→ E-BOOK

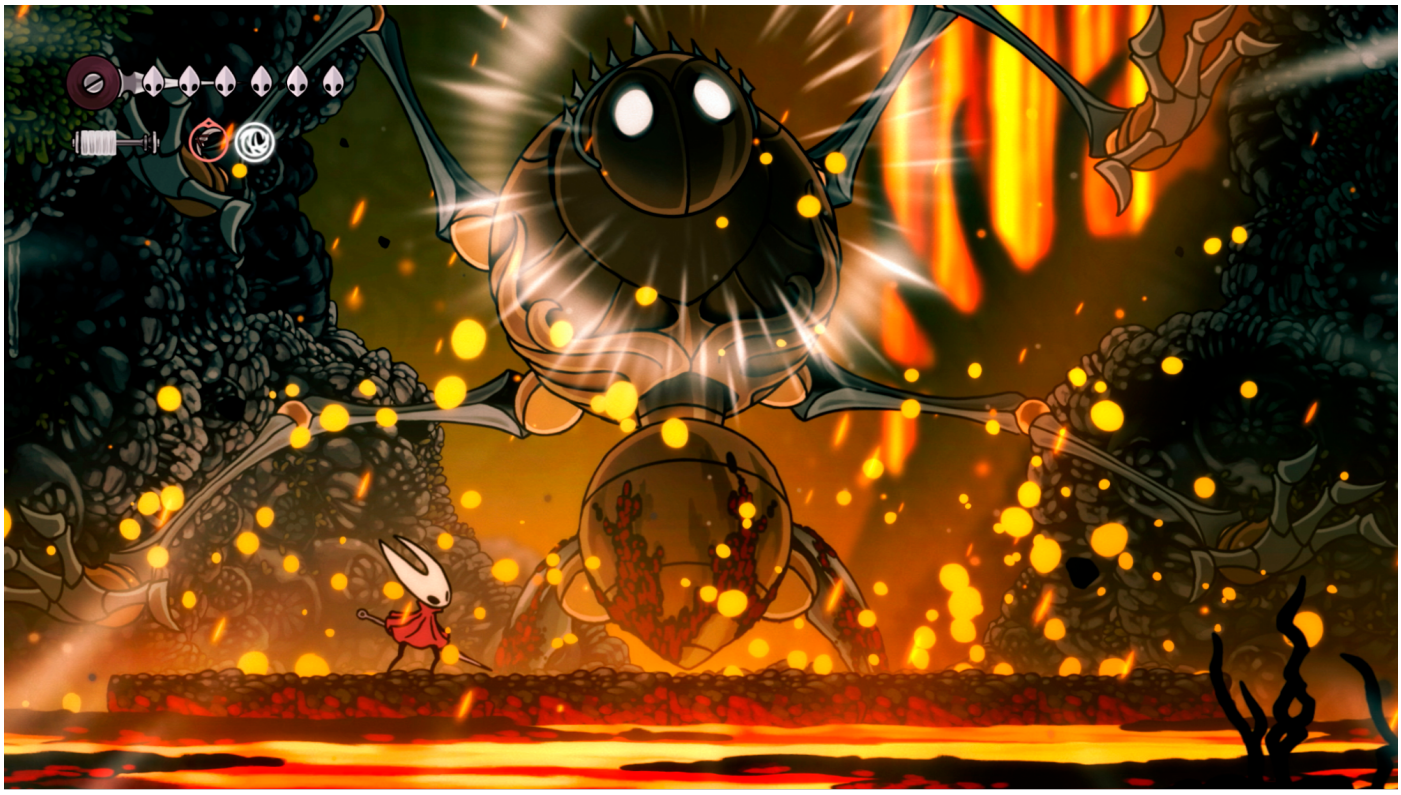
アーティストのための 2Dゲームアート アニメーション ライティング

(Unity 6.3 LTS エディション)

Unity®

目次

はじめに



『Hollow Knight:Silksong』は、Team Cherry が Unity を使用して開発し、PC とコンソール向けに提供している 2D ゲームです。

ゲーム用プラットフォーム、Unity の 2D ツールセット、その他のグラフィックス開発ソフトウェアの進化により、リアルタイムライト、高解像度のテクスチャ、ほぼ無制限のスプライト数を持つ 2D ゲームを作成することが可能になりました。2D グラフィックスの平面性により、アーティストは、どんなアートスタイルでもどのデバイスでも、美しいゲームを自由に作成することができます。

この e-book は、独立して作業するかチームで作業するかにかかわらず、中級レベルの Unity 経験があり商業用の 2D ゲームを作りたい開発者とアーティストのためのものです。

これは、Unity の専門家による重要な貢献をもとに作成された、最も包括的な 2D ゲーム開発ガイドです。クリエイターが Unity の 2D ツールセットを最大限に活用できるように支援します。

Unity 6.3 LTS は現在、本番環境で使用できる最新の Unity リリースであり、この 2D ガイドはその 2D 機能とシステムのワークフローに関する説明を加えて更新されました。この e-book で学んだ知識は、Unity の今後のバージョンにも適用できます。

2D プロジェクトの設定、DCC ソフトウェアから Unity にスプライトを取り込む方法、Unity でのスプライト作成、洗練されたライティング、影、視覚効果の作成、タイルマップ 2D やその他の機能を使用したレベルデザイン、アニメーションに関するテクニックやワークフロー、そして、最適化のヒントの便利なリストを紹介します。

e-book 内の多くの機能は、Unity の最近の 2D サンプルのアセットとシーンを使用して説明されています。サンプルは Dragon Crashers、Happy Harvest、Gem Hunter Match、および 2D と 3D を融合させた Bunny Blitz (2026年初頭に利用可能) を使用しています。これらは Unity Asset Store から無料で入手できるので、この e-book と一緒に学習の参考として使用するためにダウンロードすることをお勧めします。

このガイドを楽しんでご利用いただき、ゲームの開発が成功することをお祈りしております。2D クリエイターの未来は明るいです。

寄稿者

原著者

Jarek Majewski は、豊富な C# スクリプティングスキルを持つプロの 2D アーティストであり、テクニカルアーティスト、独学の Unity 開発者です。彼はモバイルゲームおよびコンソールゲームのアートを作成してきました。ゲームスタジオでのプロの仕事の他に、現在開発中の自身のゲーム Ultimate Action Hero に取り組んでおり、2019 Unity 2D Challenge で 2 位に入賞しています。

Jarek は Unity 2D サンプル Dragon Crashers、Happy Harvest、Gem Hunter Match、および Bunny Blitz のアートディレクターおよびアーティストでした。

Unity からの貢献者

Eduardo Oriz がこのガイドの制作を担当しました。彼は Unity のシニアコンテンツマーケティングマネージャーで、ゲーム開発チームとともに働いた豊富な経験があります。彼は 2D ツールチームと密接に連携しており、Unity がゲーム開発者やスタジオに提供する内容について幅広く理解しています。

Rus Scammell は Unity の 2D 開発チームのプロダクトマネージャーです。Rus はゲームおよびソフトウェア開発において 15 年を超える経験があります。彼はゲーム技術に関する広範な知識を活用して、アーティスト、プログラマー、ゲームデザイナーが Unity 2D ツールとワークフローにアクセスできるようにしています。

Melvyn May は Unity の 2D 物理システムの主要開発者であり、Bunny Blitz および 2D 物理サンプルの作成における重要な協力者です。Melvyn は Unity コミュニティで活発に活動しており、ユーザーの質問に答えるために Discussions スレッドに頻繁に登場します。

