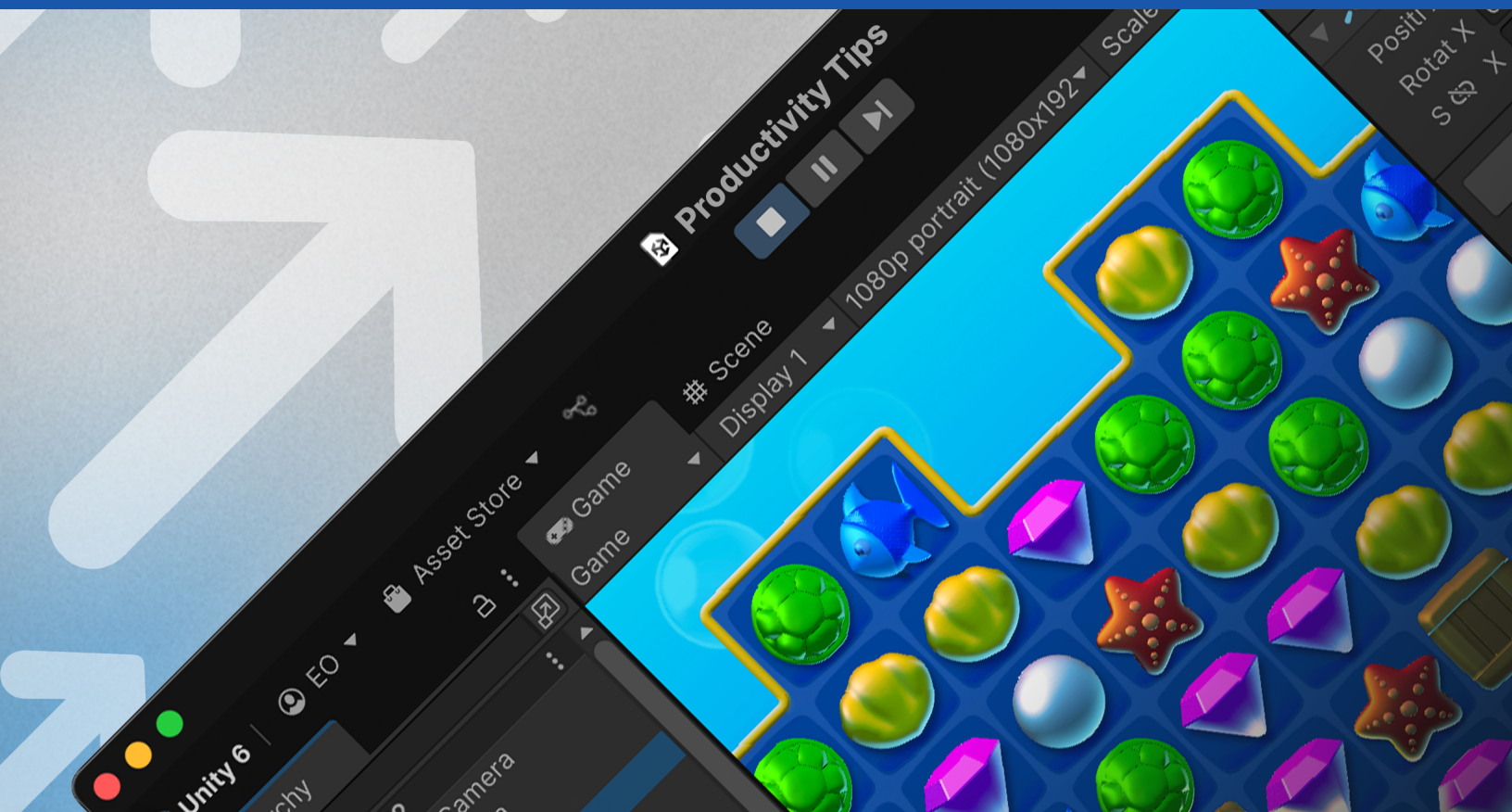




Unity 6 で 生産性を 高めるためのヒント



Contents

はじめに	7
エディターのワークフロー	8
キーボードショートカットを簡単にカスタマイズする	8
複数の Inspector で作業する	10
プリセットで好みのデフォルト設定を定義	11
シーンビューをシーンでの表示/非表示で整理する	13
シーンビューで間違ったオブジェクトを選択しないようにする	15
Inspector でアセットを直接プレビュー	17
プロジェクト内の検索を迅速化	18
Inspector でデバッグデータを表示する	20
視覚的なデバッグにカスタム Gizmo とアイコンを使用する	21
ネスト状のプレハブとプレハブバリエーションのワークフローを 活用する	22
Splines パッケージを使用してカーブパスを簡単に作成する	25
Build Profiles でビルドをカスタマイズ	26
Recorder パッケージで滑らかなゲーム映像をキャプチャする	27
エディターワークフローのヒント	28
その他のリソース	34
2D	35
スプライト	35
スプライトのパック時にアセットの重複を避ける	35
一貫性を保つようにフォルダーを構成する	36
Aseprite Importer でピクセルアートを Unity に直接取り込む	36
Aseprite からフレーム単位のアニメーションを Unity に取り込む	37
PSD Importer でインポートの往復作業を迅速化する	38

Unity 6.1 の Tilemap Editor で Rule Tile を簡単に.....	39
タイル間のテクスチャのにじみや小さな隙間を避ける.....	40
2D インバースキネマティクスシステムで自然な動きを 作成する	42
IES プロファイルを 2D ライトでシミュレートする	43
2D スプライトシェイプで自由度の高い豊かな 2D 環境を 作成する	44
カスタムスプライトソート軸	45
スプライトカスタム Lit シェーダーでカスタムライティングを 作成する	45
透明ピクセルのオーバードローを減らす.....	47
Sprite Editor で未使用領域を最小限に抑える	48
Sprite Library でプロジェクトのスプライトと アニメーションを整理する	48
スプライトシェイプの制御点を変更する	49
コンテキストメニューからスプライトを簡単にスワップする ...	50
グラフィックスとアートアセット.....	52
ライトプローブによるライトリーク.....	52
レンダリングレイヤーを使用して特定のゲームオブジェクトに 特定のライトを設定する.....	53
Decal Projector でメッシュにディテールを加える	54
カスタムシェーダーをビルトインレンダーパイプラインから URP に変換する	54
LUT テクスチャでカラーグレーディングを作成する.....	55
レンズフレアでリアルなレンズ効果と スタイル化された外観を作成する	57
シェーダーバリエントを管理する.....	58
マテリアルのバリエントを作成する	59
アセットワークフローを改善するためアセットのグループ化を 計画する	59

AssetPostProcessor API でインポートパイプラインを 自動化および迅速化する	60
プレハブバリエーションを使用してチームの作業を効率化する	61
XR やモバイル開発におけるアセットの考慮事項	61
広いテクスチャ領域をトリムシートで最適に配置する	63
AR/VR での作業時に 3D モデルを正しいスケールで エクスポートする	63
URP および HDRP プロジェクト用のシェーダーグラフ	65
UI Toolkit	66
視覚的なリファレンスでインターフェースをデザインする	66
Photoshop ファイルを扱うときに PSD Importer で 反復処理を迅速化する	67
ゲームで絵文字を使用する	67
デバイスの OS に含まれているビルトインの絵文字を 使用する	68
UI Builder でビジュアル要素に関する追加情報を表示する ...	70
ローカライゼーションを早い段階で統合して、 より多くの市場に進出する	70
グラデーションでインターフェース全体にスタイルを加える ...	71
USS 遷移で UI をアニメーション化する	72
解決済みスタイルのバウンディングボックスをエディターで 視覚化する	72
UXML ファイルをテンプレートとして再利用し ワークフローを迅速化する	73
その他のリソース	74
開発者ワークフロー	75
Awaitable クラス	75
属性で Inspector ウィンドウを強化する	76

独自のカスタムウィンドウと Inspector を作成する.....	79
カスタムメニューを作成する.....	80
再生モードへの遷移の迅速化	80
デフォルトのスクリプトテンプレートをカスタマイズする	81
Addressables を使用してプレイヤーにオンデマンドで コンテンツを配布する	81
プリプロセッサディレクティブを使用して 条件付きコンパイルされたコードを作成する	82
ScriptableObject を使用してデータとロジックを分離する ..	83
Assembly Definition を使用して スクリプトのモジュール化を促進する	86
Input System にアップグレードする	88
プロファイリングツール	88
アニメーションカーブ	91
オブジェクトプールで処理負荷を軽減する	92
その他のリソース	92
IDE とデバッグ	93
Debug.Break で実行を一時停止する	93
Debug.Assert で if ステートメントをなくす	93
コンテキストとともに Debug.Log を使用する	94
リッチテキストで重要なメッセージを目立たせる	94
デバッグログをビルドから削除する.....	94
レイキャスティングを視覚化して 物理演算のトラブルシューティングを行う	94
開発ビルドで Debug.isDebugBuild を使用する	95
Application.SetStackTraceLogType を設定する.....	95
ログをカスタマイズする	95
Visual Code のショートカットでワークフローを迅速化する ..	95

Console の Log Entry を設定して可読性を高める	96
その他のリソース	96
DevOps ワークフロー	97
Unity Version Control のヒント	97
Asset Manager	102
Unity Build Automation	105
Unity Build Server	109
サンプルプロジェクト	110
すべての Unity ユーザー向けのリソース	111

はじめに

このガイドは、Unity のツールセットでより効率的に作業するためのヒントを多数紹介しています。アーティストにもプログラマーにも役立つものです。Unity 6 の新機能と、Unity に長年組み込まれてきた時間節約のステップやワークフローについて説明しています。

個人でもチームでも、毎日 Unity で作業していると、時間もマウスのクリック数も積み重なります。皆さんの無駄な時間を減らし、生産性をより高めていただきたいと考えています。このガイドは、Unity 初心者でも経験豊富な開発者でも、ゲーム開発のあらゆる段階におけるワークフローの迅速化に役立ちます。

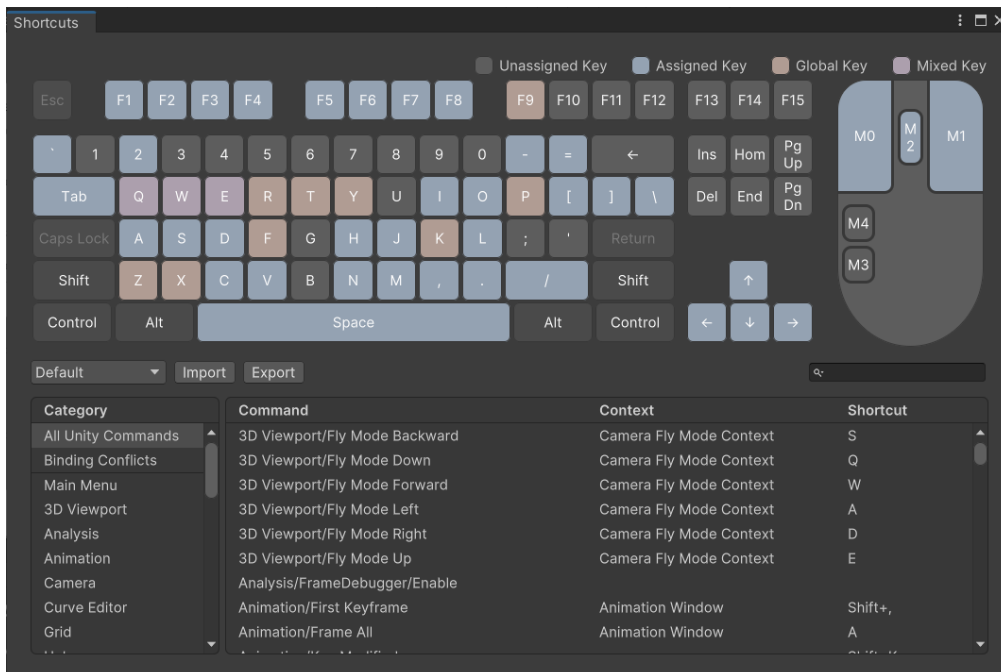
新機能の詳細については、[Unity 6 のウェブページ](#)、または Unity ドキュメントの [Unity 6.0 の新機能](#) および [Unity 6.1 の新機能](#) のセクションを参照してください。

このガイドがお役に立てば幸いです。

エディターの ワークフロー

キーボードショートカットを簡単にカスタマイズする

ショートカットマネージャー は、エディターのホットキーの表示、カスタマイズ、管理に役立つ対話型の視覚的なインターフェースです。グローバル、シーンビュー、特定のエディターツールなど、さまざまなコンテキストにショートカットを割り当てたり、頻繁に使用する既存のバインディングを視覚化したりできます。また、チームやデバイス間でショートカットプロファイルをインポート/エクスポートすることもできます。



ショートカットマネージャー

ショートカットマネージャーには Unity のメインメニューからアクセスします。

- Windows と **Linux** では、**Edit > Shortcuts** を選択します。
- macOS では、**Unity > Shortcuts** を選択します。

よく使うショートカット

よく使うデフォルトのショートカットをいくつか紹介します。

アクション	Windows	Mac
選択オブジェクトにフォーカス	F	
アイテムの複製	Ctrl + D	⌘ + D
ゲームオブジェクトの削除	Shift + Del	⌘ + Del
ビュー/移動/回転/矩形/トランスフォーム	QWERT	
ピボットモードの切り替え	Z	
ピボット回転の切り替え	X	
頂点スナップ	V	
スナップ	Ctrl + 	⌘ + 
シーンビューで選択したオブジェクトの分離	Shift + H	
最大化/元に戻す	Shift + Space	
コンテキスト内でプレハブを編集	P	

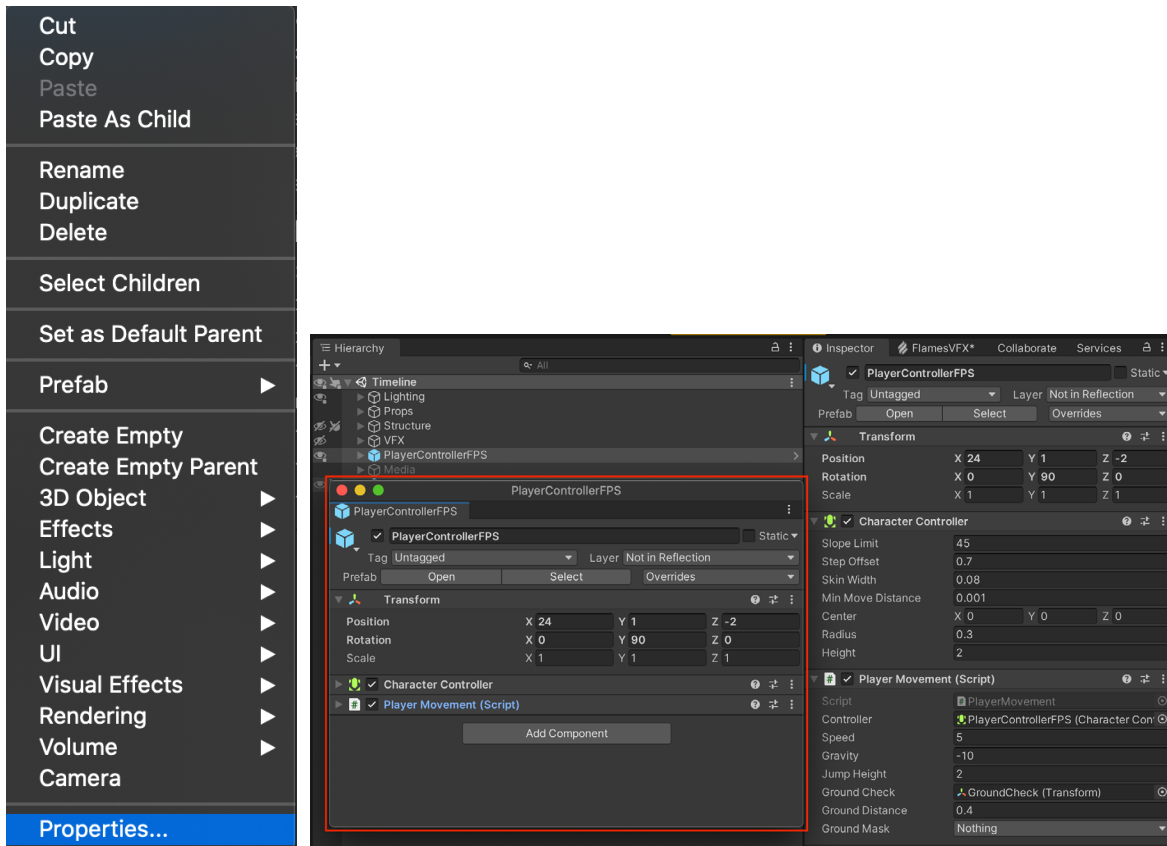
Command	Shortcut
View	Q
Move	W
Rotate	E
Scale	R
Rect	T
Transform	Y
Toggle Pivot Position	Z
Toggle Pivot Orientation	X

エディターでよく使うショートカット

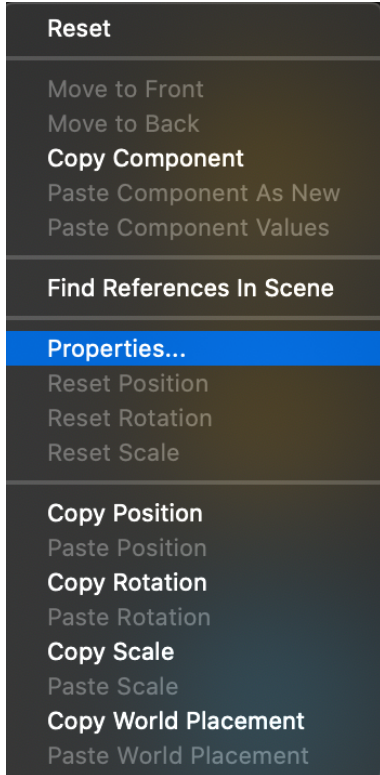
複数の Inspector で作業する

専用の **Inspector ウィンドウ** を使用すると、複数の Inspector ウィンドウを同時に表示して、特定のゲームオブジェクト、コンポーネント、アセットのプロパティを表示できます。シーン内で他のアイテムを選択しても、開いていたプロパティは常に表示されます。

ゲームオブジェクトまたはコンポーネントを右クリックし、**Properties** を選択します。これにより、他のウィンドウと同様に再配置、ドッキング、サイズ変更が可能なフローティング Inspector ウィンドウが表示されます。



専用 Inspector を使って 2 つのゲームオブジェクトを比較できます



複数の専用 Inspector を同時に開くことで、シーンに変更を加えながら複数のゲームオブジェクトを参照したり、並べて簡単に比較したりできます。

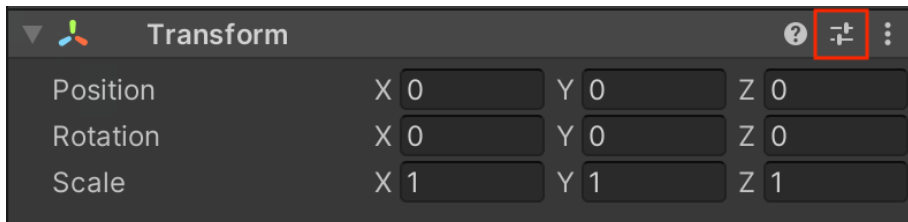
スクリーンスペースをあまり使わずに、ゲームオブジェクトの特定のコンポーネントに焦点を当てることもできます。

プリセットで好みのデフォルト設定を定義

プリセット を使用すると、Inspector でコンポーネントやアセットのデフォルト状態をカスタマイズして再利用できます。**プリセット** 作成時に、コンポーネントやアセットの現在の設定がキャプチャされ、**.preset アセット** として保存されます。

その後、この設定を同じ型の他のコンポーネントやアセットに適用できます。

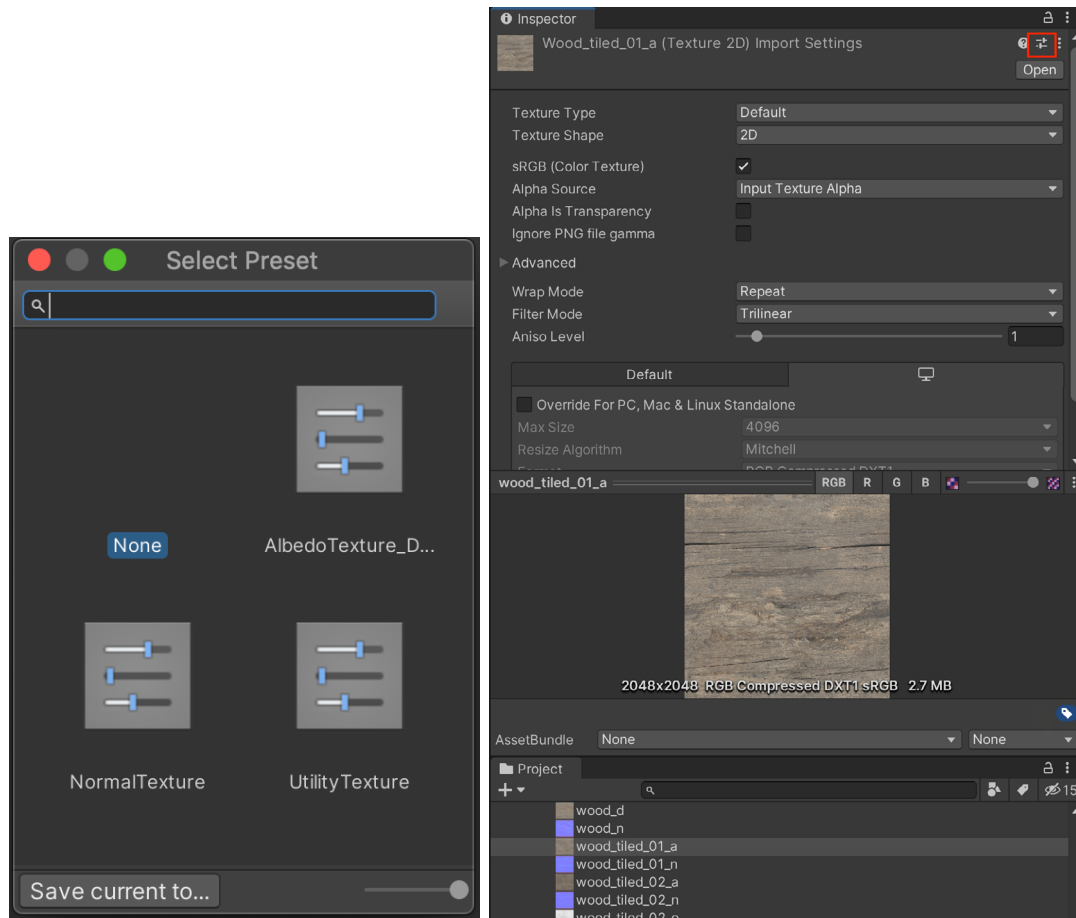
プリセットを使用して標準設定を適用したり、新しいアセットに適切なデフォルトを設定したりできます。これにより、チーム全体で一貫した基準が確保されるため、設定を見落としたせいでプロジェクトのパフォーマンスに影響が生じることがなくなります。



プリセットアイコンは、ここでは赤でハイライトされています。

プリセットを作成するには:

1. コンポーネントの右上にある **プリセット** アイコンをクリックします。
2. **Save current to...** をクリックし、プリセットをアセットとして保存します。
3. 利用可能なプリセットのいずれかをクリックすると、値のセットをロードできます。



この例では、プリセットには用途 (アルベド、法線、ユーティリティ) に応じて異なる 2D テクスチャのインポート設定があります。

プリセットのその他の実用的な使い方:

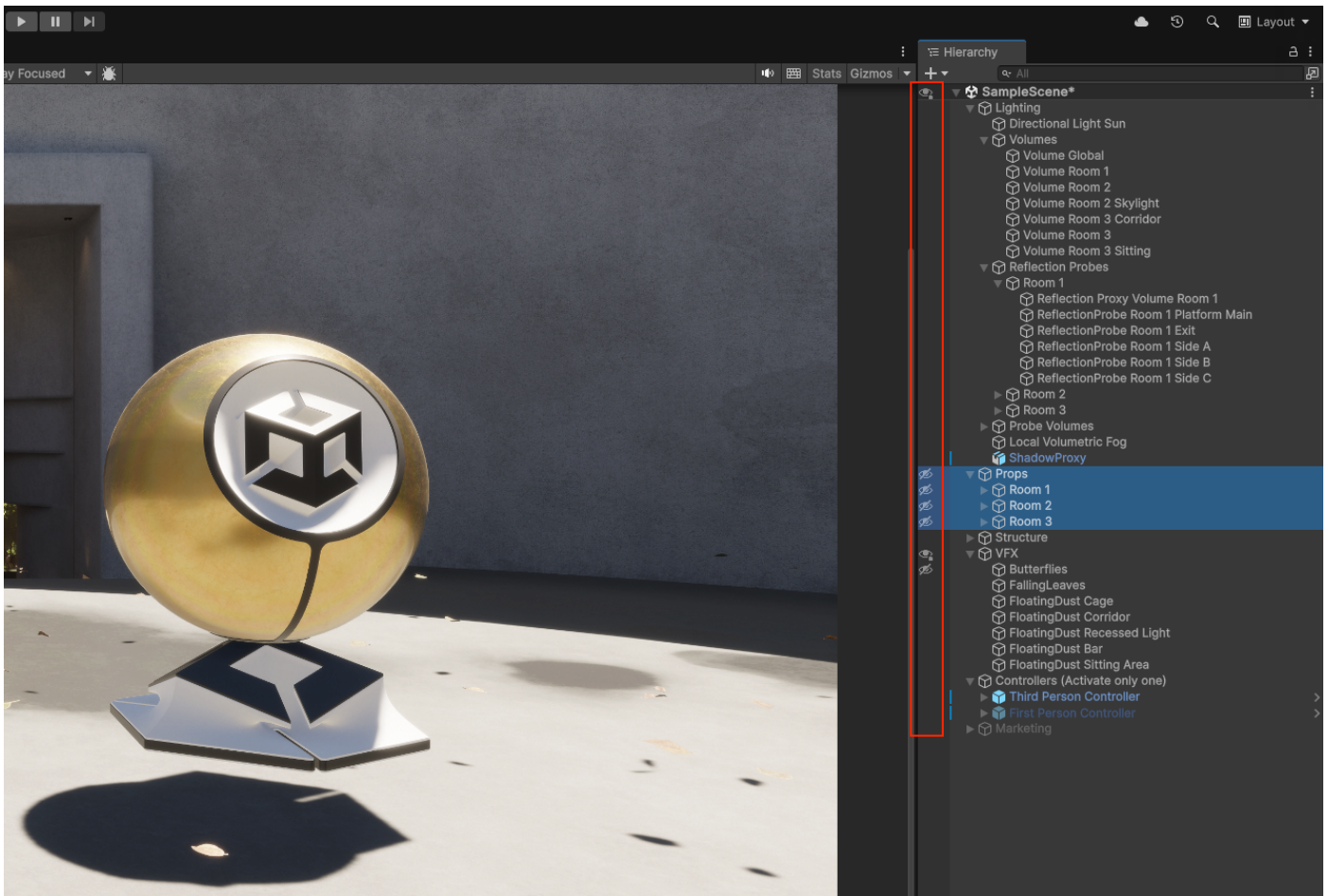
- **デフォルト値でゲームオブジェクトを作成する:** プリセットアセットを Hierarchy ウィンドウにドラッグアンドドロップすると、対応するコンポーネントがプリセット値で設定された新しいゲームオブジェクトが作成されます。
- **特定の型をプリセットに関連付ける:** **Preset Manager (Project Settings > Preset Manager)** で、型ごとに 1 つ以上のプリセットを指定できます。新しいコンポーネントを作成すると、指定したプリセットの値がデフォルトとして適用されます。
- **フィルターを使用して、複数のプリセットの中から選択する:** 同じ型に対して複数のプリセットを作成し、Preset Manager のフィルターを使用して、どのプリセットを適用するかを決定できます。

- **マネージャー設定を保存して再利用する:** マネージャーウィンドウにプリセットを使用し、設定を再利用できるようにします。例えば、同じ Tag and Layers、または Physics の設定を再適用する場合です。プリセットを使用することで、次のプロジェクトの設定時間を短縮できます。

シーンビューをシーンでの表示/非表示で整理する

シーンが大きくなった際に、特定のオブジェクトを一時的に非表示にして、ゲームオブジェクトの選択と編集を行いやすくなることができます。

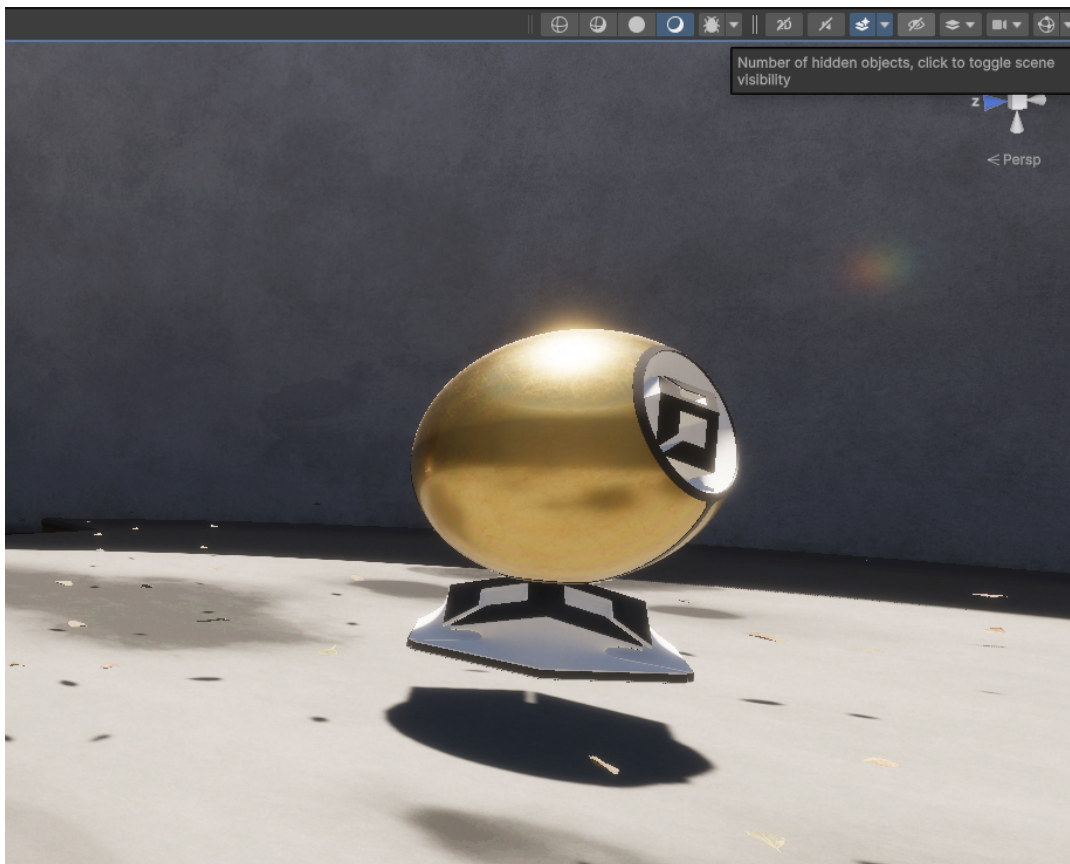
ゲームオブジェクトを非アクティブにする（その結果として意図しない動作につながる可能性があります）代わりに、[シーンでの表示/非表示](#) 制御を利用してシーンビューの可視性を切り替えることができます。**Hierarchy** ウィンドウでゲームオブジェクトの横にある目のアイコンをクリックして、シーンでの表示/非表示を切り替えます。これにより、アクティブ状態やゲーム内の可視性に影響を与えることなく、オブジェクトをシーンビューで非表示にできます。



シーンでの表示/非表示制御を使用して、シーンビューのオブジェクトを非表示にします。

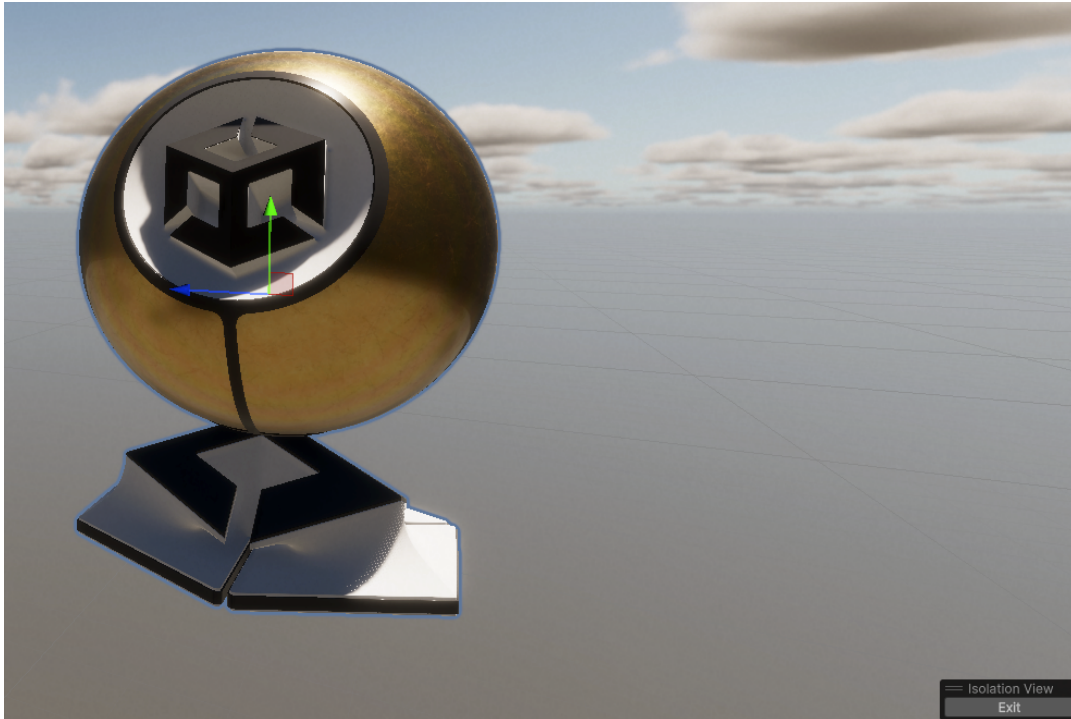
なお、Hierarchy のステータスアイコンは、親オブジェクトや子オブジェクトの表示状態によって変わります。

アイコン	状態
	このゲームオブジェクトは表示されますが、一部の子オブジェクトは非表示です。
	このゲームオブジェクトは非表示ですが、一部の子オブジェクトは表示されます。
	このゲームオブジェクトとその子オブジェクトは表示されます。ただし、子オブジェクトはゲームオブジェクトにカーソルを合わせたときにのみ表示されます。
	このゲームオブジェクトとその子オブジェクトは非表示です。



シーンビューのコントロールバーのオン/オフを切り替えて、グローバルのシーンでの表示/非表示をオーバーライドします。

分離ビュー を使用して、特定のオブジェクトとその子オブジェクトにフォーカスすることもできます。Hierarchy ウィンドウでゲームオブジェクトを選択し、**Shift + H** キーを押してオンとオフを切り替えます。これは、終了するまでシーンでの表示/非表示の他の設定をオーバーライドします。



分離ビューを使用すると、気を散らすことなくゲームオブジェクトを編集できます。

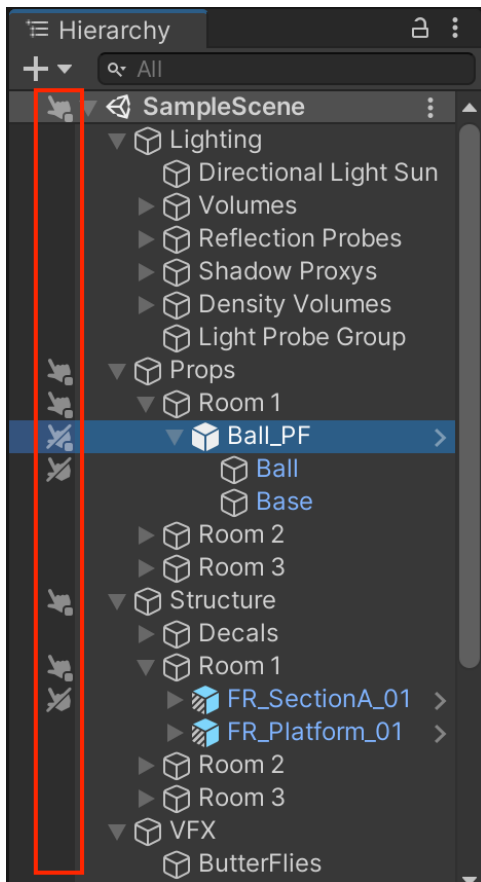
Shift + スペースキー のショートカットキーを使用すると、ビューポートを最大化し、エディターの他の部分を隠すことができます。

また、大規模なストリーミングワールドを作成する必要がある場合や、ランタイムに複数のシーンを効果的に管理したい場合、[Unity で複数のシーンを扱う](#) こともできます。ライトマップ、ナビメッシュデータ、オクルージョンカリングデータなどを [複数のシーンで同時にベイクできます](#)。また、エディター内やランタイムに [スクリプトを使用して、複数のシーンを編集](#) することもできます。

シーンビューで間違ったオブジェクトを選択しないようにする

Unity では、シーンでの表示/非表示を管理する方法と同様に、特定のゲームオブジェクトをシーンビューで選択できるかどうかを制御できます。

シーンでの表示/非表示ツールバーを使用してゲームオブジェクト選択の可/不可を切り替えることで、特定のゲームオブジェクトがシーンビューで選択されないようにできます。大規模で複雑なシーンで、意図しないゲームオブジェクトを選択して変更してしまうことを避けるために便利です。



Hierarchy の選択可/不可

ブランチ全体または単一のオブジェクトの選択可/不可を切り替えられるため、一部のゲームオブジェクトは選択できても、子や親は選択できない場合があります。状態を区別するアイコンは次のとおりです。

アイコン	状態
	このゲームオブジェクトは選択できますが、一部の子オブジェクトは選択できません。
	このゲームオブジェクトは選択できませんが、一部の子オブジェクトは選択できます。
	このゲームオブジェクトとその子オブジェクトを選択できます。子オブジェクトはゲームオブジェクトにカーソルを合わせたときにのみ表示されます。
	このゲームオブジェクトもその子オブジェクトも選択できません。

Inspector でアセットを直接プレビュー

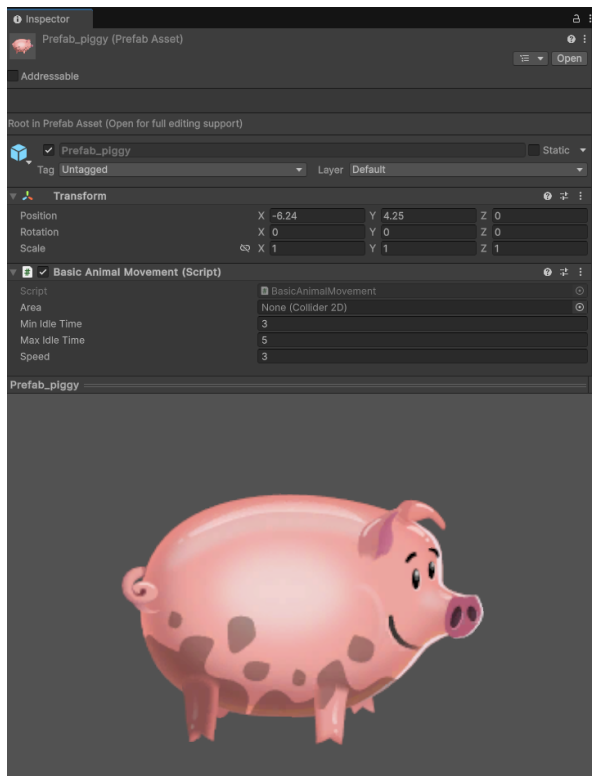
アセットプレビューは、Inspector ウィンドウの下部にある Unity エディターの視覚的なパネルです。3D モデル、マテリアル、テクスチャなどのアセットのプレビューやサムネイルを確認できます。エディターでの作業中に、迅速に、かつ対話形式でアセットを視覚的に評価できます。

モデルやプレハブなどの 3D アセットの場合、Inspector ウィンドウの下部にある Asset Preview をクリックしてドラッグすることで、オブジェクトを回転させ、さまざまな角度から確認することができます。これを使用すると、オブジェクトをシーンに加えることなく、アセットのジオメトリ、マテリアル、向きをすばやく確認できます。

アセットプレビューが小さすぎる場合は、Inspector ウィンドウのプレビュー領域の上にある分割線をドラッグしてサイズを変更できます。

アニメーション化されたモデルのようなアセットの場合、アセットに単純なデフォルトアニメーションが設定されていれば、アセットプレビューに表示されます。モーションとリギングをすばやく評価するために使用します。

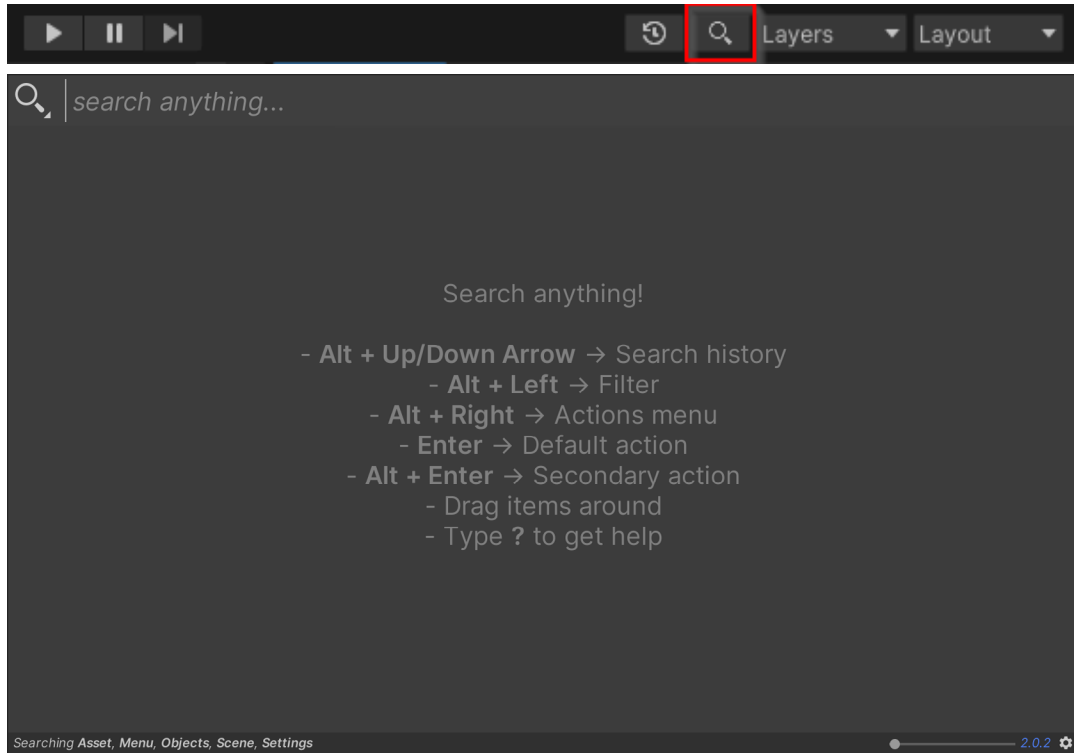
Project ウィンドウにサムネイル (アセットプレビューの縮小版) を直接表示することもできます。これを有効にするには、**Project** ウィンドウを **Two Column Layout** (2 列レイアウト) に切り替え、下部のプレビュースライダーを調整します。これにより、アセットを視覚的に閲覧したり整理したりしやすくなります。



