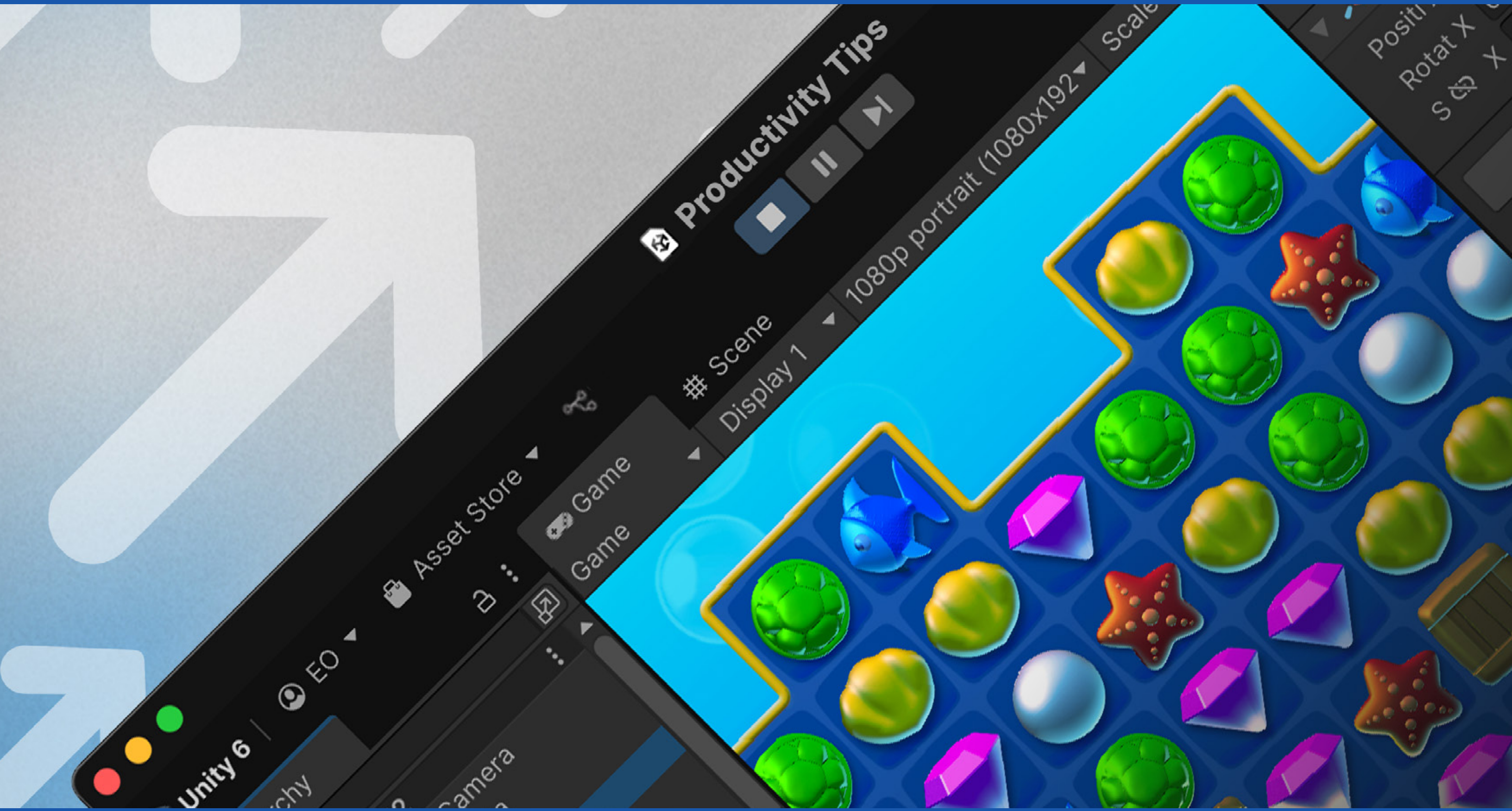
→ E-BOOK

# Tips to increase productivity with Unity 6



# Contents

# Introduction

This guide provides numerous tips on how to work faster with Unity's toolsets, regardless if you're an artist or a programmer. It covers new features in Unity 6 along with time-saving steps and workflows that have been a part of Unity for years.

When you work in Unity every day, whether you work on your own or in a team, each second or mouse click adds up. We want you to be able to waste less time and be more productive. Whether you're a new or experienced Unity developer, this guide can help you speed up workflows in every stage of your game development.

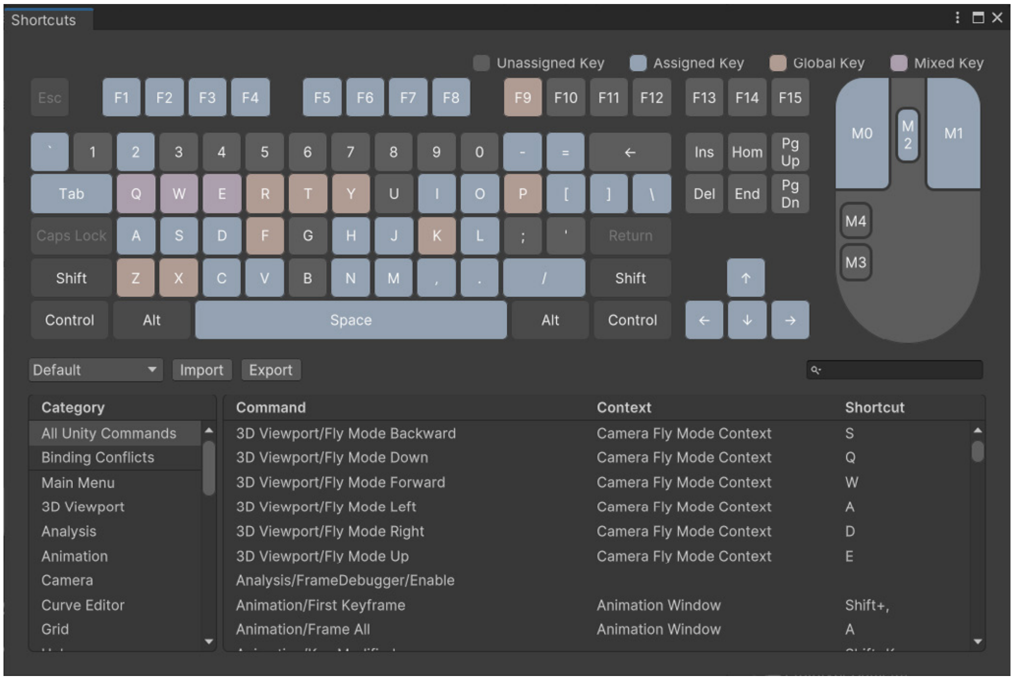
Go to the Unity 6 web page, or to the New in Unity 6.0 and New in Unity 6.1 sections of the Unity documentation to get the full story of its new features and capabilities.

We hope you find this guide helpful.

# Editor workflows

## Customize your preferred keyboard shortcuts easily

The Shortcuts Manager is an interactive visual interface to help you view, customize, and manage Editor hotkeys. It enables you to assign shortcuts to different contexts such as Global, Scene view, or specific Editor tools, and visualize existing bindings for any tools that you use frequently. You can also import/export shortcut profiles across the team or devices.



| Category |
|---|
| All Unity Commands |
| Binding Conflicts |
| Main Menu |
| 3D Viewport |
| Analysis |
| Animation |
| Camera |
| Curve Editor |
| Grid |

| Command | Context | Shortcut |
|---|---|---|
| 3D Viewport/Fly Mode Backward | Camera Fly Mode Context | S |
| 3D Viewport/Fly Mode Down | Camera Fly Mode Context | Q |
| 3D Viewport/Fly Mode Forward | Camera Fly Mode Context | W |
| 3D Viewport/Fly Mode Left | Camera Fly Mode Context | A |
| 3D Viewport/Fly Mode Right | Camera Fly Mode Context | D |
| 3D Viewport/Fly Mode Up | Camera Fly Mode Context | E |
| Analysis/FrameDebugger/Enable | | |
| Animation/First Keyframe | Animation Window | Shift+, |
| Animation/Frame All | Animation Window | A |

The Shortcuts Manager

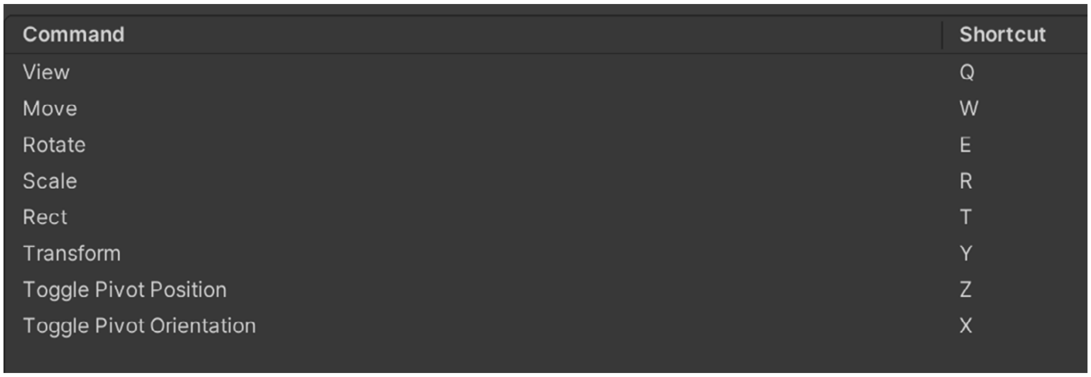
Access the Shortcuts Manager from Unity’s main menu:

- On Windows and **Linux**, select **Edit > Shortcuts**.
- On macOS, select **Unity > Shortcuts**.

**Common Shortcuts**

Here are some common default shortcuts:

| Action | Windows | Mac |
|---|---|---|
| Frame Selected | F | |
| Duplicate Items | Ctrl + D | ⌘ + D |
| Delete GameObject | Shift + Del | ⌘ + Del |
| View/Move/Rotate/Rect/Transform | Q W E R T | |
| Toggle Pivot Mode | Z | |
| Toggle Pivot Rotation | X | |
| Vertex Snap | V | |
| Snap | Ctrl + 🖱 | ⌘ + 🖱 |
| Isolate selected objects in the Scene view | Shift + H | |
| Toggle Maximize | Shift + Space | |
| Edit Prefab in Context | P | |



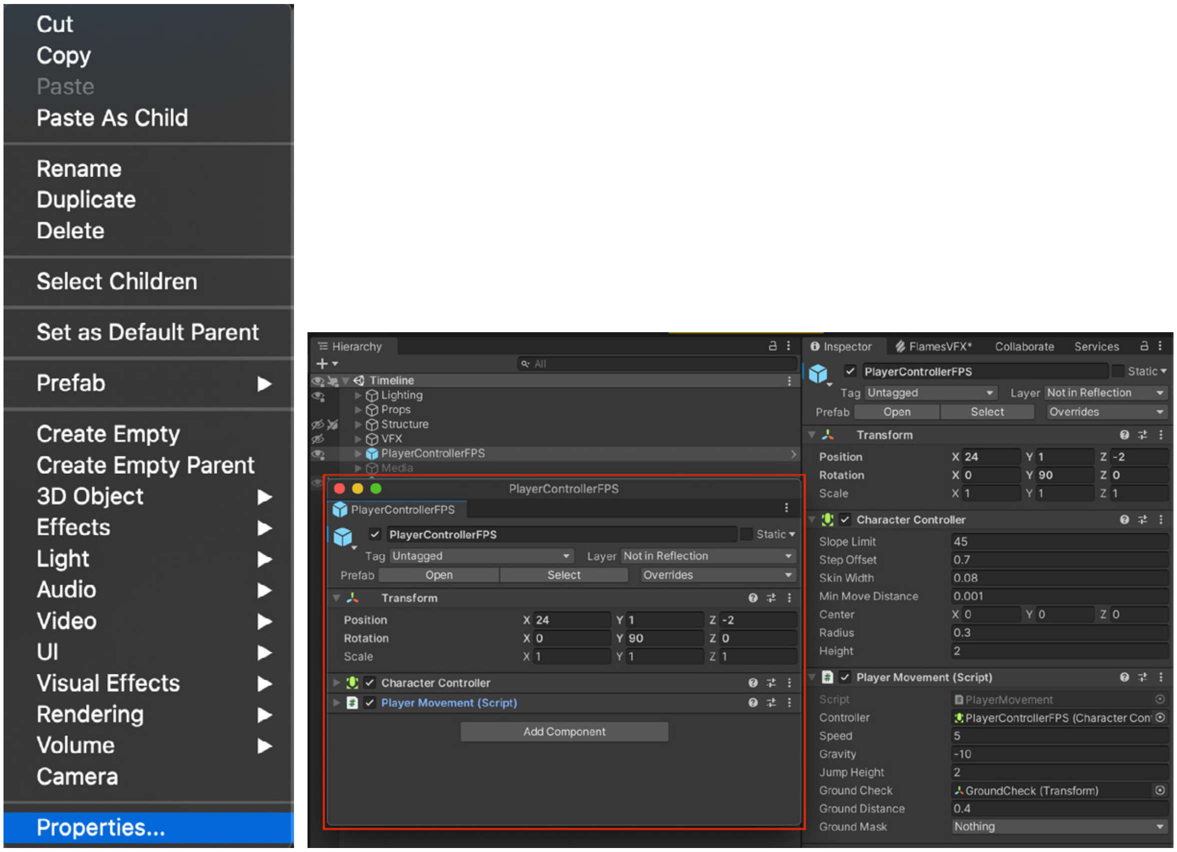
| Command | Shortcut |
|---|---|
| View | Q |
| Move | W |
| Rotate | E |
| Scale | R |
| Rect | T |
| Transform | Y |
| Toggle Pivot Position | Z |
| Toggle Pivot Orientation | X |

Common Editor shortcuts

## Work with multiple Inspectors

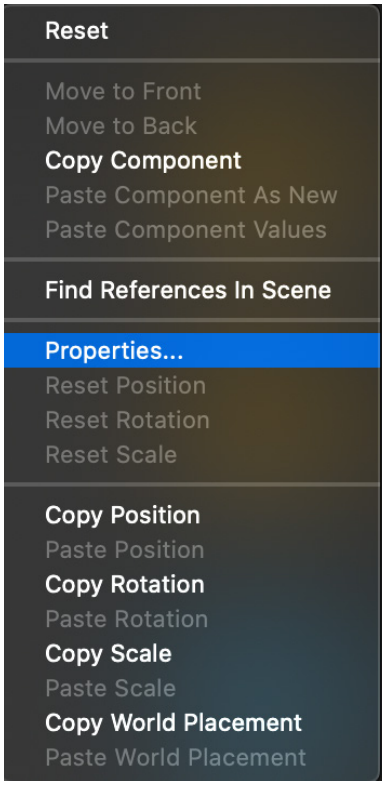
The Focused Inspector window allows you to have multiple Inspector windows at the same time to display the properties for a specific GameObject, component, or asset. It always displays the properties of the item you opened it for, even if you select something else in the Scene.

Right-click on a GameObject or component, and choose **Properties**. This reveals a floating Inspector window that you can reposition, dock, or resize like any other window.



PlayerControllerFPS
Static
Tag Untagged
Layer Not in Reflection
Transform
Position X 24 Y 1 Z -2
Rotation X 0 Y 90 Z 0
Scale X 1 Y 1 Z 1
Character Controller
Slope Limit 45
Step Offset 0.7
Skin Width 0.08
Min Move Distance 0.001
Center X 0 Y 0 Z 0
Radius 0.3
Height 2
Player Movement (Script)
Script PlayerMovement
Controller PlayerControllerFPS (Character Con
Speed 5
Gravity -10
Jump Height 2
Ground Check GroundCheck (Transform)
Ground Distance 0.4
Ground Mask Nothing

A Focused Inspector comparing two GameObjects



You can also focus on a specific component of a GameObject, requiring less screen space.

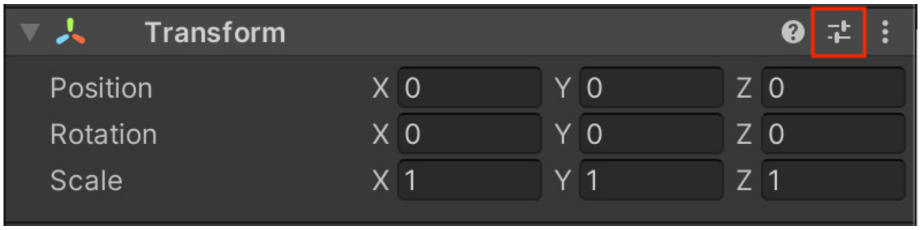
Opening multiple Focused Inspectors at the same time allows you to reference multiple GameObjects while making changes to the Scene or make easy side-by-side comparisons.

# Define your own preferred default settings with Presets

Presets enable you to customize and reuse the default state of components or assets in your Inspector. When you create a Preset, Unity captures the current settings of a component or asset and saves them as a **.preset asset**.

You can then apply this configuration to other components or assets of the same type.

Use Presets to enforce standards or to apply reasonable defaults to new assets. This ensures consistent standards across your team, so commonly overlooked settings don't impact your project's performance.

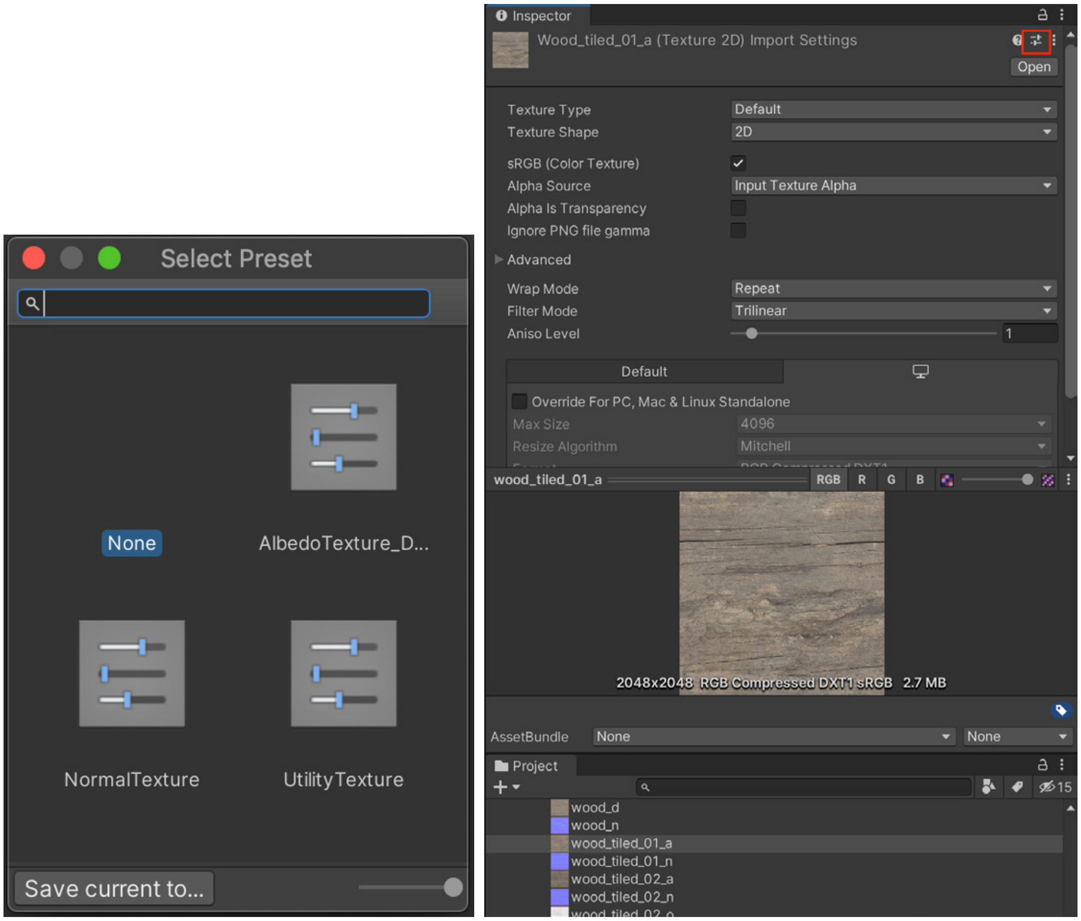


| | | | | | | |
|---|---|---|---|---|---|---|
| Position | X | 0 | Y | 0 | Z | 0 |
| Rotation | X | 0 | Y | 0 | Z | 0 |
| Scale | X | 1 | Y | 1 | Z | 1 |

The Preset icon is highlighted here in red.

To create a preset:

1. Click the **Preset** icon to the top right of the component.
2. Click **Save current to…** to save the Preset as an asset.
3. Click one of the available Presets to load a set of values.



Wood_tiled_01_a (Texture 2D) Import Settings

Texture Type: Default
Texture Shape: 2D
sRGB (Color Texture): ☑
Alpha Source: Input Texture Alpha
Alpha Is Transparency: ☐
Ignore PNG file gamma: ☐
▶ Advanced
Wrap Mode: Repeat
Filter Mode: Trilinear
Aniso Level: 1

☐ Override For PC, Mac & Linux Standalone
Max Size: 4096
Resize Algorithm: Mitchell

wood_tiled_01_a
2048x2048 RGB Compressed DXT1 sRGB 2.7 MB

AssetBundle: None None

In this example, the Presets contain different Import Settings for 2D textures depending on usage (albedo, normal, or utility).

Other practical ways to use Presets:

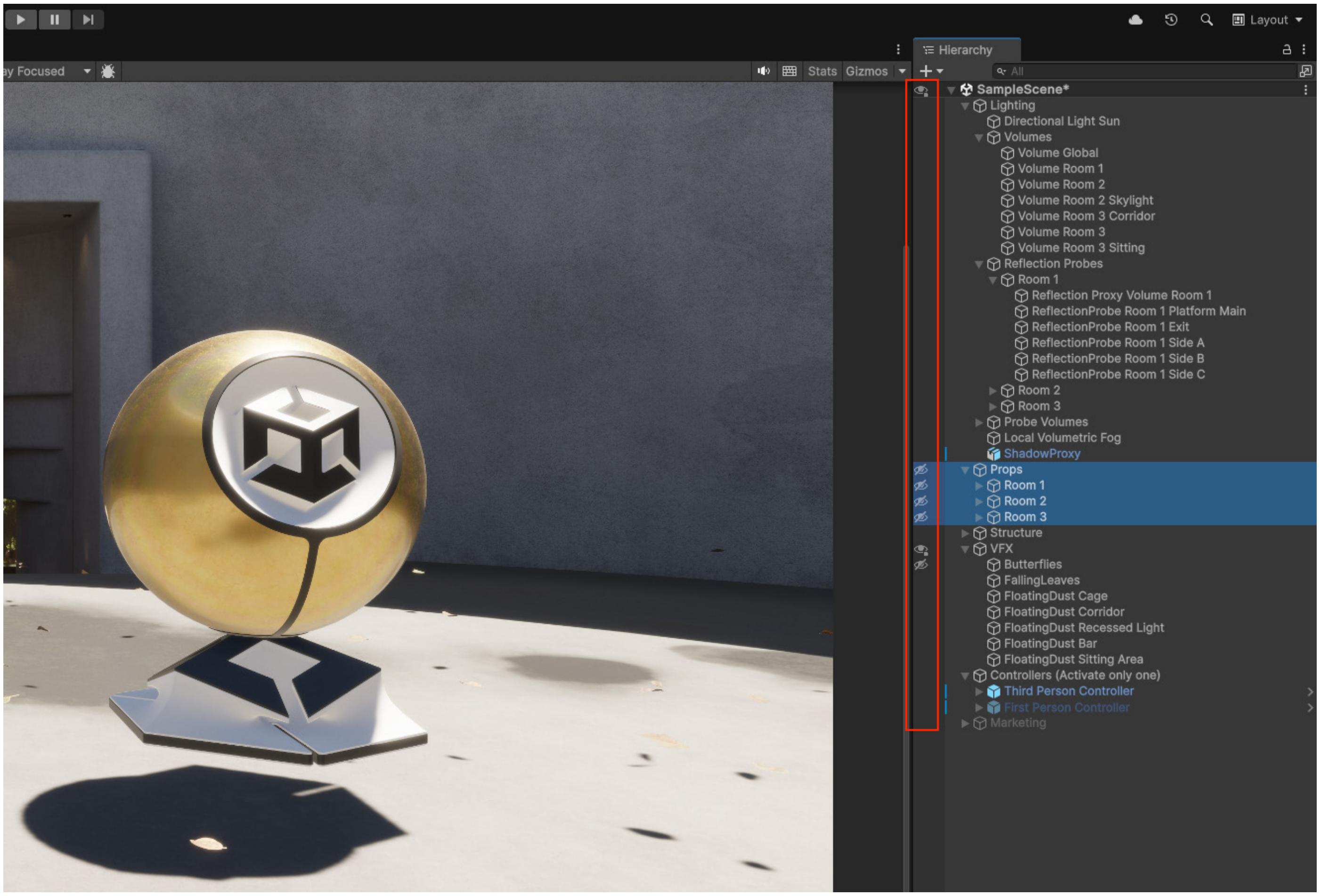
- **Create a GameObject with default values:** Drag and drop a Preset asset into the Hierarchy window to create a new GameObject with the corresponding component filled in with Preset values.
- **Associate a specific Type with a Preset:** In the **Preset Manager** (**Project Settings > Preset Manager**), specify one or more Presets per Type. Creating a new component will then default to the specified Preset values.
- **Use filters to select among multiple Presets:** You can create multiple Presets for the same Type and use the filters) in the Preset Manager to determine which Preset is applied.

- **Save and reuse manager settings:** Use Presets for a Manager window, so the settings can be reused; for example, if you plan to reapply the same Tags and Layers or Physics settings. Presets can reduce setup time for your next project.

## Unclutter your scene view with Scene visibility

As your Scene grows larger, you can temporarily hide specific objects to make selecting and editing your GameObjects easier.

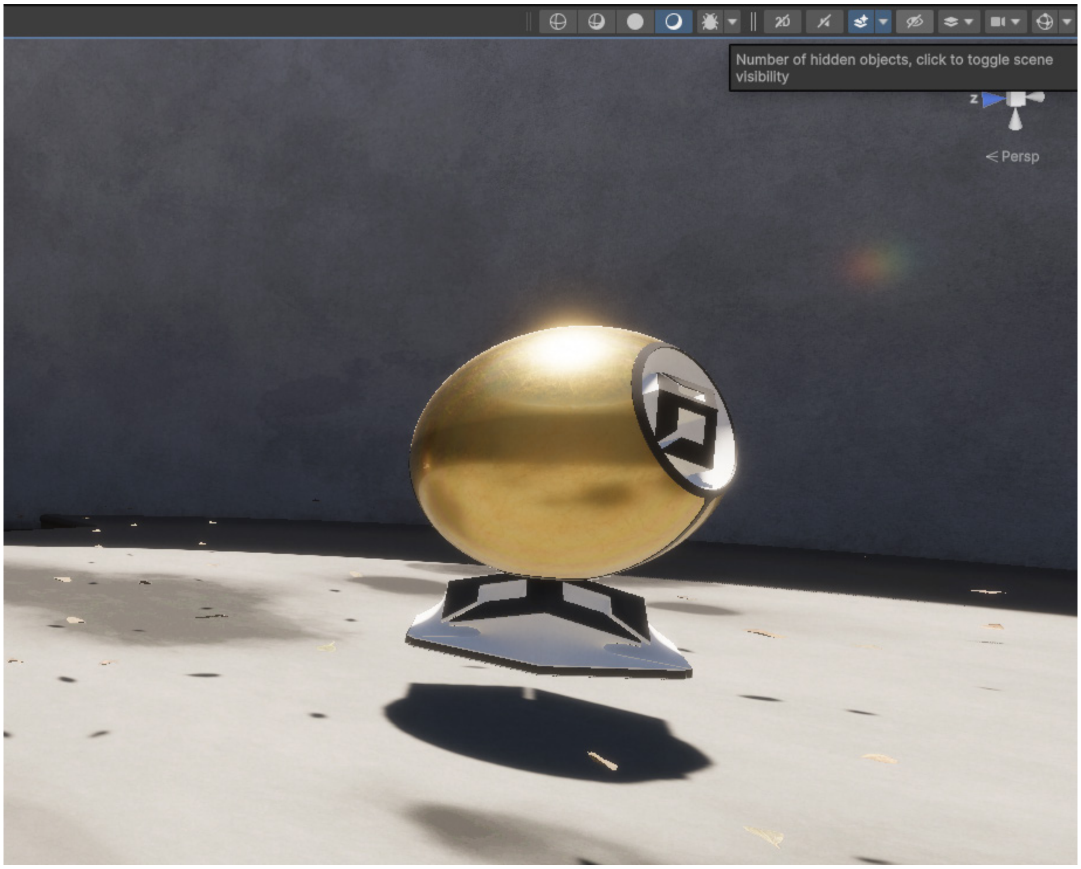
Instead of deactivating the GameObjects (which can lead to unintended behavior), you can use Scene visibility controls to toggle visibility in the Scene view. Click the eye icon next to a GameObject in the **Hierarchy** window to toggle Scene visibility. This hides the object from the Scene view without affecting its active state or in-game visibility.



Hide objects in the Scene view using Scene visibility controls.

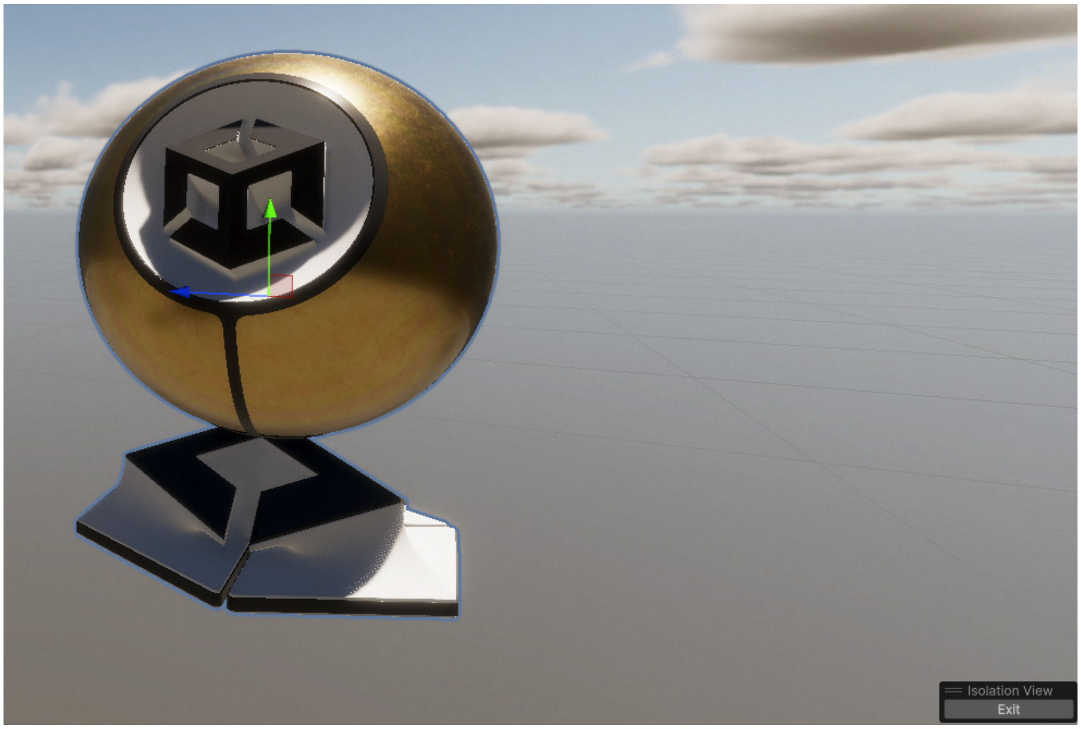
Note that the status icons may change in the Hierarchy, depending on whether parent or child objects become hidden.

| Icon | Status |
|---|---|
| 👁 | The GameObject is visible, but some of its children are hidden. |
| 👁̸ | The GameObject is hidden, but some of its children are visible. |
| 👁 | The GameObject and its children are visible, but they only appear when you hover over the GameObject. |
| 👁̸ | The GameObject and its children are hidden. |



Toggle the Scene view control bar on or off to override the global Scene visibility.

You can also use the **Isolation View** to concentrate on a specific object and its children. Select the GameObject in the Hierarchy window and press **Shift + H** to toggle it on and off. This overrides your other Scene visibility settings until you exit.



Isolation View allows you to edit a GameObject without distractions.

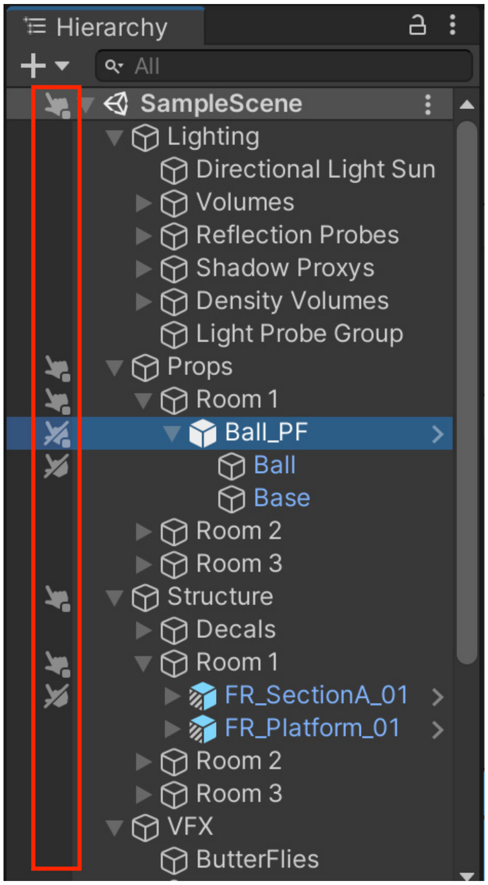
Remember that you can always use the **Shift + spacebar** shortcut to maximize the viewport and hide the rest of the Editor as well.

You can also work with multiple scenes in Unity if you need to create large streaming worlds, or want to effectively manage multiple scenes at runtime. You can bake data in multiple scenes simultaneously, such as lightmaps, NavMesh data, and occlusion culling data. Also, you can edit multiple scenes using scripts within the Editor or at runtime.

## Avoid selecting the wrong objects in your Scene view

Unity allows you to control whether specific GameObjects can be selected in the Scene view, similar to how you manage Scene visibility.

You can toggle the Pickability state of a GameObject using the Scene visibility toolbar to block specific GameObjects from being selected in the Scene view. This is useful to avoid selecting and editing an unintended GameObject in large and complex scenes.



Hierarchy pickability

Because you can toggle pickability for a whole branch or a single object, some GameObjects may be pickable but have children or parents that are not. The following icons differentiate their status.

| Icon | Status |
|---|---|
| 👆● | You can pick the GameObject, but you cannot pick some of its children. |
| ⊘● | You cannot pick the GameObject, but you can pick some of its children. |
| 👆 | You can pick the GameObject and its children (only appears when you hover over the GameObject). |
| ⊘ | You cannot pick the GameObject or its children. |

# Preview your assets directly in the Inspector

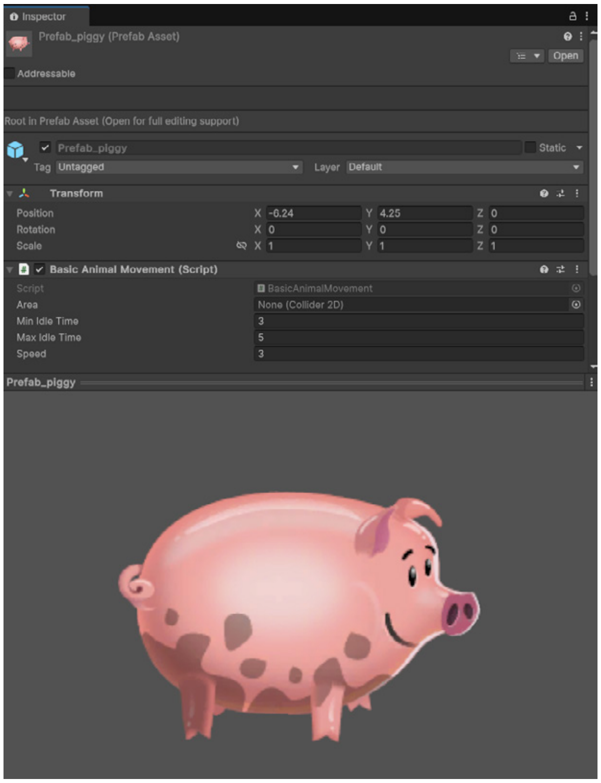
The Asset Preview is the visual panel in the Unity Editor, found in the bottom of the Inspector window, that allows you to see a preview or thumbnail of an asset, such as 3D models, materials, or textures. It provides a quick, interactive way to evaluate assets visually while working in the Editor.

For 3D assets, like models or prefabs, you can click and drag on the Asset Preview at the bottom of the Inspector window to rotate the object and inspect it from different angles. Use this to quickly check the asset's geometry, materials, or orientation without adding it to a scene.

If the Asset Preview appears too small, you can resize it by dragging the separator above the preview area in the Inspector window.

For assets like animated models, the Asset Preview often shows a simple default animation if one exists on the asset. Use this to quickly assess motion and rigging.

Unity can also display thumbnails (smaller versions of the Asset Preview) directly in the **Project** window. Enable this by switching your **Project** window to the **Two Column Layout** and adjusting the preview slider at the bottom. This makes it easier to visually browse and organize assets.



Prefab_piggy (Prefab Asset)

□ Addressable

Root in Prefab Asset (Open for full editing support)

☑ Prefab_piggy □ Static

Tag Untagged Layer Default

Transform

| | | | |
|---|---|---|---|
| Position | X -6.24 | Y 4.25 | Z 0 |
| Rotation | X 0 | Y 0 | Z 0 |
| Scale | X 1 | Y 1 | Z 1 |

☑ Basic Animal Movement (Script)

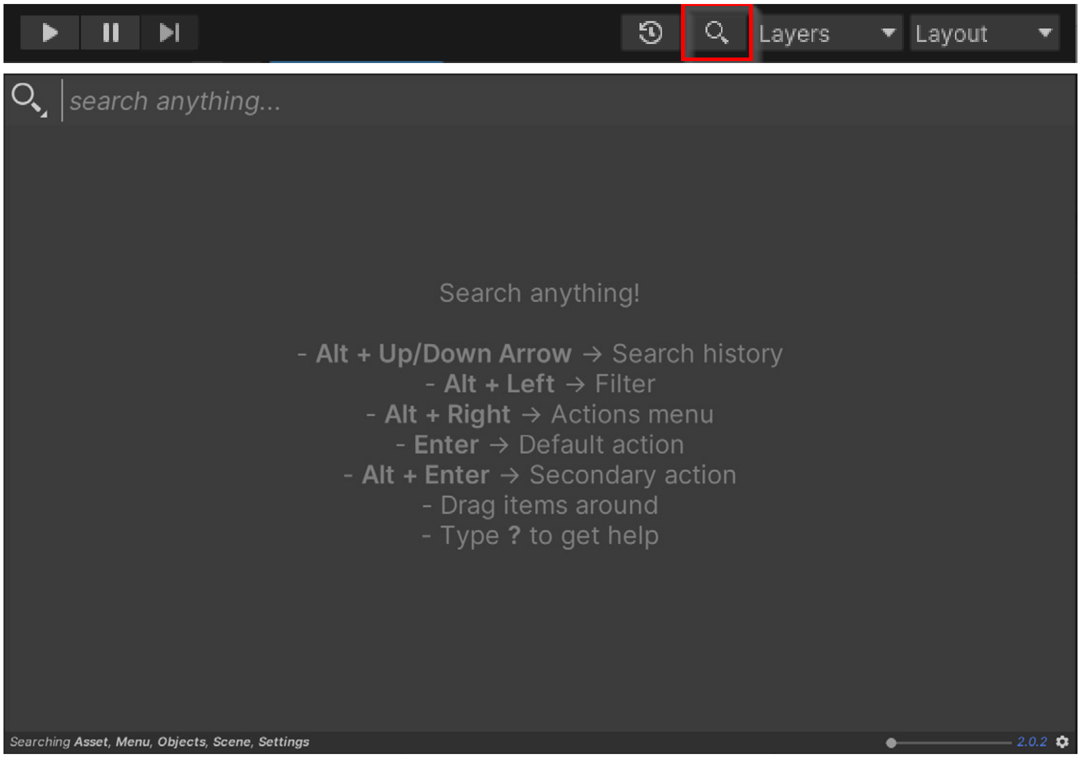
| | |
|---|---|
| Script | BasicAnimalMovement |
| Area | None (Collider 2D) |
| Min Idle Time | 3 |
| Max Idle Time | 5 |
| Speed | 3 |

Prefab_piggy

The Asset Preview window in the *Happy Harvest* sample

# Speed up your search in projects

There are several ways for you to search efficiently in Unity across assets, scene objects, menu items, packages, APIs, settings, etc. Besides the search functionality in the Hierarchy and Project views, you can also use the search button icon in the main menu bar or use the hotkey Ctrl + K on Windows or Cmd + K on macOS to activate it.



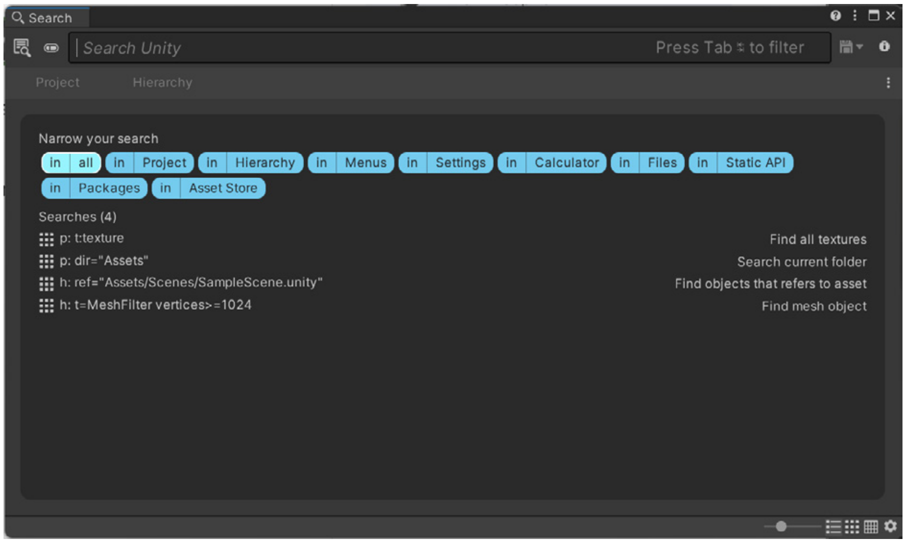
Search anything!

- **Alt + Up/Down Arrow** → Search history
- **Alt + Left** → Filter
- **Alt + Right** → Actions menu
- **Enter** → Default action
- **Alt + Enter** → Secondary action
- Drag items around
- Type **?** to get help

*Searching **Asset, Menu, Objects, Scene, Settings***

Use the hotkey or Help menu to launch QuickSearch.

This opens a search window where it's easy to filter your search.



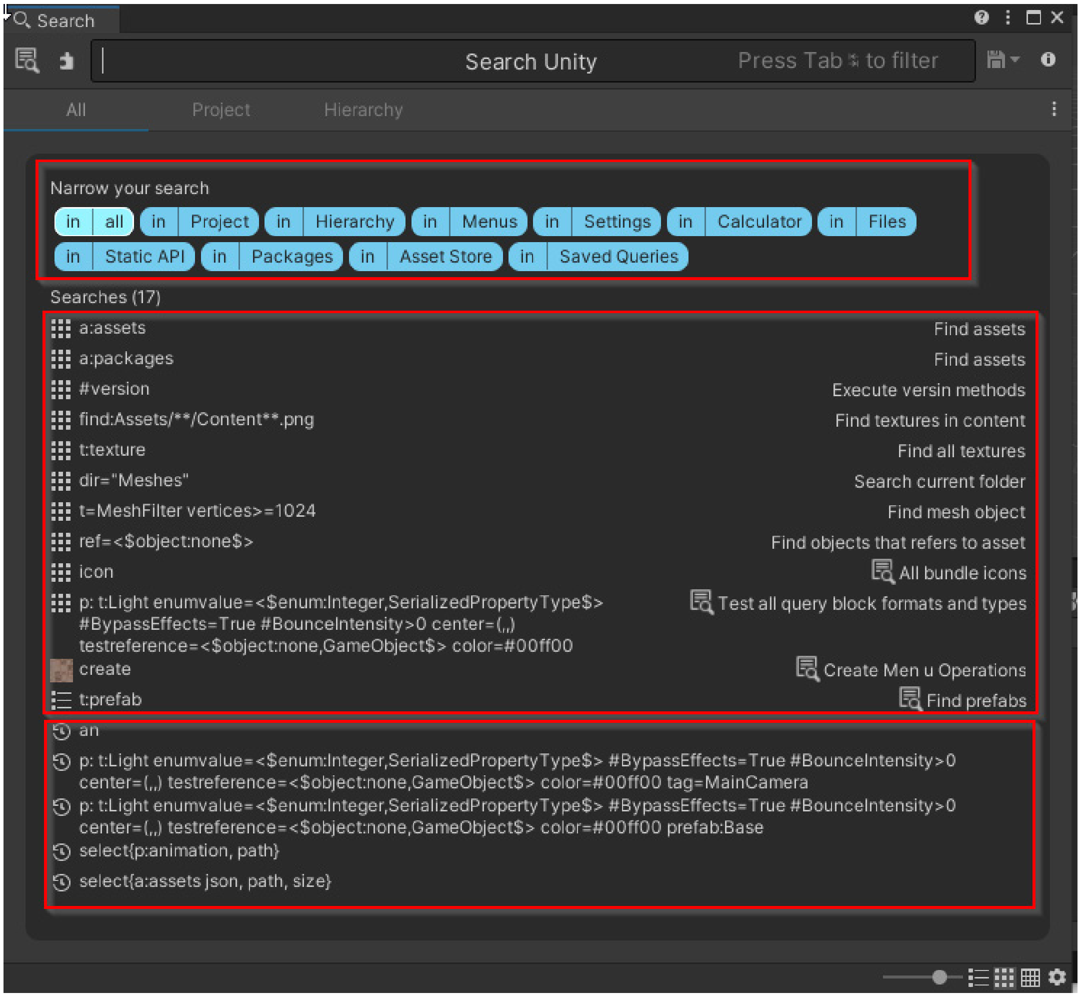
Narrow your search

in all | in Project | in Hierarchy | in Menus | in Settings | in Calculator | in Files | in Static API | in Packages | in Asset Store

Searches (4)

```
p: t:texture                                    Find all textures
p: dir="Assets"                                 Search current folder
h: ref="Assets/Scenes/SampleScene.unity"        Find objects that refers to asset
h: t=MeshFilter vertices>=1024                  Find mesh object
```

The Search window



Searches (17)

| | |
|---|---|
| a:assets | Find assets |
| a:packages | Find assets |
| #version | Execute versin methods |
| find:Assets/**/Content**.png | Find textures in content |
| t:texture | Find all textures |
| dir="Meshes" | Search current folder |
| t=MeshFilter vertices>=1024 | Find mesh object |
| ref=<$object:none$> | Find objects that refers to asset |
| icon | All bundle icons |
| p: t:Light enumvalue=<$enum:Integer,SerializedPropertyType$> #BypassEffects=True #BounceIntensity>0 center=(,,) testreference=<$object:none,GameObject$> color=#00ff00 | Test all query block formats and types |
| create | Create Men u Operations |
| t:prefab | Find prefabs |

an

p: t:Light enumvalue=<$enum:Integer,SerializedPropertyType$> #BypassEffects=True #BounceIntensity>0 center=(,,) testreference=<$object:none,GameObject$> color=#00ff00 tag=MainCamera

p: t:Light enumvalue=<$enum:Integer,SerializedPropertyType$> #BypassEffects=True #BounceIntensity>0 center=(,,) testreference=<$object:none,GameObject$> color=#00ff00 prefab:Base

select{p:animation, path}

select{a:assets json, path, size}

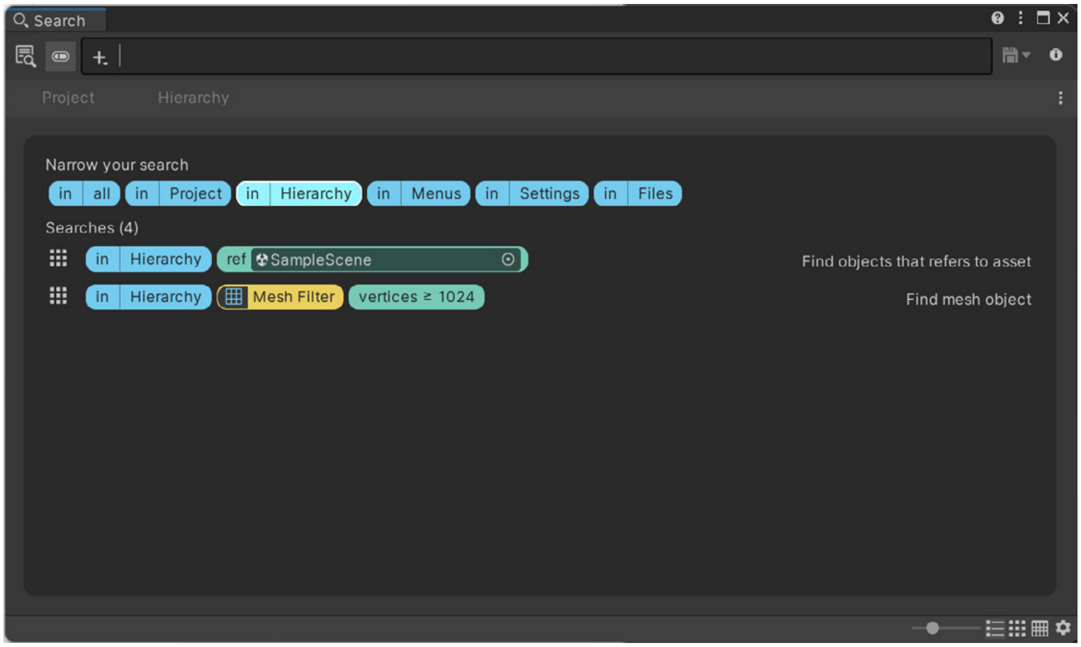
The middle section shows a list of queries available for the currently selected search area. You can click on any query to execute it. When searching for names, you can also search by type. Use the dropdown to select **Type** or the **t:** shorthand syntax, e.g., t:scene (to search all scenes) or t:texture (to search all textures).