Unity の 2D 開発チームのその他のメンバーは、このガイドのレビューと検証に貢献しています。2026 年以降に Unity でユーザーが作成する 2D ゲームを楽しみにしています。

Unity の 2D - 基本

新しい 2D プロジェクトを開く方法

この e-book に沿って次のインストールの手順に従ってください。

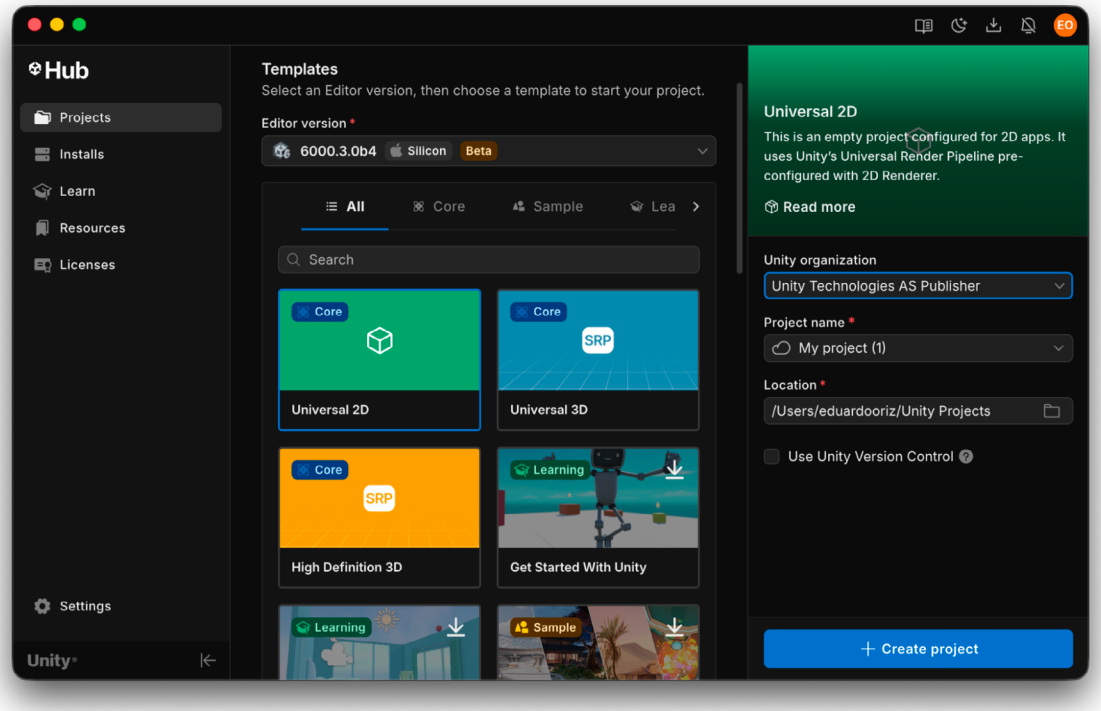
1. Unity Hub から最新の Unity 6 バージョンをインストールします。この e-book の作成の時点では、Unity 6.3 が最新の公式リリースです。
2. Hub で、ユニバーサル 2D テンプレートを使用して新しいプロジェクトを作成します。これはユニバーサルレンダーパイプライン (URP) を使用するテンプレートです。Unity の 2021 年以降のバージョンでは、2D レンダラーはすでに URP と連携するように設定されています。

2D テンプレートには以下のようないくつかのパッケージが含まれています。

- **2D Animation**: ランタイムでネイティブのスケルタルアニメーションを実装します。
- **2D Aseprite Importer**: Aseprite というピクセルアート作成ツールから .aseprite ファイルのインポートを可能にします。
- **2D Common**: 複数の 2D パッケージで使用されるコードが含まれます。
- **2D Pixel Perfect**: スナップやアップスケールの回転などの機能を備え、異なる解像度や比率で鮮明なピクセルアートを確保するためのカメラコンポーネントを提供します。
- **2D PSD Importer**: マルチスプライトキャラクターアニメーションに便利なレイヤー化された .psd および .psb ファイルをサポートします。
- **2D Sprite**: スプライトプロパティを設定するために使用されるスプライトエディターウィンドウです。
- **2D SpriteShape**: 形状のアウトラインに沿ってスプライトをタイリングし、アウトラインの角度に基づいてスプライトのデフォームおよびスワップを自動的に実行します。

— **2D Tilemap Editor**: 大規模なグリッドベースの世界を構築するためのタイルマップを作成および編集するためのものです。

— **2D Tilemap Extras**: カスタムタイルやブラシなど、2D タイルマップで使用するための追加スクリプトが含まれます。



Unity Hub ウィンドウ

Universal 2D テンプレートには、2D ゲームに最適化されたプロジェクト設定がいくつか用意されています。

— 画像はスプライトとしてインポートされ、Sprite モードに設定されます。

— シーンビュー は 2D に設定されています。

— デフォルトのシーンには、Global Light 2D が含まれています。

— デフォルトのスプライトレンダラーのマテリアルは Sprites-Lit-Default です。

— カメラのデフォルト位置は 0, 0, -10 です。

— カメラは Orthographic に設定されています。

— Lighting ウィンドウ設定は以下の通りです。

  — すべての Global Illumination が無効に設定されています。

  — Skybox Material は、なしに設定されています。

  — Environment Lighting の Source は Color に設定されています。

2D ゲームのための URP

ユニバーサルレンダーパイプライン (URP) は 2D および 3D ゲームのための Unity のデフォルトレンダラーです。これは Unity のビルトインレンダーパイプラインの後継であり、学習やカスタマイズがしやすく、Unity がサポートするすべてのプラットフォームに効率よく拡張できるよう設計されています。Unity 6.x サイクルは、ビルトインレンダーパイプラインと同等の機能を提供し、多くの面でその品質レベルやパフォーマンスを上回っています。

URP は 2Dライティングのためのグラフィックスパイプラインを提供しており、2D ライトやライティング効果を作成することができます。URP は Sprite Renderer、Tilemap Renderer、Sprite Shape Renderer などの 2D レンダラーや、MeshRendererや SkinnedMeshRenderer などの 3D レンダラーと統合されています。また、Shader Graph、VFX Graph、ポストプロセスエフェクト、およびカメラスタッキングと互換性もあります。

