

스프라이트가 광원의 영향을 받게 하려면 Sprite Lit Shader를 사용합니다. 거의 모든 캐릭터와 오브젝트가 이 경우에 해당합니다. 언릿 또는 발광형 표면에는 Sprite Unlit를 사용합니다. 불, 광원, 전기, 마법 주문, 홀로그램, 유령 등의 캐릭터가 이 경우에 해당합니다.

셰이더를 생성하려면 먼저 파일 이름을 지정합니다. 그러면 초기 셰이더가 생성됩니다. 그런 다음 파일을 더블 클릭하여 편집 모드로 진입합니다.



Shader Graph 창

그러면 Shader Graph 창이 열립니다. 중앙의 작업 공간 내에 [마스터 스택](#)이 있습니다. 이는 Shader Graph의 최종 출력이며 셰이더의 최종적인 모습을 표현합니다. 마스터 스택은 셰이더별로 하나만 있어야 합니다.



마스터 스택에는 버텍스 및 프래그먼트(또는 픽셀) 기능에 해당하는 두 가지 컨텍스트가 있습니다.

왼쪽에는 **블랙보드**가 보입니다. 여기에는 인스펙터에 표시되는 프로퍼티와 키워드가 포함됩니다.

오른쪽 상단에는 **Graph Inspector**가 있습니다. 여기에는 선택한 노드의 설정을 변경할 수 있는 Node Settings, 전체 그래프의 설정을 변경하는 Graph Settings라는 두 개의 탭이 있습니다.

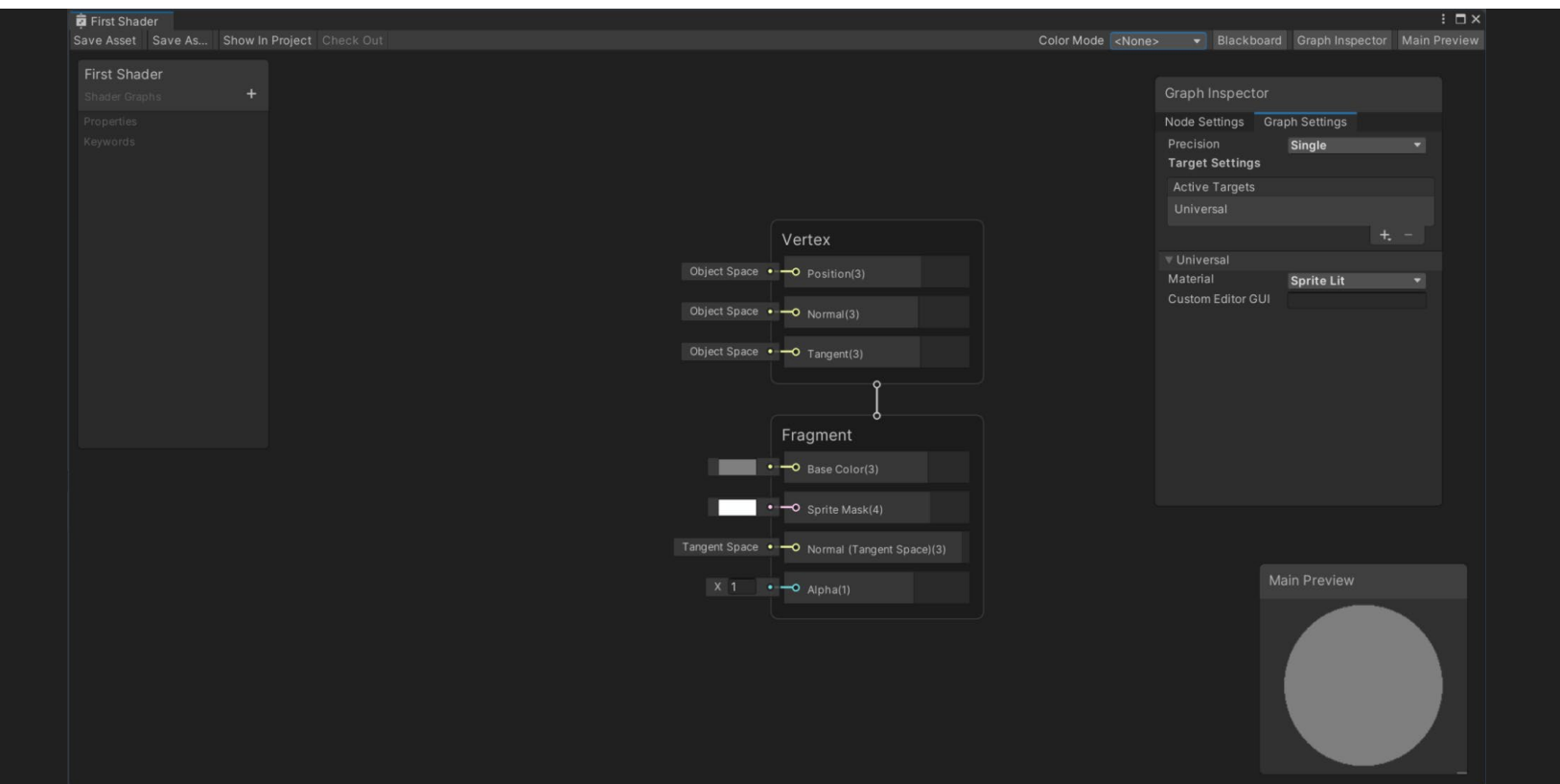
하단의 **Main Preview**에서 최종 셰이더를 확인할 수 있습니다.

Shader Graph의 **노드**는 입력, 효과, 상호 작용을 담는 기본 요소입니다. 그래프는 왼쪽에서 오른쪽으로 진행되므로 노드의 왼쪽에 입력 슬롯, 오른쪽에 출력 슬롯이 있습니다.

2D Shader Graph를 만드는 모든 단계는 **여기**에서 찾을 수 있습니다. Shader Graph 기술 자료는 **여기**에서 확인할 수 있으며, Shader Graph의 각 부분과 그 로직을 설정하는 방법에 대한 자세한 내용을 알아볼 수 있습니다.

다음 섹션에서는 드래곤 크래쉬와 잼 헌터 매치의 몇 가지 Shader Graph 예시를 살펴보세요.

셰이더를 생성하려면 먼저 파일 이름을 지정합니다. 그러면 초기 셰이더가 생성됩니다. 그런 다음 파일을 더블 클릭하여 편집 모드로 진입합니다.



Shader Graph 창



그러면 Shader Graph 창이 열립니다. 중앙의 작업 공간 내에 **마스터 스택**이 있습니다. 이는 Shader Graph의 최종 출력이며 셰이더의 최종적인 모습을 표현합니다. 마스터 스택은 셰이더별로 하나만 있어야 합니다.

마스터 스택에는 버텍스 및 프래그먼트(또는 픽셀) 기능에 해당하는 두 가지 컨텍스트가 있습니다.

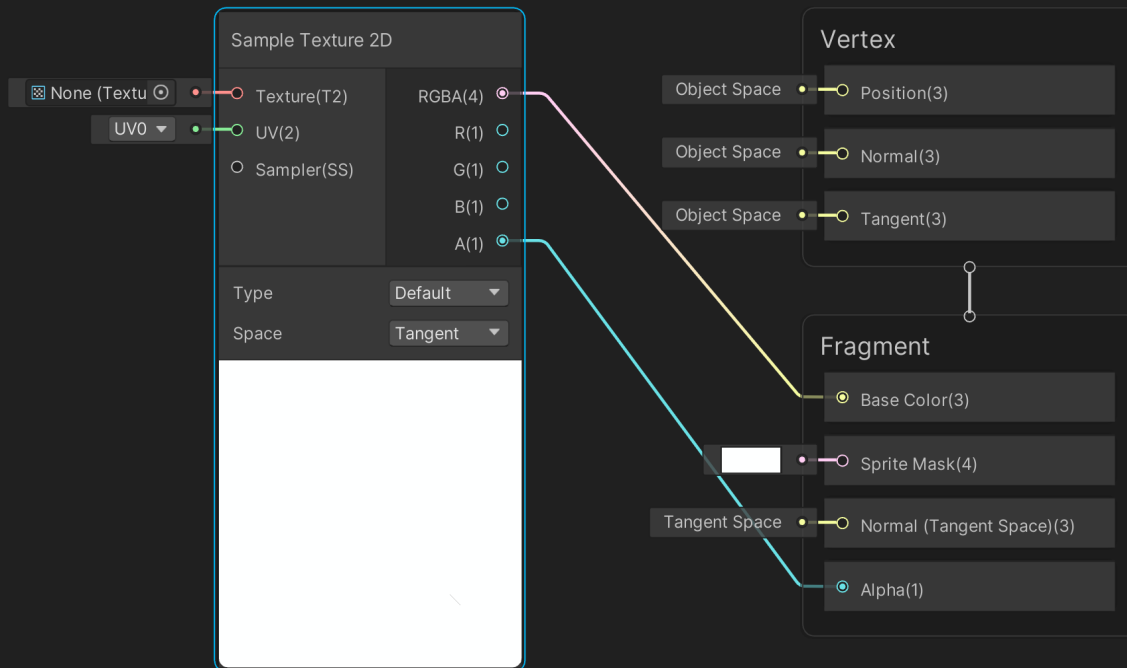
왼쪽에는 블랙보드가 보입니다. 여기에는 인스펙터에 표시되는 프로퍼티와 키워드가 포함됩니다.

오른쪽 상단에는 Graph Inspector가 있습니다. 여기에는 선택한 노드의 설정을 변경할 수 있는 Node Settings, 전체 그래프의 설정을 변경하는 Graph Settings라는 두 개의 탭이 있습니다. 인스펙터에서 Precision 모드를 전환할 수 있으며 저사양 기기에서는 Half로 설정하는 것이 좋습니다. 셰이더 머티리얼을 릿과 언릿 간에 전환할 수도 있습니다.

하단의 Main Preview에서 최종 셰이더를 확인할 수 있습니다.

Shader Graph의 노드는 입력, 효과, 상호 작용을 담는 기본 요소입니다. 그래프는 왼쪽에서 오른쪽으로 진행되므로 노드의 왼쪽에 입력 슬롯, 오른쪽에 출력 슬롯이 있습니다.

새 노드를 추가하려면 작업 공간의 아무 위치에서 오른쪽 클릭하고 컨텍스트 메뉴에서 **Create Node**를 선택합니다. 모든 노드가 유형별로 그룹화된 목록이 제공됩니다. **Input > Texture > Sample Texture 2D**를 선택합니다.

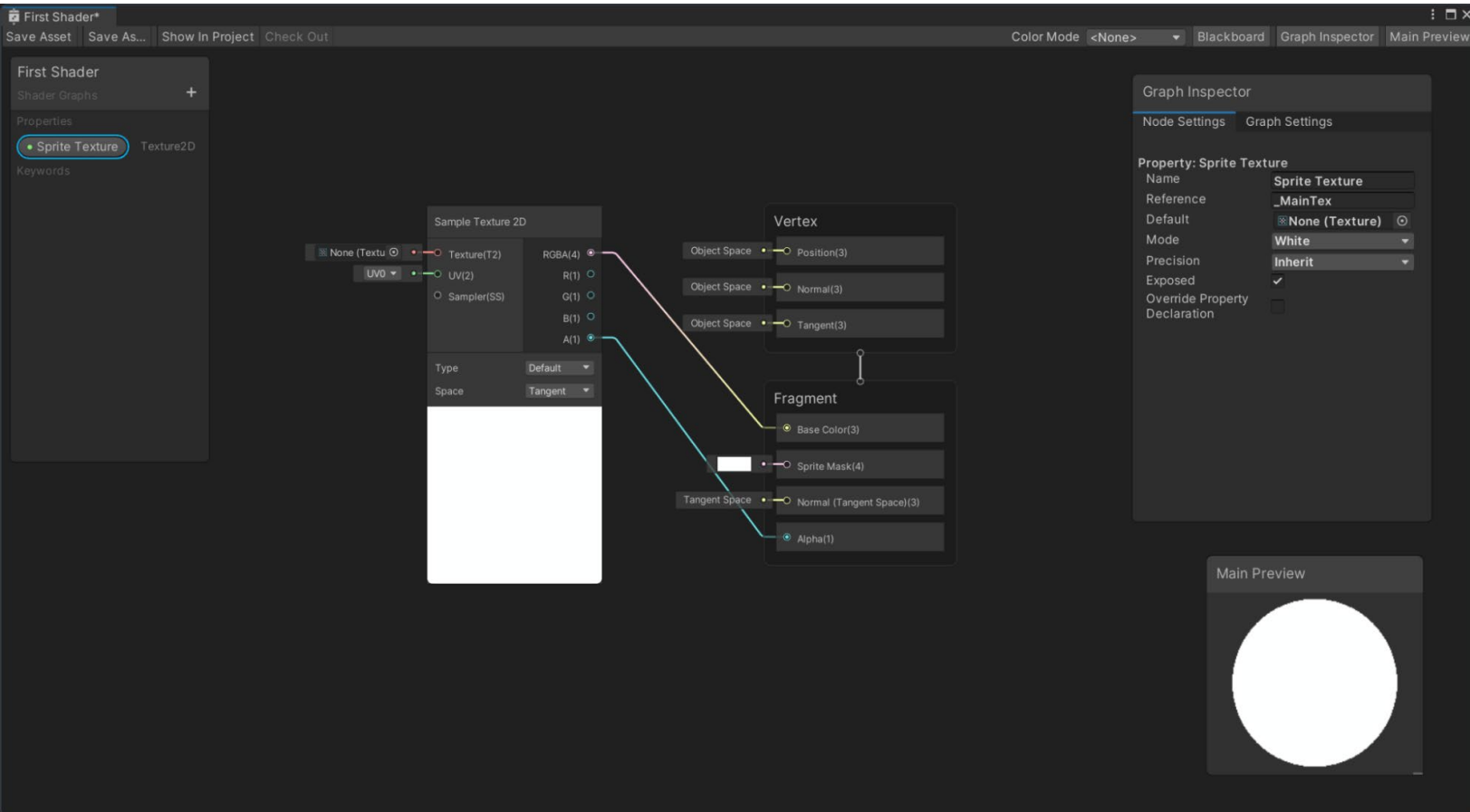


텍스처 2D 노드 예시



Sample Texture 2D 노드의 RGBA(4) 슬롯을 마스터 스택 입력 슬롯의 Base Color(3)에 연결합니다. A(1) 슬롯을 Alpha(1)에 연결합니다. 이제 마스터 스택의 Fragment 컨텍스트에서는 제공된 텍스처의 컬러와 알파를 Sample Texture 2D 노드에 사용합니다.