『Happy Harvest』サンプルのアセットプレビューウィンドウ

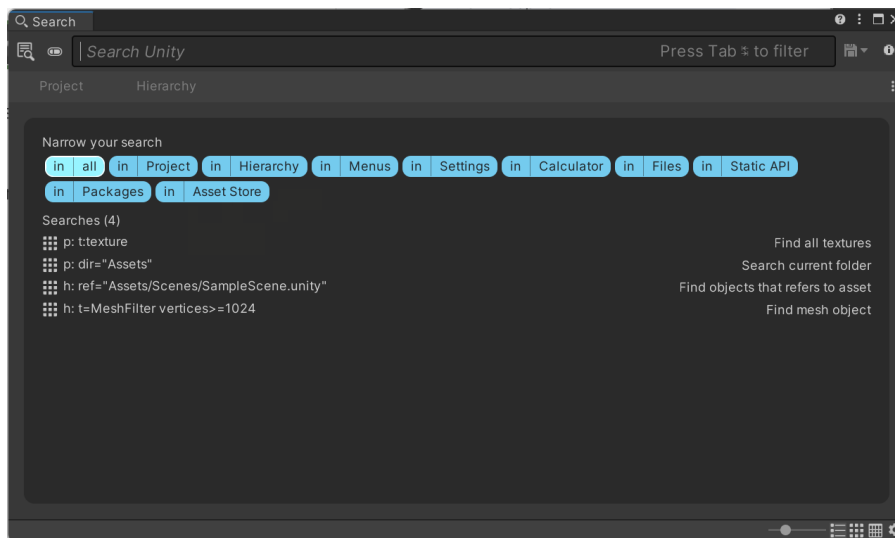
プロジェクト内の検索を迅速化

Unity でアセット、シーンオブジェクト、メニューアイテム、パッケージ、API、設定などを効率的に検索する方法はいくつかあります。Hierarchy ウィンドウと Project ウィンドウの検索機能に加えて、メインメニューバーの検索ボタンアイコンや、Windows では Ctrl + K、macOS では Cmd + K のホットキーを使用して呼び出すこともできます。

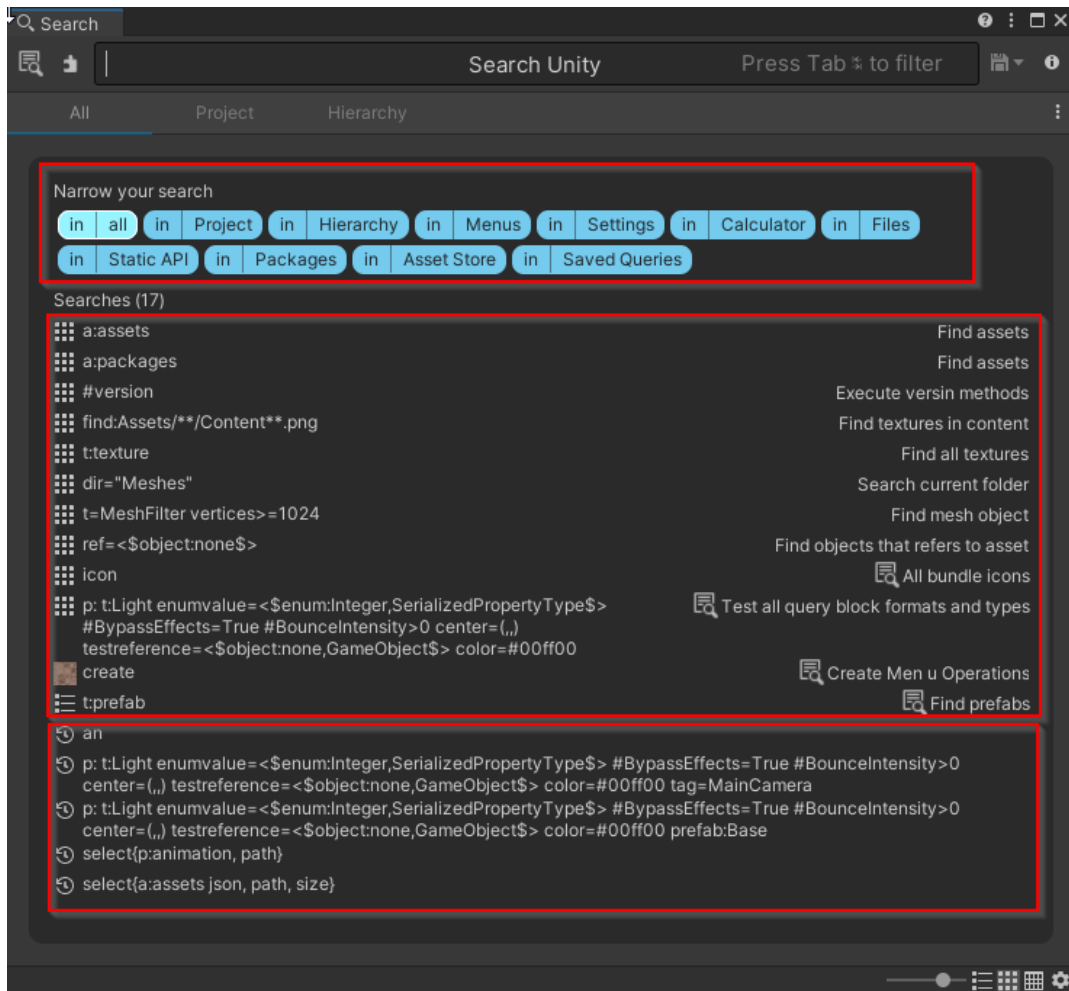


QuickSearch を起動するには、ホットキーまたはヘルプメニューを使用します。

検索ウィンドウが開き、検索で簡単にフィルターをかけられます。



Search ウィンドウ



中央のセクションには、現在選択されている検索領域で利用可能なクエリの一覧が表示されます。任意のクエリをクリックして実行できます。名前を検索するときは、種類で検索することもできます。ドロップダウンの **Types** (種類) から選択するか、短縮構文の **t:** を使用します。短縮構文は例えば、t:scene (すべてのシーンを検索) や t:texture (すべてのテクスチャを検索) となります。

さらに、Search ウィンドウでは、さまざまな方法でオブジェクトを視覚化できます。コンパクトなリストビュー、大きなリストビュー、複数サイズのグリッドアイコンがあります。オブジェクトを表形式で表示することもできます。

Unity 内外での検索の詳細については、[QuickSearch ガイド](#) を参照してください。

クエリビルダーを使用してカスタム検索フィルターを作成する

複雑な検索クエリを扱っていて、プロジェクトを探索する際にさらにサポートが必要な場合は、Query Builder ワークフローを使用してください。ビルダーのトグル (検索フィールドの横のパズルボタン) で有効にできます。

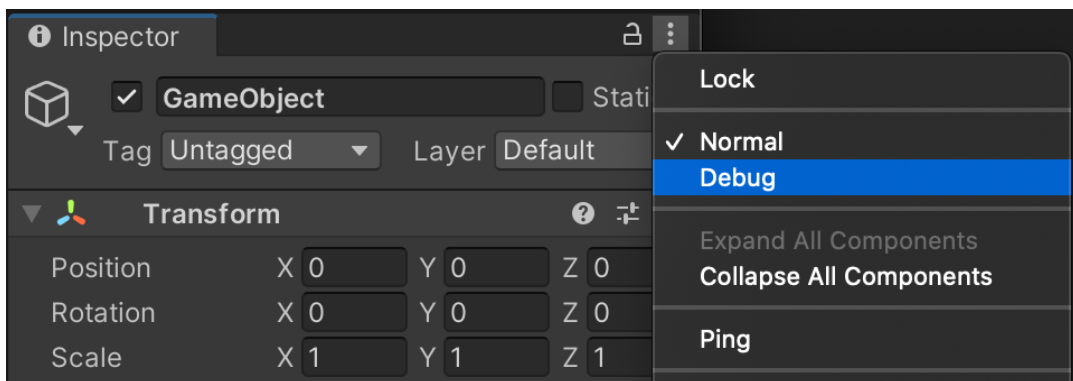


Query Builder (クエリビルダー)

よく使うタスクのカスタムクエリを作成して保存することで、時間を節約できます。例えば、特定のタグでプレハブを見つけたり、使用されていないアセットを見つけたりすることが多い場合は、再利用可能なクエリを定義してこのプロセスを自動化します。クエリビルダーもエディターに統合され、パッケージではなくなりました。これにより、開発チームのワークフローを効率化できます。

Inspector でデバッグデータを表示する

各ゲームオブジェクトの Inspector を Normal (ノーマル) モードと Debug (デバッグ) モードの間で切り替えることができます。**その他** (:) ボタンをクリックしてコンテキストメニューを開き、適切なモードを選択します。デバッグモードを使用すると、非公開のフィールド、非表示の参照、Unity の内部データなど、シリアル化されたすべてのフィールドが Inspector に表示されます。これは、コンポーネントの標準的な公開インターフェースを超えたトラブルシューティングや検査に役立ちます。

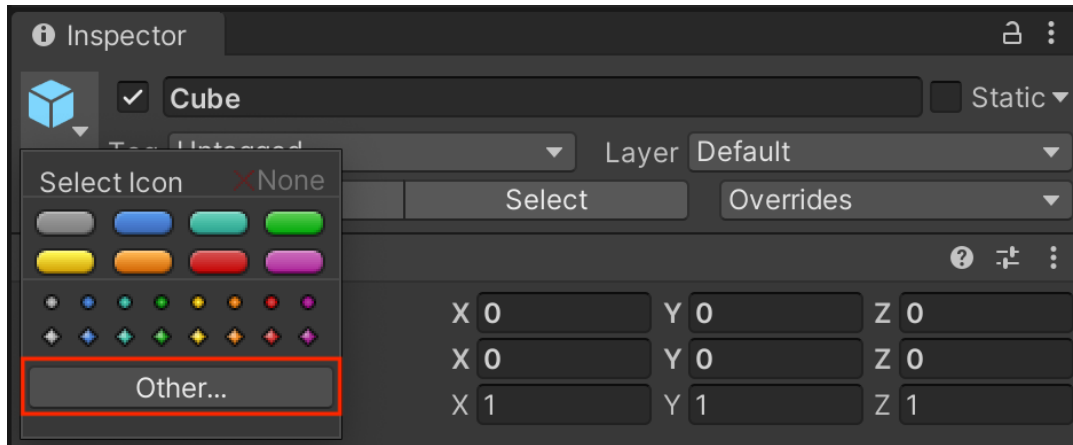


Inspector の Debug (デバッグ) モード

視覚的なデバッグにカスタムギズモとアイコンを使用する

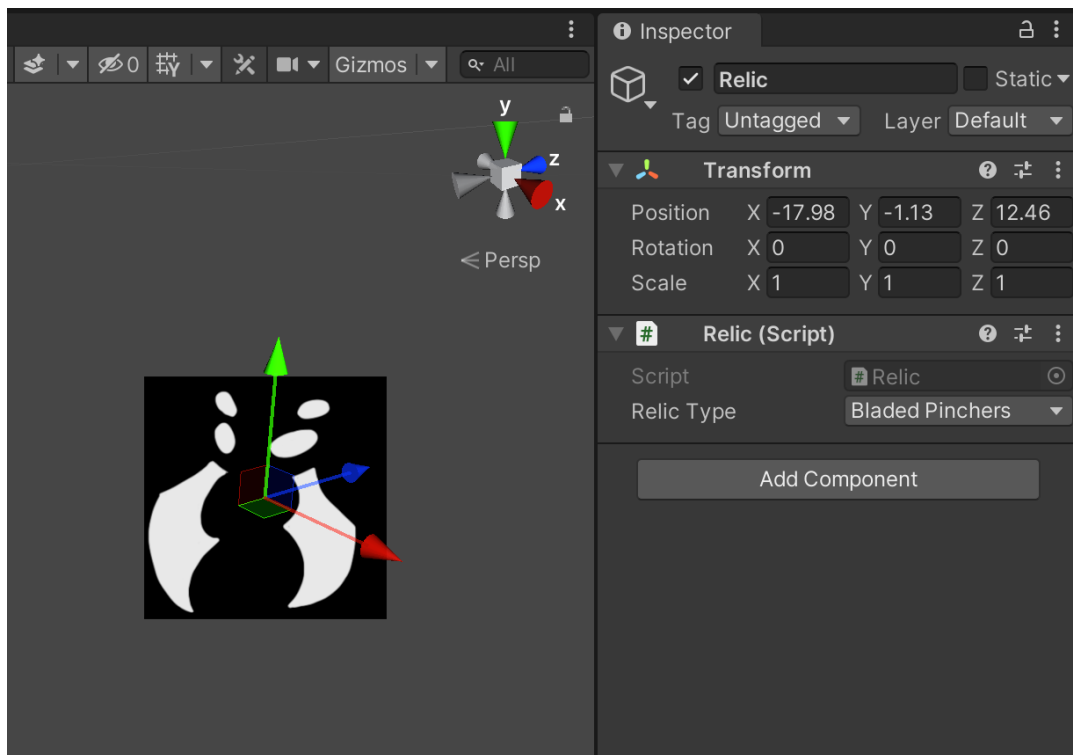
ギズモはシーンビューに表示される視覚的なオーバーレイで、開発中にゲームオブジェクトを見つけて識別するために役立ちます。トリガー、スポーンポイント、スクリプトなど、レンダリングされていない要素を視覚化する際に特に便利です。

Select Icon (アイコン選択) メニューを使用して、ゲームオブジェクトのアイコンをカスタマイズできます。**Other** (その他) を選択して独自のアイコンを定義します。



Inspector のドロップダウンを使用してギズモを切り替えます。

スクリプトでギズモを作成し、対話型にすることもできます。例えば、ギズモはカスタムコンポーネントの大きさや影響範囲を定義するために役立ちます。



この例では、選択に基づいてスクリプトがギズモを変更しています。

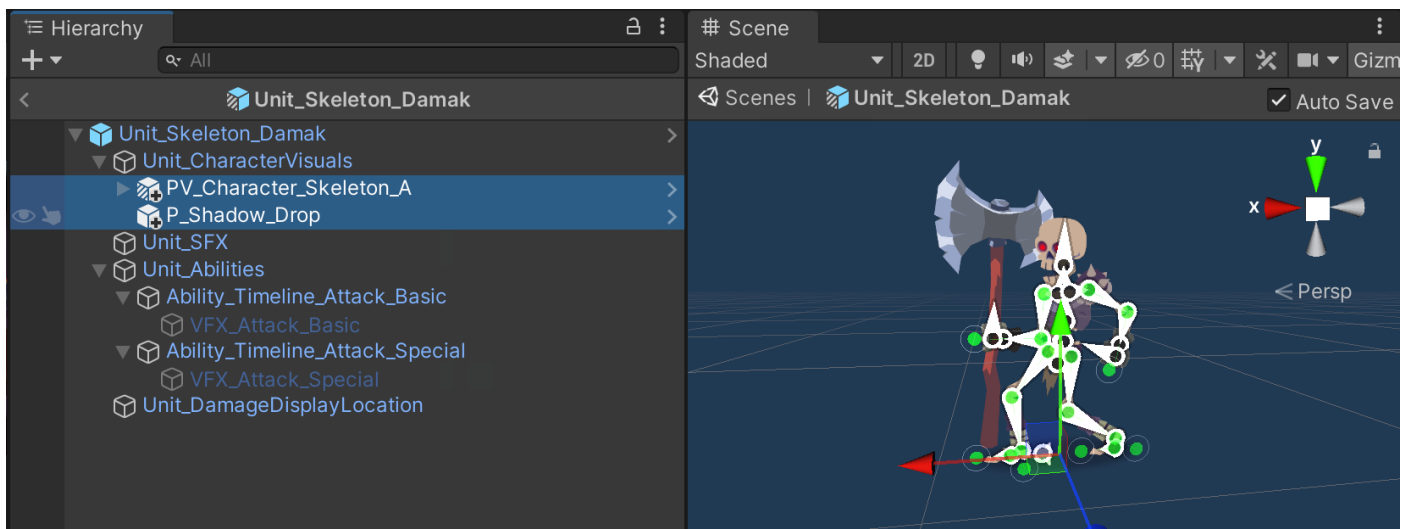
シーンコントロールバーの **ギズモダイアログ** を使用して、特定のギズモを切り替えたり、すべてのギズモの有効/無効を一括して切り替えたりできます。

使用例については、[Creating Custom Gizmos for Development](#) (開発用カスタムギズモの作成) を参照してください。また、[ギズモ](#) と [ハンドル](#) の API も確認してください。

ネスト状のプレハブとプレハブバリエーションのワークフローを活用する

プレハブを使用すると、設定が完了したゲームオブジェクトを保存して再利用し、シーンを柔軟かつ効率的に構築できます。他にもできることがあります。プレハブをネストしたり、プレハブバリエーションを作成したり、プレハブを使用してランタイムにゲームオブジェクトをインスタンス化したりできるのです。

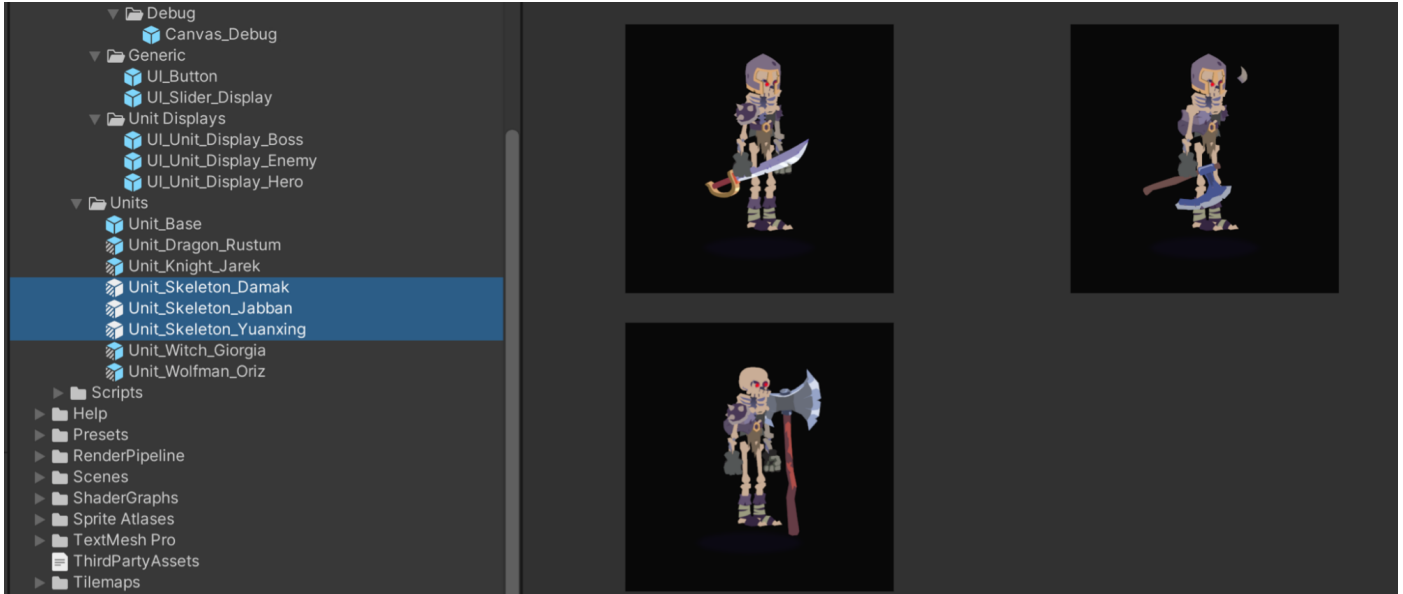
ネスト状のプレハブ を使用すると、プレハブ同士を親子関係にできます。これで、部屋や家具といった小さなプレハブで構成された、建物などの大きなプレハブを作成できます。アーティストや開発者が複数いるチームでアセットの開発を分担して、それぞれがコンテンツの別の部分を同時に作業できるようになり、効率が向上します。



『Dragon Crashers』サンプルプロジェクト内のネスト状のプレハブの例

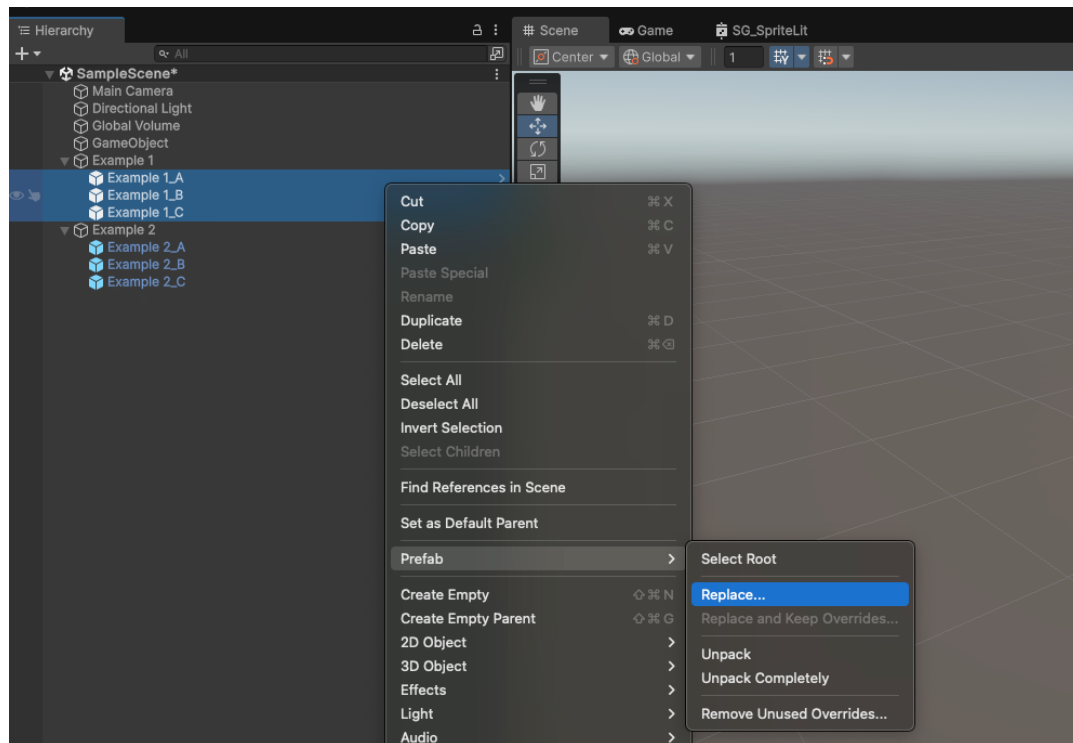
プレハブバリエーション を使用すると、オブジェクト指向プログラミングにおける継承のように、プレハブから別のプレハブを派生させることができます。バリエーションを変更するには、特定の部分をオーバーライドするだけです。オリジナルに影響を与える心配は無用です。変更をすべて取り消し、ベースのプレハブに戻すことも、いつでもできます。

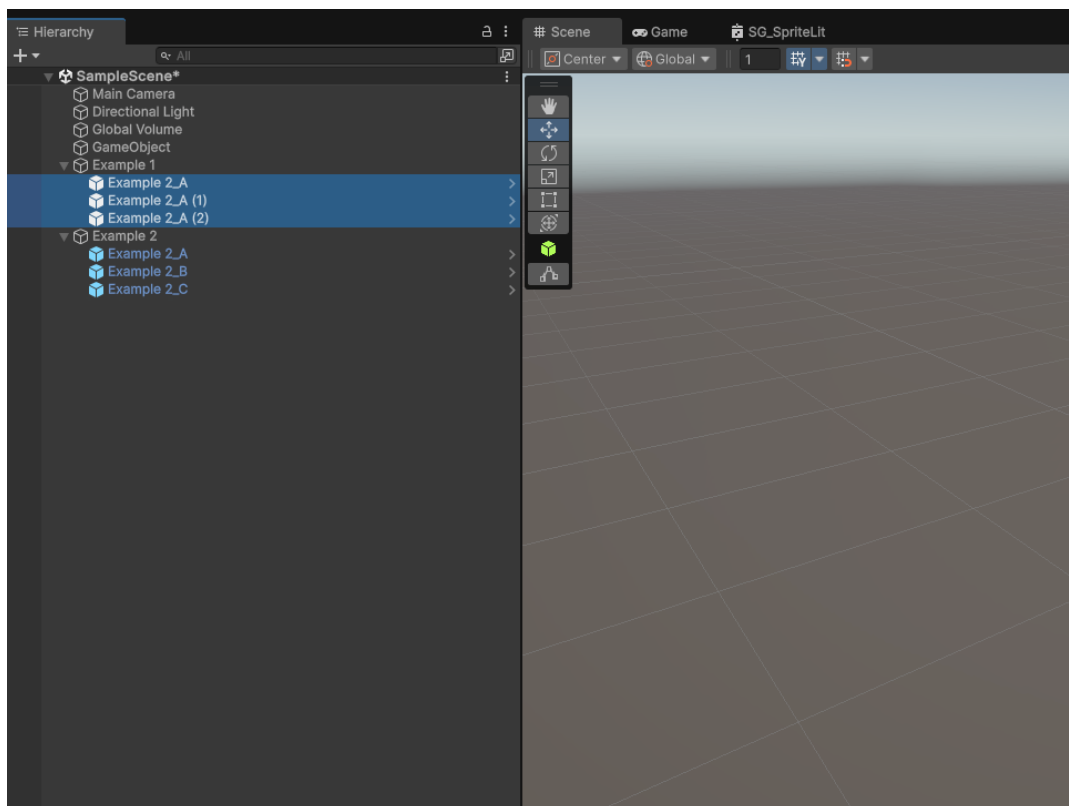
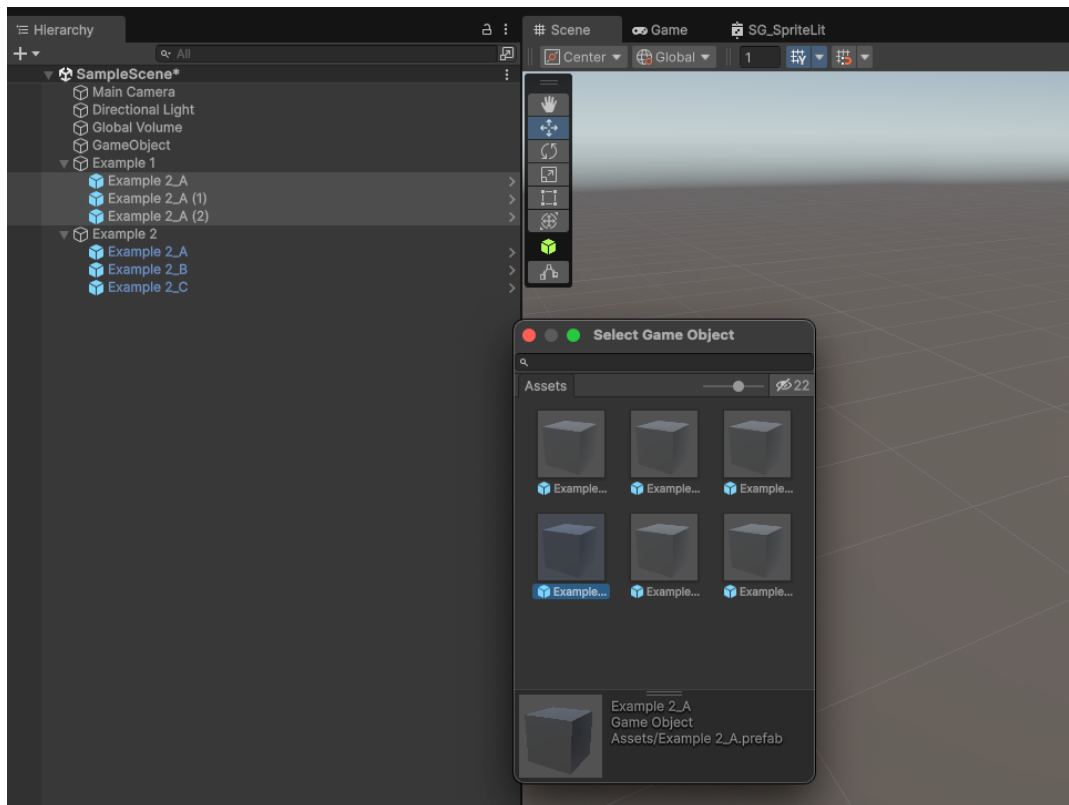
また、すべてのバリエーションを一度に変更したい場合は、ベースのプレハブ自体を直接変更します。



『Dragon Crashers』サンプルプロジェクトのプレハブバリエーション

シーン内の 1 つまたは複数のプレハブを別のプレハブに置き換えるには、Ctrl (Windows) または Cmd (macOS) を押しながら、新しいプレハブを古いプレハブにドラッグします。

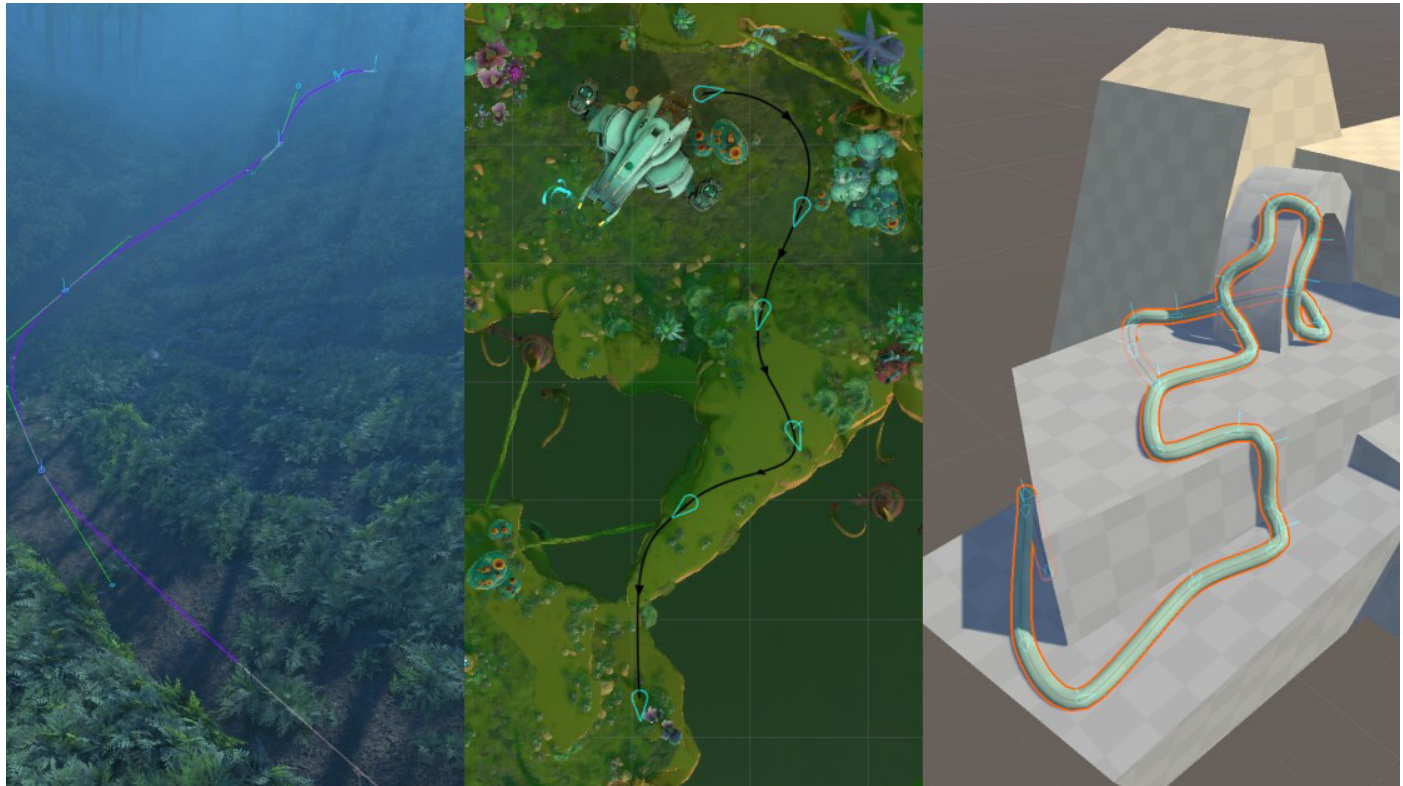




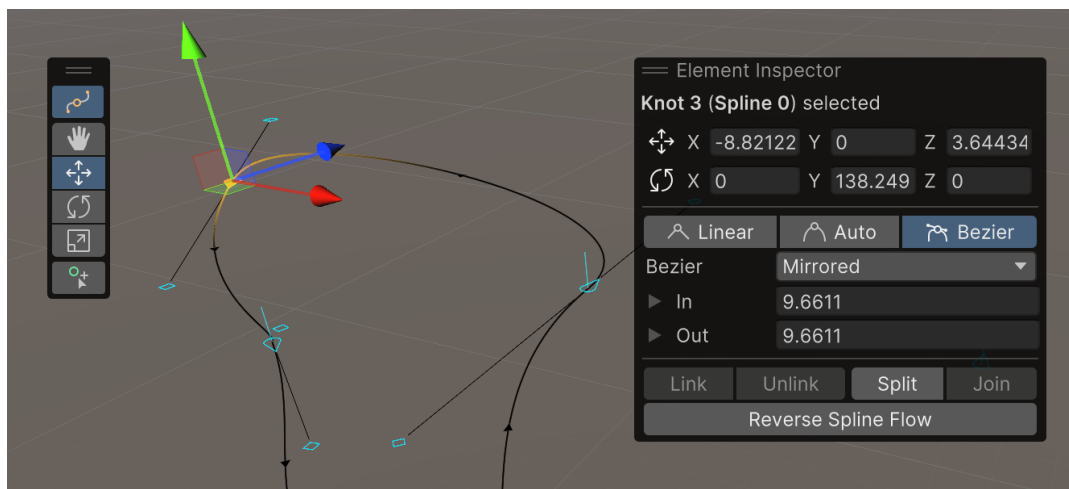
Hierarchy ウィンドウでのプレハブの置き換え

Splines パッケージを使用してカーブパスを簡単に作成する

Splines パッケージ を使用すると、エディター内で直接、スプラインベースのパスの作成と編集ができます。スプラインは、道路、川、レール、カメラトラックなどの要素や、パースペクティブのビジュアル要素やゲームプレイ要素を実現できる、滑らかで曲線的なパスを定義するために役立ちます。



スプラインの使用事例 (左から順に): 森を通る道、アニメーションパス、チューブメッシュ。Package Manager の Splines ページで、インストール後に実現できるすべてのことを紹介した他のサンプルを確認できます。



Unity におけるスプラインのハンドルとコントロールは、一般的な DCC アプリケーションのベクターまたは 3D 描画ツールに類似しています。

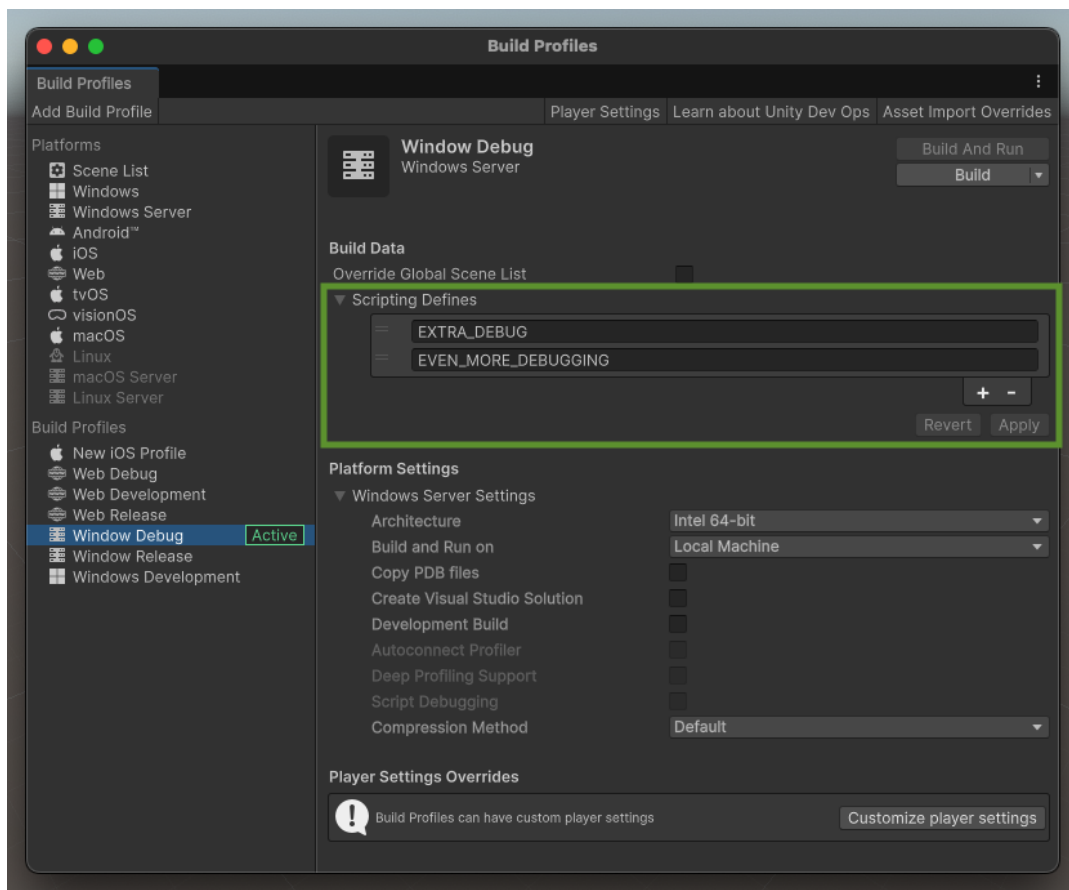
スプラインを作成すると、プログラマーとアーティストの双方がさまざまな方法で使用できます。例えば、プログラマーはスプラインのポイントを読み取り、[API](#) を使用してゲームロジックに使用できます。

『[How to get started with the splines package \(Splines パッケージで始めよう\)](#)』動画チュートリアルでは、Splines パッケージを使用してスプラインの一般的な使用事例を実現する方法、ゲームオブジェクトの位置と回転をスプラインに沿ってアニメーション化する方法、スプラインに沿ってプレハブをインスタンス化して環境を作成する方法を紹介しています。

Build Profiles でビルドをカスタマイズ

Build Profile ワークフローは、Unity 6 のビルドを設定する新しい方法です。この新しいワークフローは、任意のプラットフォームに対して複数の構成をサポートし、それぞれに異なる設定をすることができます。

Unity 6 でカスタムのビルドプロファイルを作成し、特定のプラットフォームやニーズ（デバッグ、テスト、本番環境など）に合わせてさまざまなビルド設定を効率的に構成できます。これにより、ワークフローを最適化するビルド設定をすばやく切り替えることができます。



Build Profiles 内の Scripting Defines と Player Settings Overrides を使用して、デバッグツールの有効化やスプラッシュイメージのカスタマイズなど、ビルド固有の設定を行い、バージョン管理システムを通じて簡単に共有できます。

その他のリソース:

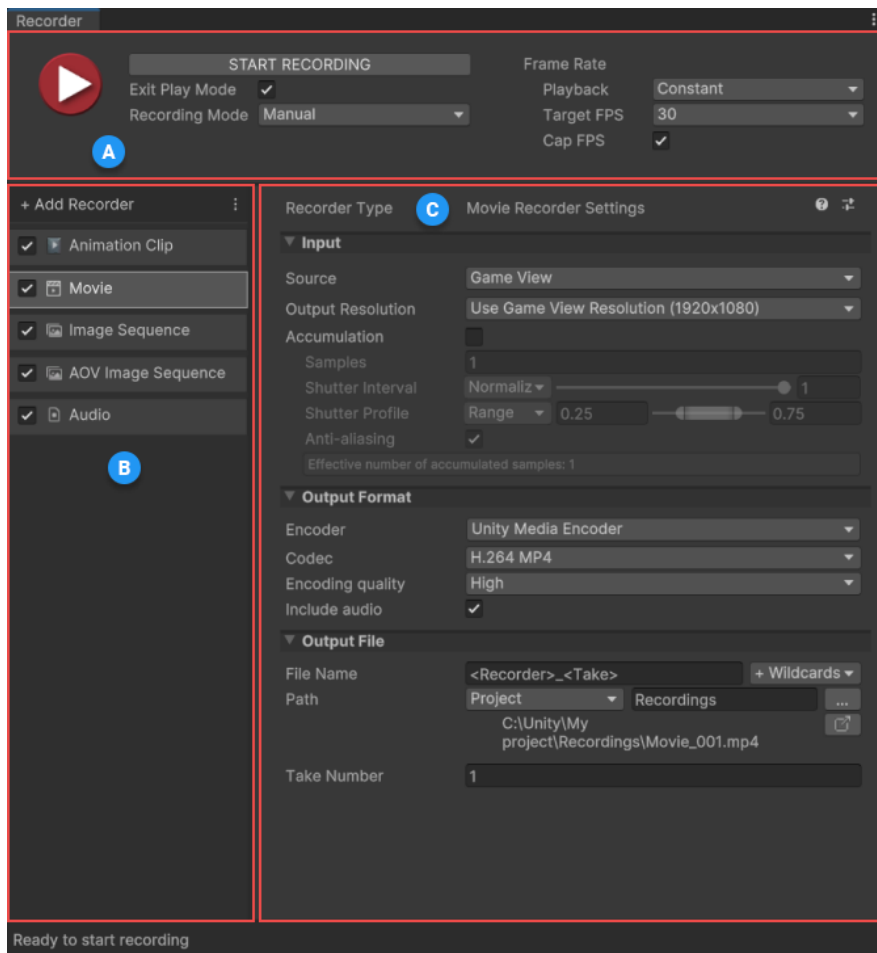
- [What you need to know about Build Profiles in Unity 6](#)
- [Everything you need to know about Build Profiles in Unity 6 | Unite 2024](#) (Unity 6 のビルドプロファイルについて知っておくべきことすべて)

Recorder パッケージで滑らかなゲーム映像をキャプチャする

[Unity Recorder パッケージ](#) を使用すると、再生モード中にプロジェクトのデータをキャプチャしてエクスポートできます。これは、ゲームプレイ、映像作品、その他のランタイムシーケンスをビデオファイル、画像シーケンス、アニメーションデータとして記録するのに便利です。

Recorder ウィンドウは、Unity エディターのメインメニューの **Window** にあります。**Recorder** ウィンドウを開くと、前回の録画セッションの値が復元されるので、すぐに作業の再開や調整を行えます。