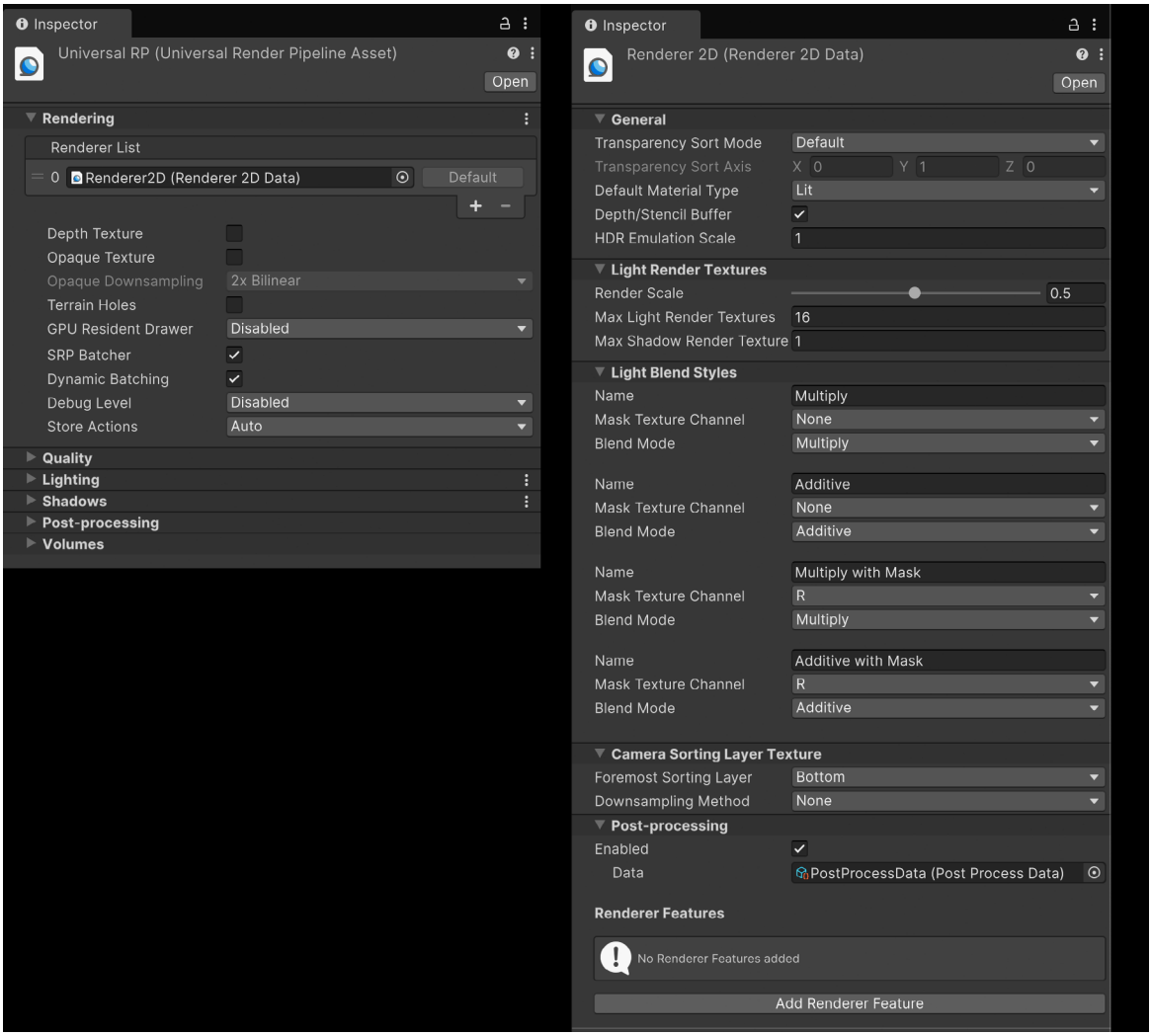
URP 設定

URP の基本設定は、プロジェクト内の以下の 2 つのアセットで利用できます。

— **ユニバーサルレンダーパイプラインアセット** は、Quality、Lighting (3D)、Shadow (3D)、および Post-processing の設定が可能です。

  異なるハードウェアごとに設定を変えたい場合は、複数の URP アセットで品質設定を行い、必要に応じて切り替えることができます。
  URP アセットは **Renderer List** を介して **Renderer Data Asset** に リンクされています。新しい URP アセットを作成すると、1 つのアイテム (同時に作成され、デフォルトとして設定される Renderer Data Asset) で構成される Renderer List が作成されます。このリストには、代わりの Renderer Data Asset を加えることができます。

— **Renderer Data Asset**は、レンダラーが動作するレイヤーのフィルタリングを行い、レンダリングパイプラインに介入してシーンのレンダリング方法をカスタム化するために使用できます。このアセットで、URP の高レベルのレンダリングロジックとパスを制御します。フォワードパスとディファードパス、そして 2D ライト、影、ライトのブレンドスタイルなどの機能を可能にする 2D レンダラー をサポートします。URP を拡張して、独自のレンダラーを作成することもできます。



Unity 6 の 2D プロジェクトにおける URP 設定アセット

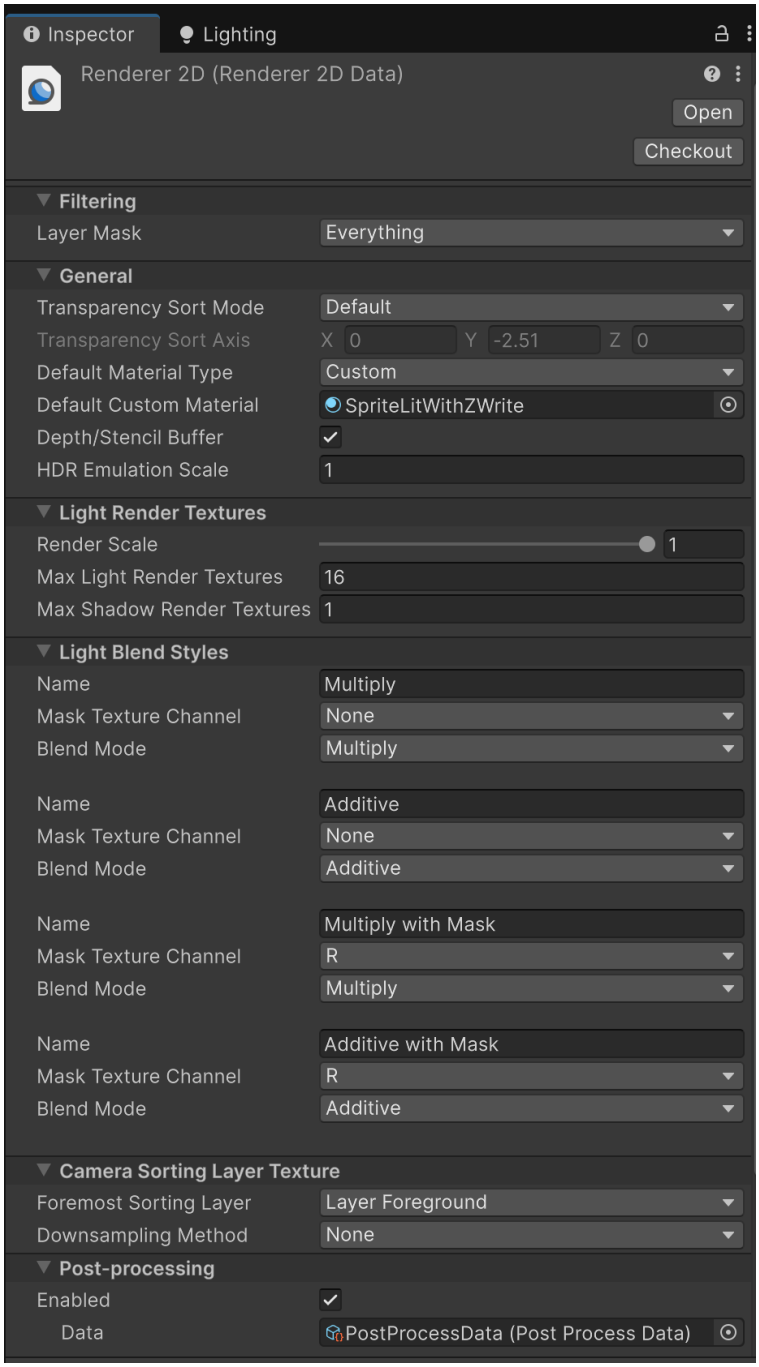
この e-book は、URP に関する技術的および視覚的な可能性すべてを紹介しているわけではありません。URP の完全な機能を理解するには、Unity から以下の e-book をダウンロードしてください。