셰이더 외부에서 이 텍스처를 참조하려면 프로퍼티가 필요합니다.



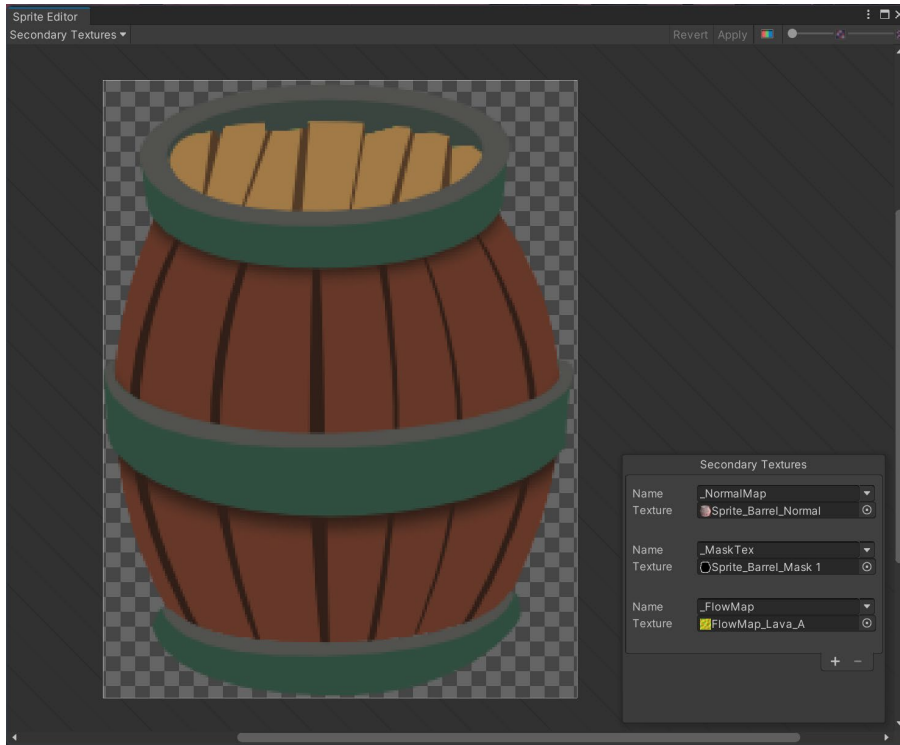
노출된 프로퍼티는 그래프에서 사용할 파라미터를 수정할 수 있는 인스펙터 뷰에 표시됩니다.

블랙보드에 있는 더하기 버튼을 클릭하고 Texture 2D를 선택합니다. 새롭게 생성된 프로퍼티를 선택하면 Graph Inspector에 옵션이 표시됩니다. 프로퍼티 이름을 변경할 수 있지만 Reference 옵션을 _MainTex로 변경하는 것이 더 중요합니다.

텍스처의 경우 각 스프라이트 렌더러에서 설정된 스프라이트를 사용합니다. 스프라이트 렌더러는 현재 머티리얼 셰이더의 _MainTex 프로퍼티를 검사하고 해당 스프라이트 텍스처를 셰이더의 텍스처로 설정합니다. 마지막으로 블랙보드에서 작업 공간으로 이 프로퍼티를 드래그하고 Sample Texture 2D 노드의 Texture(T2) 입력 슬롯에 연결합니다. 톨바에서 Save Asset 버튼을 누르면 셰이더가 컴파일됩니다.

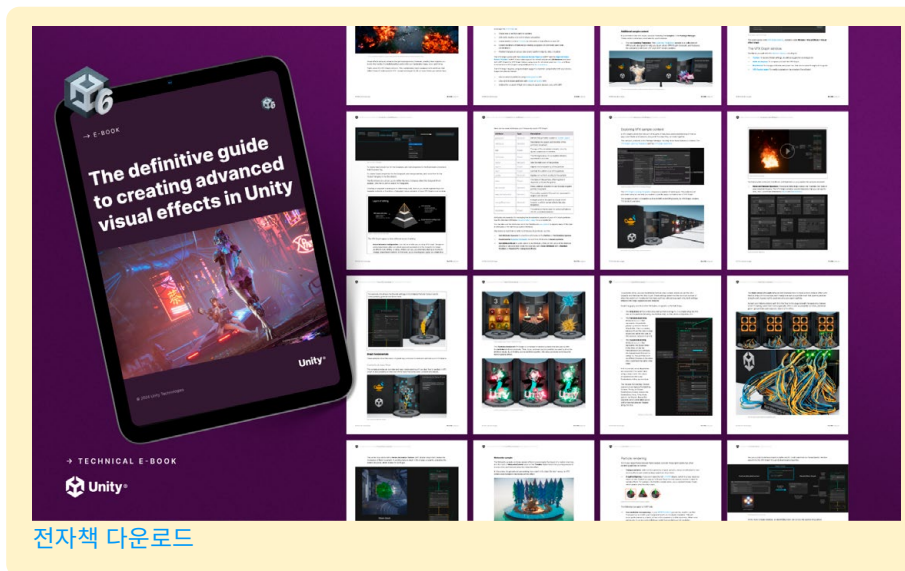
Unity의 다른 셰이더처럼 Shader Graph를 사용할 수 있습니다. 프로젝트 창에서 오른쪽 클릭하고 **Create > Material**을 선택한 후 목록에서 셰이더를 선택합니다. 해당하는 모든 프로퍼티를 커스터마이징할 수 있습니다.

생성된 머티리얼을 스프라이트에 적용하여 셰이더를 테스트할 수 있습니다. 물론 이 예시는 텍스처만 사용한 가장 간단한 셰이더입니다. 나머지 노드도 자유롭게 실험해 보시기 바랍니다. 노멀 맵과 마스크 맵(스프라이트 마스크)을 이 셰이더에 추가해 볼 수도 있습니다. 힌트: 참고로 노멀 맵을 제대로 표시하려면 Sample Texture 2D Type을 Normal로 설정해야 하며, Sprite Editor의 Secondary Textures 창에서 노멀 맵과 마스크 맵 텍스처의 레퍼런스 이름을 확인할 수 있습니다. 행운을 빕니다!



나무통 스프라이트의 보조 텍스처. 텍스처 이름을 확인하고 자체 텍스처를 추가할 수 있습니다. Shader Graph에서 텍스처는 이름으로 참조됩니다.

VFX Graph의 다양한 기능에 대한 전체 안내를 보려면 Unity 6에서 고급 시각 효과를 만드는 방법에 대한 포괄적인 가이드를 다운로드하세요.

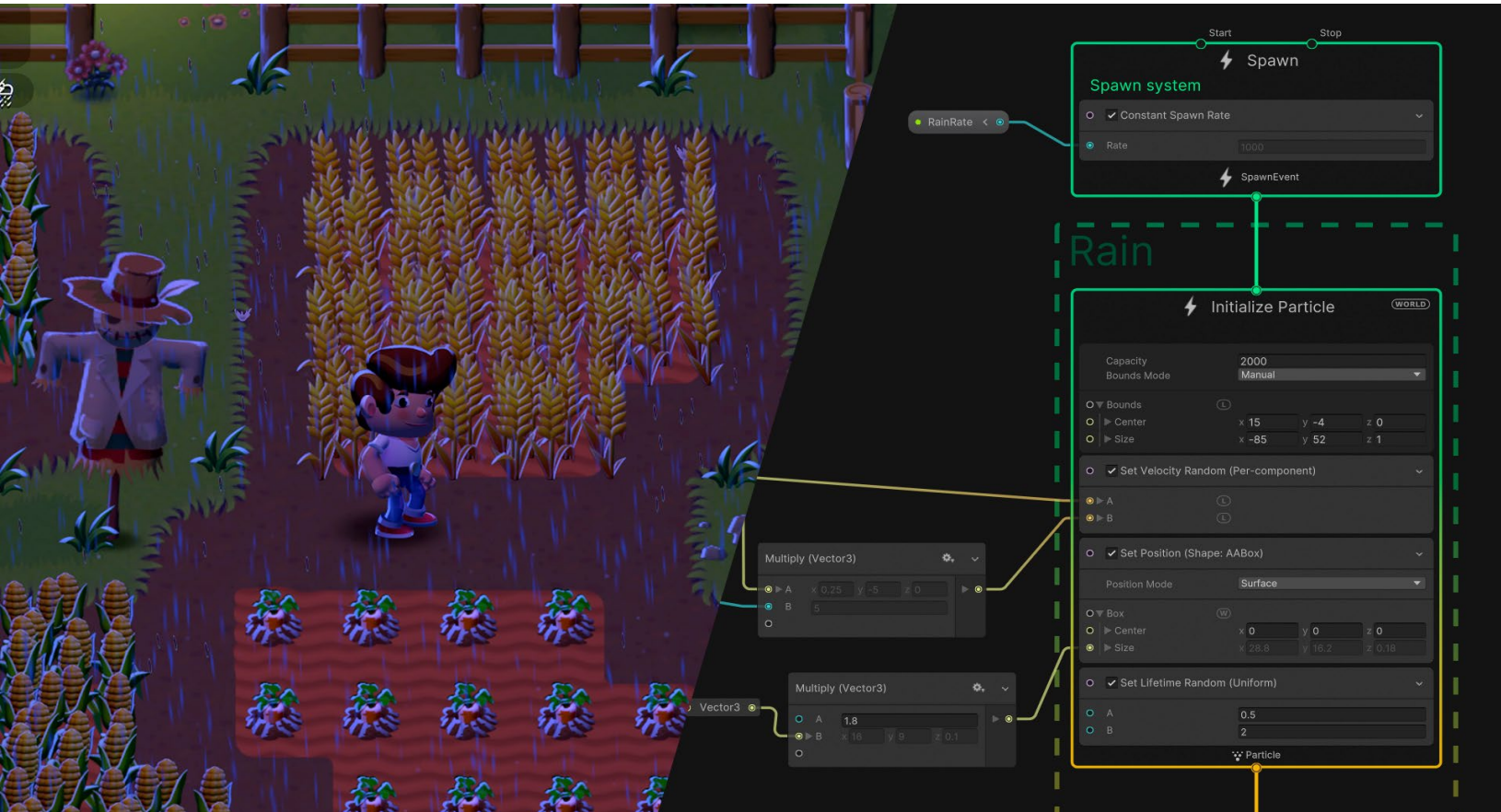


전자책 다운로드



Unity 샘플의 VFX

해피 하비스트의 비 효과



해피 하비스트에서 VFX Graph로 비 효과 설정

비는 게임에서 일반적인 파티클 기반 효과입니다. 다음은 VFX Graph를 사용하여 해피 하비스트에서 비 효과를 만드는 데 사용되는 주요 단계입니다.

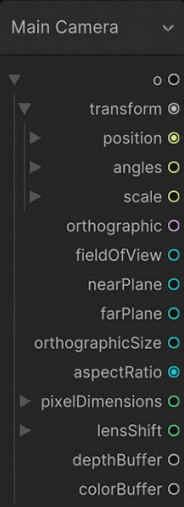
1. 비 파티클(계층 구조의 **Visual Effect Rain**)은 지면 위의 정렬 레이어에 렌더링되지만 오브젝트 뒤에 있습니다. 물방울 파티클이 지면에 닿을 때 물 스플래시 효과가 생성되어 실제 비가 반응하는 방식을 시뮬레이션하기 때문입니다. 비가 램프 기둥과 같은 수직 표면에 표시되면 어색해 보일 수 있습니다.