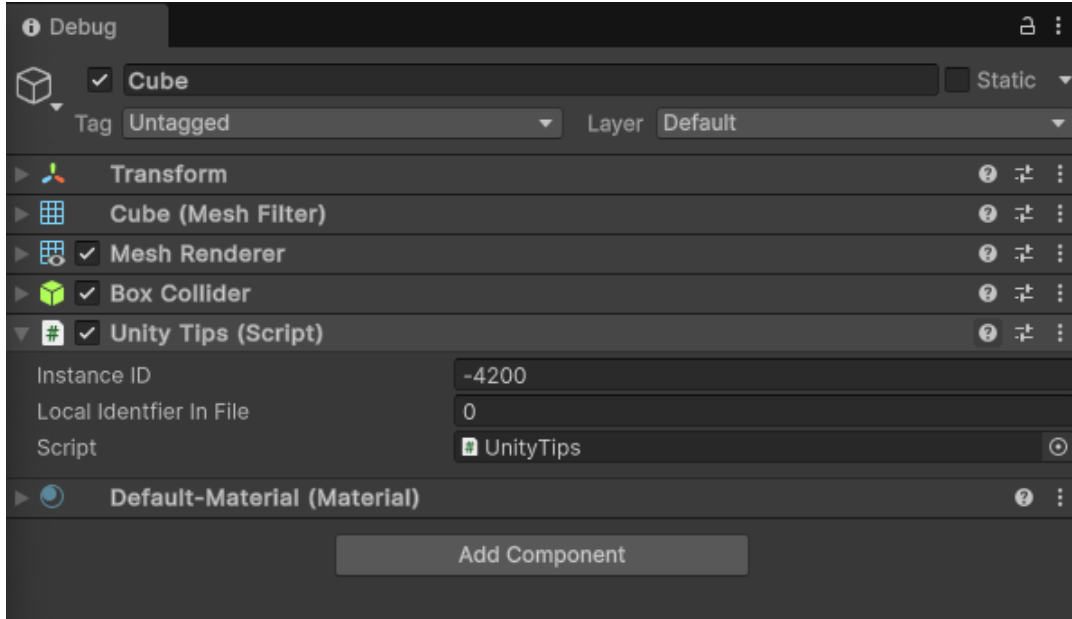
[透明な背景の画像や映像をキャプチャ](#) できます。Unity Recorder は、アルファチャンネルをキャプチャし、透明度を出力できます。ただし、使用しているレンダーパイプラインと Recorder の設定に依存する特定の条件下でのみ可能です。



Unity の Recorder ウィンドウ

エディターワークフローのヒント

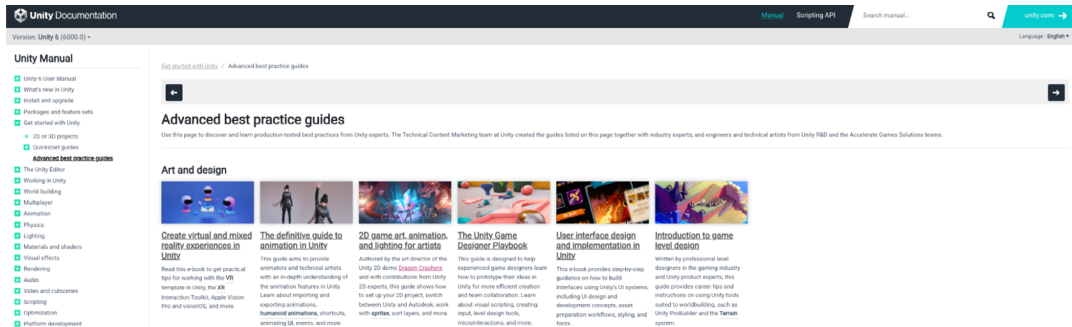
1. **[HelpURL]** 属性を使用して、**Inspector** の疑問符アイコンをドキュメントやその他のリソースにリンクできます。Inspector で疑問符アイコンをクリックすると、指定された URL がユーザーのデフォルトのブラウザで開きます。



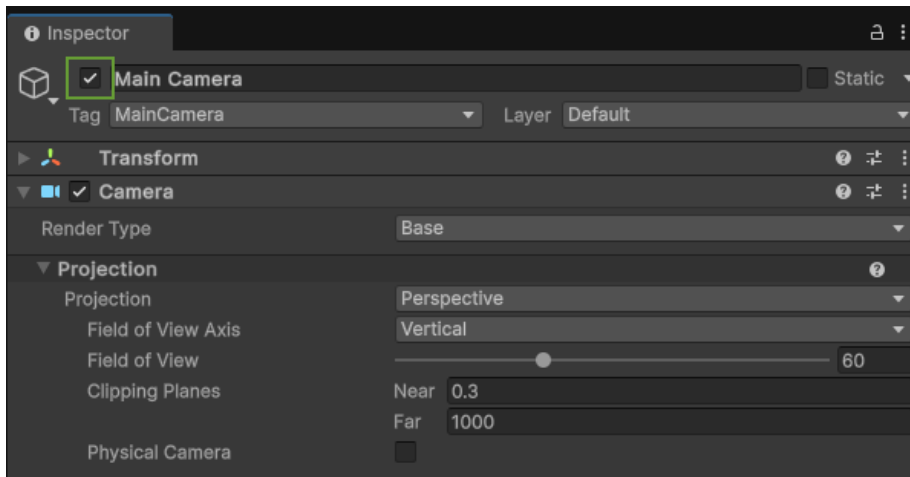
```

1  using UnityEngine;
2
3  [HelpURL ("https://docs.unity3d.com/Manual/best-practice-guides.html")]
4  public class UnityTips : MonoBehaviour
5  {
6      // Start is called once before the first execution of Update after the MonoBehaviour is created
7      void Start()
8      {
9      }
10
11     // Update is called once per frame
12     void Update()
13     {
14     }
15 }
16
17

```

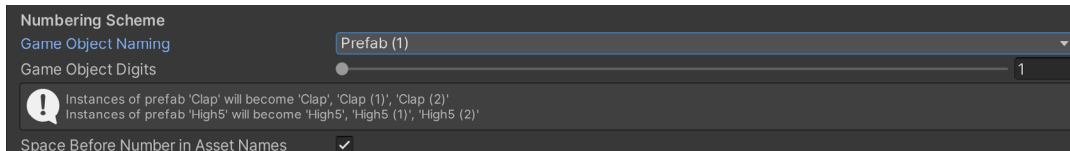


- 数回のマウスクリックを毎日節約できます。**Shift + Alt + A** (Windows) または **Shift + Option + A** (macOS) を押すと、現在選択中のゲームオブジェクトのアクティブ/非アクティブが切り替わります。



選択したゲームオブジェクトをアクティブまたは非アクティブにします。

- ゲームオブジェクトの番号付けをカスタマイズできます。**Project Settings** の **Editor** タブにあります。ここで、命名オプションと、インスタンス番号の桁数、間隔の有無を選択します。



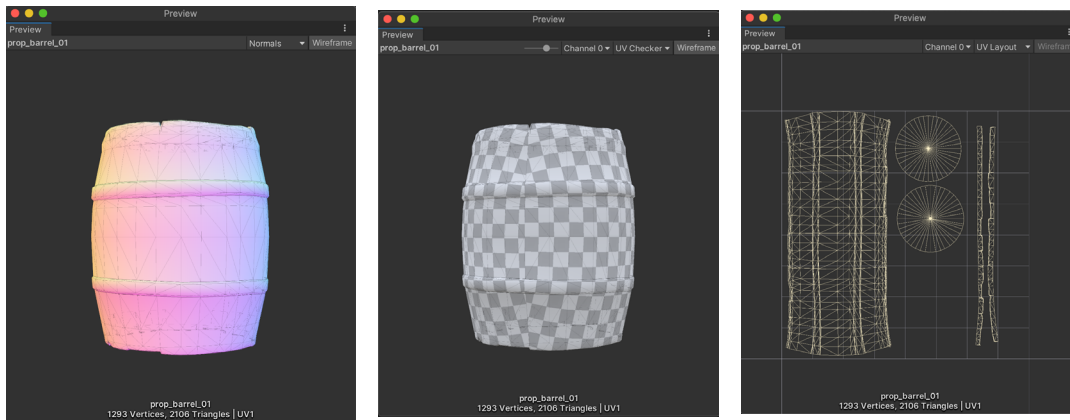
Numbering Scheme (番号付け規則)

- Hierarchy** ウィンドウでゲームオブジェクトをカットアンドペーストできます。コンテキストメニューから **Paste As Child** (子として貼り付け) を行うこともできます。



Paste As Child (子として貼り付け)

5. 選択したオブジェクトにシーンビューでフォーカスするには、ショートカット **F** を使用します。再生モードで **Shift + F** を押すと、選択した移動中のゲームオブジェクトにロックされます。
6. UV、法線、接線、その他のメッシュ情報を Inspector のプレビューに表示できます。



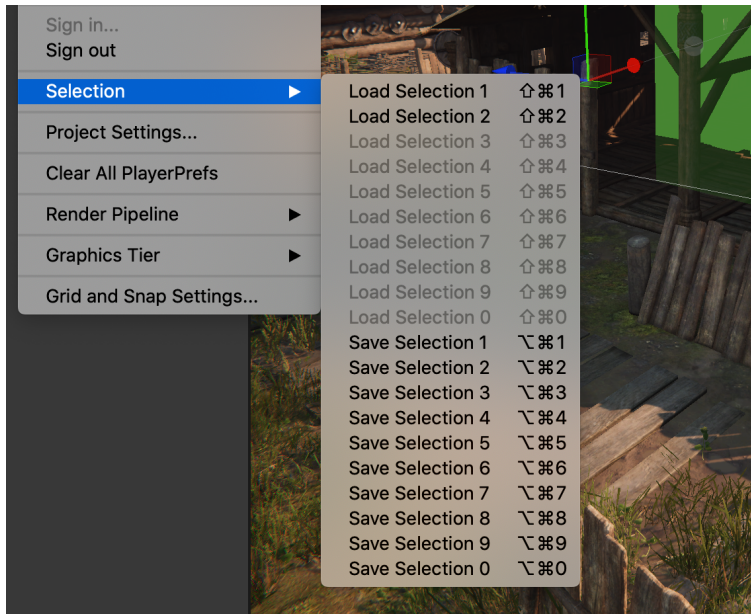
Inspector プレビュー

7. Layers メニューを使用して、シーンビューをわかりにくくするおそれのあるレイヤー (UI など) の表示をオフにできます。状態を誤って変更しないよう、レイヤーをロックできます。



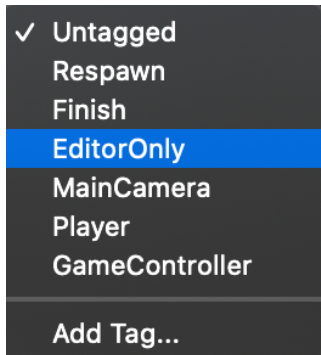
レイヤーの切り替えと編集

8. シーン内で同じオブジェクトを頻繁に選択する場合は、**Edit > Selection** で示されるホットキーで、すばやく保存したりロードしたりできます。



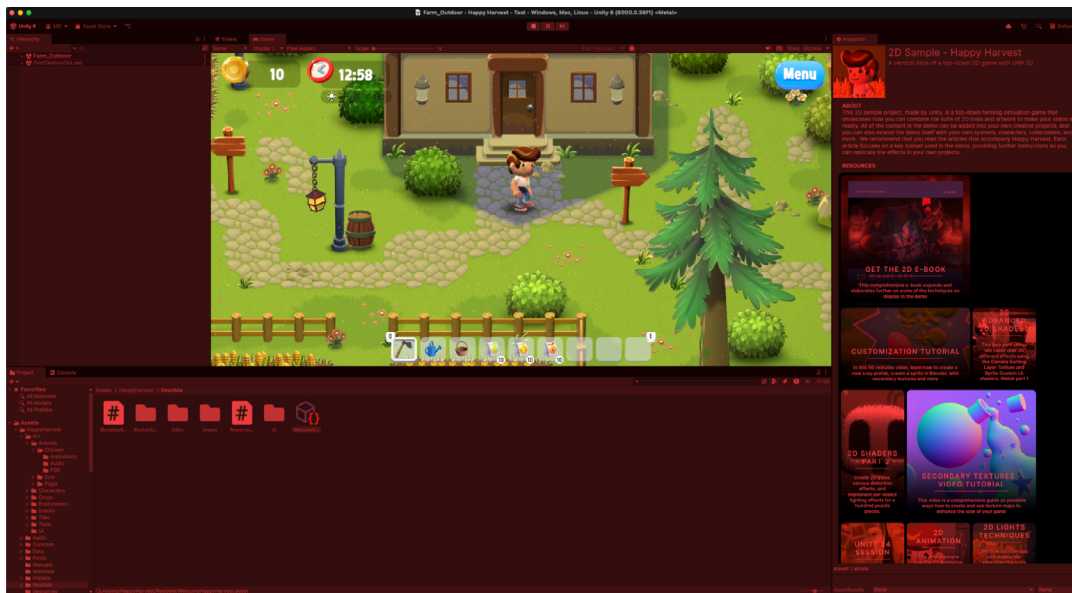
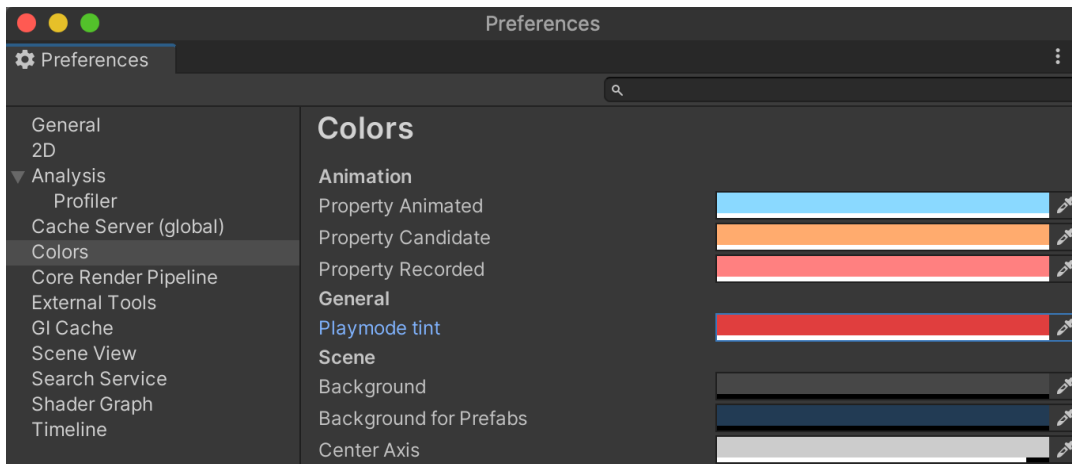
選択対象の保存 (Save) とロード (Load)

9. **EditorOnly** タグで、アプリケーションのビルドに表示されないゲームオブジェクトを指定できます。



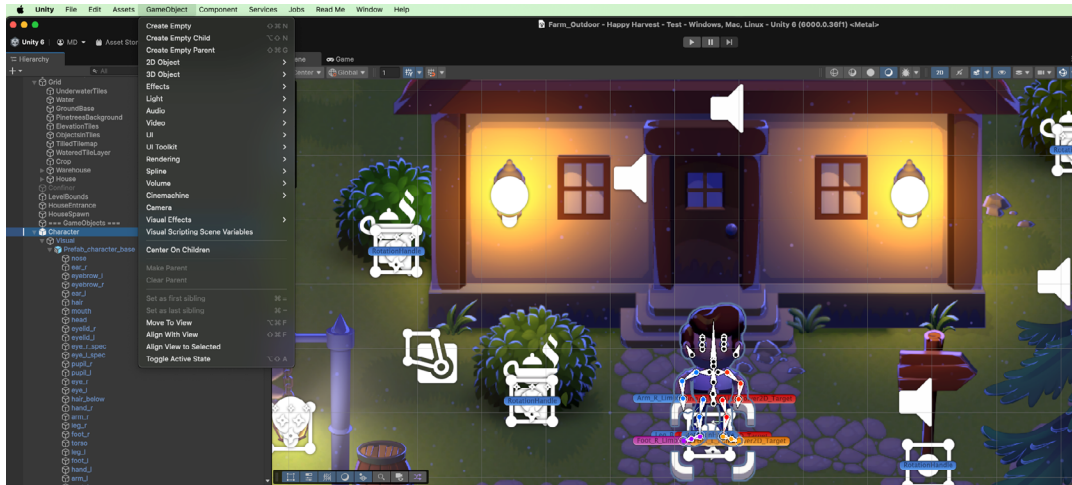
EditorOnly タグ

10. **Unity > Preferences > Colors** でエディターの色を変更して、特定の UI 要素やオブジェクトをより迅速に見つけられるようにできます。例えば、再生モードになっていることがわかるよう再生モードの色合いを調整し、終了時に変更を保存し忘れないような工夫が考えられます。



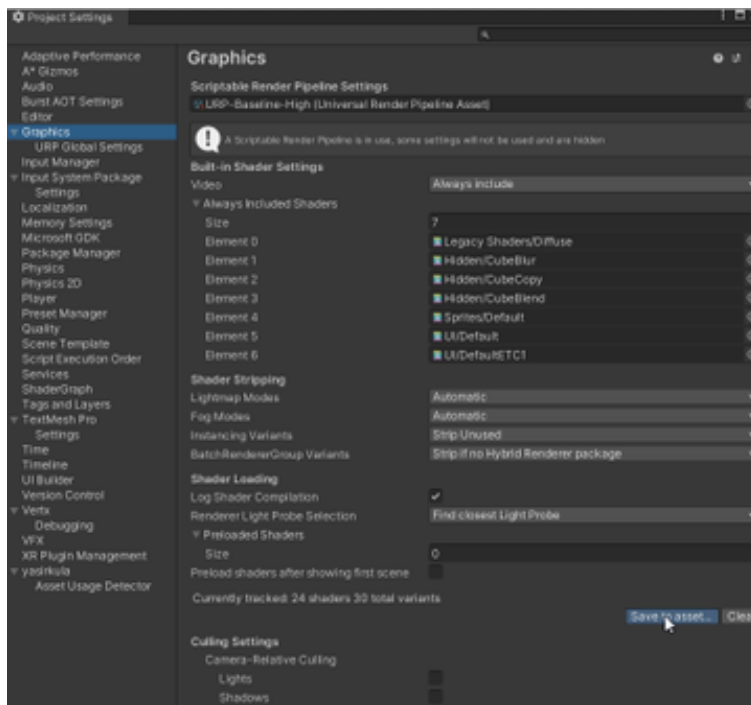
Playmode tint (再生モードの色合い)

11. カメラを設定する際、**GameObject > Align With View** で、ゲームカメラをシーンカメラに合わせることができます。また逆に、**Align View to Selected** で、シーンカメラを Hierarchy 内の別のカメラに合わせることができます。



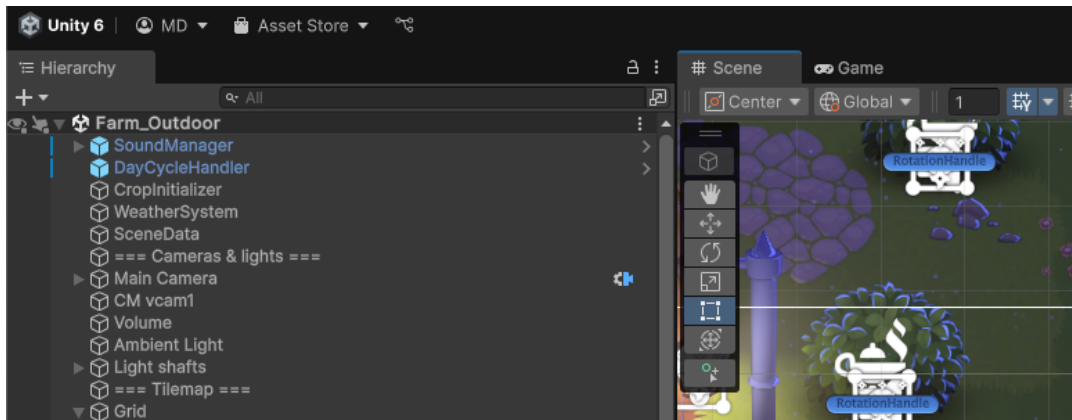
Align With View オプション

12. エディターがシーンで使用するシェーダーバリエーションのリストを生成できます。**Edit > Project Settings > Graphics** を開きます。Shader Loading でシェーダーとバリエーションの数を確認します。**Save to asset...** を選択して、シェーダーバリエーションコレクションアセットを作成します。



Graphics ウィンドウ

13. ウィンドウのタブ (**Project**、**Scene**、**Game** など) をダブルクリックすると、そのウィンドウがエディター内で全画面になります。



Scene をダブルクリック



Scene をダブルクリックした後

その他のリソース

生産性を素早く高めるヒントをさらに知りたい方は、以下のリソースを参照してください。

🔗 6. 結婚式の準備

- [Speed up your workflows in Unity with these keyboard shortcuts](#) (キーボードショートカットで Unity のワークフローを迅速化)
- [Scalable art assets - 6 tips for improving workflows](#) (スケーラブルなアートアセット - ワークフローを改善するための 6 つのヒント)
- [Top tips for scripting in Unity 2022 LTS](#) (Unity 2022 LTS におけるスクリプティングに関する主なヒント)

2D

スプライト

2D プロジェクトでは、[スプライト](#) を使用してビジュアルを作成します。スプライトには異なるテクスチャアセットが含まれている可能性があり、バッチ処理を適切に行わないと多くのドローコールが必要になる場合があります。

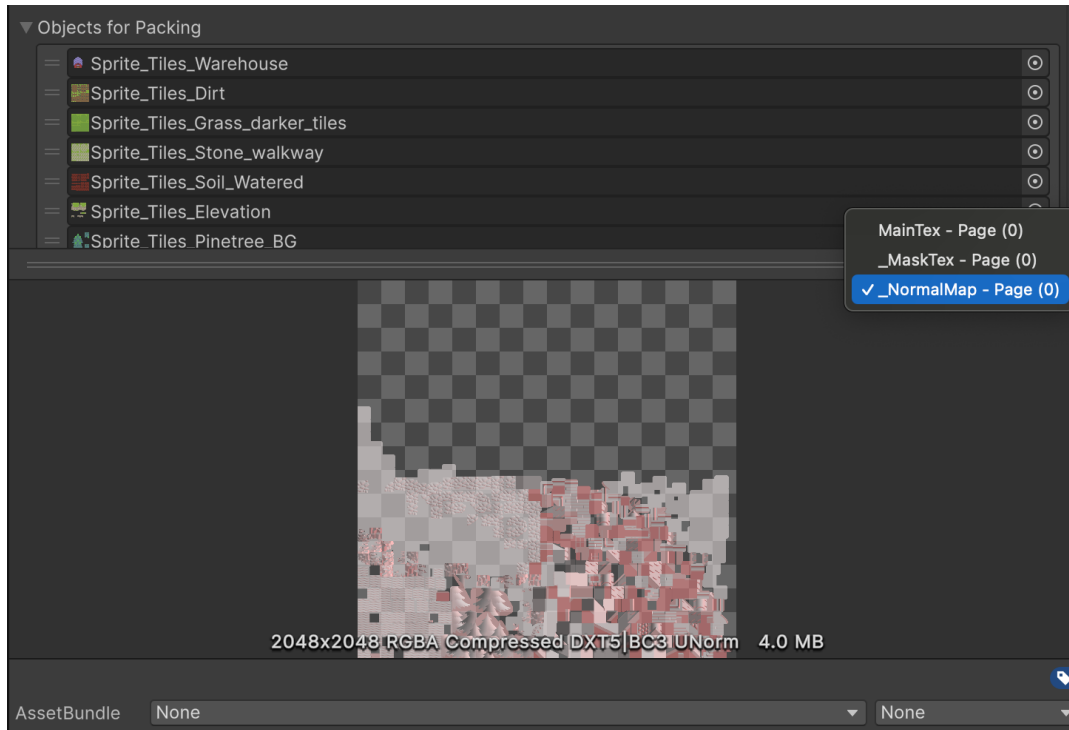
リソースを最適化するには、**スプライトアトラス (Asset > Create > Sprite Atlas)** を使用します。これにより、同じマテリアルとパックされたテクスチャを使ってスプライトをバッチ処理できるようになります。

スプライトのパック時にアセットの重複を避ける

2D URP では、スプライトに法線マップやマスクマップを加えることができ、スプライトアトラスにパックすることもできます。これは、メモリの最適化、ドローコールの削減、ロード効率など、性能と品質の面から重要です。

スプライトアトラスにフォルダーを加える場合、通常のスプライトまたはアルベドテクスチャのみにします。法線マップとマスクマップは、ドロップダウンリストで確認できるそれぞれのアトラスに自動的に加えられます。通常のバックアトラスに不必要に加えられ、スペースを消費しないよう注意してください。

法線マップやマスクマップのアトラスを整理しておく、それぞれの圧縮設定を微調整できます。例えば、法線マップの解像度は半分に、通常のスプライトの解像度をフル解像度にするなど、開発中のプラットフォームに応じて必要な調整を行うことができます。



2D スプライトをパックする際の 3 種類のアトラス

一貫性を保つようにフォルダーを構成する

2D スプライト内のフォルダー構造で、アセットの検索と管理が容易になり、ロード時間に応じてアセットを整理することでメモリ管理に役立ち、さまざまなゲーム要素を明確に分離できます。

Unity プロジェクトのフォルダーを整理するのに役立つ、一般的なフォルダー構造を以下に紹介します。

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

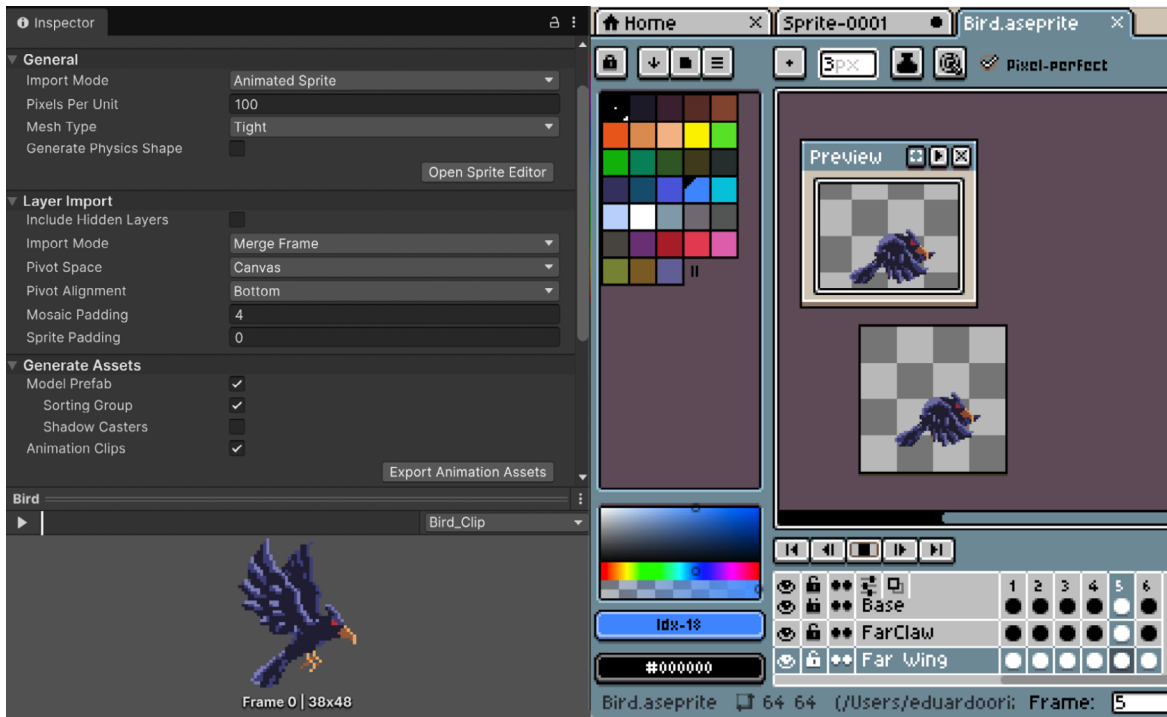
Aseprite Importer でピクセルアートを Unity に直接取り込む

スプライトエディター兼ピクセルアートツールの [Aseprite](#) は、ピクセルアーティスト向けの業界標準の DCC アプリケーションになっています。

Unity の Aseprite Importer を使用すると、何百もの .PNG スプライトを個別にエクスポートする必要がなくなり、**.aseprite** ファイルをプロジェクトに直接加えることができます。

Aseprite で作業し、**Save** をクリックすると、Unity で変更をすぐ確認できます。Aseprite Importer には、.aseprite ファイルのドラッグアンドドロップのサポート、ゲームオブジェクトとアニメーションクリップをすぐに使用できる形での自動セットアップ、包括的な 2D 機能セットの統合が備わっています。

Unity 6 では、タイルマップを含む **.aseprite** ファイルから**タイルマップアセット**を自動生成できます。ソースへの変更は自動的に反映されます。

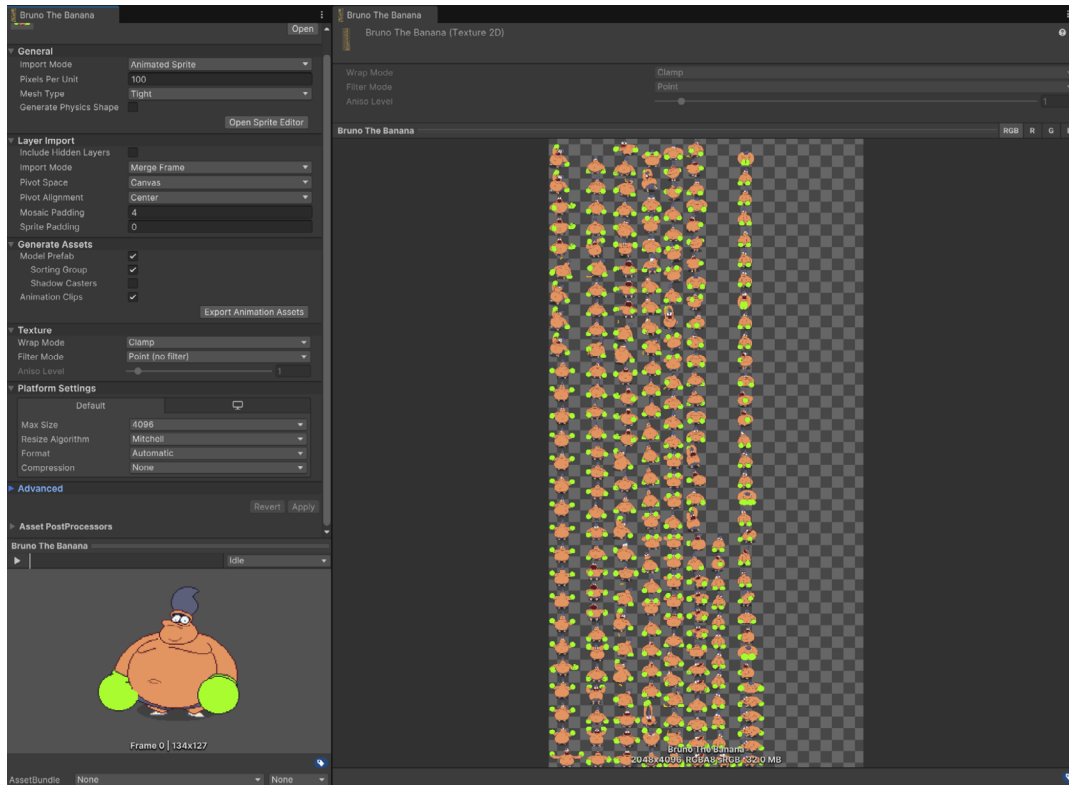


Unity Project ウィンドウの Aseprite Importer: Unite 2024 の講演の[このセクション](#)では、Aseprite Importer の動作を実演する新しい 2D ワークフローと AI 強化機能を紹介しています。

Aseprite からフレーム単位のアニメーションを Unity に取り込む

Unity で Aseprite Importer を使用する際に最も役立つコツの 1 つは、インポート前に Aseprite ファイルを**タグ**と**スライス**で整理しておくことです。これらの機能を使用すると、エディター内で直接、アニメーションやスプライトの分割処理をスムーズかつ自動的に行うことができます。

例えば、“Idle”や“Walk”などのタグを使用してアニメーションを定義して整理したり、スプライトの特定の領域や、頭部、武器などの個別のパーツをマークしてスライスを使用したりできます。その結果、インポートプロセスが合理化され、時間を節約できます。



Aseprite から Unity にインポートされたアニメーション

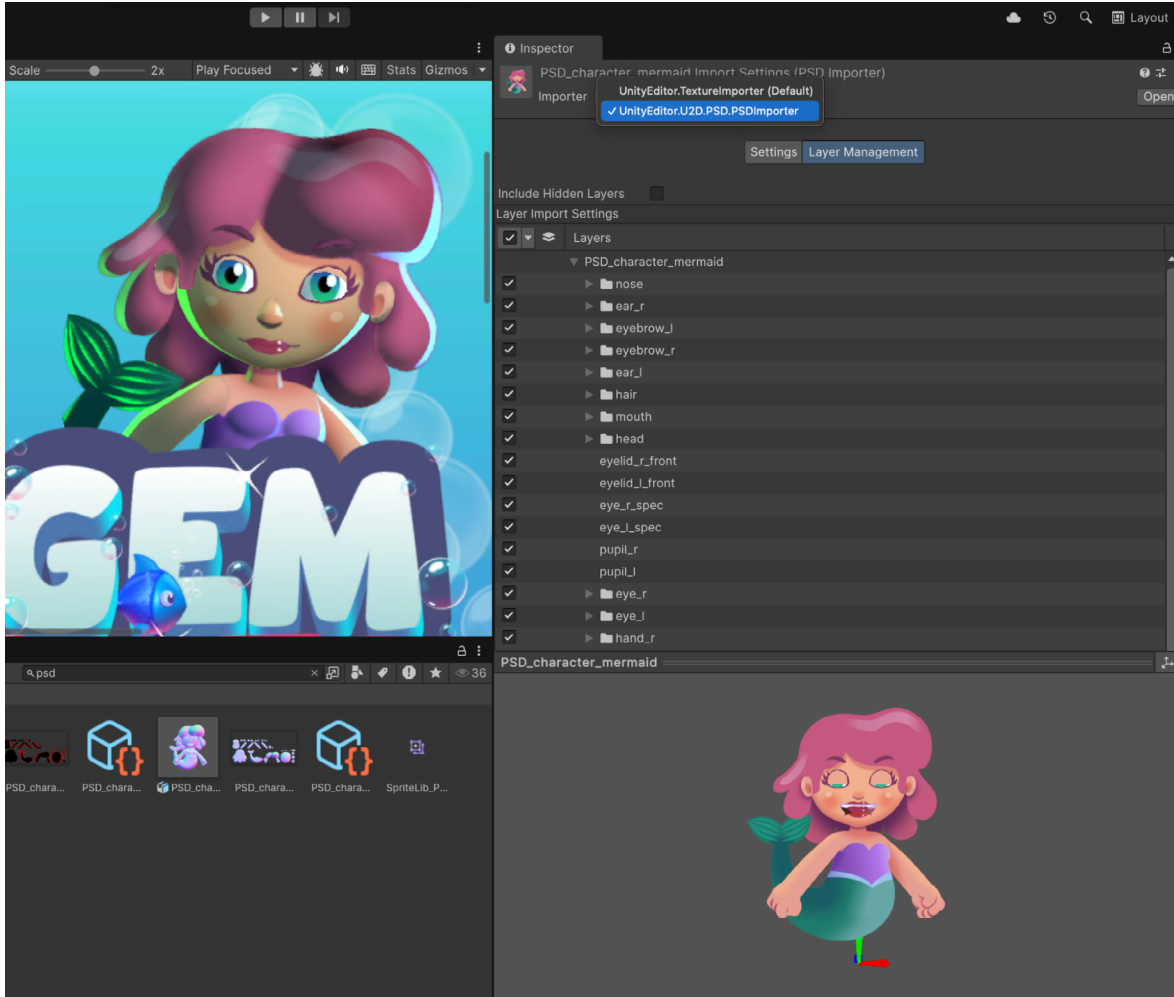
PSD Importer でインポートの往復作業を迅速化する

PSD Importer は Adobe Photoshop のファイルを Unity にインポートし、インポートしたソースファイルに基づいてスプライトのプレハブを生成します。

PSD Importer を使用すると、2D スプライト、UI 要素、その他 Photoshop で作成されたアセットのインポートプロセスを迅速化できます。従来、アーティストは Photoshop ファイル内のすべてのレイヤーを .png ファイルとしてエクスポートして Unity で使用し、変更のたびに .png ファイルを再度エクスポートする必要がありました。Unity がレイヤー化された .psd 形式をサポートするようになったので、こうした手間は不要になりました。

手足などのパーツが異なるレイヤーに配置されている 2D 用のアニメーションキャラクターを、Unity にインポートできます。キャラクターをリグ設定して後でアニメーション化できます。

すべての PSD レイヤーをスタンドアロンのスプライトにする場合は、**Character Rig** オプションをオフにして、プレハブオブジェクトを削除し、すべてのレイヤーをスタンドアロンのスプライトとして使用できるスライス済みの単純なスプライトを得られます。



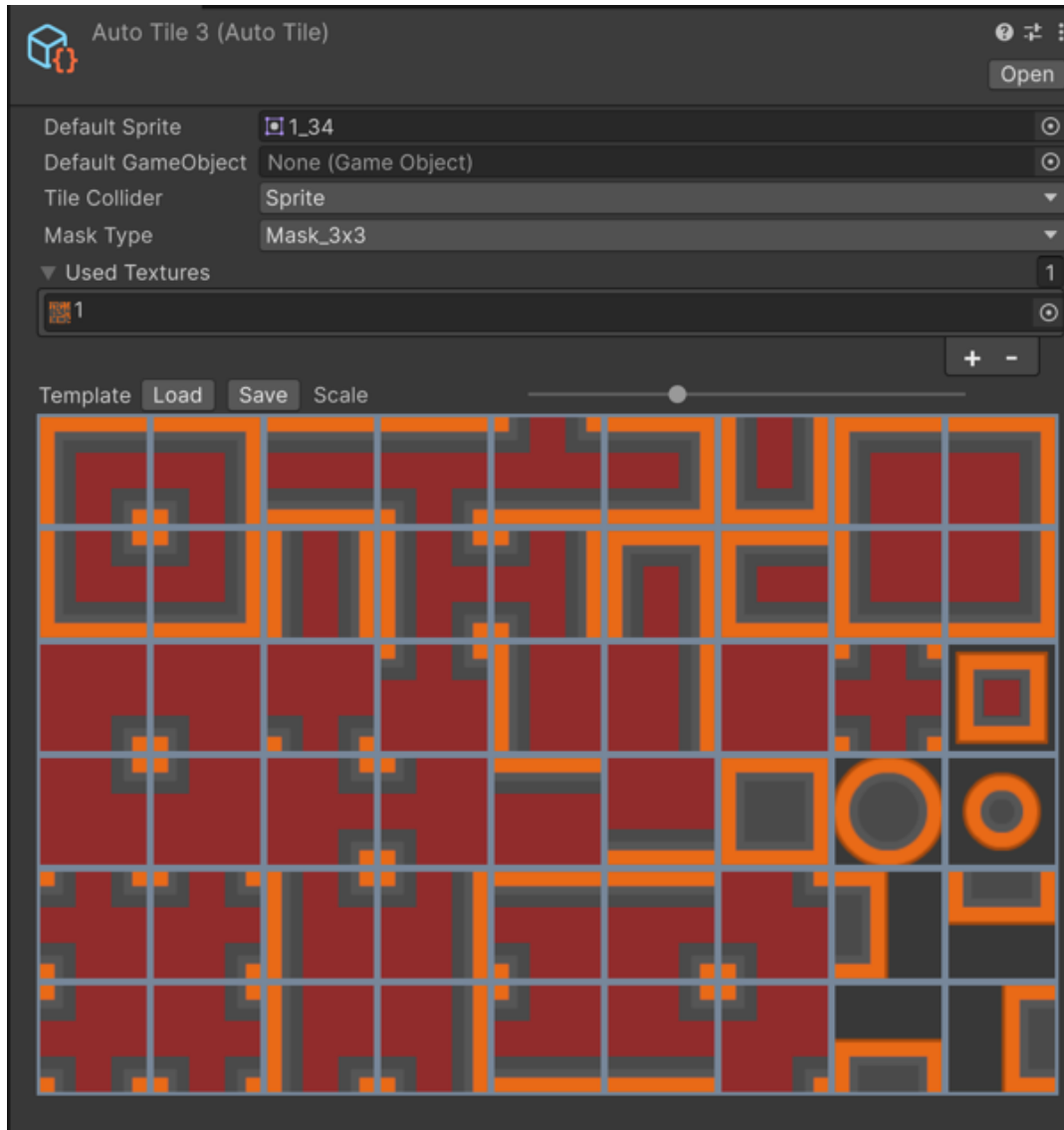
レイヤー化された PSD ファイルは、デフォルトのテクスチャインポーターではなく、PSD Importer でインポートしてください。これにより、すべてのインポーター機能を利用できるようになります。

さらに役立つヒントは、『[From Photoshop to Unity with PSD Importer \(PSD Importer で Photoshop から Unity へ\)](#)＊』の動画チュートリアルを参照してください。Y 娯楽鑑評敵偶月

Unity 6.1 の Tilemap Editor で Rule Tile を簡単に

Unity のタイルマップシステムは、グリッドベースの 2D レベルを作成するために [タイルアセット](#) を保存、管理します。これにより、Unity 内でのレベルデザインのサイクルを簡単に作成し、繰り返すことができます。

タイルを扱う際の困難の 1 つは、すべての隣接タイルのアセットを作成し、それを Rule Tile アセットに割り当てることです。Unity 6.1 では、[AutoTile](#) という新しいオプションによってこのプロセスが簡略化されています。タイルシートを読み込み、その内側の領域を選択して保存することで、タイルパレットで使用できるタイルアセットを生成できます。

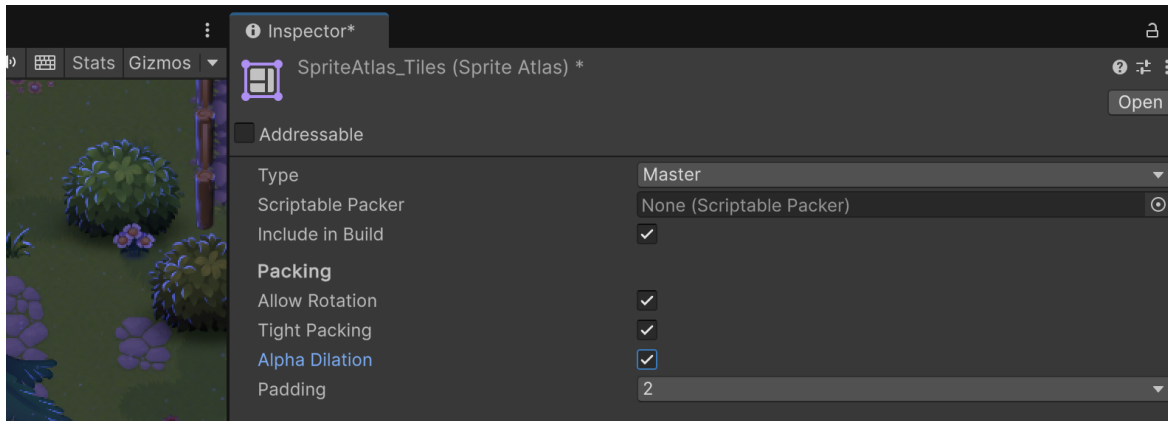


AutoTile の例

タイル間のテクスチャのにじみや小さな隙間を避ける

アートやタイルマップに労力を注ぐのであれば、補間やエッジのスムージングによるタイル間ののにじみやわずかな隙間は避けたいところです (スプライトがスムージングされていないピクセルアートゲームでは問題になりません)。