e-book を入手する

この e-book を入手する

2D レンダラーの設定

Unity 6.3 における 2D レンダラーの設定の概要です。詳細についてはドキュメントを参照してください。



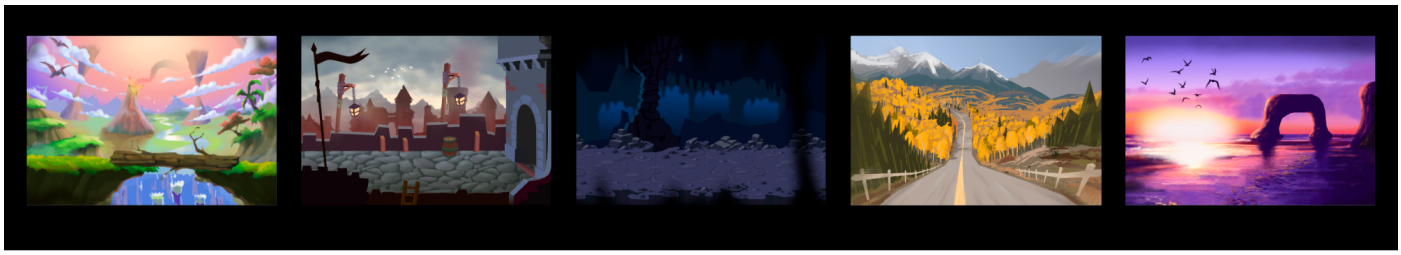
- **Layer Mask:** レンダリングする予定のレイヤーです。デフォルトでは Everything に設定されています。
- **Transparency Sort Mode:** 使用するカメラの種類に基づいて スプライトをソート します。
- **Default Material Type:** 新しい 2D オブジェクトに加えるマテリアルを定義する際に便利です。2D ライトを使用する場合は Lit、使用しない場合は Unlit、または独自のマテリアルには Custom を選択してください。
- **Depth/Stencil Buffer:** デフォルトで有効になっています。Z 深度と透視法を使用することで深度効果を利用できます。
- **HDR Emulation Scale:** 動的範囲を使用する際の再マップに使用されるスケールです。
- **Light Render Textures:** 2D ライトテクスチャを作成する際のスケールと制限値です。値を小さくすることで、2D ライトを使用する際のパフォーマンスを向上させます。
- **Light Blend Styles:** 2Dライトに使用できる 4 つのブレンドスタイルです。RGB 値を加算、乗算、減算し、マスクマップチャンネルのみに影響を与えます。
- **Camera Sorting Layer Texture:** 2D レンダラーによって作成された画像に Shader Graph で アクセスし、その画像に効果 (水の反射など) を適用することができます。
- **Post-Processing:** 処理後にフレームに効果を追加します。

2D アートとアセットの準備

アートの構想段階では、プロジェクトの技術的側面に影響を与えるような多くの決定を下す必要があります。

ゲームデザインでは、ゲームのアートスタイル、アクション、UI を表現するためにスクリーンショットや一連の画像のモックアップを作成することが一般的です。モックアップによって、短時間でコンセプトが実現可能かどうかを判断し、最終的なゲームの見た目を把握できます。

シンプルなサムネイルモックアップを使用して、複数のアートディレクションを確認することもできます。これらのモックアップは簡略化された形状で、全体的な外観、カメラアングル、オブジェクトのサイズ、カラーパレット、コントラストに焦点を当てます。サムネイルを使用することは、チームにさまざまなアプローチを提示し、テストとイテレーションを行うための素晴らしい方法です。



シンプルなサムネイルを描画してアイデアを試す

モックアップを作成する際に以下を考慮する必要があります。

- ゲームのカメラアングルと視点をどのようにするか。
- 対象のプラットフォームの画面サイズに対して、プレイヤーキャラクターのサイズはどのくらいにするか。
- アートスタイルは対象のプラットフォーム、テーマ、オーディエンスに合っているか。例えば、そのスタイルはカジュアルゲーマー、若いプレイヤー、戦略ゲームのファンなどにとって魅力的か。
- アートスタイルがグラフィックスの方針全体とどのように調和しているか。
    - ゲームプレイで素早い反応時間が必要な場合、プレイヤーキャラクター、敵、発射物などの要素を背景に対して一目で視認できるようにしておくのが望ましい。
    - モバイルゲームは、日光の下や小さな画面でも見えるように、明るく、そしてコントラストを高くする必要がある。
- UI 要素のサイズ、位置、可視性はどのようにするか。

技術的な考慮事項

モックアップ段階では、以下のような重要な技術的問題を考慮する必要があります。

- **アニメーション:** どの要素をスケルタルアニメーションでアニメーション化するか考えます。また、どの要素にフレームごとのアニメーションを加えるか、またはどの要素をシェーダーでアニメーション化するかを考えます。

— **環境:** タイルマップで作成するか、またはスプライトシェイプで作成するか、それとも、シーンにプラットフォームスプライトを手動で配置するかを考えます。これらのツールの外観を模倣するためにモックアップをペイントして、モックアップをゲーム内で直接使用できるようにできます。

— **ソート:** Unity でのソートを想定し、それと同様にモックアップのレイヤーをグループ化して、スプライトのソートを計画します。少し時間はかかるかもしれませんが、本番環境で数千のスプライトをソートする必要がなくなります。

— **ライティング:** スプライトをあらかじめ影やライトを描き込む形で表現するか、それともリアルタイムのライティングを使用するかを考えます。1 つのコツとして、ライティングせずに画像をペイントし、次に画像編集ソフトウェアで別のレイヤーに影と光を加えることが挙げられます。そうすれば、本番環境で必要に応じていつでもライティングの外観を変更できます。

この段階で、最終ゲームでの外観にできるだけ近いアセットを作成するよう時間をかけてください。これにより、より迅速にコンセプトからゲーム制作に移行できます。

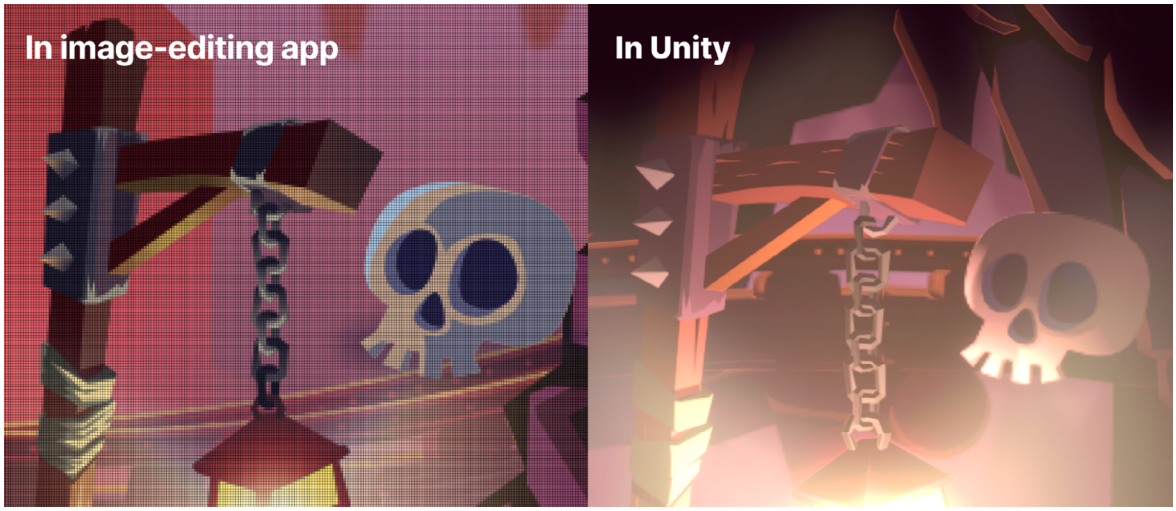
Unity サンプル Dragon Crashers の元のコンセプトアートと最終版の比較