실제 세계에서 광원은 비를 포함한 모든 환경에 영향을 미칩니다. 해피 하비스트에서 비 효과를 위한 VFX Graph는 비 파티클을 비추기 위해 Output 컨텍스트에서 스프라이트 릿 기반 Shader Graph를 사용합니다. 비에 대한 릿 Shader Graph가 없으면 톤이 하루 동안 일어나는 빛의 변화에 맞게 변경될 수 없습니다.

프레임별 물 스플래시: 비 파티클이 사라지면 **Trigger On Die** 블록이 GPU 이벤트를 트리거하여 단일 플립북 파티클을 생성하고 간단한 물 스플래시 애니메이션을 재생합니다. 해피 하비스트에서는 텍스처 내의 다양한 프레임은 표시하기 위해 **Flipbook** Shader Graph UV 블록을 사용합니다. 또는 VFX Graph에서 **Flipbook Player** 블록을 사용해도 동일한 효과를 얻을 수 있습니다.

- 벼락의 섬광:** 카메라 위치의 중앙에서 초기화된 VFX Graph 기반의 단일 파티클을 생성하고 화면을 가득 채울 만큼 충분히 크게 만들면, 이 효과를 간단하면서도 효과적으로 생성할 수 있습니다. Output 노드의 Color Over Lifetime 블록은 시간이 지남에 따라 그레디언트를 평가하여 파티클의 컬러를 변경합니다. 해당 그레디언트에서 투명도 변화를 통해 벼락의 깜박임과 사라짐을 시뮬레이션할 수 있습니다.

카메라 타일링을 통한 최적화



비는 해피 하비스트의 전체 씬에 영향을 미치지만, 카메라를 통해 보이는 부분은 일부에 불과하기 때문에 계산된 파티클의 많은 부분이 실제로는 필요하지 않습니다. 인스펙터에서 **Frustum Culling** 옵션을 선택하여 이러한 파티클을 렌더링하지 않도록 하거나, 데모에서 사용된 기능인 카메라 타일링을 사용하면 됩니다.



카메라 타일링을 사용하면 씬 뷰와 일치하는 박스에서 파티클을 생성하고 렌더링하여, 파티클이 씬의 모든 곳에서 시뮬레이션되는 것처럼 보이게 할 수 있습니다. 카메라 외부에서는 파티클이 생성되지 않으므로, 생성되는 파티클 수와 효과의 할당 용량을 추가 설정 없이도 줄일 수 있습니다.

해피 하비스트에서는 카메라 타일링을 사용하기 위해 비 생성을 카메라 크기에 맞게 설정했으며, 이때 Set Position의 셰이프를 **AABBox**로 설정하였습니다. 카메라 형식(크기를 수동으로 입력하거나, MainCamera를 직접 사용하거나, aspectRatio 슬롯을 사용)은 크기 멀티플라이어로 씬의 카메라 설정에 맞게 조정됩니다. AABBox는 컬링으로 인한 팝핑(popping) 현상을 피하기 위해 실제 카메라 크기보다 약간 큼니다. Shape Spawn 모드는 2D 프로젝트의 전체 볼륨에서 생성할 필요가 없기 때문에 Surface로 설정됩니다(Z축은 사용되지 않음).

이 설정이 완료되면 시스템의 출력에 **Tile/Warp Positions** 블록을 추가해야 하며, 이 블록은 카메라 트랜스폼에 연결되어 위치와 카메라 치수에 맞게 조정됩니다. 이는 시스템의 **Initialize 컨텍스트**에서도 마찬가지로 적용해야 합니다.

이 기술은 3D의 비 효과에도 적용되지만, 뎀스 효과를 원할 경우 Z축에 대해 0이 아닌 값을 사용해야 하며 Shape Spawn 모드는 Volume으로 설정해야 합니다.

아늑한 모닥불



농부는 바쁜 하루를 마치고 VFX Graph로 만들어진 모닥불 앞에서 불멍을 즐깁니다.

게임에서 활용할 수 있는 또 다른 일반적인 효과는 불입니다. 해피 하비스트에서는 마당과 농부의 집 벽난로에 아늑한 모닥불이 있습니다. 불 효과는 세 가지 파티클 세트로 구성되는데 불길, 불꽃, 연기가 모두 **VFX_Fire**라는 동일한 그래프에서 생성됩니다.

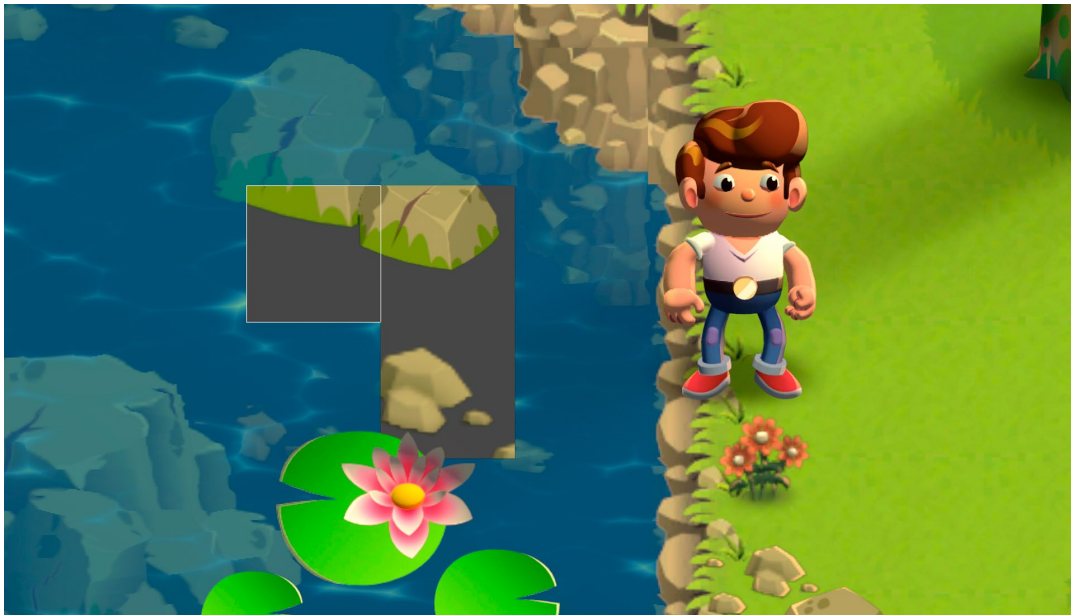


불길 애니메이션

부드러운 느낌의 불길 애니메이션은 Shader Graph의 셰이더 레벨에서 구현되며, GPU와 셰이더의 성능을 활용하여 무엇을 구현할 수 있는지 보여 줍니다.

프로젝트에서 **ShaderGraph_Fire** 파일을 열면 단일 스프라이트(**Mask texture2D**)가 사용되어 불길에 타원형 셰이프를 내기 위한 마스크를 생성하는 것을 알 수 있습니다. 애니메이션과 스프라이트는 노이즈와 불길 및 배경을 위한 틸트 컬러가 적용된 스크롤 **보로노이** 패턴을 통해 생성됩니다. 알파 마스크는 동일한 불길 셰이프를 사용하지만, 더 눈에 띄는 외곽선을 생성하는 **Saturate Operator**가 적용됩니다. 불 효과에 세련된 디테일을 더하는 불꽃과 연기 효과는 표준 릿 Shader Graph를 사용합니다. Velocity, Scale 및 Initial Position 블록에 대한 변경 사항을 조합하면 단일 스프라이트만으로도 생동감 있는 효과를 생성할 수 있습니다. 불꽃 및 연기 효과의 그래프에는 자체 프로젝트에서 효과를 재현하는 데 도움이 되는 참고 노트가 포함되어 있습니다.

물 타일 애니메이션



물에 추가된 효과는 데모의 밝고 만화 같은 스타일을 조성하는 데 유용합니다.

물은 만화 같은 커스틱 또는 투명 효과가 있는 파도를 사용하여 데모의 전체 아트 스타일에 맞는 효과를 냅니다. 물 셰이더는 타일맵과 원활하게 작동하도록 최적화되어 있으며, 데모 환경의 대부분은 이 타일맵으로 만들어져 있습니다.

기본 2D 릿 머티리얼인 **Water**는 타일맵의 Tilemap Renderer에서 찾을 수 있습니다. 이는 물 셰이더가 있는 커스텀 머티리얼로 교체되었습니다. 타일맵 내부의 모든 타일은 동일한 머티리얼을 사용하여 렌더링되므로, 이 타일맵은 물 타일에만 사용됩니다.

ShaderGraph_Water 그래프에서 물 효과는 늘리고 반전시킨 보로노이 노이즈 패턴을 통해 생성되어 커스틱을 시뮬레이션하며, 이는 물 커스틱에서 일반적인 방법입니다. 진동 모션은 타일링과 오프셋 UV의 시간 경과에 따른 결과물로, 여기에 노이즈 효과도 적용됩니다.

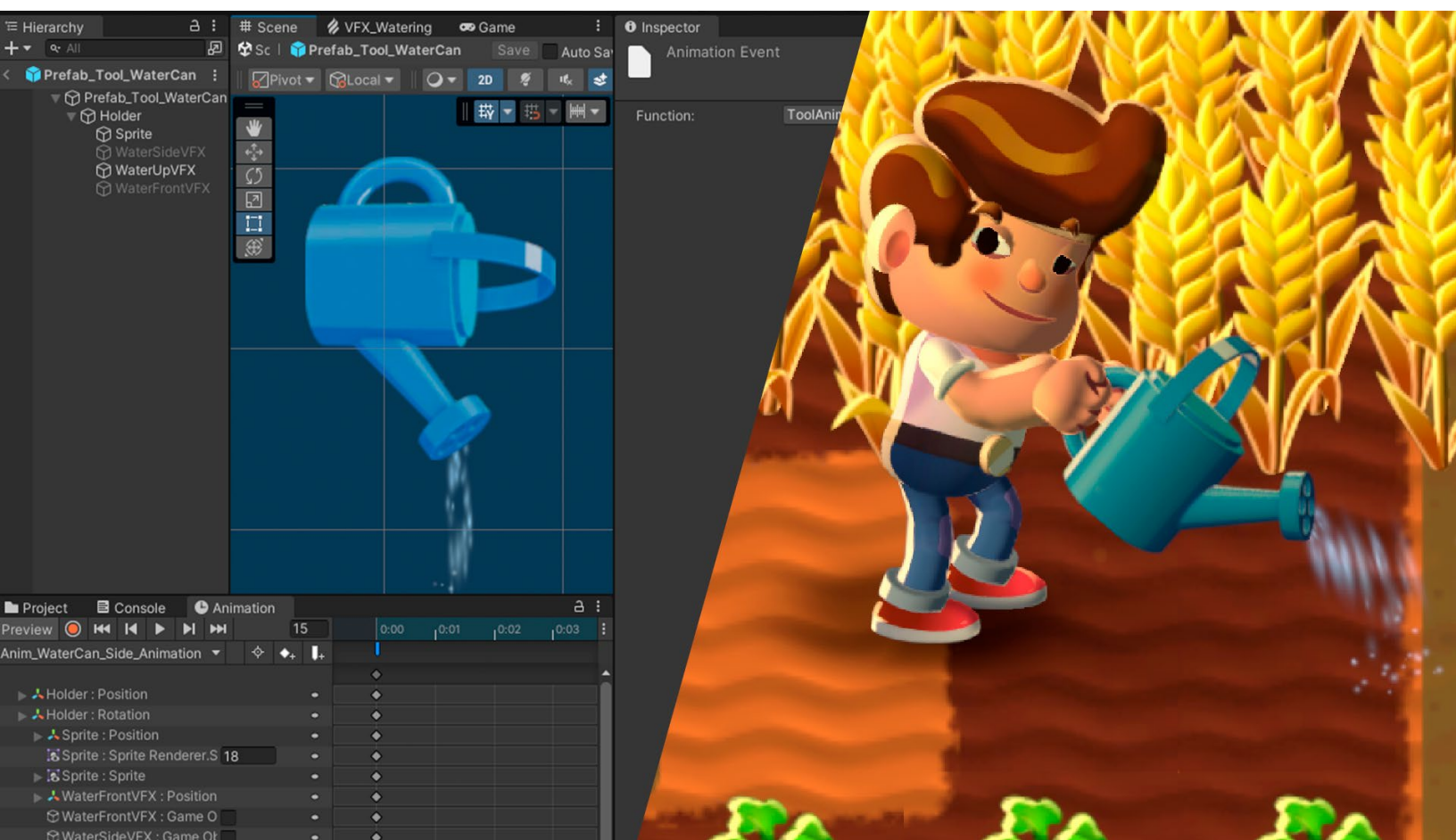


굴절 효과

굴절 효과는 Lerp 노드를 사용하여 표면 텍스처에 블렌딩됩니다. 굴절 효과에는 [Camera Sorting Layer](#) 텍스처가 사용됩니다. 이 텍스처는 지정된 정렬 레이어까지 카메라가 렌더링한 화면을 캡처한 스냅샷이며, 따라서 그 이전 정렬 레이어에 있는 모든 요소들이 이 텍스처에 포함됩니다. 해피 하비스트에서는 렌더링이 **Objects** 정렬 레이어에서 이루어지며, 이 레이어는 캐릭터 같은 요소에서 사용하는 레이어입니다. Camera Sorting Layer 텍스처의 UV는 Normal 및 Height 노드를 사용해 생성된 노멀 맵으로 왜곡되며, 이때 표면 텍스처에도 사용되는 보로노이 텍스처가 사용됩니다.