タイルシートを使用していない場合は、継ぎ目を滑らかにするためにスプライトアトラスを使用してスプライトをパックし、TilemapRenderer で内部ソートすることを推奨します。これにより、プロジェクトの整理、性能、創造的な制御が強化されます。デフォルトの設定でもほとんどのスプライトで機能しますが、タイルマップでは細かい調整が必要になる場合があります。スプライトのパック時の回転を避ける機能や、アルファ拡張は、タイルの鋭いエッジを維持するのに役立ちます。



Sprite Atlas (スプライトアトラス) の Alpha Dilation (アルファ拡張) オプション



左の画像では、タイルの端にじみが見られますが、スプライトアトラスの設定を調整することで修正されました。

2D インバースキネマティクスシステムで自然な動きを作成する

Unity には、[2D Animation](#) と呼ばれるスケルタルアニメーション用の完全なソリューションが備わっています。リギングの動きには、Unity の 2D インバースキネマティクス (IK) システムを使用します。IK を使用すると、各ボーンを手動で調整することなく、膝を曲げたり手を伸ばしたりといった自然で動的な動きを作成できます。



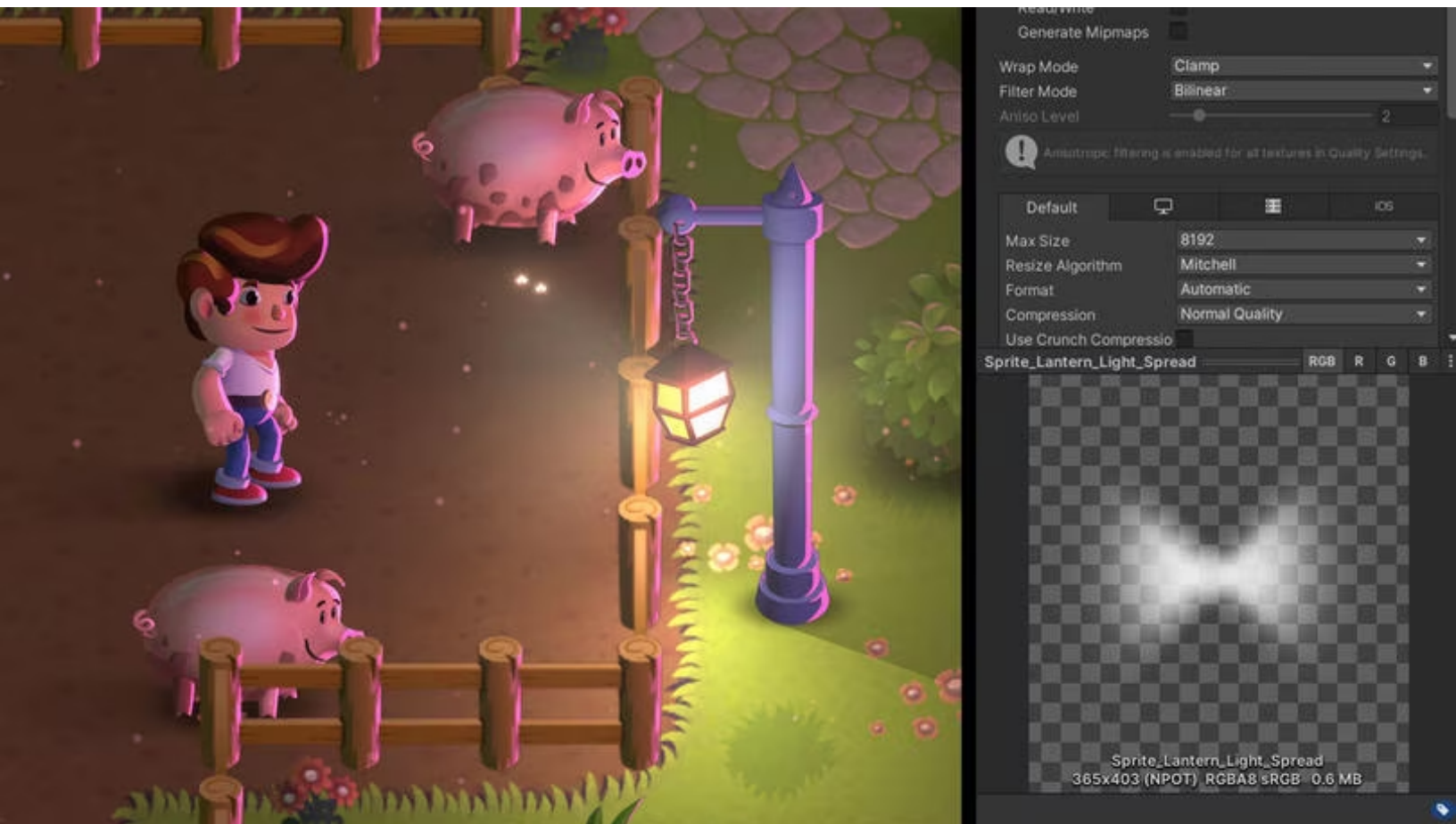
『Happy Harvest』サンプルプロジェクトのメインキャラクターのアニメーションプロセスにおける 2D IK の使用

Sprite Editor でボーンの階層を慎重にデザインし、適切に制御します。例えば、手のボーンを腕のボーンの親にします。単純なアニメーションで階層をテストし、高レベル側のボーンに対する調整が子の側にも適切に影響することを確認します。

[アニメーションブレンドツリー](#) を使用すると、歩行、走行、アイドル状態などのアニメーションを、速度や入力方向などの変数に基づいて滑らかに遷移させることができます。これにより、ステート間で急激に変化することのない、自然なアニメーションを実現できます。

IES プロファイルを 2D ライトでシミュレートする

IES プロファイルを使用してライトの減衰をシミュレートすることは、3D ソフトウェアでは一般的です (HDRP など)。Unity の 2D ライトシステムは、ライトの色、強度、減衰、ブレンド効果などのパラメーターを簡単に設定でき、非常に柔軟性に富んでいます。光源の減衰効果をさらに正確にしたい場合は、スプライトタイプのライトを使用し、カスタムメイドライトの効果を 사용합니다。

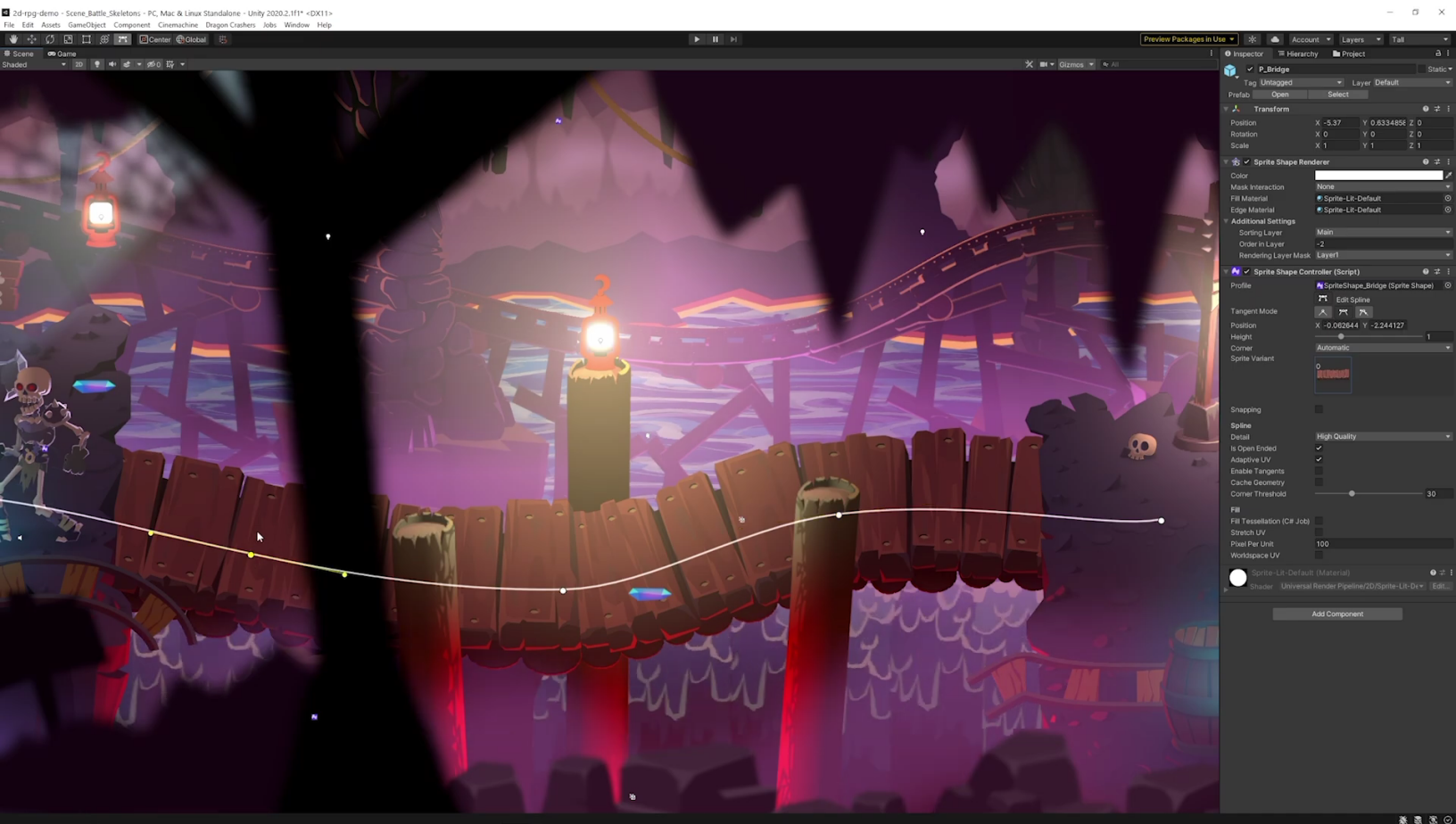


ぶら下がったランプの周りにハローがあるスプライトを使用している『Happy Harvest』サンプルプロジェクト

Odd Bug Studio の Martin Reinmann 氏による [この記事](#) と、ユニバーサルレンダーパイプライン (URP) の 2D ライトとシャドウのテクニックに関する [このページ](#) から、[2D ライティング](#) に関してさらなるヒントを得ることができます。

2D スプライトシェイプで自由度の高い豊かな 2D 環境を作成する

2D スプライトシェイプ を使うと、視覚的で直感的なワークフローにより、自由な形状の豊かな 2D 環境を自由に作成できます。形状の輪郭に沿ってスプライトをタイル状に並べ、輪郭の角度に応じて自動的にデフォームおよびスワップします。



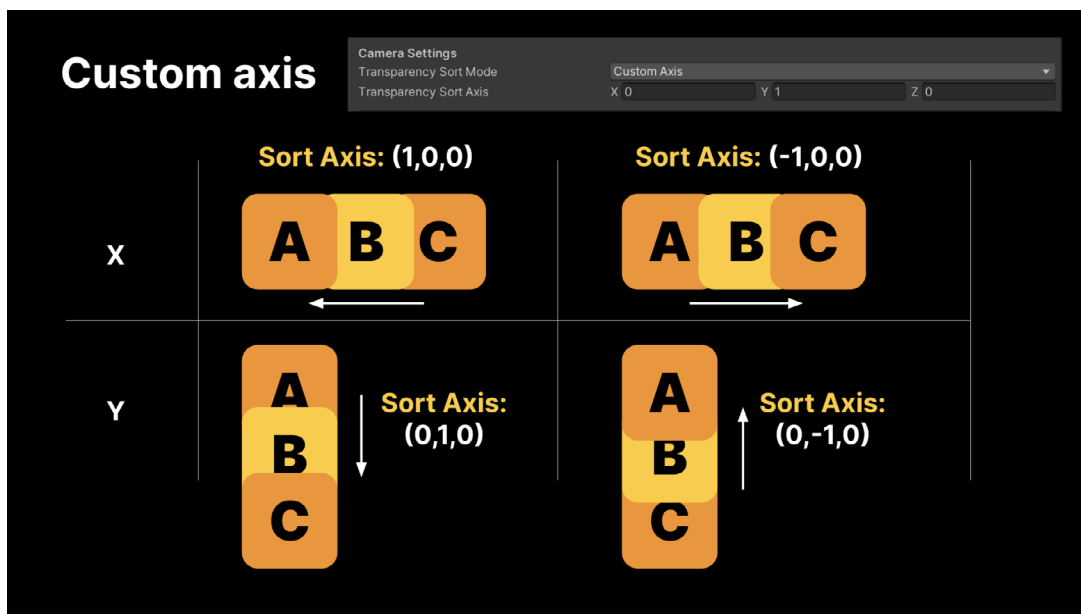
『Dragon Crashers』2D サンプルプロジェクトでの 2D スプライトシェイプの使用例

また、Physics 2D システムでパスや形状を使用する場合は、スプライトシェイプによって Polygon Collider 2D が自動生成されます。

カスタムスプライトソート軸

スプライト を、好みの方向で並べ変えます。同一レイヤーと同一ソート順のスプライトが複数あり、特定の方法で描画順を解決する必要がある場合に役立ちます。例えば、個々のカードが少し重なっているカードゲームで、画面下部のカードを他のカードの上に描画したいとします。

URP の 2D Renderer Asset で、**General > Transparency Sort Mode** に移動し、**Custom Axis** を選択します。例えば、**Transparency Sort Axis** を (0, 1, 0) とすると、Y 軸に沿って上から下にソートされます。



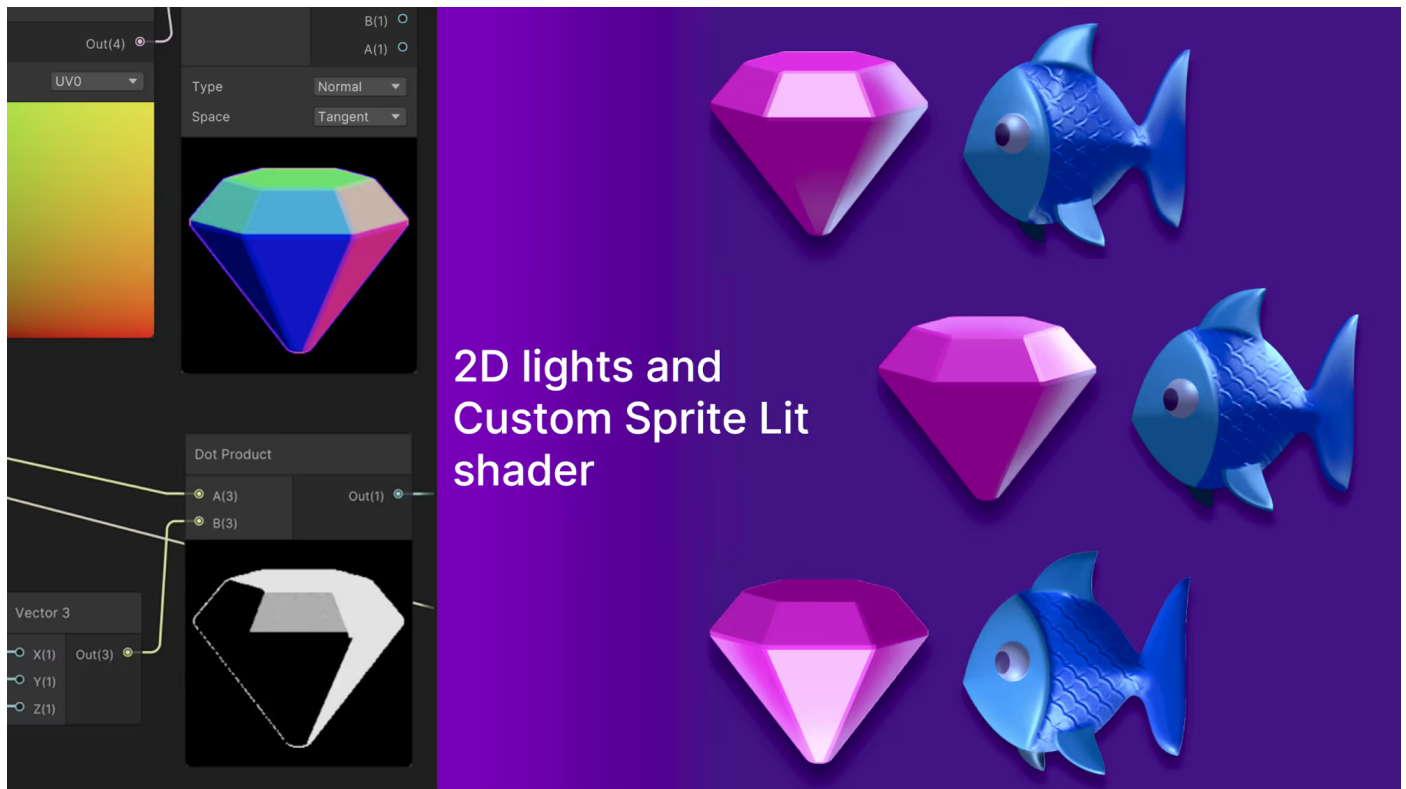
Transparency Sort Mode と Sort Axis (ソート軸) の例。その他のヒントについては、[こちらの動画チュートリアル](#)を確認してください (以前のバージョンの Unity を使用していますが、引き続き適用可能です)。

スプライトカスタム Lit シェーダーでカスタムライティングを作成する

シーン全体のライトに依存しないライティング効果を作成する場合は、**スプライトカスタム Lit シェーダー** の使用を検討してください。

これは、サンプル『[Gem Hunter Match](#)』の視覚効果 (VFX) 作成に使用されている手法の 1 つです。このシェーダーがシーンのライティングの代わりとなり、2D ライトのテクスチャ情報を変更して、各ピースのライティングを制御できます。その結果、スプライトの創造的なイルミネーションが可能になり、ピース上を動くきらめくような効果を実現できます。

ライトの位置データがシェーダー内に移されるため、シーン内に実際のライトオブジェクトを配置する必要がなく、シーンがすっきりします。シェーダー内のオブジェクトごとのライティングをカプセル化することで分離性が高まり、大規模な編集にも適したものとなり、バッチ処理が可能な場合の性能が向上します。



『Gem Hunter Match』の spriteカスタム Lit シェーダー

spriteカスタム Lit シェーダーを使用した 2D spriteのオブジェクトごとのライティングによる創造的な制御方法については、[Find a treasure trove of lighting and visual effects in our new match-3 sample Gem Hunter Match](#) (新しい 3 マッチパズルサンプルの『Gem Hunter Match』でライティングと視覚効果の宝庫を見つける) を参照してください。



透明ピクセルのオーバードローを減らす

オーバードローを避けることで、性能が向上します。[Import Settings](#) で **Mesh Type** を **Tight** (デフォルトオプション) に切り替えます。可能であれば、重なり合うグラフィックスを 1 つのスプライトにマージし、ゲーム内で使用されない背景レイヤーのスプライトを無効にするようにしてください。これにより、オーバードロー領域と、隣接するスプライトと重なる可能性が減少します。



1920×1080 game
(2,073,600 pixels)
@ 60 fps

Overlapping pixels

1x Overdraw

2x Overdraw

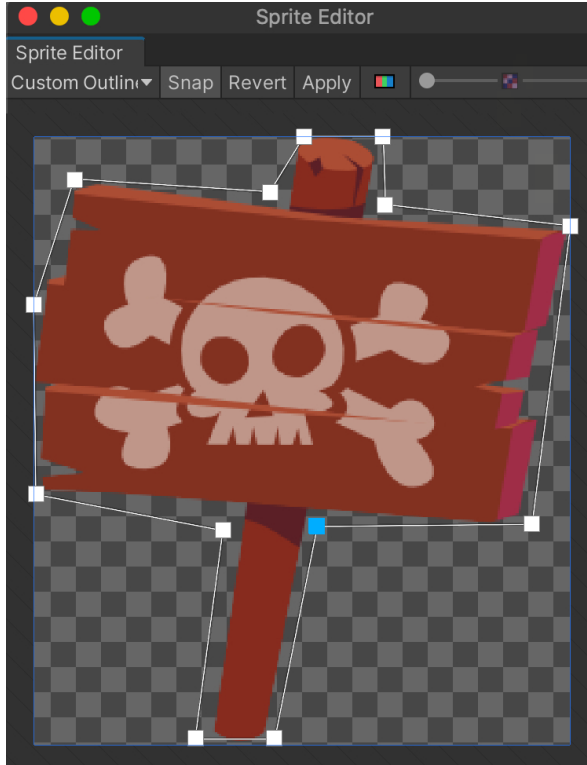
3x Overdraw

**Pixels in that area are
overdrawn 4x every frame**

スプライト間のオーバードローを減らします。

Sprite Editor で未使用領域を最小限に抑える

[2D Sprite Editor](#) を使用して各スプライトの周囲の [カスタムアウトライン](#) を定義することもできます。これにより、スプライトをパックする際に未使用の領域を最小限に抑え、オーバードローを回避できます。

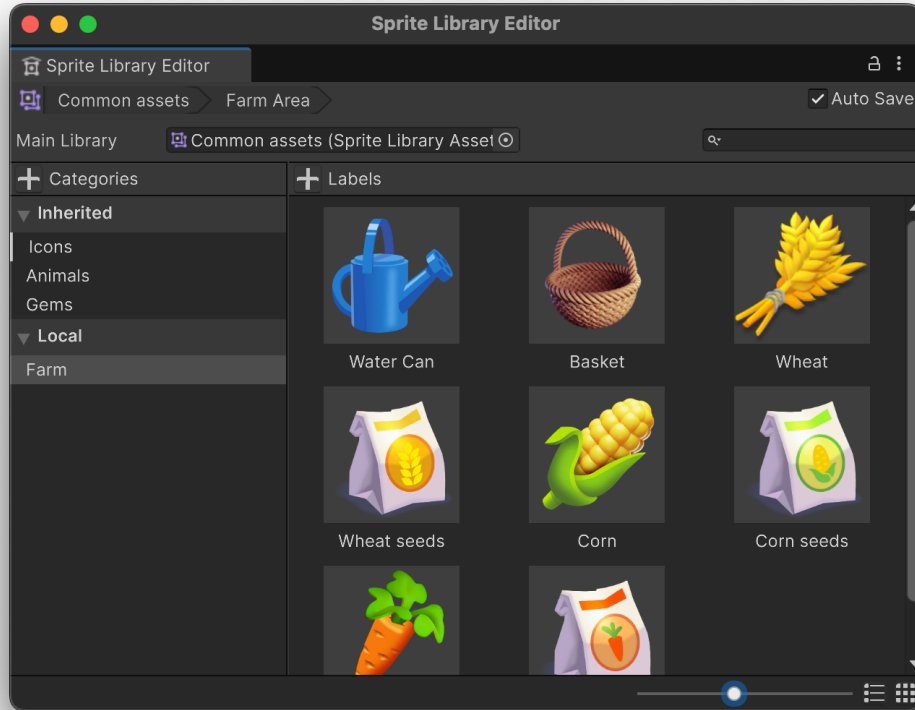


Sprite Editor の Custom Outline (カスタムアウトライン)

Sprite Library でプロジェクトのスプライトとアニメーションを整理する

[Sprite Library](#) (スプライトライブラリ) を利用して、レベルデザインや複数パーツで構成されたキャラクター用のスプライトを簡単に整理できます。

スプライトライブラリは、レンダリング性能向上のためにスプライトをまとめるスプライトアトラスとは異なります。ただし、スプライトライブラリ、スプライトアトラス、Addressables には互換性があるため、これらを組み合わせて使用することで、大規模なプロジェクトを整理することができます。



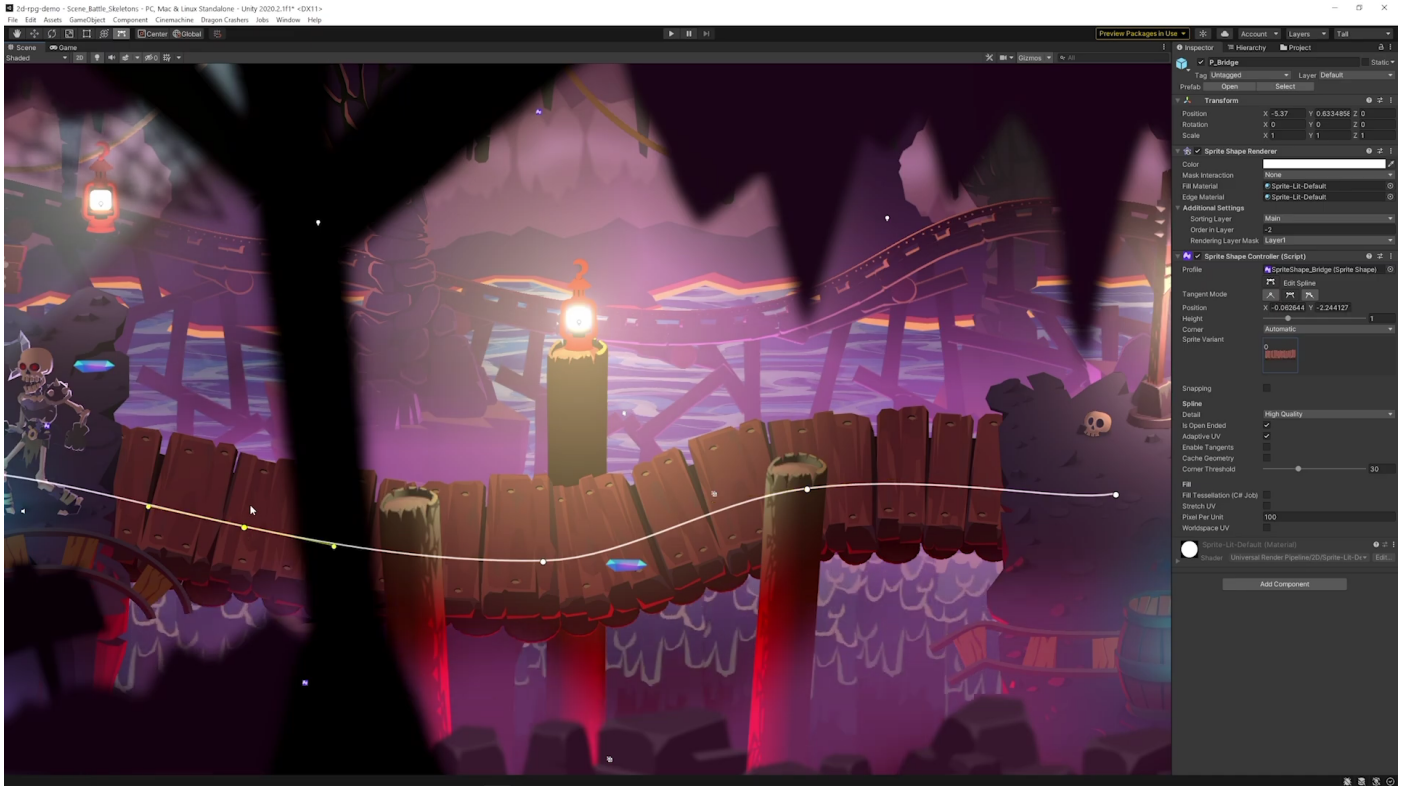
Sprite Library Editor ウィンドウ

スプライトシェイプの制御点を変更する

スプライトシェイプ は、丘や、動的に変形可能な道路、トレイル、レース場、水域、さらには装飾要素など、自然な 2D 地形のデザインに役立ちます。

また、ゲームプレイ機能を作成することもできます。Unity の 2D 物理演算と組み合わせることで、リアルタイムに反応し再形成される相互作用する領域 (破壊可能な地形やデフォーム可能なプラットフォームなど) を作成できます。

スプライトシェイプの制御点は変更できます。[API](#) を使用すると、ランタイムまたはエディター使用時に制御点にアクセスできます。この機能の処理には負荷がかかりますが、慎重に統合すれば、ゲームにさらなる優位性をもたらすことができます。



2D URP サンプルプロジェクト『Dragon Crashers』でのスプライトシェイプの制御点の変更。

コンテキストメニューからスプライトを簡単にスワップする

Sprite Resolver は 2D Sprite Library システムと連携して動作するコンポーネントで、見た目の異なるスプライトをランタイムに動的にスワップする機能を提供します。

ゲームオブジェクトのスプライトを Inspector で手動で割り当てることなく変更できるため、プロジェクトのスプライトをより強力かつ柔軟に管理できます。

Sprite Resolver の優れた使用事例をいくつか紹介します。

- キャラクターのカスタマイズシステム
- NPC やキャラクターの動的な表情や雰囲気
- 季節やゲーム内イベントに基づく環境スプライトの変更
- オブジェクトの視覚的な向上や劣化
- 効率化と整理のためにアニメーションのスプライトスワップを合理化

Unity 6 では、Inspector ウィンドウだけでなく、エディターの他のウィンドウでも Sprite Resolver を使用できます。



Sprite Library の Sprite Resolver コンポーネント

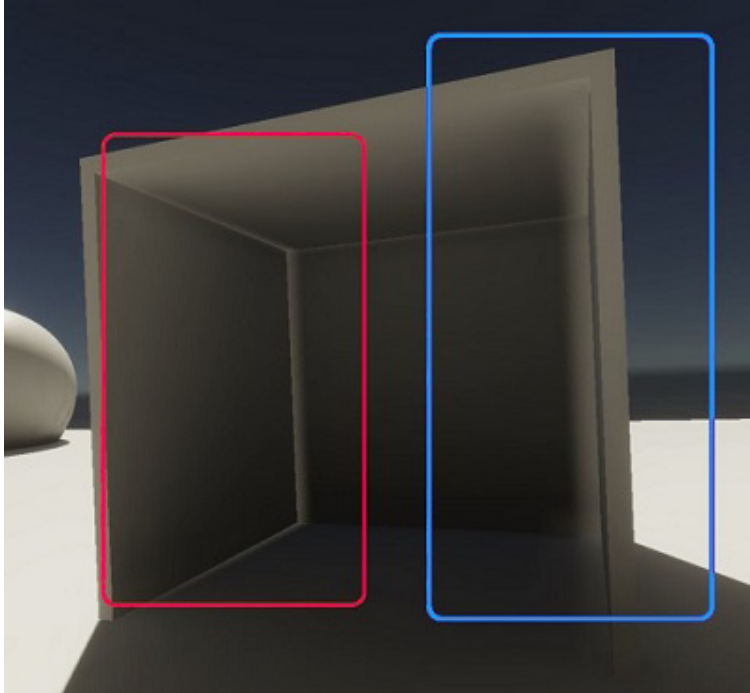
グラフィックスと アートアセット

ライトプローブによるライトリーク

ライトリークは通常、ジオメトリが自身には見えていないライトプローブからのライトを受け取ったときに発生します。例えば、ライトプローブが壁の反対側にある場合などです。

これは複雑なトピックですが、ライトリークの修正に役立つテクニックをいくつか紹介します。

- [Create thicker walls](#) (壁を厚くする)
- [Add an Adaptive Probe Volumes Options Override to your scene](#) (Adaptive Probe Volume Options のオーバーライドをシーンに加える)
- [Enable Rendering Layers](#) (レンダリングレイヤーを有効にする)
- [Adjust Baking Set properties](#) (ベイク設定のプロパティを調整する)
- [Use a Probe Adjustment Volume](#) (Probe Adjustment Volume を使用する)

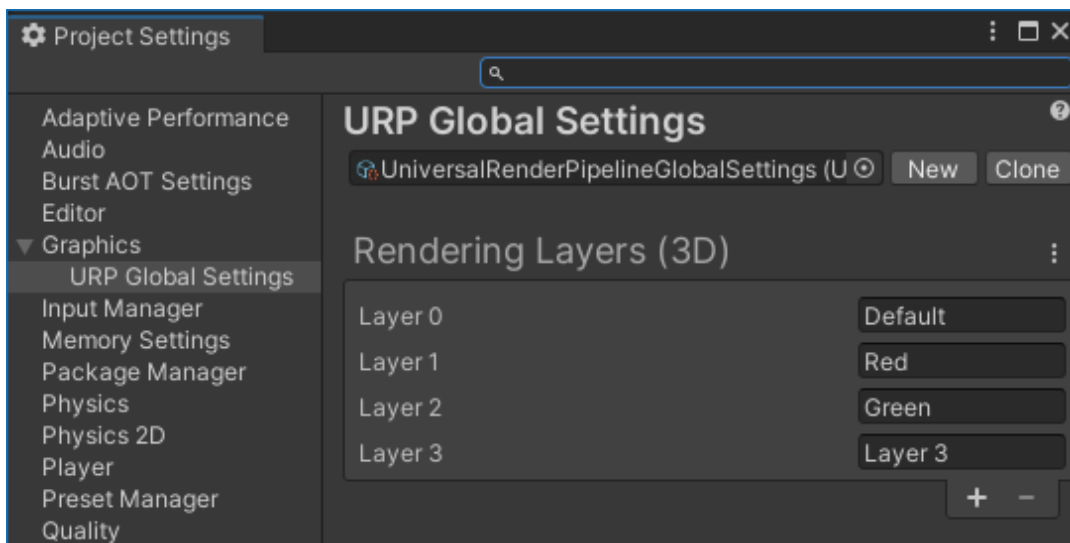


ライトリークの例

レンダリングレイヤーを使用して特定のゲームオブジェクトに特定のライトを設定する

Rendering Layers (レンダリングレイヤー) 機能を使用すると、特定のライトが特定のゲームオブジェクトにのみ影響するように設定できます。

Custom Shadow Layers (カスタムシャドウレイヤー) プロパティを使用すると、特定のゲームオブジェクトが特定のライトからのみ影を落とすように設定できます (それらのライトがゲームオブジェクトに影響を与えない場合でも)。

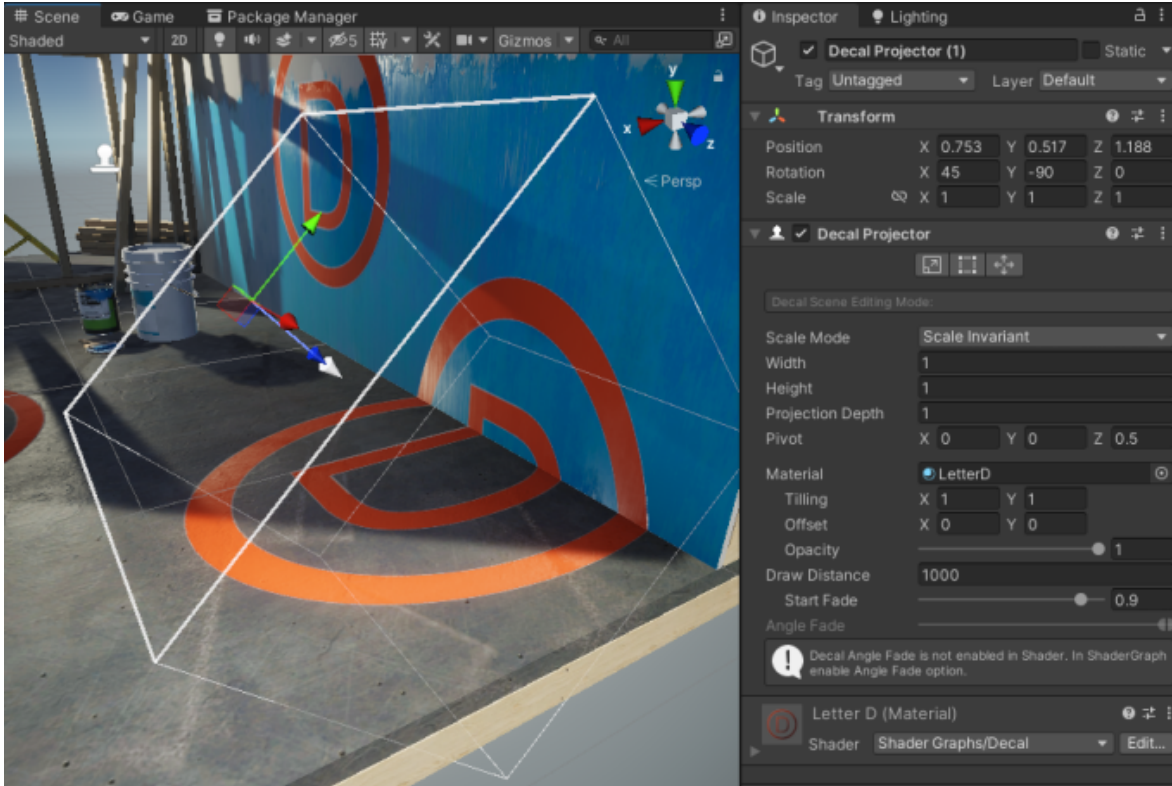


URP Global Settings の Rendering Layers

Decal Projector でメッシュにディテールを加える

Decal Projector コンポーネントは、メッシュにディテールを加える素晴らしい方法です。銃弾の穴、足跡、看板、亀裂などの要素に使用できます。

Decal Projector は投影フレームワークを使用しているため、平らでない表面や曲面に適応します。URP で Decal Projector を使用するには、レンダラーデータアセットを見つけて、**Decal Renderer Feature** を加える必要があります。



シーンビューの Decal Projector

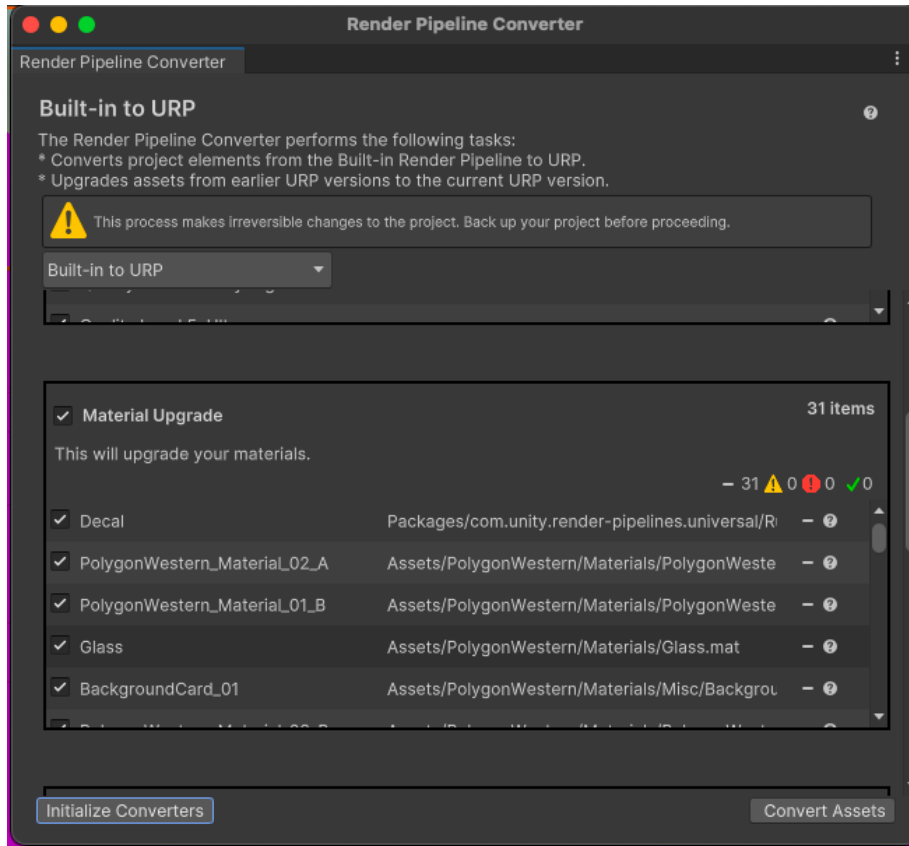
カスタムシェーダーをビルトインレンダーパイプラインから URP に変換する

URP ドキュメント内の [この表](#) を参照することで、各 URP シェーダーがビルトインレンダーパイプラインのどのシェーダーに対応しているかを確認できます。

コンバーターを 1 つ以上選択して、**Initialize Converters** または **Initialize And Convert** をクリックします。どちらのオプションを選択しても、プロジェクトがスキャンされ、変換が必要なアセットが各コンバーターパネルに加えられます。

Initialize Converters を選択する場合、各項目のチェックボックスの選択を解除することで、変換を制限できます。ここで **Convert Assets** をクリックすると、変換プロセスが始まります。**Initialize And Convert** を

選択する場合、コンバーターの初期化後、変換が自動的に始まります。完了後、エディター内でアクティブなシーンを再度開くように求められる場合があります。

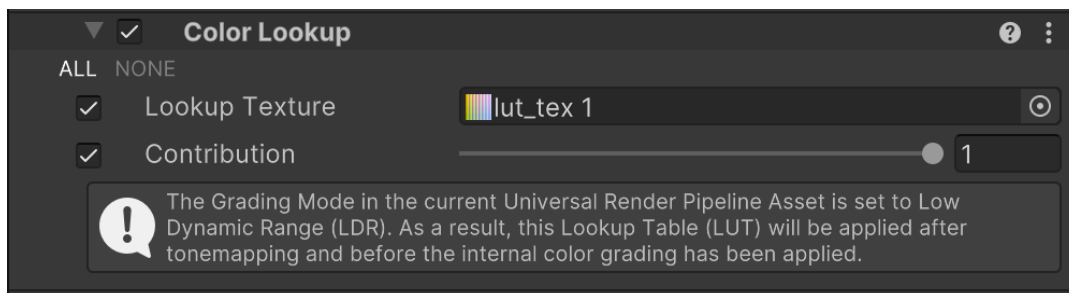


レンダーパイプラインコンバーター

LUT テクスチャでカラーグレーディングを作成する

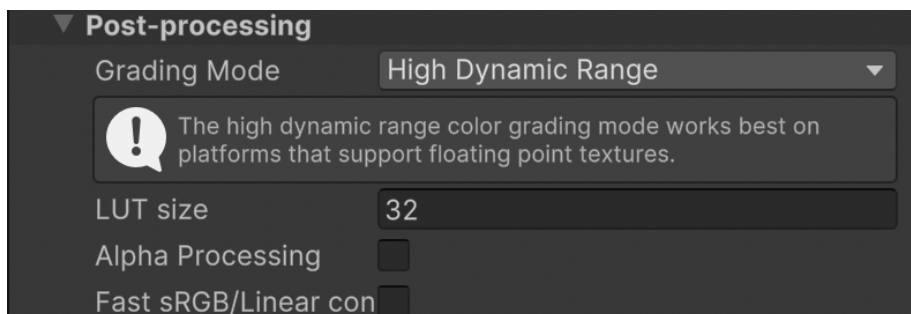
Color Grading (カラーグレーディング) 効果は、Unity が生成した最終画像の色と輝度の調整や補正を行うものです。これにより、プロジェクトのルックアンドフィールを変えることができます。

Lookup Texture と **Contribution** で、ボリュームに設定するポストプロセスエフェクトのカラーグレーディング効果の動作を制御します。