アセットの解像度

Unity の 2D ツールは、最初は 3D ゲーム用に構築されたエディターから進化しました。そのため、いくつかの独特の機能があります。例えば、シーンビューの 2D スプライトは、画面解像度に厳密には結びついていません。Unity のスプライトは、メッシュに描画されたテクスチャであり、簡単にスケールできます。2D ゲームのカメラもスケーラブルなので、必要に応じてズームインおよびズームアウトできます。

したがって、Unity で 2D コンテンツを作成するには、Adobe Photoshop、Serif の Affinity Photo、GIMP、Krita などの従来のラスターグラフィックスソフトウェアでの作業とは異なるアプローチが必要です。

これらの従来のアプリケーションでは、設定された解像度で特定のドキュメントキャンバスサイズが利用可能であり、すべてのレイヤーがこの解像度に結びついています。各レイヤーのピクセルサイズとドキュメントのピクセルサイズの間の比率は 1:1 です。



ゲームアセットを描画する際、ピクセルサイズは一定であり、画像は常にピクセルグリッドに収まるのでとても良い外観が得られます。しかし、Unity では、スプライトには個別の解像度を設定できます。カメラのズームレベルも最終的な外観に影響を与えます。

ただし、Unity では、画面とアセットの解像度は互いに独立しているため、スプライトの解像度を計算する必要があります。

まず対象となるプラットフォームから始めます。なぜなら、ハードウェアの性能によって設定できる最大解像度が決まるからです。

モバイルデバイスの場合、解像度の範囲は広いものの、1920x1080 と想定すると安全です。これにより、ローエンドからハイエンドまでのデバイスに対応できます。

Steam の調査 によると、PC の場合はデスクトップゲーマーの大多数はフル HD (1920x1080) を使用しています。ウルトラワイドスクリーンへの対応も検討する場合もあるかもしれません。21:9 などの画面比率をテストして、不要なレベルの領域をカメラで表示しないようにしてください。

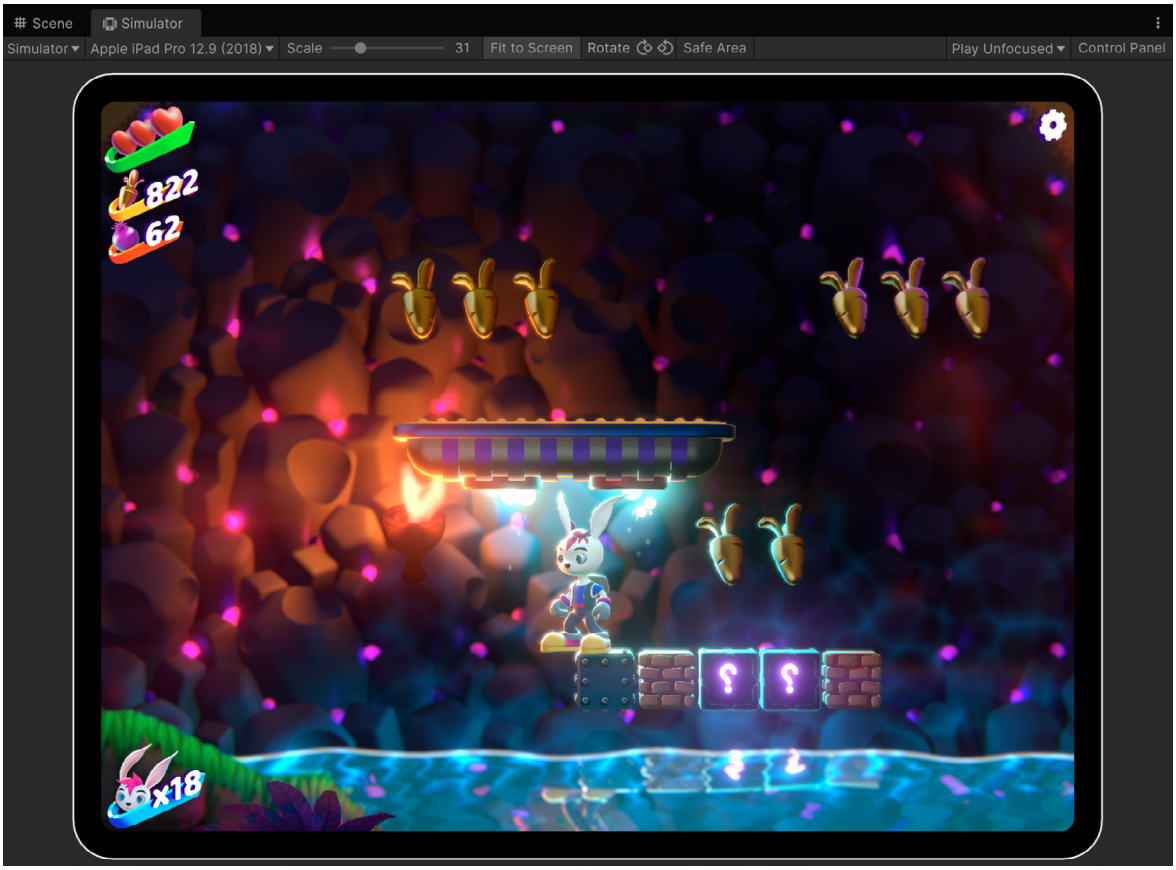
フル HD または 4K を対象とする場合、以下の 2 つの最善の方法を心に留めておいてください (以下のルールはピクセルアートには適用できません)。

— 対象デバイスの最高解像度でアートをペイントしてください。ラスターアートを拡大しないでください。ピクセル化やぼやけの原因になります。

— すべてのアセットに対して単一の解像度を維持し、必要に応じて後で解像度を下げてローエンドデバイスに対応させるようにしてください。

役立つコツの 1 つは、必要なサイズの 2 倍の大きさでアートを描画し、Unity にエクスポートする際に 50% に縮小することです。この技術により、スプライトの仕上がりが良くなり鮮明に見え、ブラシの線がぶれなくなります。アートは縮小されるので、細部にこだわりすぎないようにしてください。これは手描きのアートで生じる小さな欠陥を隠すためのよい方法です。もちろん、ゲームに手描き風のスタイルを望む場合は、この方法は試さないでください。