로스트 크립트 2D 데모를 사용하는 [이 튜토리얼](#)에서 2D 프로젝트의 다른 물 효과에 대한 단계별 튜토리얼을 알아보세요.

게임플레이에서의 VFX



물뿌리개 프리팹에는 캐릭터가 향하는 방향에 따라 스프라이트를 변경하는 세 개의 애니메이션 클립이 연결되어 있습니다.

지금까지 다른 효과는 해피 하비스트의 게임플레이와 상호 작용하지 않습니다. 이들은 환경의 일부이며, 데모에서 발생하는 일과 관계없이 재생됩니다. 그러나 대부분의 게임에서는 환경 또는 캐릭터의 행동이 발생할 때 일부 효과를 재생하도록 트리거해야 합니다. 해피 하비스트에서 트리거된 효과를 살펴보겠습니다.



아이템 사용 시 효과

일부 효과는 물뿌리개와 같은 캐릭터 툴에 연결되어 있습니다. 물뿌리개의 프리팹 **Prefab_Tool_WaterCan**에는 캐릭터가 향하는 방향(위, 아래, 옆)에 따라 스프라이트를 변경하는 세 개의 애니메이션 클립이 있습니다. 이 애니메이션에는 Tool Animation Event Handler 컴포넌트 내에서 함수를 호출하는 **애니메이션 이벤트**가 있습니다.

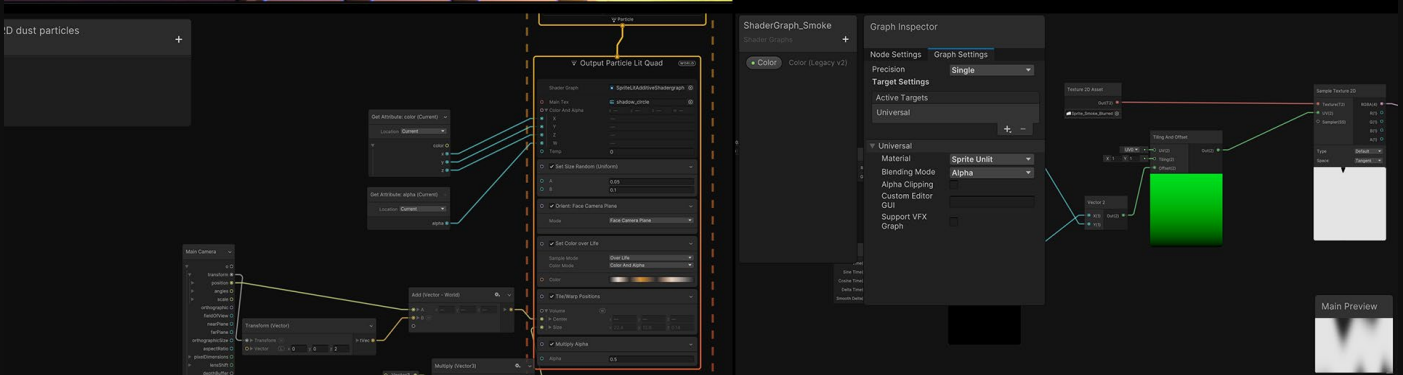
이 스크립트는 인스펙터에서 참조된 위치별로 VFX Graph의 해당 인스턴스를 활성화하거나 비활성화합니다.

타일로 반복 작업할 때의 효과

땅 파기, 수확 등 일부 효과는 타일에서 발생합니다. 이렇게 하면 캐릭터 애니메이션이나 위치 변경에 관한 문제가 일어나지 않습니다. 땅을 갈 때가 되면 경작 효과가 해당 셀에서 재생되고, 수확할 때가 되면 수확 효과가 셀에서 호출됩니다.

경작 효과가 트리거되면 **Grid** 게임 오브젝트에 연결된 **TerrainManager** 컴포넌트로부터 **TillingPuff_prefab** 효과를 참조합니다. 이는 **타일의 중앙**으로 이동한 후 재생됩니다. 작물 수확과 같은 다른 효과는 **Data/Crops** 스크립터를 오브젝트에서 참조되는 VFX Graph를 호출합니다. **TerrainManager** 스크립트 초기화 시, 필요할 때 재생할 여러 VFX 프리팹을 인스턴스화하는 VFX 풀 디렉터리가 생성됩니다(스크립트의 변수 **m_HarvestEffectPool**).

작은 디테일이 만들어 내는 차이



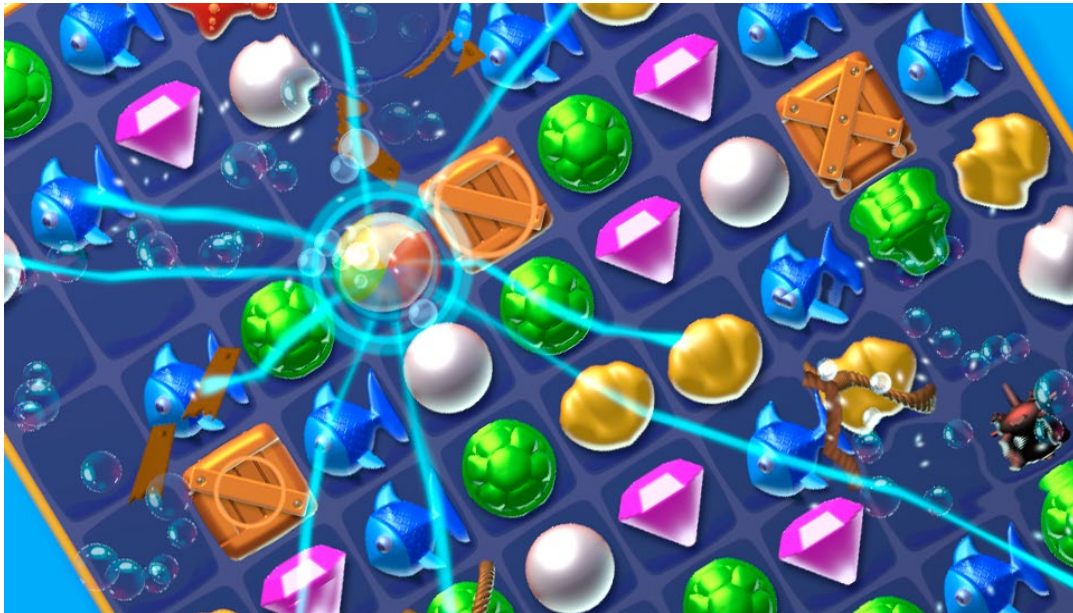


해피 하비스트에서는 작은 효과들이 데모에 세련미를 더하고 탐험을 더욱 즐겁게 만들어 줍니다. 이러한 효과 중 일부는 다음과 같습니다.

- **VFX_DustParticles:** 환경에 먼지 퍼뜨리기
- **VFX/Water:** 미세한 물결과 스플래시
- **VFX_Leaves:** 플레이어가 지나갈 때 덤불에서 떨어지는 잎 트리거
- **Character:** 캐릭터에 연결된 연기 효과 트리거. 이는 빌트인 파티클 시스템으로 만들어졌습니다.
- **P_VFX_Moths:** 스포트라이트 근처에서 날아다니는 나방 트리거. 이 또한 빌트인 파티클 시스템으로 만들어졌습니다.

위 목록의 각 VFX Graph 기반 효과에는 각 노드의 목적을 설명하는 참고 사항이 포함되어 있습니다.

젼 헌터 매치의 VFX



젠 헌터 매치의 VFX Graph 및 Shader Graph 기반 시각 효과를 보면 각 그래프에 설명 주석이 포함되어 있어, 각 시스템이 2D 광원과 함께 작동하는 방식을 이해할 수 있습니다.

젠 헌터 매치의 특수 효과는 다음과 같습니다.

- **언릿 파티클을 표시하는 VFX Graph 기반 효과:** 이 효과는 Output Particle Quad를 사용합니다.
- **커스텀 셰이더를 사용하는 VFX Graph 기반 파티클 효과:** 파티클을 렌더링하기 위해 Shader Graph를 사용합니다. 예를 들어 **VFX_CellHoldDistortion**이라는 효과는 출력에서 2D 셰이더를 사용하며, 셰이더에서는 **Support VFX Graph** 옵션이 활성화되어 있습니다. Output 노드에 2D Shader Graph를 추가하면 출력은 다음 중 하나가 됩니다.

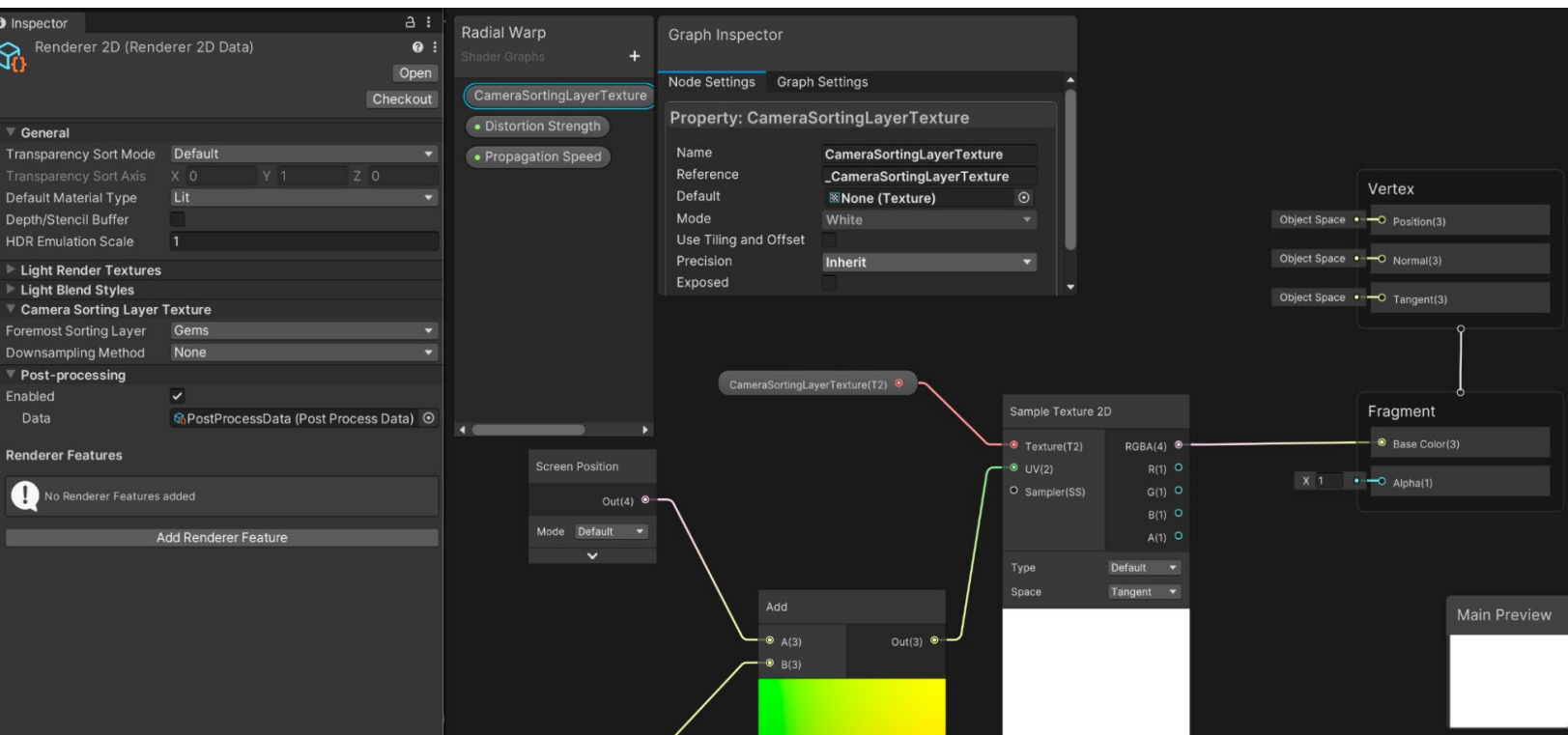


- **Output Particle Sprite Unlit Quad:** Shader Graph의 마스터 노드가 Sprite Unlit일 경우 파티클에 효과를 적용할 수 있지만 2D 광원에는 반응하지 않습니다.
- **Output Particle Sprite Lit Quad:** Shader Graph의 마스터 노드가 Sprite Lit일 경우 2D 광원에 반응합니다.