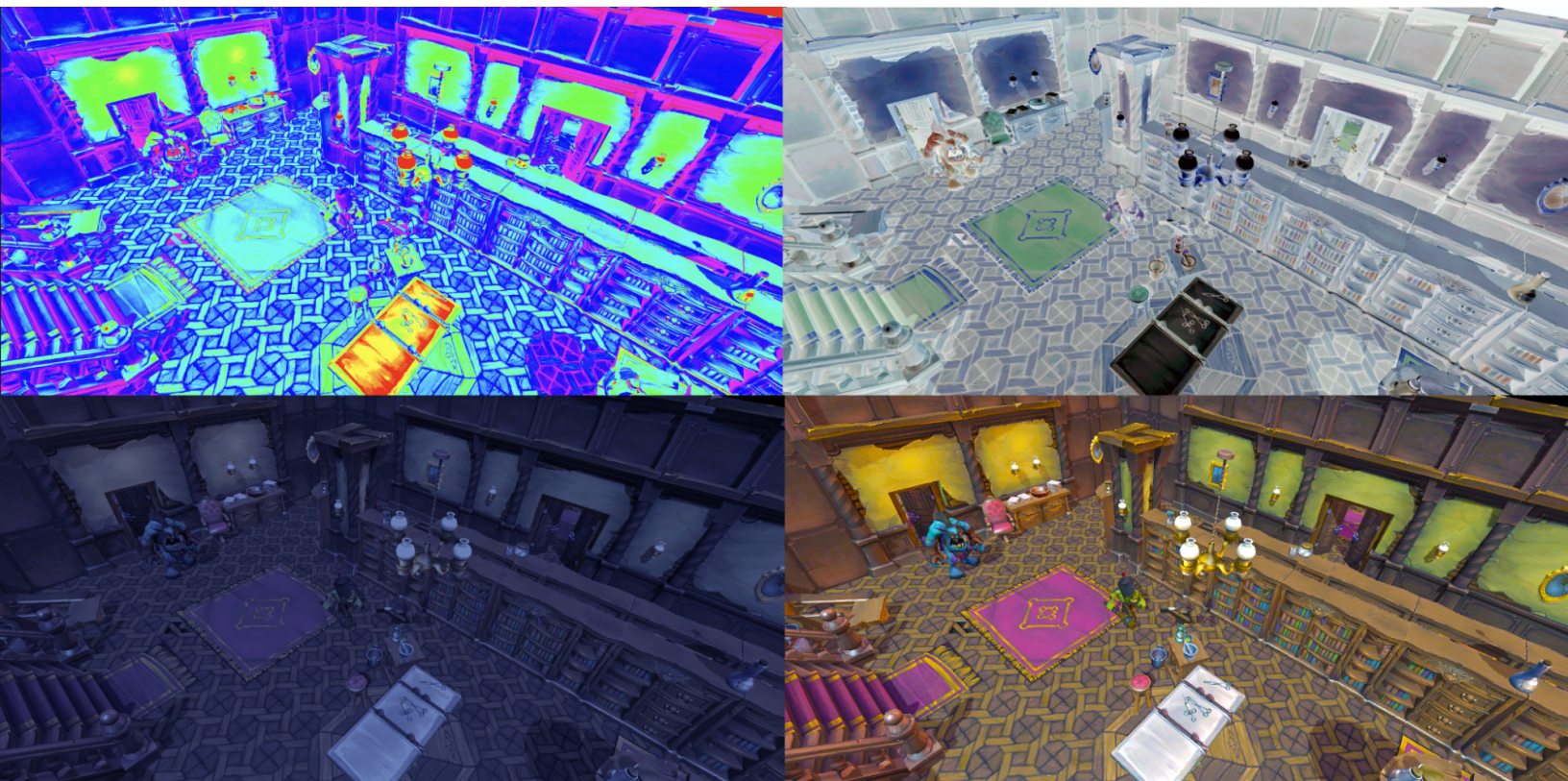


Color Lookup の Lookup Texture と Contribution の設定

複数の低ダイナミックレンジ (LDR) ルックアップテクスチャ間のボリュームブレンディングはサポートされていますが、正しく機能するのは、サイズが同じ場合のみです。このため、プロジェクト全体で LUT サイズを統一することを推奨します (256 × 16 または 1024 × 32)。



URP アセット設定での LUT サイズの設定



さまざまな LUT テクスチャの使用

レンズフレアでリアルなレンズ効果とスタイル化された外観を作成する

レンズフレア は、カメラのレンズに明るい光が当たったときに現れるアーティファクトです。レンズフレアは、1つの明るい輝きとして現れることもあれば、カメラの絞りと同じ多角形の複数色のフレアとして現れることもあります。

レンズフレアに慣れるには、Package Manager からサンプルをインストールしてください。レンズフレア効果を含む一連の定義済みフレアアセットを利用できるようになります。また、レンズフレアを参照するためのテストシーンも含まれており、自分でレンズフレアをビルドする際にも役立ちます。



レンズフレアのサンプルでは、リアルなレンズエフェクトからよりスタイリッシュに見えるものまで、さまざまなプリセットが用意されています。これは HDRP サンプルシーンを使用している例です。

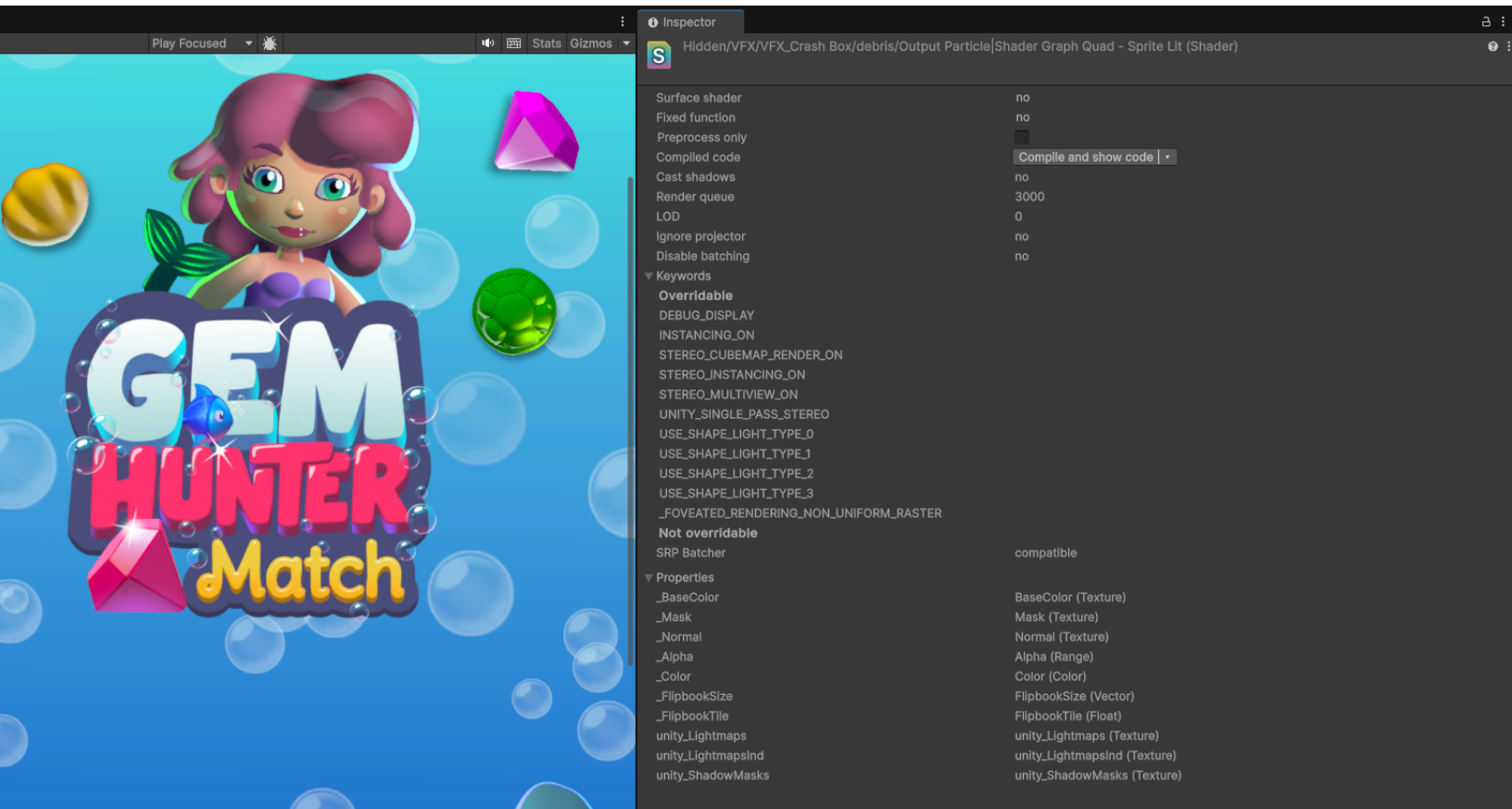
レンズフレアは、レンズフレア要素で構成されています。それぞれの要素は、フレアが生成するさまざまなアーティファクトを表しています。要素の形状は、ポリゴン、円、またはカスタム画像にすることができます。

要素パラメーターでは、**Color**、**Transform** 位置、デフォーメーション、またはフレア要素が複数の要素を持つレンズフレア効果の一部である場合の位置、カラー、スケールを微調整できます。光源にアタッチされている場合、フレア要素でティントカラーを使用できるため、同じ Flare アセットをさまざまな光源で再利用できます。

レンズフレアがどのように機能するかについては、[こちらの SIGGRAPH の講演](#) で詳しく学ぶことができます。

シェーダーバリエントを管理する

ビルドのサイズと時間を減らすために、シェーダーバリエントは慎重に管理してください。バリエントが特定のマテリアルでのみ必要な場合は、`#pragma multi_compile` の代わりに `#pragma shader_feature` を使用します。`shader_feature` バリエントは、マテリアルで使用されない場合やランタイムで有効にならない場合は取り除かれますが、`multi_compile` バリエントは常に含まれます。シェーダーの **Inspector** の **Keywords** を確認します。



『Gem Hunter Match』サンプルのユニバーサルレンダーパイプライン (URP) /Lit (Shader) ウィンドウの Keywords セクション

マテリアルのバリエーションを作成する

マテリアルバリエーションを使用すると、テンプレートやマテリアルプレハブを作成できます。ベーステンプレートに基づいて、テンプレートマテリアルと共通のプロパティを持ち、異なるプロパティのみをオーバーライドするバリエーションを作成できます。テンプレートマテリアルのオーバーライドされていない共通プロパティを変更すると、変更が自動的にバリエーションマテリアルに反映されます。また、マテリアルの特定のプロパティをロックして、バリエーションでオーバーライドできないようにすることもできます。



マテリアルバリエーションの例。いずれもベースマテリアルは同じで、色のプロパティのみが異なります。

より複雑な設定では、マテリアルバリエーションのバリエーションを作成できます。このマテリアルの継承階層により、再利用性が高まり、プロジェクトにおけるマテリアル作成のイテレーションスピードとスケーラビリティが向上します。

アセットワークフローを改善するためアセットのグループ化を計画する

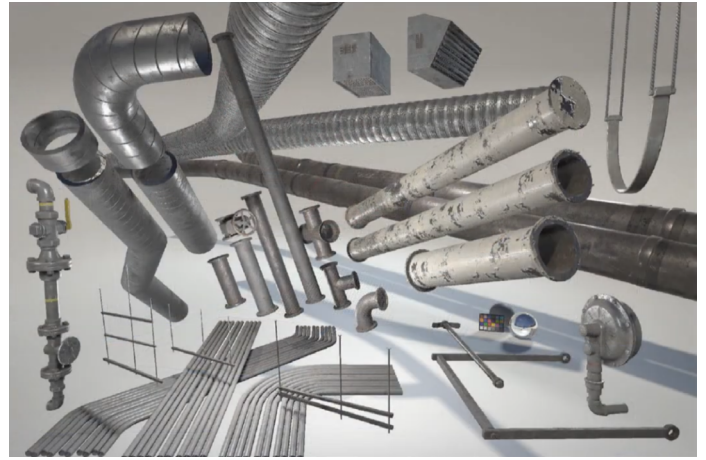
制作前の計画段階で、制作する必要があるアセットのリスト、プロジェクトの技術要件に基づくアセット制作バジェット、開発コストなど、アセットワークフローのさまざまな側面についてチームで合意する必要があります。

必要なレベル要素をグループ化する 1 つの方法として、次のものがあります。

- **大型要素:** 壁、地面、崖、建物の屋根などは、個別のテクスチャでは作成できず、タイルまたはモジュール式の要素を使用する必要があります。
- **中型の小道具:** 樽、木箱、岩、扉など、個別のテクスチャシートを含む要素です。
- **頻繁に再利用される小型要素:** 小石、葉、ネジ、ボルトなど、オブジェクトが接地していることを見せたり、スケールを伝えるために使用されたりするアイテムです。



種類、サイズ、形状別のアセット制作計画

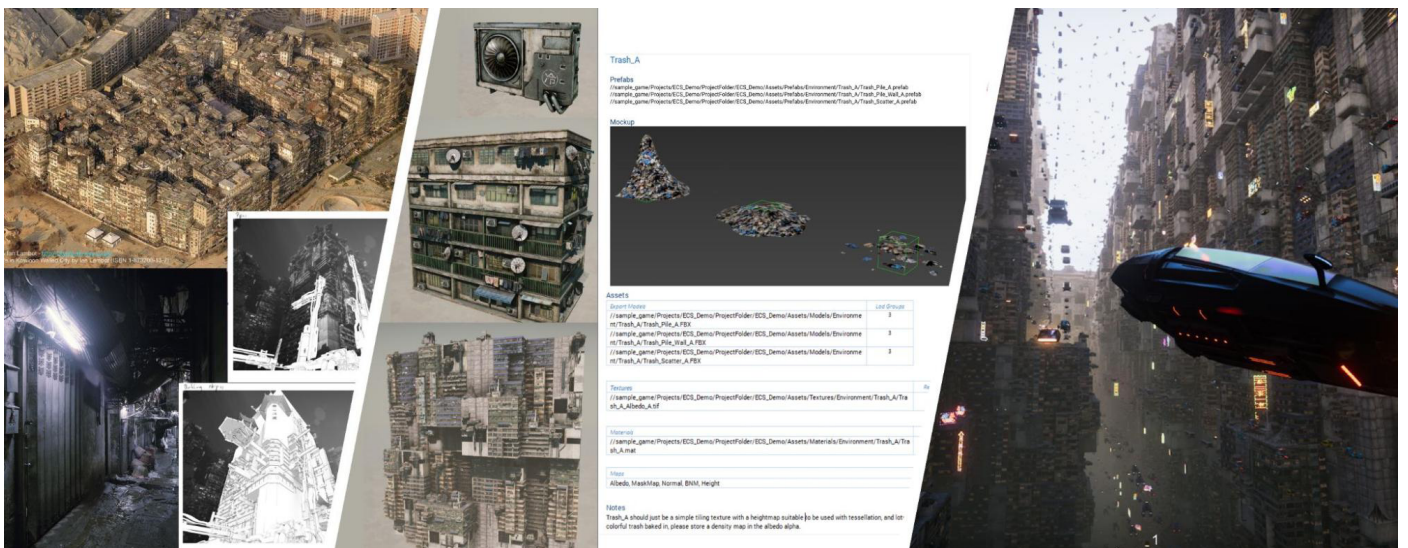


各アセットの時間、ポリゴン数、テクスチャ解像度のバジェットを検討し、それに応じてタスクに優先順位を付けます。そして、必須ではないアセットを作成し、必須アセットの数を必要最小限に抑えます。

AssetPostProcessor API でインポートパイプラインを自動化および迅速化する

何千ものアセットを扱う制作では、各アセットの仕様を Inspector で手動設定することは避けてください。

アセットファイルの設定と検証を自動化するには、[AssetPostProcessor API](#) を使用します。これにより、インポートパイプラインにフックして、アセットのインポート前後にスクリプトを実行できます。スクリプト用のアセットを加えて必要な変更を行うだけです。1 つのスクリプトから設定を変更すれば、プロジェクト内のアセットに自動的に変更が再適用されます。



左から順に: AssetPostProcessor を使用したムードボード、コンセプトアート、プレハブアセット、ディレクトリ、命名基準の例、および Unity による『Megacity』デモの最終ビジュアル

プレハブバリエーションを使用してチームの作業を効率化する

アセットインポートワークフローのもう 1 つの重要な側面は、3D モデルからのプレハブの作成です。制作中に FBX ファイルを直接プレハブに変換することは推奨しません。なぜなら、ジオメトリの階層変更などを FBX に対して行うとプレハブが壊れてしまうからです。その場合、シーン内のすべてのインスタンスを更新されたバージョンに置き換える必要があります。この問題を解決するのが Prefab Variant (プレハブバリエーション) です。

プレハブバリエーションを使用すると、シーンに依存しないアートアセットの制作が可能になるため、プロトタイプ作成に最適です。ゲームワールドの構築時の共同でのイテレーション作業で役立ちます。元のモデルのバリエーションを作成するなど、通常のプレハブの利点だけでなく、FBX モデルのプレハブへの“弱い”参照としても機能します。これにより、プレハブを再作成することなく、モデル要素を簡単に追加、分割、削除できます。制作アーティストが FBX モデルに変更を加えると、プレハブバリエーションもそれに応じて更新されます。また、プレハブバリエーションを使用すると、アーティストはプロジェクト自体を変更することなく、アートの変化をコンテキスト内で確認できます。

同時に、ネスト状のプレハブを使用して、シーンでの共同作業を円滑にすることを検討してください。



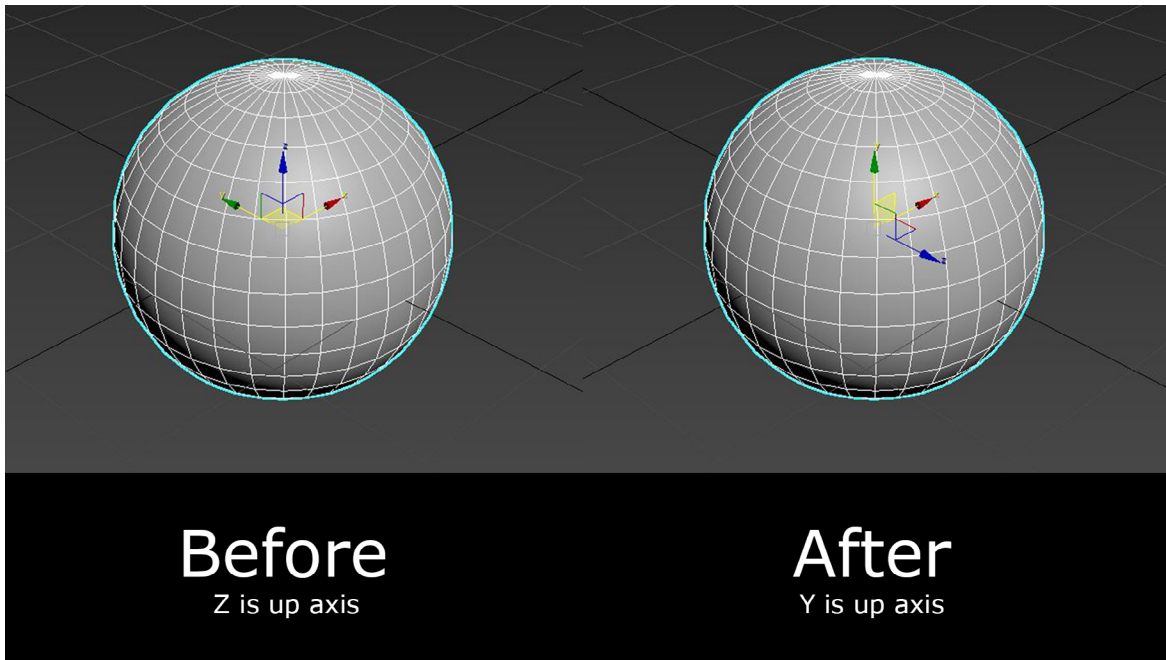
共同作業が多い制作に推奨される Prefab Variant (プレハブバリエーション) ワークフロー

XR やモバイル開発におけるアセットの考慮事項

XR やモバイルゲーム用のアートアセットの制作および調達の早い段階での決断が、開発工程後半における時間の節約につながります。小道具からキャラクターまで、アートアセットを効率的に作成するためのヒントを紹介します。

- アートアセットのコンセプトをはっきりさせて作成を始めます。コンセプトは、手描きのような単純なもので説明することで、特に初期段階では、インタラクションや見た目をすばやく計画することができます。
- 大規模なスタジオでは、アイデアに命を吹き込むコンセプトアーティストがいますが、個人や小規模なチームの場合は、アイデアを表現するために AI が役立ちます。
- 3D モデリングソフトウェア (Blender、Autodesk Maya、3ds Max など) を使用してモデルを構築します。特にモバイル AR では性能のためにポリカウント (ポリゴン数) を考慮する必要があります。ポリゴン数の多いモデルを最初に作成し、後で最適化します。

- 3D モデリングソフトウェアでモデルのピボットポイントを設定し、その向きが Unity のものと一致するようにします。3ds Max でピボットポイントを設定する方法の例を下に示します。この例では、Z 軸をアップ軸として使用します。



アップ軸の調整

- 次の 2 つのステップで、Unity でのモデルの向きを修正することもできます。
 - a. Hierarchy に空のゲームオブジェクトを作成します。
 - b. モデルをこのゲームオブジェクト内にネストし、その位置をすべての軸で 0,0,0 に設定して、完全に中央揃えにします。

この方法により、すべての軸で回転が 0 の中立的な状態を保つ親ゲームオブジェクトにモデルが固定されます。また、モデルは Z 軸が正しく前方を向き、すべての軸でスケールが 1 に統一された状態が保たれます。さらに、Importer で Bake Axis Conversion オプションを試すこともできます。
- テクスチャを適用して、モデルをリアルな、あるいは様式化された見た目になります。詳細なテクスチャについては、UV マッピングの使用を検討してください。ゲームの美観によっては、PBR Textures が必要になる場合があります。
- Unity の物理ベース (PBR) シェーダーには通常、2 つの主要コンポーネントが含まれます。Metallic-roughness (メタリック/粗さ) ワークフローと Specular-glossiness (反射/光沢) ワークフローです。それぞれマテリアルの特性の異なる側面を処理します。その結果、よりリアルで躍動的な見た目を実現し、Unity ベースのゲームやシミュレーションの 3D シーンとオブジェクトの現実感を高めます。

広いテクスチャ領域をトリムシートで最適に配置する



環境全体のさまざまなメッシュにトリムシートを使用します。

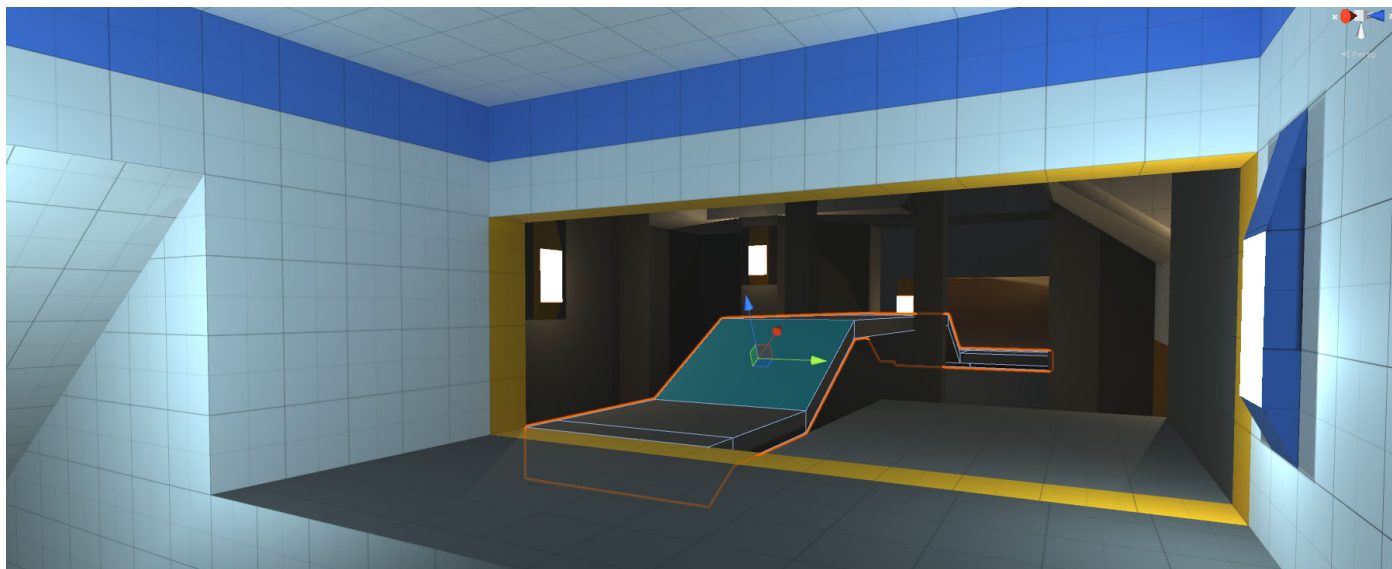
トリムシートは、3D モデリングやゲーム開発で使用するテクスチャツールです。成形、エッジ、境界線など、さまざまなトリムパターンやディテールを含む単一のテクスチャで構成されます。

トリムシートを使用すると、視覚的に複雑で多様なモデルのデザインを可能にする一方で、必要な個々のテクスチャの数が減るため、ゲームのパフォーマンスを最適化できます。

AR/VR での作業時に 3D モデルを正しいスケールでエクスポートする

3D メッシュのプロトタイプ作成とモデリングは、好みの DCC で行うか、Package Manager にある [ProBuilder](#) を使用して Unity で直接行うことができます。レベルのモックアップを作成し、オブジェクトのスケールを調整して、Unity でどのように見えるかをテストしてエクスポートします。スケールが正しければ、さらに改良を加えて後で Unity に戻すことができます。

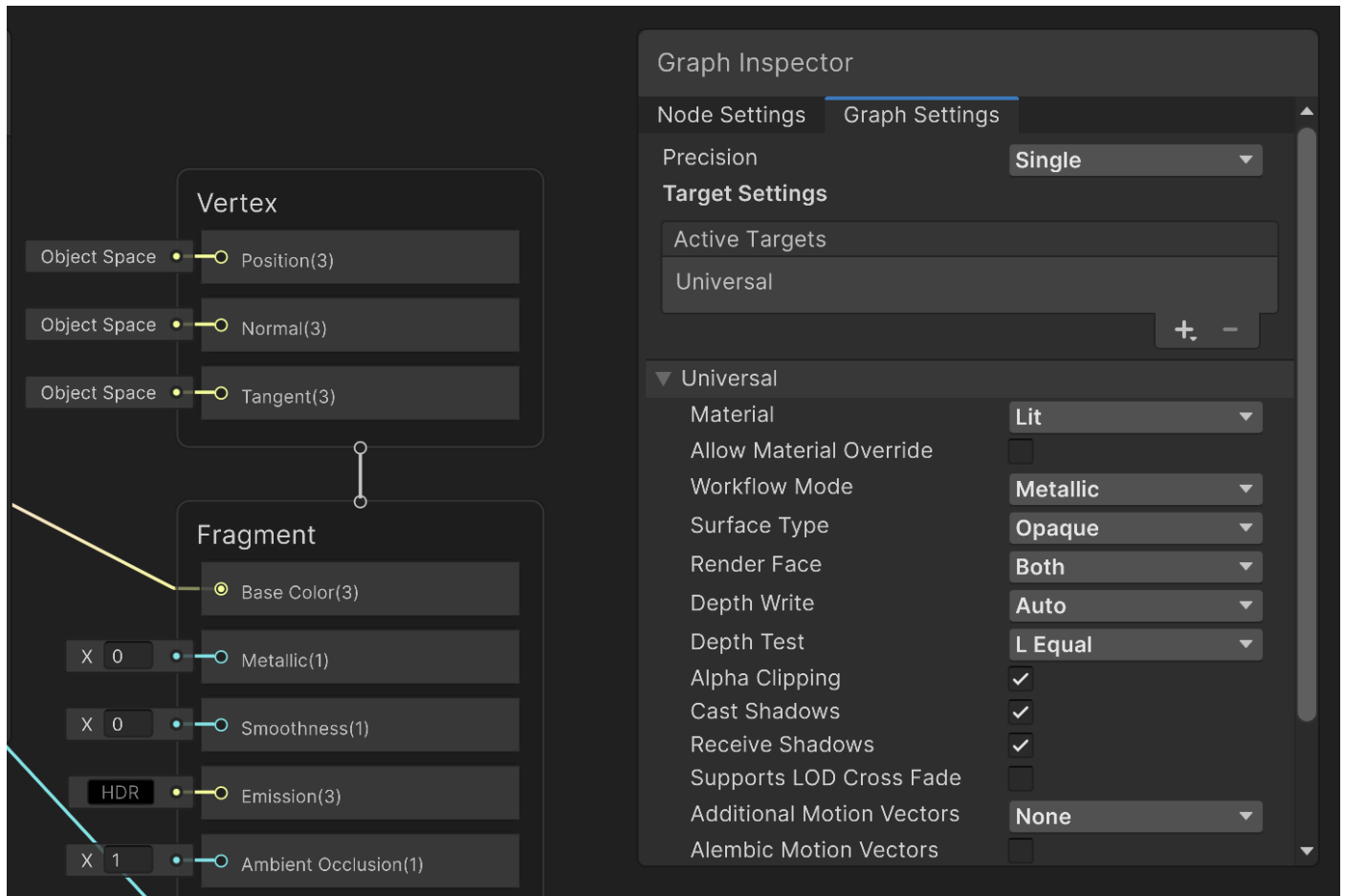
Unity では、1 単位が正確に 1 メートルに対応するよう設計されており、測定スケールを直感的に扱えます。インポートしたアセットのスケールを測定するには、シーンにキューブのゲームオブジェクトを導入するだけです。このキューブの寸法 1 m x 1 m x 1 m は、あらゆるアセットのスケールを即時かつ正確に評価する基準として機能します。環境や小道具を作成するときに重要なのは、美観と性能のバランスを取り、滑らかで魅力的な AR/VR 体験を実現することです。



ProBuilder でレベルのプロトタイプを作成し、FBX Exporter で 3D モデリングソフトウェアにエクスポートして微調整できます。

URP および HDRP プロジェクト用のシェーダーグラフ

[シェーダーグラフ](#) で Lit シェーダーや Unlit シェーダーを作成すると、URP プロジェクトと HDRP プロジェクトの両方で柔軟にアセットを利用できます。シェーダーグラフでは、マスターノードでレンダーパイプラインを定義できますが、ロジックは保持されるため、両方のパイプラインで動作するシェーダーを簡単に得られます。



マスタースタック

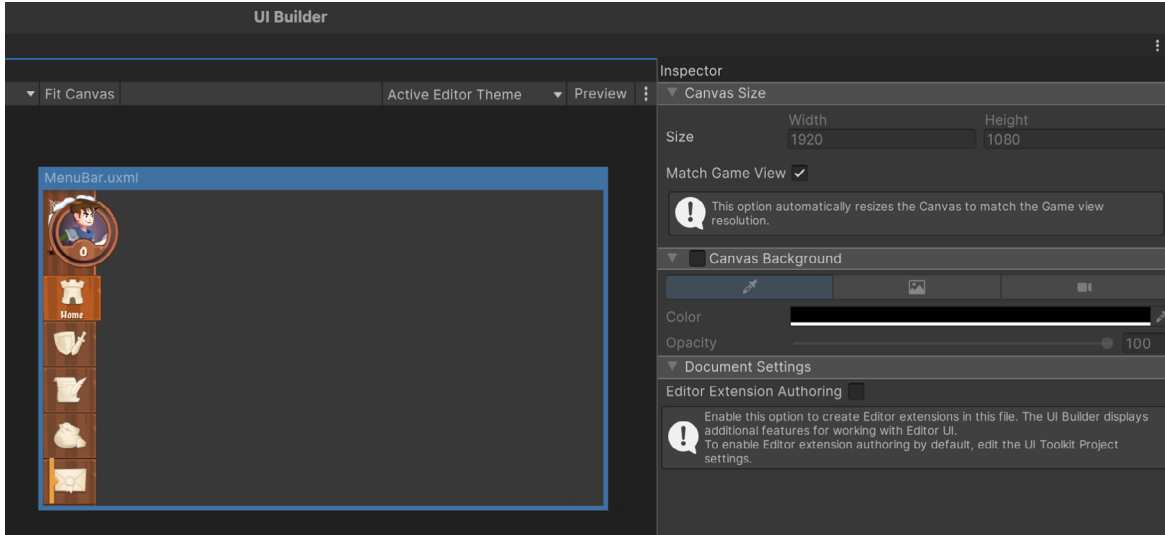
UI Toolkit

視覚的なリファレンスでインターフェースをデザインする

Canvas Background を有効にすることで、色や背景画像の上で、要素に対して視覚的にスタイリングを行いやすくなります。Hierarchy ペインで UXML ファイルを選択した後、最終的な UI に近い Canvas Background を選ぶことで、スタイルの変更を実際の使用状況に合わせて判断できます。

Canvas Background には、いくつかのオプションがあります。

- **背景色:** ゲーム環境の特定の色調や色相を表します
- **画像:** 背景として使用するスプライトやテクスチャの選択用です (モックアップ画面やリファレンスアートの再現に便利です)
- **カメラ:** 現在のゲームプレイを背景に表示し、実際のゲームのコンテキストで UI を確認できます



UXML ドキュメントの Canvas (キャンバス): Color (色) や Image (画像) のオプションを使用して見た目を調整します。

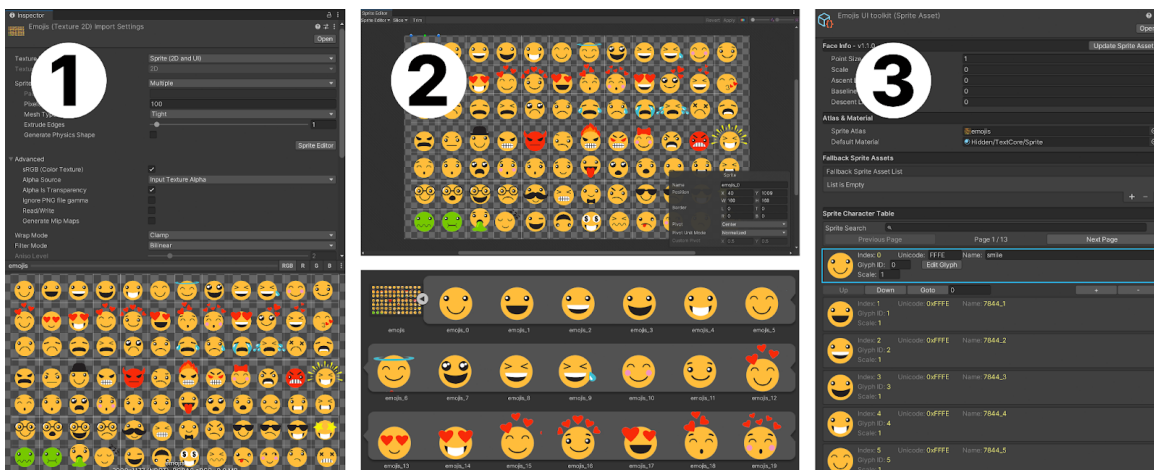
Photoshop ファイルを扱うときに PSD Importer で反復処理を迅速化する

マルチレイヤーの PSD ファイルに含まれるスプライトは、プロジェクトへのインポートに使用する PSD Importer でファイルを保存するたびに Unity によって自動的に更新されます。これにより、プレースホルダーを素早く作成し、ゲームビューで変更を確認しながら反復的に改良できます。ファイルを切り替えたり、チーム内の他の開発者からサポートを受けたりすることなく、コンテキスト内で作業を確認できるため、大幅な時間の節約になり、作業品質の向上につながります。

ゲームで絵文字を使用する

リッチテキストタグを使って絵文字のようなスプライトをテキストに加えることができます。そのためには、グラデーションアセットに似た [スプライトアセット](#) を使用します。

複数のスプライトをインポートする場合は、ドローコールを減らすために、スプライトを単一のアトラスにパックします。スプライトアトラスには、対象プラットフォームに適した解像度を設定するようにしてください。



スプライトをインポートし、Assets/Resources 内にスプライトアセットを作成し、必要に応じて各グリフの情報を調整します。

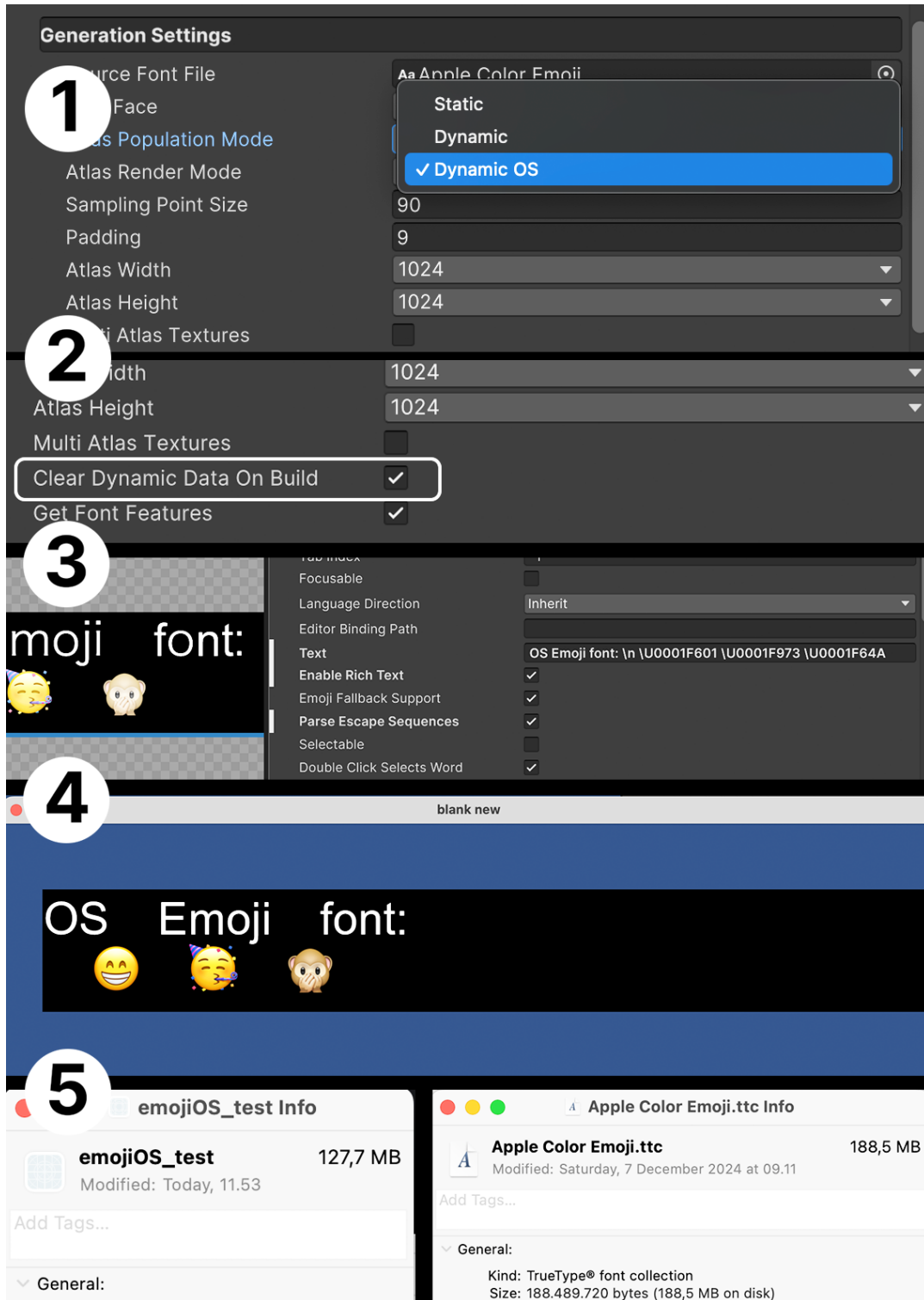


デバイスの OS に含まれているビルトインの絵文字を使用する

iOS や Android などの特定のランタイムプラットフォームを対象にしている場合は、プロジェクトにソースフォントを加える代わりに、システムにビルトインされている絵文字フォントを使用できます。これにより、メモリを節約し、絵文字の大規模なコレクションをアプリケーションにパッケージする必要がなくなります。

プロジェクトで OS の絵文字を使用する手順は、以下のとおりです。

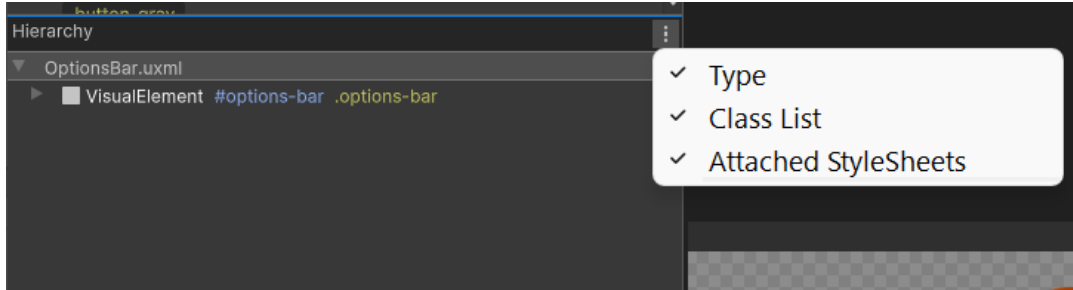
1. 対象システムが使用しているフォントからフォントアセットを作成します。iOS では Apple Emoji (この例で使用)、Android では Noto Color Emoji (現在は [COLRv0](#) のみサポート) という名称です。フォントアセットが **Color** タイプであることを確認し、Atlas Population Mode を **Dynamic OS** に設定します。これにより、アセットの格納スペースにソースフォントを置く必要がなくなります。
2. フォントアセットで **Clean Dynamic Data On Build** がオンになっていることを確認します
3. UI Builder で **Parse Escape Sequences** を有効にし、macOS や Windows の絵文字キーボードを使用するか、UTF 形式で目的の絵文字を入力します。例えば、\U0001F601 でスマイリーを入力できます。各絵文字の UTF は、フォントアセットの文字コード表で確認できます。
4. macOS で実行されるビルドでは、OS のフォントに従って絵文字が表示されます。
5. 試してみると、スタンドアロンの絵文字フォントより **ビルドサイズ** が小さいことがわかります。これは絵文字フォントがプロジェクトに含まれていないにもかかわらず、OS にビルトインされているフォントによって適切な絵文字がレンダリングされていることを示します。



UI Builder でビジュアル要素に関する追加情報を表示する

Hierarchy ヘッダーの縦ドット (:) をクリックすると、UI 要素をさらに視覚化できます。

Hierarchy ペインでは、要素タイプの横に追加情報が表示されます。選択すると、**#options-bar** 名前セクターと **options-bar** スタイルクラスセクターが表示されます。



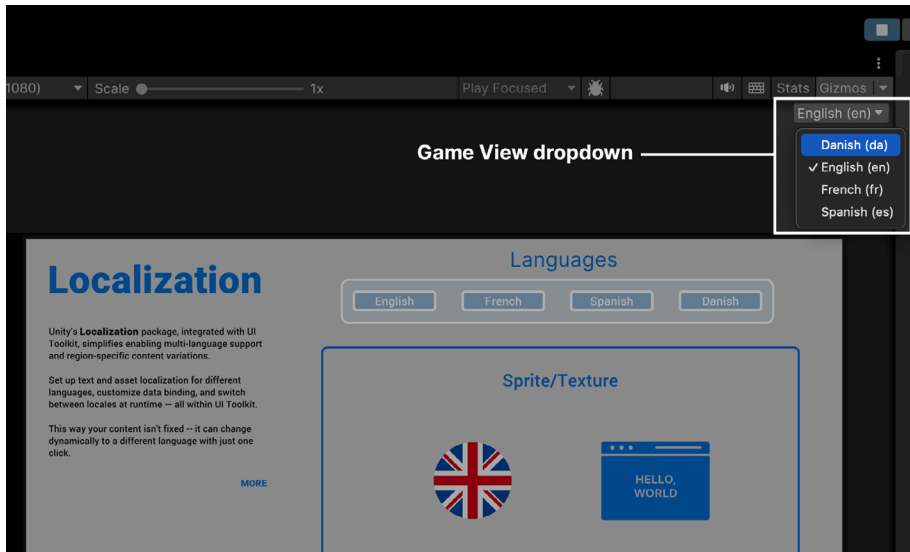
Hierarchy では、さまざまなセクターでフィルタリングを行うことができます。