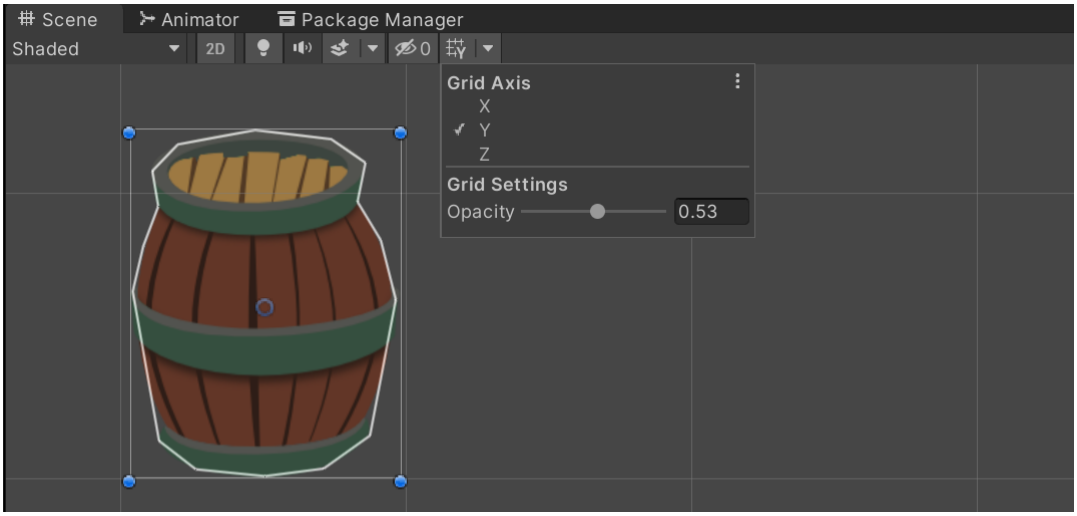
解像度を選択したら、ゲームビューでアートの外観をテストして、対象のデバイスでどのように表示されるかを確認してください。



エディターのゲームビューとシミュレータービューで、さまざまな比率と解像度でゲームがどのように見えるかを簡単にプレビューできます。

わかりやすくするために、透視投影ではなく平行投影ビューでスプライトの解像度を計算してください。視覚化を容易にするために、スケールを 1,1,1 に設定し、Z 深度を 0 に設定します。

Unity のグリッドとユニットを使用して、スプライトの配置と外観を一定に維持し、カメラのズームやオブジェクトのサイズを計算します。



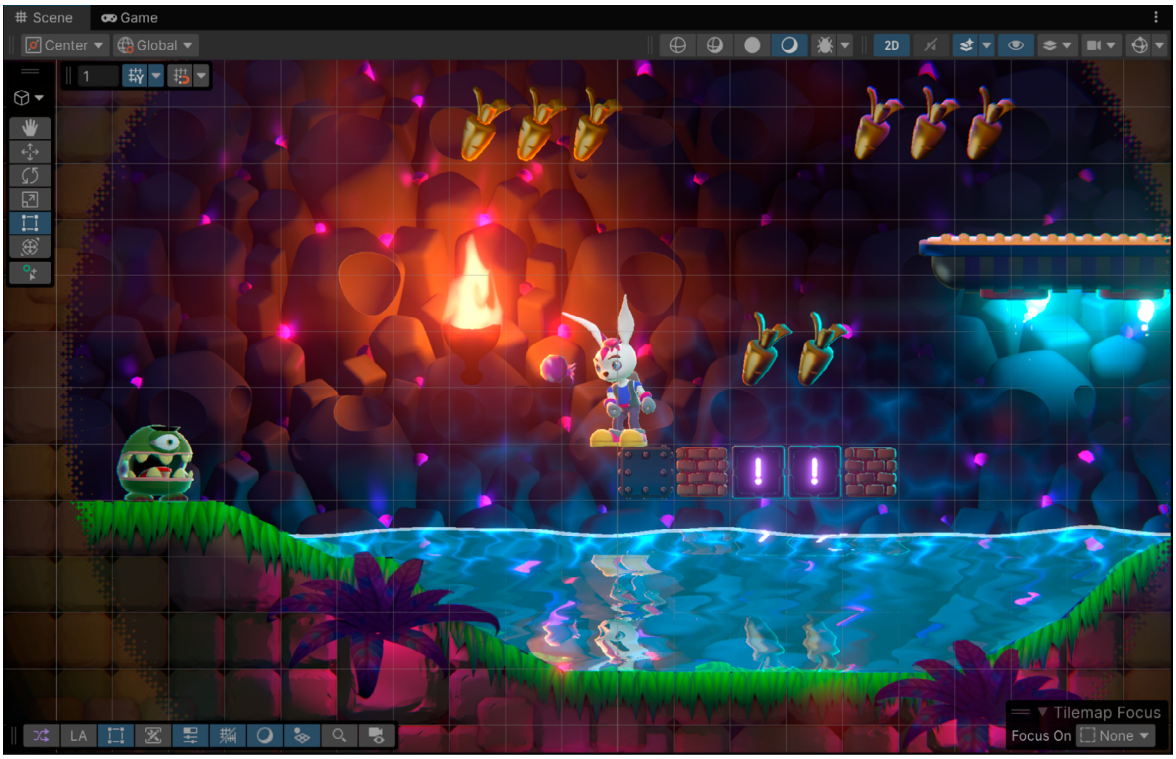
シーンビューでの Unity のグリッド

シーンビューのグリッドによって Unity ユニットが視覚化されます。1 Unity ユニットは 1 メートルに等しいと仮定します。まず基本サイズを設定し、ゲーム全体で一貫性を保ちます。

プレイヤーキャラクターのサイズから始めます。プレイヤーキャラクターの背の高さは 0.5 から 2 ユニットの間に保ってください。タイルマップ を使用している場合は、タイルサイズを 1 ユニットに設定します。

キャラクターやオブジェクトが、ゲーム内の他のビジュアル要素と比べて小さすぎたり大きすぎたりすると、Transform の値がおかしくなったり、物理演算で問題が発生したりする恐れがあります。

メインシーンでの基本オブジェクト (プレイヤーキャラクター、敵、コレクタブル、レベルの危険要素) のサイズを確立したら、平行投影カメラの Size プロパティを設定してズームレベルを選択します。その後、カメラサイズの値を確認します。この値に 2 を乗算すると、Unity ユニットでのカメラの垂直サイズが得られます。



画面に表示されるゲームプレイエリアの大きさを早めに決定しましょう。例えば、余裕のないプラットフォーマーを開発している場合、ゲームデザインやアートの準備のために、早い段階でこの大きさを確定しておく必要があります。

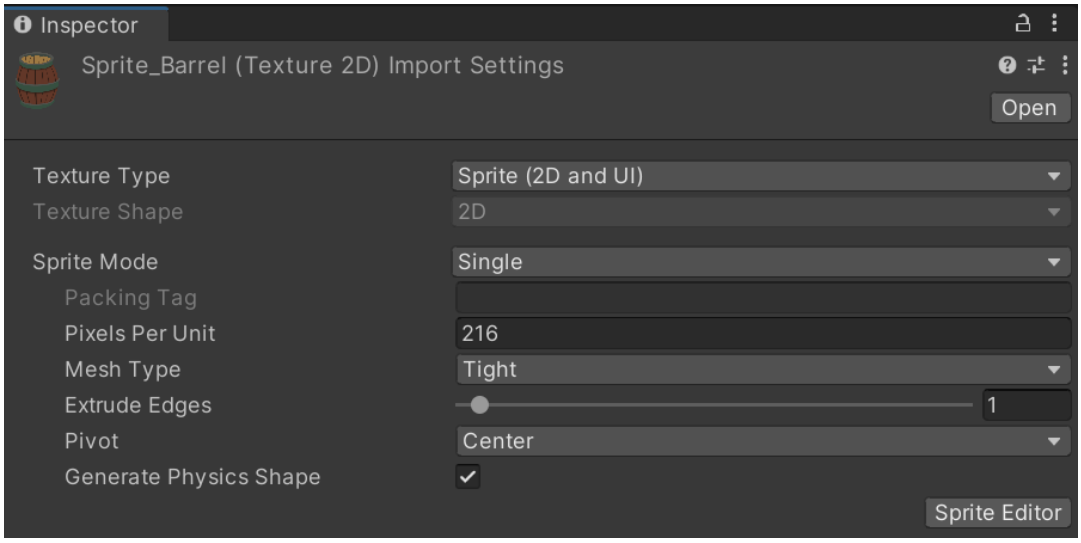
平行投影カメラのサイズは、カメラの中心から上部までの垂直サイズを示すユニットで表されます。