완성도 높은 효과는 보통 특정 시점에 [GPU 이벤트](#)로 생성되는 여러 개의 Spawn 노드를 사용하여 부드럽게 연결한 일련의 효과 시퀀스로 구성됩니다. 자체 파티클 효과를 만들 때는 각 VFX Graph에 대해 인스펙터에서 올바른 정렬 레이어를 사용해야 합니다.

Camera Sorting Layer Texture

Radial Warp 셰이더는 URP 2D [Camera Sorting Layer Texture](#) 설정을 사용합니다. 이 유용한 기능을 통해 URP 2D Renderer 설정의 지정된 정렬 레이어까지 생성된 그래픽스를 가져와 Shader Graph에서 효과를 적용하는 데 사용할 수 있습니다. Camera Sorting Layer Texture는 [해피 하비스트](#) 샘플에서 물 굴절 효과를 구현하는 데 사용되었고, [드래곤 크래셔](#)에서는 연기 왜곡 효과에 사용되었습니다. 이 샘플에서는 충격파를 표현하는 왜곡 효과를 적용하여 점을 매칭했을 때의 시각적인 효과를 더하는 데 사용되었습니다.



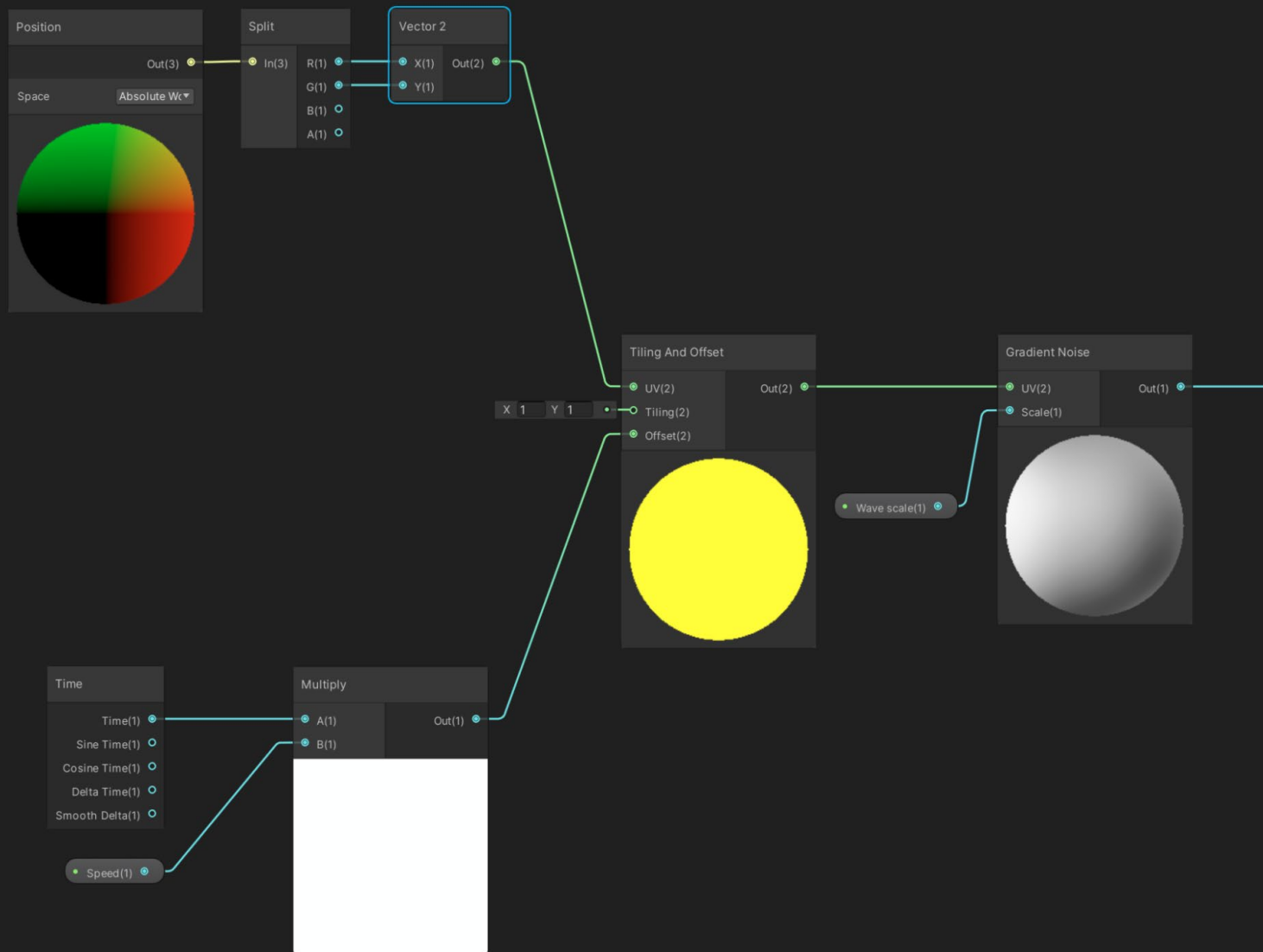
가장 왼쪽에 있는 Renderer 2D Data 에셋과 Radial Warp 셰이더는 화면의 이미지를 셰이더의 기본 컬러에 대한 텍스처로 읽고, UV에서 변형을 적용합니다.



드래곤 크래셔의 Shader Graph 예시

버텍스 변위

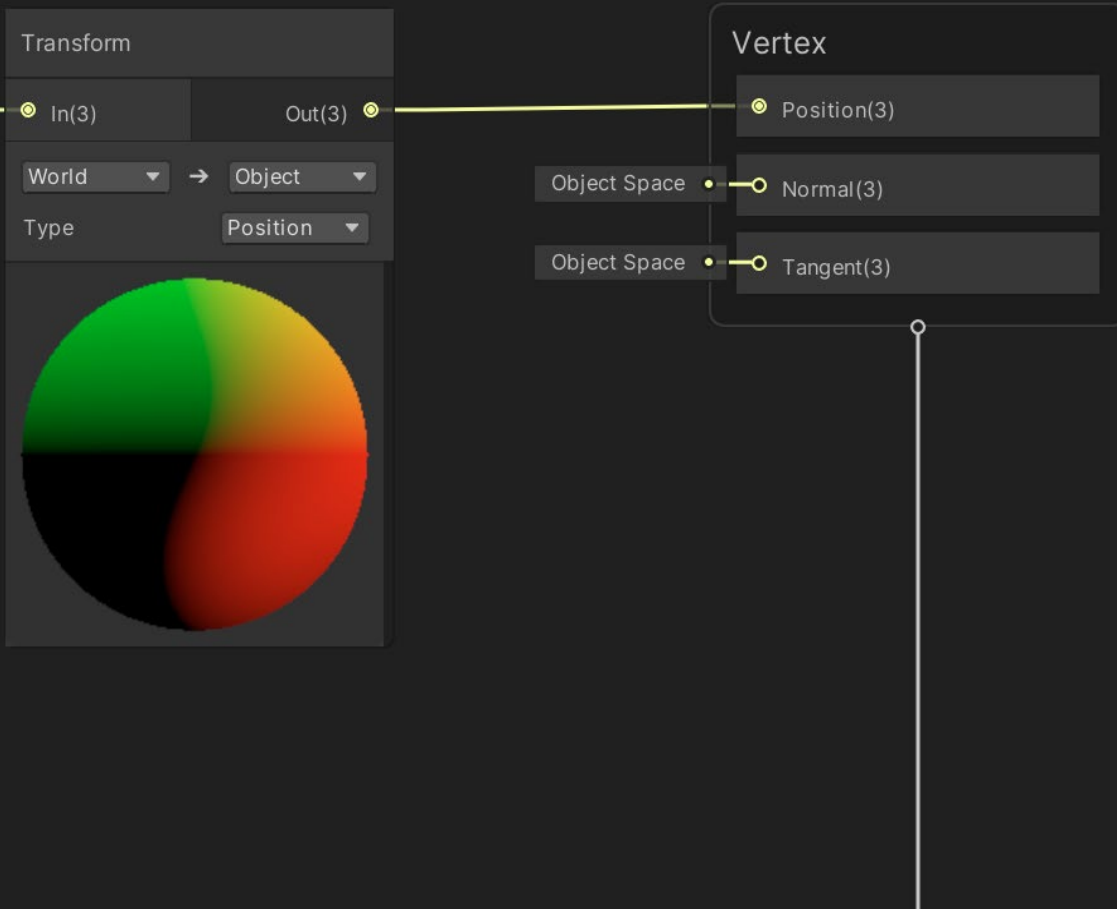
버텍스 변위 셰이더는 메시의 지오메트리 내에 있는 버텍스의 위치에 영향을 미칩니다. 버텍스는 다양하게 이동시킬 수 있으며 움직이는 노이즈를 적용하는 것이 가장 간단한 방법입니다. 이 셰이더는 나뭇잎, 걸려 있는 덩굴과 밧줄, 깃발처럼 흔들리는 오브젝트에 적합합니다. 드래곤 크래셔 프로젝트에서 이 Shader Graph 파일의 예시(파일 이름: ShaderGraph_Sprite_Lit_Waving)를 확인할 수 있습니다.



흔들리는 셰이더의 주요 부분

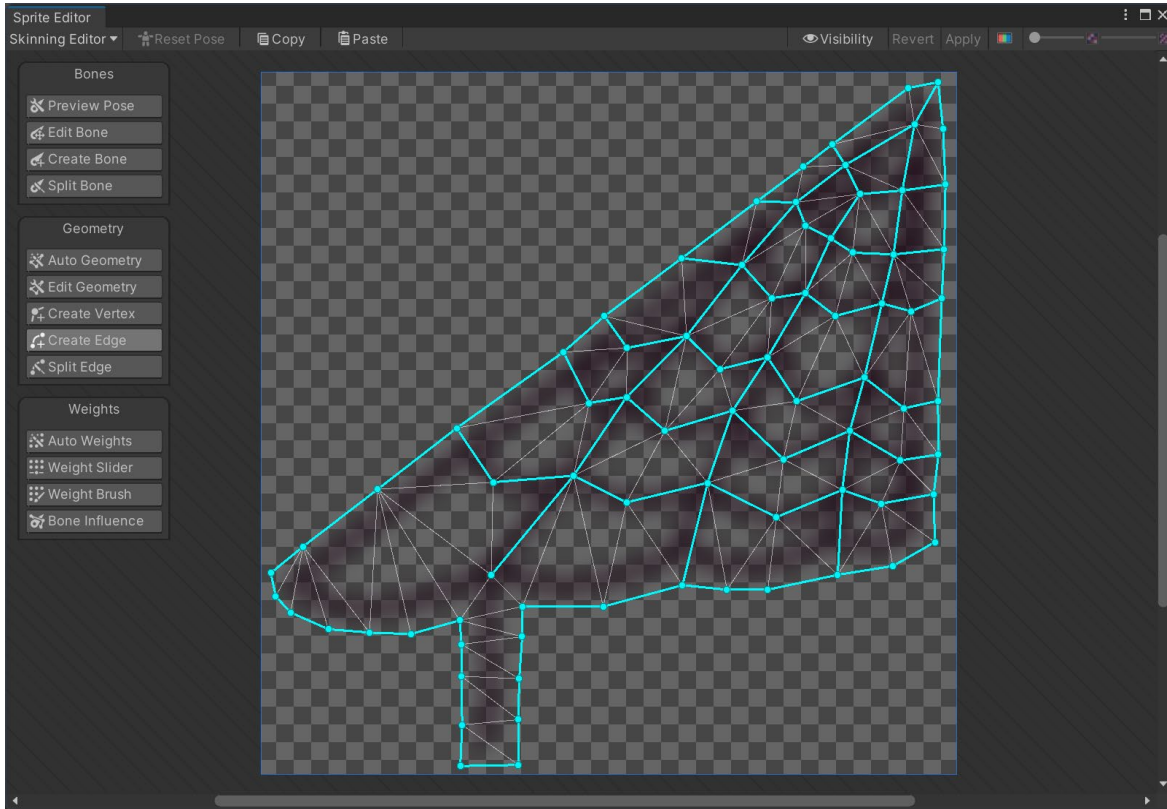


흔들리는 애니메이션을 구현할 때는 그래디언트 노이즈가 웨이브 마스크로 사용됩니다. UV 오프셋은 Speed라는 플롯을 곱한 Time 노드에 의해 애니메이션화됩니다. 이 모든 요소가 Vertex 컨텍스트의 버텍스 위치에 영향을 미칩니다.



Shader Graph에서 Vertex 컨텍스트는 버텍스 위치를, Fragment 마스터 스택은 픽셀 정보를 수정합니다.