一部のセクターには、プレフィックスとして **.unity-** が付いています。これらは、すべての要素に適用されるデフォルトのスタイルです。これらの値は、定義されたセクターによってオーバーライドされます。

ローカライゼーションを早い段階で統合して、より多くの市場に進出する

Localization パッケージを UI Toolkit と統合することで、ローカライゼーションプロセスを簡略化できます。この統合により、プレイヤーがどこにいても、地域固有のコンテンツを楽しめます。

ゲームビューのロケールドロップダウンを使用してさまざまな言語で UI をプレビューし、各ロケールで要素が正しく表示されるか確認します。



ローカライゼーションをプレビューするには、ゲームビューのロケールドロップダウンを使用します。

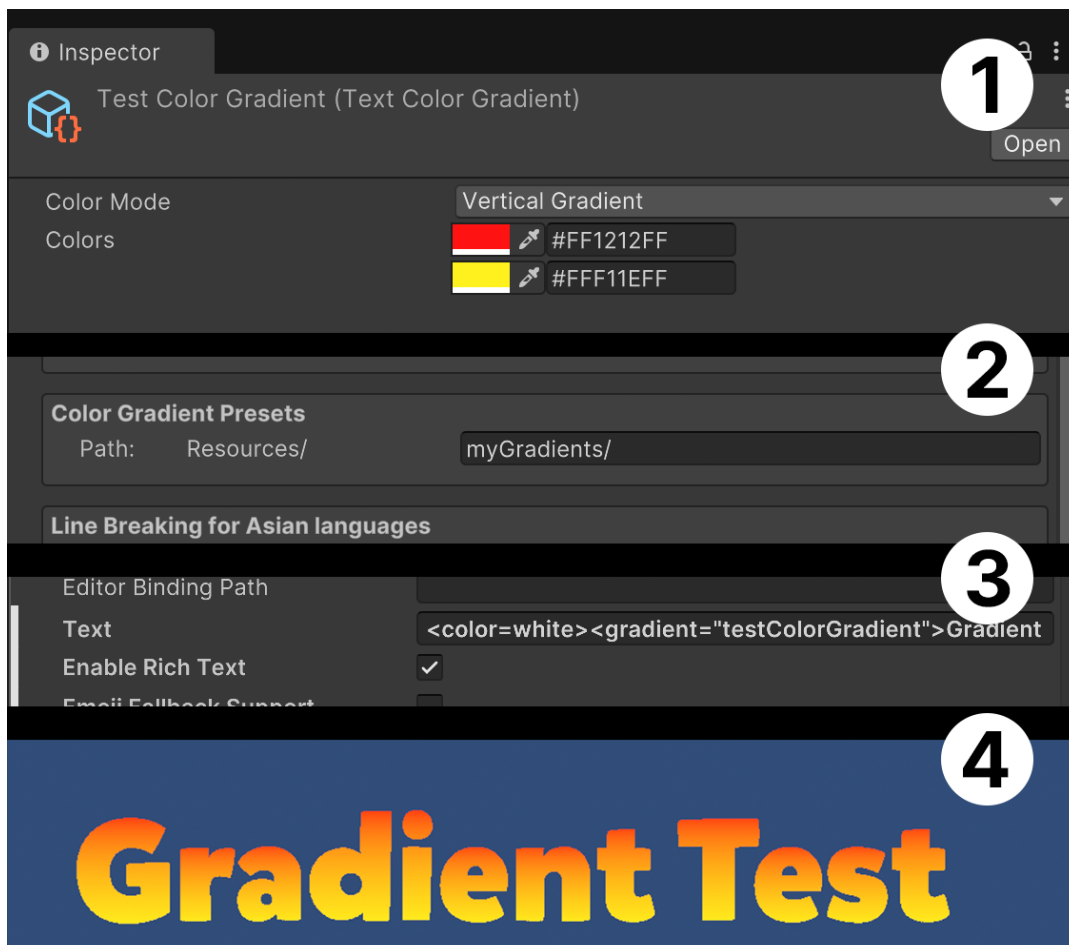
グラデーションでインターフェース全体にスタイルを加える

UI Toolkit では、<gradient> タグを使用してグラデーションを適用できます。**Rich Text** が有効になっていることを確認し、UI Builder 内またはゲームビューで、変更の効果が生じていることを確認します。

1. グラデーションカラーアセットは、**Create > Text Core > Gradient Color** で作成します。このファイルは、**Assets/Resources** または Resources のサブフォルダー内に配置してください。
2. Panel Settings から参照する **Text Settings アセット** を作成します。アセットで Color Gradient Presets を探し、アセットがあるフォルダーまたは Resources 内のサブフォルダーを指定します。
3. UI Builder 内に、リッチテキストタグ <color=white><gradient="testColorGradient">Gradient Test</gradient></color> を加えます。

このカラータグでフォントの色を白に復元することにより、グラデーションが意図したとおりの外観になります。参照するグラデーションはステップ 1 で作成したアセット名と一致する必要があります。**Rich Text** が有効になっていることを確認します。

4. UI Builder 内またはゲームビューで変更の効果が生じていることを確認できます。



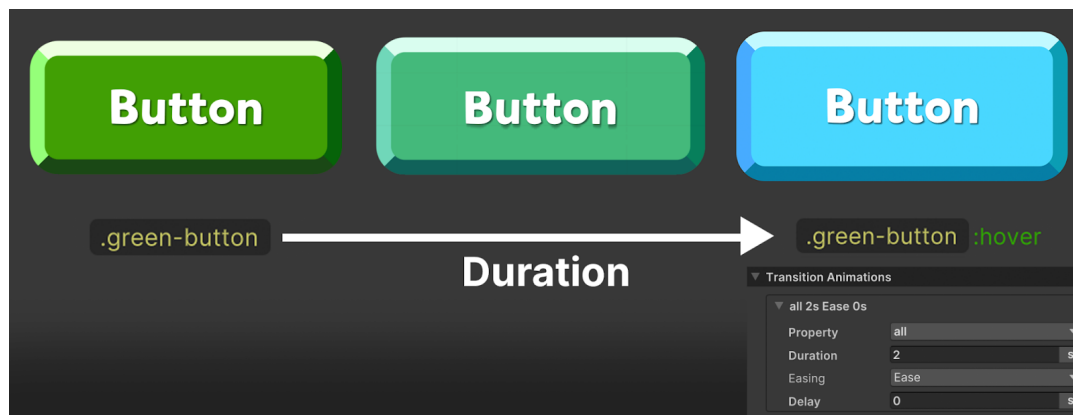
UI Toolkit での <gradient> タグの使用

USS 遷移で UI をアニメーション化する

ビジュアル要素については、疑似クラス (:active、:inactive、:hover など) で独自のセレクターを設定できるので、その他のコードは必要ありません。疑似クラスによってスタイルの変更がトリガーされると、定義された遷移が変更を自動的にアニメーション化します。

例: ホバー (:hover) やクリック (:active) でボタンが拡大縮小したり、ユーザーインタラクションやその他のイベントに応じて要素がフェードアウトしたり非表示になったりします。

こうした状態変化はユーザーのアクションによってトリガーされますが、:enabled および :disabled の疑似クラスの状態は任意に変更できるので、これらの疑似クラスに関連する USS アニメーションを手動でトリガーできます。これにより、コードから自由にアニメーションをトリガーすることができます。

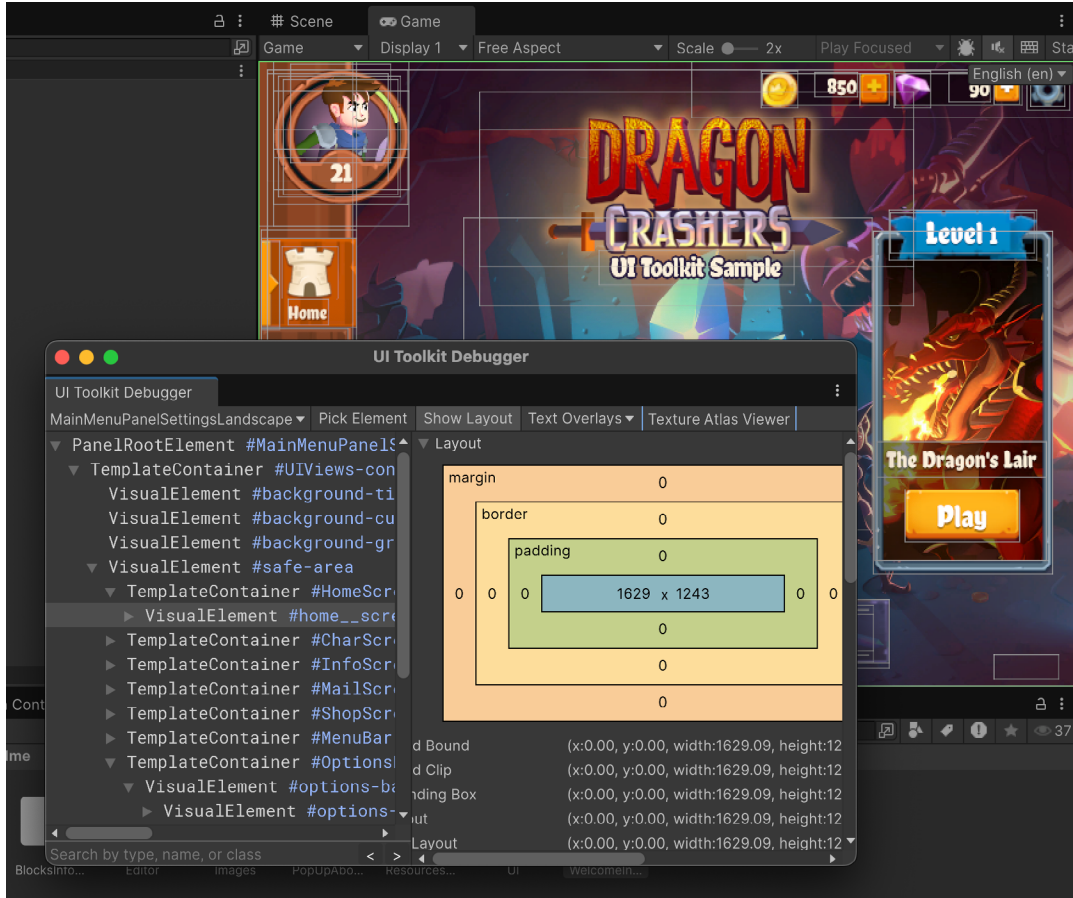


USS 遷移アニメーションの例

解決済みスタイルのバウンディングボックスをエディターで視覚化する

バウンディングボックスを使用してレイアウトやアラインメントの問題を特定し、エディター内で対話的に UI 構造をデバッグできます。

UI Toolkit を使用して解決済みバウンディングボックスを視覚化するには、**Window > UI Toolkit > Debugger** に移動して、**UI Toolkit Debugger** を開きます。次に、**Pick Element** ツールを使用して UI 要素を選択し、**Show Layout** 機能を有効にします。このオプションは、Game ビューの UI の上にバウンディングボックスとレイアウト情報を直接表示します。

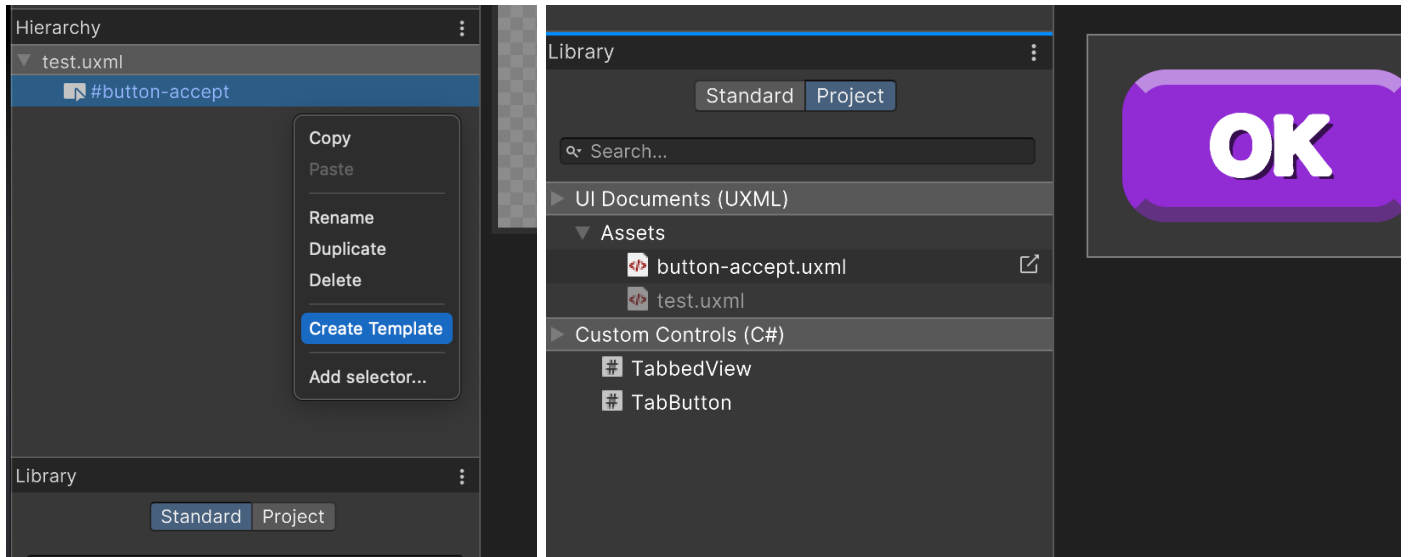


UI Toolkit Debugger には、UI のデバッグに役立つ便利な機能が備わっています。

UXML ファイルをテンプレートとして再利用しワークフローを迅速化する

UXML ファイルはプレハブと同じように使用できます。例えば、インベントリ内で何度もスポーンする必要があるアイテムアイコンとカウント番号を含む UXML レイアウトのプロジェクトがあるとしたら。

UXML を右クリックすると、テンプレートを作成するオプションが表示されます。テンプレートは、後で Hierarchy ペインのほかのビジュアル要素に加えたり、コードからインスタンス化したりできます。作成されたファイルは、Library と Project ウィンドウで確認できます。



テンプレートは再利用可能な UXML であり、Library ペインの Project タブで利用できます

その他のリソース

さまざまなデバイスで UI Toolkit を使用して UI を作成する詳細な手順については、e-book『[Create scalable and performant UI with UI Toolkit in Unity 6](#)』(Unity 6 で UI Toolkit を使用したスケーラブルで高性能な UI の作成) をダウンロードしてください。

この e-book で参照しているサンプルプロジェクト [UI Toolkit Sample –『Dragon Crashers』](#)は Unity Asset Store から入手できます。UI Toolkit サンプルは、ユーザー自身のアプリケーションで UI Toolkit を活用する方法を示しています。このデモでは、ミニ RPG の 2D プロジェクト『Dragon Crashers』の一部で、Unity 6 UI Toolkit ワークフローをランタイムに使用したフル機能のインターフェースを使用しています。

さらに、Unity Asset Store の [QuizU プロジェクト](#) とその関連記事もぜひ確認してください。

- [UI Toolkit サンプルプロジェクト QuizU](#)
- [QuizU: State patterns for game flow](#) (ゲームフローの状態パターン)
- [QuizU: Managing menu screens in UI Toolkit](#) (UI Toolkit のメニュー画面管理)
- [QuizU: The Model View Presenter pattern](#) (モデルビュープレゼンターパターン)
- [QuizU: Event handling in UI Toolkit](#) (UI Toolkit でのイベント処理)
- [QuizU: UI Toolkit performance tips](#) (UI Toolkit のパフォーマンスに関するヒント)

開発者ワークフロー

Awaitable クラス

Unity 6 では [Awaitable](#) クラスが導入されました。これは Unity 内での C# async/await ワークフローをサポートするために特別に設計された、軽量で割り当てを伴わない型です。コルーチンに代わる、性能に優れた選択肢を提供します。これにより、Unity のフレームベースの更新サイクルと自然に統合された非同期コードの記述が容易になります。

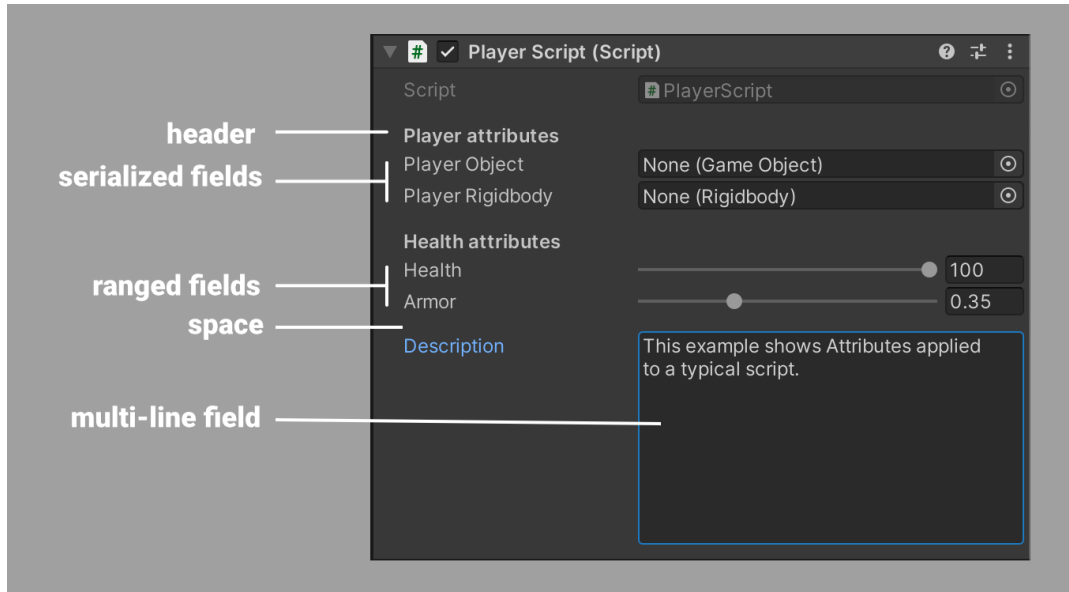
```
using UnityEngine;
using System.Threading.Tasks;

public class LogWithDelay : MonoBehaviour
{
    private async void Start()
    {
        Debug.Log("Message 1");
        await Task.Delay(1000); // 1 秒待つ

        Debug.Log("Message 2");
        await Task.Delay(1000); // もう 1 秒待つ
    }
}
```

属性で Inspector ウィンドウを強化する

Unity には、Inspector でヘッダー、間隔、範囲フィールドを作成するなどの特別な動作を示すために、クラス、プロパティ、関数の上に配置できるさまざまな属性があります。



Inspector フィールドに影響する属性

C# では、属性名は角カッコでくくります。これらはスクリプトに記述できる属性のうちよく使われるものです。

属性	説明	例
SerializeField	非公開フィールドをシリアル化し、Inspector で表示できるようにします。	<code>[SerializeField]</code> <code>private GameObject m_myObject;</code>
Range	float や int 変数を特定の範囲に限定します。Inspector にスライダーとして表示します。	<code>[Range(1,6)]</code> <code>public int IntegerRange;</code> <code>[Range(0.2f, 0.8f)]</code> <code>public float m_floatRange;</code>
HideInInspector	変数をシリアル化しますが、Inspector では非表示です。	<code>[HideInInspector]</code> <code>public Int p = 5;</code>

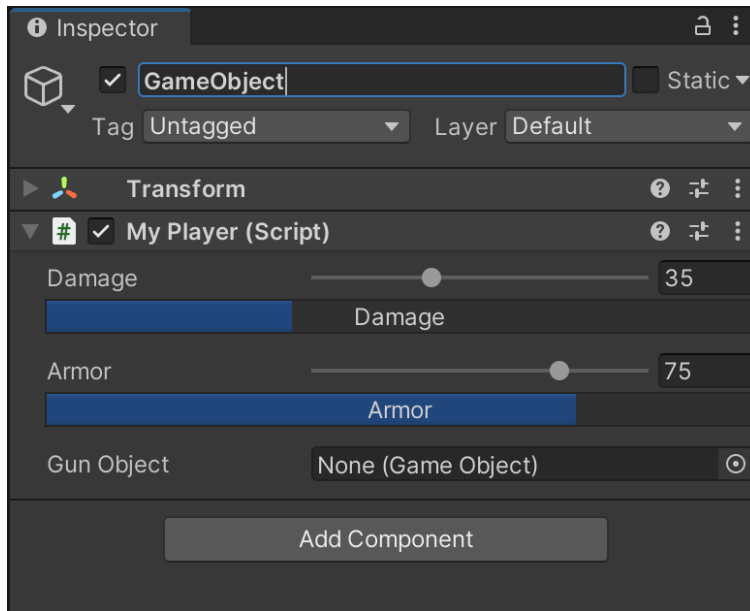
RequireComponent	依存する必要なコンポーネントが自動的に加えられ、設定エラーを回避します。 注意: この属性では、コンポーネントがゲームオブジェクトに加えられた瞬間のみチェックします。	<pre>// PlayerScript はゲームオブジェクトに Rigidbody が必要 [RequireComponent(typeof(Rigidbody))] public class PlayerScript:Monobehaviour { private Rigidbody m_rBody; void Start() { m_rBody = GetComponent<Rigidbody>(); } }</pre>
Tooltip	Inspector のフィールド上にマウスポインタを置くと、ツールチップが表示されます。	<pre>public class PlayerScript:Monobehaviour { [Tooltip("Health value between 0 and 100.")] int m_health = 0; }</pre>
Space	フィールド間に小さな余白を挿入し (追加テキストなし)、フィールドを視覚的に分離します。	<pre>[Space(10)] // 10 ピクセルの余白を挿入 int p = 5;</pre>
Header	太字のテキストと余白を挿入し、Inspector で変数を整理するために役立ちます。この属性は、グループ化する最初のフィールドにのみ記述します。	<pre>public class PlayerScript:Monobehaviour { [Header("Health Settings")] private int m_health = 0; private int m_maxHealth = 100; [Header("Shield Settings")] private int m_shield = 0; private int m_maxShield = 0; }</pre>

Multiline	文字列を複数行のテキストフィールドで編集できるようにします。任意の int を渡して行数を指定します。 ヒント: スクリプトに注釈を付けて自分自身や他のユーザーへのメモとして使用します。	<pre>[Multiline] public string textToEdit; [Multiline(20)] public string m_moreTextToEdit;</pre>
SelectionBase	子オブジェクトにメッシュが含まれている可能性がある空のゲームオブジェクトを選択する場合に便利です。ベースオブジェクトの任意のコンポーネントに属性を加えます。エディターでオブジェクトを選択する際、子ではなく [SelectionBase] 属性を持つゲームオブジェクトが選択されます。	<pre>// これをベースのゲームオブジェクトに加える [SelectionBase] public class PlayerScript: MonoBehaviour { } </pre>
ColorUsage	[ColorUsage] 属性を使用すると、色フィールドで選択できる色を制御できます。パラメーターに応じて、HDR を有効にしたり、アルファチャンネルを無効にしたりできます。	<pre>public ColorUsageAttribute(bool showAlpha, bool hdr, float minBrightness, float maxBrightness, float minExposureValue, float maxExposureValue);</pre>
RunOnce	プロジェクト開始時に関数を一度だけ自動的に実行する場合に使用します。静的な標準と [RuntimeInitialization] 属性を一緒に使用することで、簡単に実現できます。ブートローダーのように 1 回だけ実行するプロジェクト設定ロジックで使用します。QuizU サンプルでも示されています。	<pre>public RuntimeInitializeOnLoadMethodAttribute(RuntimeInitializeLoadType loadType);</pre>

利用可能な多くの属性のほんの一例です。変数の値を保ったまま変数名を変更することも、あるいは、空のゲームオブジェクトを用意せずにロジックを呼び出すこともできます。独自の `PropertyAttribute` を作成して、スクリプト変数のカスタム属性を定義することもできます。属性の完全なリストについては、スクリプティング API を参照してください。

独自のカスタムウィンドウと Inspector を作成する

Unity の最も強力な機能の 1 つは、拡張可能なエディターです。[UI Toolkit](#) パッケージを使用して、カスタムウィンドウや [カスタム Inspector](#) などのエディター UI を作成することを推奨します。



カスタムエディターにより、Inspector での MyPlayer スクリプトの表示方法が変更されます。

UI Toolkit または [IMGUI](#) を使用してカスタムエディタースクリプトを実装する方法の詳細については、[ユーザーインターフェース \(UI\) の作成](#) を参照してください。[UI Toolkit](#) の簡単な紹介については、『[Getting Started with Editor Scripting](#)』(エディタースクリプティングの開始) チュートリアルを参照してください。

カスタムメニューを作成する

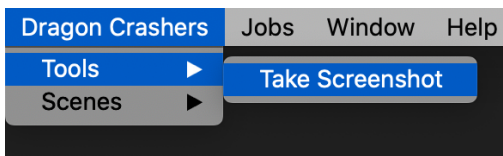
Unity には、エディターのメニューとメニュー項目を簡単にカスタマイズできる **MenuItem** 属性があります。これはスクリプト内の任意の静的メソッドに適用できます。

プロジェクトで頻繁に使用する機能がある場合は、メニュー項目にまとめます。これにより、1 つの PropertyAttribute モディファイアだけで基本的なユーザーインターフェースを構築できます。

```

1  using UnityEditor;
2  using UnityEngine;
3  using System;
4  using System.IO;
5
6  public class ScreenshotTaker
7  {
8      [MenuItem("Dragon Crashers/Tools/Take Screenshot")]
9      public static void TakeScreenshot()
10     {
11         if (!Directory.Exists("Screenshots"))
12             Directory.CreateDirectory("Screenshots");
13
14         ScreenCapture.CaptureScreenshot(string.Format("Screenshots/{0}.png",
15             DateTime.Now.ToString("yyyy-MM-dd HH.mm.ss")));
16     }
17 }
18

```



MenuItem 属性により、静的メソッド (Take Screenshot) がアタッチされた単純なインターフェースが作成されています。

再生モードへの遷移の迅速化

再生モードに入ると、ビルドと同じようにプロジェクトが起動し、実行されます。再生モード中にエディターで行った変更は、再生モードを終了するとリセットされます。

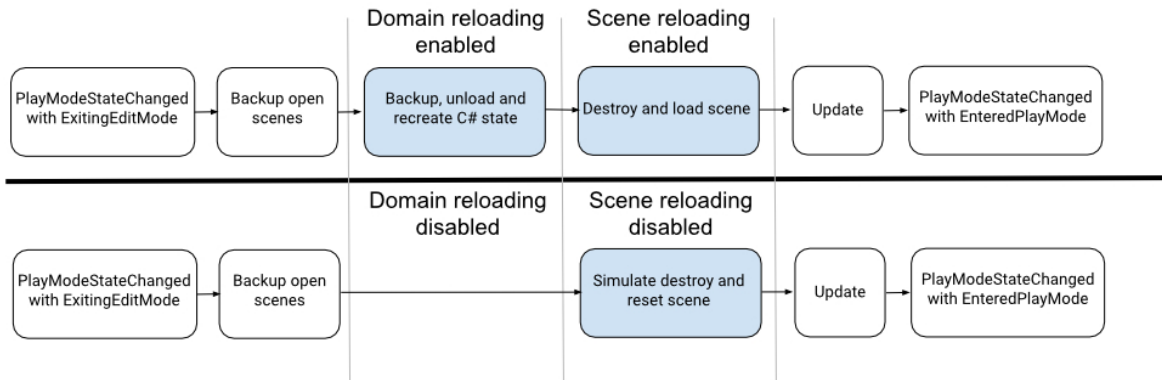
Unity は再生モードに入るたびに以下の 2 つの重要なアクションを実行します。

- **Domain Reload** (ドメインの再ロード): スクリプトの状態をバックアップ、アンロード、再作成します。
- **Scene Reload** (シーンの再ロード): シーンを破棄し、再度ロードします。

この 2 つのアクションは、スクリプトやシーンが複雑になればなるほど時間がかかります。

これ以上スクリプトを変更する予定がない場合は、**Enter Play Mode Settings (Edit > Project Settings > Editor)** を使用すると、コンパイル時間を少し節約できます。ドメインの再ロード、シーンの再ロード、またはその両方を無効にするオプションが用意されています。これにより、再生モードの開始と終了を迅速化できます。

スクリプトをさらに変更する予定がある場合は、ドメインの再ロードを有効にする必要があることに注意してください。同様に、シーンの階層を変更した場合は、シーンの再ロードを有効にする必要があります。それらを有効にしない場合、予期しない動作が発生する可能性があります。



ドメインの再ロードとシーンの再ロードを無効にした場合の影響。

デフォルトのスクリプトテンプレートをカスタマイズする

新しいスクリプトを作成するたびに同じ変更を加えていることに気づきましたか？ 直感的に名前空間を加えたり、更新イベント関数を削除したりしていませんか？ 好みの開始点にスクリプトテンプレートを設定することで、キー入力を減らし、チーム全体で一貫性を保つことができます。

新しいスクリプトまたはシェーダーを作成するたびに、Unity は **%EDITOR_PATH%\Data\Resources\ScriptTemplates** に保存されているテンプレートを使用します：

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

また、シェーダーやその他のビヘイビアスクリプト、アセンブリ定義のテンプレートも用意されています。

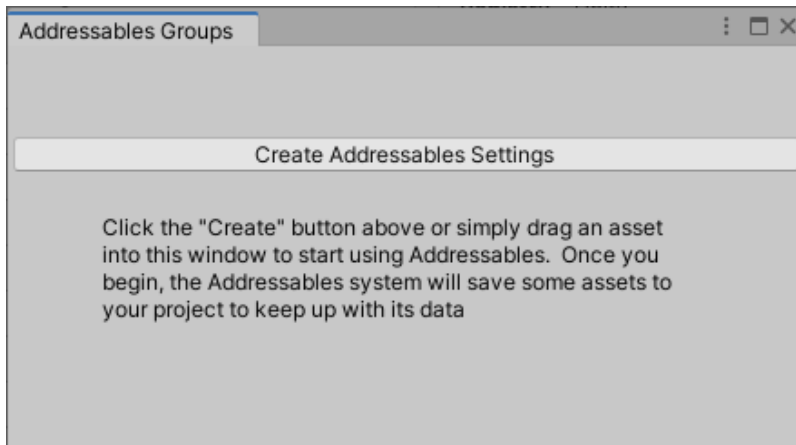
プロジェクト固有のスクリプトテンプレートの場合は、Assets/ScriptTemplates フォルダーを作成します。スクリプトテンプレートをこのフォルダーにコピーして、デフォルトをオーバーライドします。

また、すべてのプロジェクトのデフォルトのスクリプトテンプレートを直接変更することもできますが、変更する前に必ずオリジナルをバックアップしてください。

Addressables を使用してプレイヤーにオンデマンドでコンテンツを配布する

Addressables とアセットバンドルは、ゲームを論理ブロックで構造化するための強力なツールです。これらのブロックは個別にエクスポートでき、必要に応じてメインの実行ファイルに加えることができます。アセットのロードやアンロード、アセットバンドルの設定、ビルド、ロードに使用され、その後オンデマンドでプレイヤーに

配布できます。Addressables システムはアセットバンドルの上に構築されており、依存関係の解決とバンドルのロードを行います。



Unity プロジェクトでの Addressables システムの初期化前

Addressables を初めて使用する場合は、Unity ドキュメントの [Get started](#) (スタートガイド) ページを確認してください。

効果的なアセット管理のヒント:

- Addressables を最初から活用し、すべての新しいアセットが Addressable として登録されていることを確認します。
- アセットをタイプ別に整理するのではなく、ロードされて一緒に使用される頻度でグループ化することを目指します。これにより、ランタイムのメモリ使用量が改善され、起動時間が短縮され、その結果、ゲームのリテンションも向上します。
- 小さなバンドルを目指します。依存関係チェーンが短くなり、ランタイムメモリ使用量が減るからです。

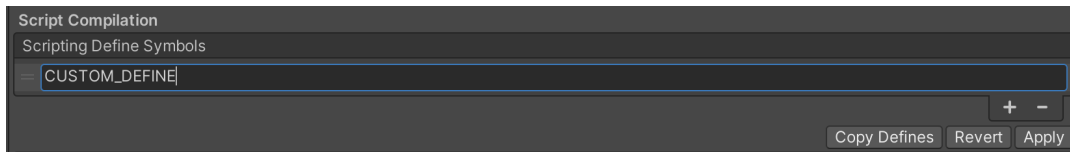
詳細は、[Effective asset management in Unity with Addressables](#) (Unity における Addressables を使用した効果的なアセット管理) の記事を参照してください。

プリプロセッサディレクティブを使用して条件付きコンパイルされたコードを作成する

[プラットフォーム依存コンパイル](#)機能を使用すると、ターゲットプラットフォーム、Unity バージョン、スクリプティングバックエンドに基づいて、条件付きでコードをコンパイルし実行できます。

これは、クロスプラットフォームのコードを書くときや、デバイス固有の動作に合わせて最適化するとき、バージョン固有の API を管理するときに便利です。

エディターでテストするときは、独自のカスタム #define ディレクティブを指定することができます。[Player](#) 設定の **Other Settings** パネルを開き、**Scripting Define Symbols** に移動します。



Script Compilation の Scripting Define Symbols

ScriptableObject を使用してデータとロジックを分離する

ScriptableObject を使用することが、データとロジックを分離し、クリーンなコーディング手法を推進するために役立ちます。つまり、意図しない副作用を引き起こさずに変更を加えやすくなり、テストのしやすさとモジュール性が向上します。また、アーティストやデザイナーなど、プログラマー以外の人と共同作業を行う場合にも便利です。彼らはコードに触れることなくゲームデータを編集できます。

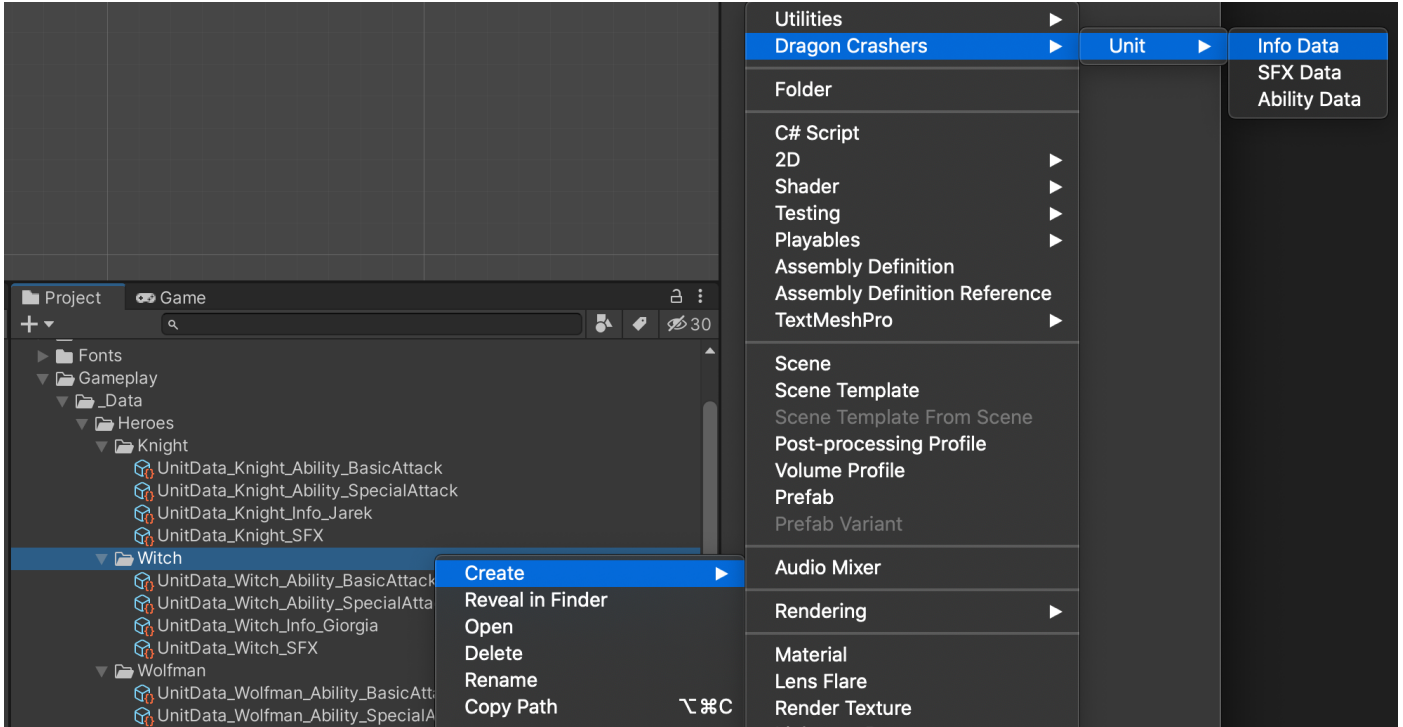
『[Dragon Crashers](#)』は典型的な使用例です。**UnitInfoData** クラスは **ScriptableObject** を継承します。各インスタンスには、ユニットの名前、スプライト、体力の設定が含まれています。このデータはゲームプレイ中も一定なので、ScriptableObject 内に格納するのに特に適しています。

```
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;

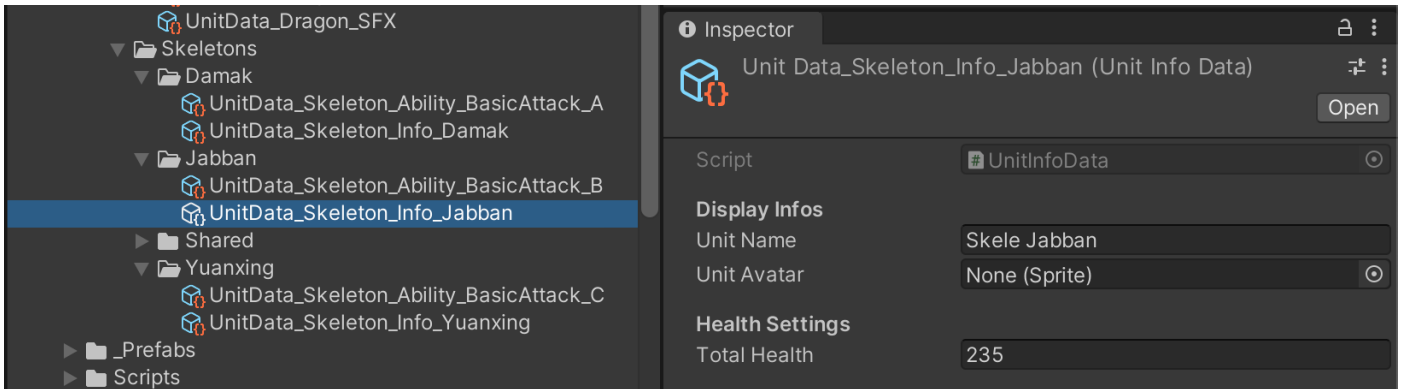
        [Header("Health Settings")]
        public int totalHealth;
    }
}
```

ScriptableObject はデータコンテナオブジェクトを定義します。



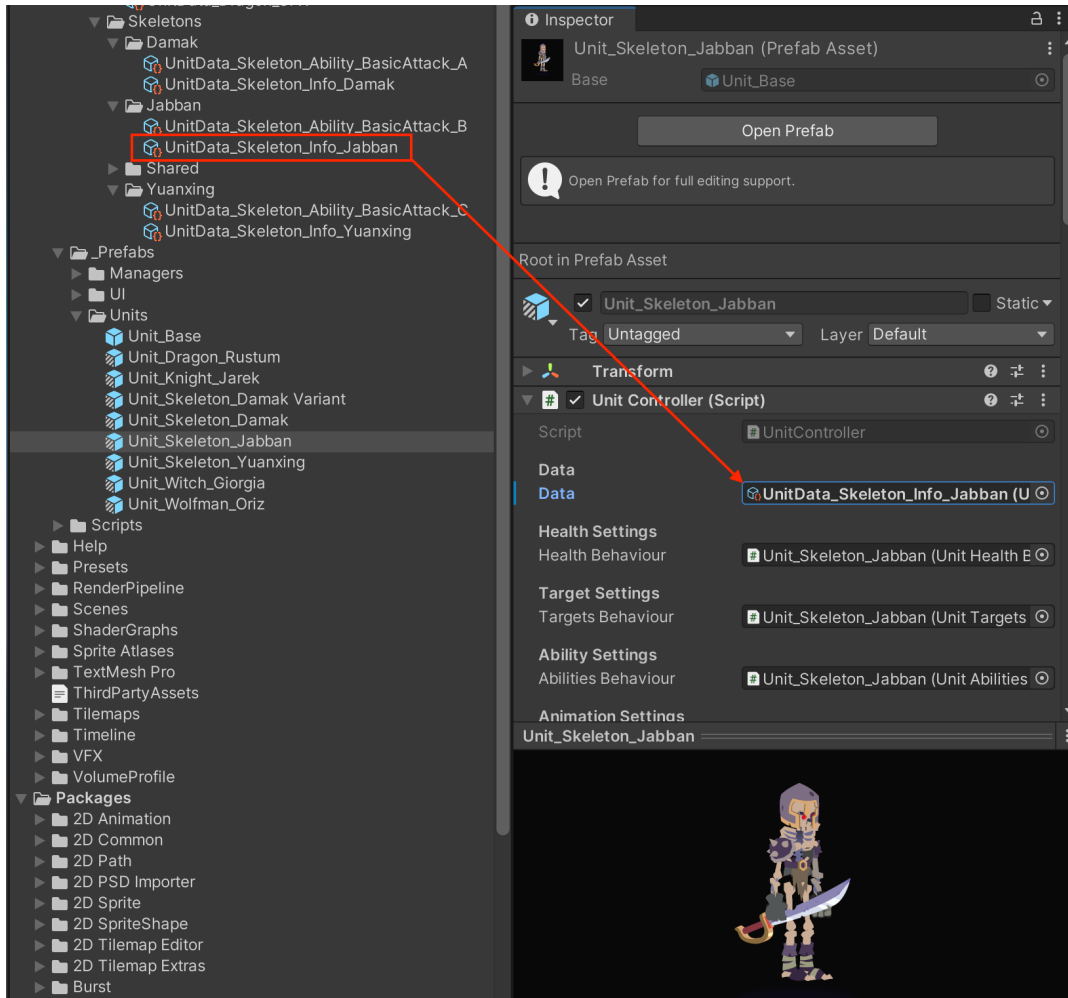
CreateAssetMenu 属性は、ScriptableObject アセットの生成に役立つコンテキストメニュー項目を生成します。各ユニットには、サウンドエフェクトや特殊能力用の追加の ScriptableObject があります。

Project ウィンドウで作成されたアセットは、Inspector を使用して正しい値を入力できます。Unit Name (ユニット名)、Unit Avatar (Sprite) (ユニットアバター (スプライト))、Total Health (総体力) といったものです。