Additionally, the Search window lets you visualize objects in various ways: Compact list view, big list view or multiple sizes of grid icons. You can also display objects in a table.

See the QuickSearch guide to learn more about searching both inside and outside of Unity.

## Use the Query Builder to create custom search filters

If you're working with complicated search queries and you need more help when exploring your project, use the Query Builder workflow. It can be activated with the builder toggle (see puzzle button next to the Search Field).

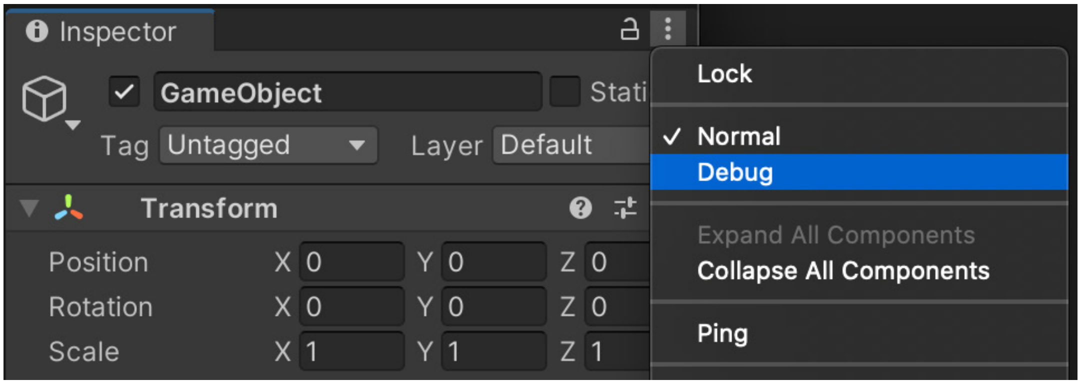


The Query Builder

Save time by creating and saving custom queries for common tasks. For example, if you frequently need to find prefabs by certain tags or locate unused assets, define a reusable query to automate this process. The Query Builder is also integrated with the Editor, and no longer a package, so that you can streamline workflows for your development team.

# Expose debug data in the Inspector

You can toggle each GameObject's Inspector between Normal and Debug mode. Click the **More Items** (⋮) button to open the context menu and choose the desired mode. When using the Debug mode, all serialized fields, including private ones, hidden references, and internal Unity data are exposed in the Inspector. This can be useful for troubleshooting or inspecting components beyond their standard public interface.

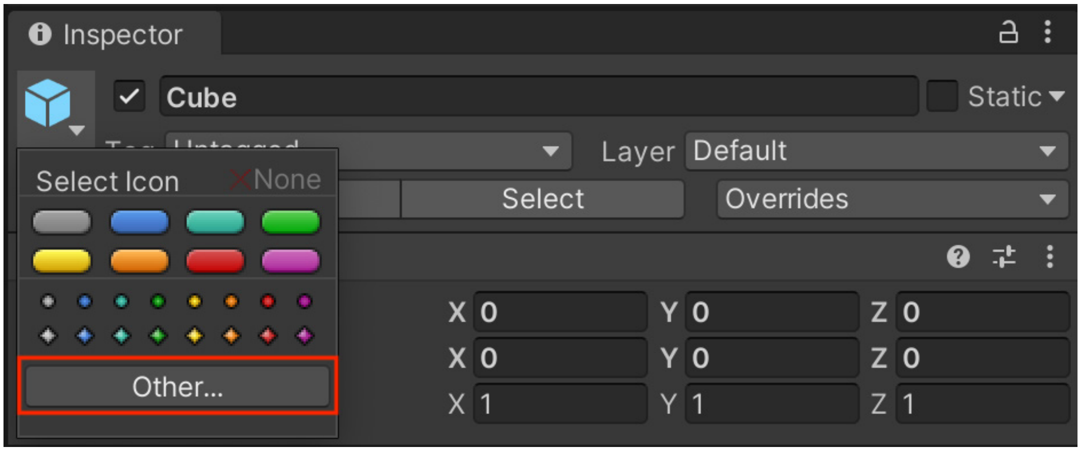


Inspector Debug mode

# Use Custom Gizmos and icons for visual debugging

Gizmos are visual overlays that appear in the Scene view, helping you locate and identify GameObjects during development. They can be especially useful for visualizing non-rendered elements like triggers, spawn points, or scripts.

You can modify the icons for a GameObject using the Select Icon menu to your liking. Choose **Other** to define your own icon.



Cube

Layer Default

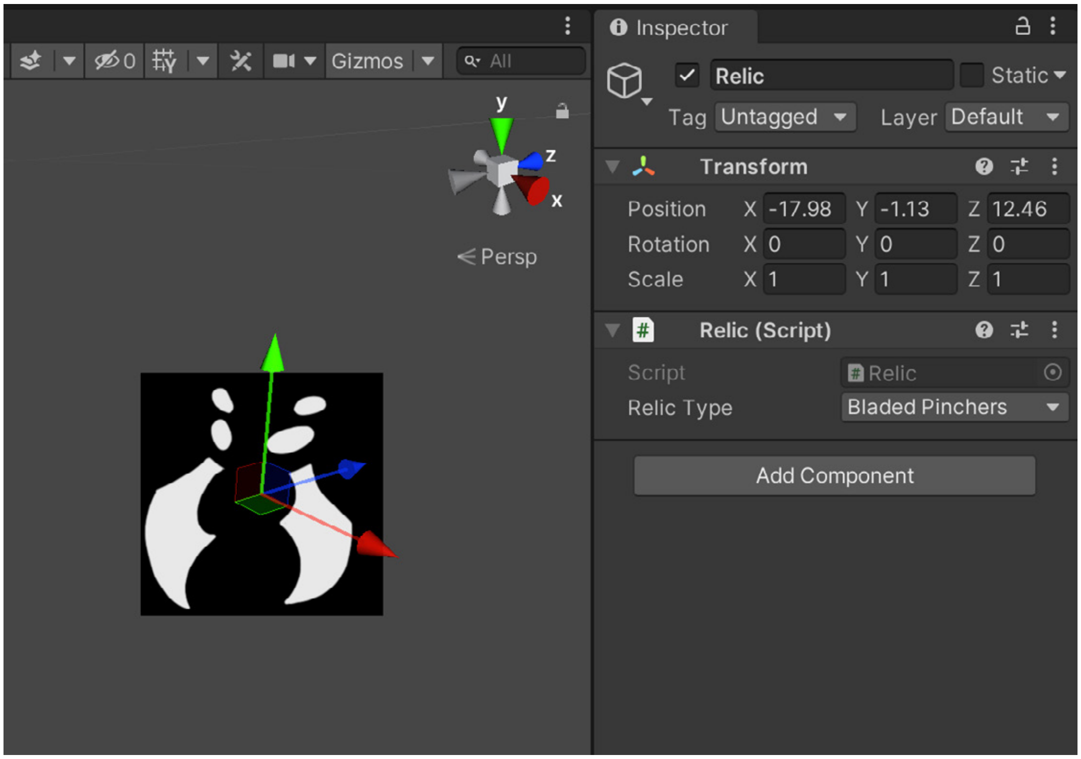
X 0 Y 0 Z 0

X 0 Y 0 Z 0

X 1 Y 1 Z 1

Use the drop-down in the Inspector to switch gizmos.

You can also create gizmos with scripts and make them interactive. For example, a gizmo could help you define a volume or area of influence for a custom component.



Relic

Tag Untagged Layer Default

| | X | Y | Z |
|---|---|---|---|
| Position | -17.98 | -1.13 | 12.46 |
| Rotation | 0 | 0 | 0 |
| Scale | 1 | 1 | 1 |

Script: Relic

Relic Type: Bladed Pinchers

In this example, a script changes the gizmo based on a selection.

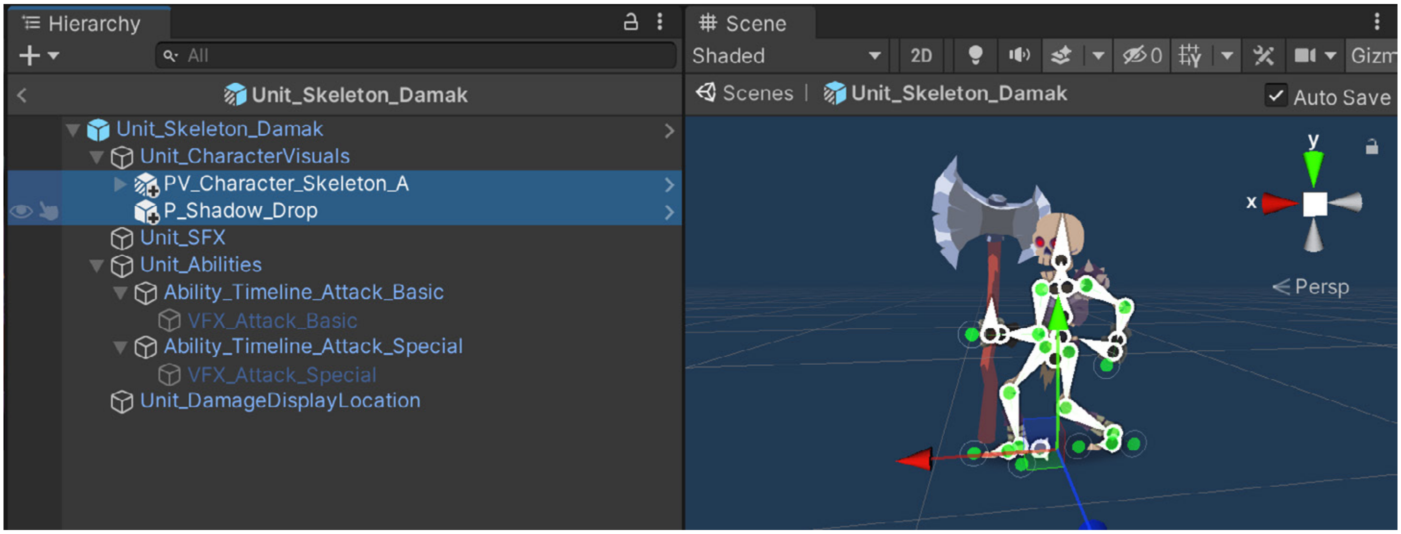
Use the **Gizmos** dialogue in the Scene control bar to toggle specific gizmos or globally enable/disable all of them.

See Creating Custom Gizmos for Development for usage examples. Also, review the APIs for Gizmos and Handles.

## Leverage nested prefabs and prefab variant workflows

Prefabs allow you to save and reuse fully configured GameObjects and lets you build your scenes flexibly and efficiently. However, you can do so much more with prefabs. You can nest prefabs, create prefab variants, or use prefabs to instantiate GameObjects at runtime.

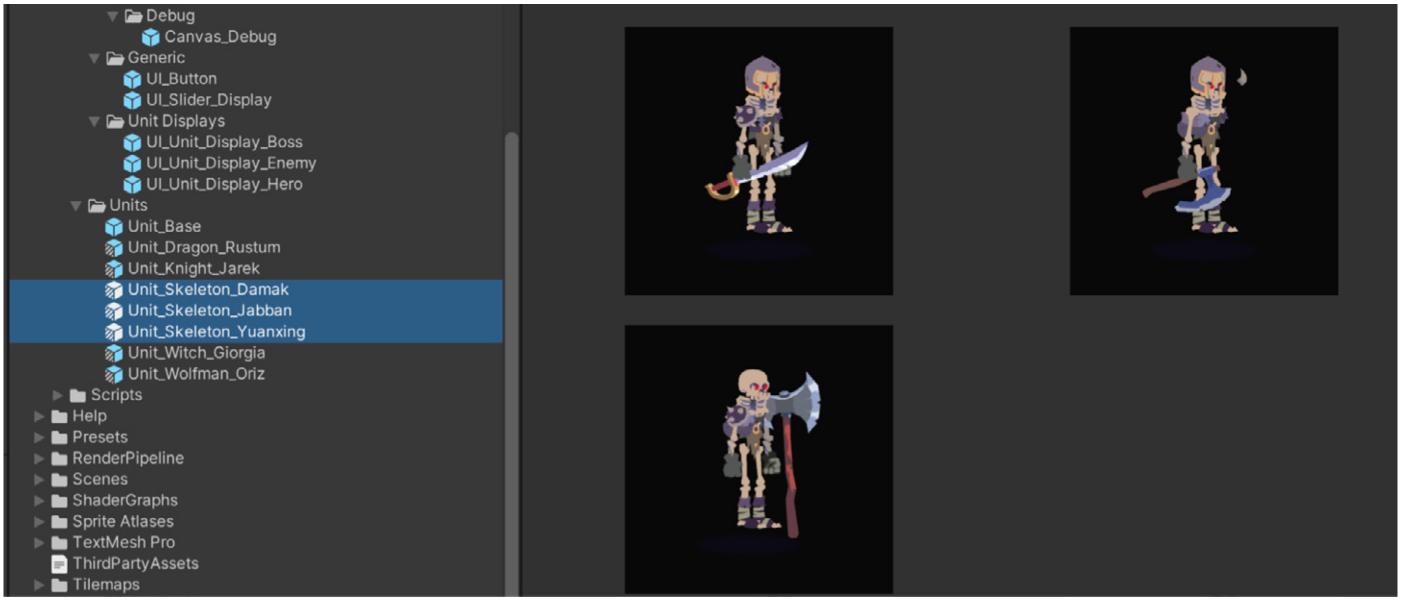
**Nested prefabs** allow you to parent prefabs to one another. You can now create a larger Prefab, such as a building, composed of smaller prefabs for the rooms and furniture. This makes it efficient to split development of your assets over a team of multiple artists and developers, who can all work on different parts of the content simultaneously.



An example of nested prefabs in the *Dragon Crashers* sample project

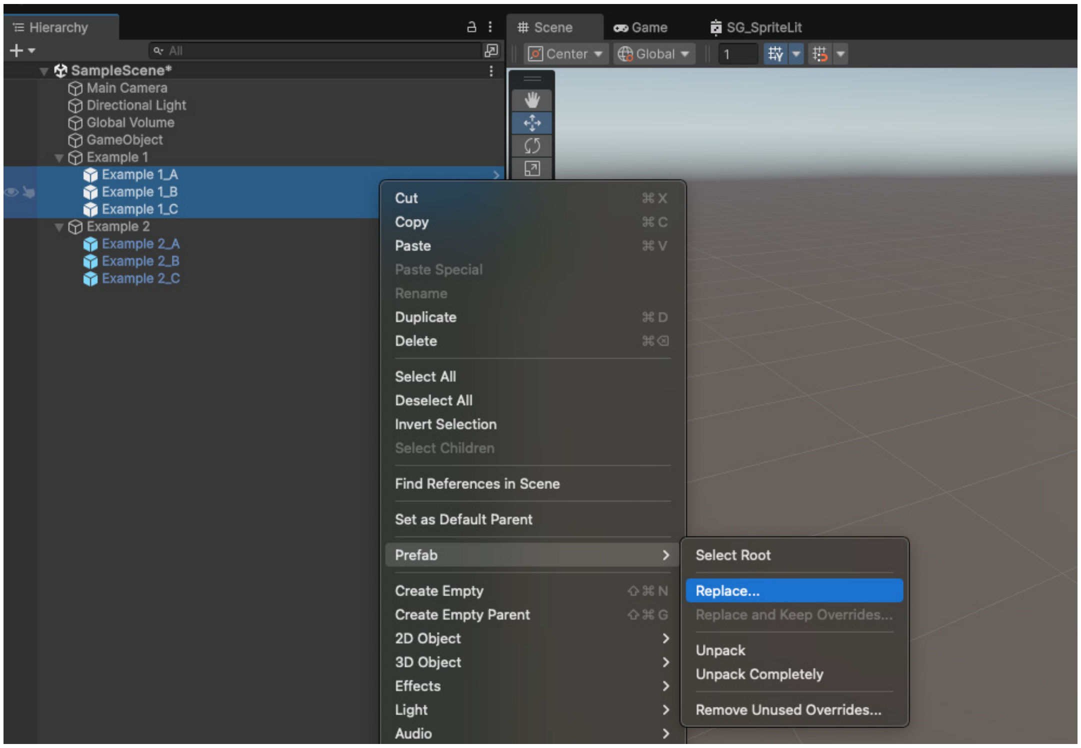
A **prefab variant** allows you to derive a prefab from other prefabs, much like inheritance in object-oriented programming. To change the variant, just override certain parts without worry of impacting the original. You can also remove all modifications and revert to the base prefab at any time.

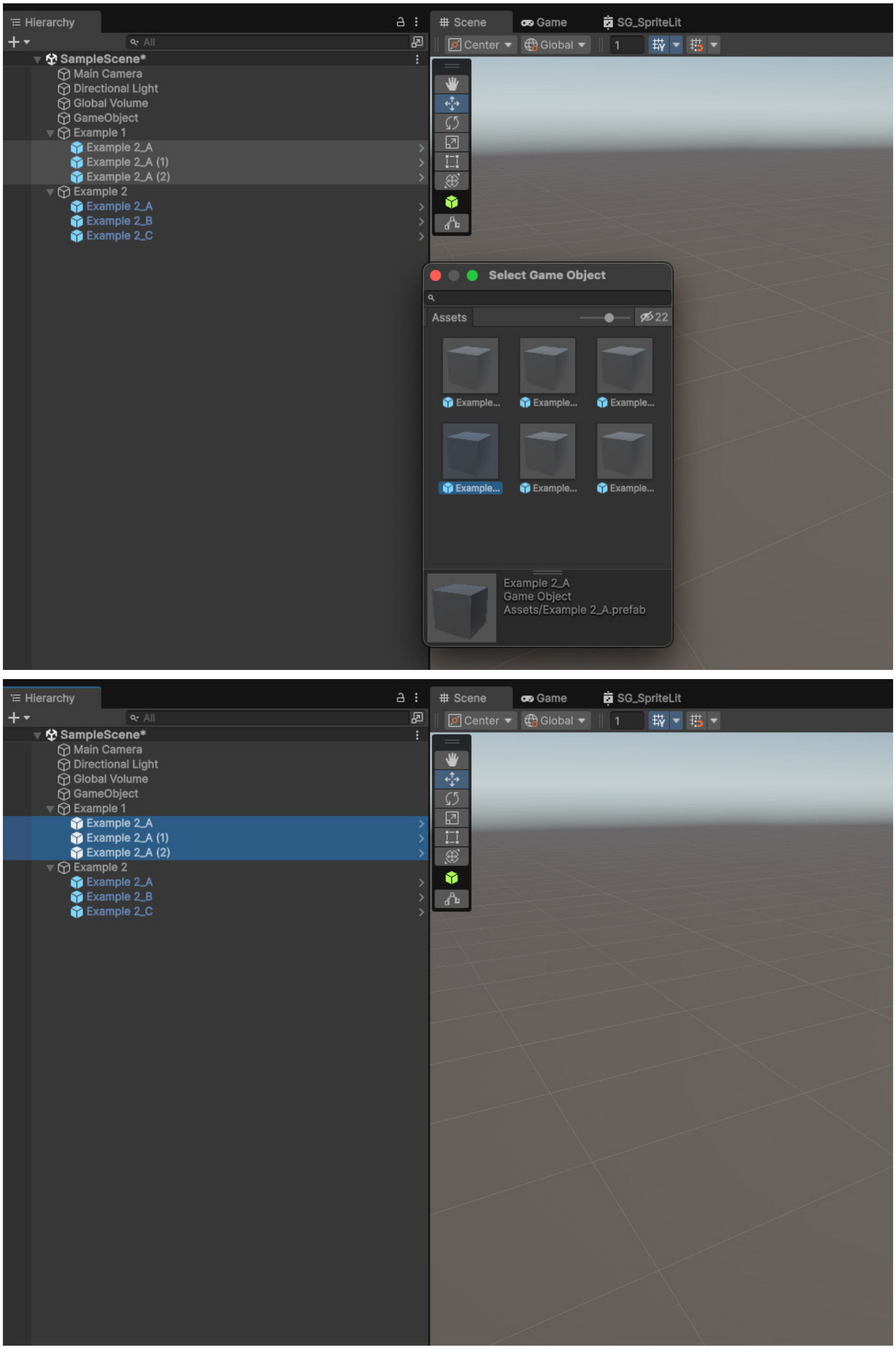
Alternatively, if you want to change all of your variants at once, apply changes directly onto the base prefab itself.



Prefab variants in the *Dragon Crashers* sample project

To replace one or many prefabs in a scene with another prefab, press Ctrl (on Windows) or Cmd (on macOS) and drag the new prefab onto the old ones.

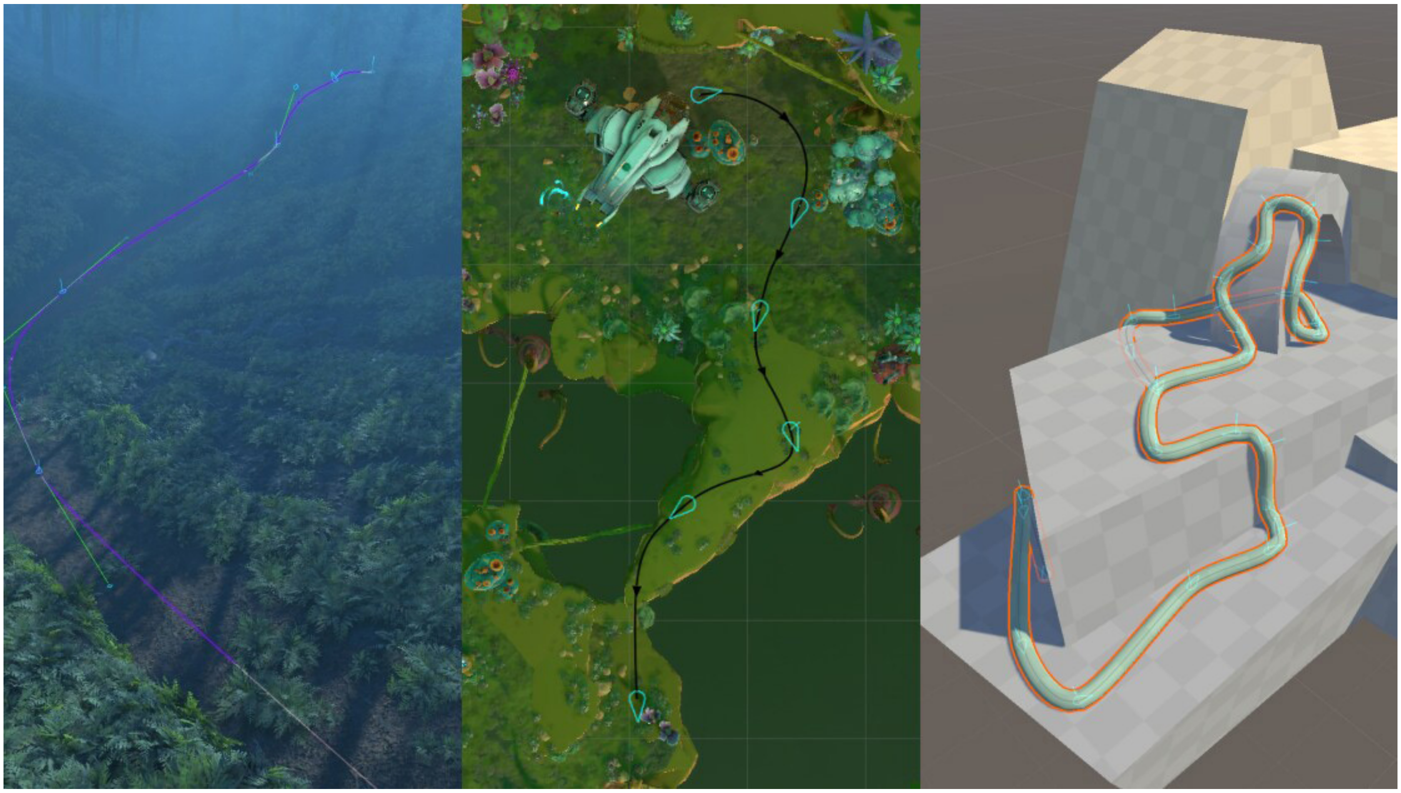




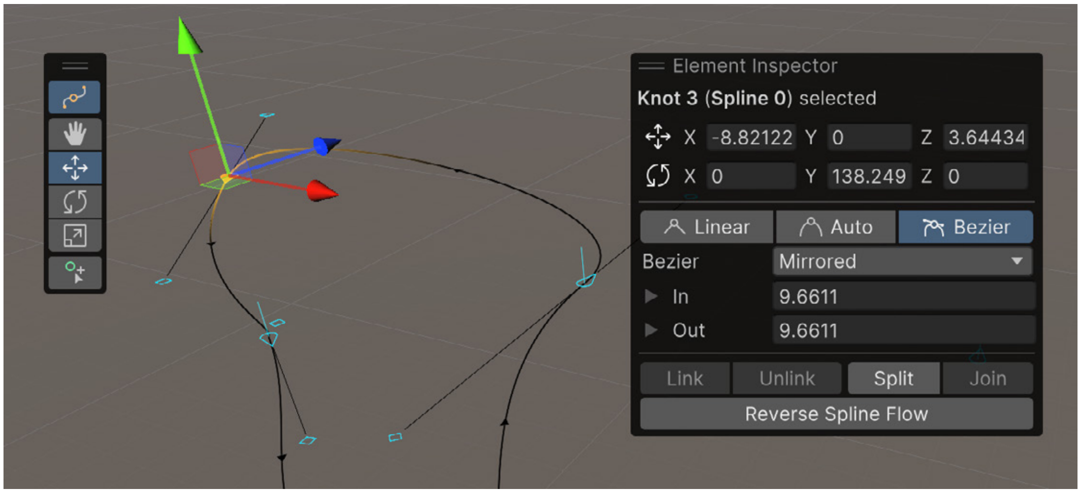
Replacing a prefab in the Hierarchy window

# Create curved paths with ease using the Splines package

The Splines package allows you to create and edit spline-based paths directly within the Editor. Splines are useful for defining smooth, curved paths that can drive elements such as roads, rivers, rails, camera tracks, or any path-based visual or gameplay element.



Examples of use cases for splines, left to right: A road through a forest, an animation path, tube meshes; find other samples showcasing all the possibilities in the Splines Package Manager page once it's installed



Element Inspector

Knot 3 (Spline 0) selected

X -8.82122 Y 0 Z 3.64434

X 0 Y 138.249 Z 0

Bezier: Mirrored

In: 9.6611

Out: 9.6611

The handles and controls for splines in Unity resemble vector or 3D drawing tools from well-known DCC applications.

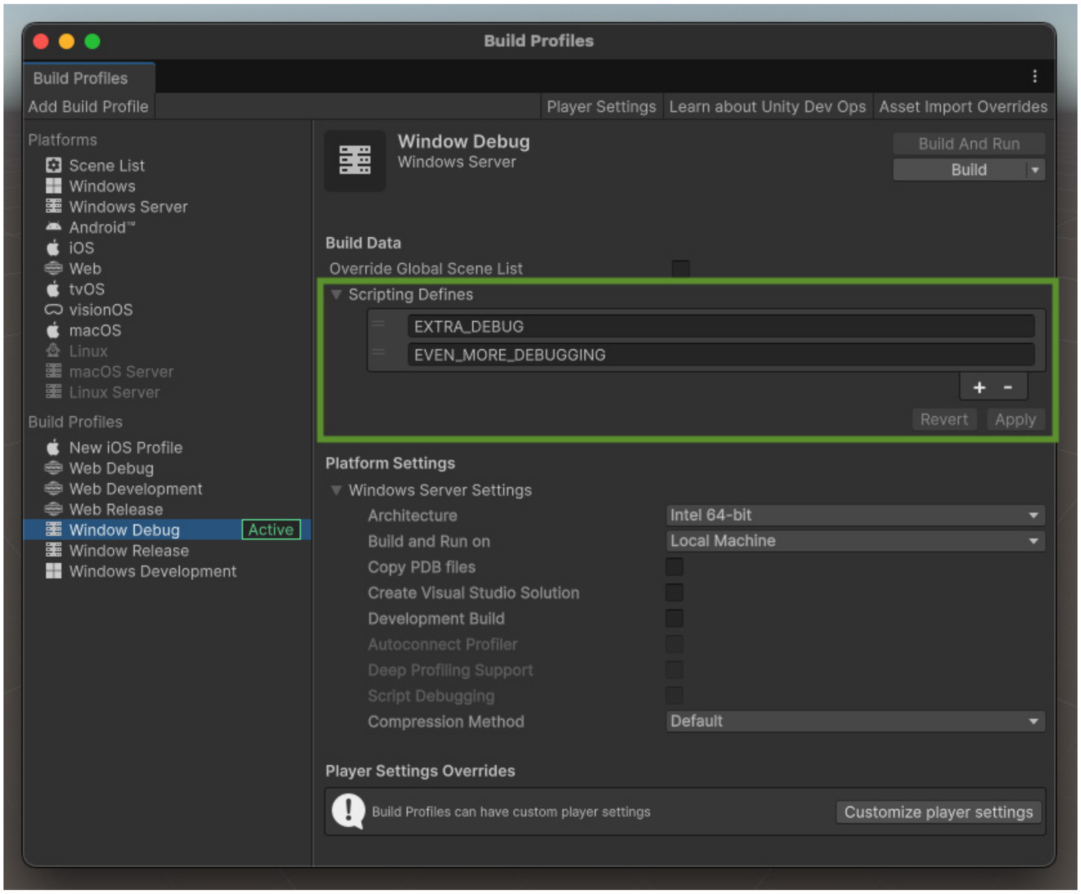
Once you've created a spline, both programmers and artists can use it in interesting ways. For example, programmers can read points of the spline and use them in the game logic with the APIs.

Check out the *How to get started with the splines package* video tutorial to learn how to use the Splines package to solve common spline use cases, animate a GameObject's position and rotation along a spline, and instantiate prefabs along a spline to create environments.

## Customize your builds with Build Profiles

The Build Profile workflow is a new way to configure builds in Unity 6. This new workflow supports multiple configurations for any platform, each with different settings.

Create custom Build Profiles in Unity 6 to efficiently configure different build settings for specific platforms or needs (e.g., debugging, testing, production). This allows you to quickly switch between build configurations that optimize your workflow.



Scripting Defines:
- EXTRA_DEBUG
- EVEN_MORE_DEBUGGING

Architecture: Intel 64-bit
Build and Run on: Local Machine
Compression Method: Default

Build Profiles can have custom player settings

Use the Scripting Defines and Player Settings Overrides options within Build Profiles to set build-specific configurations, such as enabling debug tools or customizing splash images, and share these easily with your team through version control.

More resources:

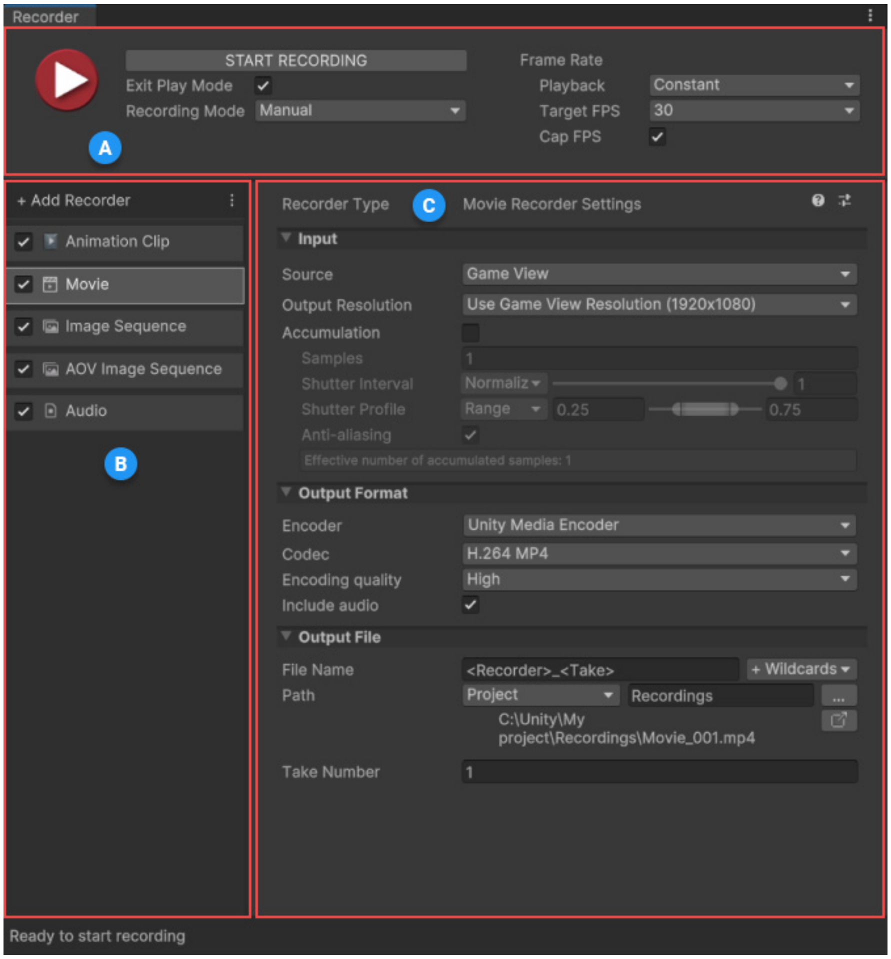
- What you need to know about Build Profiles in Unity 6
- Everything you need to know about Build Profiles in Unity 6 | Unite 2024

# Capture smooth game footage with the Recorder package

The Unity Recorder package allows you to capture and export data from your project during Play Mode. This is useful for recording gameplay, cinematics, or other runtime sequences as video files, image sequences, or animation data.

You can find the Recorder window in the Unity Editor main menu under **Window**. When you open the **Recorder** window, Unity restores the values from the last recording session, allowing for a quick restart or adjustments.

You can capture images or footage with a transparent background. Unity Recorder can capture the alpha channel and output transparency under certain conditions that depend on the render pipeline you're using and the Recorder settings.



Exit Play Mode ☑
Recording Mode: Manual

Frame Rate
Playback: Constant
Target FPS: 30
Cap FPS ☑

☑ Animation Clip
☑ Movie
☑ Image Sequence
☑ AOV Image Sequence
☑ Audio

Recorder Type: Movie Recorder Settings

Input
Source: Game View
Output Resolution: Use Game View Resolution (1920x1080)
Accumulation □
Samples: 1
Shutter Interval: Normaliz 1
Shutter Profile: Range 0.25 0.75
Anti-aliasing ☑
Effective number of accumulated samples: 1

Output Format
Encoder: Unity Media Encoder
Codec: H.264 MP4
Encoding quality: High
Include audio ☑

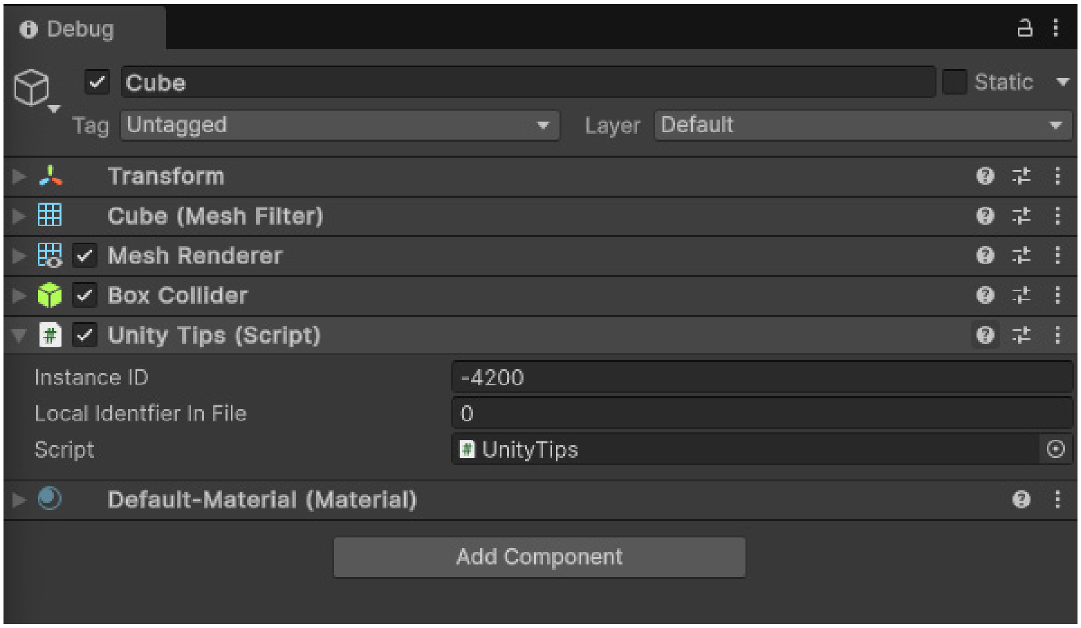
Output File
File Name: <Recorder>_<Take>
Path: Project Recordings
C:\Unity\My project\Recordings\Movie_001.mp4
Take Number: 1

Ready to start recording

The Unity Recorder window

# Editor workflow tips

1. Use the **[HelpURL]** attribute to link the question mark icon in your **Inspector** to your documentation or other resources. When clicked from the Inspector it will open the specified URL in the user's default browser.



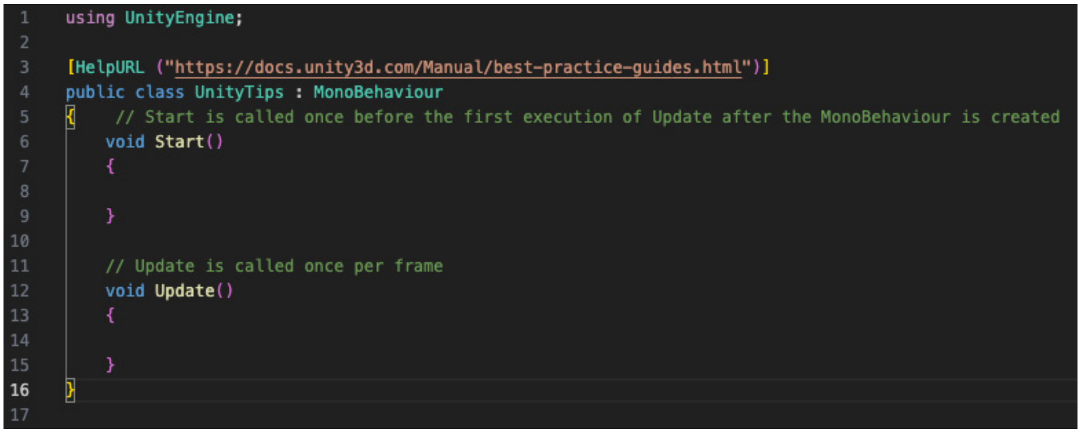
☑ Cube □ Static

Tag Untagged Layer Default

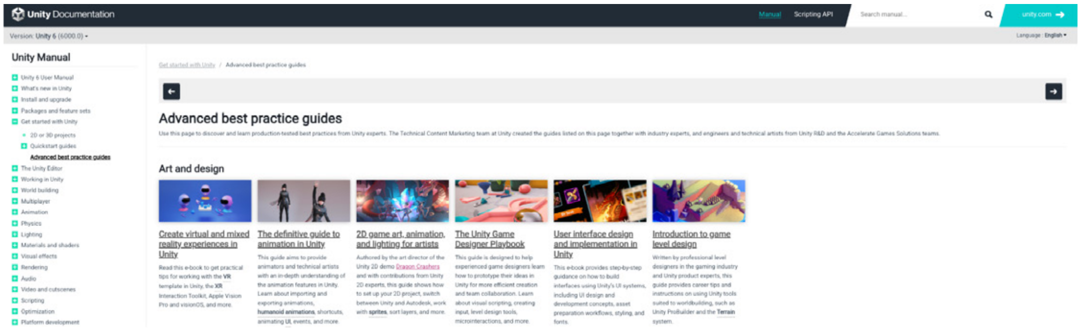
Instance ID -4200

Local Identfier In File 0

Script UnityTips



```
using UnityEngine;

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

    }

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

    }
}
```



Advanced best practice guides

Use this page to discover and learn production-tested best practices from Unity experts. The Technical Content Marketing team at Unity created the guides listed on this page together with industry experts, and engineers and technical artists from Unity R&D and the Accelerate Games Solutions teams.

Art and design

Create virtual and mixed reality experiences in Unity
Read this e-book to get practical tips for working with the VR template in Unity, the XR Interaction Toolkit, Apple Vision Pro and visionOS, and more.

The definitive guide to animation in Unity
This guide aims to provide animators and technical artists with an in-depth understanding of the animation features in Unity. Learn about importing and exporting animations, humanoid animations, shortcuts, animating UI, events, and more.

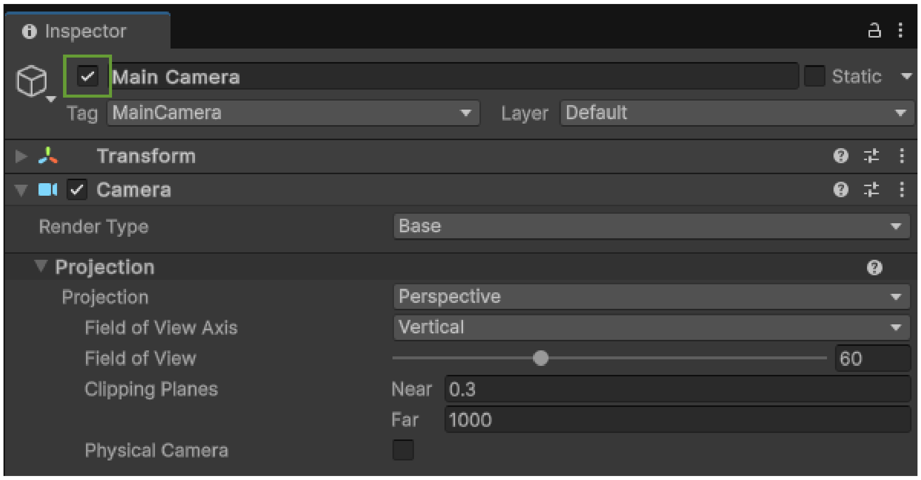
2D game art, animation, and lighting for artists
Authored by the art director of the Unity 2D demo Dragon Crashers and with contributions from Unity 2D experts, this guide shows how to set up your 2D project, switch between Unity and Autodesk, work with sprites, sort layers, and more.

The Unity Game Designer Playbook
This guide is designed to help experienced game designers learn how to prototype their ideas in Unity for more efficient creation and team collaboration. Learn about visual scripting, creating input, level design tools, microinteractions, and more.

User interface design and implementation in Unity
This e-book provides step-by-step guidance on how to build interfaces using Unity's UI systems, including UI design and development concepts, asset preparation workflows, styling, and fonts.

Introduction to game level design
Written by professional level designers in the gaming industry and Unity product experts, this guide provides career tips and instructions on using Unity tools suited to worldbuilding, such as Unity ProBuilder and the Terrain system.

2. Save a few mouse clicks each day. Hold **Shift + Alt + A** (Windows) or **Shift + Option + A** (macOS) to activate or deactivate the currently selected GameObject.



Inspector

☑ Main Camera □ Static

Tag MainCamera Layer Default

Transform

☑ Camera

Render Type Base

Projection

Projection Perspective

Field of View Axis Vertical

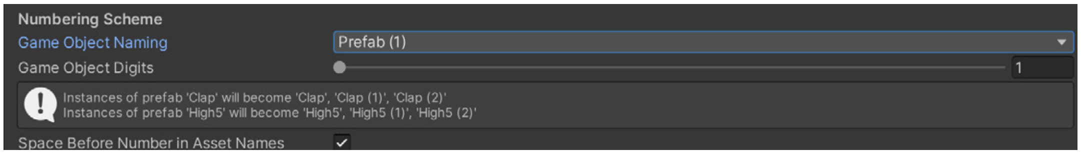
Field of View 60

Clipping Planes Near 0.3 Far 1000

Physical Camera □

Activate or deactivate the selected GameObject.

3. Unity allows you to customize the numbering scheme of GameObjects. Find it in the **Project Settings** in the Editor tab. Define the options for the naming here as well as the padding and spacing of the instance number.



Numbering Scheme

Game Object Naming Prefab (1)

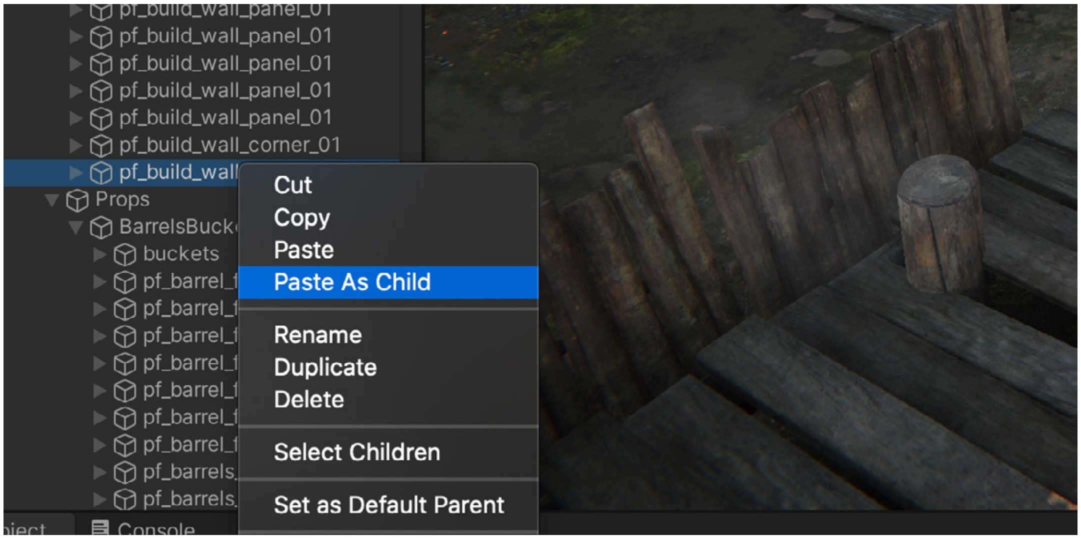
Game Object Digits 1

Instances of prefab 'Clap' will become 'Clap', 'Clap (1)', 'Clap (2)'
Instances of prefab 'High5' will become 'High5', 'High5 (1)', 'High5 (2)'

Space Before Number in Asset Names ☑

Numbering Scheme

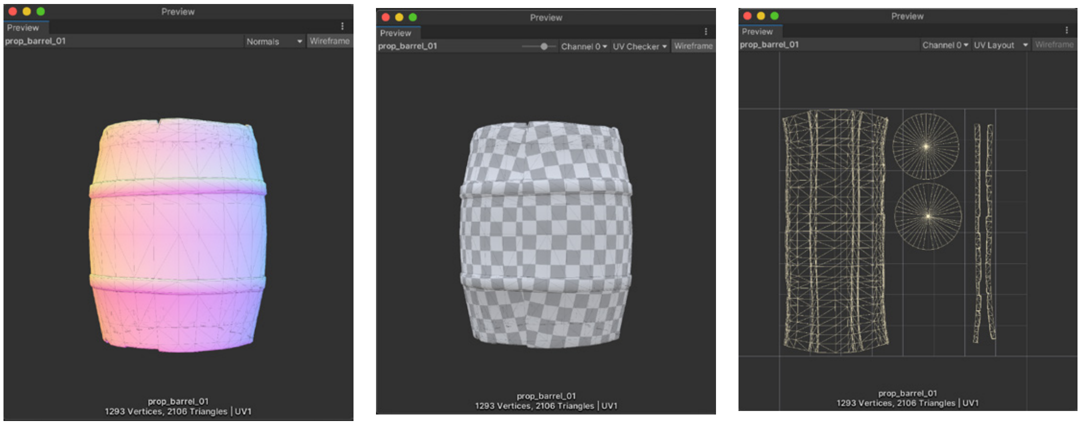
4. Cut and paste GameObjects in the Hierarchy window. You can also **Paste As Child** from the context menu.



Paste As Child

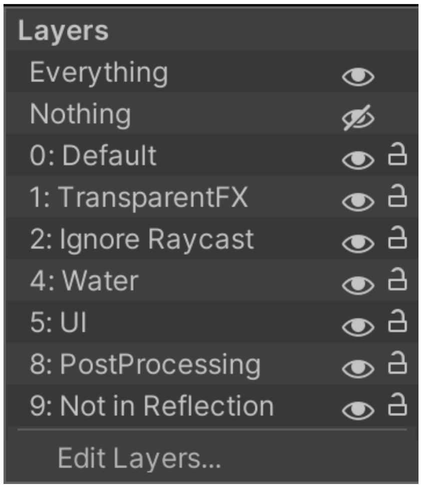
5. Use the **F** shortcut to frame the selected object in Scene view. In Play mode, press **Shift + F** to lock onto a selected GameObject that is moving.