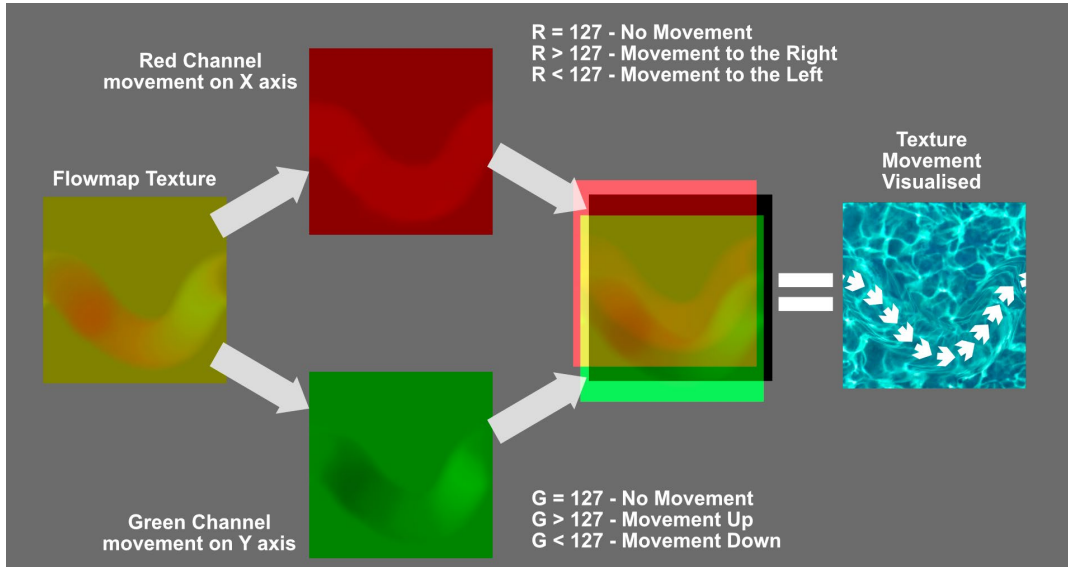
Waving 셰이더의 영향을 받는 스프라이트는 버텍스 개수가 충분히 있어야 하며 그렇지 않으면 애니메이션이 투박한 느낌으로 구현됩니다. 버텍스를 편집하려면 Skinning Editor의 Geometry 툴을 사용하세요.



거미줄 세이더가 적용된 모습 ([여기](#)에서 애니메이션 확인)

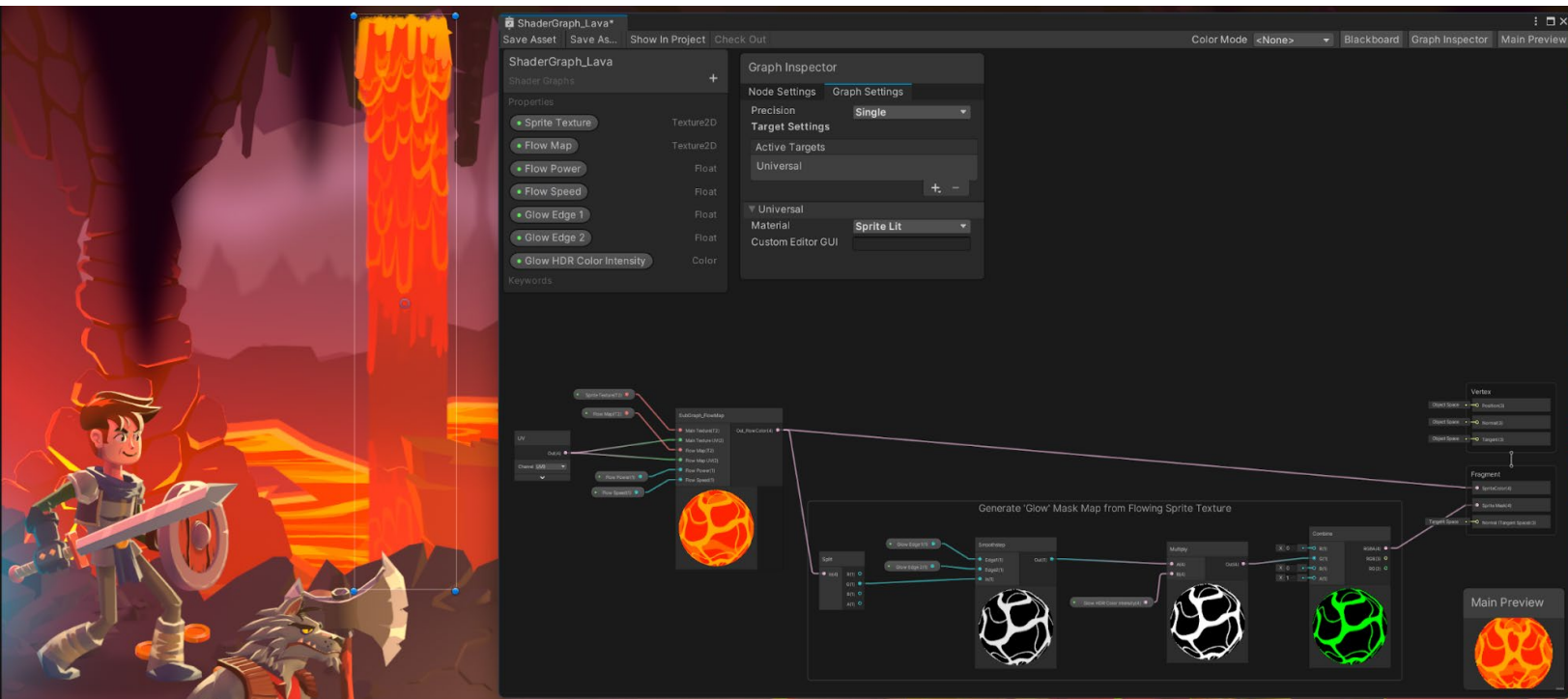
플로 맵

플로 맵은 방향 정보를 저장하는 텍스처입니다. 드래곤 크래셔 데모에서 ShaderGraph_Lava라는 플로 맵 셰이더를 확인할 수 있습니다. 이 셰이더는 플로 맵 텍스처를 사용하여 메인 텍스처 UV 좌표의 방향을 제어합니다. 빨간색과 녹색은 모든 프레임에서 픽셀이 따르는 XY 방향을 나타내는 데 사용되며, 메인 텍스처의 픽셀이 흐르는 듯한 효과를 구현합니다.



플로 맵 텍스처 컬러에 대한 설명: 빨간색은 X축에 따라 픽셀의 움직임을 제어하고, 녹색은 Y축에 따라 픽셀의 움직임을 결정합니다. 셰이더는 화살표로 표시된 방향으로 메인 텍스처의 픽셀을 움직입니다.

SubGraph FlowMap에서 이 효과를 구현하는 방법을 알아보세요.

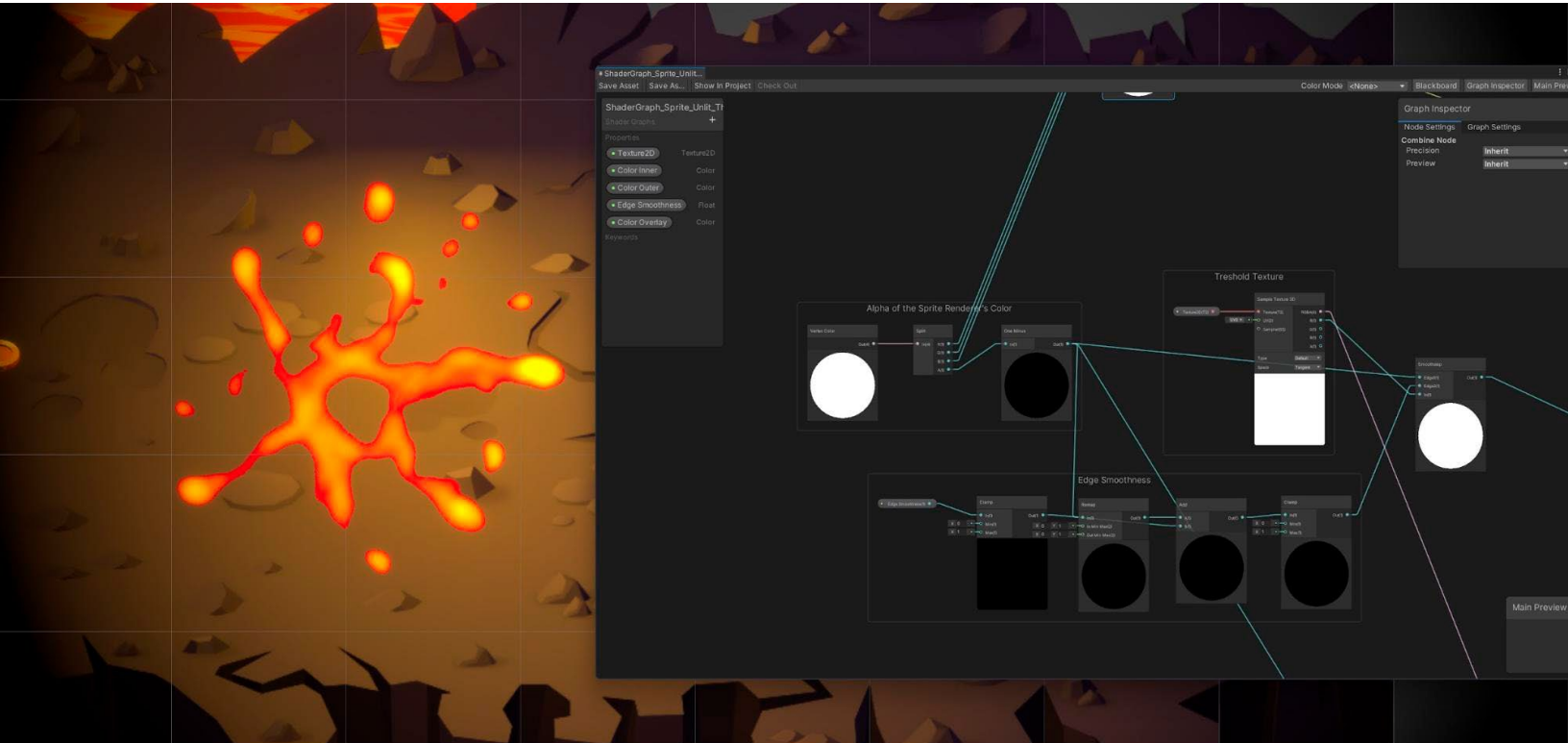


손으로 칠한 플로 맵은 용암의 흐름에 점성 효과와 만화적인 효과를 더하며, 프로젝트 아트 방침과 일치하는 결과물을 만들어 냅니다. 플로 맵 제작에 사용할 수 있는 몇 가지 툴에는 [Flow Map Generator / Visualizer](#) 및 [Flow Map Painter](#)가 있습니다.



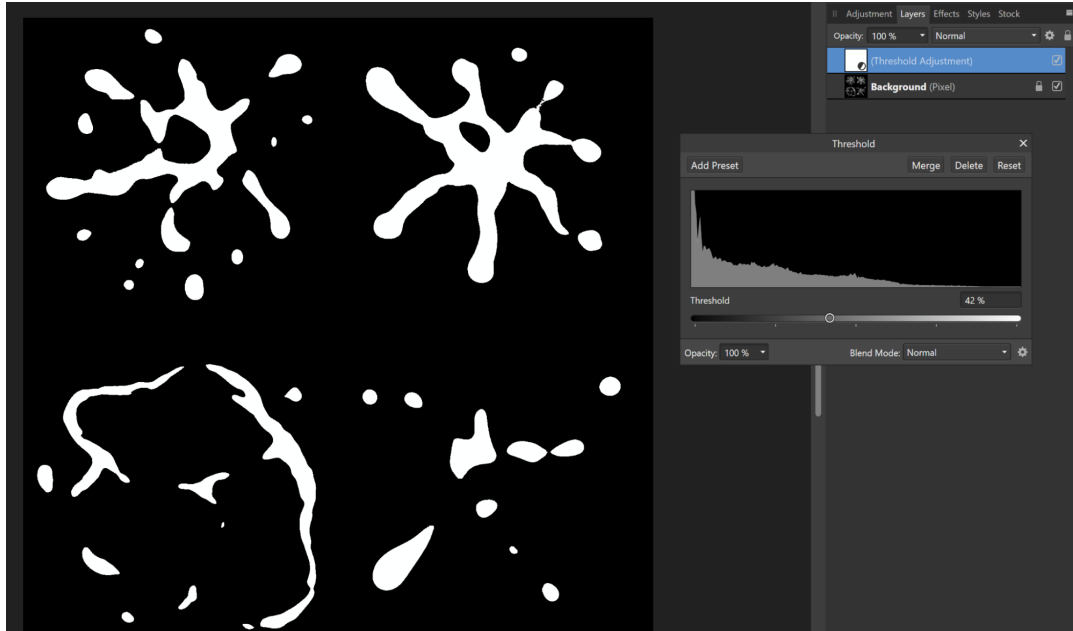
유체 프랍 애니메이션용 임계값 애니메이션

단일 텍스처로 매끄러운 애니메이션을 생성하는 ‘애니메이션화된 알파 클리핑’도 사용해 볼 만한 셰이더 애니메이션 기법입니다. 이 기법은 각 프레임에서 알파 값에 기반하여 특정 범위의 픽셀을 표시하는 방법으로 구현됩니다. 텍스처가 단일 채널이므로 효과를 간단히 구현할 수 있습니다. 드래곤 크래셔에서 ShaderGraph_Sprite_Unlit_ThresholdAnim라는 이름의 예시를 찾아보시기 바랍니다.