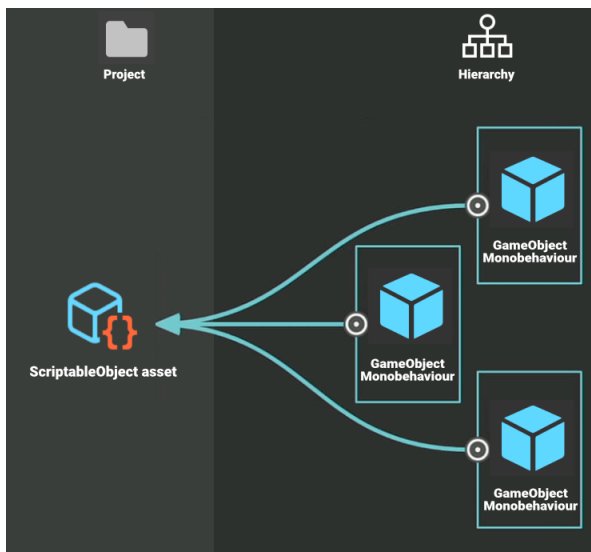


Inspector を使用して、ScriptableObject アセットの値を入力します。これらの値はゲームプレイ中に変化しません。

ゲームオブジェクト (この場合は UnitController など) は、ScriptableObject アセットを参照できます。シーンが突然ユニットでいっぱいになっても、ScriptableObject アセットのデータは複製されないため、メモリを節約できます。



Monobehaviour オブジェクト (上図の UnitController) が、プロジェクト内の ScriptableObject データアセットを参照しています。



ScriptableObject でメモリを節約し、整理整頓します。ゲームオブジェクトが多数ある場合でも、プロジェクトの資産に静的データと設定を一度定義するだけです。

シーンにプレハブのインスタンスを 1,000 個加えたとしても、アセットに格納されている同じデータを参照します。値のセットを 1 回設定すれば、一貫性が保証されます。

ゲームの規模が大きくなり、ユニットの種類が増えても、ScriptableObject アセットをさらに作成して、適切に差し替えるだけです。一元的に保存されたアセットを調整するだけで、ゲームプレイデータを維持できます。

ScriptableObject は、ゲームプレイ中に変化する可能性があるアプリケーションのセーブファイルの [永続データ](#) 保持の代わりにはなりません。静的なゲームプレイ設定やデフォルト値を保存するためにより適したワークフローです。JSON や XML のデータ解析とは異なり、ScriptableObject アセットを読み込んでもガベージは発生しません (おまけに、より高速です)。

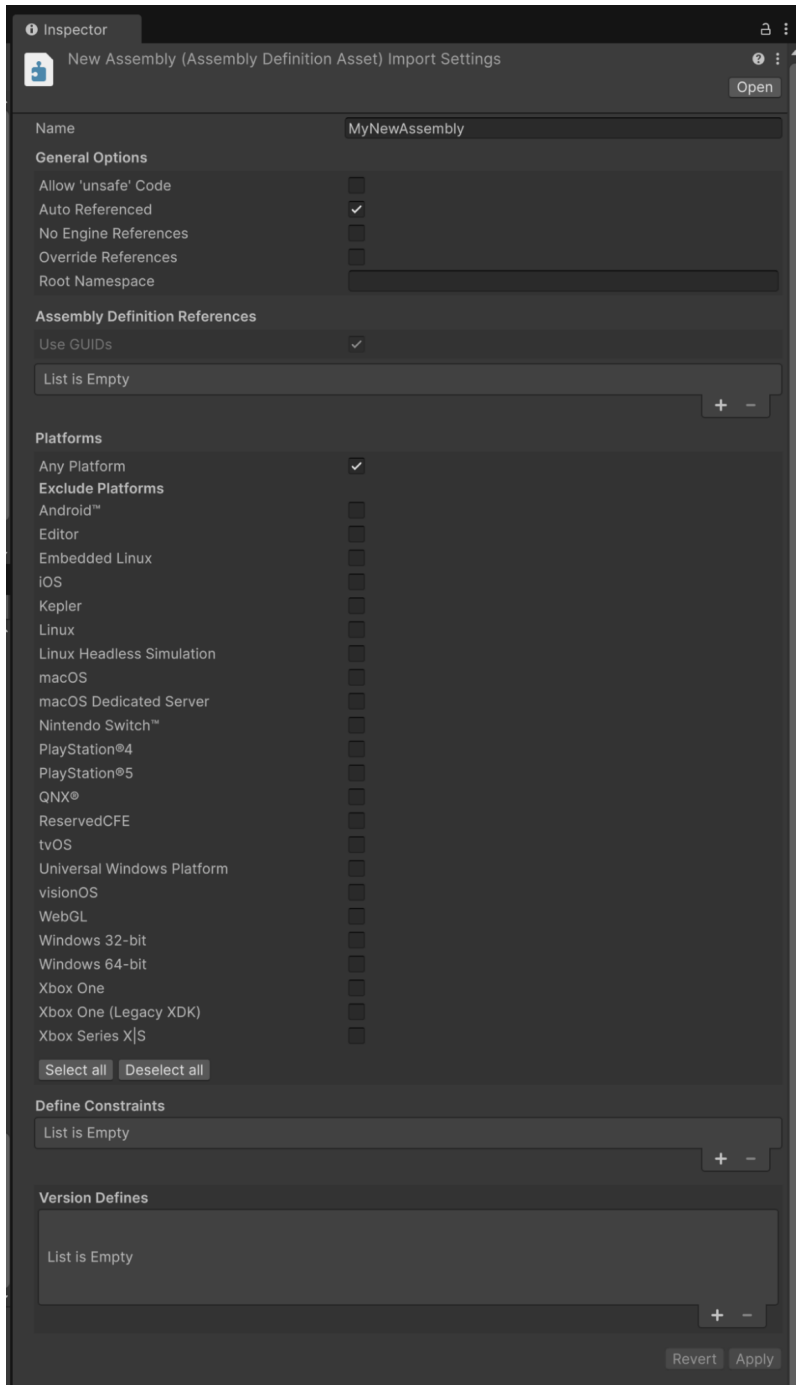
ScriptableObject に関するその他のリソース:

- [Create modular architecture in Unity with ScriptableObjects \(Unity で ScriptableObject を使用してモジュラー・ゲーム・アーキテクチャを作成する\)](#)
- [ScriptableObjects Paddle Ball デモプロジェクト](#)
- [ScriptableObject のドキュメント](#)

Assembly Definition を使用してスクリプトのモジュール化を促進する

アセンブリは、関連する型とリソースを単一の論理的な単位にまとめた、コンパイル済みの C# コードライブラリです。Unity では、Assembly Definition (アセンブリ定義) ファイル (.asmdef) を使用してアセンブリを管理できます。スクリプトをカスタムアセンブリにまとめることで、モジュール性と再利用性を促進し、コンパイル時間を短縮できます。これにより、スクリプトがデフォルトのアセンブリに自動的に加えられなくなり、他のスクリプトへのアクセスを制限できます。

アセンブリ定義の使用によりプロジェクトをクリーンアップしていてエディタースクリプトがビルドに入ってしまう場合は、Editor フォルダーに Assembly Definition を作成し、Platform として Editor のみを選択します。



Inspector の Assembly Definitions 設定

Input System にアップグレードする

まだアップグレードしていない場合は、[Input System パッケージ](#)を確認してください。Input Manager よりも新しく、より柔軟なシステムで、あらゆる種類の入力デバイスを使用して Unity コンテンツを制御できます。これは“Input System パッケージ”または単に“Input System”と呼ばれます。また、リバインド可能な制御、入力アクションアセット、入力とゲームプレイロジックの明確な分離もサポートされており、古い機能のシステムに比べて大きな利点があります。

はじめに以下のリソースを確認してください。

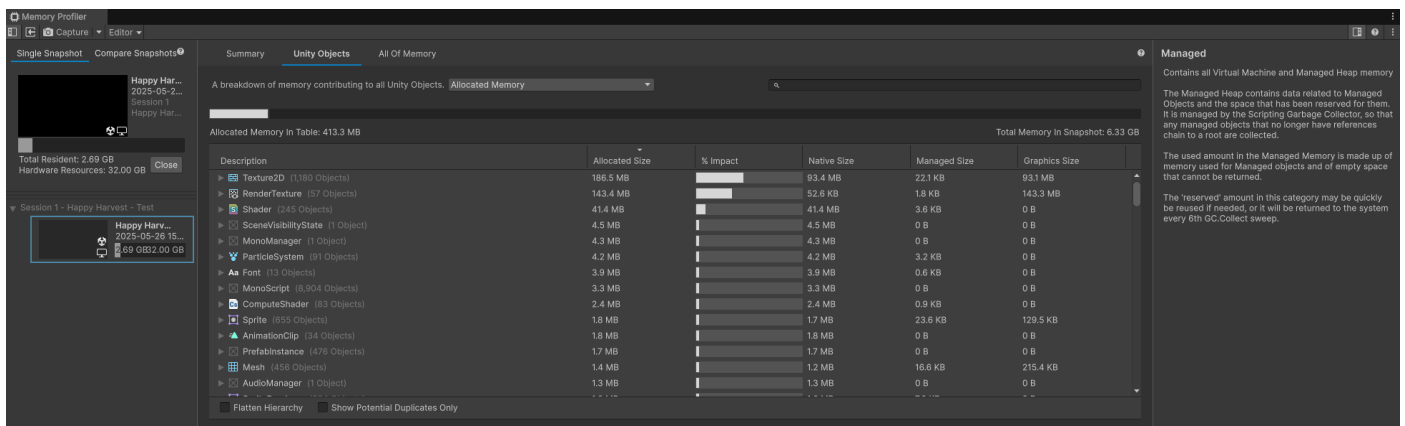
- [Prototype mobile games faster with the Input System in Unity 6 | Unite 2024](#) (Unity 6 の Input System でモバイルゲームのプロトタイプ作成を加速 | Unite 2024)
- [Get up and running with the Input System](#) (Input System を使い始める)
- [Unity Input System 7-video tutorial series](#) (Unity Input System - 7 本の動画チュートリアル)

プロファイリングツール

Memory Profiler でメモリパフォーマンスを最適化する

[Memory Profiler](#) を使用すると、プロジェクトのメモリ使用状況をキャプチャして分析でき、メモリリークの特定、メモリスパイクの抑制、ランタイム性能の最適化に役立ちます。重要な瞬間（シーンのロード、長時間のプレイセッション後など）にメモリスナップショットを取得し、それらを比較して適切にリリースされていないオブジェクトを追跡します。

コードでリソースを作成するときは、Memory Profiler で識別できるよう名前を付けることを習慣化してください。また、メモリリークを避けるために、割り当てたものはすべて忘れずにリリースしてください。



Memory Profiler のスナップショット

ProfilerMarker を使用して性能に影響するコードを特定する

BehaviourUpdate のような一般的なマーカーの下に集計されたパフォーマンスデータを確認するだけでなく、特定の関数の正確な実行時間を個別に測定できます。[ProfilerMarker](#) を使用してスクリプトコードブロックにマークを付けることで、プロファイリングの詳細度を高めることができます。その情報は Profiler の CPU Usage に表示され、Unity Recorder でもキャプチャできます。特定のコードセクションのどこで時間が費やされているかを詳細に把握できるため、性能のボトルネックを特定し、コードを最適化することが容易になります。

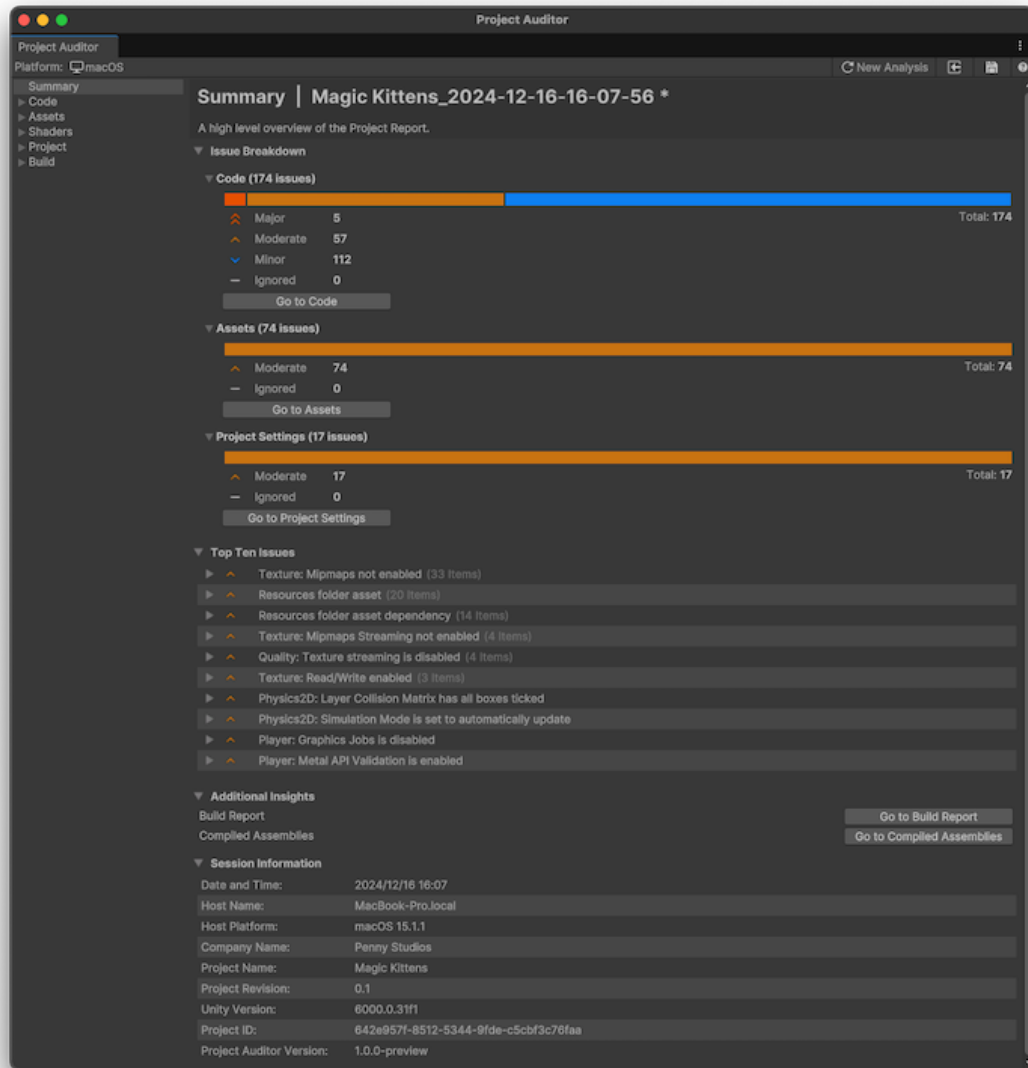
```
using UnityEngine;
using Unity.Profiling;
public class UnityTips : MonoBehaviour {
    private static readonly ProfileMarker SetupProfileMarker = new ProfileMarker("Setup");
    private static readonly ProfileMarker ExpensiveProfileMarker = new ProfileMarker("Expensive");

    public void UpdateLogic() {
        SetupProfileMarker.Begin();
        // パフォーマンス負荷の高い要素、初期化子などを設定する
        SetupProfileMarker.End();
        using (ExpensiveProfileMarker.Auto()) { //これは自動的に開始および終了する
            // ここでは、高負荷なものを指定する。
        }
    }
}
```

プロジェクトのパフォーマンス監査を実施する

[Project Auditor](#) (Unity 6.1 ではパッケージとして導入) は、プロジェクトのパフォーマンス分析、ベストプラクティスの維持、潜在的な問題やボトルネックの特定に役立ちます。

数回のクリックでプロジェクト全体をスキャンして、大量のスクリプト呼び出し、未使用アセット、過剰なエンティティ数など、非効率に関する詳細なレポートを得ることができます。



Project Auditor の Summary ビュー

生成されるレポートは、エラー、警告、情報など、重大度別に分類されます。これにより、メモリの過剰割り当てや過剰なガベージコレクションなど、エラーや警告への対処にまず集中できます。

Project Auditor は、開発の重要な段階（マイルストーン、ベータ版リリース、最終ビルドなどの前）で実行することを推奨します。そうすることで、性能のボトルネック、未使用アセット、古いコードを早期に発見でき、プロジェクトの規模拡大に伴って問題も大きくなってしまいうことを防げます。

Project Auditor は、カスタムルールとフィルターを使用してカスタマイズできます。例えば、未使用や実験的な使用が想定されている特定のスクリプトやアセットを分析から除外したり、ビルドターゲット、解像度、テキスト圧縮、その他のプロジェクト設定に特定のルールを設定して、それらが“バジェット”に合わせて最適化されるようにしたりできます。

アニメーションカーブ

カスタム lerp 関数で補間を制御する

`Mathf.Lerp(a, b, t)` は補間係数 `t` を 0 と 1 の間に制限します。つまり、`a` と `b` の範囲外の値は返しません。オーバーシュート (`t > 1`) またはアンダーシュート (`t < 0`) の値が必要な場合は、代わりに `Mathf.LerpUnclamped(a, b, t)` を使用します。これにより、補間を完全に制御し、外挿や慣性を伴う動きなどの効果を実現できます。

```
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()
    {
        // t を [0, 1] の範囲に制限
        float clamped = Mathf.Lerp(start, end, t);
        // 制限のない t 値を使う
        float unclamped = Mathf.LerpUnclamped(start, end, t);

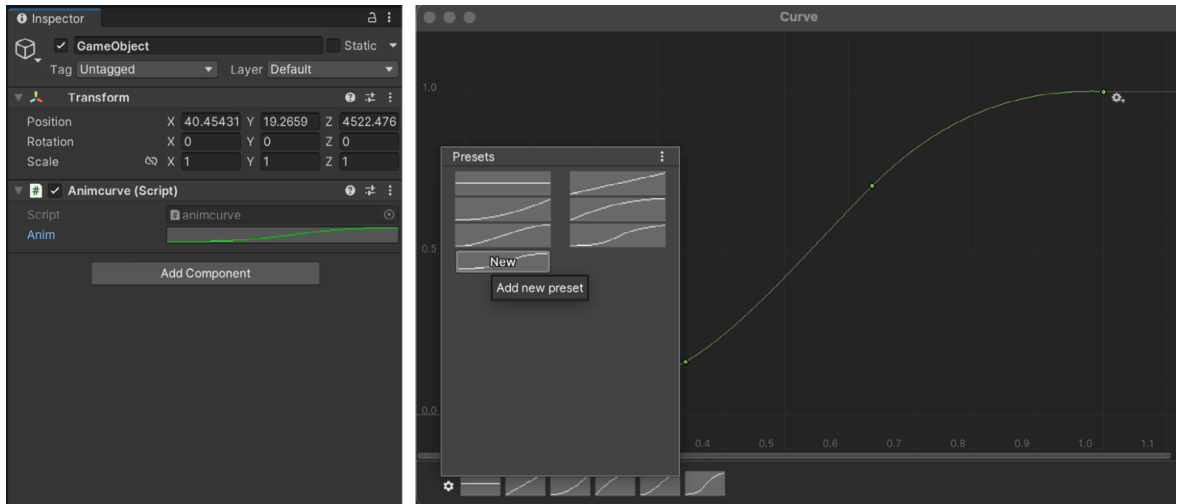
        // 出力は 10
        Debug.Log($"Mathf.Lerp: {clamped} (t = {t})");
        // 出力は 15
        Debug.Log($"Mathf.LerpUnclamped: {unclamped} (t = {t})");
    }
}
```

Unity での拡張 lerp の使用例

AnimationCurve をアニメーション以外にも使用する

アニメーションカーブは通常、[AnimationClip](#) で [コンポーネント](#) のプロパティ値をアニメーション化するために使用されますが、任意の Float 値を動的に制御することもできます。

アニメーションカーブは、`public` 変数として、またはシリアル化されたときに、Inspector で編集できます。編集モードまたはランタイムで保存、エクスポート、ロードできます。編集可能な [接線](#) により、キー間のカーブの形状を制御できます。



Inspector の Animation Curve (アニメーションカーブ) プロパティ: クリックすると Curve Editor が開き、歯車アイコンを選択してカーブを調整し、独自のライブラリに保存できます。

ブログ記事 [Animation Curves, the ultimate design lever](#) (アニメーションカーブ、究極のデザインツール) では、プロジェクトでアニメーションカーブを使用する実践的なヒントや事例を紹介しています。

オブジェクトプールで処理負荷を軽減する

オブジェクトプール は、繰り返し行われる生成と破棄の呼び出しに必要な CPU の処理負荷を軽減することで、性能を最適化するデザインパターンです。これを使うと、既存のゲームオブジェクトを何度でも再利用できます。

オブジェクトプールの使い方は、アプリケーションによって異なります。GC (ガベージコレクション) スパイクを引き起こすリスクがあるので、多数のオブジェクトをインスタンス化するたびにコードをプロファイリングすることが一般的な指針です。

ゲームプレイにスタッターのリスクをもたらす重大なスパイクが検出された場合は、オブジェクトプールの使用を検討してください。ただし、オブジェクトプールは、プールの複数のライフサイクルを管理する必要があるため、コードベースが複雑になる可能性がある点に注意が必要です。また、早めに作成したプールが多すぎると、ゲームプレイに不要なメモリを確保する結果になることもあります。

オブジェクトプールの詳細については、e-book [Level up your code with design patterns and SOLID](#) (デザインパターンと SOLID でコードをレベルアップ) と、Unity Asset Store から無料で入手できるその [サンプルプロジェクト](#) を参照してください。

その他のリソース

- [Use a C# style guide for clean and scalable game code \(Unity 6 edition\)](#) (C#スタイルガイドを使って、クリーンでスケーラブルなゲームコードを実現しよう- Unity 6編)
- [The Unity game designer playbook](#) (Unity ゲームデザイナープレイブック)
- [Create modular game architecture in Unity with ScriptableObjects](#) (Unity で ScriptableObject を使用してモジュラー・ゲーム・アーキテクチャを作成する)
- [Effective asset management in Unity with Addressables](#)
- [What you need to know about Build Profiles in Unity 6](#)

IDE とデバッグ

Debug.Break で実行を一時停止する

Inspector で特定の値を確認する必要があるものの、アプリケーションを手動で一時停止することが困難な場合、[Debug.Break](#) を使用してコード内で実行を一時停止できます。

Debug.Assert で if ステートメントをなくす

Debug.Assert は、設定した条件をランタイムにチェックし、その条件が false を返した場合にコンソールにエラーメッセージを表示します。常に行われる **Debug.Log** とは異なり、アサーション (Debug.Assert) は想定外の状態を検出するためのもので、コード内の前提条件を検証する手段としてより効果的です。

```
// この if ステートメントは、リリース時には不要
if (health > maxhealth)
{
    Debug.LogError("Current health is greater than maxhealth!", this);
}
// アサーションを使用
Debug.Assert(health < maxhealth, "Current health is greater than max-
health!", this);
```

コンテキストとともに Debug.Log を使用する

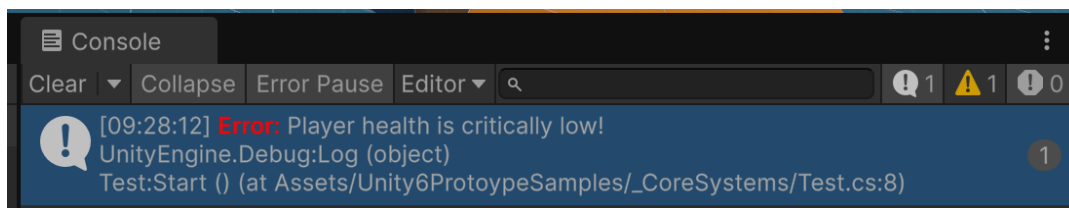
Debug.Log を使用する際、オブジェクト (通常はゲームオブジェクトかコンポーネント) を第 2 引数として渡すことができます。これにより、ログメッセージが Console でそのオブジェクトに関連付けられ、メッセージをクリックすると、Hierarchy 内の関連するオブジェクトが強調表示されます。

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

リッチテキストで重要なメッセージを目立たせる

Unity の Console は、**Debug.Log** メッセージで一部の **リッチテキスト** (、<i>、<color> など) をサポートしています。これを使用して、ログ出力の一部を強調表示したり、色分けしたりして、開発中に重要なメッセージを目立たせることができます。

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



デバッグログをビルドから削除する

デバッグ ログ出力 API は開発以外のビルドから自動的に削除されません。そのため、**デバッグログ** 呼び出しをカスタムメソッドでラップし、**[Conditional]** 属性で装飾します。

対応する **Scripting Define Symbol** を Player Settings から削除すると、デバッグログはすべてコンパイル対象外になります。これは、**#if... #endif** プリプロセッサブロックでラップするのと同じです。

例については、こちらの [General Optimizations](#) (一般的な最適化) ガイドを参照してください。

レイキャスティングを視覚化して物理演算のトラブルシューティングを行う

物理演算のトラブルシューティングに際しては、**Debug.DrawLine** と **Debug.DrawRay** を使って、指定した開始点と終了点の間に線を描画すると、レイキャスティングの視覚化に役立ちます。

```
void Start()
{
    // 2.5 秒間、原点から 5 ユニットの白線を引く
    Debug.DrawLine(Vector3.zero, new Vector3(5, 0, 0), Color.white,
2.5f);
}
```

開発ビルドで Debug.isDebugBuild を使用する

Debug.isDebugBuild を使用すると、アプリケーションが開発ビルドとして実行されているかどうかを確認できます。これにより、リリースビルドに影響を与えることなく、ログ出力、診断、テストユーティリティなどのデバッグ専用コードを条件付きで実行できます。

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

Application.SetStackTraceLogType を設定する

Application.SetStackTraceLogType または **PlayerSettings** の同等のチェックボックスを使用して、スタックトレースを表示するログメッセージの種類を決定します。スタックトレースは便利ですが、動作が遅く、ガベージを発生させます。

Stack Trace*			
Log Type	None	ScriptOnly	Full
Error	☐	☒	☐
Assert	☐	☒	☐
Warning	☐	☒	☐
Log	☐	☒	☐
Exception	☐	☒	☐

エディターウィンドウの PlayerSettings の Stack Trace

ログをカスタマイズする

Logger API を使用すると、カスタム Logger (ロガー) インスタンスを作成して設定でき、より高度なログ出力やモジュール式のログ出力を実現できます。ビルトインの **Debug.unityLogger** を使用できますが、独自のロガーを作成すると、ログの形式、フィルター、出力チャンネルをより細かく制御できます。

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

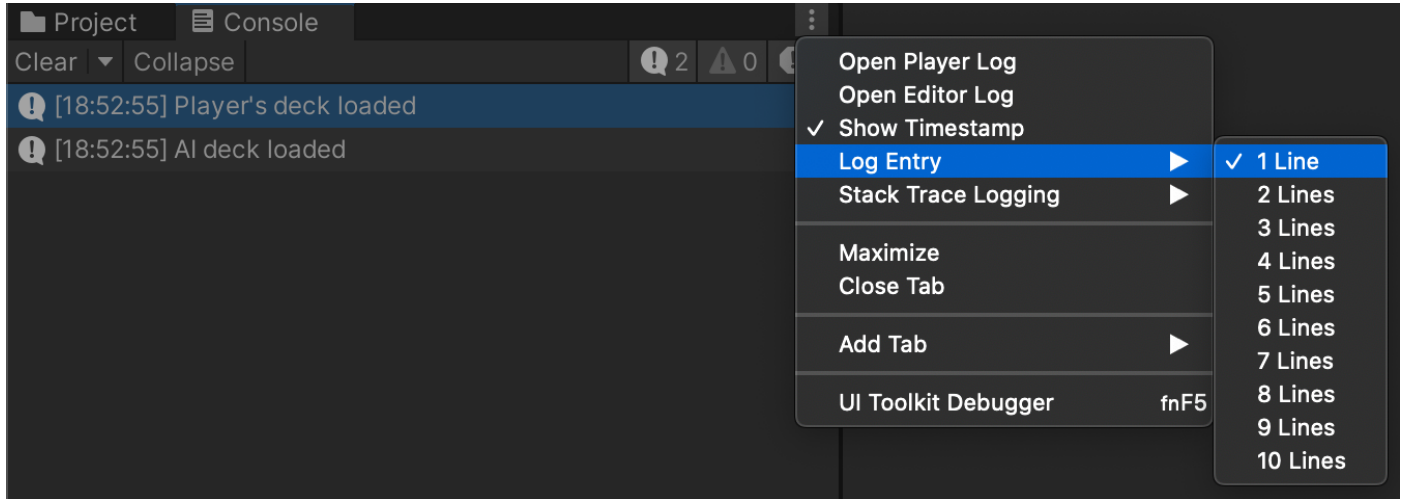
Visual Code のショートカットでワークフローを迅速化する

IDE として Visual Code を使用する場合は、次の **ショートカット** が役立つでしょう。

- [Windows](#)
- [Mac](#)

Console の Log Entry を設定して可読性を高める

デフォルトでは、Console の Log Entry は 2 行です。可読性を高めるために、必要に応じて 1 行または複数行を設定し、作業を効率化できます (下の画像を参照)。



Console の Log Entry オプションでログの行数を設定できます。

その他のリソース

- [How to debug game code with Roslyn Analyzers](#) (Roslyn アナライザーでゲームコードをデバッグする方法)
- [How to run automated tests for your games with the Unity Test Framework](#) (Unity Test Framework でゲームの自動テストを実行する方法)
- [Speed up debugging with Microsoft Visual Studio Code](#) (Microsoft Visual Studio Code でデバッグワークフローを迅速化)
- [How to debug your code with Microsoft Visual Studio 2022](#) (Microsoft Visual Studio 2022 でコードをデバッグする方法)
- [Testing and quality assurance tips for Unity projects](#) (Unity プロジェクトのテストと品質保証のヒント)

DevOps ワークフロー

Unity Version Control のヒント

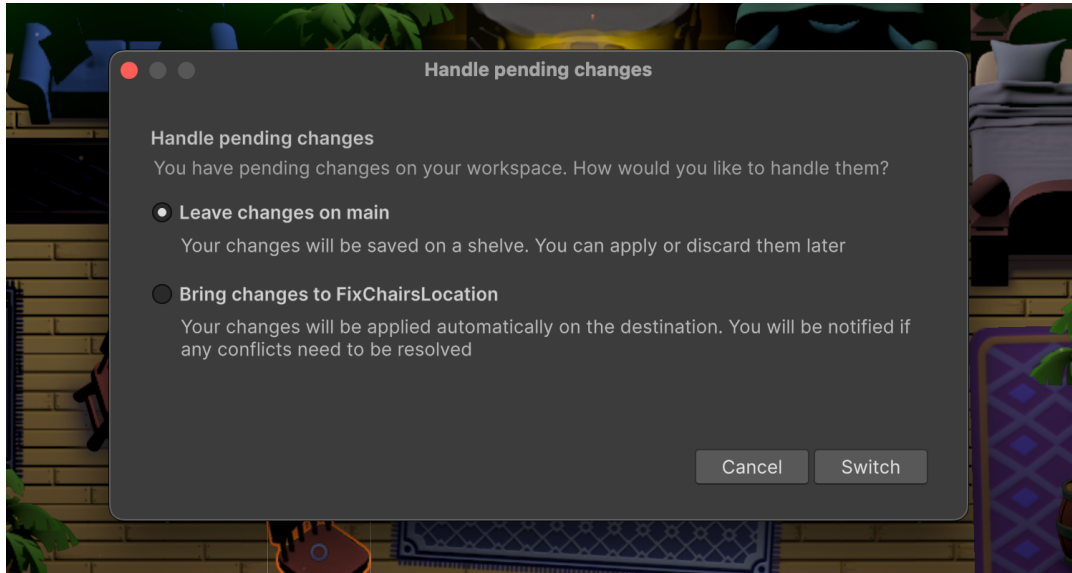
Unity Version Control (UVCS) は、あらゆる規模のゲーム開発スタジオに対応する、エンジンに依存しないスケーラブルな [バージョン管理](#) および [ソースコード管理](#) ツールです。

Unity Version Control を Unity 6 ですでに使用していれば、ここで紹介する最新の更新とヒントがチームの日常業務に役立つでしょう。

Shelve と Switch を使用して、作業内容を失うことなく複数のブランチで並行して作業する

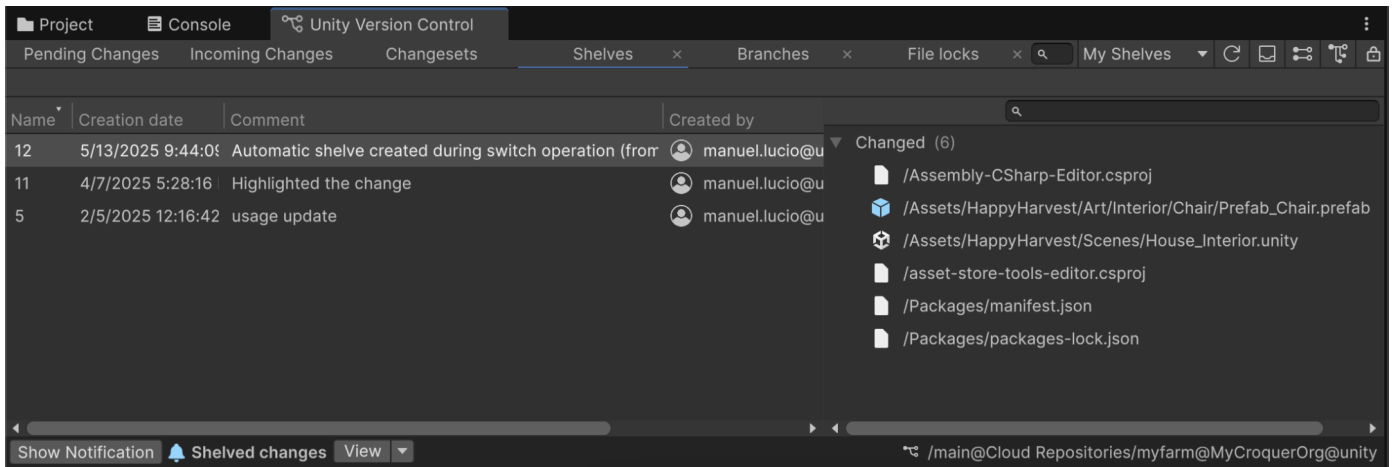
[UVCS パッケージバージョン 2.7.1](#) から、シェルフセットを使用して、ブランチを切り替えるときに現在の変更を一時的に保存できるようになりました。未完了の作業を安全に保存して簡単にアクセスできます。

Unity Version Control ウィンドウで、切り替え先のブランチを右クリックし、**Switch workspace to this branch** (このブランチにワークスペースを切り替える) を選択します。



Handle pending changes (保留中の変更の処理) ウィンドウ

次に、変更を引き継ぐか、一時的に残すかを選択します。**Shelves** オプションは柔軟性に優れており、変更を再検討したり、再適用したり、チームと共有したりできます。

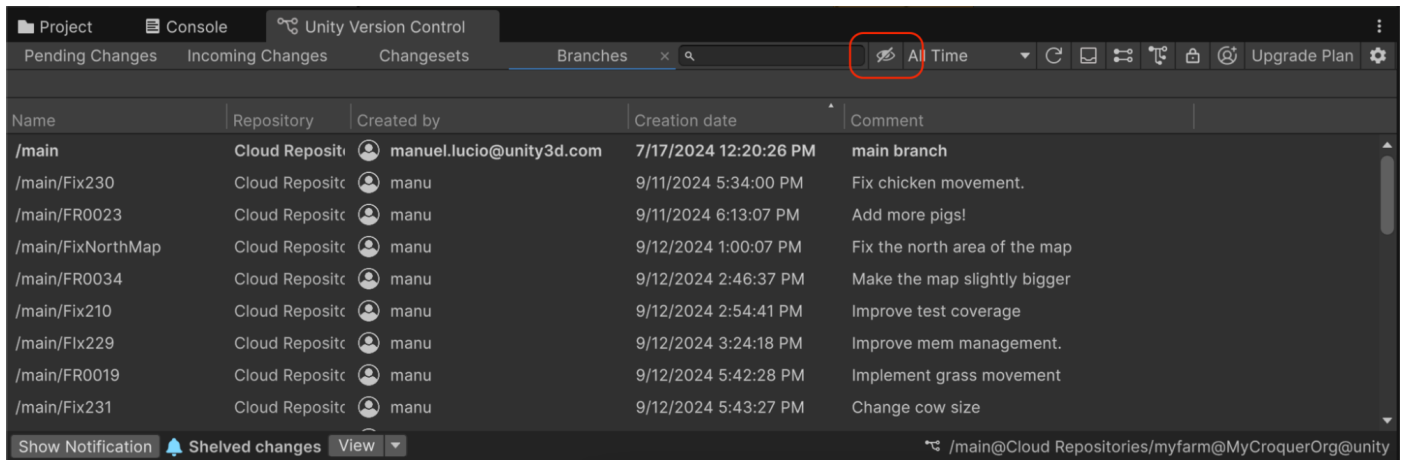


Unity Version Control ウィンドウの Shelves タブ

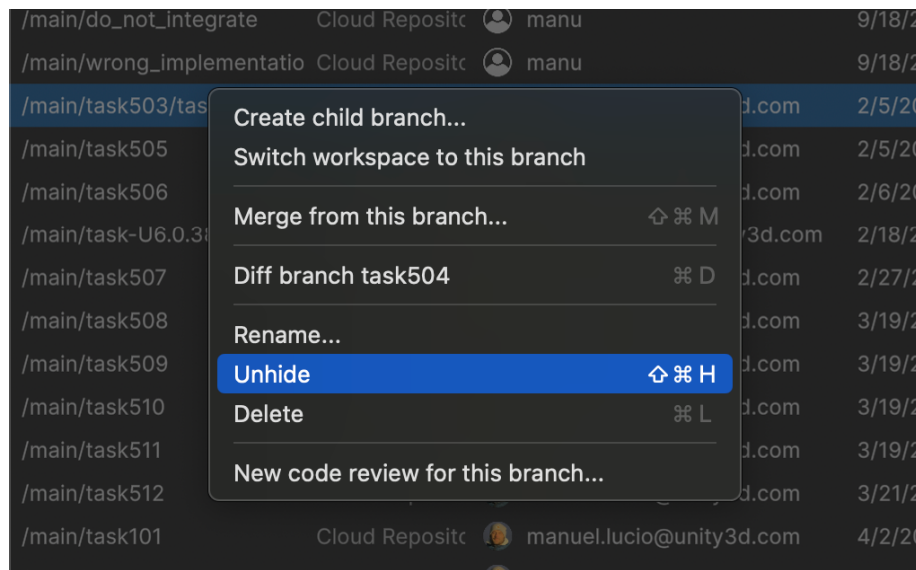
表示/非表示機能でブランチビューをすっきりさせる

プロジェクトのブランチビューが煩雑になったら、グローバルな **表示/非表示** 機能を使用してコンテキストメニューから直接ブランチを非表示にできます。これにより、すべての UVCS プラットフォーム (Unity プラグイン、Unity Dashboard、デスクトップアプリケーション) において、チーム全体でブランチが見えなくなります。

非表示ブランチは、“斜線入りの目” のアイコンか、非表示ブランチリストのコンテキストメニューで、再度表示できます (下の画像を参照)。



Unity Version Control の “斜線入りの目” アイコン

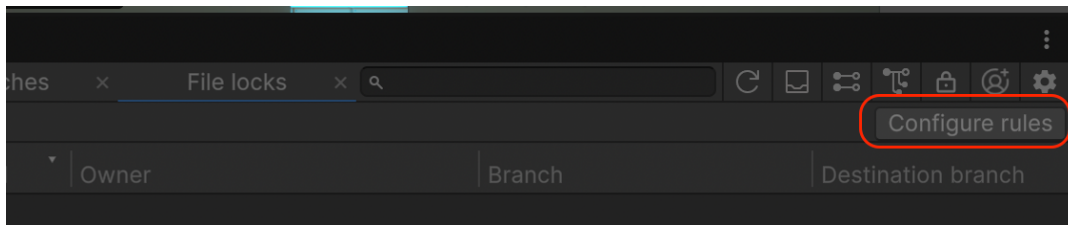


ブランチのコンテキストメニューの Unhide (隠さない = 表示) オプション

排他ロックオプションによりアセットデータの損失を回避する

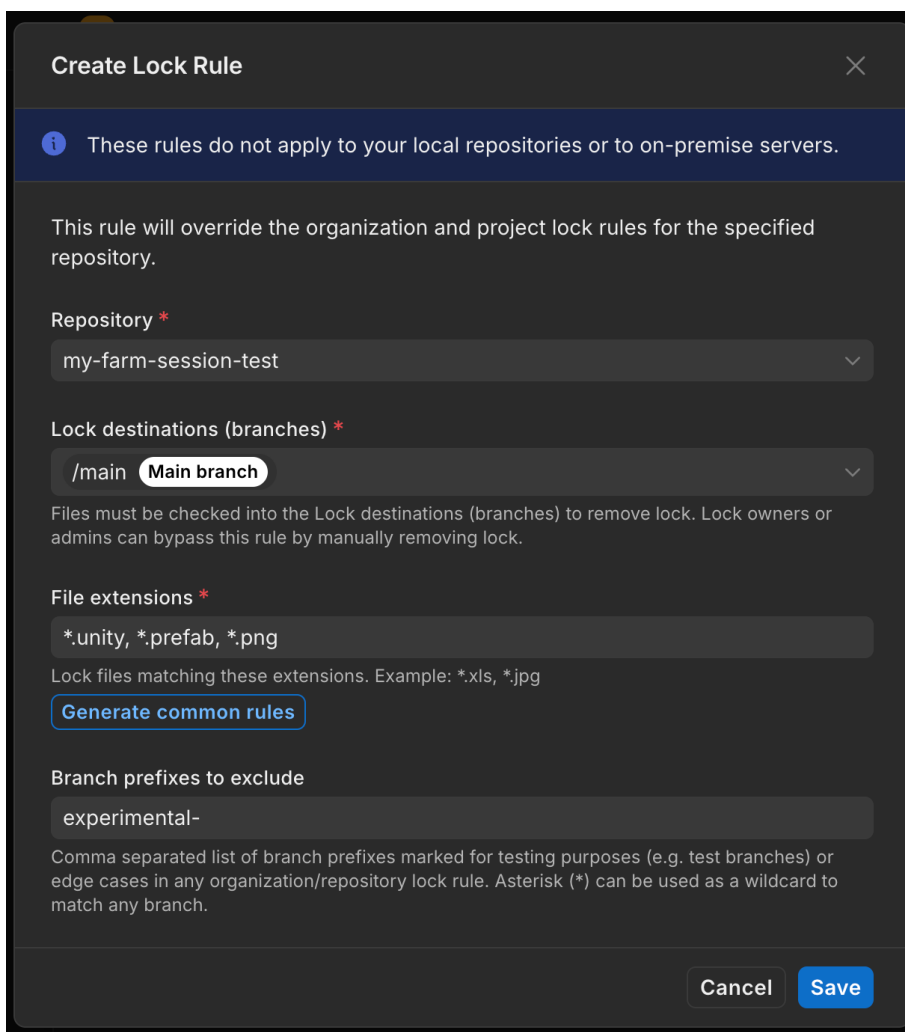
Unity エディター用の UVCS プラグインを使用すると、アセットをロックして他のユーザーによる変更を防ぐことができます。これにより、作業中のデータ損失を防ぎ、マージできないファイルに対するマージ操作を排除できます。