6. Display UVs, normals, tangents, and other Mesh information in the Inspector preview.



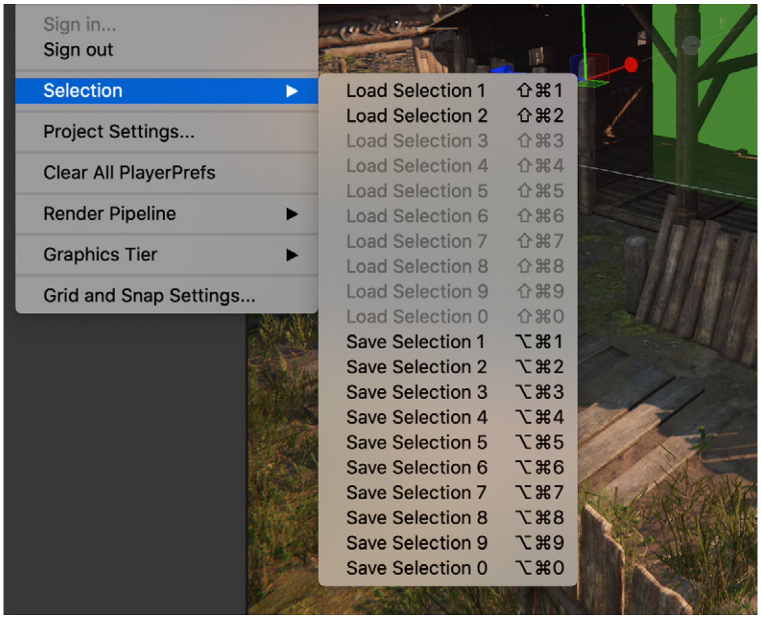
The Inspector preview

7. Use the Layers menu to toggle off the visibility of any Layers (such as UI) that may obscure your Scene view. Lock a Layer to avoid changing its state accidentally.



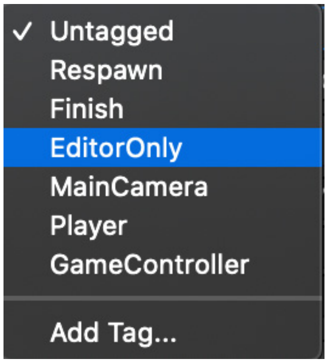
Toggle and edit Layers

8. If you frequently select the same objects in your scene, use the hotkey combos under **Edit** > **Selection** to quickly save or load a selection set.



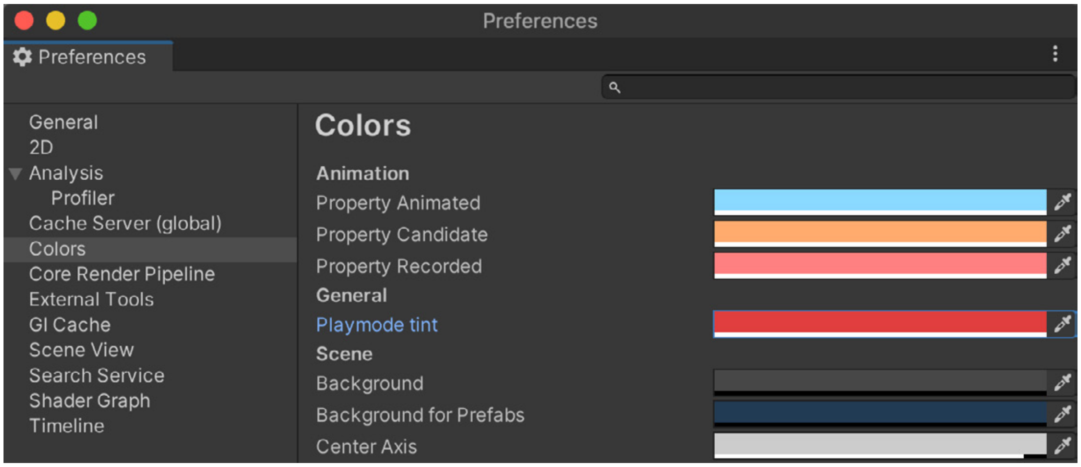
Load and Save Selections

9. Use the **EditorOnly** tag to designate GameObjects that will not appear in a build of the application.



EditorOnly tag

10. Change colors in the Editor via **Unity > Preferences > Colors** to find certain UI elements or objects more quickly. Adjust the Play mode tint to remind yourself when Play mode is active, so you don't lose any changes you intended to save on exit.



Preferences

Preferences

General
2D
Analysis
Profiler
Cache Server (global)
Colors
Core Render Pipeline
External Tools
GI Cache
Scene View
Search Service
Shader Graph
Timeline

## Colors

**Animation**

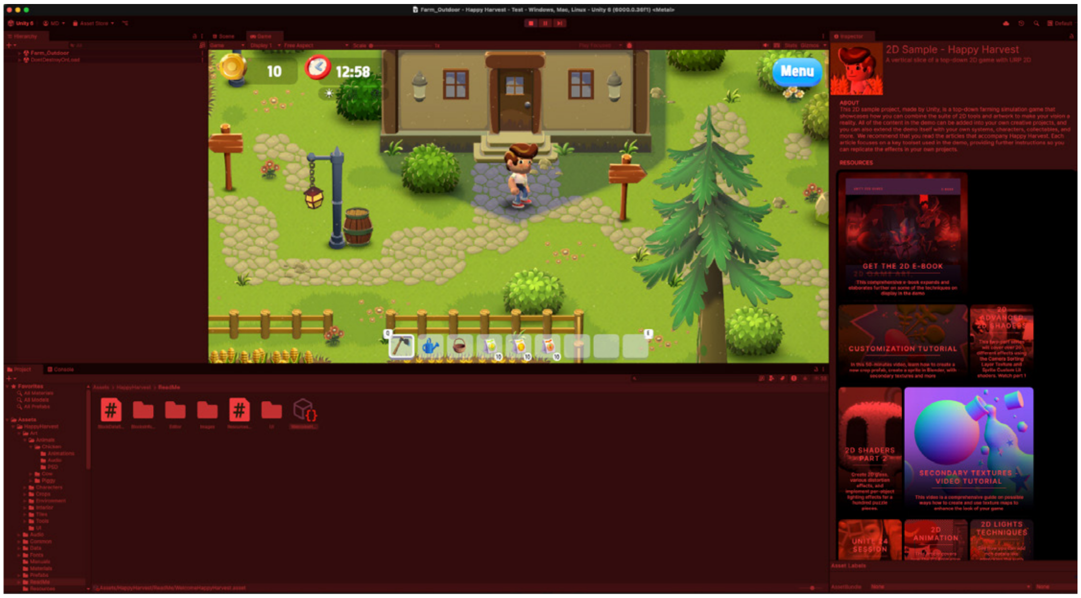
| | |
|---|---|
| Property Animated | 🟦 |
| Property Candidate | 🟧 |
| Property Recorded | 🟥 |

**General**

| | |
|---|---|
| Playmode tint | 🟥 |

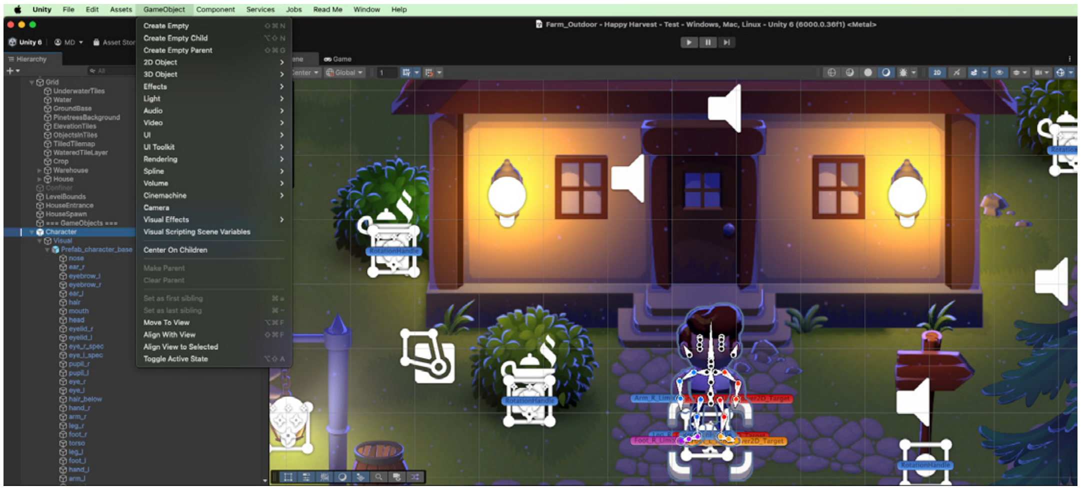
**Scene**

| | |
|---|---|
| Background | ⬛ |
| Background for Prefabs | 🟦 |
| Center Axis | ⬜ |



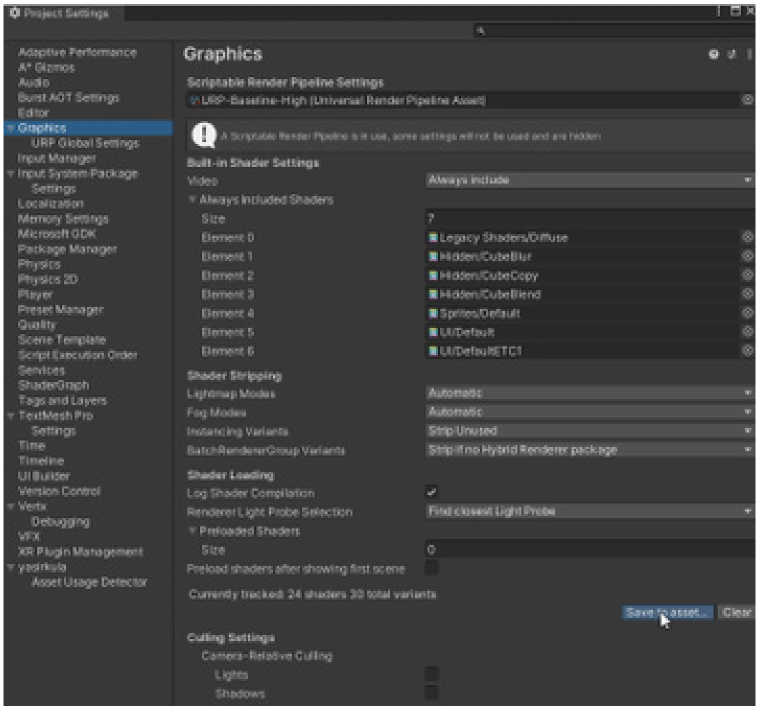
Play mode tint

11. When you set up cameras, use **GameObject > Align With View** to line up your Game camera with the Scene camera. Or, if you're matching the other way around, use **Align View to Selected** to align the Scene camera with another camera in the Hierarchy.



Align With View option

12. You can generate a list of shader variants that the Editor uses in the Scene: Go to **Edit** > **Project Settings** > **Graphics**. See how many shaders and variants you have under "Shader Loading". Select **Save to asset…** to create a shader variant collection asset.



Graphics

Scriptable Render Pipeline Settings

URP-Baseline-High (Universal Render Pipeline Asset)

A Scriptable Render Pipeline is in use, some settings will not be used and are hidden

Built-in Shader Settings

Video: Always include

Always Included Shaders

Size: 7

Element 0: Legacy Shaders/Diffuse

Element 1: Hidden/CubeBlur

Element 2: Hidden/CubeCopy

Element 3: Hidden/CubeBlend

Element 4: Sprites/Default

Element 5: UI/Default

Element 6: UI/DefaultETC1

Shader Stripping

Lightmap Modes: Automatic

Fog Modes: Automatic

Instancing Variants: Strip Unused

BatchRendererGroup Variants: Strip if no Hybrid Renderer package

Shader Loading

Log Shader Compilation: ☑

Renderer Light Probe Selection: Find closest Light Probe

Preloaded Shaders

Size: 0

Preload shaders after showing first scene: ☐

Currently tracked: 24 shaders 30 total variants

Culling Settings

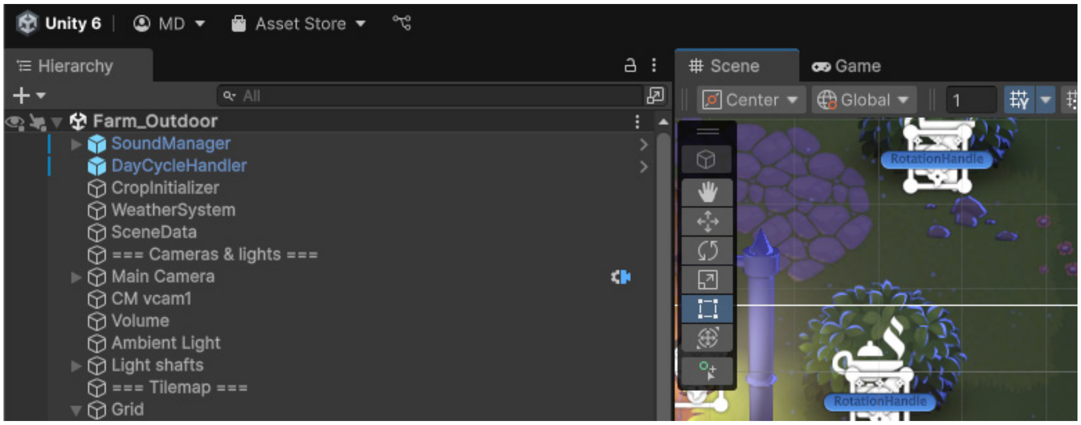
Camera-Relative Culling

Lights: ☐

Shadows: ☐

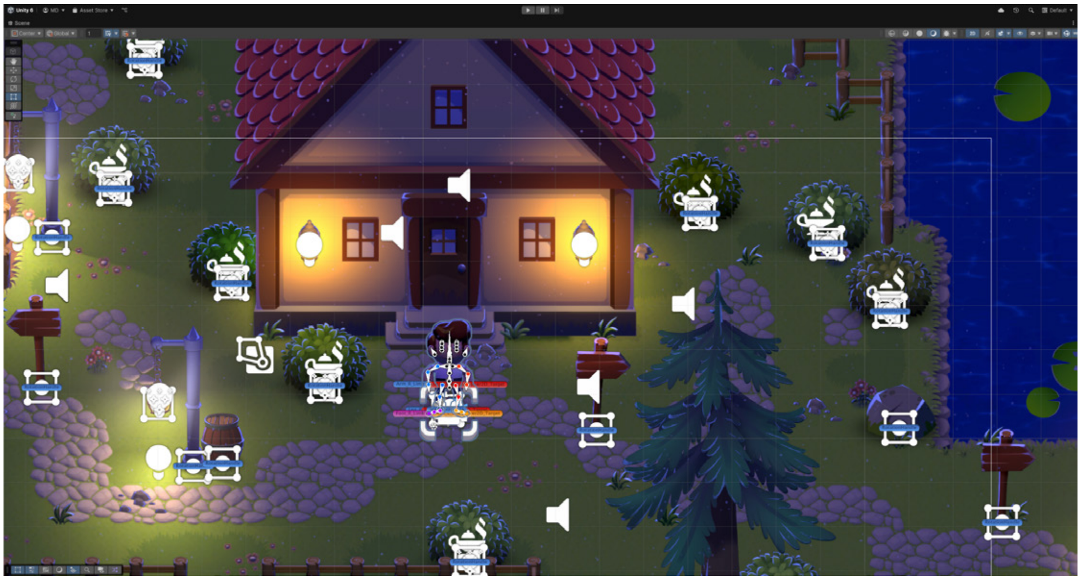
Graphics window

13. Double-click any tab (**Project**, **Scene**, **Game**, etc.) to go full screen in the Editor.



Farm_Outdoor
SoundManager
DayCycleHandler
CropInitializer
WeatherSystem
SceneData
=== Cameras & lights ===
Main Camera
CM vcam1
Volume
Ambient Light
Light shafts
=== Tilemap ===
Grid

Double-click on **Scene**



After double-clicking on **Scene**

## More resources

For more quick productivity tips make sure to check out the following resources:

— Speed up your workflows in Unity with these keyboard shortcuts

— Scalable art assets - 6 tips for improving workflows

— Top tips for scripting in Unity 2022 LTS

# 2D

## Sprites

A 2D project uses Sprites to create its visuals. These potentially contain different Texture assets and may thus require many draw calls if not batching properly.

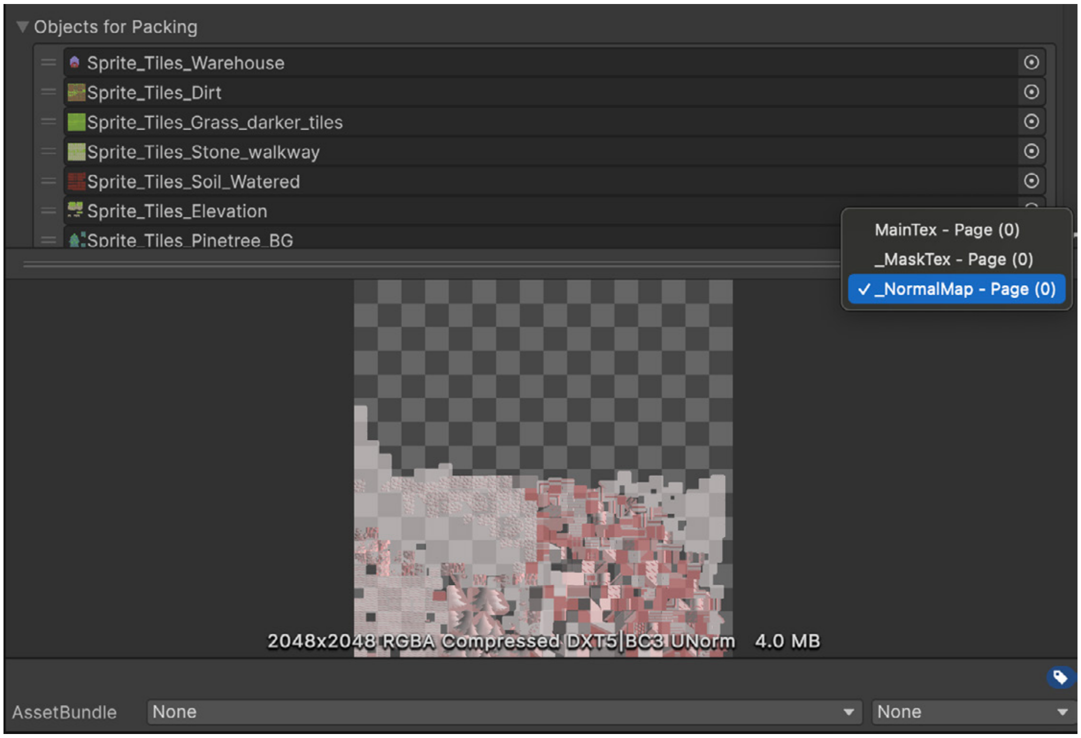
To optimize resources, use a **Sprite Atlas** (**Asset** > **Create** > **Sprite Atlas**) which will allow batching sprites using the same material and packed texture.

## Avoid duplicated assets when packing sprites

In 2D URP, sprites can contain normal maps or mask maps which you might also want to pack in a Sprite Atlas. This is important for performance and quality reasons, such as memory optimization, draw call reduction, and loading efficiency.

If you add folders to the Sprite Atlas, make sure they only contain the normal sprite or albedo texture. Normal maps and mask maps will automatically be added to their respective atlases that you can view in the drop-down list. Otherwise, they would be added unnecessarily to the normal pack atlas, taking up space in it.

Having neatly organized atlases for normal maps or mask maps allow you to fine-tune compression settings for each one. For example, normal maps can be half of the resolution, while the normal sprites can be in full resolution, or any other adjusting that you need based on the platform you're developing for.



Objects for Packing

- Sprite_Tiles_Warehouse
- Sprite_Tiles_Dirt
- Sprite_Tiles_Grass_darker_tiles
- Sprite_Tiles_Stone_walkway
- Sprite_Tiles_Soil_Watered
- Sprite_Tiles_Elevation
- Sprite_Tiles_Pinetree_BG

2048x2048 RGBA Compressed DXT5|BC3 UNorm 4.0 MB

AssetBundle: None / None

The three types of atlases when packing 2D sprites

## Structure your folders to maintain consistency

Folder structure within 2D sprites makes it easier to locate and manage assets, helps manage memory by organizing assets by load time, provides a clear separation of different game elements, and more.

Here's a common folder structure to help organize the folders your Unity project:

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

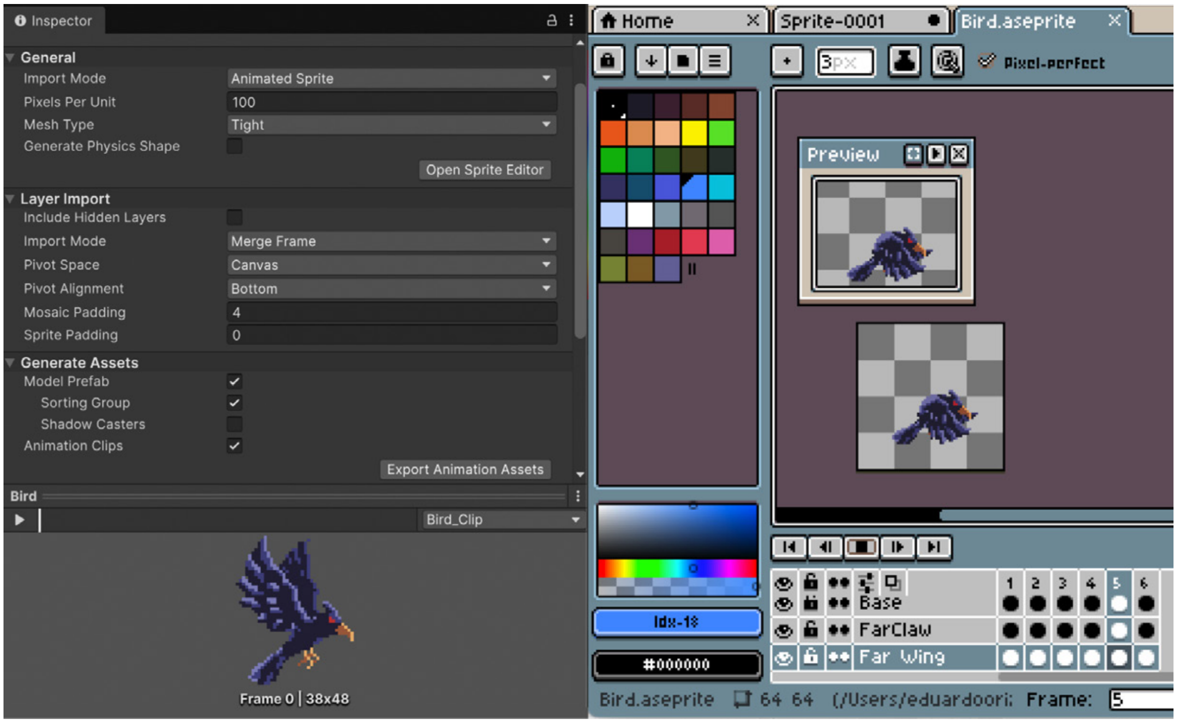
## Bring your pixel art directly to Unity with Aseprite Importer

Aseprite, a sprite editor and pixel art tool, has become an industry-standard DCC app for pixel artists.

With Unity Aseprite Importer, you don't have to export hundreds of .PNG sprites individually; you can add your **.aseprite** file directly to your project.

Work directly in Aseprite, click **Save**, and instantly see the changes in Unity. The Importer includes drag and drop support for .aseprite files, automatic setup of GameObjects and Animation clips so they're ready to use, and a comprehensive 2D feature set integration.

In Unity 6, we enabled auto-generation of Tilemap assets from **.aseprite** files containing tilemaps. Changes to the source propagate down automatically.



Inspector

General
Import Mode: Animated Sprite
Pixels Per Unit: 100
Mesh Type: Tight
Generate Physics Shape: □
Open Sprite Editor

Layer Import
Include Hidden Layers: □
Import Mode: Merge Frame
Pivot Space: Canvas
Pivot Alignment: Bottom
Mosaic Padding: 4
Sprite Padding: 0

Generate Assets
Model Prefab: ☑
Sorting Group: ☑
Shadow Casters: □
Animation Clips: ☑
Export Animation Assets

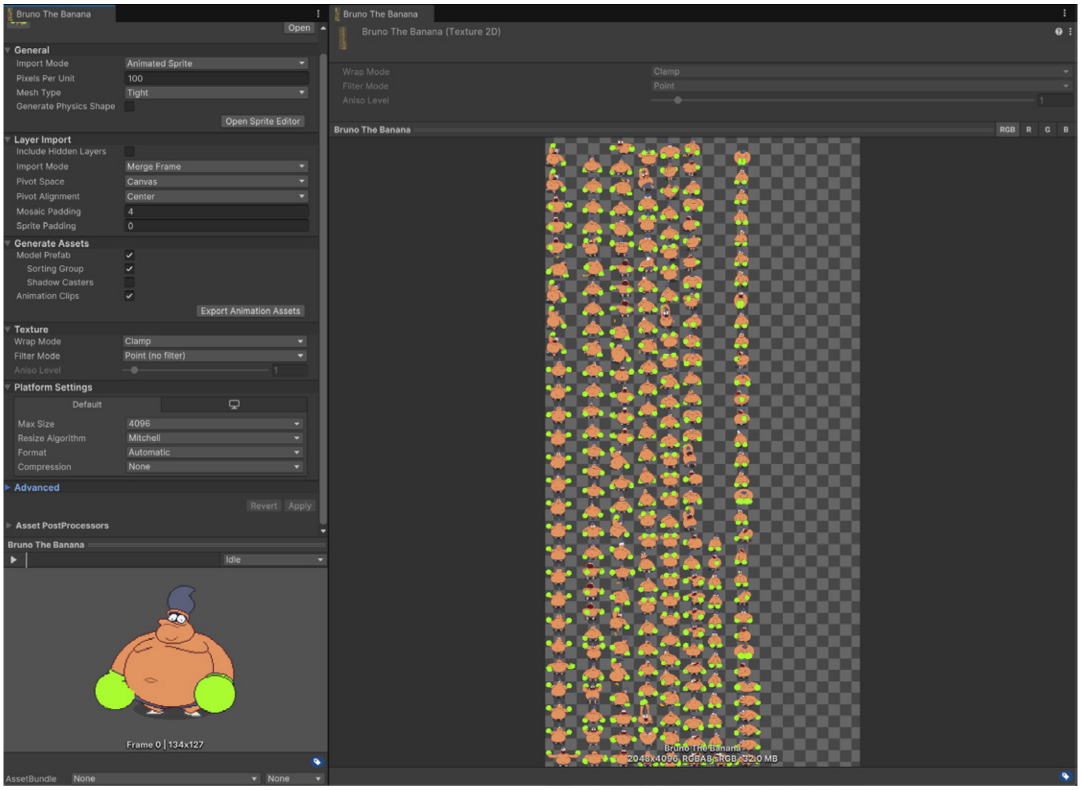
Bird
Bird_Clip
Frame 0 | 38x48

The Aseprite Importer in the Unity Project window: This section of the Unite 2024 talk Say hello to new 2D workflows and AI enhancements demonstrates the Aseprite Importer in action

## Bring your frame-by-frame animations from Aseprite to Unity

One of the most helpful tips for using the Aseprite Importer in Unity is to keep your Aseprite files well-organized with **Tags** and **Slices** before importing, as these features allow smooth and automatic handling of animations and sprite slicing directly in the Editor.

For example, use tags such as "Idle", "Walk", etc., to define and organize animations, and use Slices by marking specific areas of your sprite or separate parts, such as head, weapon, etc. This can help with streamlining the importing process and save you time.



General
Import Mode: Animated Sprite
Pixels Per Unit: 100
Mesh Type: Tight
Generate Physics Shape: ☐

Layer Import
Include Hidden Layers: ☐
Import Mode: Merge Frame
Pivot Space: Canvas
Pivot Alignment: Center
Mosaic Padding: 4
Sprite Padding: 0

Generate Assets
Model Prefab: ☑
Sorting Group: ☑
Shadow Casters: ☐
Animation Clips: ☑

Texture
Wrap Mode: Clamp
Filter Mode: Point (no filter)
Aniso Level: 1

Platform Settings
Max Size: 4096
Resize Algorithm: Mitchell
Format: Automatic
Compression: None

AssetBundle: None | None

Wrap Mode: Clamp
Filter Mode: Point
Aniso Level: 1

An imported animation into Unity from Aseprite

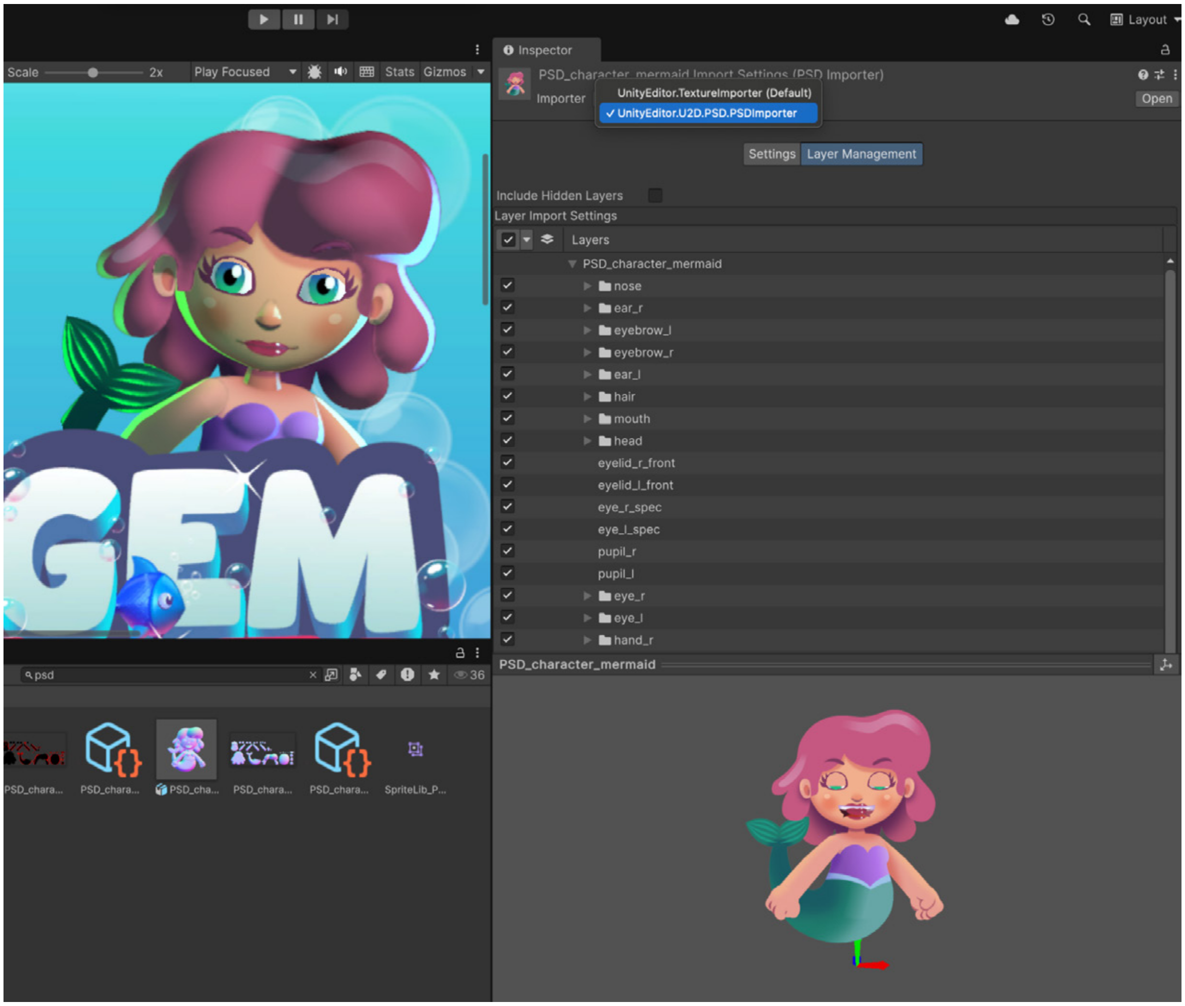
# Speed up the roundtrip import process with the PSD Importer

The PSD Importer imports Adobe Photoshop files into Unity and generates a prefab of sprites based on the imported source file.

You can speed up the importing process for 2D sprites, UI elements, or any other Photoshop-made asset with the PSD importer. Traditionally, artists would export every layer in a Photoshop file, as a .png file to use in Unity, and then they'd have to reexport .png files with every change. This is no longer needed, as Unity supports the layered .psd format.

Animated characters made for 2D, where each limb or part is in a different layer, can be imported to Unity for 2D Animation. It will set up the character so that you can rig it and animate it later.

If every PSD layer is meant to be a standalone sprite, you can uncheck the **Character Rig** option, removing the prefab object, and get a simple sliced sprite with all the layers usable as standalone sprites.



PSD_character_mermaid Import Settings (PSD Importer)

Importer: UnityEditor.TextureImporter (Default) / ✓ UnityEditor.U2D.PSD.PSDImporter

Include Hidden Layers □

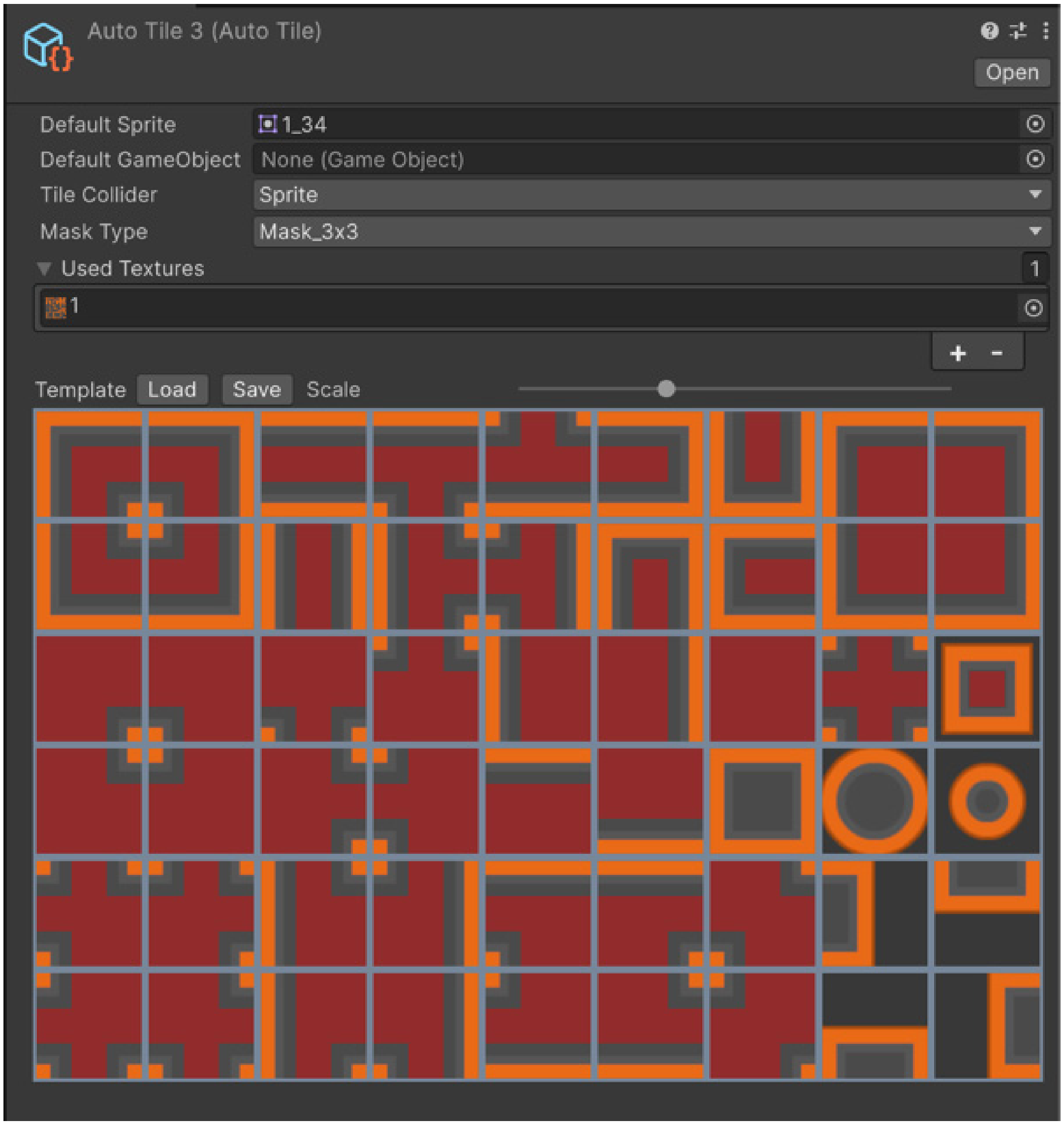
Ensure that you import your layered PSD files with the PSD Importer, and not the default texture importer, so that you have access to all the importer features.

See more helpful tips in action in this *From Photoshop to Unity with PSD Importer* video tutorial.

# Easier rule tiles with Tilemap Editor in Unity 6.1

Unity’s Tilemap system stores and handles Tile Assets for creating grid-based 2D levels, which makes it easy to create and iterate on level design cycles within Unity.

One of the challenges when working with tiles is to create all the neighbouring tiles assets and assign them to a Rule Tile Asset. In Unity 6.1, a new option, called AutoTile, simplifies this process. Load a tile sheet, select the inner areas for it, and save it to generate a tile asset that you can use in your tile palette.



Auto Tile 3 (Auto Tile)

Default Sprite: 1_34

Default GameObject: None (Game Object)

Tile Collider: Sprite

Mask Type: Mask_3x3

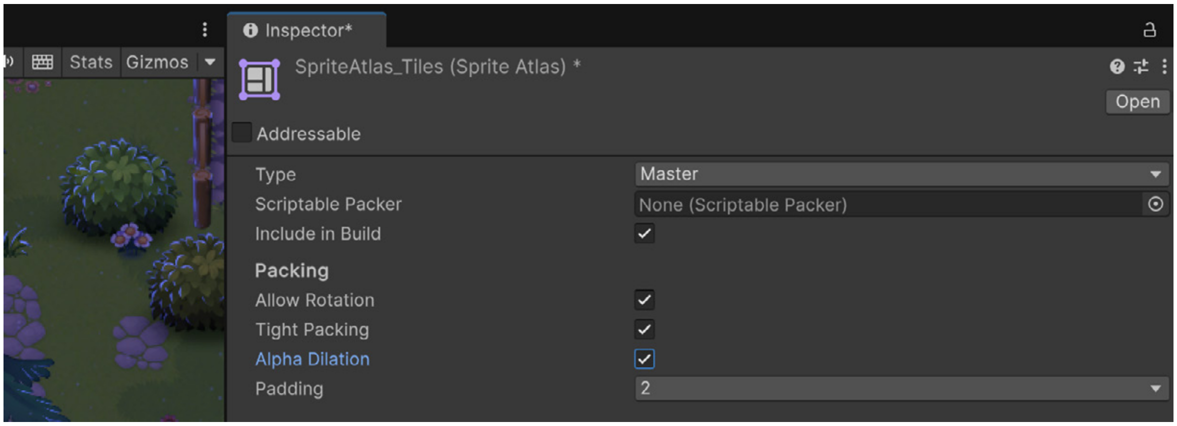
Used Textures: 1

AutoTile example

## Avoid texture bleeding or small gaps between tiles

With all the effort put into art and tilemaps, you'll want to avoid the appearance of bleeding or small gaps in between tiles due to interpolation and smoothing of edges (this isn't an issue in pixel art games, where sprites are not smoothed).

If you're not using a tile sheet then we recommend packing the sprites with Sprite Atlas to help smoothen seams, and with internal sorting in the TilemapRenderer. This will provide better project organization, performance, and creative control. Some default settings can work for most sprites but tilemaps might require a bit more tweaking. Features like avoiding rotation of sprites when packed or alpha dilation helps tiles maintain their sharp edges.



SpriteAtlas_Tiles (Sprite Atlas) *

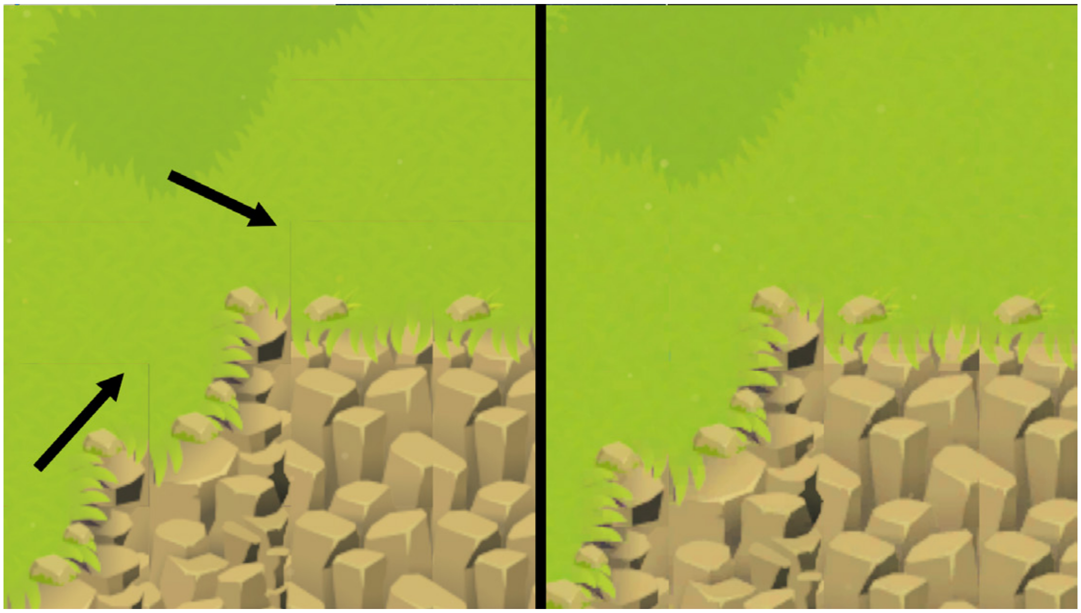
□ Addressable

| | |
|---|---|
| Type | Master |
| Scriptable Packer | None (Scriptable Packer) |
| Include in Build | ☑ |

**Packing**

| | |
|---|---|
| Allow Rotation | ☑ |
| Tight Packing | ☑ |
| Alpha Dilation | ☑ |
| Padding | 2 |

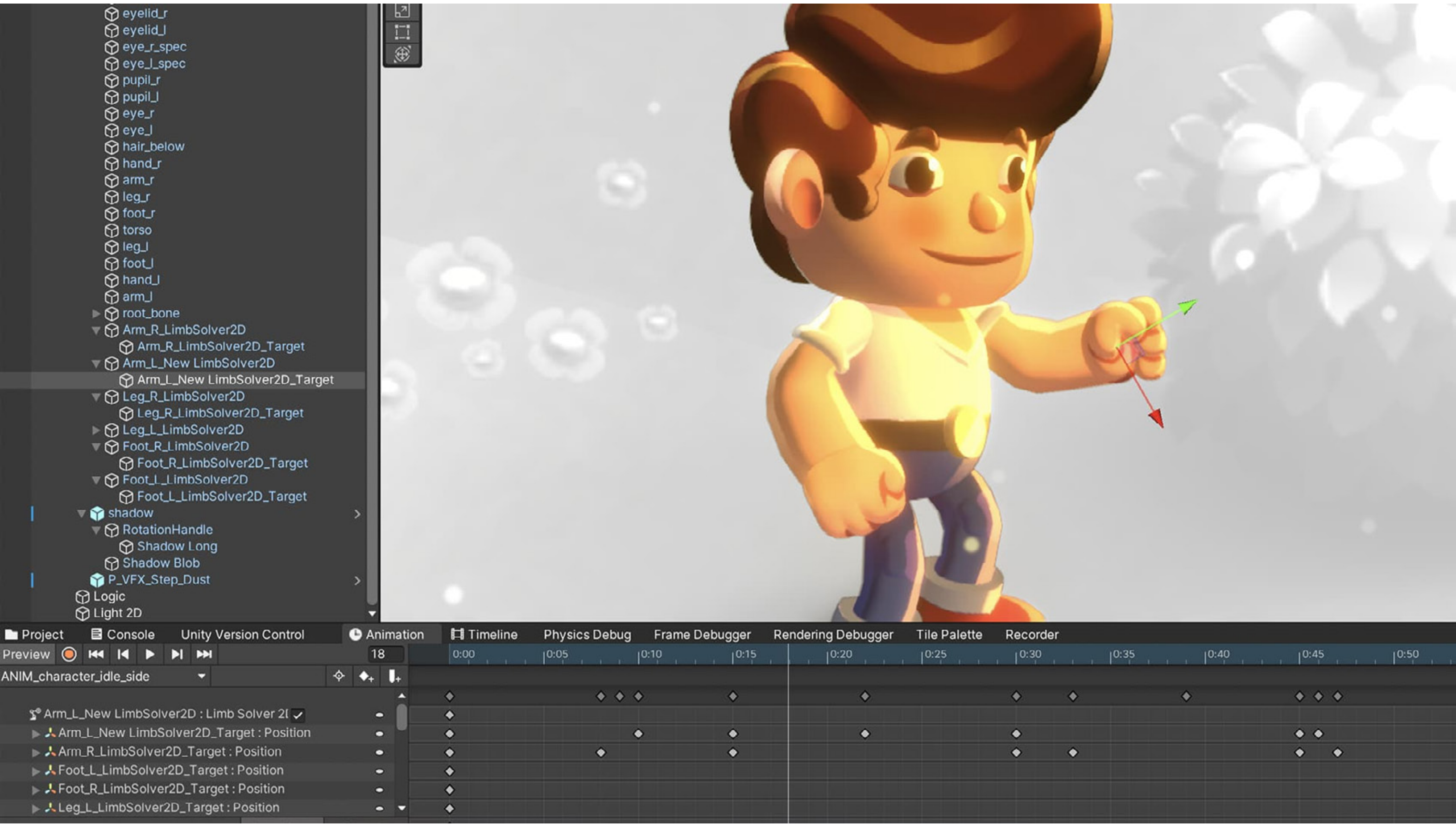
Alpha Dilation option in the Sprite Atlas



In the left image you can see some bleeding edges on tiles, which was fixed by adjusting settings in the Sprite Atlas.

# Use the 2D Inverse Kinematics system to create natural movements

Unity comes with a complete solution for skeletal animation called 2D Animation. Use Unity's 2D Inverse Kinematics (IK) system for rigging movement. Use IK to create natural and dynamic movements, such as bending knees or reaching out, without manually adjusting each bone.



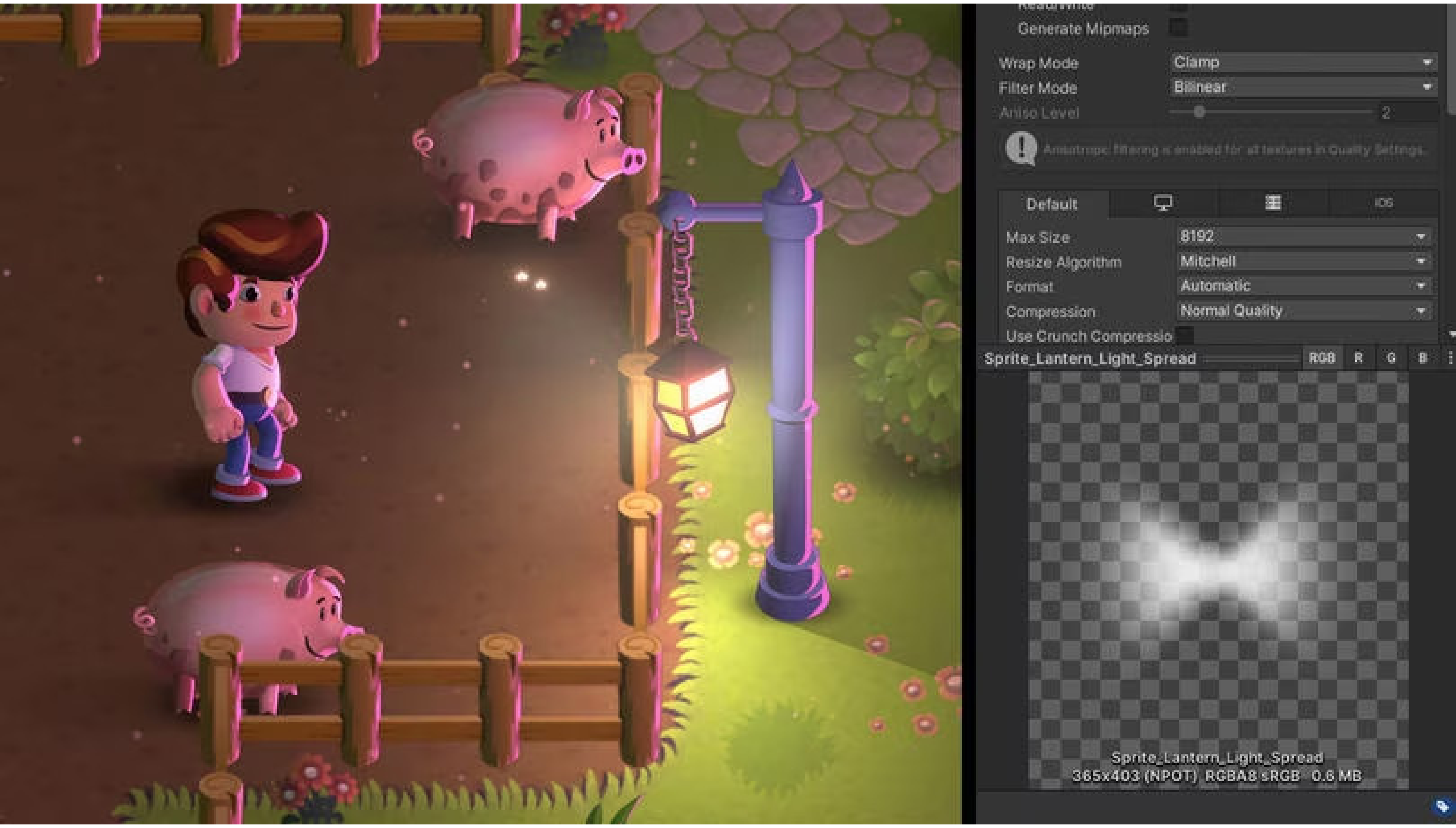
Using 2D IK during the animation process for the main character in the *Happy Harvest* sample project

Carefully design bone hierarchies in the Sprite Editor to ensure proper control. For example, parent the hand bone to the arm bone. Test the hierarchy with simple animations to ensure that any adjustments to a higher-level bone also affect its children properly.