カメラサイズが 5 の場合、画面の中心から上部までが 5 ユニットということであり、合計の高さは 10 ユニットになります。例えば、4K 解像度を対象にする場合、画面 (またはカメラ) の高さは 2160 ピクセルです。簡単な計算で、アートに必要なユニットあたりのピクセル数 (PPU) を求めることができます。

2160 :10 = 216

垂直方向の最大解像度 : (平行投影カメラのサイズ *2) = スプライトの PPU

つまり、ネイティブの 4K 解像度で美しく表示するために、ゲーム内のすべてのスプライトで約 216 PPU の解像度が必要になるということです。



スプライトの PPU を設定する

簡単な例を挙げます。カメラをズームインおよびズームアウトさせたい場合は、それを考慮する必要があります。平行投影ビューで最大ズーム率をサイズ 3 に設定する場合、PPU は 360 (2160: (3*2)) である必要があります。

スケルタルアニメーションを使用する場合、スプライトの解像度を推奨 PPU よりも少し高く設定してください。スケルトンのメッシュに回転、伸縮、歪みが生じ、極端な状態になった場合は、奇妙な結果や見栄えの悪い結果を生じる可能性があるため、これは必要な設定です。

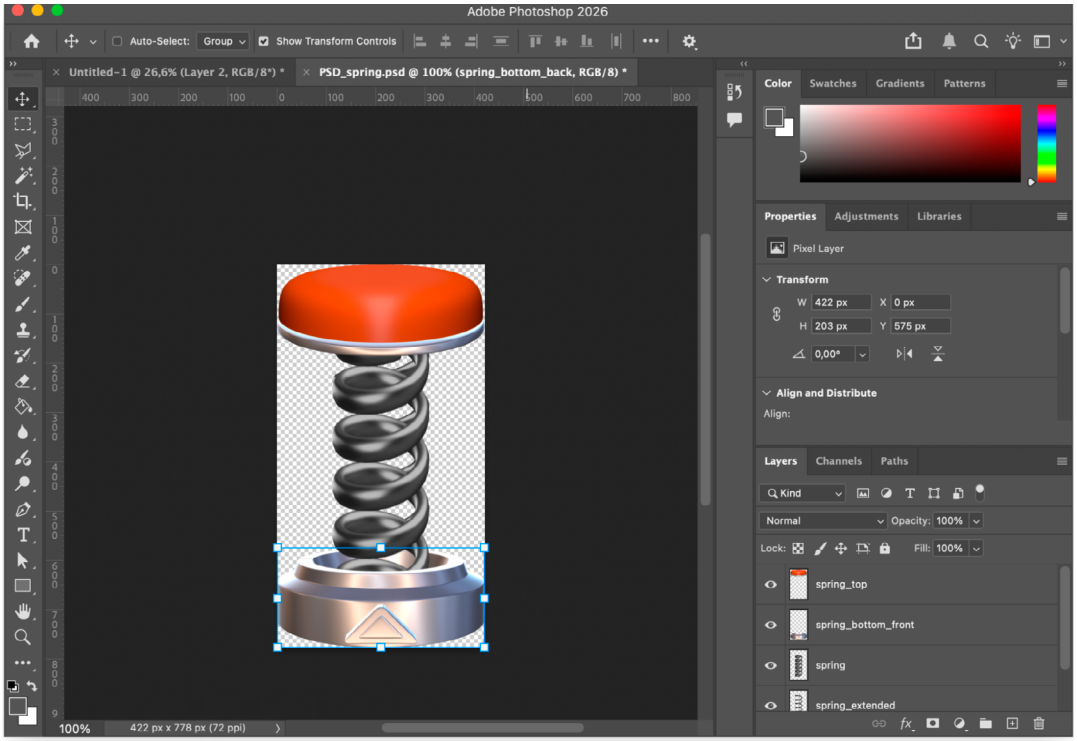
デバイスでゲームをテストして、どのように見えるかを確認してください。多くの場合、高解像度のアセットを使用する利点は明白には現れません。代わりに、この描画時間とデバイスメモリを、より重要な他のゲーム要素 (主要キャラクター、視覚効果、または UI など) に割り当てることができます。モバイルデバイスでは、ゲームサイズが重要です。どのアセットを縮小できるかを確認し、メモリ制限に注意してください。2D アセットの解像度の詳細については、こちらのブログ記事を参照してください。

Photoshop の PSD ファイルを Unity に取り込む

スプライトは PNG ファイル形式でエクスポートするのが最も一般的なワークフローですが、PSD ファイルを Unity に直接インポートすることもできます。デフォルトでは、2D PSD Importer パッケージがインストールされていない場合、Unity は PSD レイヤーを 1 つの画像にフラット化します。これは、2D テンプレートからプロジェクトを開始した場合に該当します。

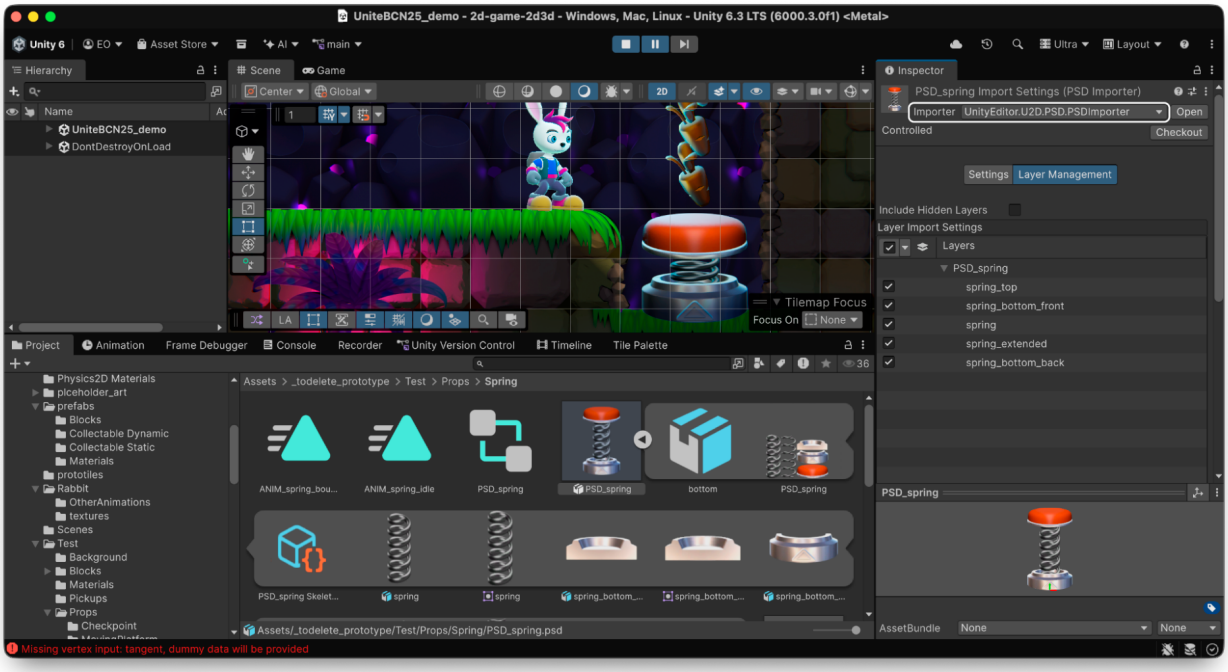
Unity の 2D PSD Importer パッケージは、レイヤーを個別のスプライトとしてインポートするオプションを提供します。このパッケージは 2D アニメーション用に設計されていますが、これを使用して、1つの PSD ファイルに含まれる通常のスプライトを複数インポートすることもできます。