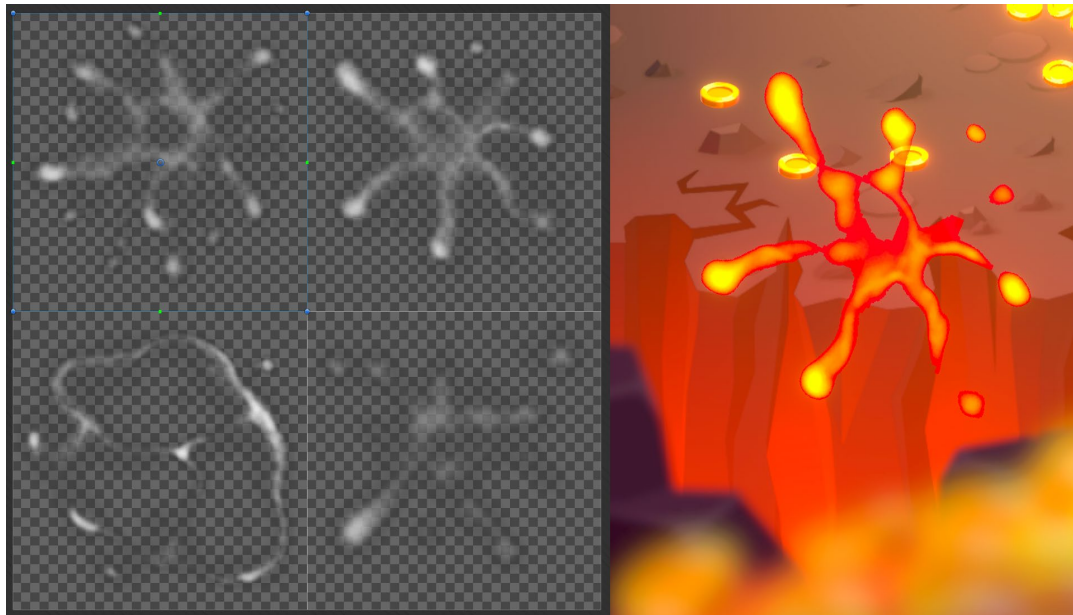


이 효과를 구현하는 데는 Smoothstep 노드가 사용됩니다. 텍스처의 빨간색 채널은 $\ln(1)$ 슬롯과 연결되며, 버텍스 컬러의 알파는 Smoothstep 효과의 가장자리와 평활도를 평가합니다. 버텍스 컬러는 스프라이트 렌더러의 Color 프로퍼티를 통해 제어되므로 스프라이트의 불투명도를 변경하면 텍스처가 애니메이션화됩니다. 그러면 파티클에도 셰이더를 사용할 수 있습니다.

Affinity Photo를 사용하는 경우, 이 셰이더의 텍스처를 흑백으로 칠하고 임계값 조정을 사용해 값 슬라이더를 움직여 애니메이션을 미리 보고 확인하세요.



Affinity Photo에서 임계값 애니메이션 미리 보기



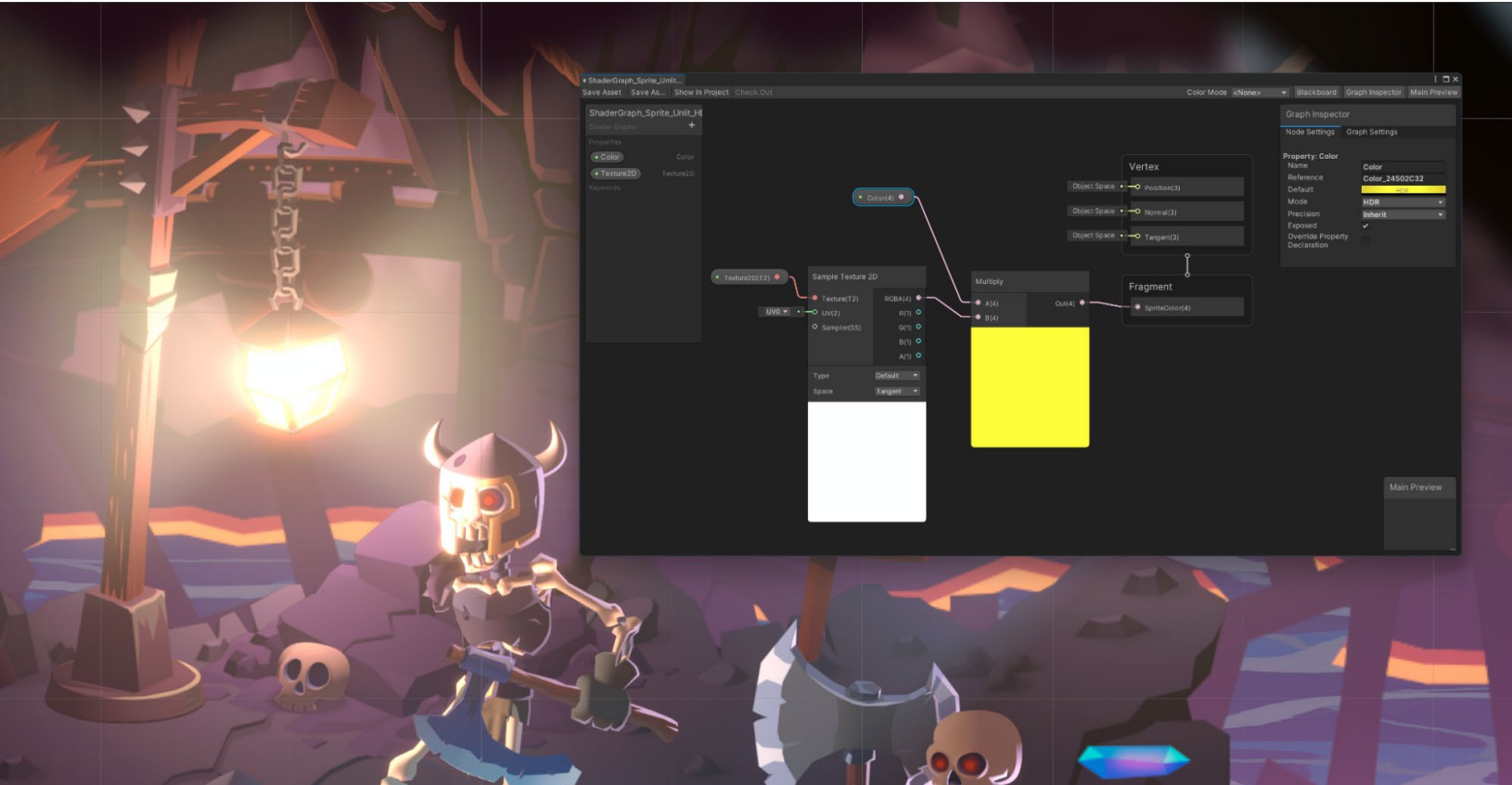
용암 효과에 사용되는 텍스처와 인게임 모습

이 효과를 활용하면 프레임 바이 프레임 애니메이션 없이도 액체나 흩뿌려진 형상을 구현할 수 있습니다. [표면 장력](#)과 관련된 커뮤니티 게시물에서 이 애니메이션에 대해 자세히 알아보세요.



광원에 글로우 추가

드래곤 크래서에 적용된 또 다른 효과는 랜턴에서 나오는 강렬한 글로우입니다. 이러한 효과를 생성하려면 HDR로 향상된 스프라이트 셰이더와 블룸 포스트 프로세싱 효과를 함께 사용합니다.



드래곤 크래서의 HDR 셰이더(ShaderGraph_Sprite_Unlit_HDRInt)

이 효과를 만들려면 HDR 컬러 모드를 사용하는 Color 노드로 텍스처를 곱합니다. Material에서 HDR 값을 1을 초과하는 수로 올리고 메인 카메라의 블룸 포스트 프로세싱 효과를 활성화합니다.

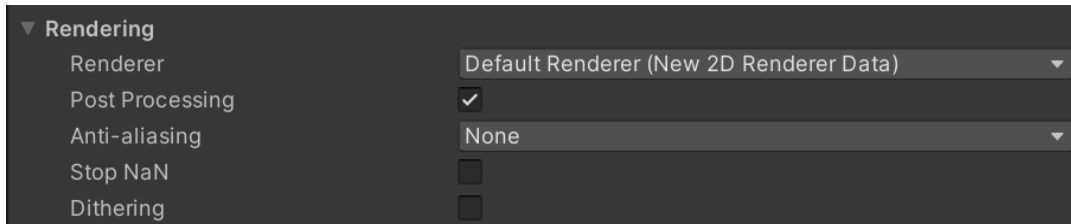
포스트 프로세싱

포스트 프로세싱은 최종 이미지 프레임에 효과를 추가하여 게임의 그래픽을 향상합니다. 실제 비디오 카메라로 찍은 듯한 모습을 시뮬레이션하거나, 완전히 스타일라이즈드 형식으로 연출할 수도 있는 효과입니다. URP에는 [포스트 프로세싱 기능](#)이 포함되어 있으므로 별도의 패키지를 설치할 필요가 없으며 효과를 설정하는 과정도 매우 간단합니다.



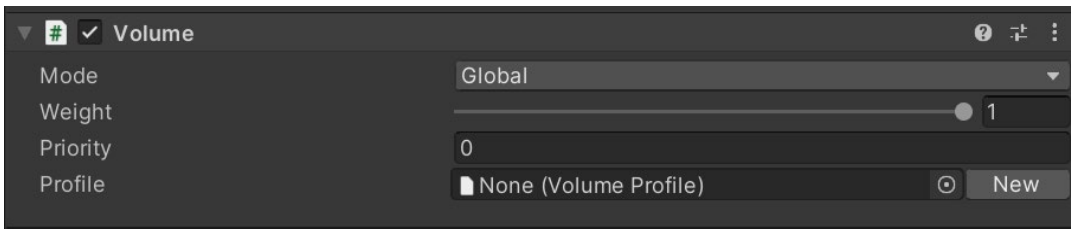
왼쪽은 포스트 프로세싱이 적용되지 않은 카메라 출력 이미지입니다. 오른쪽은 포스트 프로세싱이 추가된 동일한 이미지입니다. 차이점이 두드러지도록 효과가 과하게 적용되었습니다. 특히 모바일 게임에는 포스트 프로세싱 효과를 주의해서 적용하세요.

시작하려면 메인 카메라를 선택하고 Post Processing 옵션을 선택합니다.