Use Animation Blend Trees for smooth transitions between animations like walking, running, or idle states based on variables like speed or input direction. This results in natural-looking animations without abrupt changes between states.

# Simulate IES profiles with 2D lights

It’s common in 3D software to simulate the fall-off of lights with IES profiles, for example in HDRP. Unity’s 2D light system is very flexible with easy-to-configure parameters like light colors, intensity, fall-off, and blending effects. If you want to be more precise about the fall-off effect of a light source, use a light of the type sprite and use your custom-made light effect.



Generate Mipmaps □
Wrap Mode: Clamp
Filter Mode: Bilinear
Aniso Level: 2
Anisotropic filtering is enabled for all textures in Quality Settings.

Max Size: 8192
Resize Algorithm: Mitchell
Format: Automatic
Compression: Normal Quality
Use Crunch Compressio □

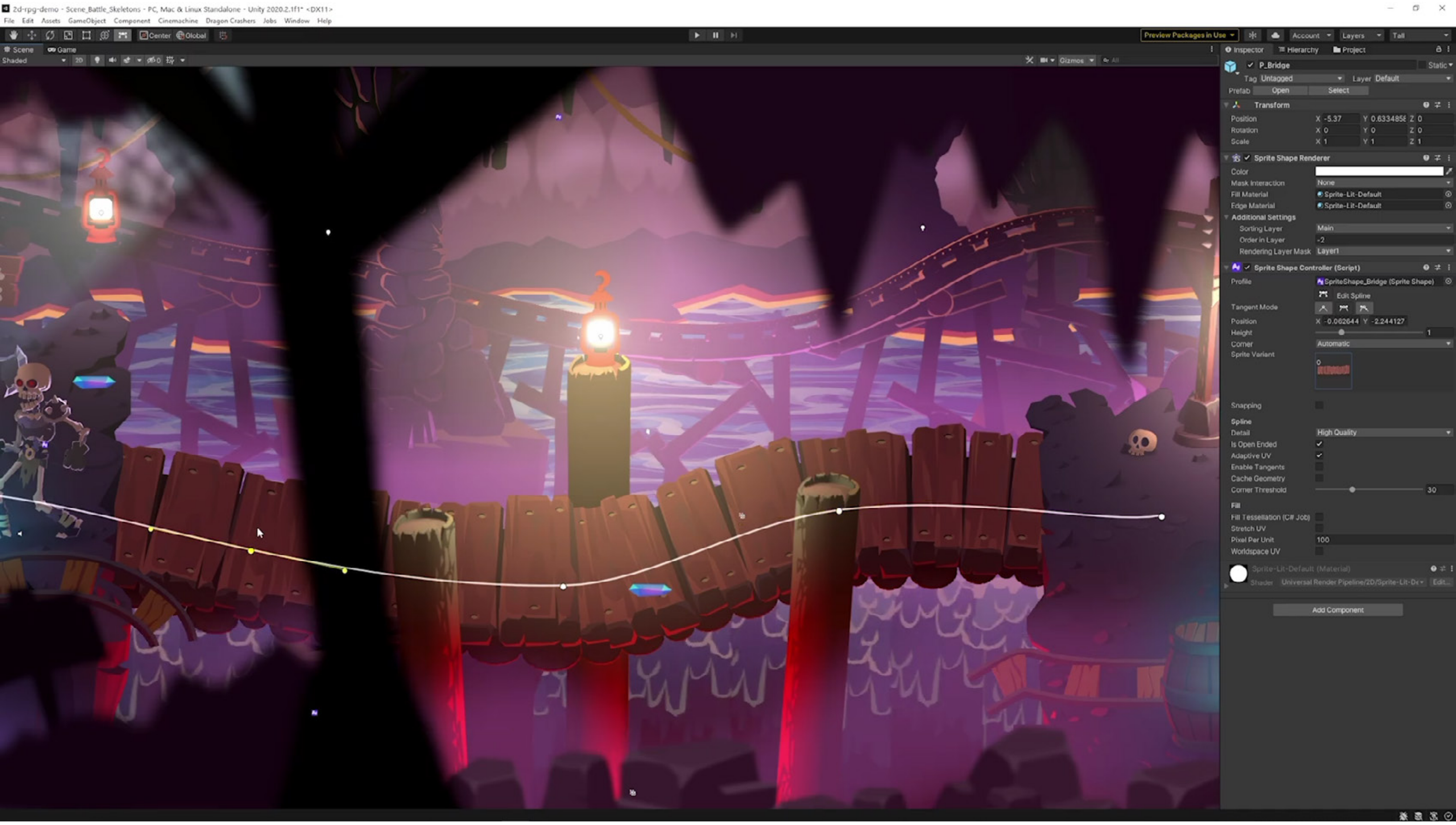
Sprite_Lantern_Light_Spread
365x403 (NPOT) RGBA8 sRGB 0.6 MB

Using a sprite with a halo around a hanging lamp in the *Happy Harvest* sample project

Get more tips on 2D lighting from this article by Martin Reinmann of Odd Bug Studio, and this page on 2D light and shadow techniques in the Universal Render Pipeline.

# Create rich free-form 2D environments with 2D Sprite Shape

2D Sprite Shape gives you the freedom to create rich free-form 2D environments with a visual and intuitive workflow. It tiles sprites along a shape's outline, automatically deforming and swapping them based on the outline angle.


2d-rpg-demo - Scene_Battle_Skeletons - PC, Mac & Linux Standalone - Unity 2020.2.1f1* <DX11>
File Edit Assets GameObject Component Cinemachine Dragon Crashers Jobs Window Help
Center
Global
Preview Packages in Use
Account
Layers
Tall
Scene
Game
Shaded
2D
Gizmos
All
Inspector
Hierarchy
Project
Add Component

P_Bridge ☑ — Static
Tag: Untagged — Layer: Default
Prefab: Open | Select

Transform
Position: X -5.37, Y 0.6334858, Z 0
Rotation: X 0, Y 0, Z 0
Scale: X 1, Y 1, Z 1

Sprite Shape Renderer ☑
Color
Mask Interaction: None
Fill Material: Sprite-Lit-Default
Edge Material: Sprite-Lit-Default
Additional Settings
Sorting Layer: Main
Order in Layer: -2
Rendering Layer Mask: Layer1

Sprite Shape Controller (Script) ☑
Profile: SpriteShape_Bridge (Sprite Shape)
Edit Spline
Tangent Mode
Position: X -0.062644, Y -2.244127
Height: 1
Corner: Automatic
Sprite Variant: 0
Snapping: □
Spline
Detail: High Quality
Is Open Ended: ☑
Adaptive UV: ☑
Enable Tangents: □
Cache Geometry: □
Corner Threshold: 30
Fill
Fill Tessellation (C# Job): □
Stretch UV: □
Pixel Per Unit: 100
Worldspace UV: □

Sprite-Lit-Default (Material)
Shader: Universal Render Pipeline/2D/Sprite-Lit-De — Edit...

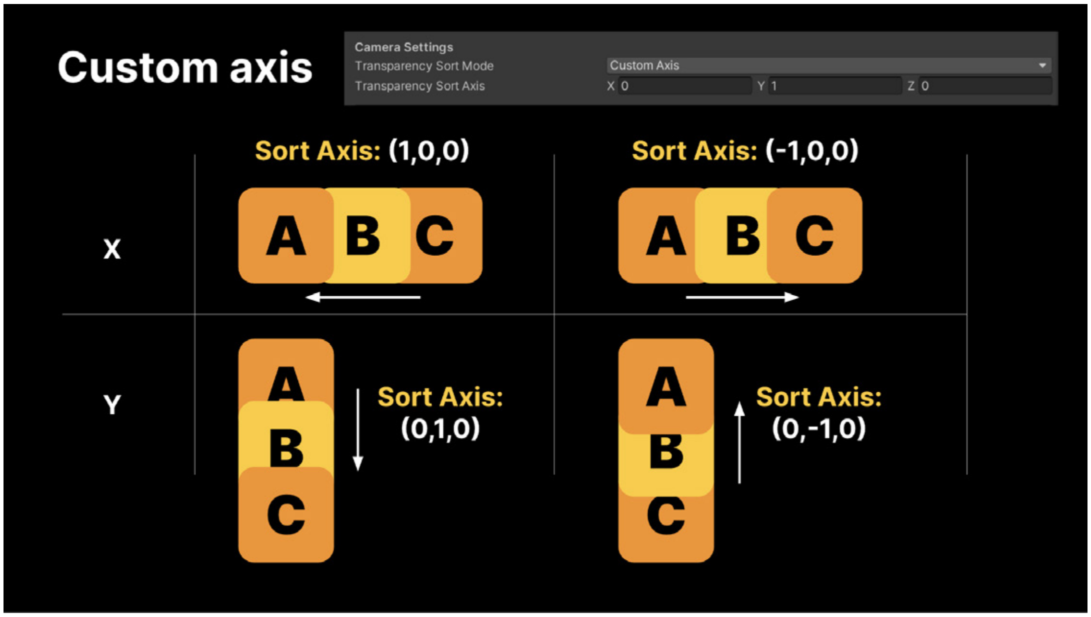
One example of how 2D Sprite Shape is used in the *Dragon Crashers* 2D sample project

Sprite Shape also autogenerates a Polygon Collider 2D if you want to use paths or shapes with the Physics 2D system.

## Custom Sprite Sort Axis

Sort your sprites based on your preferred direction. This can be helpful if you have a number of sprites within the same layer and sorting order and you need to resolve the sorting in a particular way. For example, imagine a card game where the individual cards overlap a bit and you want the card at the bottom of the screen to be rendered on top of the others.

In URP, in the 2D Renderer Asset, select **General > Transparency Sort Mode**, and choose **Custom Axis**. For example, use (0, 1, 0) for the **Transparency Sort Axis** to sort along the Y axis from top to bottom.



Camera Settings
Transparency Sort Mode: Custom Axis
Transparency Sort Axis: X 0 Y 1 Z 0

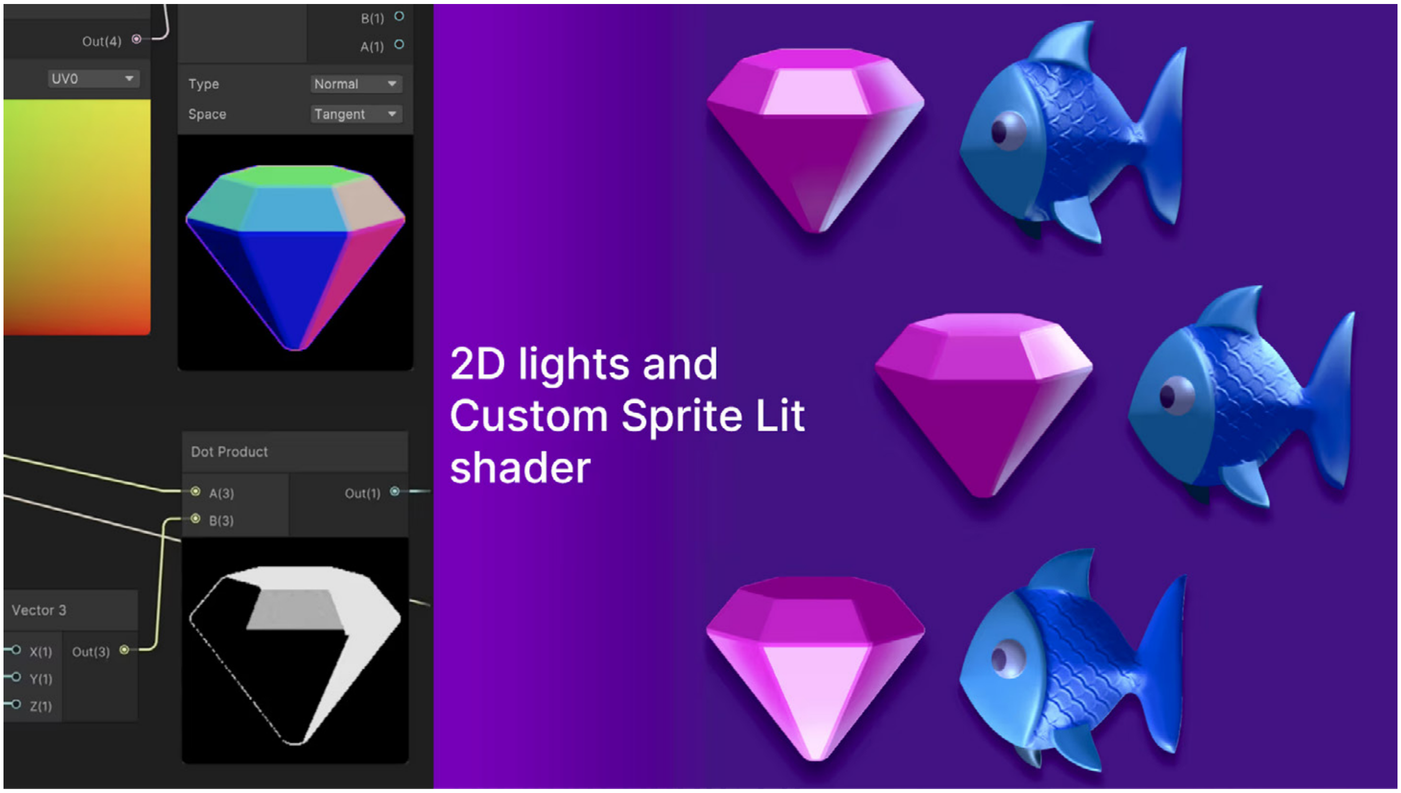
Transparency Sort Mode and Sort Axis example; check this video tutorial for other tips (based on a previous Unity version but still applicable)

## Create custom lighting with Sprite Custom Lit shaders

If you're looking to create lighting effects that are independent of global scene lights, consider using a Sprite Custom Lit shader.

It's one of the techniques used for creating the visual effects in the sample *Gem Hunter Match*. This shader substitutes for scene lighting, allowing the team to modify the 2D light texture information and control the lighting on each piece. The result is creative illumination of the sprites, like the shimmery effect that moves over the pieces.

The light position data is moved into the shader, eliminating the need for actual light objects in the scene, which also helps to keep it neat. The encapsulated per-object lighting in the shader works well for better isolation and editing at scale and improves performance where batching is possible.

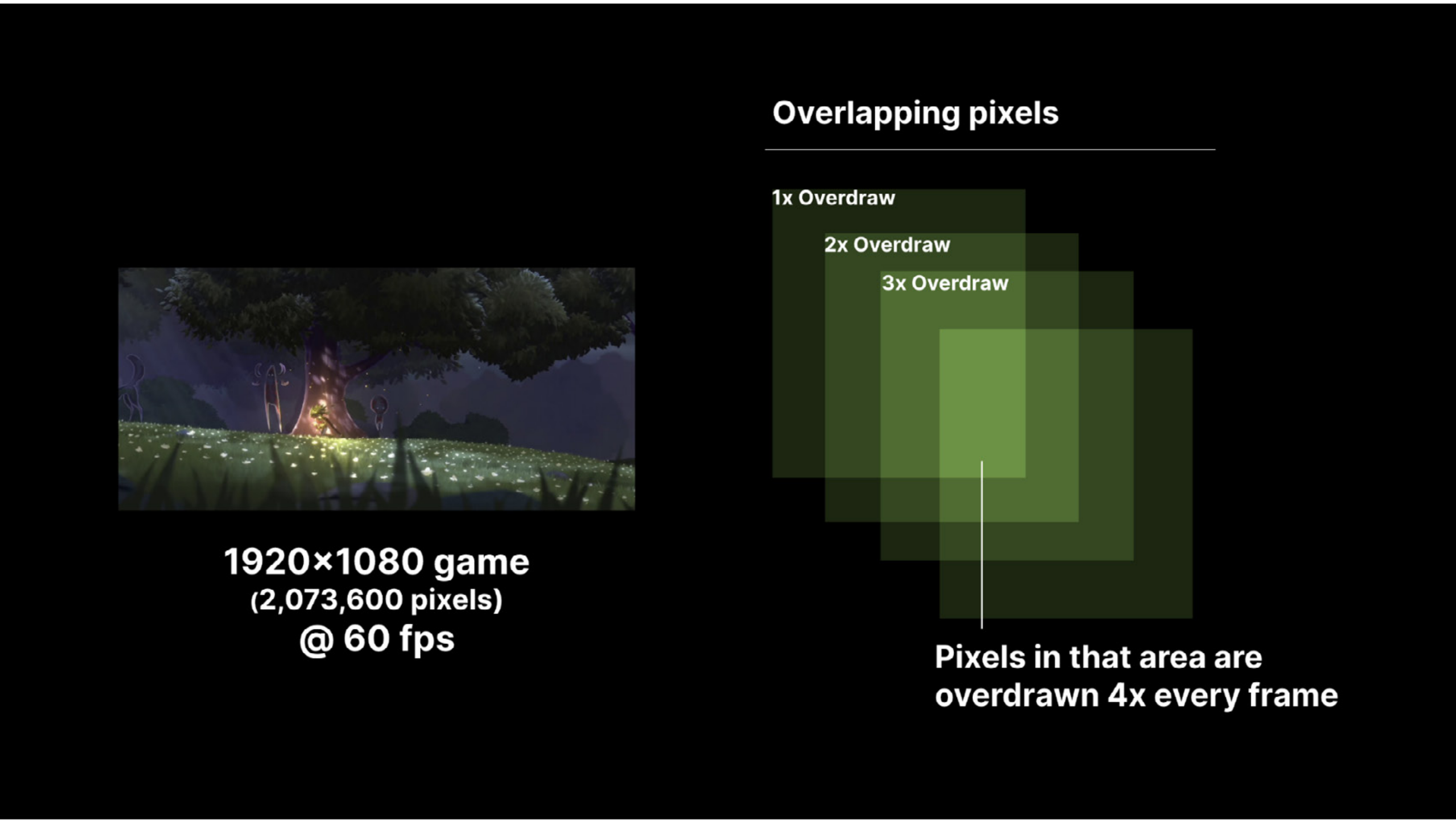


The Sprite Custom Lit Shader in *Gem Hunter Match*

Learn how the team used the Custom Sprite Lit Shader for full creative control with per-object lighting on 2D sprites in the *Find a treasure trove of lighting and visual effects in our new match-3 sample Gem Hunter Match* blog post.

## Reduce overdraw of transparency pixels

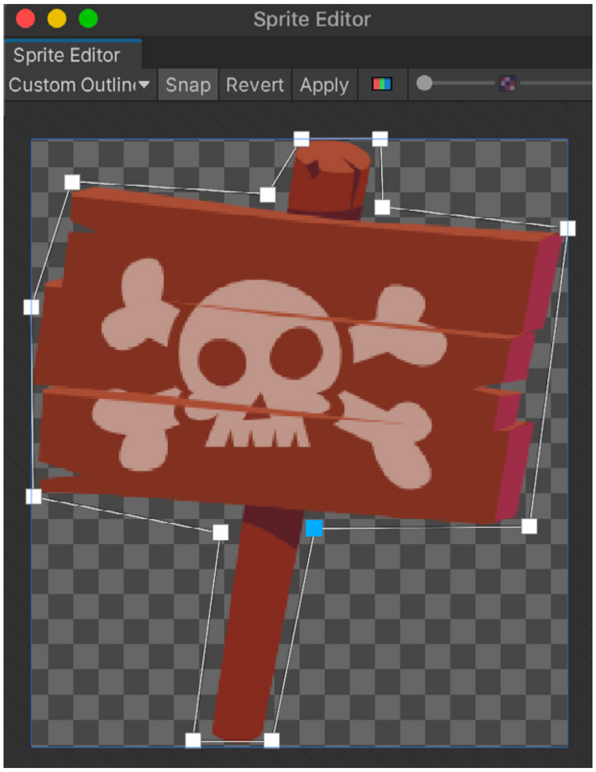
Avoid overdrawing of pixels to improve performance. Switch the **Mesh Type** to **Tight** (default option) in the Import Settings. Merge overlapping graphics in a single sprite whenever possible, and try to disable sprites that could be in a background layer with no use in the game. This reduces the overdraw area and potential overlap with neighboring sprites.



Reduce overdraw between sprites.

## Minimize unused areas with the Sprite Editor

You can also define a custom outline around each sprite using the 2D Sprite Editor; this can minimize the unused areas when packing the sprites and avoid overdraw.

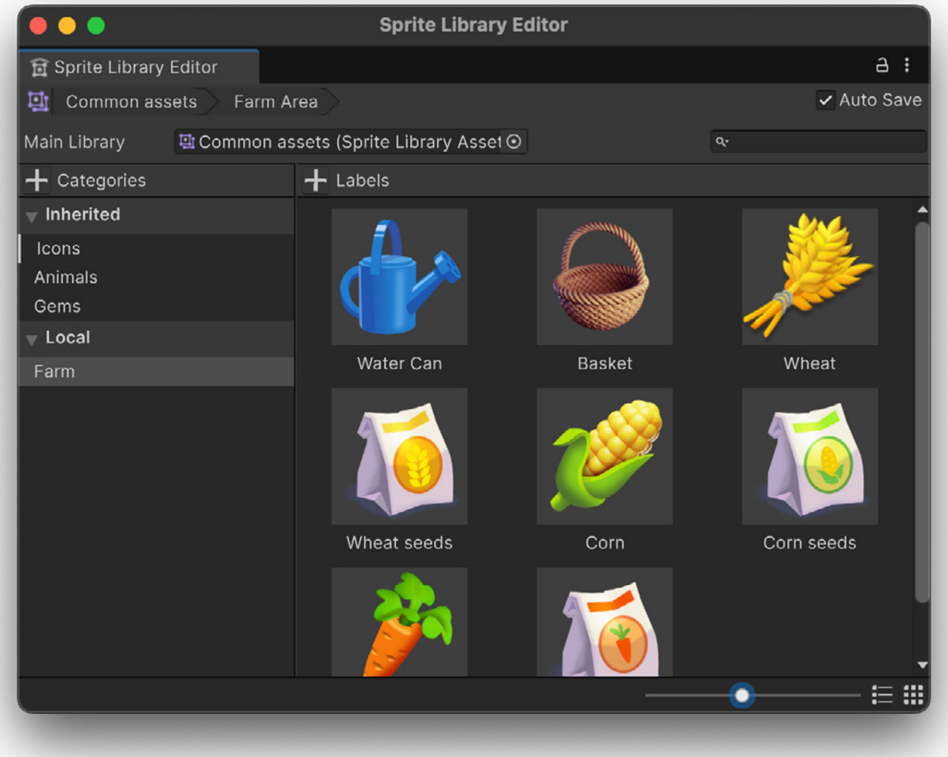


The Sprite Editor with a custom outline

## Organize your project sprites and animation with Sprite Libraries

Make use of the Sprite Libraries to easily organize your sprites for level design and multi-part characters.

Sprite Libraries are different from Sprite Atlases that are designed to group for rendering performance. However, because Sprite Libraries, Sprite Atlases, and Addressables are compatible, you can use them in combination to keep your larger projects organized.



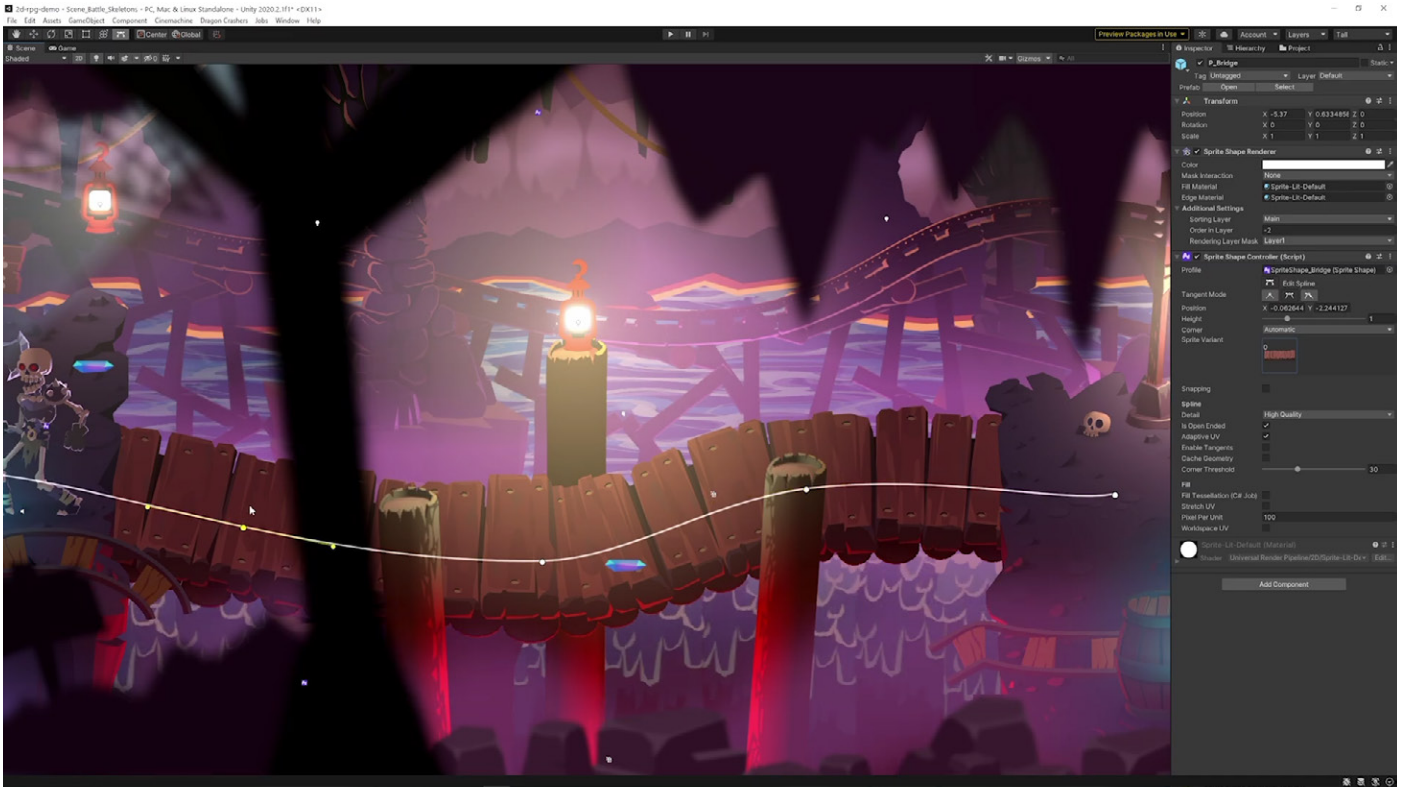
The Sprite Library Editor window

## Modifying Sprite Shapes control points

The Sprite Shape can help you design organic 2D terrain, creating hills, dynamic roads, trails, racetracks, or bodies of water, as well as decorative elements.

You can also create gameplay features – combine with Unity 2D physics to create interactive regions (e.g., destructible terrain or deformable platforms) that can adapt and reshape in real time.

You can modify the control points of the sprite shape; the API gives you access to the points at runtime or Editor time. This has a performance cost, but when it's carefully integrated it can give your game an extra edge.


2d-rpg-demo - Scene_Battle_Skeletons - PC, Mac & Linux Standalone - Unity 2020.2.1f1* <DX11>
File Edit Assets GameObject Component Cinemachine Dragon Crashers Jobs Window Help
Center
Global
Scene
Game
Shaded
2D
Gizmos
Preview Packages in Use
Account
Layers
Tall
Inspector
Hierarchy
Project
Static
Open
Select
Edit Spline
Add Component

P_Bridge

Tag: Untagged | Layer: Default

Prefab

**Transform**

| | | | |
|---|---|---|---|
| Position | X -5.37 | Y 0.6334856 | Z 0 |
| Rotation | X 0 | Y 0 | Z 0 |
| Scale | X 1 | Y 1 | Z 1 |

**Sprite Shape Renderer**

| | |
|---|---|
| Color | |
| Mask Interaction | None |
| Fill Material | Sprite-Lit-Default |
| Edge Material | Sprite-Lit-Default |
| **Additional Settings** | |
| Sorting Layer | Main |
| Order in Layer | -2 |
| Rendering Layer Mask | Layer1 |

**Sprite Shape Controller (Script)**

| | |
|---|---|
| Profile | SpriteShape_Bridge (Sprite Shape) |
| Tangent Mode | |
| Position | X -0.062644 Y -2.244127 |
| Height | 1 |
| Corner | Automatic |
| Sprite Variant | 0 |
| Snapping | □ |
| **Spline** | |
| Detail | High Quality |
| Is Open Ended | ☑ |
| Adaptive UV | ☑ |
| Enable Tangents | □ |
| Cache Geometry | □ |
| Corner Threshold | 30 |
| **Fill** | |
| Fill Tessellation (C# Job) | □ |
| Stretch UV | □ |
| Pixel Per Unit | 100 |
| Worldspace UV | □ |

Sprite-Lit-Default (Material)

Shader: Universal Render Pipeline/2D/Sprite-Lit-De

Modifying Sprite Shapes control points in the *Dragon Crashers* - 2D URP sample project.

# Swap sprites conveniently from the contextual menu

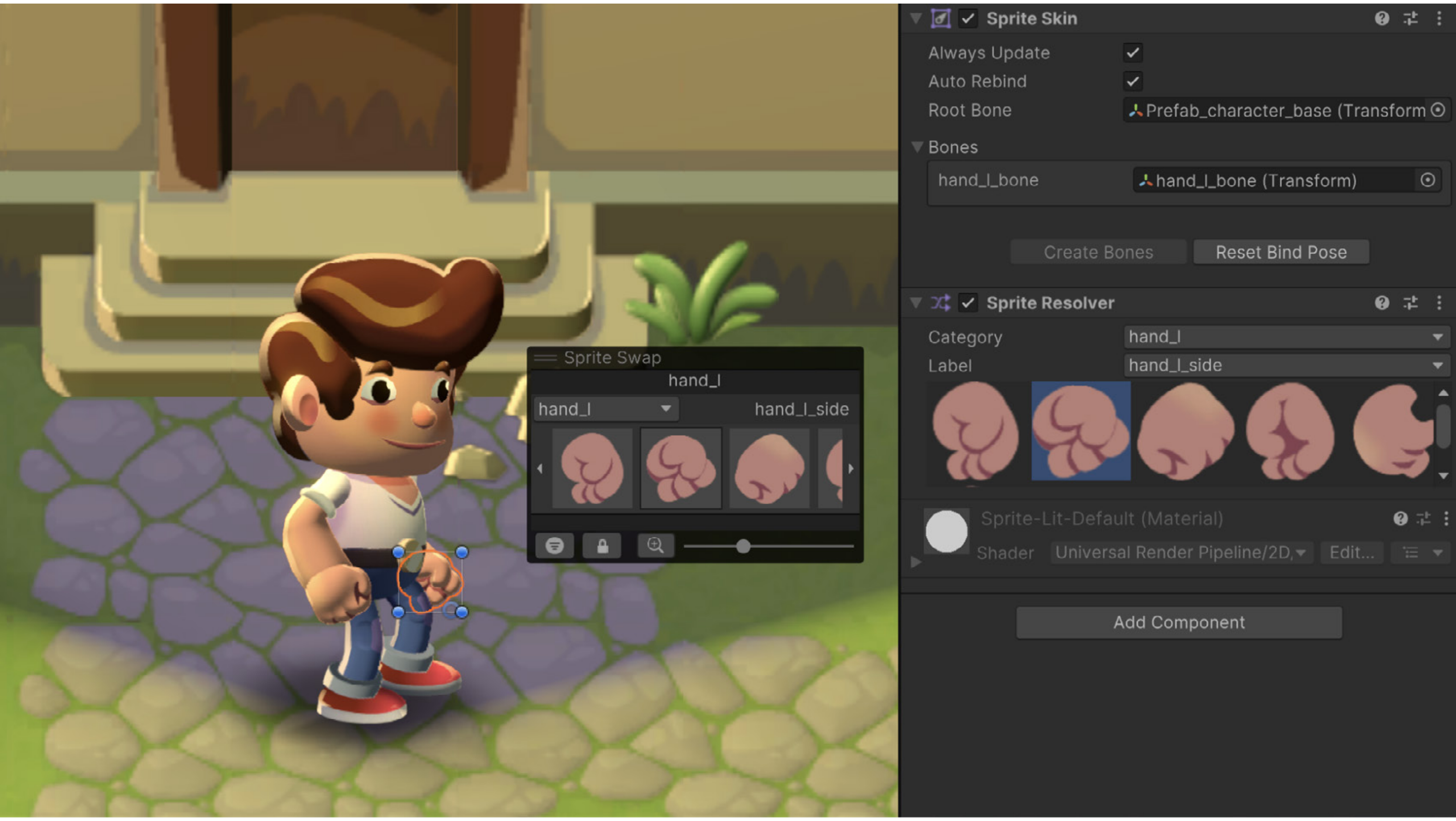
The Sprite Resolver is a component that works alongside the 2D Sprite Library system and provides functionality to dynamically swap between different sprite visuals at runtime.

It allows you to change the sprite of a GameObject without manually assigning it in the Inspector, enabling more powerful and flexible sprite management for your projects.

Here are some of the best use cases for the Sprite Resolver:

- Character customization systems
- Dynamic facial expressions or moods for NPCs or characters
- Environmental sprite changes based on seasons or in-game events
- Visual upgrades or degradations of objects
- Streamlining sprite swaps in animations for greater efficiency and organization

In Unity 6 you can make use of Sprite Resolvers in the Inspector window, as well as in the Editor layout.



Sprite Skin

Always Update ☑

Auto Rebind ☑

Root Bone: Prefab_character_base (Transform

Bones

hand_l_bone: hand_l_bone (Transform)

Sprite Resolver

Category: hand_l

Label: hand_l_side

Sprite-Lit-Default (Material)

Shader: Universal Render Pipeline/2D,

The Sprite Resolver component in the Sprite Library

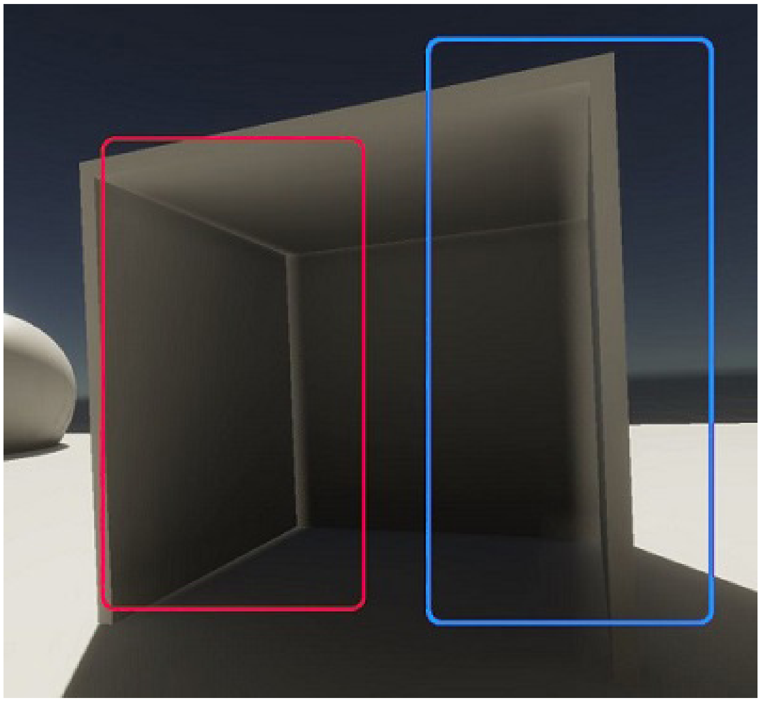
# Graphics and art assets

## Light leaks with light probes

Light leaks often occur when geometry receives light from a light probe that isn't visible to the geometry, for example due to the light probe being on the other side of a wall.

This is a complex topic, but here are a few techniques that could help fix light leaks:

- Create thicker walls
- Add an Adaptive Probe Volumes Options Override to your scene
- Enable Rendering Layers
- Adjust Baking Set properties
- Use a Probe Adjustment Volume

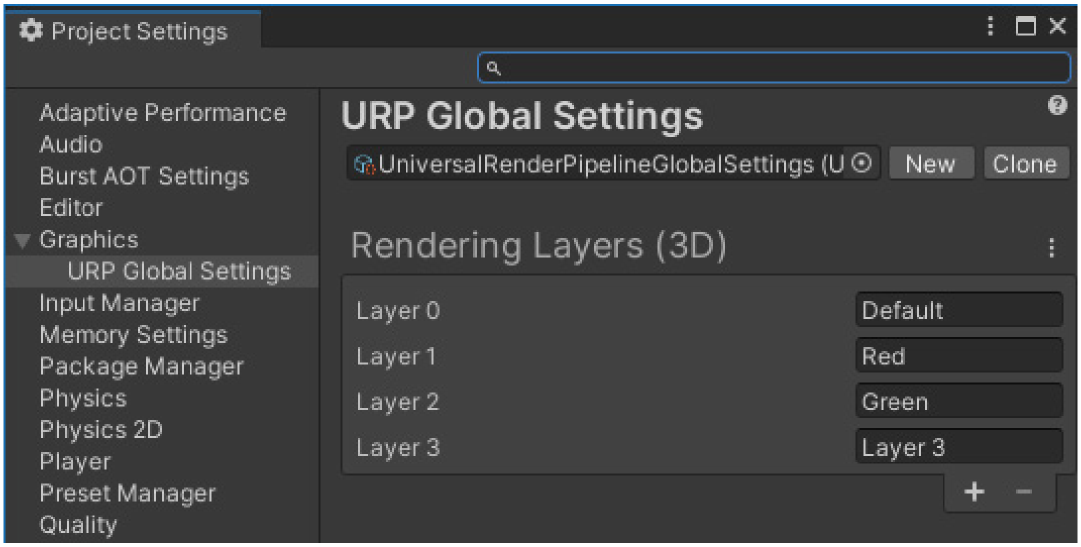


An example of a light leak

## Configure specific lights for specific GameObjects with Rendering Layers

The Rendering Layers feature lets you configure certain lights to affect only specific GameObjects.

With the Custom Shadow Layers property, you can configure certain GameObjects to cast shadows only from specific lights (even if those lights do not affect the GameObjects).



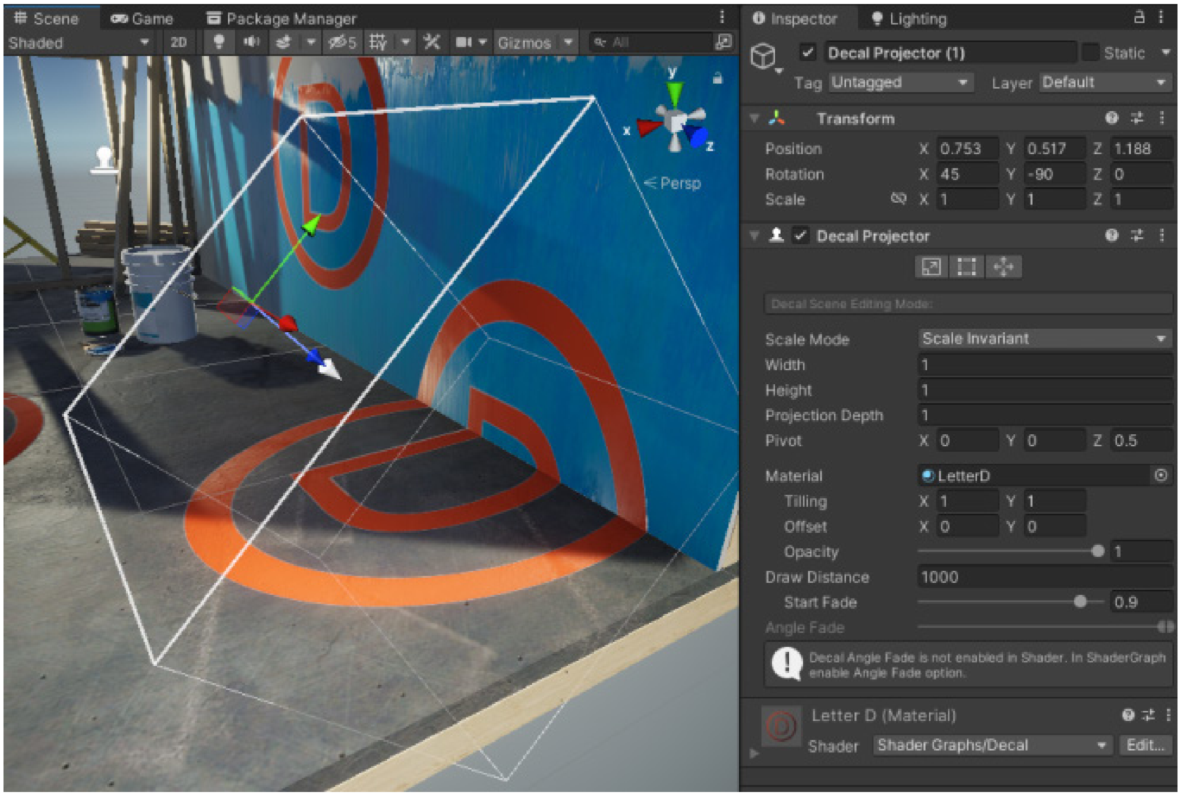
| | |
|---|---|
| Layer 0 | Default |
| Layer 1 | Red |
| Layer 2 | Green |
| Layer 3 | Layer 3 |

Rendering Layers in the URP Global settings

# Add details to meshes with the Decal Projector

The Decal Projector component provides you with a great way of adding detail to a mesh. Use them for elements such as bullet holes, footsteps, signage, cracks, and more.

Because they use a projection framework, they conform to an uneven or curved surface. To use a Decal Projector with URP, you need to locate your Renderer Data asset and add the **Decal Renderer Feature**.



☑ Decal Projector (1) □ Static
Tag Untagged Layer Default

Transform
Position X 0.753 Y 0.517 Z 1.188
Rotation X 45 Y -90 Z 0
Scale X 1 Y 1 Z 1

☑ Decal Projector
Decal Scene Editing Mode:
Scale Mode Scale Invariant
Width 1
Height 1
Projection Depth 1
Pivot X 0 Y 0 Z 0.5
Material LetterD
Tilling X 1 Y 1
Offset X 0 Y 0
Opacity 1
Draw Distance 1000
Start Fade 0.9
Angle Fade
Decal Angle Fade is not enabled in Shader. In ShaderGraph enable Angle Fade option.

Letter D (Material)
Shader Shader Graphs/Decal Edit...

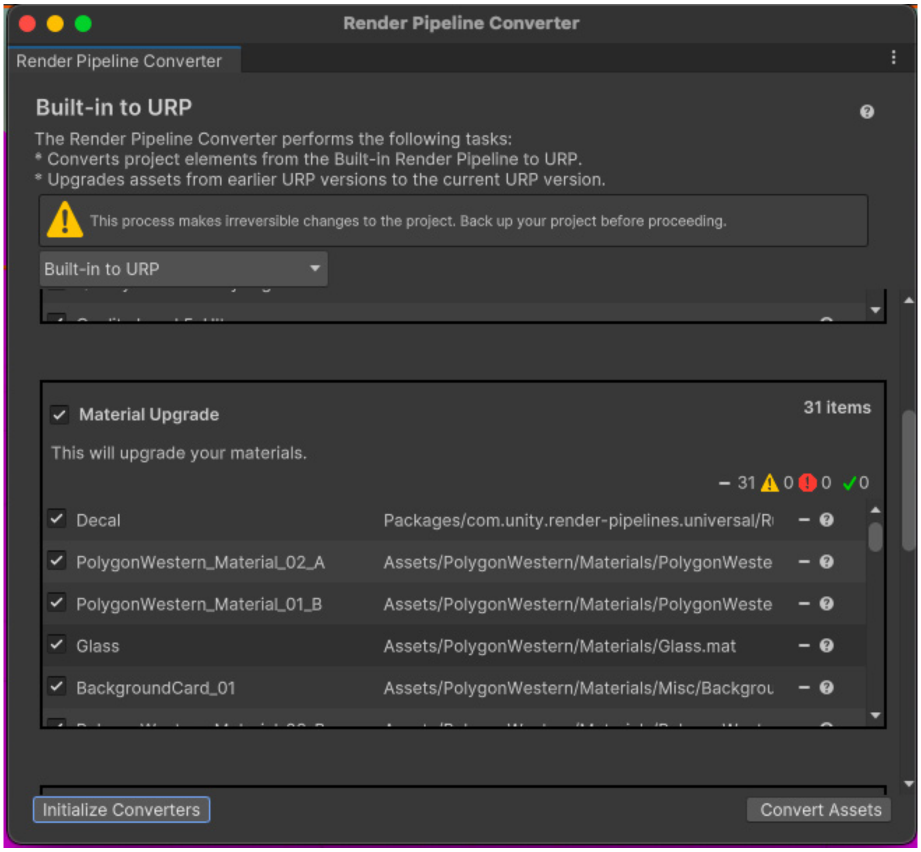
A Decal Projector in the Scene view

# Convert custom shaders from the Built-In Render Pipeline to URP

Refer to this table in the URP documentation to see how each URP shader maps to its Built-In Render Pipeline equivalent.

Once you select one or more of the converters, either click **Initialize Converters** or **Initialize And Convert**. Whichever option you choose, the project will be scanned and those assets that need converting will be added to each of the converter panels.

If you choose **Initialize Converters** you can limit the conversions by deselecting the items using the checkbox provided for each one. At this stage, click **Convert Assets** to start the conversion process. If you choose **Initialize And Convert**, the conversion starts automatically after the converters are initialized. Once it's complete you might be asked to reopen the scene that is active in the Editor.



Built-in to URP

The Render Pipeline Converter performs the following tasks:
* Converts project elements from the Built-in Render Pipeline to URP.
* Upgrades assets from earlier URP versions to the current URP version.

This process makes irreversible changes to the project. Back up your project before proceeding.

☑ Material Upgrade — 31 items

This will upgrade your materials.

– 31 ⚠ 0 ❗ 0 ✓ 0

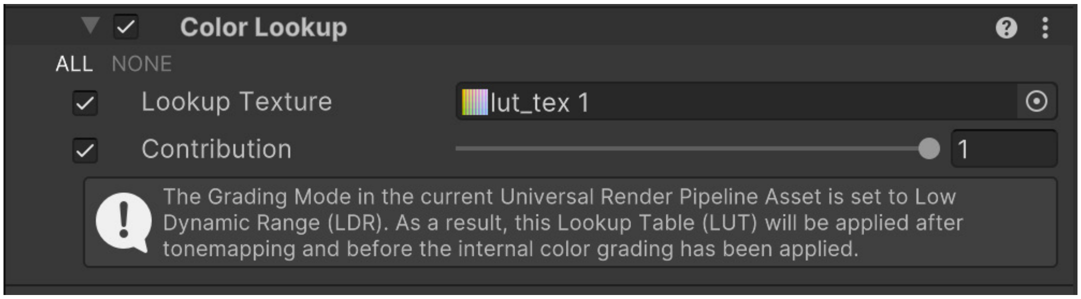
| | | |
|---|---|---|
| ☑ Decal | Packages/com.unity.render-pipelines.universal/R | – ? |
| ☑ PolygonWestern_Material_02_A | Assets/PolygonWestern/Materials/PolygonWeste | – ? |
| ☑ PolygonWestern_Material_01_B | Assets/PolygonWestern/Materials/PolygonWeste | – ? |
| ☑ Glass | Assets/PolygonWestern/Materials/Glass.mat | – ? |
| ☑ BackgroundCard_01 | Assets/PolygonWestern/Materials/Misc/Backgrou | – ? |

The Render Pipeline Converter

## Create color grading with LUT Textures

The Color Grading effect alters or corrects the color and luminance of the final image that Unity produces. You can use this to alter the look and feel of your project.

Use the **Lookup Texture and Contribution** settings to control how the Color Grading effect operates in the post-processing effect that you need to add to your Volume settings.



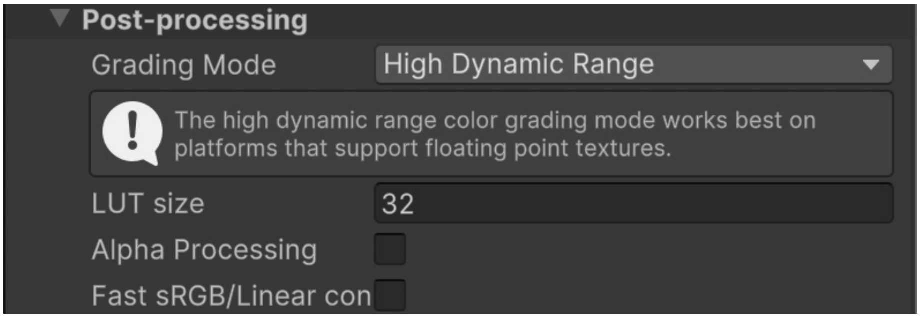
☑ Lookup Texture: lut_tex 1

☑ Contribution: 1

The Grading Mode in the current Universal Render Pipeline Asset is set to Low Dynamic Range (LDR). As a result, this Lookup Table (LUT) will be applied after tonemapping and before the internal color grading has been applied.

The Lookup Texture and Contribution settings in Color Lookup

Volume blending between multiple Low Dynamic Range (LDR) lookup textures is supported but only works correctly if they're the same size. For this reason, it's recommended to stick to a single LUT size for the whole project (256×16 or 1024×32).