ロック機能を有効にするには、**File locks** タブの **Configure rules** に移動します。



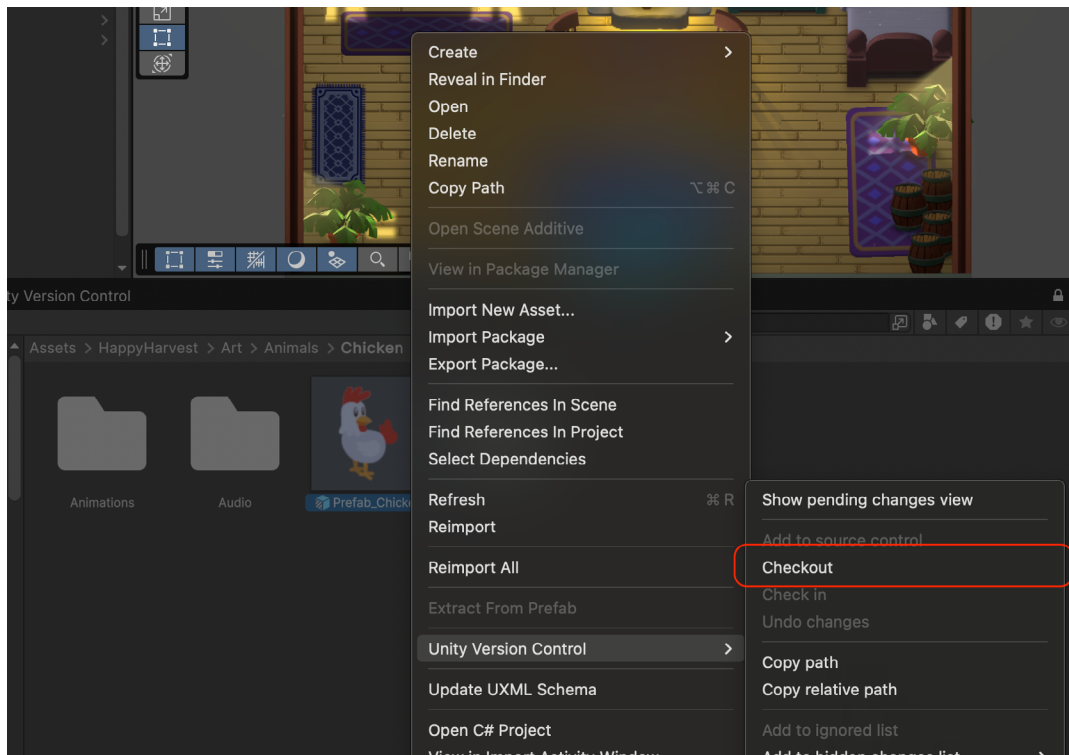
File locks タブの Configure rules オプション

次に、ロックサポートを有効にするために必要なリポジトリ、転送先のブランチ (通常はメインブランチ)、およびファイル拡張子を設定します。



Unity Version Control の Create Lock Rule

競合を避けるため、変更前に必ずアセットをチェックアウトしてください。



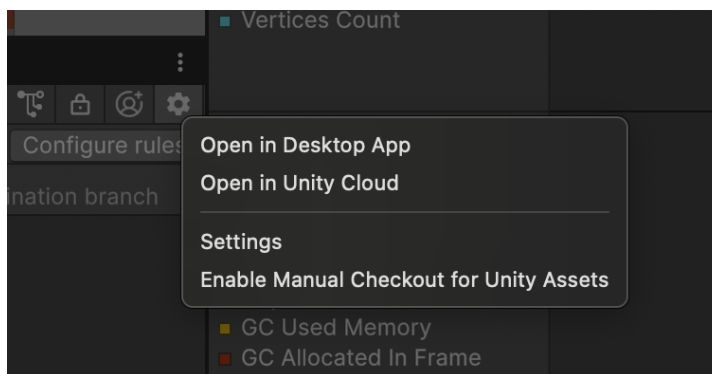
Unity Version Control コンテキストメニューの Checkout (チェックアウト) オプション

スマートロックシステムの詳細については、[Smart Locks](#) (スマートロック) のページを参照してください。

“Enable manual checkout for Unity Assets” で変更を保護する

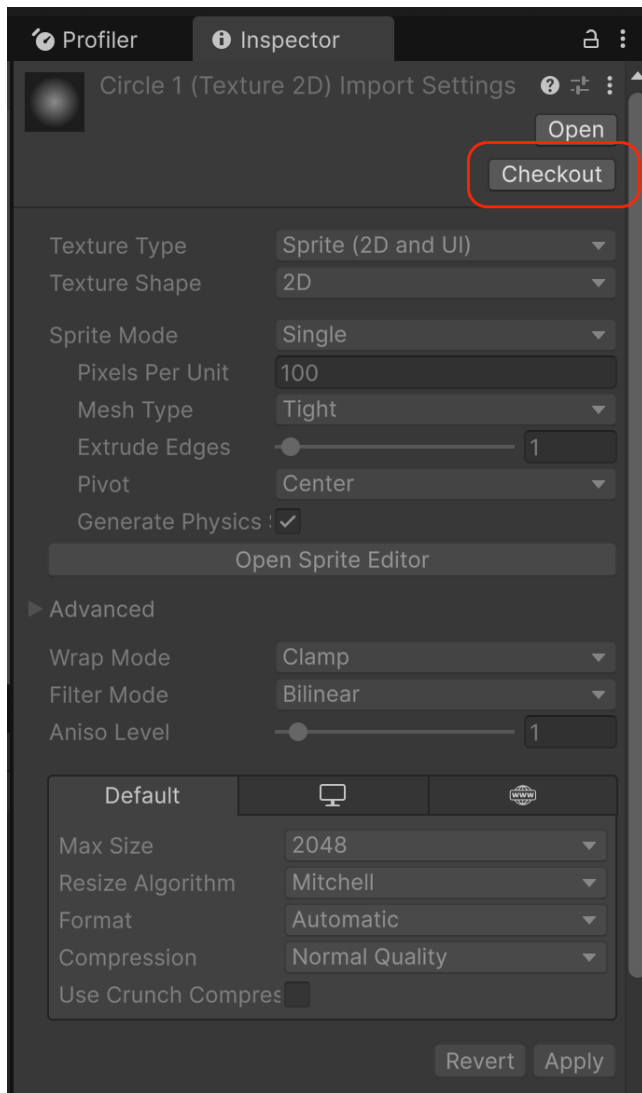
マージできないアセットを扱う際、ファイルを変更して保存すると、そのファイルは UVCS Unity プラグインにより自動的にチェックアウトされます。しかし、このワークフローでは、変更の反映前に、他のユーザーが同じファイルをロックしてしまう可能性があります。

これを回避するには、**歯車** アイコンをクリックして UVCS プラグイン設定に移動し、**Enable manual checkout for Unity Assets** (Unity アセットの手動チェックアウトを有効にする) オプションを選択します。



Enable Manual Checkout for Unity Assets オプション

この設定により、アセットが明示的にチェックアウトされるまで読み取り専用になります。Inspector で **Checkout** (チェックアウト) オプションをクリックすると、ファイルが編集可能になり、変更のための排他ロックが設定されます。



Inspector の Checkout オプション

Asset Manager

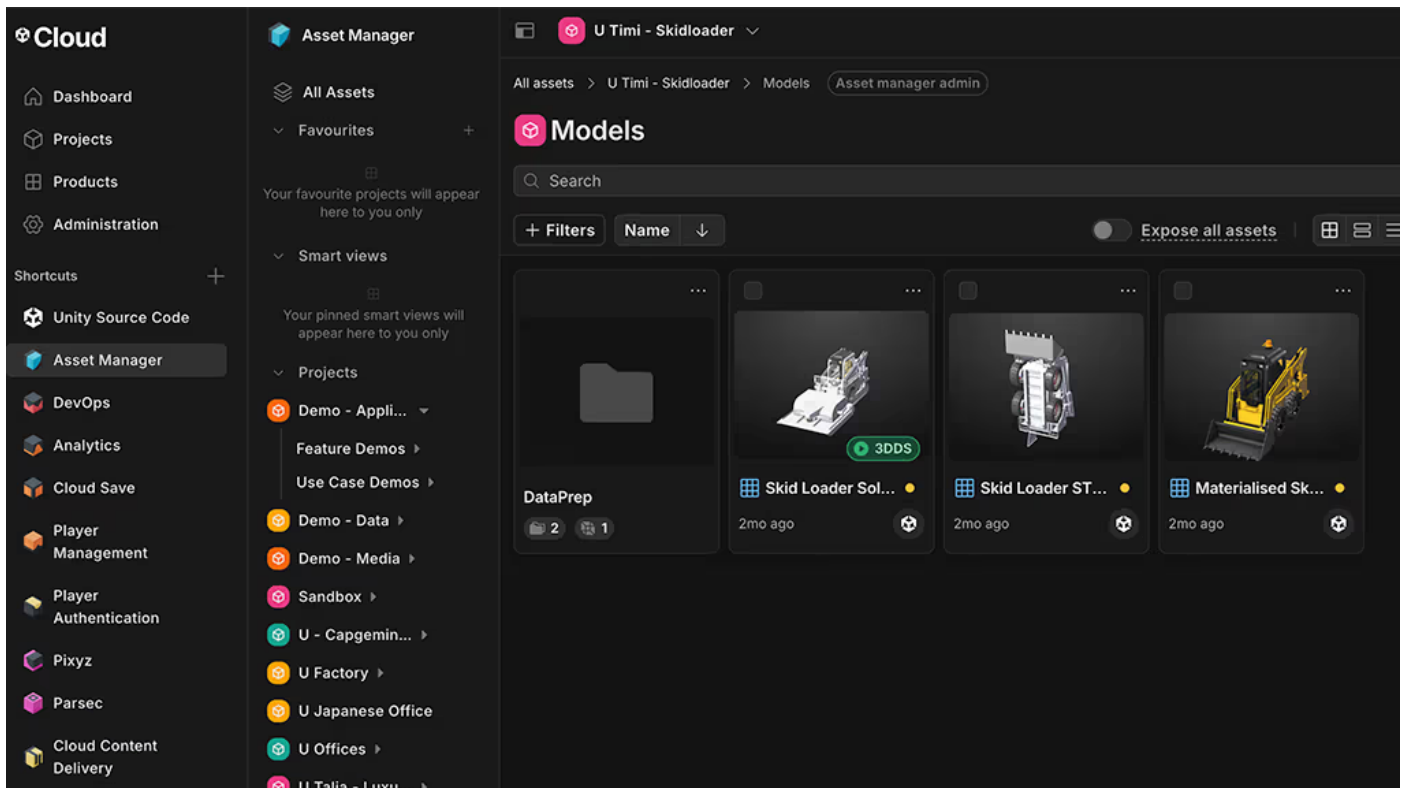
モデル、テクスチャ、オーディオファイルなどのアセットを管理するためのデジタルアセット管理システムを探しているのであれば、[Unity Asset Manager](#) はいかがでしょうか。[70 種類を超えるファイル形式](#) をサポートしており、チームがプロジェクト間でアセットをシームレスに一元管理、整理、検索、使用できます。

Asset Manager は、必要に応じて、ウェブインターフェース、またはエディターとの統合を介して使用します。

好みのブラウザーでアセットを管理、表示、整理する

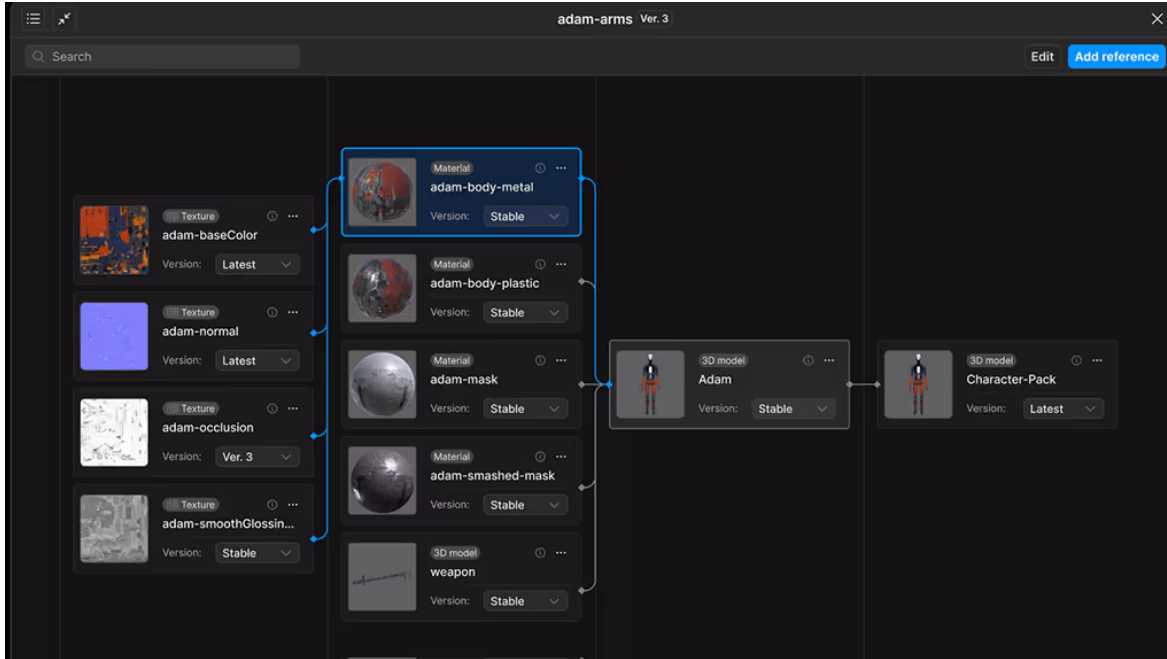
ウェブインターフェースにより、プロジェクトに関わるすべてのチームメンバーや関係者の間で非同期かつ迅速な共同作業が可能になります。ウェブインターフェースを通じて、コレクションの作成と管理、コレクション間またはプロジェクト間のアセットの移動、バージョン履歴による変更の追跡を行うことができます。

関連するコレクションやタグを加えながら、1 つまたは複数のファイルを簡単にアップロードできるため、アセットの構造化や、チームとの共同作業によるレビューや改善が簡単になります。カスタムメタデータでアセットを強化できます。カスタムメタデータでは、管理者、所有者、マネージャーがテキスト、ブーリアン、数値などの新しいメタデータフィールドを定義でき、すべてのユーザーが AI 提案タグなどのタグを加えることができるため、全体的に整理されます。



Asset Manager の ウェブインターフェースでアセットを操作

Dependency Viewer では、アセットの関係が一目でわかり、各アセットが何を必要とし、何に依存しているかがわかります。上流と下流の両方の依存関係を確認できます。

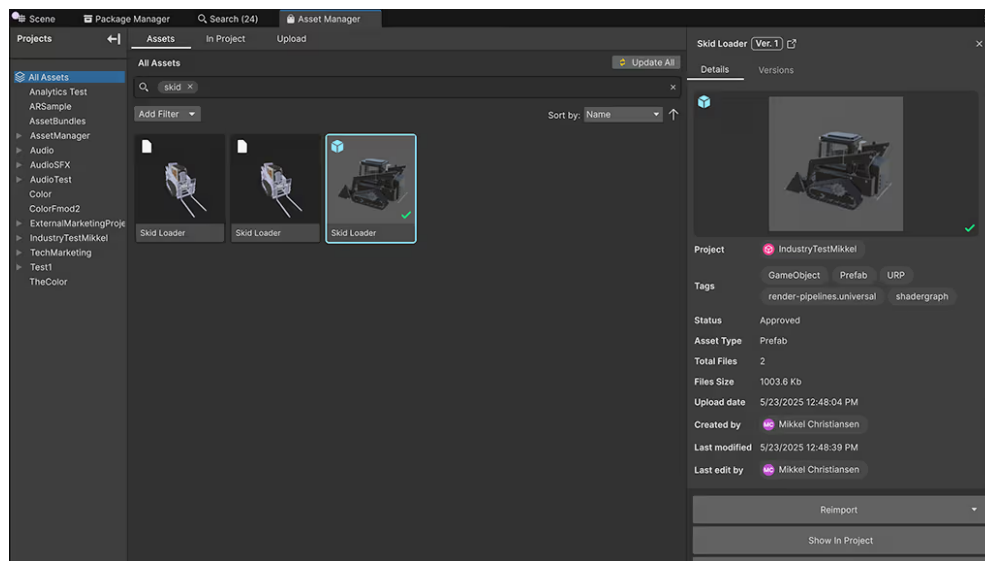


Dependency Viewer でのアセット依存関係の例

エディターで作業を中断することなくアセットを管理する

Unity Asset Manager パッケージは Unity エディターと統合されており、ワークフローを中断することなくアセットを管理できます。インストール後、Unity Cloud にアセットをアップロードしたり、Unity Cloud からアセットをインポートしたりできるほか、エディターの検索インターフェースを通じてすべてのプロジェクトからアセットにアクセスできます。

また、このエディター内ソリューションにより、プレハブ、スクリプト、その他のプロジェクト横断コンポーネントの再利用が容易になり、イテレーション時間が短縮され、チームはワークフローの改善と最適化に集中できます。



Unity エディターから Asset Manager へのアクセス



特定のアセットを効率的に見つけるには、作成者、状態、種類、更新日時、インポート状態でフィルターをかけるようにします。プラグインはアセット間の依存関係を自動的に検出します。例えば、複数のプレハブが同じマテリアルを使用している場合、マテリアルのインスタンスは 1 つだけアップロードされます。これにより、ストレージが最適化され、不要な重複が防止されます。

Unity Asset Manager の詳細については、以下のリソースを参照してください。

- 動画チュートリアル: [A quick guide to the Asset Manager in Unity](#) (Unity Asset Manager のクイックガイド)
- Unity Learn: [Unity Asset Manager: Quick start guide](#) (Unity Asset Manager: クイックスタートガイド)
- 記事: [Introduction to Asset Manager transfer methods in Unity](#) (Unity Asset Manager における転送方法の概要)

Unity Build Automation

Unity DevOps Build Automation (以前の名称は Cloud Build) は、クラウド上でビルドを実行してデプロイでき、すぐに利用可能な継続的インテグレーション/継続的デプロイ (CI/CD) ソリューションです。ビルドとリリースの頻度を上げ、より高品質で革新的なリリースを行う助けとなります。

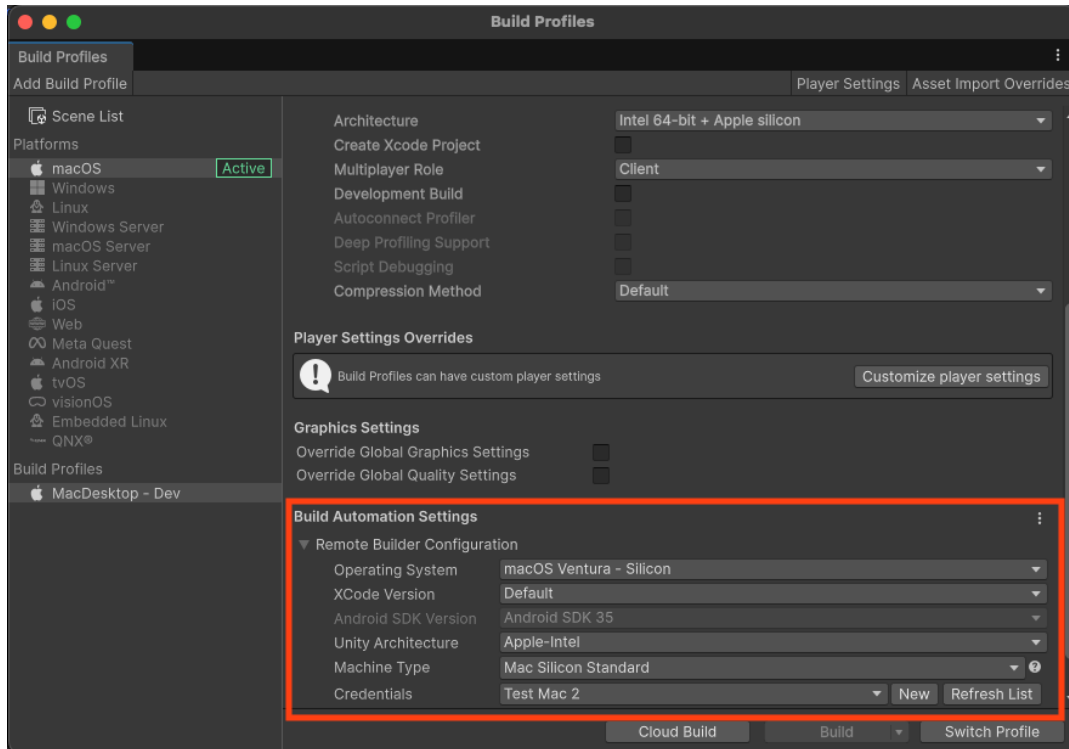
Build Automation は、任意のソース管理リポジトリに数分で接続でき、変更がバージョン管理システムにコミットされたら、手動または自動でビルドを実行するように設定できます。また、Build Automation は iOS、Android、Windows、Unity Web などの複数のプラットフォームに対応しているため、プラットフォームごとに固有のビルド環境を維持する必要がありません。

[Unity 6.1](#) から、Build Automation がエディターに完全に統合され、Build Automation の設定を使用してローカルブランチから迅速な検証ビルドを実行できるようになりました。

ビルド設定をチーム全体で共有して再利用する

Build Automation パッケージは、Build Profiles ウィンドウまたは Package Manager からインストールできます。

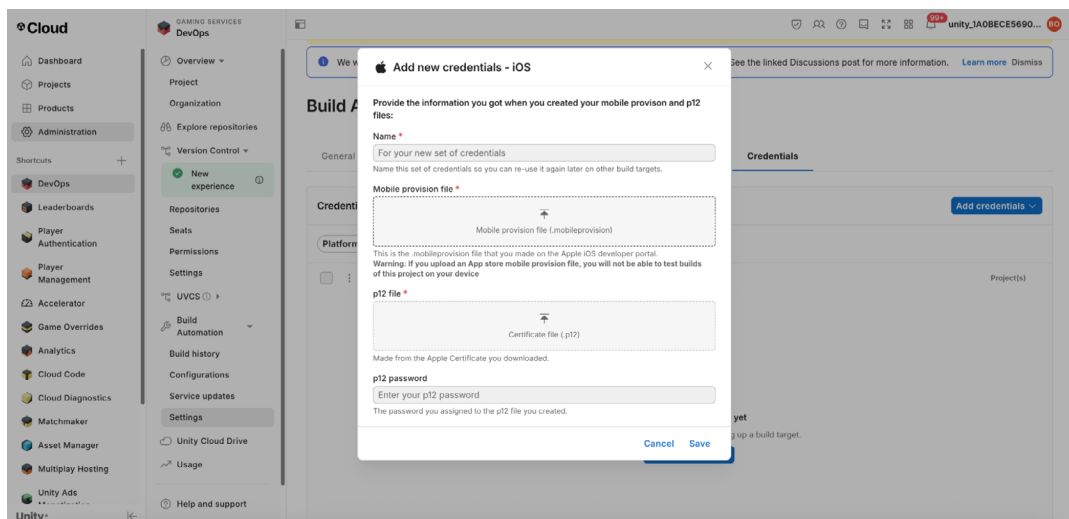
パッケージをインストールすると、選択したプロファイルの下に **Build Automation Settings** というラベルの新しいセクションが表示されます。これらの設定により、Unity によるビルドの自動化方法を定義できます。



Build Profiles ウィンドウの Build Automation Settings セクション

Android、iOS、その他認証情報が必要なプラットフォームでは、Unity Dashboard で認証情報を設定する必要があります。新しい認証情報を作成するには、**Credentials** ドロップダウンの横にある **New** ボタンをクリックします。

これにより、Build Automation ダッシュボードに新しいウィンドウが開き、新しい認証情報を作成できます。認証情報とアプリケーション署名の詳細については、[Unity Build Automation ドキュメント](#) を参照してください。

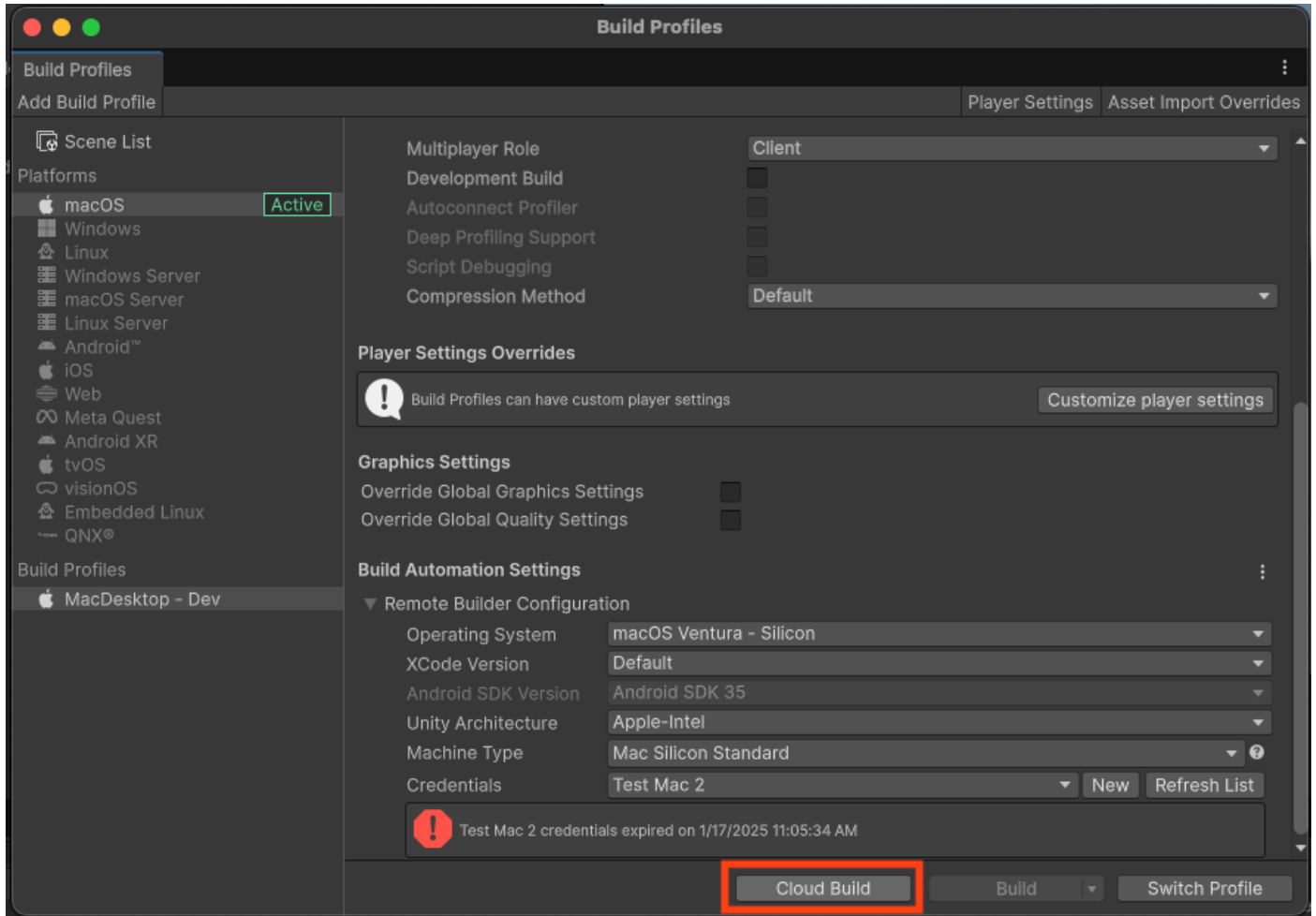


Unity Dashboard の認証情報設定ウィンドウ

認証情報を保存したら、Build Profiles ウィンドウに戻り、**Refresh list** をクリックします。ドロップダウンリストに認証情報が表示されます。ビルド設定はビルドプロファイルに自動的に保存されるため、チーム全体で共有して再利用できます。

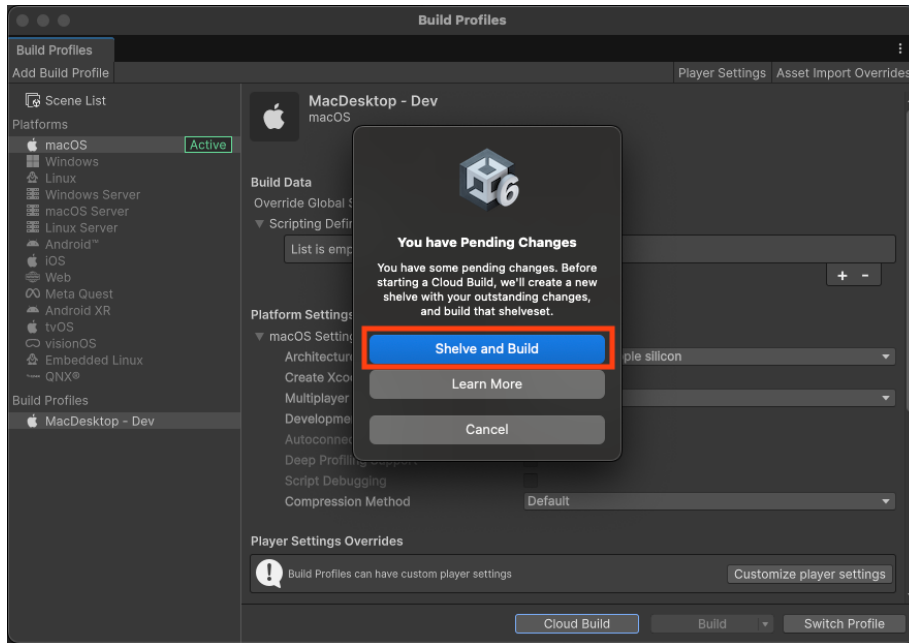
Shelve and Build で作業中の変更をテストする

設定を行い、ビルドを実行する準備ができたなら、ビルドプロファイルの **Cloud Build** ボタンをクリックします。クラウド上でビルド処理が始まります。



ビルドプロファイルの Cloud Build 設定

プロジェクトに保留中の変更がある場合は、**Shelve and Build** オプションを選択するよう求められます。ローカルの変更をメインリポジトリにプッシュすることなくビルドできるため、作業中の変更のテストに最適です。



Pending Changes (保留中の変更) の Shelve and Build

保留中の変更がない場合、現在チェックアウトされているブランチの最新コミットを使用してビルドが作成されます。

Build History でビルドの進捗を追跡し、トラブルシューティングを行い、結果をダウンロードする

ビルドが始まるとすぐに、エディターに **Build History** ウィンドウが開き、進捗状況とビルドの状態を確認できます。このビューでは、ログとビルドアーティファクトへのアクセスが可能で、トラブルシューティングや結果のダウンロードが容易になります。

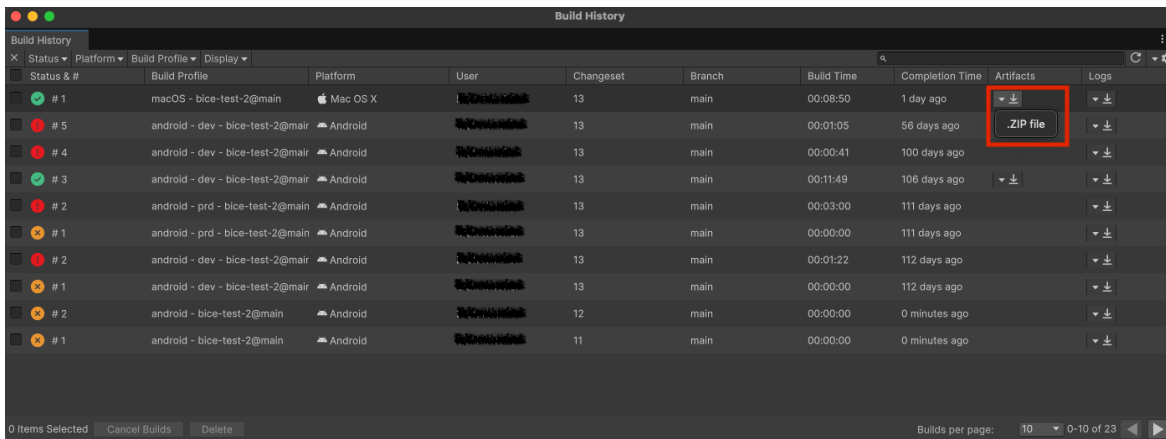
Build History										
Status & #	Build Profile	Platform	User	Changeset	Branch	Build Time	Completion Time	Artifacts	Logs	
✓ #1	macOS - bice-test-2@main	Apple OS X	[redacted]	13	main	00:08:50	1 day ago	↓	↓	↓
✗ #5	android - dev - bice-test-2@mair	Android	[redacted]	13	main	00:01:05	56 days ago	↓	↓	↓
✗ #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	↓	↓	↓

Build History ウィンドウ

各ステータスは一目でわかるように色分けされており、失敗したビルドの再実行やログのダウンロードなどのアクションをとることができます。各状態表示の意味については、[このドキュメント](#) ページを参照してください。

ビルドアーティファクトのリンクをチームメンバーと共有する

ビルドが正常に完了したら、ビルドアーティファクトのダウンロードや、チームメンバーとのリンクの共有ができます。



成功したビルドのアーティファクトをダウンロードします

失敗したビルドの場合は、ログをダウンロードしてさらに詳しく調べることができます。

Build Automation について詳しくは、『[Getting started with Build Automation in Unity](#)』(Unity でのビルド自動化入門) 動画チュートリアルおよび Unity Learn の『[Unity Build Automation: Quick start guide](#)』(Unity Build Automation: クイックスタートガイド) チュートリアルを参照してください。

Unity Build Server

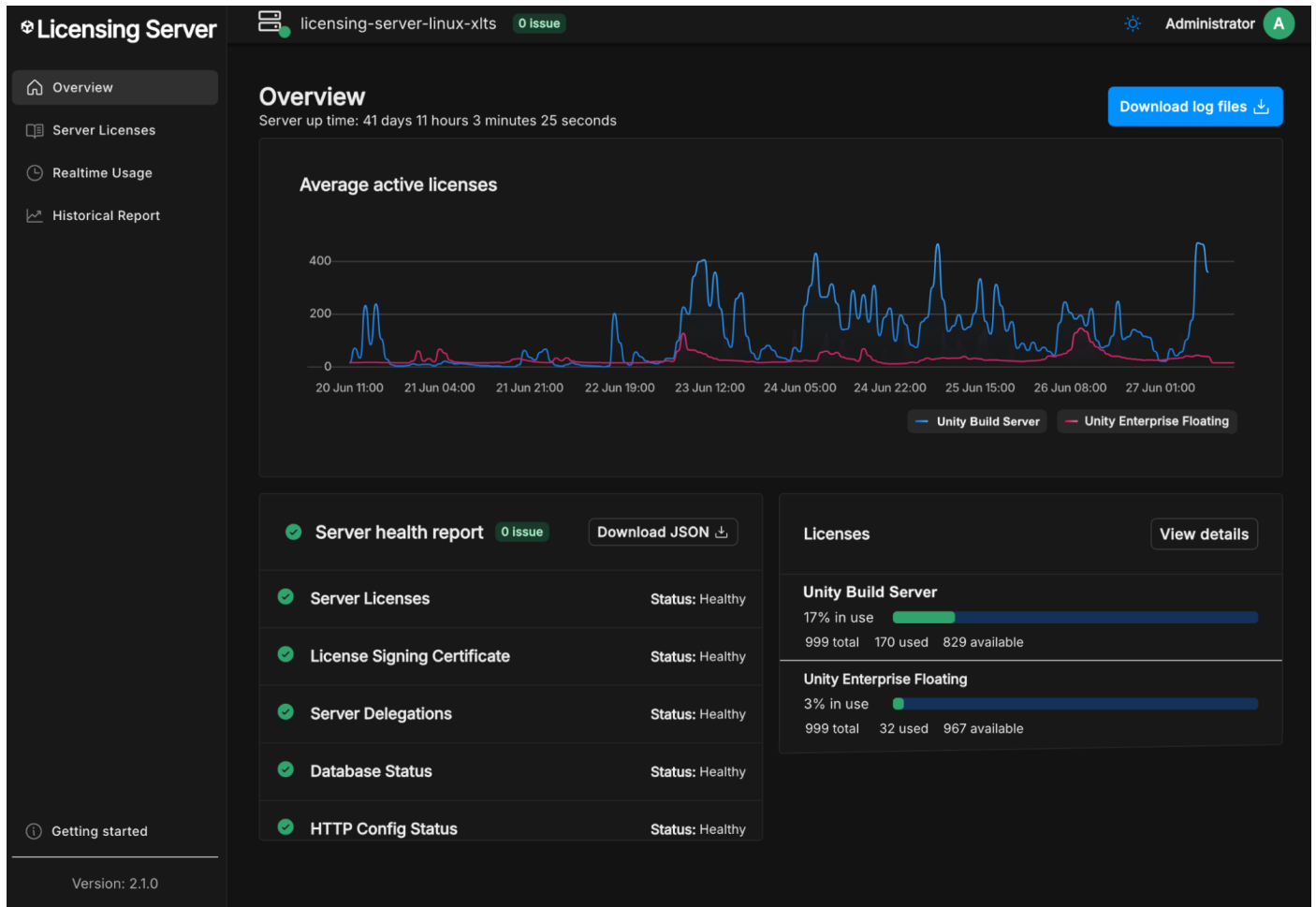
Unity プロジェクトのビルド、コンパイル、パッケージ化のプロセスを合理化および自動化したい場合、特に大規模なチームや組織では、[Unity Build Server](#) をお試しください。

これは Unity のフローティングビルドライセンスを可能にするライセンス管理ツールで、各マシンに Unity エディターのフルライセンスがなくても複数のマシンでビルドできます。自動ビルドパイプラインや継続的インテグレーション (CI) 環境への有用な追加機能です。

Unity Build Server の使用を開始するには、Unity Licensing Server ソフトウェアをネットワークの中央サーバーにデプロイし、Unity Build Server ライセンスをこのサーバーに設定します。その後、プロジェクトをビルドする必要があるときに Build Server にビルドライセンスを要求するよう各ビルドマシンを設定できます。

Build Server ソリューションで使用される Unity Licensing Server は、同じサーバーで複数種類のサブスクリプションに対応できます。Unity Enterprise Floating や Unity Industry Floating サブスクリプションを使用している開発者に、同じサーバーからフローティングライセンスを配布できます。下の画像の例を参照してください。

Unity Build Server の要件、設定方法、使用を開始する方法については、[Unity Licensing Server](#) のドキュメントを参照してください。



2 つの有効なサブスクリプション: Unity Build Server (ビルドマシン用) と Unity Enterprise Floating (開発者用)。

サンプルプロジェクト

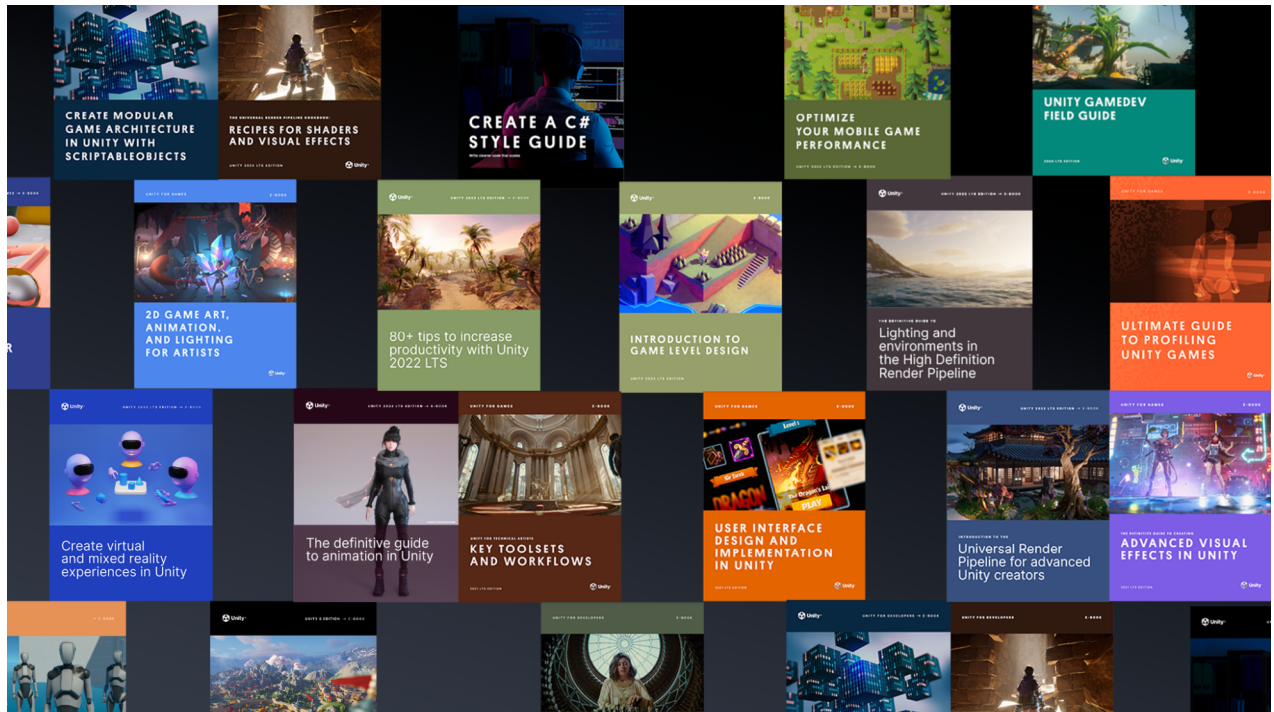
UI Toolkit や UI Builder のワークフローを紹介するゲームインターフェースのサンプルプロジェクト、さまざまなデザインパターンやプロジェクトアーキテクチャを示すサンプル、Unity 6 のユニバーサルレンダーパイプライン (URP) の 2D ライティングと視覚効果 (VFX) の機能を紹介するものなどを、これらのプロジェクトで網羅しています。

- [Happy Harvest - 2D サンプルプロジェクト](#)
- [Gem Hunter Match - 2D サンプルプロジェクト](#)
- [Dragon Crashers - UI Toolkit サンプルプロジェクト](#)
- [QuizU - UI Toolkit サンプル](#)
- [Paddle Game ScriptableObjects](#)



- [Level up your code with design patterns and SOLID](#) (デザインパターンと SOLID でコードをレベルアップ)
- [C# Code style guide](#) (C# コードスタイルガイド)
- [Fantasy Kingdom](#)
- [Shader Graph サンプル](#)
- [VFX Learning Templates サンプルコンテンツ](#)
- [URP 3D サンプル](#)
- [HDRP サンプルシーン](#)
- [HDRP Time Ghost](#)
- [VR Multiplayer Template](#) (VR マルチプレイヤーテンプレート)
- [MR multiplayer template](#) (MR マルチプレイヤーテンプレート)
- [UGS サンプル](#)

すべての Unity ユーザー向けのリソース



[Unity best practices hub](#) (Unity ベストプラクティスハブ) では、上級 Unity 開発者およびクリエイター向けのさらに多くの e-book をダウンロードできます。業界の専門家や Unity のエンジニア、テクニカルアーティストによって作成された 30 以上のガイドの中から選んでください。Unity のツールセットやシステムを使った効率的なゲーム開発のベストプラクティスを紹介しています。

また、[Unity ブログ](#)、[Unity コミュニティフォーラム](#)、[Unity Learn](#)、や[#unitytips](#) ハッシュタグから、ヒントやベストプラクティス、ニュースを見つけることができます。



unity.com