메인 카메라에서 Post Processing 활성화

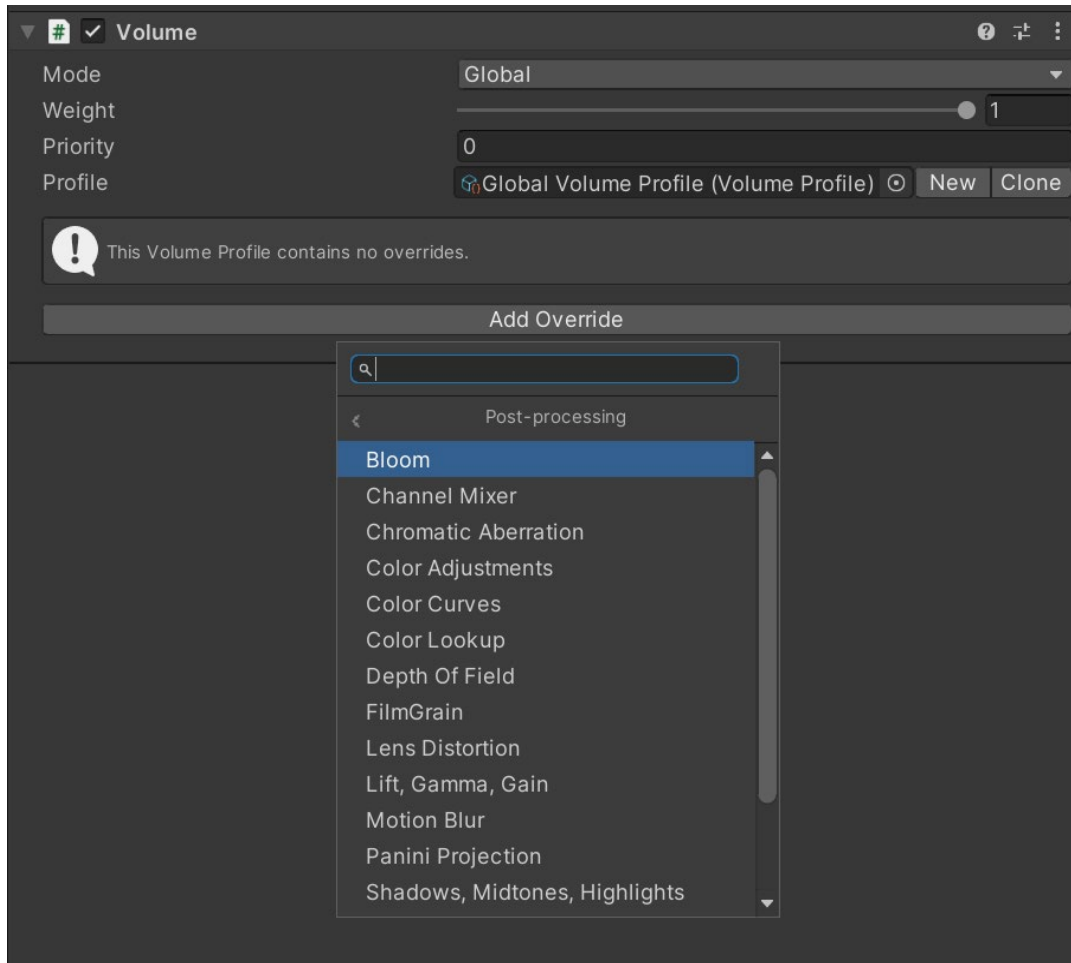
포스트 프로세싱은 **볼륨** 시스템을 사용하므로 새로운 포스트 프로세싱 볼륨을 추가하려면 **GameObject > Volume > Global Volume**을 선택합니다.



볼륨 프로파일 정의

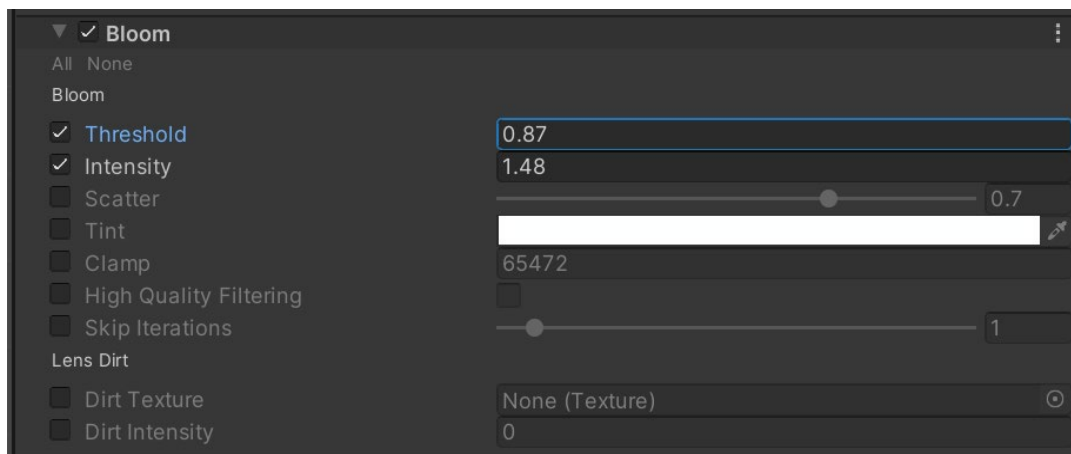
이제 New 버튼을 클릭하여 볼륨 프로파일을 만듭니다. 볼륨 프로파일은 포스트 프로세싱 효과를 저장하며 볼륨과 씬 간에 쉽게 공유될 수 있습니다.

포스트 프로세싱 효과를 추가하려면 Add Override 버튼을 클릭한 후 목록에서 원하는 효과를 선택합니다.



블룸 포스트 프로세싱 효과 선택

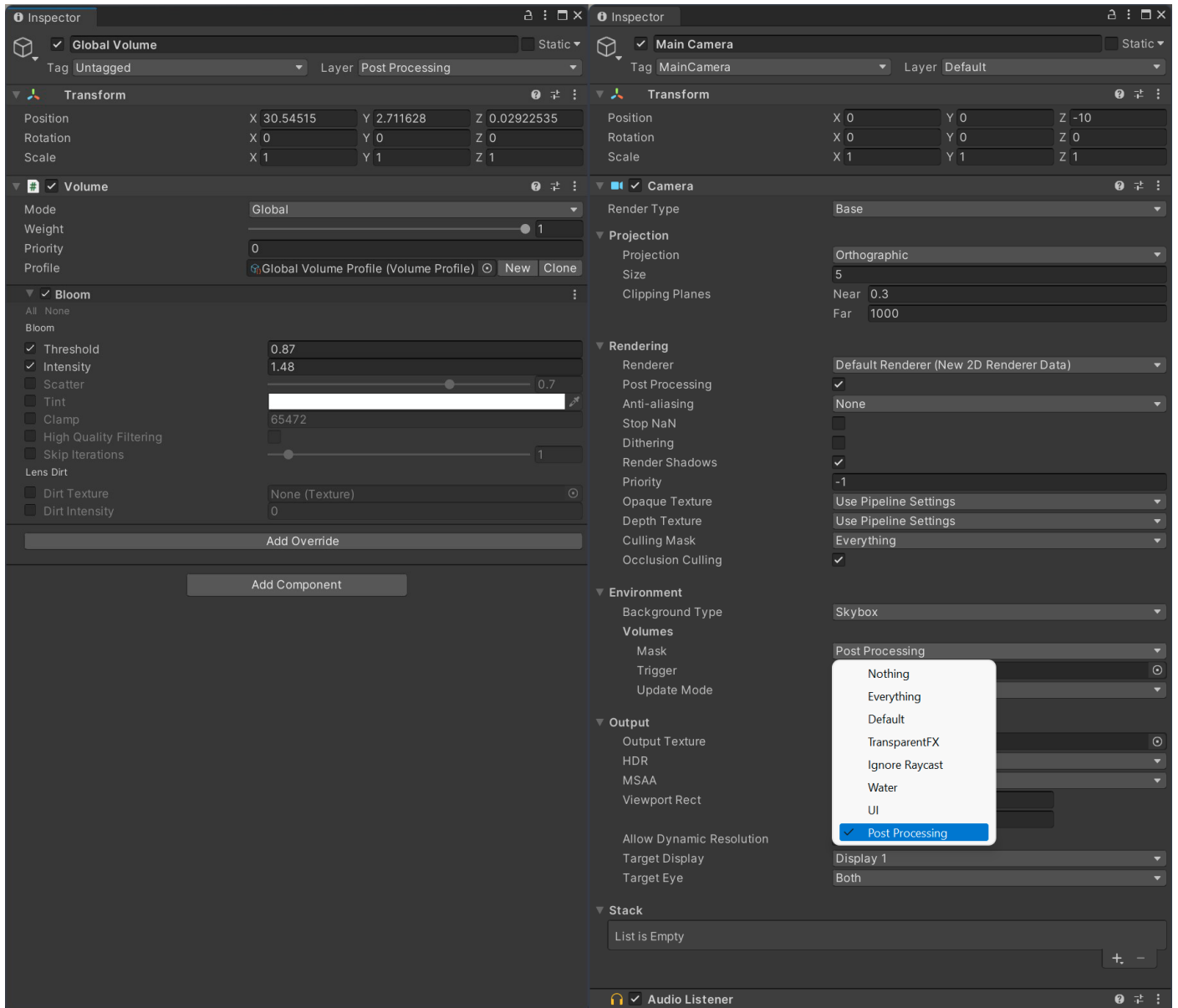
Bloom을 먼저 선택해 보겠습니다.



블룸 효과 프로퍼티 정의

효과의 프로퍼티를 조정하려면 그 옆에 있는 체크박스를 선택합니다.

참고로 Global Volume의 게임 오브젝트 레이어는 Main Camera의 Mask 옵션에서 선택한 레이어와 일치해야 합니다.



카메라의 Post Processing 체크박스를 활성화하고 카메라의 Volumes Mask 드롭다운 목록에서 볼륨의 게임 오브젝트 레이어를 선택합니다.

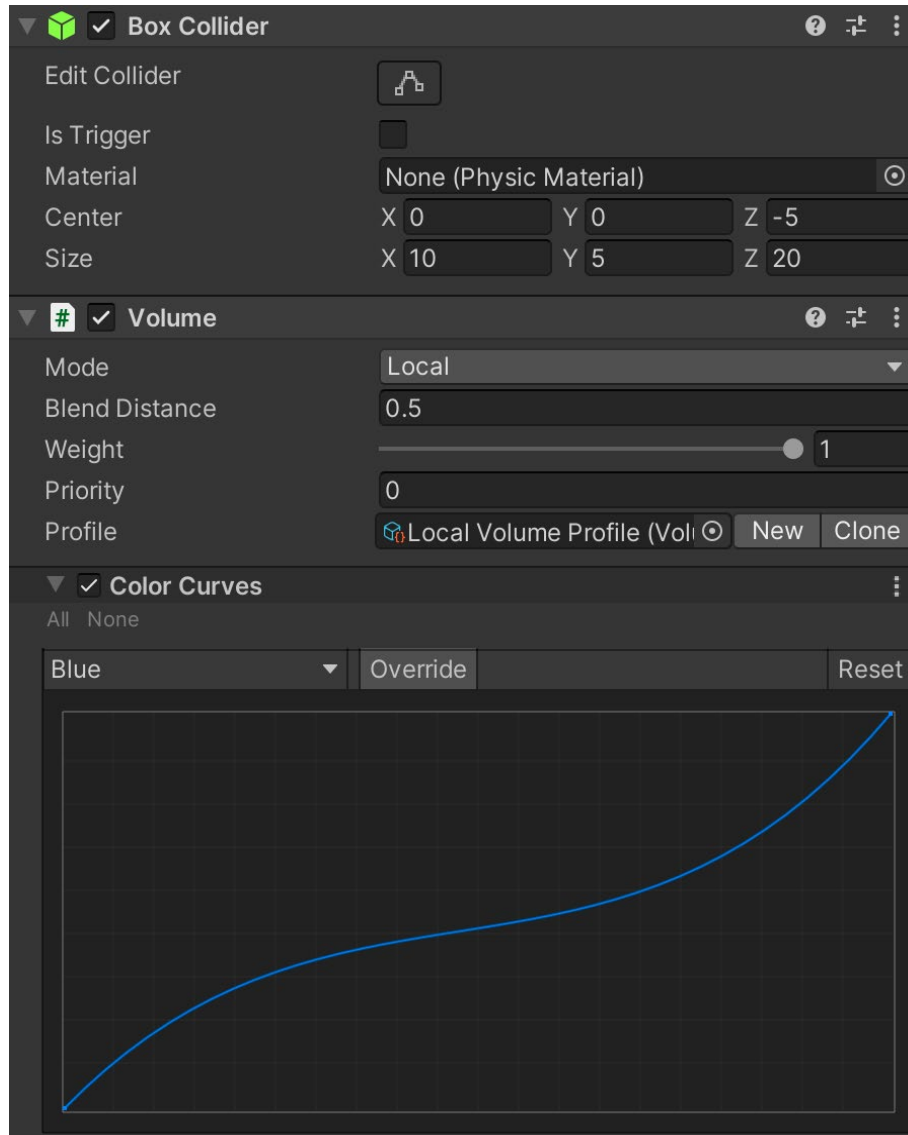
모든 요소가 올바르게 작용하면 블룸 효과가 게임 뷰에 표시됩니다.

Global Volume이 사용되었으므로 씬에서 볼륨 오브젝트가 활성화 상태일 때는 블룸 효과도 항상 표시됩니다.

로컬 볼륨

포스트 프로세싱 효과를 레벨 전체에 추가할 수 있지만 플레이어가 건물에 진입할 때처럼 레벨의 일부분에 효과를 설정할 수도 있습니다.

이 작업을 위해서는 다른 포스트 프로세싱 프로파일이 연결된 또 다른 볼륨 오브젝트를 생성해야 합니다.



볼륨 오브젝트에 3D 박스 콜라이더 추가

Volume의 Mode를 Local로 설정한 후 볼륨 오브젝트에 3D 박스 콜라이더를 추가해야 합니다. 카메라가 박스 콜라이더 경계에 진입할 때 포스트 프로세싱이 적용되므로 Z축이 카메라 오브젝트가 담길 만큼 커야 합니다. 기본적으로 콜라이더는 매우 작으므로 씬 뷰에서 크기를 재차 확인하세요.

Blend Distance 값을 늘리면 카메라가 볼륨 영역에 진입하는 순간 급작스럽게 효과가 켜지는 현상을 방지할 수 있습니다.



unity.com/kr