Post-processing

Grading Mode: High Dynamic Range

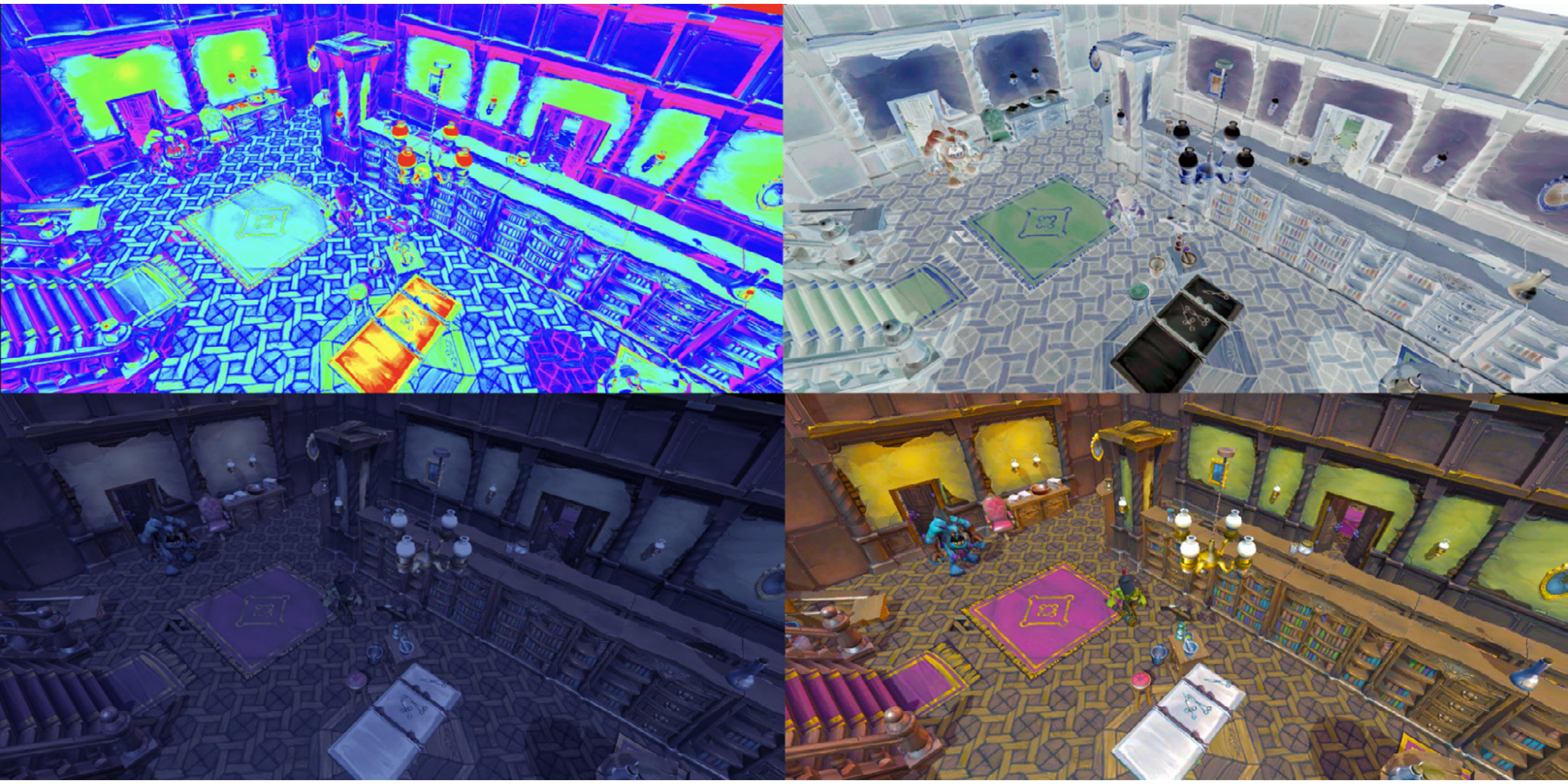
The high dynamic range color grading mode works best on platforms that support floating point textures.

LUT size: 32

Alpha Processing: □

Fast sRGB/Linear con: □

Setting the LUT size in the URP Asset settings

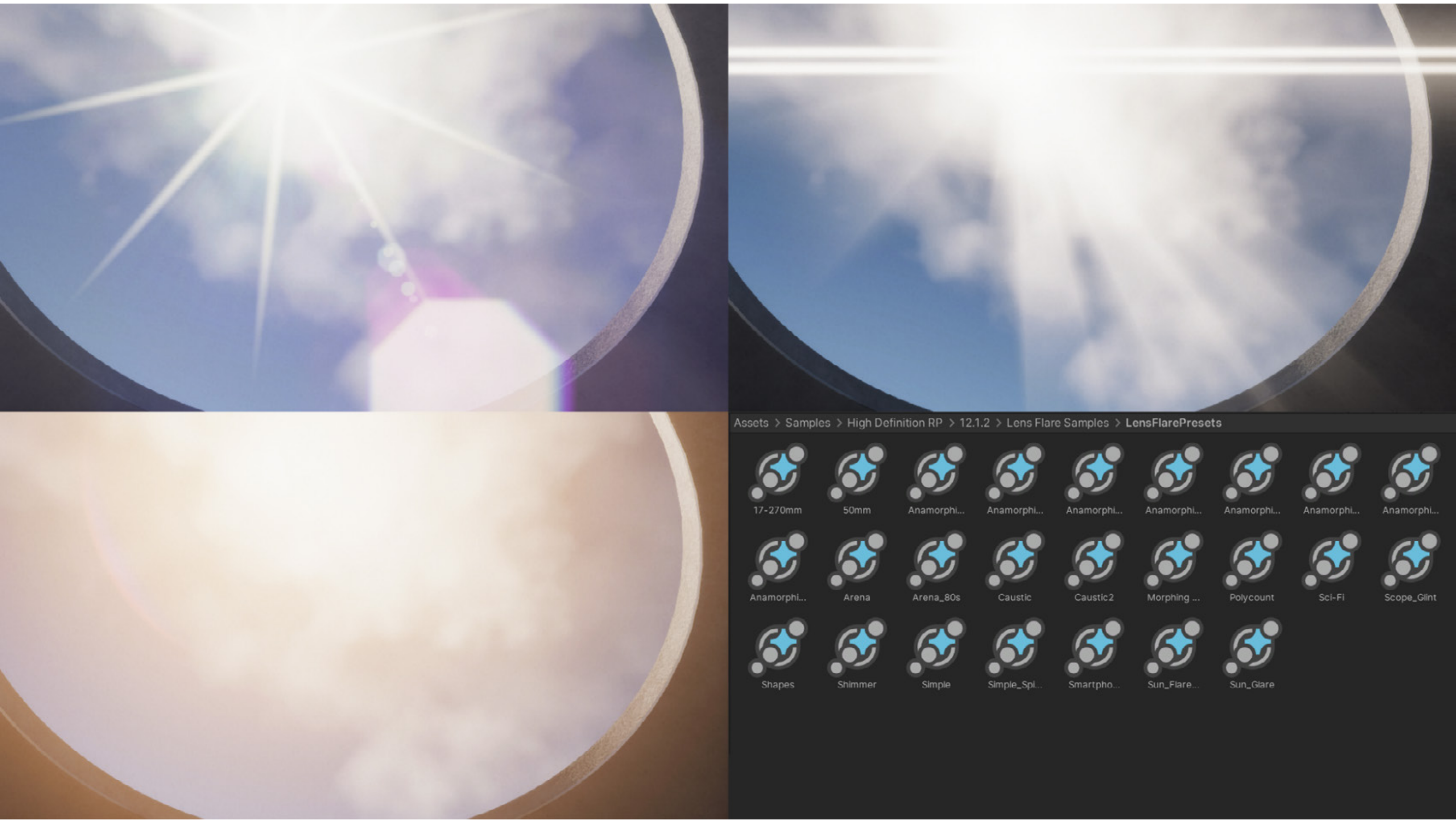


Using various LUT textures

# Create realistic lens effects and stylized looks with lens flares

A lens flare is an artifact that appears when bright light is shining onto a camera lens. A lens flare can appear as one bright glare or as numerous colored polygonal flares matching the aperture of the camera.

Get familiar with lens flares by installing the samples from the Package Manager. This will add a set of predefined Flare assets to include lens flare effects. It also contains a test scene to browse lens flares and help you build your own.



The lens flare samples come with presets ranging from realistic lens effects to more stylized looks, example using the HDRP sample scene.

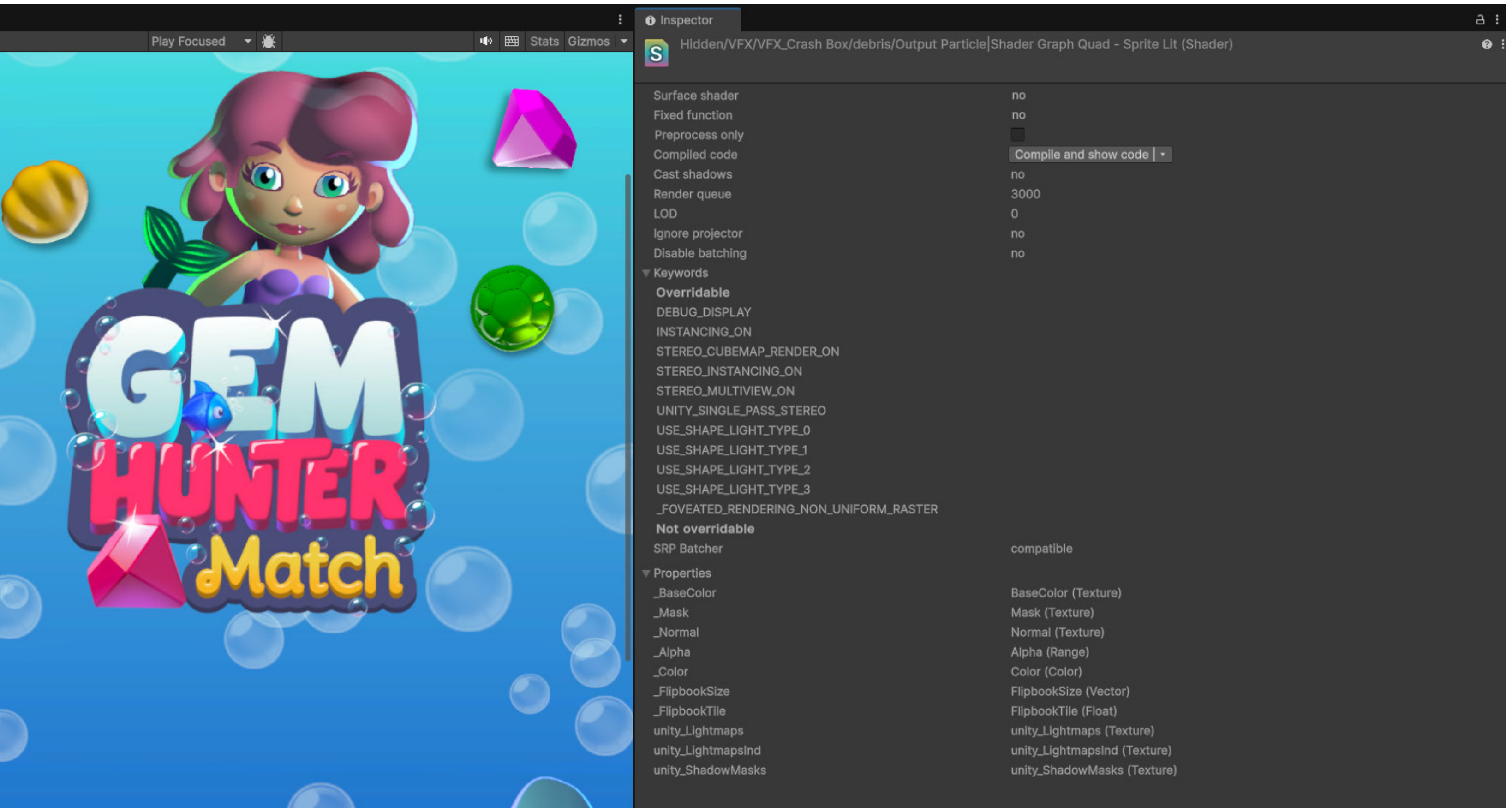
Lens flares are made out of Lens Flare Elements. Each element represents the different artifacts that the flare produces. The element's shape can be a polygon, circle, or a custom image.

The element parameters allow you to tweak the **Color**, **Transform** position, and deformation, or how it's positioned, colored, and scaled when the flare element is part of a Lens Flare effect with multiple elements. Remember that if it's attached to a light source, the flare elements can use the tint color, making it possible to reuse the same Flare asset with many different light sources.

Watch this SIGGRAPH talk to learn more about how lens flares work.

# Manage shader variants

To reduce build size and time manage shader variants carefully. Use #pragma shader_feature instead of #pragma multi_compile when variants are only needed by specific materials. shader_feature variants are stripped unless used by materials or enabled at runtime, while multi_compile variants are always included. Check the keywords in the shader's **Inspector**.



Hidden/VFX/VFX_Crash Box/debris/Output Particle|Shader Graph Quad - Sprite Lit (Shader)

| | |
|---|---|
| Surface shader | no |
| Fixed function | no |
| Preprocess only | □ |
| Compiled code | Compile and show code |
| Cast shadows | no |
| Render queue | 3000 |
| LOD | 0 |
| Ignore projector | no |
| Disable batching | no |

Keywords

**Overridable**

DEBUG_DISPLAY
INSTANCING_ON
STEREO_CUBEMAP_RENDER_ON
STEREO_INSTANCING_ON
STEREO_MULTIVIEW_ON
UNITY_SINGLE_PASS_STEREO
USE_SHAPE_LIGHT_TYPE_0
USE_SHAPE_LIGHT_TYPE_1
USE_SHAPE_LIGHT_TYPE_2
USE_SHAPE_LIGHT_TYPE_3
_FOVEATED_RENDERING_NON_UNIFORM_RASTER

**Not overridable**

| | |
|---|---|
| SRP Batcher | compatible |

Properties

| | |
|---|---|
| _BaseColor | BaseColor (Texture) |
| _Mask | Mask (Texture) |
| _Normal | Normal (Texture) |
| _Alpha | Alpha (Range) |
| _Color | Color (Color) |
| _FlipbookSize | FlipbookSize (Vector) |
| _FlipbookTile | FlipbookTile (Float) |
| unity_Lightmaps | unity_Lightmaps (Texture) |
| unity_LightmapsInd | unity_LightmapsInd (Texture) |
| unity_ShadowMasks | unity_ShadowMasks (Texture) |

Keywords section in the Universal Render Pipeline/Lit (Shader) window in the *Gem Hunter Match* sample

## Create variants of a material

With Material Variants, you can create templates or material prefabs. Based on a base template, you can create variants that share common properties with the template material and override only the properties that differ. If you change common and non-overridden properties in the template material, the changes automatically reflect in the variant material. You can also lock certain properties on materials so they can't be overridden in the variants.

An example of Material Variants; these all share the same base material and only differ in the color property

In a more complex setup, you can create variations of a Material Variant. The material inheritance hierarchy promotes reusability and improves iteration speed and scalability of material authoring in your project.

## Planning asset groups to improve your asset workflow

As part of the preproduction planning, you and your team should agree on aspects of the asset workflow, such as the list of required assets to be created, the budget for them based on your project's technical requirements, and the development cost.

One way you can group required level elements is the following:

- **Large elements**: These elements can't be authored with unique textures but must use tiling or modular elements, such as walls, the ground, cliffs, roofs of buildings, and so on.
- **Medium-sized props**: These are props that have unique texture sheets, like barrels, crates, rocks, and doors.
- **Highly repetitive, small elements**: Think of items used to ground objects or convey scale, including pebbles, leaves, screws, and bolts.

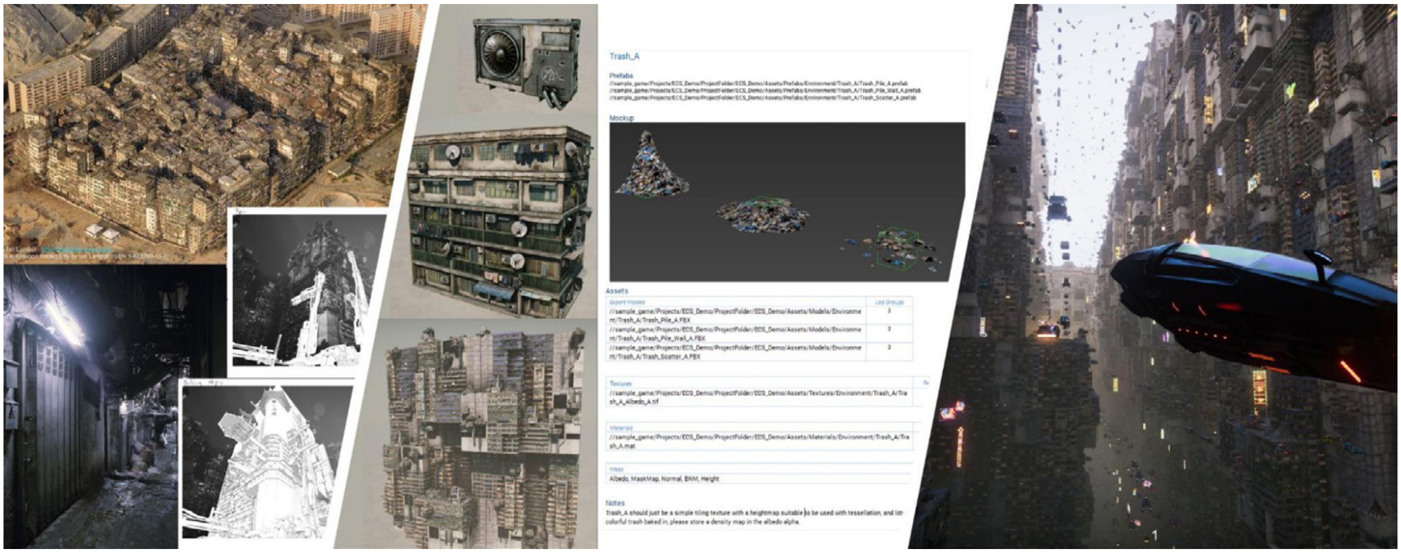
Planning asset production by type, size, and shape

Budget the time, poly count, and texture resolution of each asset, and prioritize the tasks accordingly. Then create non-essential assets and reduce the number of essential assets to the bare minimum.

## Automate and speed up your import pipeline with the AssetPostProcessor API

In a production involving thousands of assets, avoid relying on manually configuring the specifications of each asset from the Inspector.

To automate the process of validating asset files, use the AssetPostProcessor API. This helps you hook into the import pipeline and run scripts prior to, or after, importing assets; just add assets for the scripts to make the necessary changes. You can change the settings from a single script that will automatically reapply the changes to the assets in the project.



Trash_A

Prefabs
//sample_game/Projects/ECS_Demo/ProjectFolder/ECS_Demo/Assets/Prefabs/Environment/Trash_A/Trash_Pile_A.prefab
//sample_game/Projects/ECS_Demo/ProjectFolder/ECS_Demo/Assets/Prefabs/Environment/Trash_A/Trash_Pile_Wall_A.prefab
//sample_game/Projects/ECS_Demo/ProjectFolder/ECS_Demo/Assets/Prefabs/Environment/Trash_A/Trash_Scatter_A.prefab

Mockup

Assets

| Export Models | Lod Groups |
|---|---|
| //sample_game/Projects/ECS_Demo/ProjectFolder/ECS_Demo/Assets/Models/Environment/Trash_A/Trash_Pile_A.FBX | 3 |
| //sample_game/Projects/ECS_Demo/ProjectFolder/ECS_Demo/Assets/Models/Environment/Trash_A/Trash_Pile_Wall_A.FBX | 3 |
| //sample_game/Projects/ECS_Demo/ProjectFolder/ECS_Demo/Assets/Models/Environment/Trash_A/Trash_Scatter_A.FBX | 3 |

| Textures | Re |
|---|---|
| //sample_game/Projects/ECS_Demo/ProjectFolder/ECS_Demo/Assets/Textures/Environment/Trash_A/Trash_A_Albedo_A.tif | |

| Materials |
|---|
| //sample_game/Projects/ECS_Demo/ProjectFolder/ECS_Demo/Assets/Materials/Environment/Trash_A/Trash_A.mat |

| Maps |
|---|
| Albedo, MaskMap, Normal, BNM, Height |

Notes
Trash_A should just be a simple tiling texture with a heightmap suitable to be used with tessellation, and lot-colorful trash baked in, please store a density map in the albedo alpha.

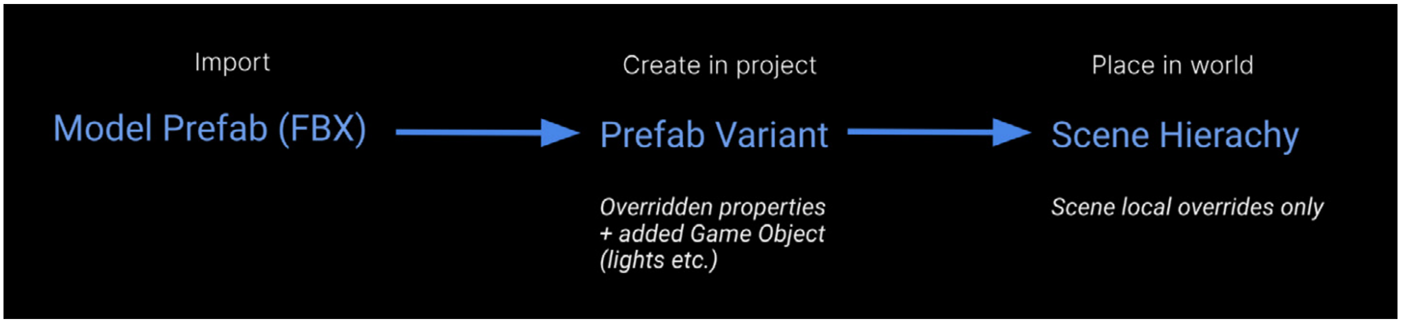
From left to right: Examples of the mood board, concept art, prefab assets, directory, and naming standards using the AssetPostProcessor, and final visuals for the *Megacity* demo by Unity

## Use Prefab Variants for more efficient team work

Another critical aspect of the asset import workflow is the creation of prefabs from 3D models. Directly converting the FBX model into a prefab during production isn't recommended as it will break the prefab if changes are made to the FBX file, such as hierarchy changes in the geometry. This will require you to replace all the instances of it in the scene with an updated version. The solution to this problem lies in Prefab Variants.

Prefab Variants enable the scene-independent production of your art assets, making them ideal to use while prototyping. They are recommended for collaborative iteration while building the game world. They offer the benefit of regular prefabs, like making variations of the original model, but also act as a "weak" reference to the FBX model prefab. This allows you to conveniently add, split, or remove model elements without having to recreate the prefab. Production artists can make changes to the FBX models, and the Prefab Variant will update accordingly. Prefab Variants also allow artists to see art changes in context without altering the project itself.

At the same time, consider using Nested Prefabs to facilitate collaboration on the scene.



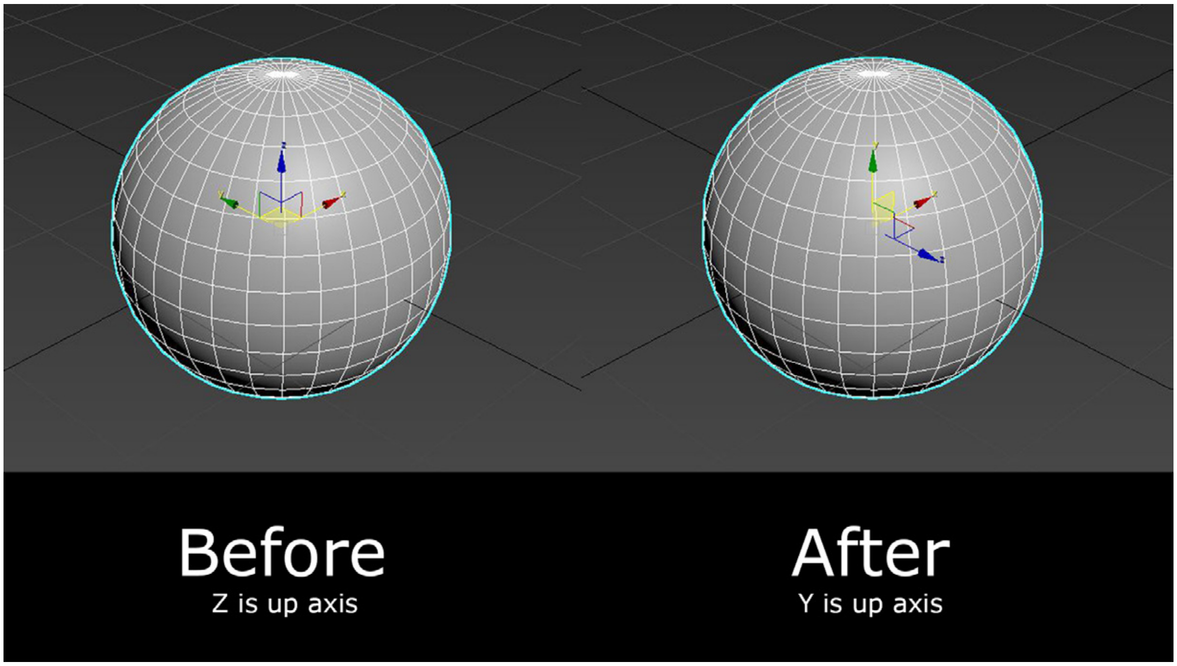
A recommended Prefab Variant workflow for productions with many collaborators

## Asset considerations for XR or mobile development

Decisions that you take early on in the process of creating or sourcing the art assets for your XR or mobile games can save you time later in the development cycle. From small props to characters, follow these tips for efficient art asset creation.

- Start with a clear concept of what you want to create. Concepts can be illustrated with something as simple as hand drawings, particularly at an early stage, to quickly plan out interactions or visuals.
  - In larger studios you might have a concept artist that can bring ideas to life; for individuals or small teams AI can help portray your idea.
- Use 3D modeling software (like Blender or Autodesk's Maya or 3ds Max) to build your model. Keep poly count (number of polygons) in mind for performance, especially for mobile AR. You can create high-poly models initially, and then optimize them later.

- Set the pivot point of the model in your 3D modeling software so that its orientation matches that of Unity. Below is an example of how to set the pivot point in 3ds Max, which uses the z-axis as the up axis.



Adjusting the up axis

- You can also correct the orientation of a model in Unity with these two steps:

  a. Create an empty GameObject in the Hierarchy.

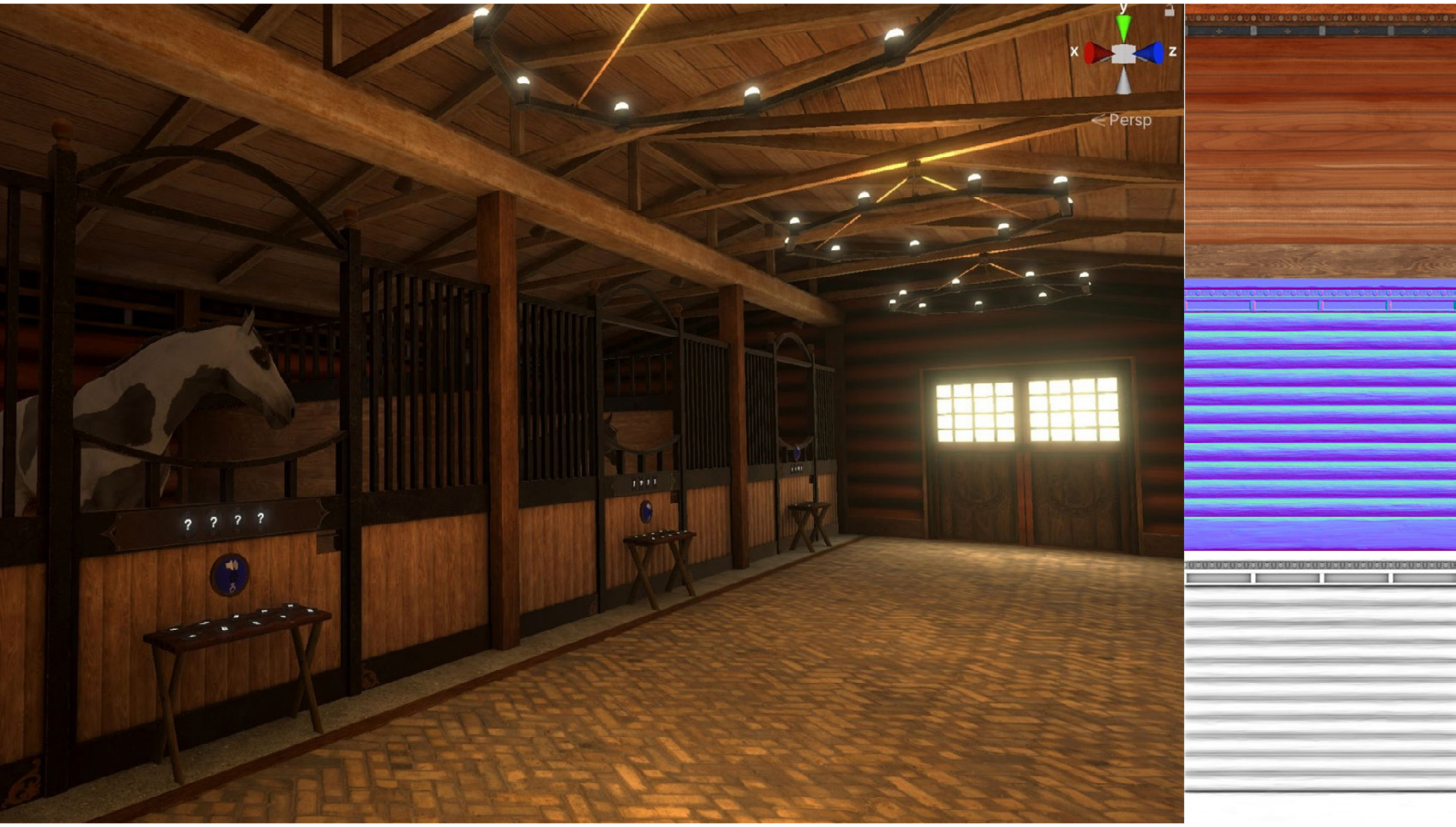
  b. Nest your model within this GameObject, ensuring its position is set to 0,0,0 across all axes to achieve perfect centering.

  This method results in your model being anchored to a parent GameObject that maintains a neutral rotation of 0 across all axes. It also guarantees the model is oriented correctly with the z-axis facing forward and maintains a uniform scale of 1 on all axes. Additionally, you can also try the Bake Axis Conversion option in the Importer.

- Apply textures to give your model a realistic or stylized look. Consider using UV mapping for detailed textures. Depending on what your aesthetic is for your game you might require PBR Textures.

- Physically based (PBR) shaders in Unity typically involve two main components: Metallic-roughness workflow and Specular-glossiness workflow, each handling different aspects of material properties. The result is a more lifelike and dynamic visual appearance, enhancing the realism of 3D scenes and objects in Unity-based games or simulations.

## Populate large textured areas optimally with trim sheets



Use trim sheet for various meshes across an environment.

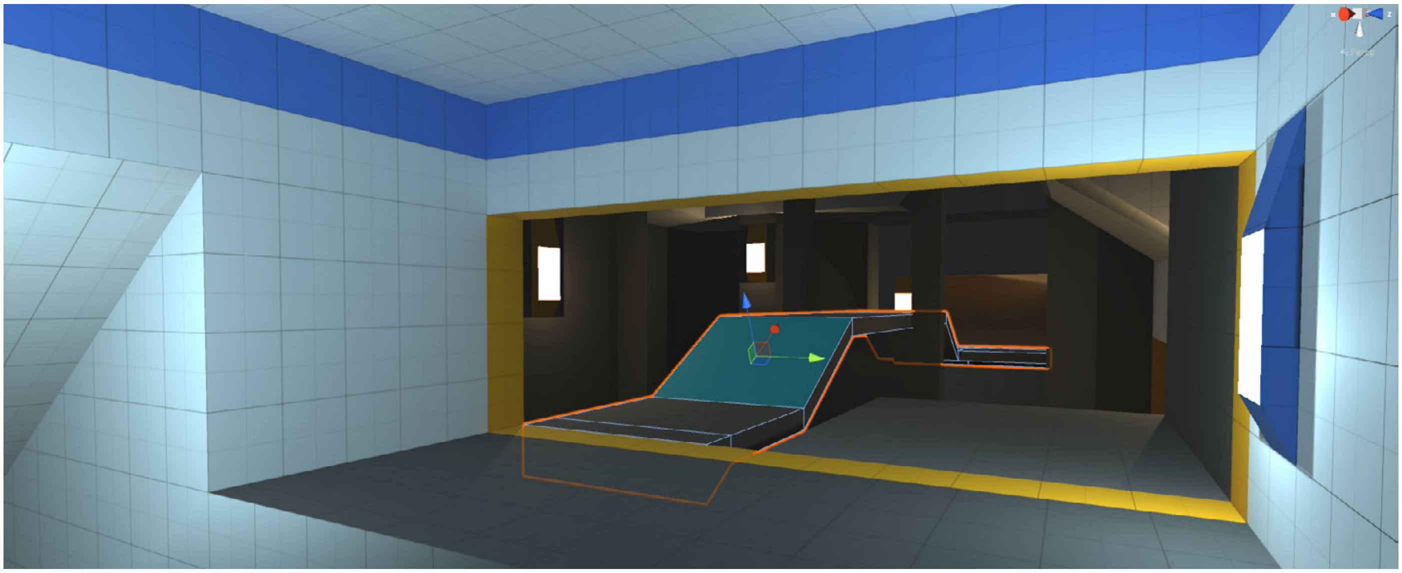
A trim sheet is a texturing tool used in 3D modeling and game development. It consists of a single texture containing a variety of trim patterns and details, like moldings, edges, and borders.

You can use trim sheets for optimizing game performance, as it reduces the number of individual textures required, while still allowing for visually complex and varied designs on the models.

## Export to your 3D models to scale when working in AR and VR

You can prototype and model your 3D meshes in your preferred DCC or directly in Unity with ProBuilder, which is included in the Package Manager. You can mock up levels, scale objects, test how they look in Unity, and export, once the scale is correct, for further refinement to later bring them back to Unity.

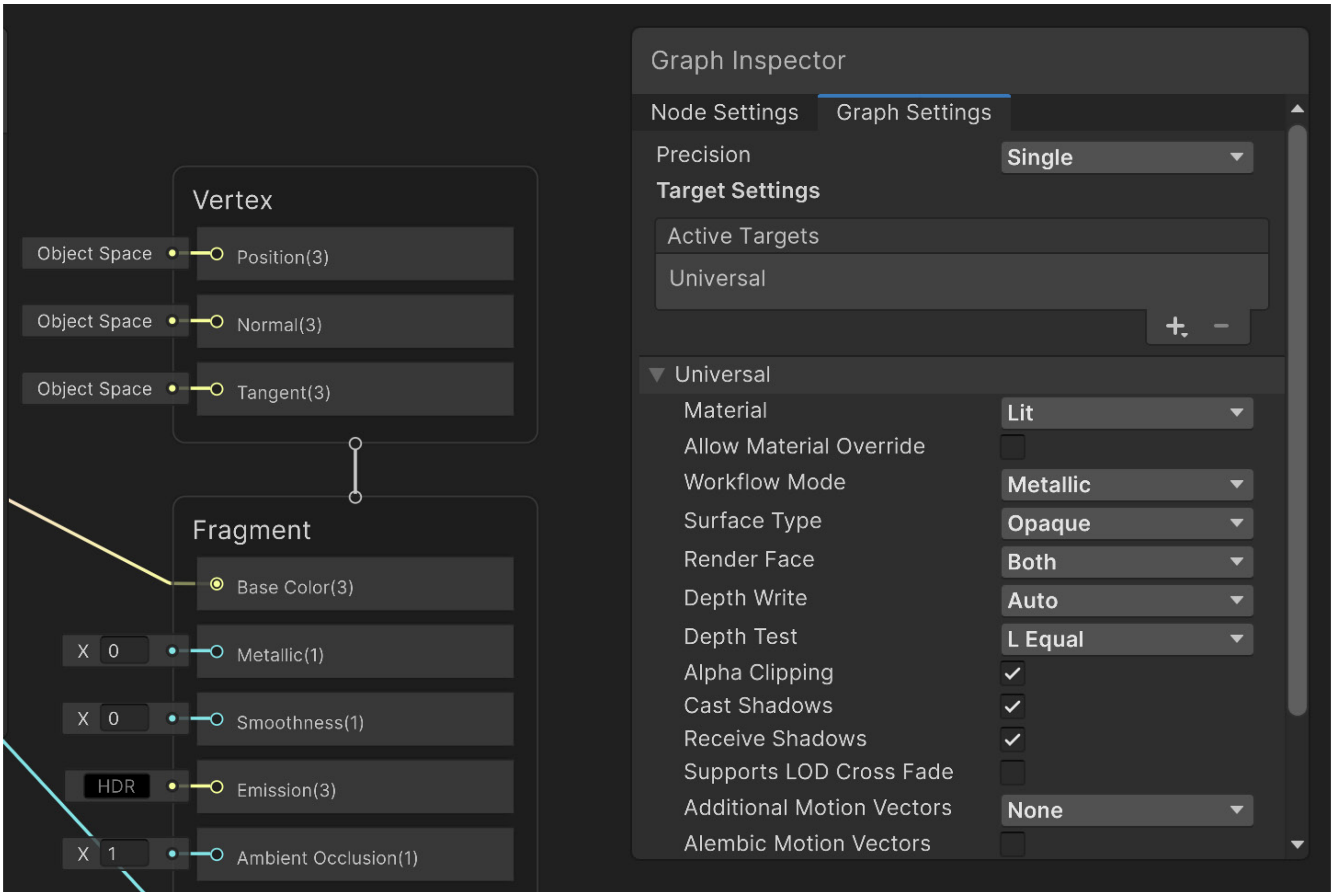
In Unity, the measurement scale is intuitively designed, where 1 unit corresponds precisely to 1 meter. To gauge the scale of your imported assets, simply introduce a cube GameObject into your scene. This cube, with dimensions of 1m x 1m x 1m, serves as an immediate and accurate reference for assessing the scale of any asset. When creating your environments and props the key is to balance aesthetics and performance, ensuring a smooth and engaging AR/VR experience.



You can prototype levels with ProBuilder and then export them with FBX Exporter to 3D modeling software to tweak them.

# Shader Graph for URP and HDRP projects

Creating Lit and/or Unlit shaders in Shader Graph can help ensure your assets are more easily adaptable to use in both URP and HDRP projects. In Shader Graph you can define the render pipeline in the Master Node but keep the logic intact, making it easier to have shaders that work in both pipelines.



Precision: Single

**Target Settings**

Active Targets

Universal

▼ Universal

Material: Lit

Allow Material Override ☐

Workflow Mode: Metallic

Surface Type: Opaque

Render Face: Both

Depth Write: Auto

Depth Test: L Equal

Alpha Clipping ☑

Cast Shadows ☑

Receive Shadows ☑

Supports LOD Cross Fade ☐

Additional Motion Vectors: None

Alembic Motion Vectors ☐

Master stack

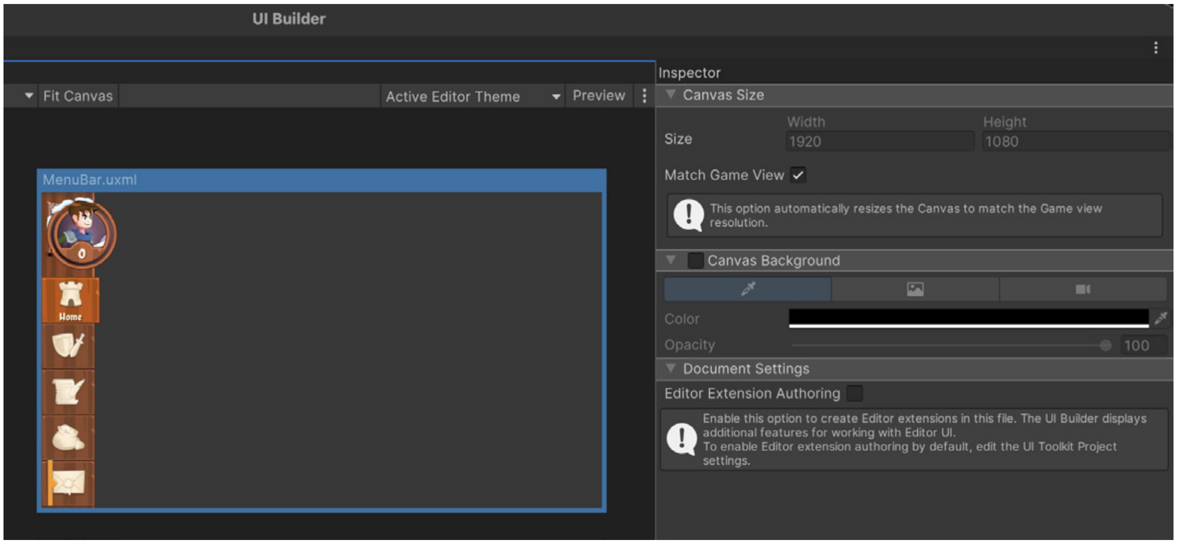
# UI Toolkit

## Design your interface with a visual reference

Enabling the Canvas background can help you visualize your element styling over a color or background image. Select the UXML file in the Hierarchy pane and then choose a Canvas background that approximates the final UI interface to judge style changes in context.

The Canvas background provides a few different options:

- **Background Color**: Represents a specific shade or hue of the game environment
- **Image**: For choosing a sprite or texture as the background (useful for replicating mockup screens or reference art)
- **Camera**: Displays the current gameplay in the background, enabling you to see the UI in context of the actual game



UI Builder

Fit Canvas | Active Editor Theme | Preview

Inspector

Canvas Size

Size — Width: 1920 — Height: 1080

Match Game View ☑

This option automatically resizes the Canvas to match the Game view resolution.

Canvas Background

Color

Opacity 100

Document Settings

Editor Extension Authoring □

Enable this option to create Editor extensions in this file. The UI Builder displays additional features for working with Editor UI.
To enable Editor extension authoring by default, edit the UI Toolkit Project settings.

The Canvas of a UXML document: Use the Color and Image options to adjust its appearance.

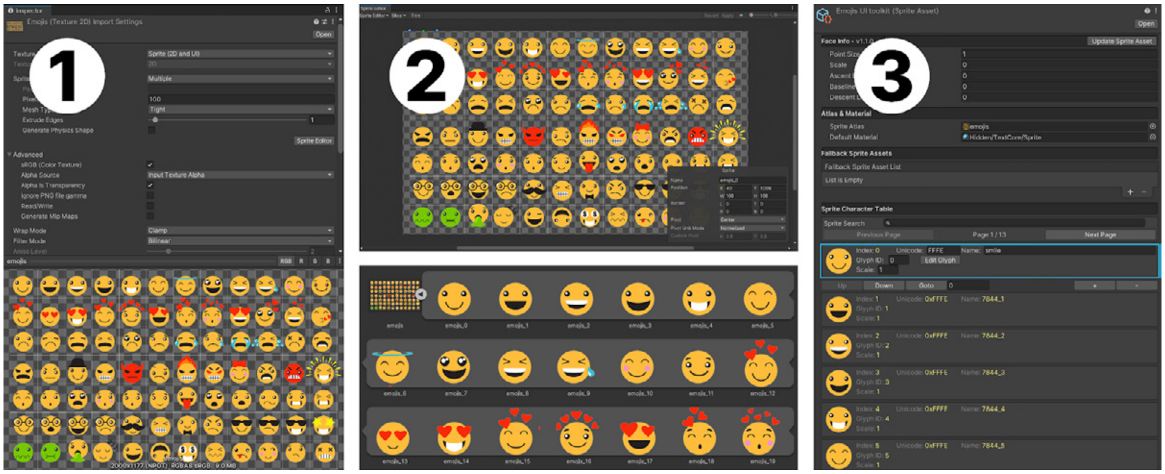
## Iterate faster with PSD Importer when working with Photoshop files

Unity will automatically refresh the sprites included in a multi-layered PSD file every time you save the file with the PSD Importer used to import it into your project. This allows you to create a quick placeholder and iterate on it while viewing changes in the Game view. This can be a great time saver, and improve the quality of the work, by letting you see it in context without swapping files or needing support from a fellow developer in your team.

## Use emojis in your game

You can include sprites like emojis in your text via rich text tags. To use them, you'll need to use a sprite asset similar to the Gradient asset.

When importing multiple sprites, pack them into a single atlas to reduce draw calls. Make sure that the sprite atlas has a suitable resolution for your target platform.



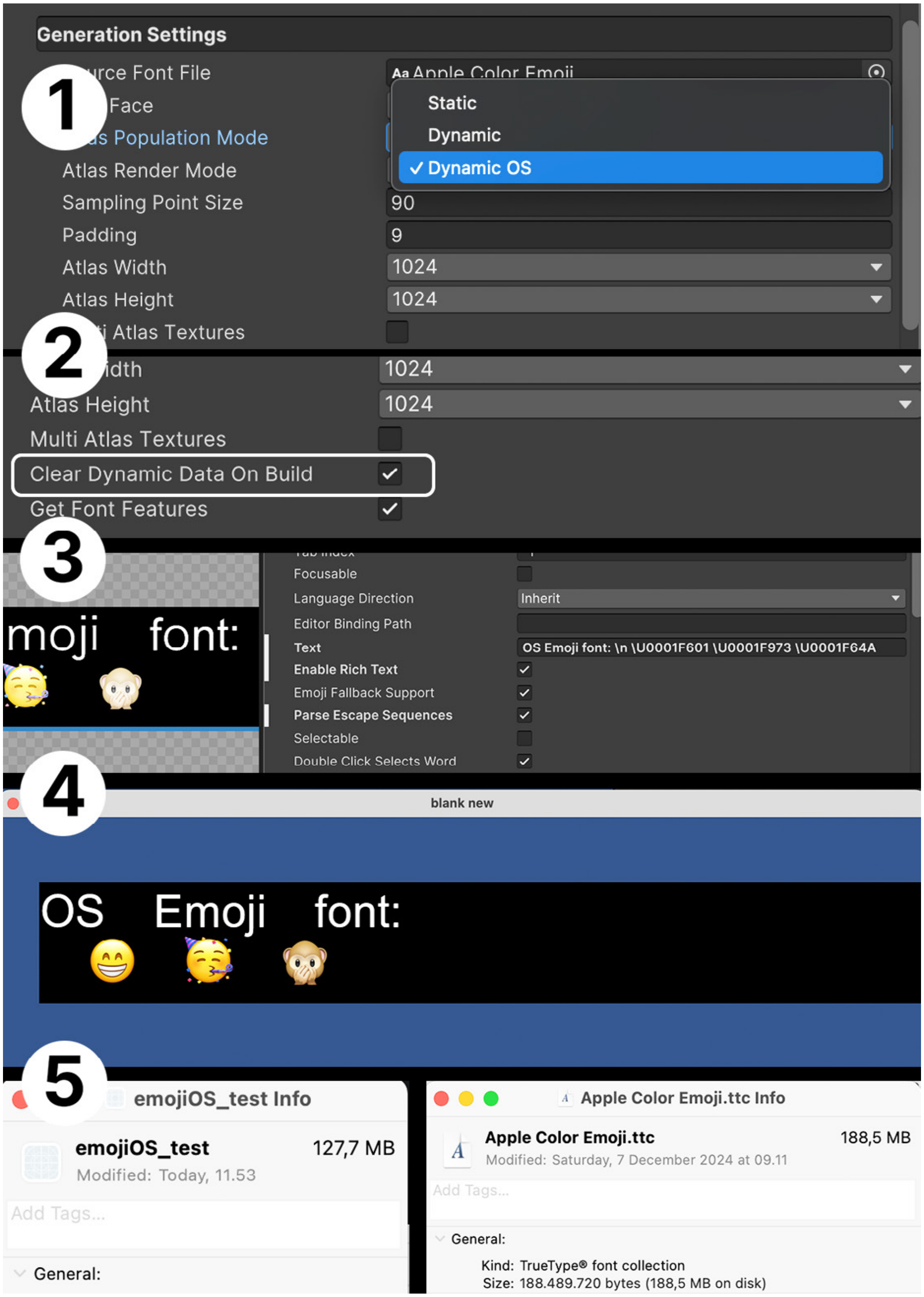
Import the sprites, create the Sprite Asset inside Assets/Resources, and adjust the info of each glyph as needed.

# Use the built-in emojis included with a device's OS

If you are targeting a specific runtime platform, such as iOS or Android, you can make use of the system's built-in emoji font instead of including the source font in your project. This can save memory and eliminate the need to package a large collection of emojis with your application.

These are the steps to use OS emojis in your project:

1. Create a Font asset from the font that your target system uses. On iOS the font is called Apple Emoji (used in this example), and on Android it's called Noto Color Emoji (currently only COLRv0 is supported). Make sure the Font Asset is of the type **Color**, and then set the atlas population mode to **Dynamic OS** which doesn't require you to include the source font in your asset saving space.
2. Ensure **Clean Dynamic Data On Build** is checked on the Font Asset
3. Enable **Parse Escape Sequences** on UI Builder and enter the desired emojis using the emoji keyboard from MacOS or Windows or in UTF format, for example, you would introduce a smiley as `\U0001F601`. You can check the UTF of each emoji in the Character Table of the Font Asset.
4. The build running on MacOS displays the emojis according to the OS font.
5. We can observe that in our test, the **build size** is smaller than the standalone emoji font proving that it was not included in the project but still being used to render the appropriate emojis.



**Generation Settings**

- urce Font File: Aa Apple Color Emoji
- Face
- as Population Mode: Static / Dynamic / ✓ Dynamic OS
- Atlas Render Mode
- Sampling Point Size: 90
- Padding: 9
- Atlas Width: 1024
- Atlas Height: 1024
- i Atlas Textures: □

- idth: 1024
- Atlas Height: 1024
- Multi Atlas Textures: □
- Clear Dynamic Data On Build: ☑
- Get Font Features: ☑

moji font:

- Focusable: □
- Language Direction: Inherit
- Editor Binding Path
- **Text**: OS Emoji font: \n \U0001F601 \U0001F973 \U0001F64A
- **Enable Rich Text**: ☑
- Emoji Fallback Support: ☑
- **Parse Escape Sequences**: ☑
- Selectable: □
- Double Click Selects Word: ☑

blank new

OS Emoji font:
😁 🥳 🙊

emojiOS_test Info

emojiOS_test 127,7 MB
Modified: Today, 11.53
Add Tags...
General:

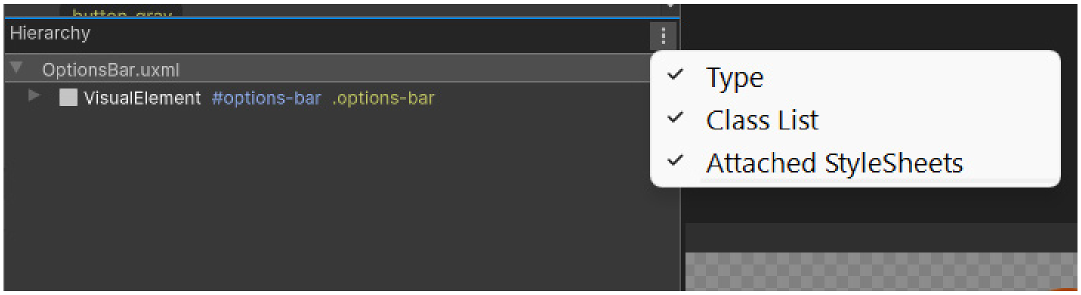
Apple Color Emoji.ttc Info

Apple Color Emoji.ttc 188,5 MB
Modified: Saturday, 7 December 2024 at 09.11
Add Tags...
General:
Kind: TrueType® font collection
Size: 188.489.720 bytes (188,5 MB on disk)

# Show additional info relative to the Visual Element on UI Builder

Click the vertical ellipsis (⋮) in the Hierarchy header to further visualize the UI elements.

In the Hierarchy pane, additional information appears next to the element Type. The **#options-bar** Name selector and **.options-bar** Style Class selector appear when checked.



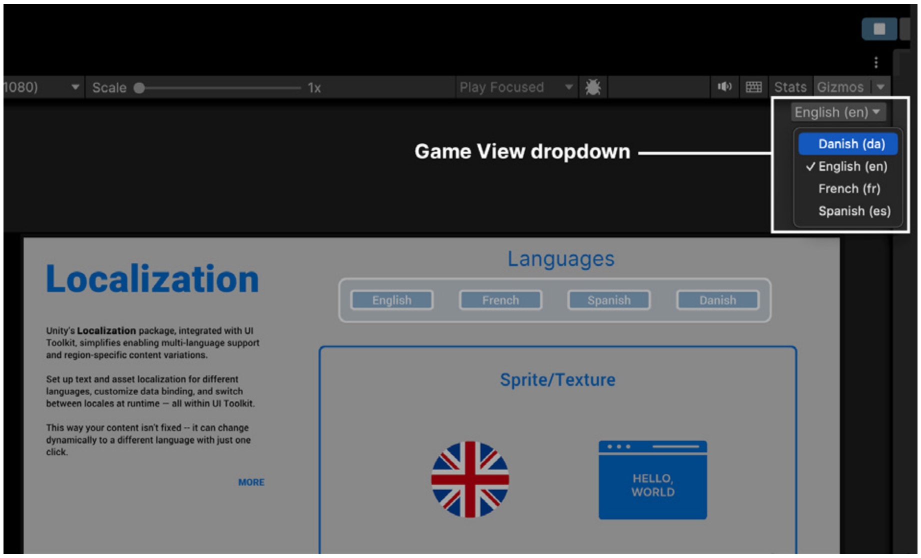
Filter for different selectors in the Hierarchy.

You might notice that some selectors begin with the **.unity**- prefix. These are default styles that apply to all elements. Any defined selectors will override these values.

# Reach more markets with integrating localization early on

You can simplify the localization process by integrating the Localization package with UI Toolkit. This integration lets you provide region-specific content for your players, no matter where they might be.

Use the Game View Locale drop-down to preview the UI in different languages, ensuring elements display correctly in each Locale.



Unity's **Localization** package, integrated with UI Toolkit, simplifies enabling multi-language support and region-specific content variations.

Set up text and asset localization for different languages, customize data binding, and switch between locales at runtime — all within UI Toolkit.

This way your content isn't fixed -- it can change dynamically to a different language with just one click.

Use the Game View Locale drop-down to preview the localization.

## Add stylization throughout the interface with Gradients

In UI Toolkit you can apply Gradients via the `<gradient>` tag. Make sure **Rich Text** is enabled and see the changes take effect inside UI Builder or in the Game view.

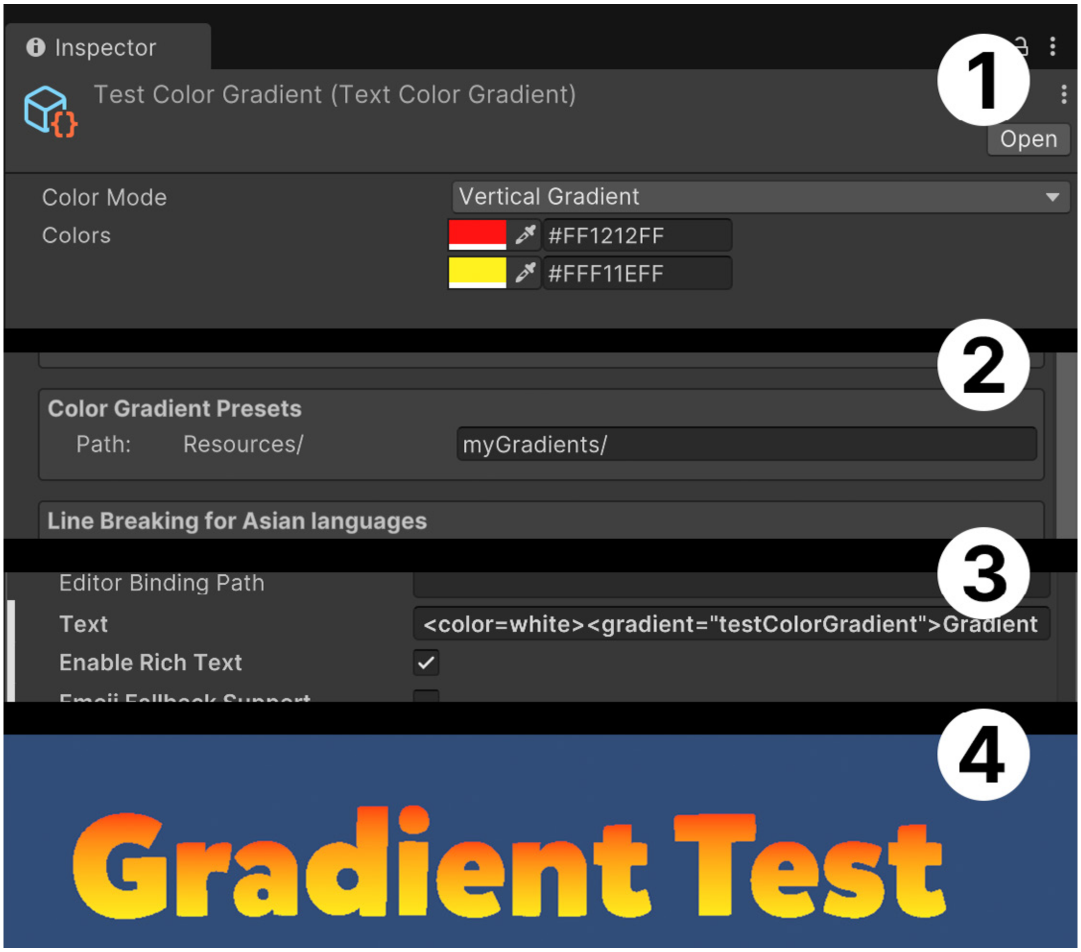
1. Create a gradient color asset via **Create > Text Core > Gradient Color**. Make sure to place this file inside **Assets/Resources** or any subfolder under Resources.

2. Create a Text Settings asset to refer to from the Panel Settings. In the asset look for Color Gradient Presets, and indicate the folder or subfolder inside Resources where the asset is.

3. Add the following rich text tags inside UI Builder: `<color=white><gradient=”testColorGradient”>Gradient Test</gradient></color>`.

   The color tag restores the font color to white so the gradient looks as intended, while the referred gradient has to match the asset name created in step 1. Make sure **Rich Text** is enabled.

4. You can see the changes take effect inside UI Builder or in the Game view.



Test Color Gradient (Text Color Gradient)

Color Mode: Vertical Gradient

Colors: 🟥 #FF1212FF<br>🟨 #FFF11EFF

Color Gradient Presets

Path: Resources/ myGradients/

Line Breaking for Asian languages

Editor Binding Path

Text: <color=white><gradient="testColorGradient">Gradient

Enable Rich Text: ✓

Emoji Fallback Support

Gradient Test

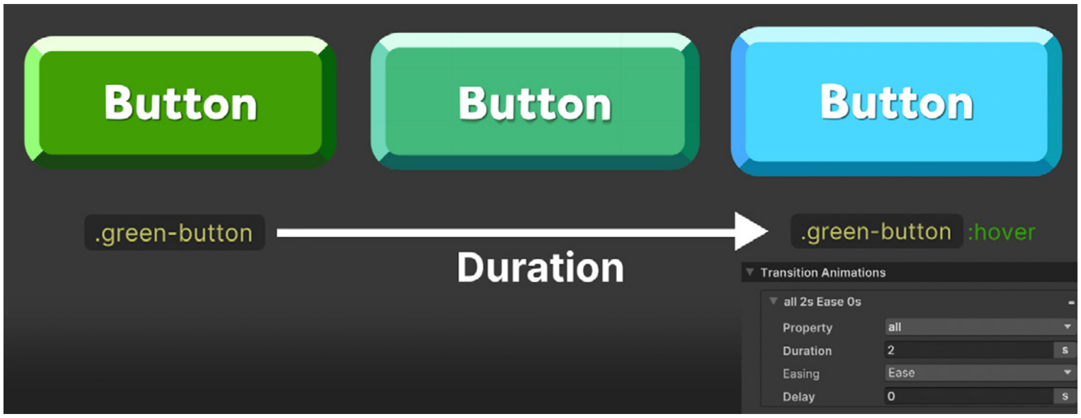
Using the <gradient> tag in UI Toolkit

## Animate UI with USS transitions

For visual elements, animations don't require additional code because pseudo-classes (`:active`, `:inactive`, `:hover`, etc.) can have their own selectors. Whenever a pseudo-class triggers a style change, any defined transitions will automatically animate the change.

For example: A button can grow or shrink when hovered over (`:hover`), clicked (`:active`), or elements can fade out or become invisible based on user interaction or other events.

Those changes of states are triggered by user actions but you can arbitrarily change the pseudo-class :enabled and :disabled which will manually trigger the USS animations relative to those pseudoclasses. This gives you an option to trigger animations at your will from code.



Transition Animations

all 2s Ease 0s

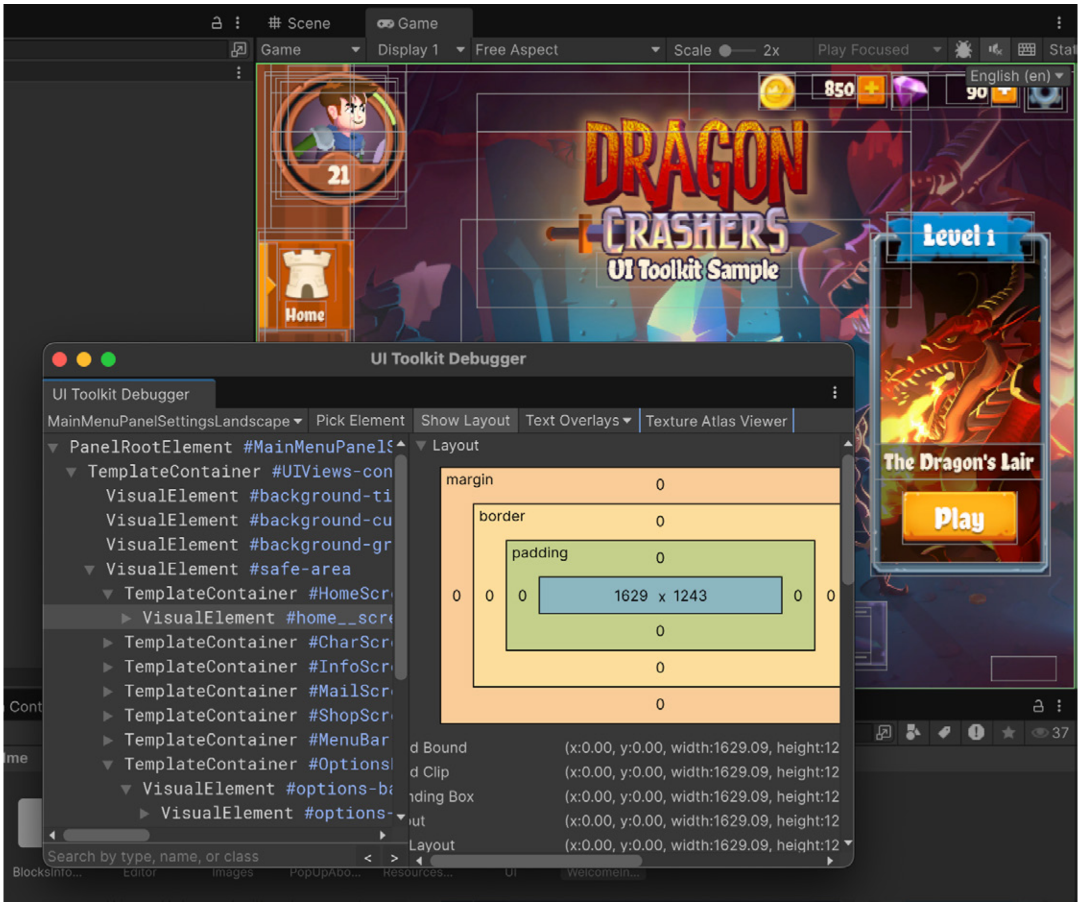
Property: all
Duration: 2 s
Easing: Ease
Delay: 0 s

An example of a USS transition animation

## Visualize resolved styles bounding boxes in the Editor

Use bounding boxes to identify layout issues, alignment problems, and interactively debug your UI structure within the Editor.

To visualize resolved bounding boxes using UI Toolkit, go to **Window** > **UI Toolkit → Debugger** to open the **UI Toolkit Debugger.** Then use the **Pick Element** tool to select your UI element and enable the **Show Layout** feature. This option displays bounding boxes and layout information directly on top of your UI in the Game view.



```
▼ PanelRootElement #MainMenuPanelS
  ▼ TemplateContainer #UIViews-con
      VisualElement #background-ti
      VisualElement #background-cu
      VisualElement #background-gr
    ▼ VisualElement #safe-area
      ▼ TemplateContainer #HomeScr
        ▶ VisualElement #home__scre
      ▶ TemplateContainer #CharScr
      ▶ TemplateContainer #InfoScr
      ▶ TemplateContainer #MailScr
      ▶ TemplateContainer #ShopScr
      ▶ TemplateContainer #MenuBar
      ▼ TemplateContainer #Options
        ▼ VisualElement #options-ba
          ▶ VisualElement #options-
```

margin 0 0 0 0; border 0 0 0 0; padding 0 0 0 0; 1629 x 1243

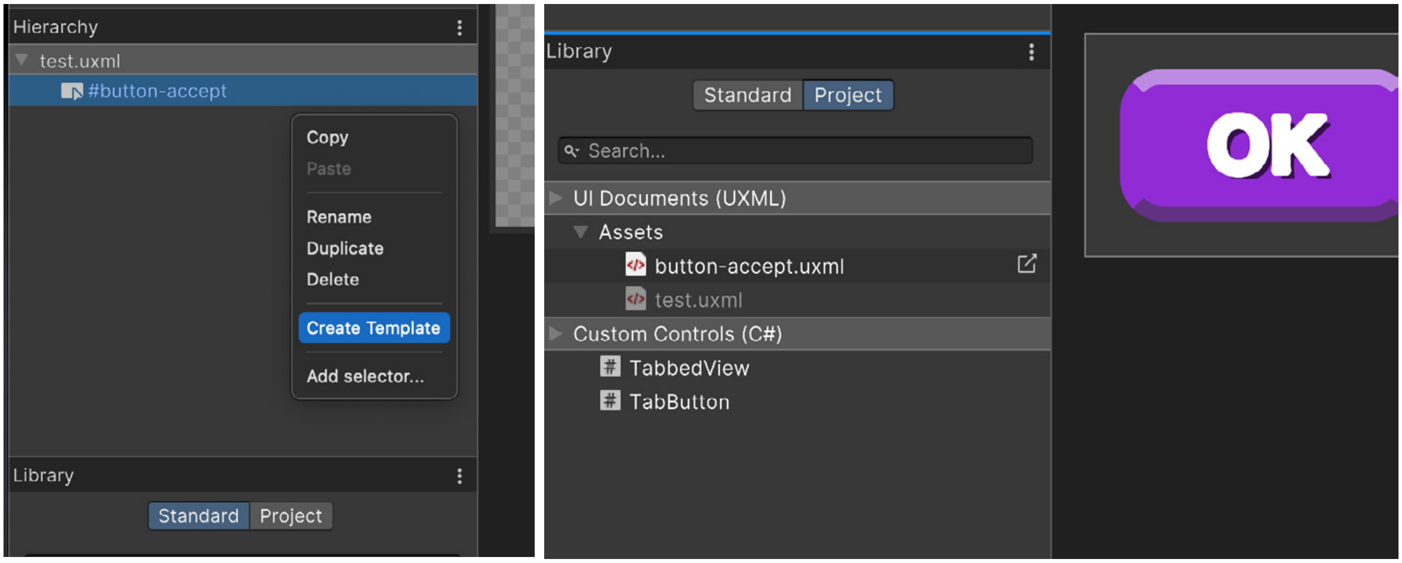
```
d Bound       (x:0.00, y:0.00, width:1629.09, height:12
d Clip        (x:0.00, y:0.00, width:1629.09, height:12
nding Box     (x:0.00, y:0.00, width:1629.09, height:12
out           (x:0.00, y:0.00, width:1629.09, height:12
Layout        (x:0.00, y:0.00, width:1629.09, height:12
```

The UI Toolkit Debugger includes several handy features to help debug your UI.

## Reuse UXML files as templates to speed up your workflow

UXML files can be used similar to prefabs. For example, you could have a project with a UXML layout that contains an item icon and count number that you need to spawn many times inside an inventory.

If you right-click on any UXML you get the option to create a Template, which can later be added to any other visual element in the Hierarchy pane or instantiated from code. Once created you can find it in your Library and Project view.



Templates are reusable UXML and are available in the Library pane in the Project tab

## More resources

Download the e-book *Create scalable and performant UI with UI Toolkit in Unity 6* to get in-depth instructions on how to create UI with UI Toolkit across a wide range of devices.

A sample project accompanies the e-book. UI Toolkit Sample – *Dragon Crashers* is available in the Unity Asset Store. The UI Toolkit sample demonstrates how you can leverage UI Toolkit for your own applications. This demo involves a full-featured interface over a slice of the 2D project *Dragon Crashers*, a mini-RPG, using the Unity 6 UI Toolkit workflow at runtime.

Additionally, make sure to check out the QuizU project on the Unity Asset Store and the supporting articles:

— The UI Toolkit sample project *QuizU*

— QuizU: State patterns for game flow

— QuizU: Managing menu screens in UI Toolkit

— QuizU: The Model View Presenter pattern

— QuizU: Event handling in UI Toolkit

— QuizU: UI Toolkit performance tips

# Developer workflows

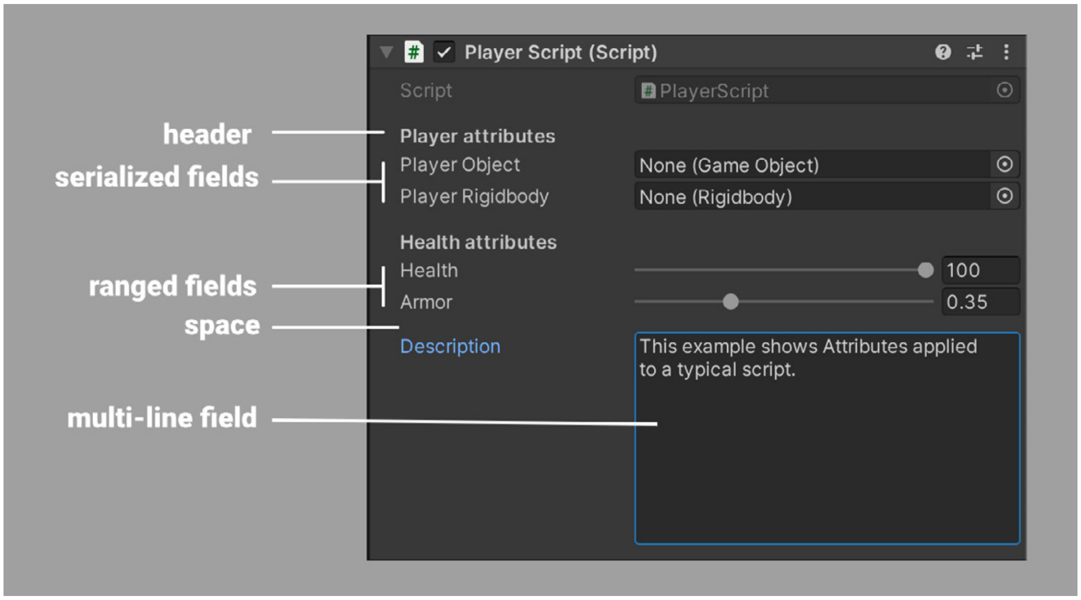
## Awaitable class

Unity 6 introduces the Awaitable class, a lightweight, allocation-free type designed specifically to support C# async/await workflows within Unity. It provides a performance-friendly alternative to coroutines. It makes it easier for you to write asynchronous code that integrates cleanly with Unity's frame-based update cycle.

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

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

        Debug.Log(“Message 2”);
        await Task.Delay(1000); // Wait another second
    }
}
```

# Enhance your Inspector window with attributes

Unity has a variety of attributes that can be placed above a class, property, or function to indicate special behavior such as creating headers, spacing, or ranged fields in the Inspector.



Player Script (Script)

Script: PlayerScript

Player attributes

Player Object: None (Game Object)

Player Rigidbody: None (Rigidbody)

Health attributes

Health: 100

Armor: 0.35

Description: This example shows Attributes applied to a typical script.

Attributes affecting the Inspector fields

C# contains attribute names within square brackets. These are some common attributes you can add to your scripts.

| Attribute | Description | Example |
|---|---|---|
| SerializeField | This forces Unity to serialize a private field and makes it visible in the Inspector. | `[SerializeField]`<br>`private GameObject m_myObject;` |
| Range | This attribute takes a float or int variable restricted to a specific range. The field appears as a slider in the Inspector. | `[Range(1,6)]`<br>`public int IntegerRange;`<br><br>`[Range(0.2f, 0.8f)]`<br>`public float m_floatRange;` |
| HideInInspector | This hides a variable in the Inspector while serializing it. | `[HideInInspector]`<br>`public Int p = 5;` |

| | | |
|---|---|---|
| RequireComponent | This automatically adds required components as dependencies to avoid setup errors.<br><br>Note: This attribute only checks the moment that the component is added to a GameObject. | // PlayerScript requires the GameObject to have a Rigidbody<br>[RequireComponent(typeof(Rigidbody))]<br>public class PlayerScript: Monobehaviour<br>{<br>    private Rigidbody m_rBody;<br><br>    void Start()<br>    {<br>      m_rBody = GetComponent<Rigidbody>();<br>    }<br>} |
| Tooltip | This shows a tooltip when the user hovers a mouse over a field in the Inspector. | public class PlayerScript: Monobehaviour<br>{<br>    [Tooltip(“Health value between 0 and 100.”)]<br>    int m_health = 0;<br> } |
| Space | This adds a small space between your fields (without any additional text) to create visual separation between your fields. | [Space(10)] // 10 pixel of spacing added<br>int p = 5; |
| Header | This adds some bold text and spacing to help organize your variables in the Inspector. Only add this to the first field that you want to belong to the group. | public class PlayerScript: Monobehaviour<br>{<br>    [Header(“Health Settings”)]<br>    private int m_health = 0;<br>    private int m_maxHealth = 100;<br><br>    [Header(“Shield Settings”)]<br>    private int m_shield = 0;<br>    private int m_maxShield = 0;<br>} |

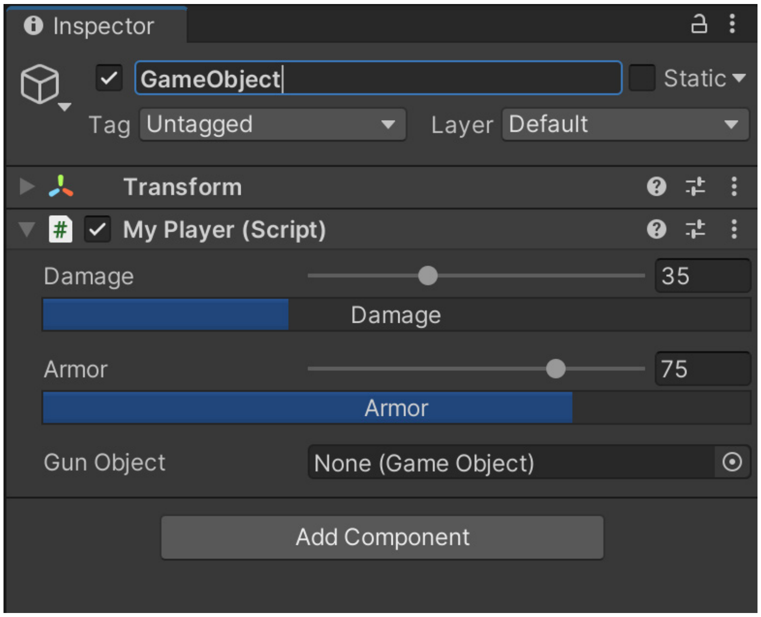
| | | |
|---|---|---|
| Multiline | This makes the string editable with the multiline text field. Pass in an optional int to designate the number of lines.<br>Tip: Use this for annotating scripts with notes to yourself or another user. | `[Multiline]`<br>`public string textToEdit;`<br><br>`[Multiline(20)]`<br>`public string m_moreTextToEdit;` |
| SelectionBase | This is useful for selecting an otherwise empty GameObject whose children may contain meshes. Add the attribute to any component on the base object. When picking objects in the Editor, the GameObject containing the [SelectionBase] attribute gets selected rather than the children. | `// add this to the base GameObject`<br>`[SelectionBase]`<br>`public class PlayerScript: Monobehaviour`<br>`{`<br><br>`}` |
| ColorUsage | The [ColorUsage] attribute lets you control what colors can be selected in a color field. You can enable HDR and/or disable the alpha channel, depending on the parameters. | public **ColorUsageAttribute**(bool **showAlpha**, bool **hdr**, float **minBrightness**, float **maxBrightness**, float **minExposureValue**, float **maxExposureValue**); |

| | | |
|---|---|---|
| RunOnce | Need to automatically run a function only once when your project starts? Using the static standard with the [RuntimeInitialization] attribute is an easy way to do it. Use it Performing one-time project setup logic like a boatloader as demonstrated in the QuizU sample. | `public RuntimeInitializeOnLoadMethodAttribute(RuntimeInitializeLoadType loadType);` |

This is just a small sample of the numerous attributes available. Do you want to rename your variables without losing their values? Or invoke some logic without needing an empty GameObject? You can even create your own PropertyAttribute to define custom attributes for your script variables. See the Scripting API for a complete list of attributes.

## Create your own custom windows and Inspectors

One of Unity's most powerful features is its extensible Editor. We recommend that you use the **UI Toolkit** package to create Editor UIs such as custom windows and custom Inspectors.



☑ GameObject □ Static
Tag Untagged
Layer Default
Damage 35
Damage
Armor 75
Armor
Gun Object None (Game Object)

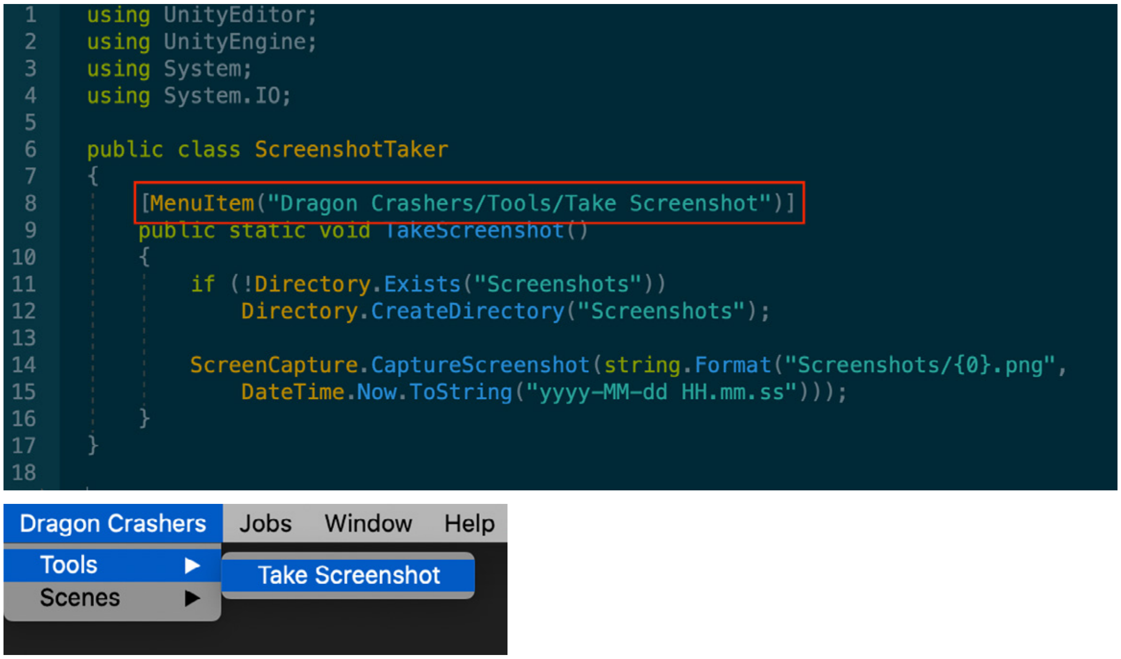
A custom Editor modifies how the MyPlayer script displays in the Inspector.

See Creating user interfaces (UI) for more detail on how to implement custom Editor scripts using either UI Toolkit or IMGUI. For a quick introduction to **UI Toolkit**, watch the Getting Started with Editor Scripting tutorial.

## Create custom menus

Unity includes a simple way to customize Editor menus and menu items, the **MenuItem** attribute. You can apply this to any static method in your scripts.

If you have functions for your project that you will use frequently, organize them into menu items. This allows you to build a basic user interface with just a single PropertyAttribute modifier.



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

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

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

The MenuItem attribute creates a simple interface to attach the static method (Take Screenshot).

## Speed up the Enter Play time

When you enter Play mode, your project starts and runs as it would in a build. Any changes you make in the Editor during Play mode reset when you exit Play mode.

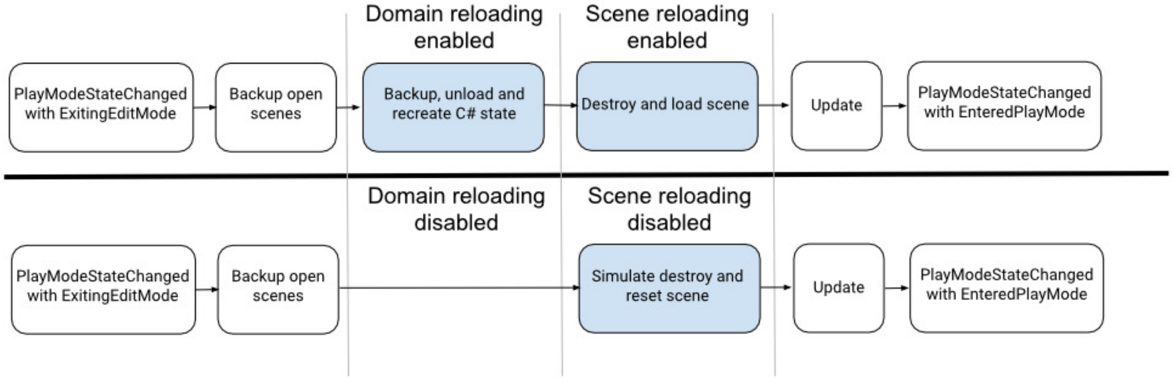
Unity performs two significant actions every time you enter Play mode:

- **Domain Reload:** Unity backs up, unloads, and recreates scripting states.
- **Scene Reload:** Unity destroys the Scene and loads it again.

These two actions take more and more time as your scripts and scenes become more complex.

If you don't plan on making any more script changes, the **Enter Play Mode Settings (Edit > Project Settings > Editor)** can save you a bit of compile time. Unity gives you the option to disable either Domain Reload, Scene Reload, or both. This can speed up entering and exiting Play mode.

Just remember that if you do plan on making further script changes, you need to reenable Domain Reload. Likewise, if you modify the Scene Hierarchy, you should reenable Scene Reload. Otherwise, unexpected behavior could result.


Domain reloading enabled
Scene reloading enabled
PlayModeStateChanged with ExitingEditMode
Backup open scenes
Backup, unload and recreate C# state
Destroy and load scene
Update
PlayModeStateChanged with EnteredPlayMode
Domain reloading disabled
Scene reloading disabled
PlayModeStateChanged with ExitingEditMode
Backup open scenes
Simulate destroy and reset scene
Update
PlayModeStateChanged with EnteredPlayMode

The effects of disabling the Reload Domain and Reload Scene settings.

## Customize the default Script templates

Do you find that you make the same changes every time you create a new script? Do you instinctively add a namespace or delete the update event function? Save yourself a few keystrokes and create consistency across the team by setting up the script template for your preferred starting point.

Every time you create a new script or shader, Unity uses a template stored in **%EDITOR_PATH%\Data\Resources\ScriptTemplates**:

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

There are also templates for shaders, other behavior scripts, and assembly definitions.

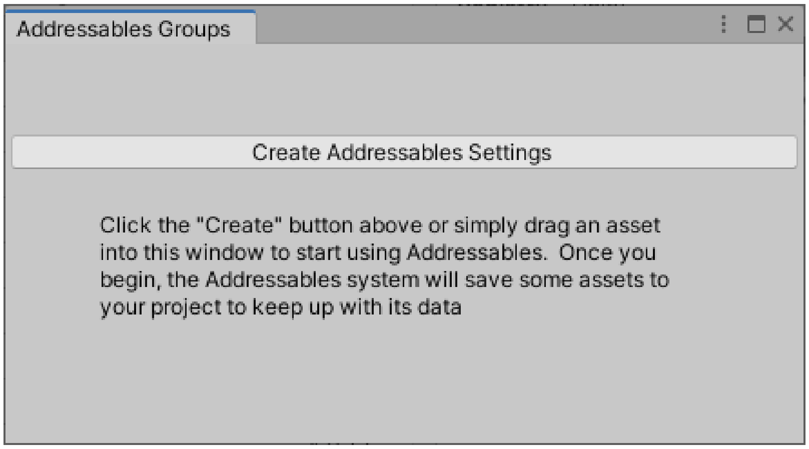
For project-specific script templates, create an Assets/ScriptTemplates folder. Copy the script templates into this folder to override the defaults.

You can also modify the default script templates directly for all projects, but make sure that you back up the originals before making any changes.

## Distribute content to your players on demand with Addressables

Addressables and Asset Bundles are powerful tools to structure your game in logical blocks that can then be exported separately and added to the main executable whenever needed.

They are used to load and unload assets, to configure, build, and load asset bundles that you can then distribute to your players on demand. The Addressables system is built on top of Asset Bundles, taking care of dependencies resolution and bundle loading for you.



Click the "Create" button above or simply drag an asset into this window to start using Addressables.  Once you begin, the Addressables system will save some assets to your project to keep up with its data

Before initializing the Addressables system in a Unity project

If you're new to Addressables, make sure you check out the Get started page in Unity Documentation.

Tips for effective asset management:

- Leverage Addressables from the start and ensure that every new asset is registered as an Addressable.
- Aim to group assets by how often they are loaded and used together, instead of organizing them by type. This will improve runtime memory usage, reduce boot time, and as a result improve game retention as well.
- Aim for small bundles because it leads to shorter dependency chains and lower runtime memory usage.

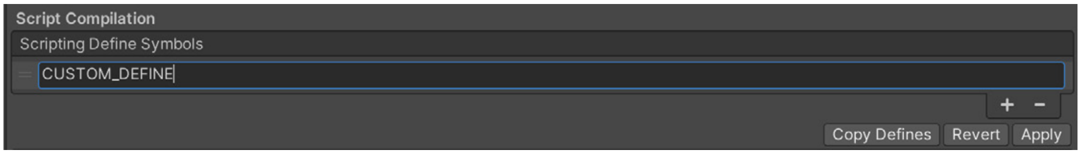
Read more in the *Effective asset management in Unity with Addressables* article.

## Create conditionally compiled code with Preprocessor directives

The platform-dependent compilation feature allows you to conditionally compile and execute code based on the target platform, Unity version, or scripting backend.

This can be useful when you write cross-platform code, optimize for device-specific behavior, or manage version-specific APIs.

You can supply your own custom #define directives when testing in the Editor. Open the **Other Settings** panel of the Player settings, and navigate to **Scripting Define Symbols**.



Script Compilation

Scripting Define Symbols

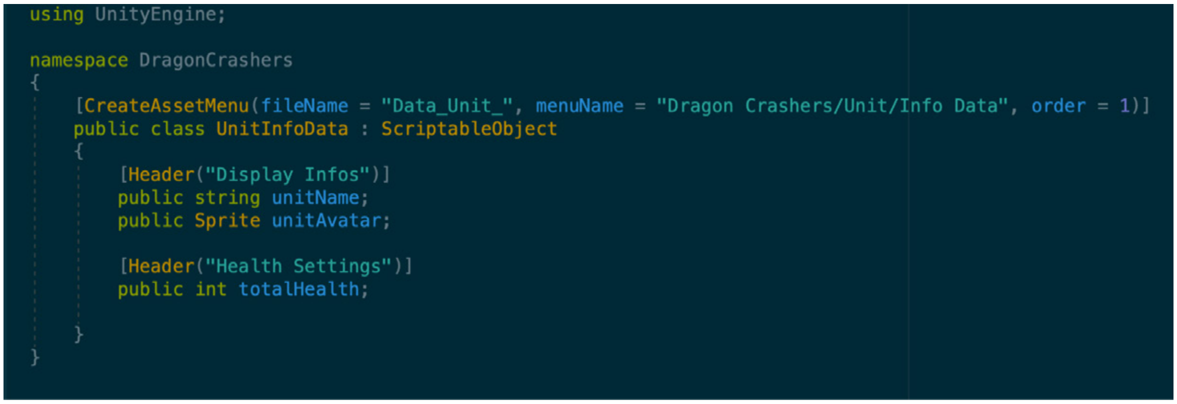
CUSTOM_DEFINE

Scripting Define Symbols in Script Compilation

## Use ScriptableObjects to separate data from logic

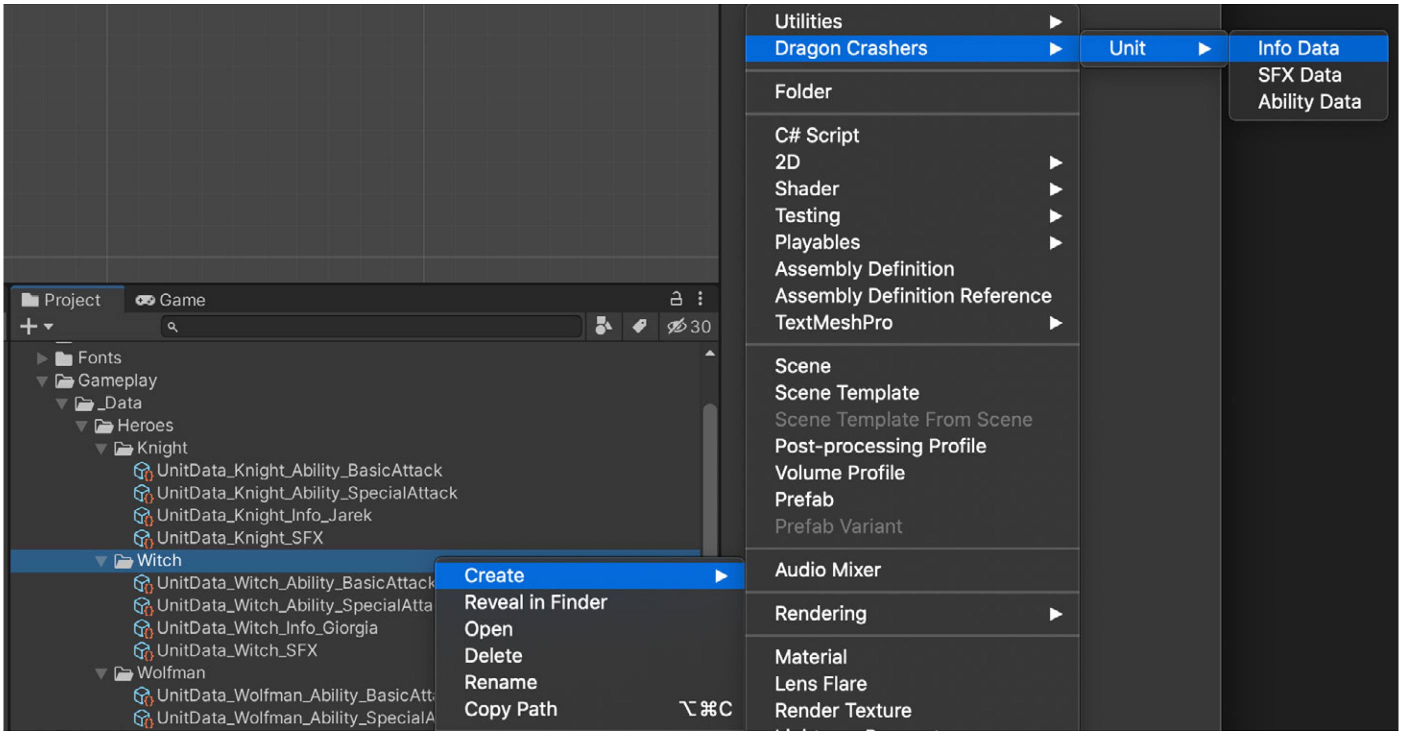
ScriptableObjects can help you promote clean coding practices by separating data from logic. This means it's easier to make changes without causing unintended side effects, which improves testability and modularity. They're also useful when you're collaborating with non-programmers like artists and designers; they can edit game data without touching code.

*Dragon Crashers* demonstrates a typical use case. A **UnitInfoData** class inherits from **ScriptableObject**. Each of its instances contains the unit's name, sprite, and health settings. This data remains constant over the course of gameplay, making it especially suitable for storage inside a ScriptableObject.



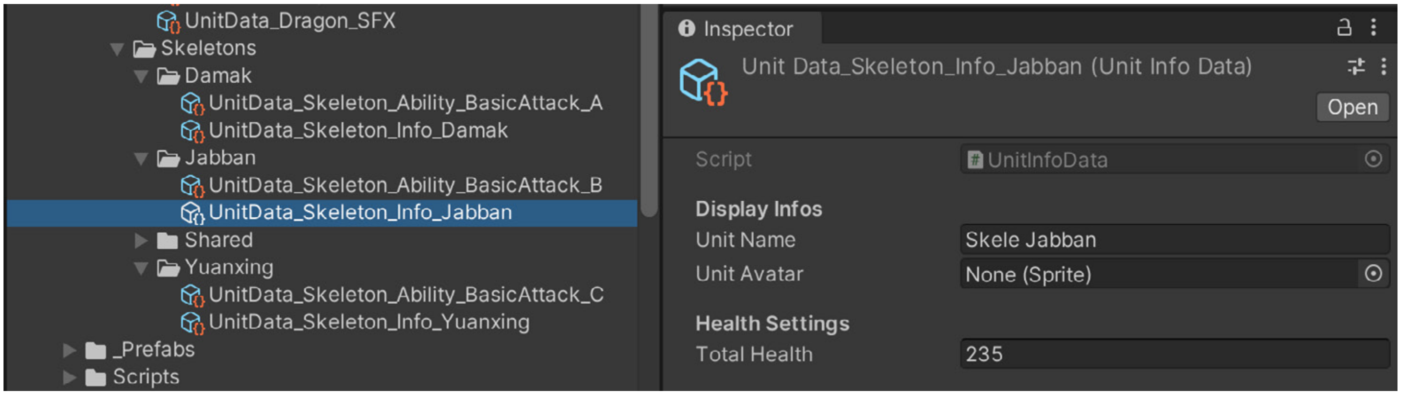
```
using UnityEngine;

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

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

    }
}
```

A ScriptableObject defines a data container object.



The CreateAssetMenu attribute generates a context menu item to help you generate a ScriptableObject asset. Each unit has additional ScriptableObjects for sound effects and special abilities.

With the assets created in the project window, you can fill in the correct values using the Inspector: Unit Name, Unity Avatar (Sprite), and Total Health.


UnitData_Dragon_SFX
Skeletons
Damak
UnitData_Skeleton_Ability_BasicAttack_A
UnitData_Skeleton_Info_Damak
Jabban
UnitData_Skeleton_Ability_BasicAttack_B
UnitData_Skeleton_Info_Jabban
Shared
Yuanxing
UnitData_Skeleton_Ability_BasicAttack_C
UnitData_Skeleton_Info_Yuanxing
_Prefabs
Scripts
Inspector
Open

Unit Data_Skeleton_Info_Jabban (Unit Info Data)

Script UnitInfoData

Display Infos

Unit Name Skele Jabban

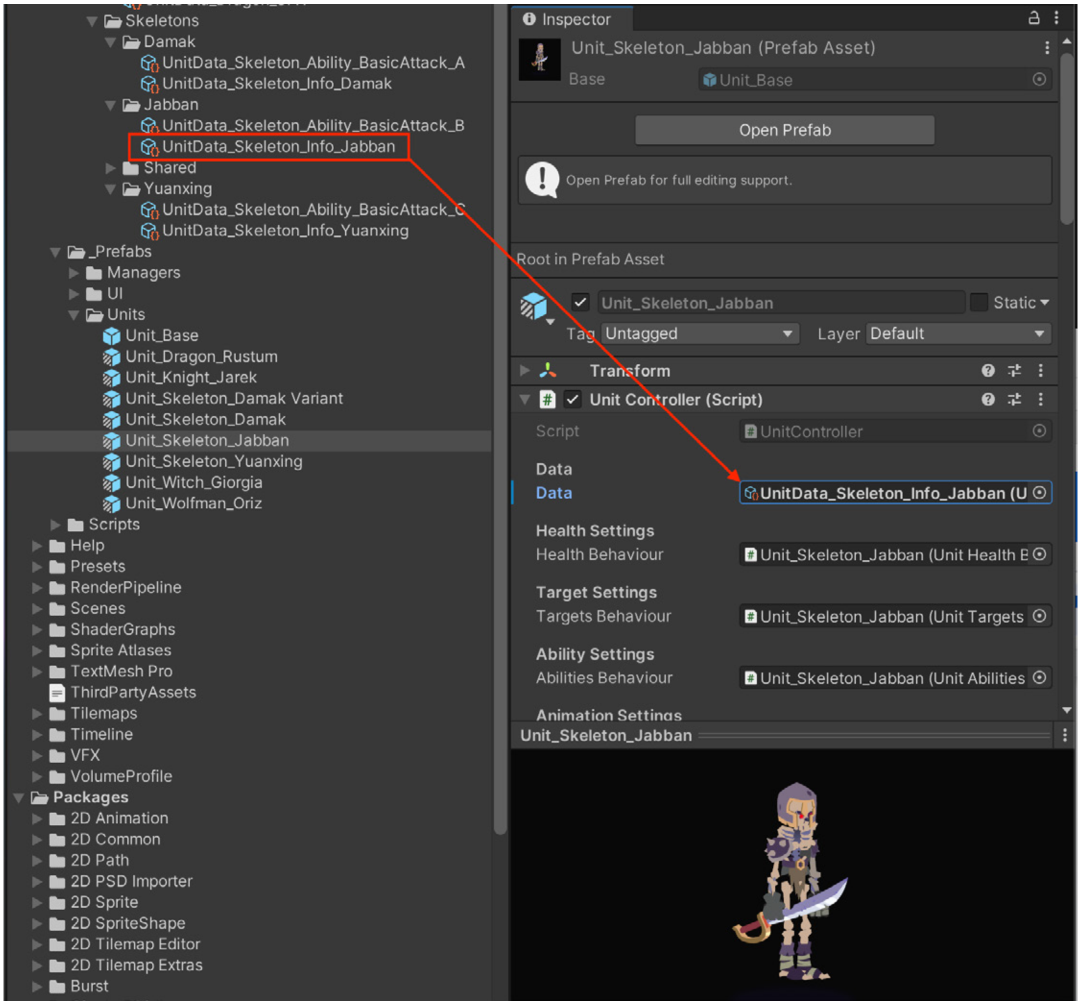
Unit Avatar None (Sprite)

Health Settings

Total Health 235

Use the Inspector to fill out values for the ScriptableObject asset. These values won't change during gameplay.

A GameObject (like the UnitController in this case) can then reference the ScriptableObject asset. If the scene suddenly fills with units, the data on the ScriptableObject asset does not duplicate, saving memory.



```
    ▼ Skeletons
      ▼ Damak
          UnitData_Skeleton_Ability_BasicAttack_A
          UnitData_Skeleton_Info_Damak
      ▼ Jabban
          UnitData_Skeleton_Ability_BasicAttack_B
          UnitData_Skeleton_Info_Jabban
      ▶ Shared
      ▼ Yuanxing
          UnitData_Skeleton_Ability_BasicAttack_C
          UnitData_Skeleton_Info_Yuanxing
  ▼ _Prefabs
    ▶ Managers
    ▶ UI
    ▼ Units
        Unit_Base
        Unit_Dragon_Rustum
        Unit_Knight_Jarek
        Unit_Skeleton_Damak Variant
        Unit_Skeleton_Damak
        Unit_Skeleton_Jabban
        Unit_Skeleton_Yuanxing
        Unit_Witch_Giorgia
        Unit_Wolfman_Oriz
  ▶ Scripts
▶ Help
▶ Presets
▶ RenderPipeline
▶ Scenes
▶ ShaderGraphs
▶ Sprite Atlases
▶ TextMesh Pro
  ThirdPartyAssets
▶ Tilemaps
▶ Timeline
▶ VFX
▶ VolumeProfile
▼ Packages
  ▶ 2D Animation
  ▶ 2D Common
  ▶ 2D Path
  ▶ 2D PSD Importer
  ▶ 2D Sprite
  ▶ 2D SpriteShape
  ▶ 2D Tilemap Editor
  ▶ 2D Tilemap Extras
  ▶ Burst
```

Inspector

Unit_Skeleton_Jabban (Prefab Asset)

Base: Unit_Base

Open Prefab

Open Prefab for full editing support.

Root in Prefab Asset

☑ Unit_Skeleton_Jabban □ Static

Tag: Untagged | Layer: Default

▶ Transform

▼ ☑ Unit Controller (Script)

Script: UnitController

Data

Data: UnitData_Skeleton_Info_Jabban (U

Health Settings

Health Behaviour: Unit_Skeleton_Jabban (Unit Health E

Target Settings

Targets Behaviour: Unit_Skeleton_Jabban (Unit Targets

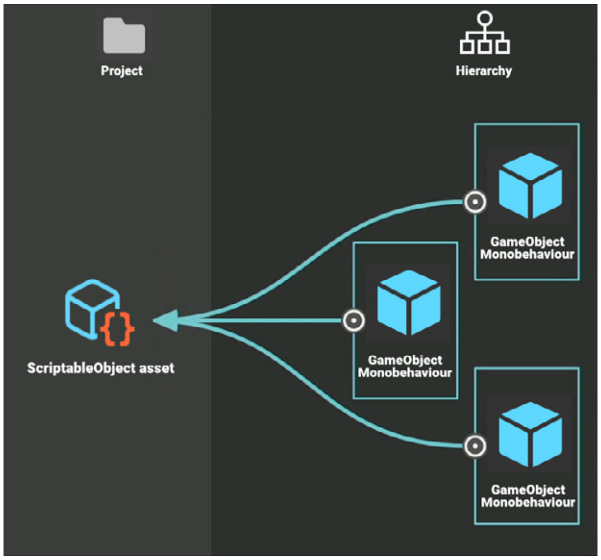
Ability Settings

Abilities Behaviour: Unit_Skeleton_Jabban (Unit Abilities

Animation Settings

Unit_Skeleton_Jabban

The Monobehaviour object (UnitController, shown above) refers to the ScriptableObject data asset in the project.



Save memory and stay organized with ScriptableObjects. Set static data and settings in the asset in the project just once, even if you have lots of GameObjects.

Even if you add a thousand instances of a prefab to your scene, they still refer to the same data stored in your asset. Setting up the set of values just once guarantees consistency.

As your game scales up with more unit types, simply create more ScriptableObject assets and swap them out appropriately. Maintain your gameplay data just by tweaking the centrally stored assets.

ScriptableObjects don't replace keeping persistent data for the rest of your application's save files, where the data may change during gameplay. It's a workflow suited more for storing your static gameplay settings and default values. Unlike parsing data from JSON or XML, reading a ScriptableObject asset won't generate garbage (and, as a bonus, it's faster).

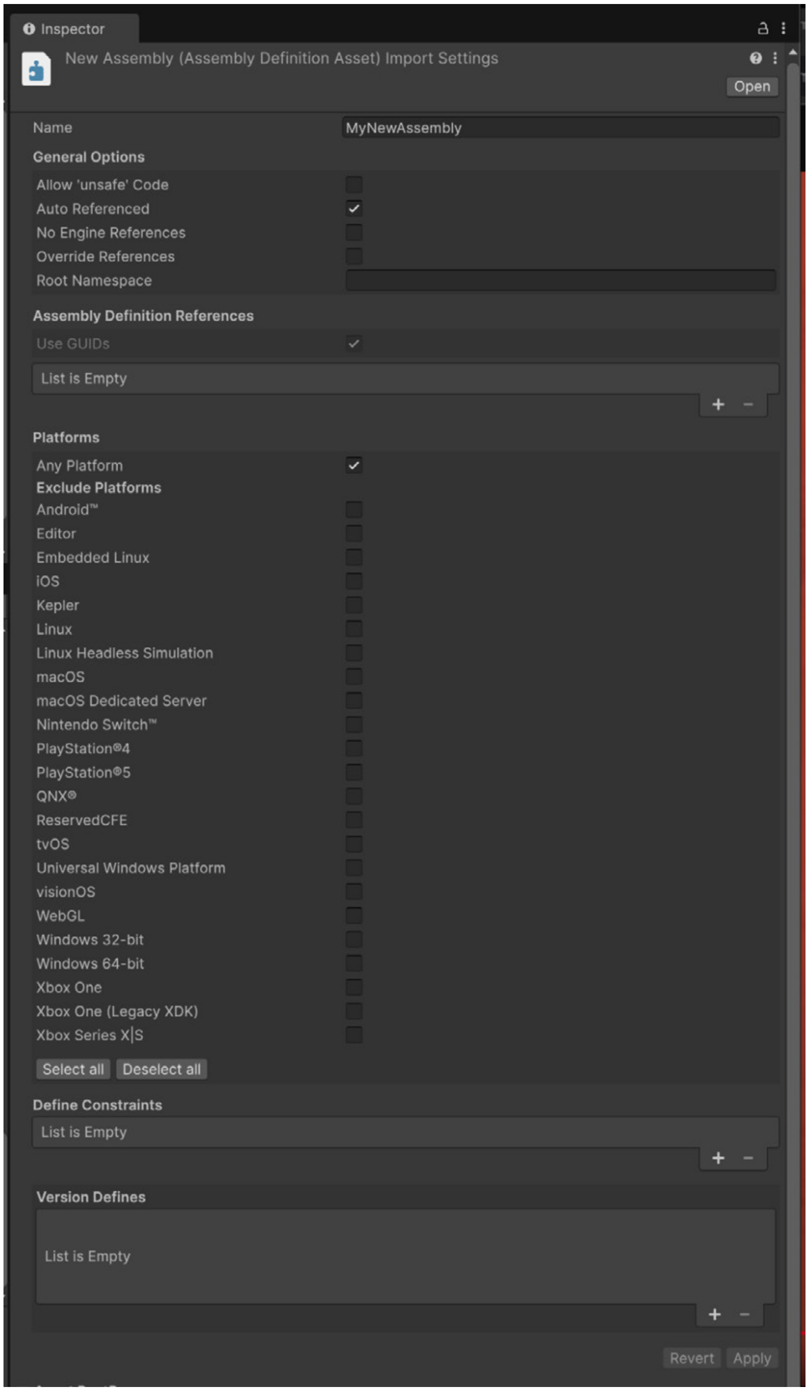
More resources on ScriptableObjects:

- Create modular game architecture with ScriptableObjects in Unity
- ScriptableObjects Paddle Ball demo project
- ScriptableObject documentation

## Promote script modularity with Assembly Definitions

An assembly is a compiled C# code library that groups related types and resources into a single, logical unit. In Unity, you can manage your assemblies using Assembly Definition Files (.asmdef). Organizing your scripts into custom assemblies promotes modularity and reusability while also decreasing compilation time. It prevents them from getting added to the default assemblies automatically and limits which other scripts they can access.

If you're cleaning up your projects with Assembly Definitions and your Editor scripts are put into your builds, then create an Assembly Definition in your Editor folder and set it to include only the Editor Platform.



New Assembly (Assembly Definition Asset) Import Settings

Name: MyNewAssembly

**General Options**

- Allow 'unsafe' Code ☐
- Auto Referenced ☑
- No Engine References ☐
- Override References ☐
- Root Namespace

**Assembly Definition References**

- Use GUIDs ☑

List is Empty

**Platforms**

- Any Platform ☑

**Exclude Platforms**

- Android™ ☐
- Editor ☐
- Embedded Linux ☐
- iOS ☐
- Kepler ☐
- Linux ☐
- Linux Headless Simulation ☐
- macOS ☐
- macOS Dedicated Server ☐
- Nintendo Switch™ ☐
- PlayStation®4 ☐
- PlayStation®5 ☐
- QNX® ☐
- ReservedCFE ☐
- tvOS ☐
- Universal Windows Platform ☐
- visionOS ☐
- WebGL ☐
- Windows 32-bit ☐
- Windows 64-bit ☐
- Xbox One ☐
- Xbox One (Legacy XDK) ☐
- Xbox Series X|S ☐

**Define Constraints**

List is Empty

**Version Defines**

List is Empty

Assembly Definitions settings in the Inspector

# Upgrade to the Input System

If you haven't upgraded already, make sure to check out the Input System package which is a newer, more flexible system than the Input Manager, which allows you to use any kind of Input Device to control your Unity content. It's referred to as "The Input System Package", or just "The Input System". It also supports rebindable controls, input action assets, and cleaner separation between input and gameplay logic giving you significant advantages over the legacy system.

To get started check out the following resources:

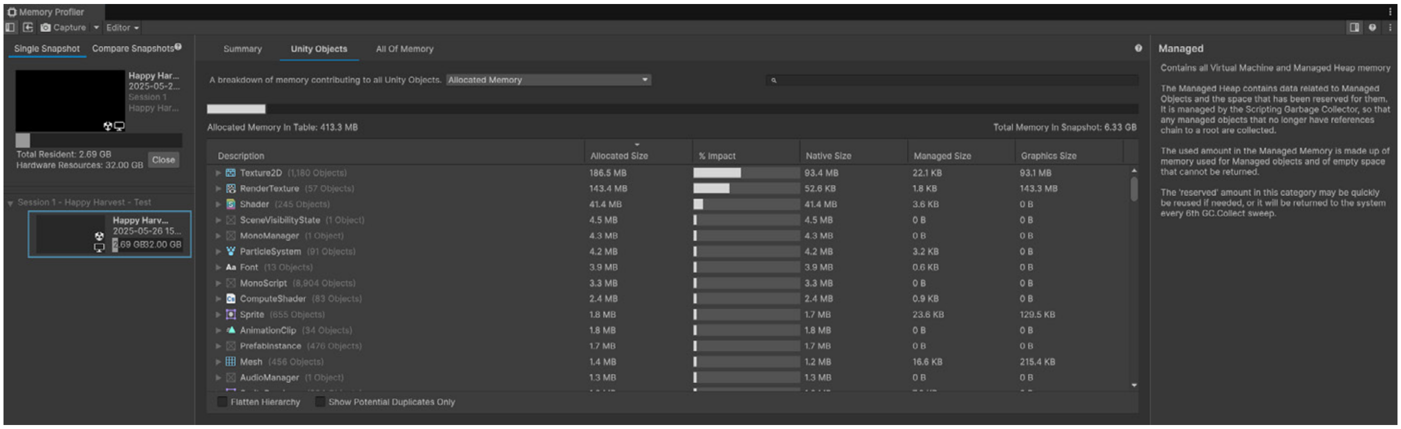
- Prototype mobile games faster with the Input System in Unity 6 | Unite 2024
- Get up and running with the Input System
- Unity Input System 7-video tutorial series

# Profiling tools

## Optimize your memory performance with Memory Profiler

The Memory Profiler lets you capture and analyze memory usage in your project to identify leaks, reduce memory spikes, and optimize runtime performance. Use it to take memory snapshots during key moments (e.g. scene loads, after long play sessions) and compare them to track down objects that aren't being released properly.

When you create resources in code make it a habit to name them in your Memory Profiler. Also, remember to release anything you allocated to avoid leaks.



Total Resident: 2.69 GB
Hardware Resources: 32.00 GB

A breakdown of memory contributing to all Unity Objects.

Allocated Memory In Table: 413.3 MB

Total Memory In Snapshot: 6.33 GB

| Description | Allocated Size | % Impact | Native Size | Managed Size | Graphics Size |
|---|---|---|---|---|---|
| Texture2D (1,180 Objects) | 186.5 MB | | 93.4 MB | 22.1 KB | 93.1 MB |
| RenderTexture (57 Objects) | 143.4 MB | | 52.6 KB | 1.8 KB | 143.3 MB |
| Shader (245 Objects) | 41.4 MB | | 41.4 MB | 3.6 KB | 0 B |
| SceneVisibilityState (1 Object) | 4.5 MB | | 4.5 MB | 0 B | 0 B |
| MonoManager (1 Object) | 4.3 MB | | 4.3 MB | 0 B | 0 B |
| ParticleSystem (91 Objects) | 4.2 MB | | 4.2 MB | 3.2 KB | 0 B |
| Font (13 Objects) | 3.9 MB | | 3.9 MB | 0.6 KB | 0 B |
| MonoScript (8,904 Objects) | 3.3 MB | | 3.3 MB | 0 B | 0 B |
| ComputeShader (83 Objects) | 2.4 MB | | 2.4 MB | 0.9 KB | 0 B |
| Sprite (655 Objects) | 1.8 MB | | 1.7 MB | 23.6 KB | 129.5 KB |
| AnimationClip (34 Objects) | 1.8 MB | | 1.8 MB | 0 B | 0 B |
| PrefabInstance (476 Objects) | 1.7 MB | | 1.7 MB | 0 B | 0 B |
| Mesh (456 Objects) | 1.4 MB | | 1.2 MB | 16.6 KB | 215.4 KB |
| AudioManager (1 Object) | 1.3 MB | | 1.3 MB | 0 B | 0 B |

Managed

Contains all Virtual Machine and Managed Heap memory

The Managed Heap contains data related to Managed Objects and the space that has been reserved for them. It is managed by the Scripting Garbage Collector, so that any managed objects that no longer have references chain to a root are collected.

The used amount in the Managed Memory is made up of memory used for Managed objects and of empty space that cannot be returned.

The 'reserved' amount in this category may be quickly be reused if needed, or it will be returned to the system every 6th GC.Collect sweep.

A snapshot of the Memory Profiler

## Use the ProfilerMarker to pinpoint performance critical code

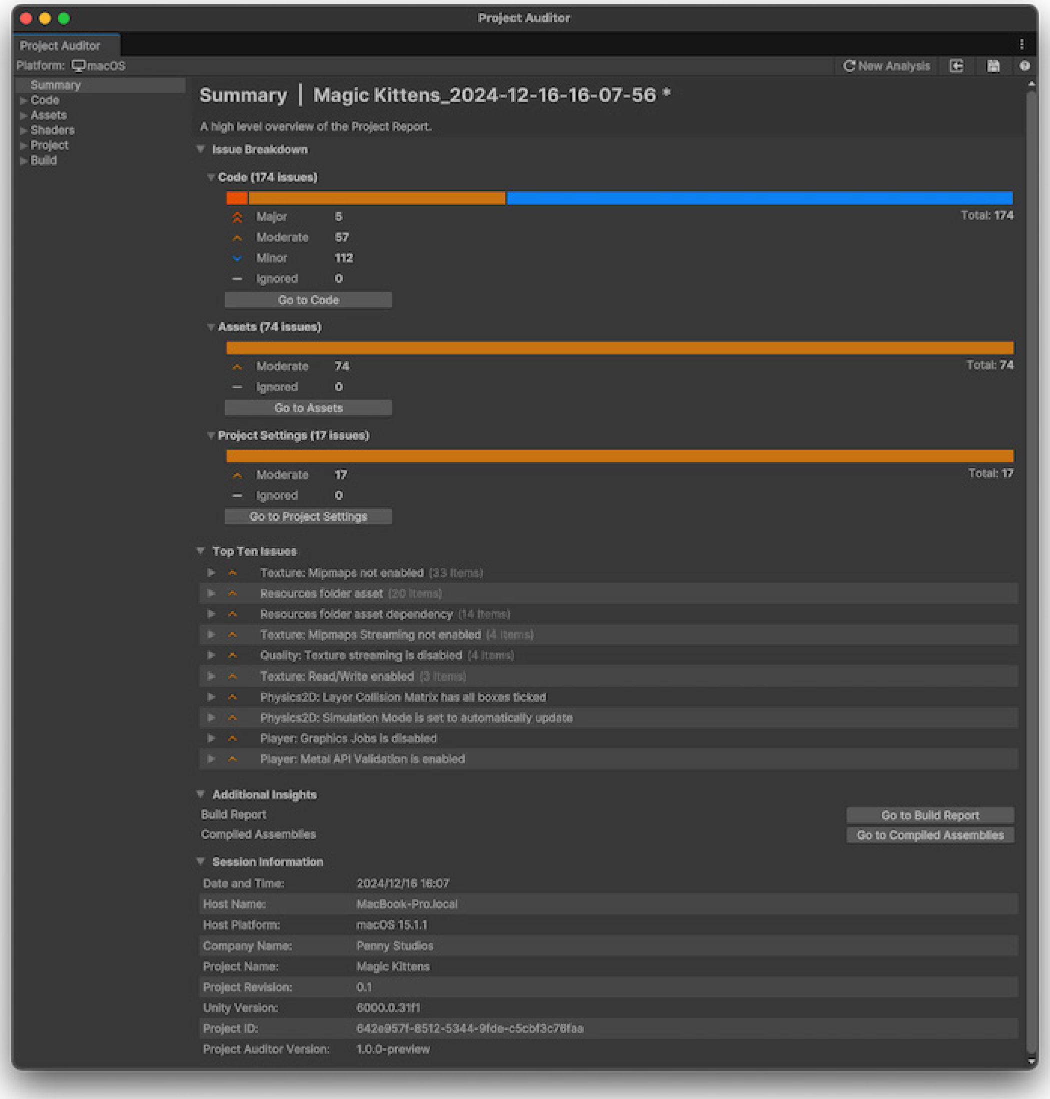
Instead of only seeing performance data aggregated under general markers like BehaviourUpdate, you can isolate and measure the exact execution time of your specific functions. Use the ProfilerMarker to mark up script code blocks as a way to increase the detail level of profiling runs. The information is then displayed in the CPU Profiler and can also be captured with the Unity Recorder. This provides you with a detailed breakdown of where time is spent in your specific code sections, making it easier to identify performance bottlenecks and optimize the code.

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

   public void UpdateLogic() {
       SetupProfileMarker.Begin();
       // Setup your performance heavy things, Initializers, and so on...
       SetupProfileMarker.End();
       using (ExpensiveProfileMarker.Auto()) { //This starts and ends automatically
       // More expensive things here.
```

## Get a performance audit on your project

Use the Project Auditor (introduced as a package in Unity 6.1) to analyze your projects performance, maintain best practices, and identify potential issues and bottlenecks.

With a few clicks you can scan your entire project and get a detailed report about inefficiencies, such as heavy scripting calls, unused assets, excessive entity counts, and more.



Summary | Magic Kittens_2024-12-16-16-07-56 *

A high level overview of the Project Report.

Issue Breakdown

Code (174 issues)

| | | |
|---|---|---|
| Major | 5 | Total: 174 |
| Moderate | 57 | |
| Minor | 112 | |
| Ignored | 0 | |

Assets (74 issues)

| | | |
|---|---|---|
| Moderate | 74 | Total: 74 |
| Ignored | 0 | |

Project Settings (17 issues)

| | | |
|---|---|---|
| Moderate | 17 | Total: 17 |
| Ignored | 0 | |

Top Ten Issues

- Texture: Mipmaps not enabled (33 Items)
- Resources folder asset (20 Items)
- Resources folder asset dependency (14 Items)
- Texture: Mipmaps Streaming not enabled (4 Items)
- Quality: Texture streaming is disabled (4 Items)
- Texture: Read/Write enabled (3 Items)
- Physics2D: Layer Collision Matrix has all boxes ticked
- Physics2D: Simulation Mode is set to automatically update
- Player: Graphics Jobs is disabled
- Player: Metal API Validation is enabled

Additional Insights

Build Report

Compiled Assemblies

Session Information

| | |
|---|---|
| Date and Time: | 2024/12/16 16:07 |
| Host Name: | MacBook-Pro.local |
| Host Platform: | macOS 15.1.1 |
| Company Name: | Penny Studios |
| Project Name: | Magic Kittens |
| Project Revision: | 0.1 |
| Unity Version: | 6000.0.31f1 |
| Project ID: | 642e957f-8512-5344-9fde-c5cbf3c76faa |
| Project Auditor Version: | 1.0.0-preview |

Project Auditor Summary view

The reports generated are categorized by severity, such as errors, warnings, and informational insights making it easy to focus on addressing errors and warnings first, such as over-allocation of memory or excessive garbage collection.

It's generally recommended to run the Project Auditor at key stages of development (e.g., before milestones, beta releases, final builds), so that you can catch performance bottlenecks, unused assets, or outdated code early, preventing problems from growing larger as your project scales.

You can customize the Project Auditor using custom rules and filters. For example, exclude certain scripts or assets from analysis that are meant to be unused and experimental, or make specific rules for your build targets, resolution, text compression, or other project settings to ensure they are optimized for your "budgets" .

# Animation curves

## Control interpolation with the custom lerp function

By default, Mathf.Lerp(a, b, t) clamps the interpolation factor `t` between 0 and 1, meaning it won’t return values outside the range between a and b. If you need values to overshoot (t > 1) or undershoot (t < 0), use Mathf.LerpUnclamped(a, b, t) instead. This gives you full control over the interpolation and allows for effects like extrapolation or momentum-based motion.

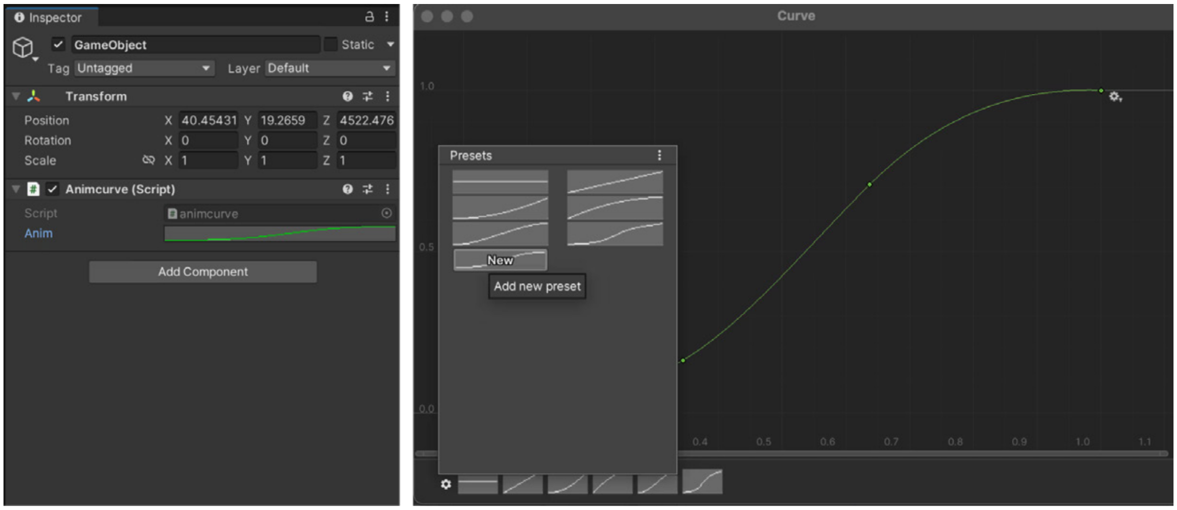
```
using UnityEngine;

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

    private void Start()
    {
        // Clamps t to [0, 1]
        float clamped = Mathf.Lerp(start, end, t);
        // Uses full t value
        float unclamped = Mathf.LerpUnclamped(start, end, t);

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

An example of using custom lerp in Unity

## Use AnimationCurve for more than just animation

AnimationCurves are typically used to animate the value of component properties in AnimationClip, but you can use them to dynamically drive any float value.

Animation Curves can be edited within the Inspector either as public variables, or when serialized. You can save, export, or load them in Edit mode or at runtime. Editable tangents make it possible to control the shape of the curve between the keys.



Inspector

☑ GameObject □ Static

Tag Untagged Layer Default

Transform

| | X | Y | Z |
|---|---|---|---|
| Position | 40.45431 | 19.2659 | 4522.476 |
| Rotation | 0 | 0 | 0 |
| Scale | 1 | 1 | 1 |

☑ Animcurve (Script)

Script animcurve

Anim

Add Component

An Animation Curve property in the Inspector: Clicking on it opens the Curve Editor, where you can adjust the curve and save it into your own library by selecting the cog icon.

Check out the blog post *Animation Curves, the ultimate design lever* for more practical tips and examples of using AnimationCurves in your project.

## Reduce processing power with object pooling

Object pooling is a design pattern that can enhance performance optimization by reducing the processing power required of the CPU to run repetitive create and destroy calls. Instead, with object pooling, existing GameObjects can be reused over and over.

How you use object pools will vary by application. A good general rule is to profile your code every time you instantiate a large number of objects, since you run the risk of causing a GC spike.

If you detect significant spikes that put your gameplay at risk of stuttering, consider using an object pool. Just remember that object pooling can add more complexity to your codebase due to the need to manage the multiple life cycles of the pools. Additionally, you may also end up reserving memory your gameplay doesn't necessarily need by creating too many premature pools.

Learn more about object pooling from the e-book *Level up your code with design patterns and SOLID* and its companion sample project that's available for free from the Unity Asset Store.

## More resources

- Use a C# style guide for clean and scalable game code (Unity 6 edition)
- The Unity game designer playbook
- Create modular game architecture in Unity with ScriptableObjects
- Effective asset management in Unity with Addressables
- What you need to know about Build Profiles in Unity 6

# IDEs and debugging

## Pause execution with Debug.Break

If you want to check certain values in the Inspector when the application is difficult to pause manually you can use Debug.Break to pause the execution in your code.

## Save an if statement with Debug.Assert

Debug.Assert checks a condition at runtime and logs an error message to the console if the condition you entered returns false. Unlike **Debug.Log**, which always runs, assertions are meant to flag unexpected states and can be more effective for validating assumptions in your code.

```
// You can save the if statement in release...
if (health > maxhealth)
{
   Debug.LogError(“Current health is greater than maxhealth!”, this);
}
//... by using an assertion
Debug.Assert(health < maxhealth, “Current health is greater than maxhealth!”, this);
```

## Use Debug.Log with context

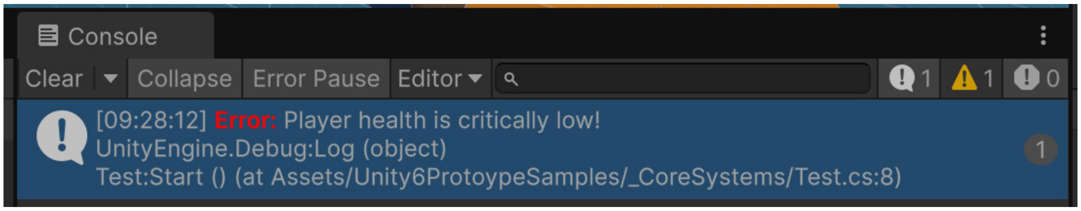
When using **Debug.Log**, you can pass in an object (typically a GameObject or component) as a second parameter. This links the log message to that object in the Console, so when you click the message, Unity highlights the associated object in the Hierarchy.

```
Debug.Log(“Enemy spawned”, gameObject);
```

## Make important messages stand out with Rich Text

Unity’s Console supports a subset of Rich Text (like <b>, <i>, <color>, etc.) in **Debug.Log** messages. You can use these to highlight, color-code, or emphasize parts of your log output, making important messages stand out during development.

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



```
[09:28:12] Error: Player health is critically low!
UnityEngine.Debug:Log (object)
Test:Start () (at Assets/Unity6ProtoypeSamples/_CoreSystems/Test.cs:8)
```

## Strip Debug Log from your builds

Unity does not strip the **Debug** logging APIs from non-development builds automatically. Wrap your **Debug Log** calls in custom methods and decorate them with the **[Conditional]** attribute.

Removing the corresponding **Scripting Define Symbol** from the Player Settings compiles out the Debug Logs all at once. This is identical to wrapping them in **#if… #endif** preprocessor blocks.

See this General Optimizations guide for an example.

## Troubleshoot Physics by visualizing your raycasting

Troubleshooting physics? **Debug.DrawLine** and **Debug.DrawRay** can help you visualize raycasting by drawing a line between specified start and end points.

```
   void Start()
    {
        // draw a 5-unit white line from the origin for 2.5 seconds
        Debug.DrawLine(Vector3.zero, new Vector3(5, 0, 0), Color.
white, 2.5f);
    }
```

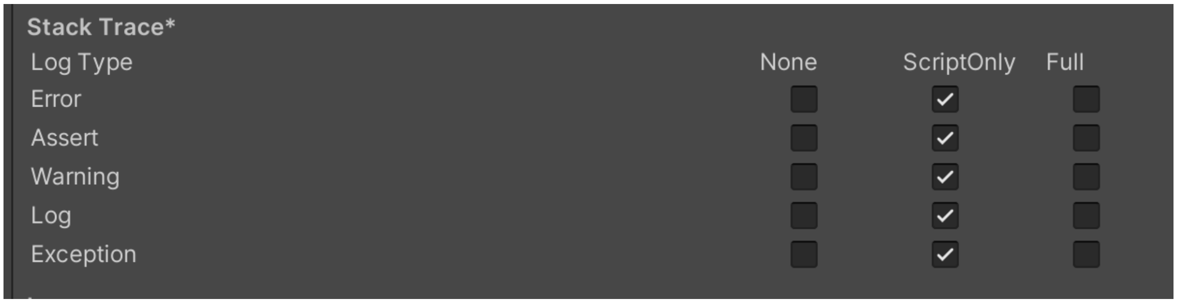
## Use Debug.isDebugBuild for development builds

Use **Debug.isDebugBuild** to check if the application is running as a Development Build. This allows you to conditionally execute debug-only code, such as logging, diagnostics, or test utilities, without affecting release builds.

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

## Set Application.SetStackTraceLogType

Use **Application.SetStackTraceLogType** or the equivalent checkboxes in PlayerSettings to decide which kinds of log messages should include stack traces. Stack traces can be useful, but they are slow and generate garbage.



**Stack Trace***

| Log Type | None | ScriptOnly | Full |
|---|---|---|---|
| Error | ☐ | ☑ | ☐ |
| Assert | ☐ | ☑ | ☐ |
| Warning | ☐ | ☑ | ☐ |
| Log | ☐ | ☑ | ☐ |
| Exception | ☐ | ☑ | ☐ |

Stack Trace in PlayerSettings in the Editor window

## Customize your log

The Logger API allows you to create and configure custom Logger instances for more advanced or modular logging. While you can use the built-in Debug.unityLogger, creating your own logger gives you finer control over log formatting, filtering, and output channels:

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

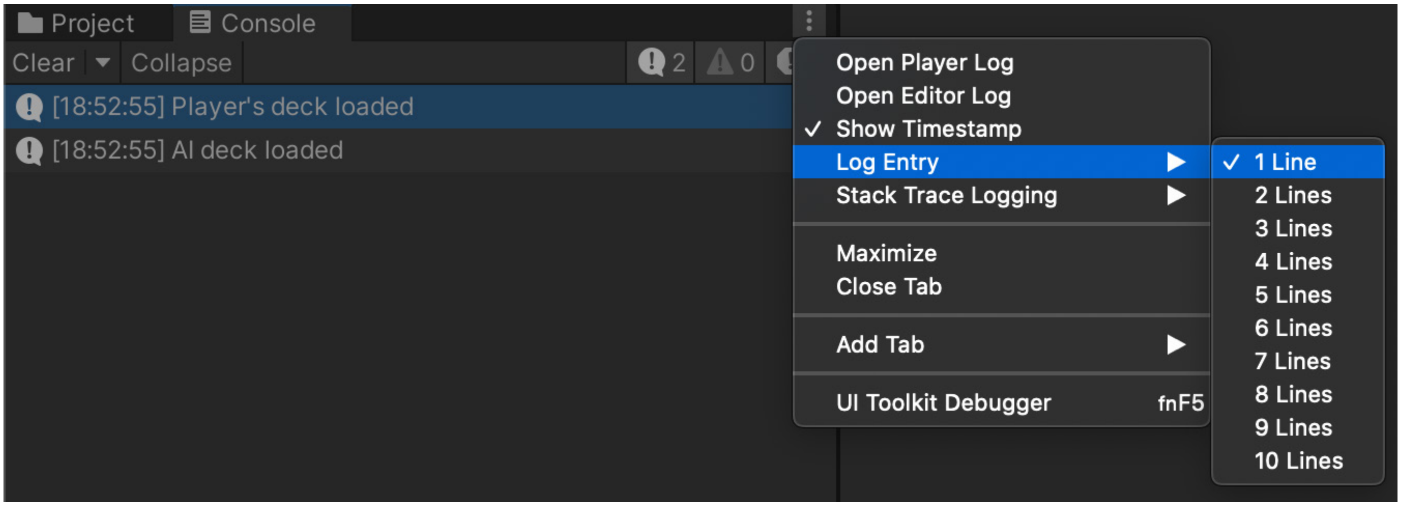
## Speed up your workflows with Visual Code shortcuts

If you use Visual Code as the IDE of choice, these shortcuts may prove useful:

- Windows
- Mac

# Configure your Console Log Entry for improved readability

By default, the Console Log Entry shows two lines. For improved readability, you can configure this to be more streamlined with one or multiple lines depending on your preferences (see image below).



```
[18:52:55] Player's deck loaded
[18:52:55] AI deck loaded
```

The Console Log Entry options allows you to set the number of lines in your log.

# More resources

- How to debug game code with Roslyn Analyzers
- How to run automated tests for your games with the Unity Test Framework
- Speed up your debugging workflow with Microsoft Visual Studio Code
- How to debug your code with Microsoft Visual Studio 2022
- Testing and quality assurance tips for Unity projects

# DevOps workflows

## Unity Version Control tips

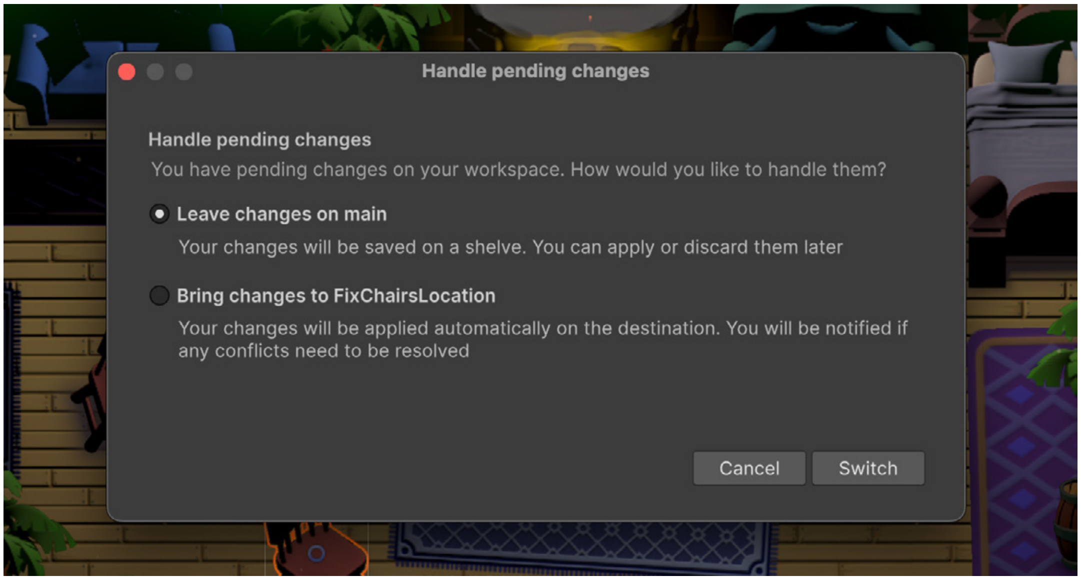
Unity Version Control (UVCS) is a scalable, engine-agnostic version control and source code management tool for game development studios of all sizes.

If you're already using Unity Version Control with Unity 6, here are some of the latest updates and tips that your team could use in your day-to-day work.

### Work on multiple branches in parallel, without losing your work, with Shelve and Switch

Starting with UVCS package version 2.7.1, you can use shelvesets to temporarily save your current changes when switching between branches, to make sure your unfinished work is safely stored and easily accessible.

In your Unity Version Control window, right-click the branch you’d like to switch to and select **Switch workspace to this branch.**



Handle pending changes

Handle pending changes

You have pending changes on your workspace. How would you like to handle them?

● Leave changes on main

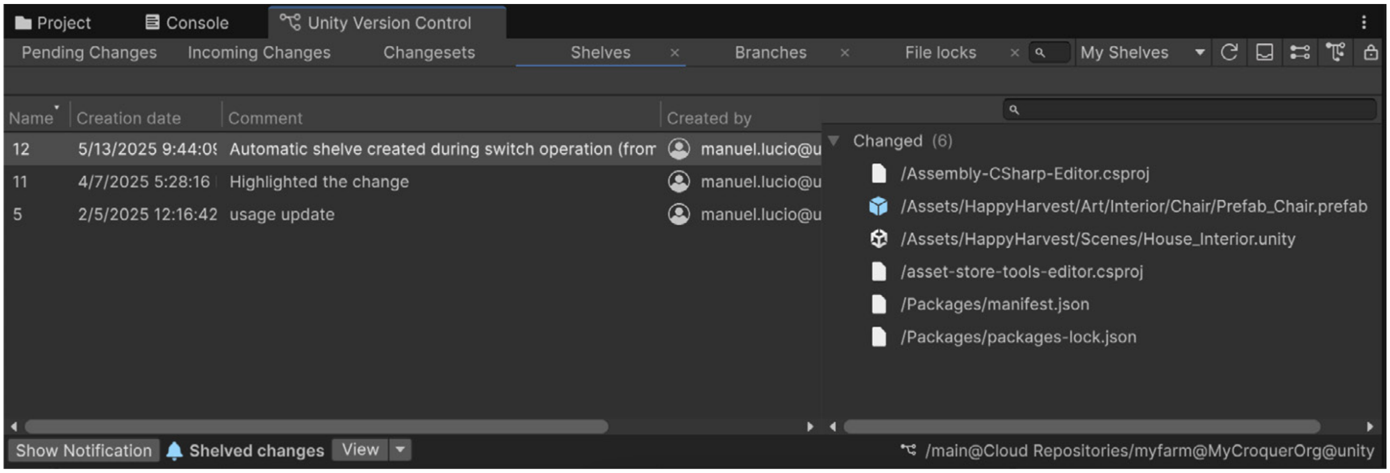
Your changes will be saved on a shelve. You can apply or discard them later

○ Bring changes to FixChairsLocation

Your changes will be applied automatically on the destination. You will be notified if any conflicts need to be resolved

The **Handle pending changes** window

Then choose whether you want to carry your changes or leave them behind temporarily. The **Shelves** option provides great flexibility by allowing you to revisit your changes, re-apply them, and share them with your team.



| Name | Creation date | Comment | Created by |
|---|---|---|---|
| 12 | 5/13/2025 9:44:0$ | Automatic shelve created during switch operation (from | manuel.lucio@u |
| 11 | 4/7/2025 5:28:16 | Highlighted the change | manuel.lucio@u |
| 5 | 2/5/2025 12:16:42 | usage update | manuel.lucio@u |

Changed (6)

- /Assembly-CSharp-Editor.csproj
- /Assets/HappyHarvest/Art/Interior/Chair/Prefab_Chair.prefab
- /Assets/HappyHarvest/Scenes/House_Interior.unity
- /asset-store-tools-editor.csproj
- /Packages/manifest.json
- /Packages/packages-lock.json

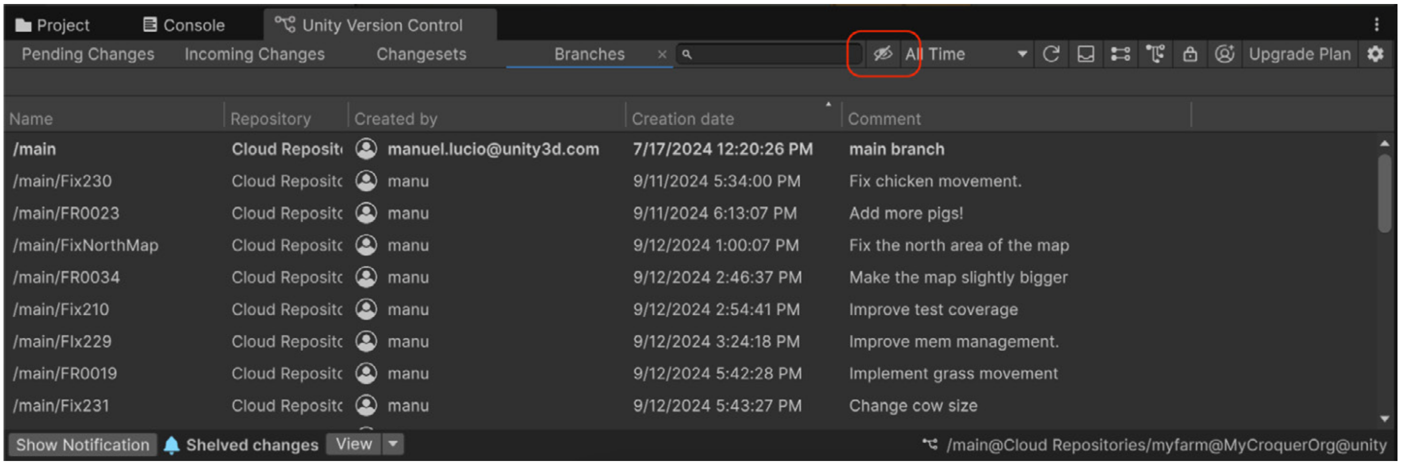
/main@Cloud Repositories/myfarm@MyCroquerOrg@unity

The Shelves tab in the Unity Version Control window

## Declutter your branches view with the Hide/Unhide feature

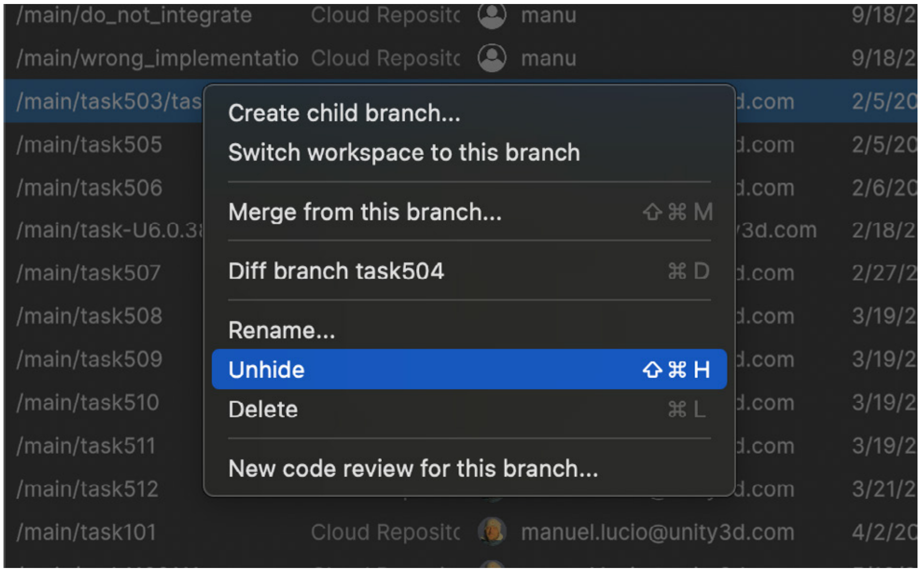
When your project's branches view becomes cluttered, you can use the global **Hide/Unhide** feature to hide branches directly from the context menu, making them invisible for your entire team across all UVCS platforms (Unity plugin, Unity Dashboard, or the Desktop application).

You can make Hidden branches visible again by using the "strikethrough eye" icon or from the context menu of the Hidden branches list (see images below).



| Name | Repository | Created by | Creation date | Comment |
|---|---|---|---|---|
| /main | Cloud Reposit | manuel.lucio@unity3d.com | 7/17/2024 12:20:26 PM | main branch |
| /main/Fix230 | Cloud Reposit | manu | 9/11/2024 5:34:00 PM | Fix chicken movement. |
| /main/FR0023 | Cloud Reposit | manu | 9/11/2024 6:13:07 PM | Add more pigs! |
| /main/FixNorthMap | Cloud Reposit | manu | 9/12/2024 1:00:07 PM | Fix the north area of the map |
| /main/FR0034 | Cloud Reposit | manu | 9/12/2024 2:46:37 PM | Make the map slightly bigger |
| /main/Fix210 | Cloud Reposit | manu | 9/12/2024 2:54:41 PM | Improve test coverage |
| /main/FIx229 | Cloud Reposit | manu | 9/12/2024 3:24:18 PM | Improve mem management. |
| /main/FR0019 | Cloud Reposit | manu | 9/12/2024 5:42:28 PM | Implement grass movement |
| /main/Fix231 | Cloud Reposit | manu | 9/12/2024 5:43:27 PM | Change cow size |

The strikethrough eye icon in Unity Version Control

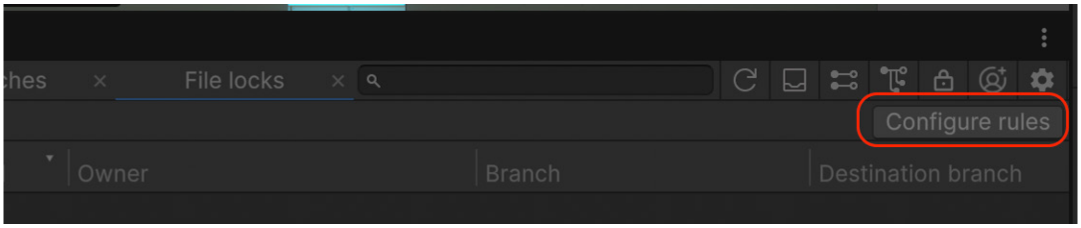


The "Unhide" option in the branch context menu

## Avoid losing asset data with the exclusive lock option

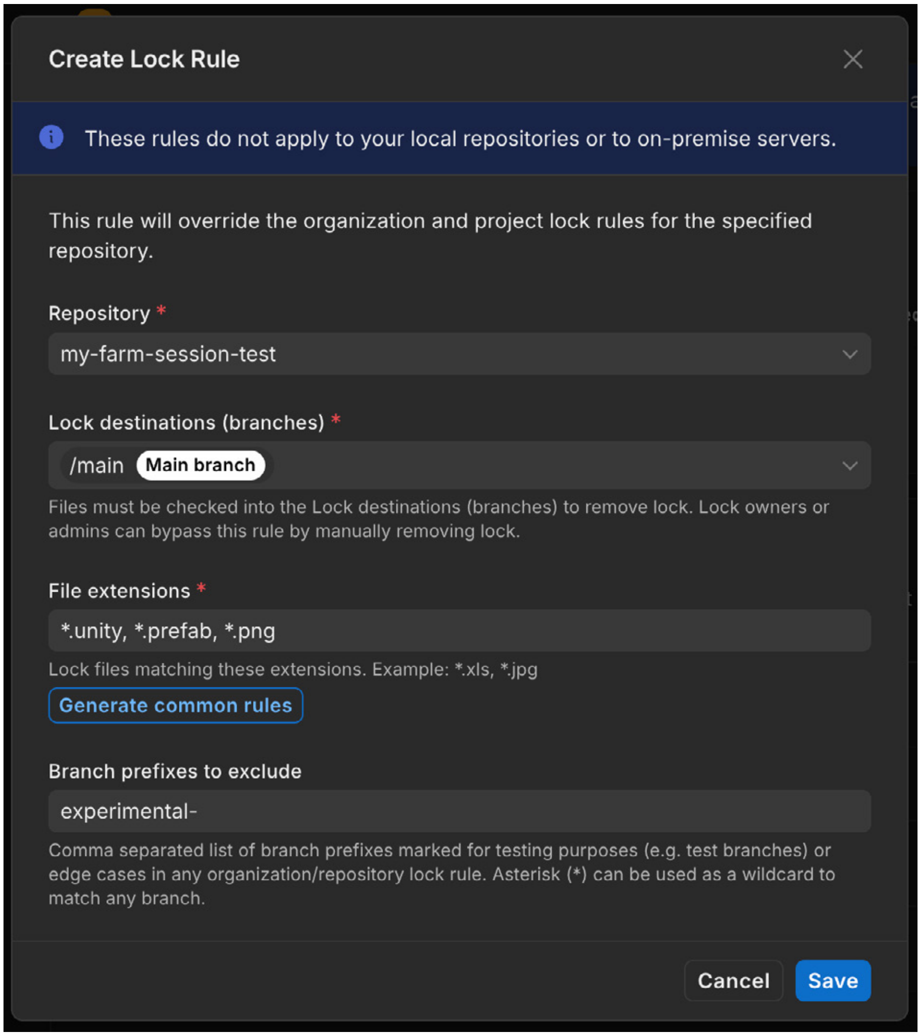
The UVCS plugin for the Unity Editor allows you to lock an asset to prevent others from modifying it. This helps the team to avoid losing any work-in-progress data and removes the merge operation for files that can't be merged.

To enable the lock support go to **Configure rules** in the **File locks** tab.



The Configure rules option in the File locks tab

Then configure the repository, the destination branch (usually the main branch), and the file extensions you want in order to enable the lock support.



Create Lock Rule

These rules do not apply to your local repositories or to on-premise servers.

This rule will override the organization and project lock rules for the specified repository.

Repository *
my-farm-session-test

Lock destinations (branches) *
/main Main branch

Files must be checked into the Lock destinations (branches) to remove lock. Lock owners or admins can bypass this rule by manually removing lock.

File extensions *
*.unity, *.prefab, *.png

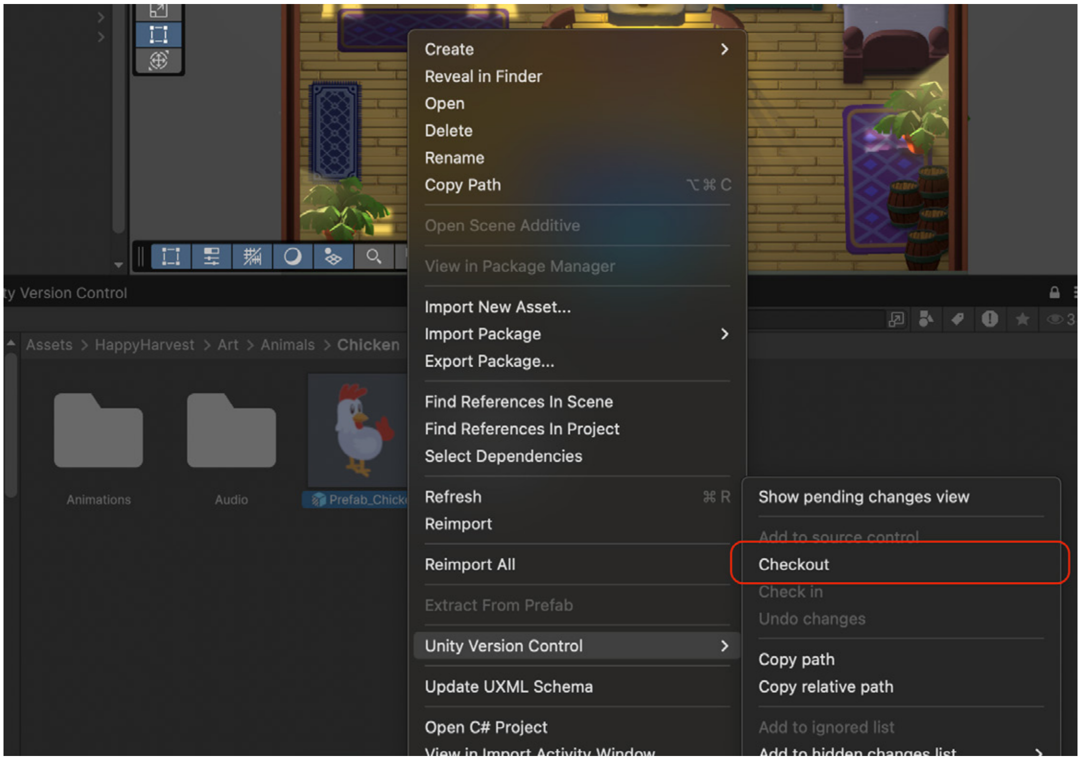
Lock files matching these extensions. Example: *.xls, *.jpg

Branch prefixes to exclude
experimental-

Comma separated list of branch prefixes marked for testing purposes (e.g. test branches) or edge cases in any organization/repository lock rule. Asterisk (*) can be used as a wildcard to match any branch.

The Create Lock Rule in Unity Version Control

To avoid conflicts, make sure to check out the assets before making changes.



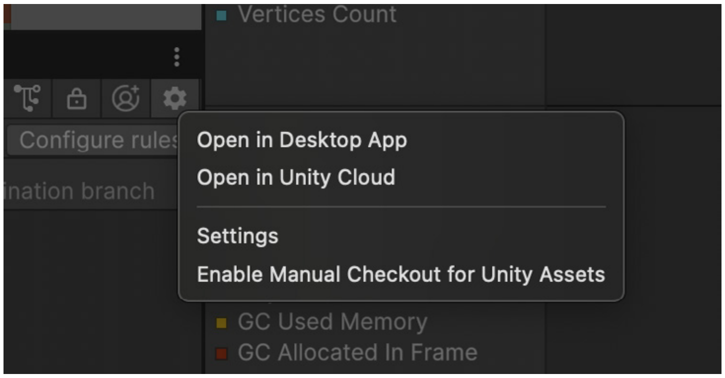
The Checkout option from the Unity Version Control context menu

If you want to learn more about the smart locks system check out the Smart Locks page.

## Protect your changes with "Enable manual checkout for Unity Assets"

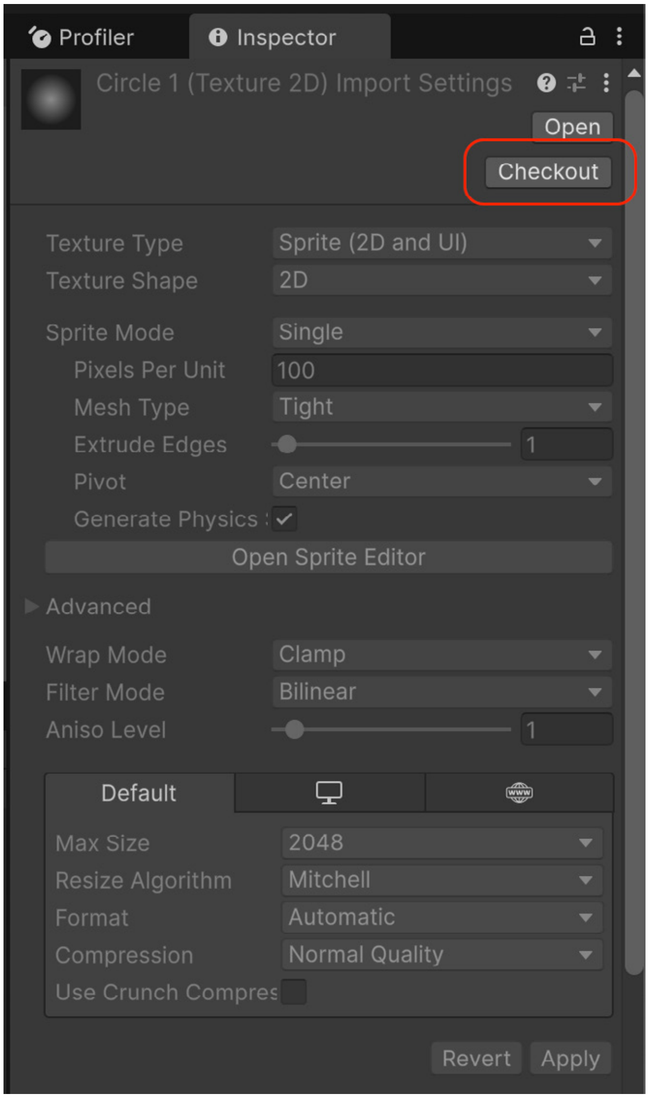
When working with unmergeable assets, the UVCS Unity plugin will automatically check out a file after you modify and save it. This workflow leaves room for someone else to lock the same file without your changes.

To avoid this from happening, go to the UVCS plugin settings via clicking on the **gear** icon, and select the option **Enable manual checkout for Unity Assets**.



The **Enable Manual Checkout for Unity Assets** option

This preference makes the asset read-only until it's explicitly checked out. When you click the **Checkout** option in the Inspector, your file becomes editable and you'll get the exclusive lock to change it.



Circle 1 (Texture 2D) Import Settings

Texture Type: Sprite (2D and UI)
Texture Shape: 2D
Sprite Mode: Single
Pixels Per Unit: 100
Mesh Type: Tight
Extrude Edges: 1
Pivot: Center
Generate Physics : ✓
Wrap Mode: Clamp
Filter Mode: Bilinear
Aniso Level: 1
Max Size: 2048
Resize Algorithm: Mitchell
Format: Automatic
Compression: Normal Quality
Use Crunch Compres: □

The Checkout option in the Inspector

## Asset Manager

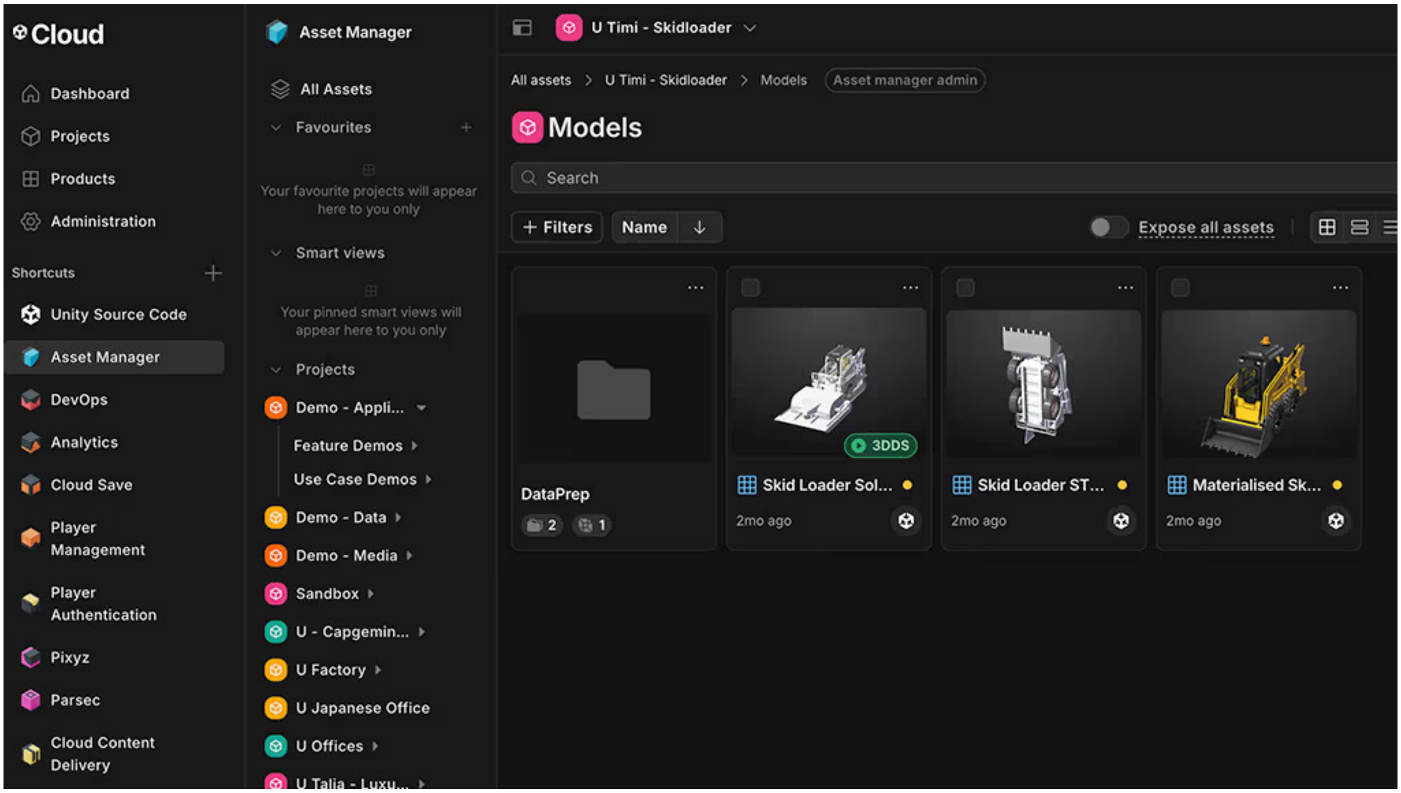
If you're looking for a digital asset management system to manage your assets, such as models, textures, audio files, etc., then check out Unity's Asset Manager. It supports over 70 file formats to help teams centralize, organize, discover, and use assets seamlessly across projects.

Use the Asset Manager via the web interface or through the Editor integration, depending on your needs.

## Manage, view, and organize your assets in your preferred browser

The web interface enables asynchronous, speedy collaboration among all team members and stakeholders involved in your projects. Through the web interface, you can create and manage collections, move assets between collections or projects, and track changes with version history.

You can easily upload single or multiple files while adding relevant collections and tags, making it straightforward to structure your assets and collaborate with your team to review and improve them. You can enrich assets with custom metadata, where admins, owners, and managers can define new metadata fields – like text, Boolean, and numeric field types – while all users can add tags, including AI-suggested tags, for overall better organization.

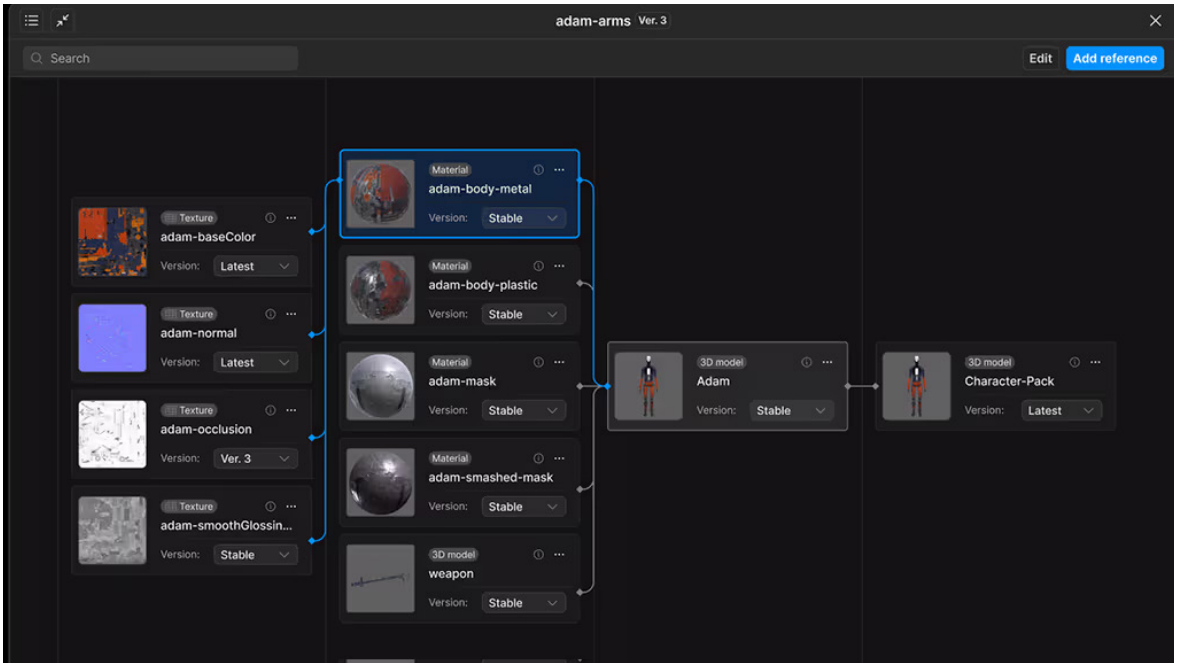


Your favourite projects will appear here to you only

Your pinned smart views will appear here to you only

Working with assets in the web interface of the Asset Manager

The **Dependency Viewer** reveals asset relationships at a glance, showing what each asset needs and what depends on it. Both upstream and downstream dependencies are available in the Dependency Viewer.

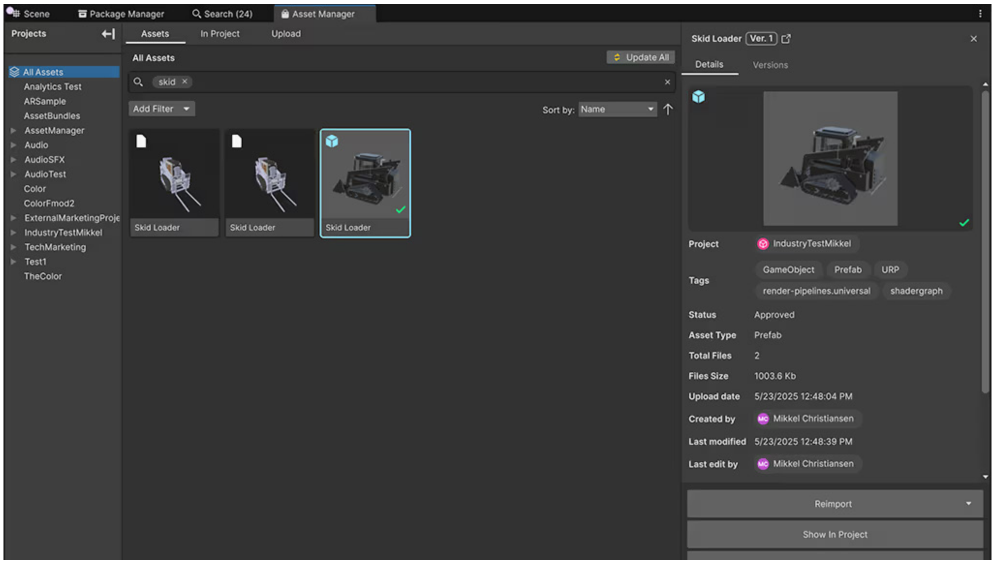


An example of asset dependencies in the Dependency Viewer

## Manage assets without any disruptions in the Editor

The Unity Asset Manager package integrates with the Unity Editor, allowing you to manage assets without disrupting your workflow. After installing it, you can upload assets to, and import them from, the Unity Cloud, as well as access assets from all your projects through the Editor's search interface.

The in-Editor solution also makes it easier to reuse prefabs, scripts, and other cross-project components, speeding up iteration time and enabling teams to focus on better workflows and optimization.



Project IndustryTestMikkel
Tags GameObject Prefab URP render-pipelines.universal shadergraph
Status Approved
Asset Type Prefab
Total Files 2
Files Size 1003.6 Kb
Upload date 5/23/2025 12:48:04 PM
Created by Mikkel Christiansen
Last modified 5/23/2025 12:48:39 PM
Last edit by Mikkel Christiansen

Accessing the Asset Manager from the Unity Editor

To help you locate specific assets efficiently, make sure to filter by creator, status, type, update time or import status. The plugin automatically detects dependencies across assets. For example, if multiple prefabs use the same material, only one instance of the material is uploaded. This ensures optimized storage and prevents unnecessary duplication.

Learn more about the Unity Asset Manager from the following resources:

- Video tutorial: A quick guide to the Asset Manager in Unity
- Unity Learn: Unity Asset Manager: Quick start guide
- Article: Introduction to Asset Manager transfer methods in Unity

## Unity Build Automation

Unity DevOps Build Automation, formerly known as Cloud Build, is a turnkey continuous integration and deployment (CI/CD) solution that can execute and deploy builds in the cloud. It empowers you to build and release more often for higher-quality, more innovative releases.

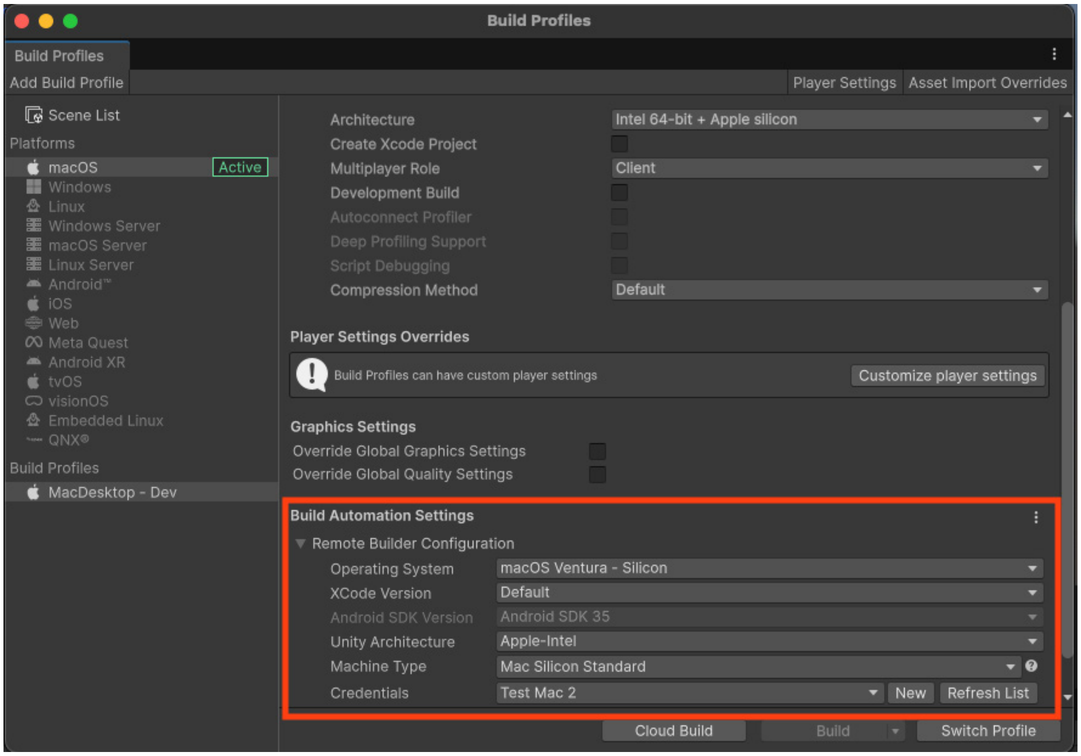
Build Automation can be connected to any source control repository in a matter of minutes and set up to execute builds manually or automatically once any change is committed to version control. Build Automation also supports multiple platforms including iOS, Android, Windows, and Unity Web, eliminating the need to maintain unique build infrastructure for every platform.

As of Unity 6.1, Build Automation is now fully integrated with the Editor and you can run quick validation builds from local branches with your build automation settings.

### Share and reuse build settings across your teams

You can install the Build Automation package from the Build Profiles window or from the Package Manager.

Once you install the package, a new section labeled **Build Automation Settings** will appear under your selected profile. These settings allow you to define how Unity automates your builds.



Architecture: Intel 64-bit + Apple silicon
Create Xcode Project ☐
Multiplayer Role: Client
Development Build ☐
Autoconnect Profiler ☐
Deep Profiling Support ☐
Script Debugging ☐
Compression Method: Default

Player Settings Overrides
Build Profiles can have custom player settings — Customize player settings

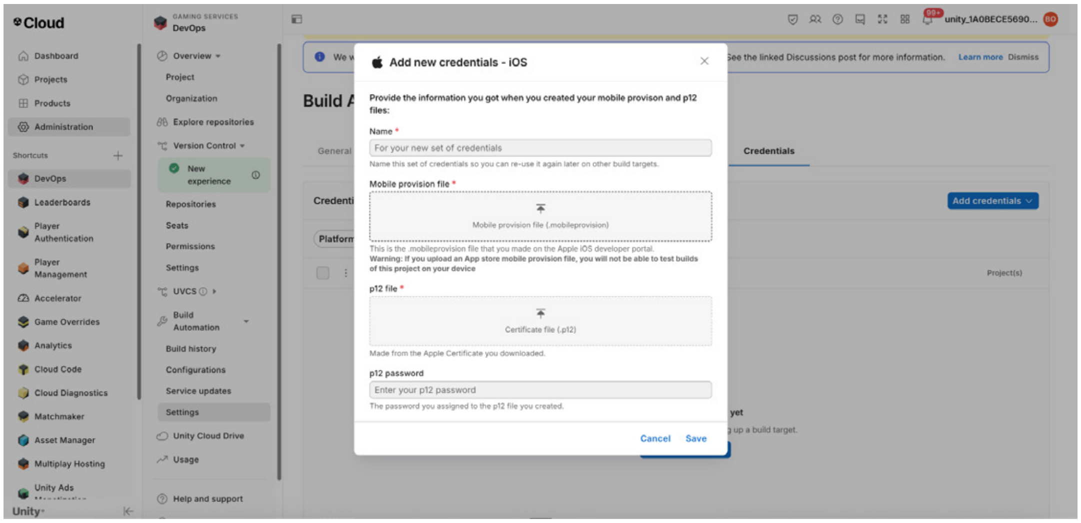
Graphics Settings
Override Global Graphics Settings ☐
Override Global Quality Settings ☐

Build Automation Settings
Remote Builder Configuration
Operating System: macOS Ventura - Silicon
XCode Version: Default
Android SDK Version: Android SDK 35
Unity Architecture: Apple-Intel
Machine Type: Mac Silicon Standard
Credentials: Test Mac 2 — New — Refresh List

The Build Automation Settings section in your Build Profiles window

For Android, iOS, and other platforms that require credentials, you'll have to configure the credentials in the Unity Dashboard. To create new credentials, click on the **New** button next to the **Credentials** drop-down.

This opens a new window in the Build Automation dashboard where you can create a new set of credentials. To learn more about credentials and app signing make sure to check out the Unity Build Automation documentation.



Add new credentials - iOS

Provide the information you got when you created your mobile provison and p12 files:

Name *
For your new set of credentials
Name this set of credentials so you can re-use it again later on other build targets.

Mobile provision file *
Mobile provision file (.mobileprovision)
This is the .mobileprovision file that you made on the Apple iOS developer portal.
**Warning: If you upload an App store mobile provision file, you will not be able to test builds of this project on your device**

p12 file *
Certificate file (.p12)
Made from the Apple Certificate you downloaded.

p12 password
Enter your p12 password
The password you assigned to the p12 file you created.

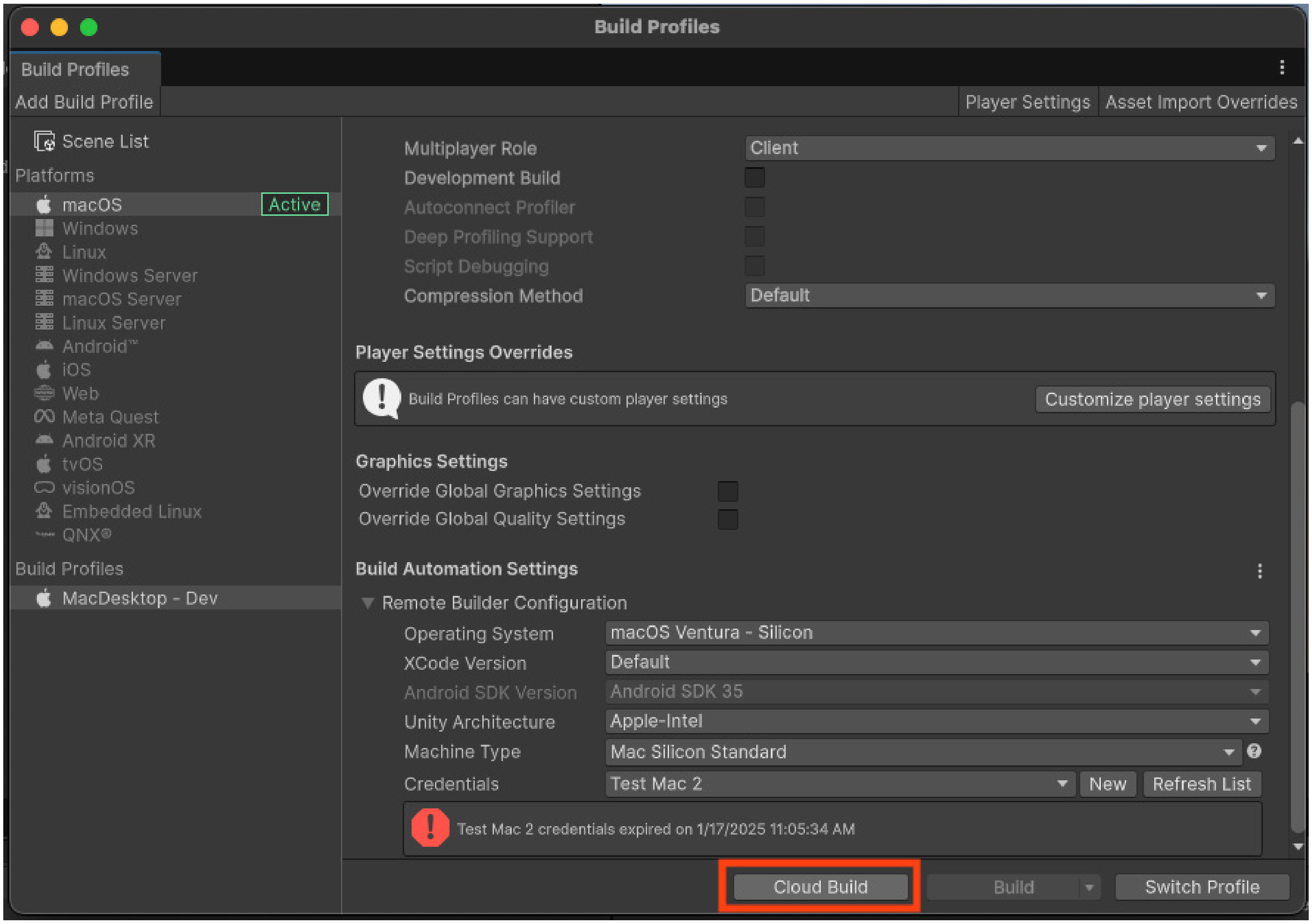
Cancel Save

Credentials configurations window in the Unity Dashboard

Once you save your credentials, go back to the Build Profiles window and click on **Refresh list**. You should see the credentials in the drop-down list. The build settings are automatically saved to your build profile, which means you can reuse and share across your team.

## Test work in progress with Shelve and Build

Once you've configured your settings, and you're ready to run a build, click the **Cloud Build** button in your build profile. Unity will then begin processing your build in the cloud.



Multiplayer Role: Client
Development Build: □
Autoconnect Profiler: □
Deep Profiling Support: □
Script Debugging: □
Compression Method: Default

Player Settings Overrides

Build Profiles can have custom player settings

Graphics Settings

Override Global Graphics Settings: □
Override Global Quality Settings: □

Build Automation Settings

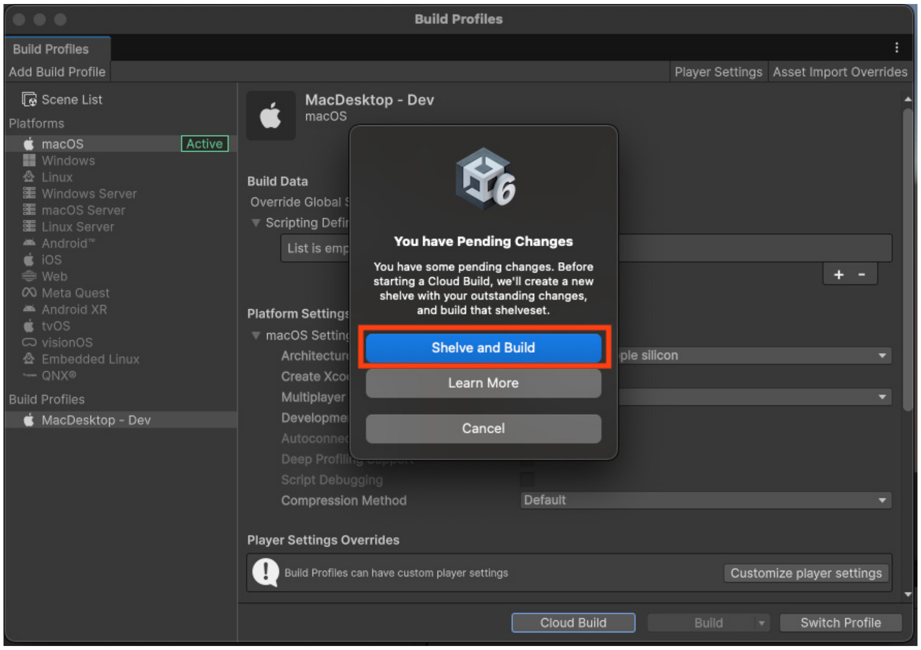
Remote Builder Configuration

Operating System: macOS Ventura - Silicon
XCode Version: Default
Android SDK Version: Android SDK 35
Unity Architecture: Apple-Intel
Machine Type: Mac Silicon Standard
Credentials: Test Mac 2

Test Mac 2 credentials expired on 1/17/2025 11:05:34 AM

Cloud Build settings in your build profile

If you have pending changes in your project, you'll be prompted with the option to **Shelve and Build**. This allows you to build your local changes without pushing them to the main repository which is ideal for testing work in progress.



You have Pending Changes

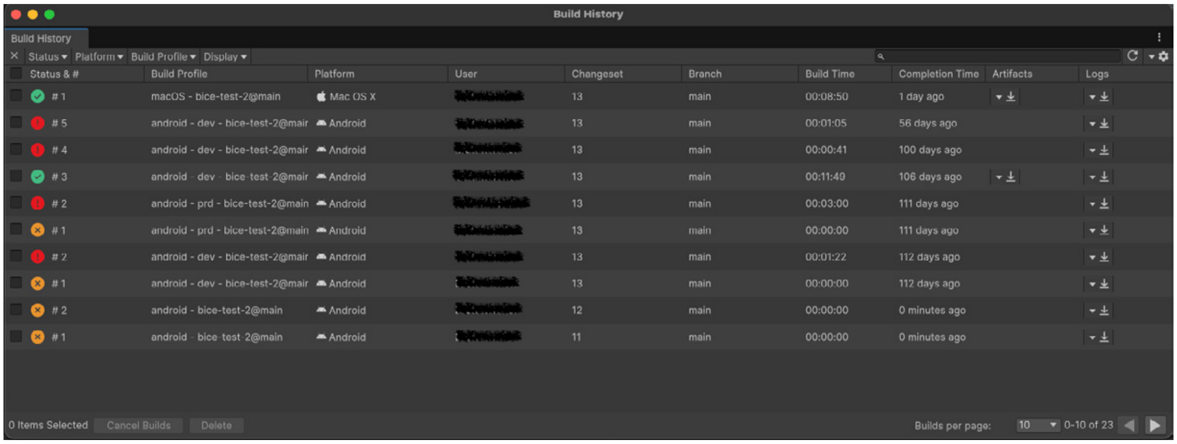
You have some pending changes. Before starting a Cloud Build, we'll create a new shelve with your outstanding changes, and build that shelveset.

Shelve and Build pending changes

If you have no pending changes, Unity will create a build using the latest commit from the branch you currently have checked out.

## Track build progress, troubleshoot, and download results with Build History

As soon as the build begins, the Editor will open the **Build History** window where you can monitor its progress and the build status. This view provides logs and access to the build artifacts, allowing you to troubleshoot or download results easily.



| Status & # | Build Profile | Platform | User | Changeset | Branch | Build Time | Completion Time | Artifacts | Logs |
|---|---|---|---|---|---|---|---|---|---|
| ✓ # 1 | macOS - bice-test-2@main | Mac OS X | [redacted] | 13 | main | 00:08:50 | 1 day ago | ▾ ⤓ | ▾ ⤓ |
| ! # 5 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:01:05 | 56 days ago | | ▾ ⤓ |
| ! # 4 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:00:41 | 100 days ago | | ▾ ⤓ |
| ✓ # 3 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:11:49 | 106 days ago | ▾ ⤓ | ▾ ⤓ |
| ! # 2 | android - prd - bice-test-2@main | Android | [redacted] | 13 | main | 00:03:00 | 111 days ago | | ▾ ⤓ |
| × # 1 | android - prd - bice-test-2@main | Android | [redacted] | 13 | main | 00:00:00 | 111 days ago | | ▾ ⤓ |
| ! # 2 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:01:22 | 112 days ago | | ▾ ⤓ |
| × # 1 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:00:00 | 112 days ago | | ▾ ⤓ |
| × # 2 | android - bice-test-2@main | Android | [redacted] | 12 | main | 00:00:00 | 0 minutes ago | | ▾ ⤓ |
| × # 1 | android - bice-test-2@main | Android | [redacted] | 11 | main | 00:00:00 | 0 minutes ago | | ▾ ⤓ |

0 Items Selected

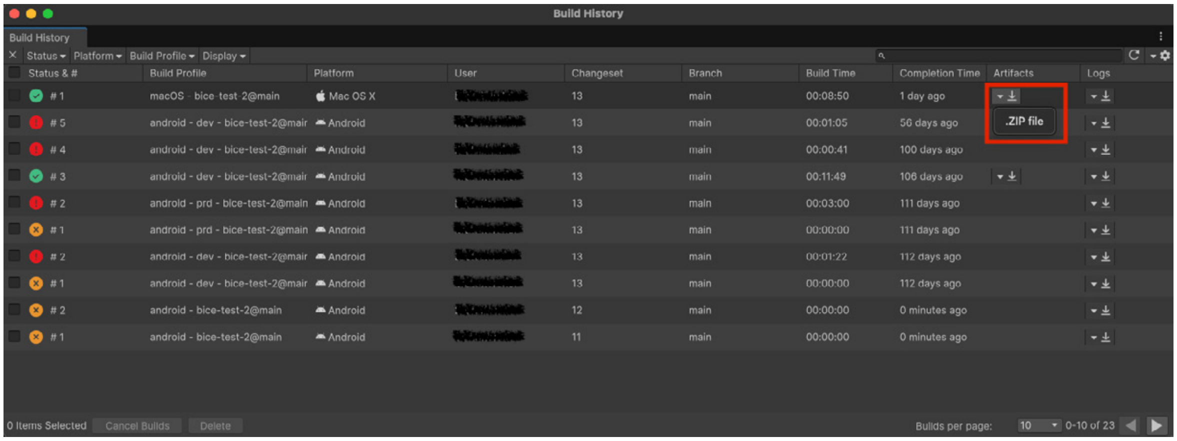
Builds per page: 10 0-10 of 23

Build History window

Each status is color-coded for quick visibility and allows you to take actions, e.g., rerun failed builds or download the log. Check out this documentation page to get a full overview of what each status symbol means.

### Share build artifacts links with your team members

Once builds are successfully completed, you can download the build artifacts or share links with team members.



| Status & # | Build Profile | Platform | User | Changeset | Branch | Build Time | Completion Time | Artifacts | Logs |
|---|---|---|---|---|---|---|---|---|---|
| ✓ # 1 | macOS - bice-test-2@main | Mac OS X | [redacted] | 13 | main | 00:08:50 | 1 day ago | ▾ ⤓<br>.ZIP file | ▾ ⤓ |
| ! # 5 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:01:05 | 56 days ago |  | ▾ ⤓ |
| ! # 4 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:00:41 | 100 days ago |  | ▾ ⤓ |
| ✓ # 3 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:11:49 | 106 days ago | ▾ ⤓ | ▾ ⤓ |
| ! # 2 | android - prd - bice-test-2@main | Android | [redacted] | 13 | main | 00:03:00 | 111 days ago |  | ▾ ⤓ |
| ✕ # 1 | android - prd - bice-test-2@main | Android | [redacted] | 13 | main | 00:00:00 | 111 days ago |  | ▾ ⤓ |
| ! # 2 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:01:22 | 112 days ago |  | ▾ ⤓ |
| ✕ # 1 | android - dev - bice-test-2@mair | Android | [redacted] | 13 | main | 00:00:00 | 112 days ago |  | ▾ ⤓ |
| ✕ # 2 | android - bice-test-2@main | Android | [redacted] | 12 | main | 00:00:00 | 0 minutes ago |  | ▾ ⤓ |
| ✕ # 1 | android - bice-test-2@main | Android | [redacted] | 11 | main | 00:00:00 | 0 minutes ago |  | ▾ ⤓ |

Download build artifacts for successful builds

For failed builds, you can download the logs for further inspection.

Learn more about Build Automation from the Getting started with Build Automation in Unity video tutorial and the Unity Build Automation: Quick start guide tutorial on Unity Learn.

## Unity Build Server

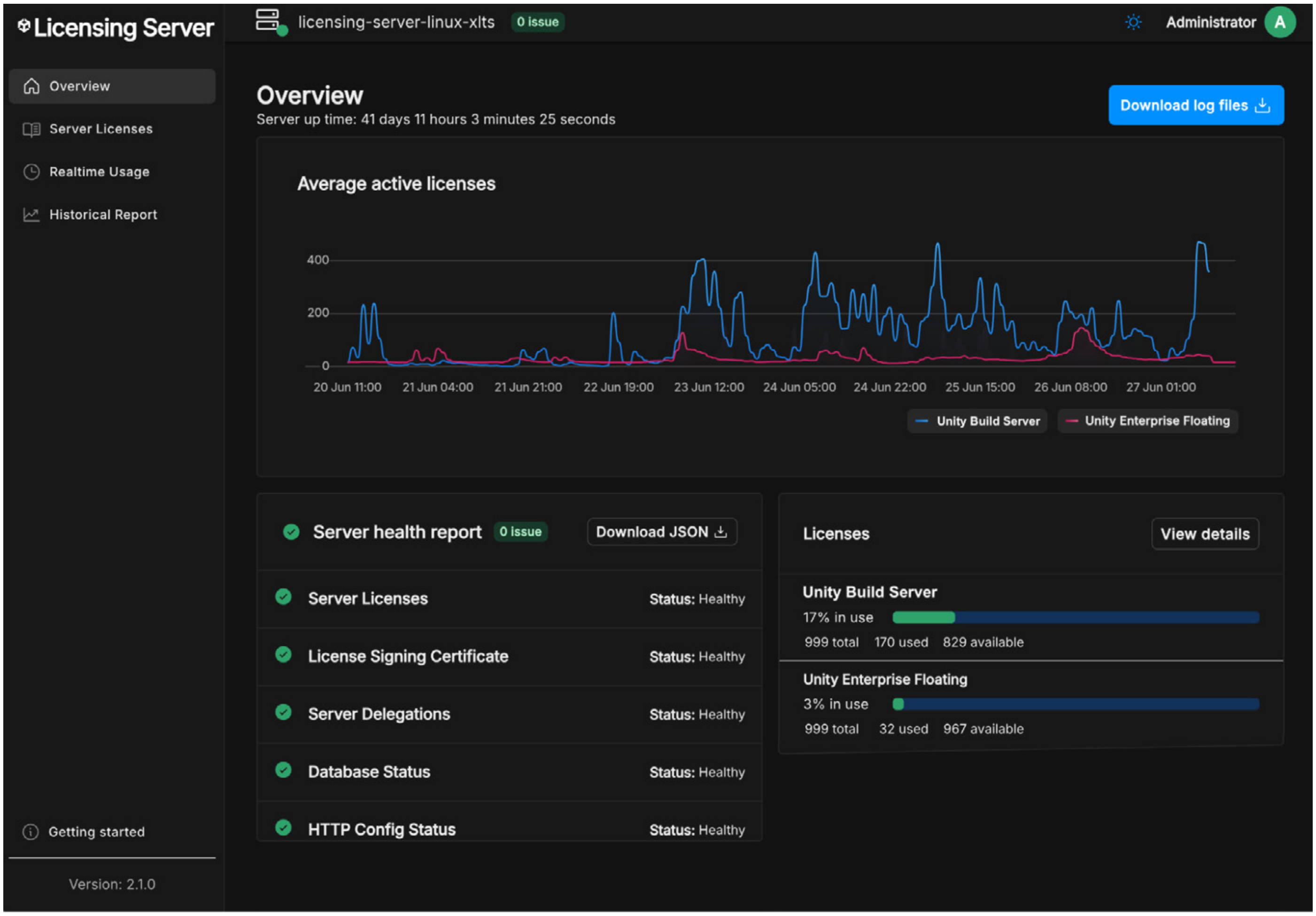
If you want to streamline and automate the process of building, compiling, and packaging Unity projects, especially across larger teams and organizations, try Unity Build Server.

This is a license management tool that enables floating build licenses for Unity, allowing multiple machines to build without requiring a full Unity Editor license for each machine. It is a great addition to your automated build pipeline and continuous integration (CI) environments.

To start using Unity Build Server, you can deploy the Unity Licensing Server software on a central server in your network and add your Unity Build Server licenses to this server. Then, you can configure each build machine to request a build license from the Build Server when it needs to build a project.

The Unity Licensing Server that is used in the Build Server solution can support more than one type of subscription on the same server. Customers can use the same server to distribute floating licenses to the developers using a Unity Enterprise Floating or a Unity Industry Floating subscription. See an example in the image below.

You can learn more about the requirements for Unity Build Server, how to set it up, and how to get started from the Unity Licensing Server documentation.



Overview

Server up time: 41 days 11 hours 3 minutes 25 seconds

Server health report 0 issue

| | |
|---|---|
| Server Licenses | Status: Healthy |
| License Signing Certificate | Status: Healthy |
| Server Delegations | Status: Healthy |
| Database Status | Status: Healthy |
| HTTP Config Status | Status: Healthy |

Licenses

Unity Build Server
17% in use
999 total 170 used 829 available

Unity Enterprise Floating
3% in use
999 total 32 used 967 available

Two active subscriptions: Unity Build Server (for build machines) and Unity Enterprise Floating (for developers).

# Sample projects

From sample projects that provide game interfaces that showcase UI Toolkit and UI Builder workflows, or samples demonstrating various design patterns and project architecture, as well as those that showcase capabilities of 2D lighting and visual effects in the Universal Render Pipeline (URP) in Unity 6, we've got you covered with these projects.

— Happy Harvest - 2D Sample Project

— Gem Hunter Match - 2D Sample Project

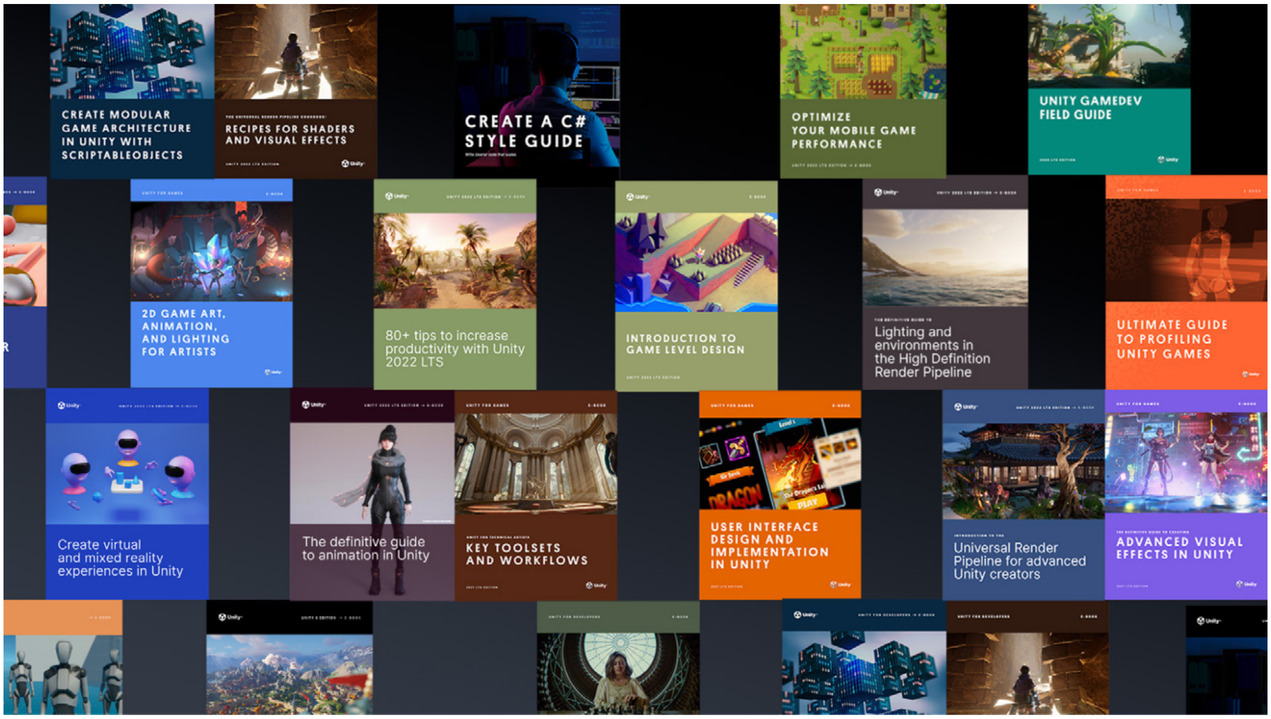
— Dragon Crashers - UI Toolkit Sample Project

— QuizU - A UI Toolkit sample

— Paddle Game ScriptableObjects

- Level up your code with design patterns and SOLID
- C# Code style guide
- Fantasy Kingdom
- Shader Graph samples
- VFX Learning Templates Sample Content
- URP 3D sample
- HDRP sample scene
- HDRP Time Ghost
- VR Multiplayer Template
- MR multiplayer template
- UGS samples

## Resources for all Unity users


CREATE MODULAR GAME ARCHITECTURE IN UNITY WITH SCRIPTABLEOBJECTS
RECIPES FOR SHADERS AND VISUAL EFFECTS
CREATE A C# STYLE GUIDE
OPTIMIZE YOUR MOBILE GAME PERFORMANCE
UNITY GAMEDEV FIELD GUIDE
2D GAME ART, ANIMATION, AND LIGHTING FOR ARTISTS
80+ tips to increase productivity with Unity 2022 LTS
INTRODUCTION TO GAME LEVEL DESIGN
Lighting and environments in the High Definition Render Pipeline
ULTIMATE GUIDE TO PROFILING UNITY GAMES
Create virtual and mixed reality experiences in Unity
The definitive guide to animation in Unity
KEY TOOLSETS AND WORKFLOWS
USER INTERFACE DESIGN AND IMPLEMENTATION IN UNITY
Universal Render Pipeline for advanced Unity creators
ADVANCED VISUAL EFFECTS IN UNITY

You can download many more e-books for advanced Unity developers and creators from the Unity best practices hub. Choose from over 30 guides, created by industry experts, and Unity engineers and technical artists, that provide best practices for efficient game development with Unity's toolsets and systems.

You'll also find tips, best practices, and news on the Unity Blog and Unity community forums, as well as through Unity Learn and the **#unitytips** hashtag.

Unity®

unity.com