PSD レイヤーにはさまざまなブレンドモード、不透明度、エフェクトを設定することが可能です。Unity で正しく表示されるスマートオブジェクトにすることもできます。



Photoshop でのプロップの作成

PSD ワークフローは、ゲームスプライトや UI 要素に使用する際に非常に便利です。編集、ペイント、または新しいレイヤーの作成が可能で、ファイルを保存すると、変更がエディターに即座に表示されます。PSD ソースファイルが Unity プロジェクトに直接存在する場合、そのファイルは Unity Version Control のようにプロジェクトのソース管理に加えることができます。



中間のスタンドアロン PNG ファイルを必要とせず、レイヤーが Unity ですぐに使用できる状態でインポートされたPSDプロップ

描画の順序

2D ゲームでは、すべてのスプライトとオブジェクトに同じ深度が設定されています。いくつかのスプライトを他のスプライトの前面に表示するにはどうしたらよいでしょうか。

Unity では、レンダラーをそのタイプと使用法に基づく優先順でソートします。レンダラーのレンダリング順をレンダーキュー を使用して指定できます。一般的に、不透明キュー と 透明キュー という 2 つの主要なキューがあります。2D レンダラーは主に透明なキュー内にあり、スプライトレンダラー、タイルマップレンダラー、およびスプライトシェイプレンダラー の各タイプが含まれます。

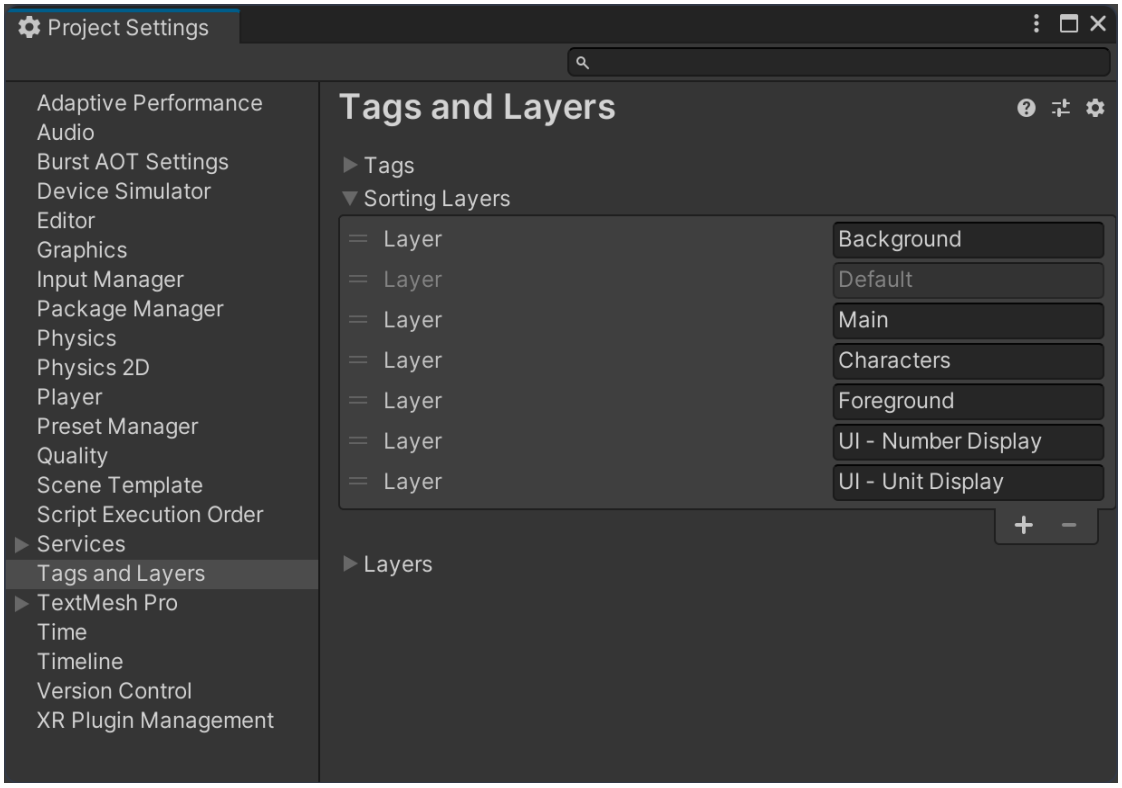
透明キュー内の 2D レンダラーは一般的に優先順位に従います。2 つ以上のオブジェクトが同じスペースを占有している場合、Unity はこのリストを確認し、どのオブジェクトを上に描画するかをチェックします。優先順位が同じで、両方のオブジェクトに同じ値が設定されている場合は、以下のリストの基準が考慮されます。

1. カメラまたはカスタム軸までの距離
2. レイヤーの順序
3. ソートレイヤー
4. ソートグループ
5. レンダーキューの指定
6. マテリアル/シェーダー

両方のオブジェクトで上記のすべての値が同じで、タイブレイカーが必要な場合、このプロセスでどちらを上に描画するかを選択する必要があります。理想的な解決策ではないため、ソートレイヤーとソートグループを使用して明確なソート順を設定するようにしてください。

ソートレイヤー

最も重要なソート基準はSorting Layers (ソートレイヤー) です。すべての 2D レンダラーにこのオプションがあり、最初に設定する必要があります。編集可能なデフォルトのソートレイヤーがあり、Project Settings を開いて Tags and Layers のオプションを指定することで編集できます。



Sorting Layers の編集

レイヤーの左側にあるハンドルをドラッグして、追加、削除、または順序の変更を行います。リストの上位にあるレイヤーは最初に描画され、カメラから遠くに表示されます。

シーンを早期に整理するためにモックアップを作成する際にも、ソートレイヤーの構造を計画します。スプライトをシーンに配置したらすぐに、そのソートグループを設定します。これらの手順を実行すると、プロジェクト開発の後半で、数千のスプライトのソート設定を変更しなければならないことに突然気づくという状況を回避することに役立ちます。

ソートレイヤー過多の回避

2D ライトはソートレイヤーに依存しているため、ソートレイヤーの構造を設定する際にライティングを考慮します。ゲーム内でライトがどのように動作するか、どのグループが影響を受けるかを事前に把握しておきます。

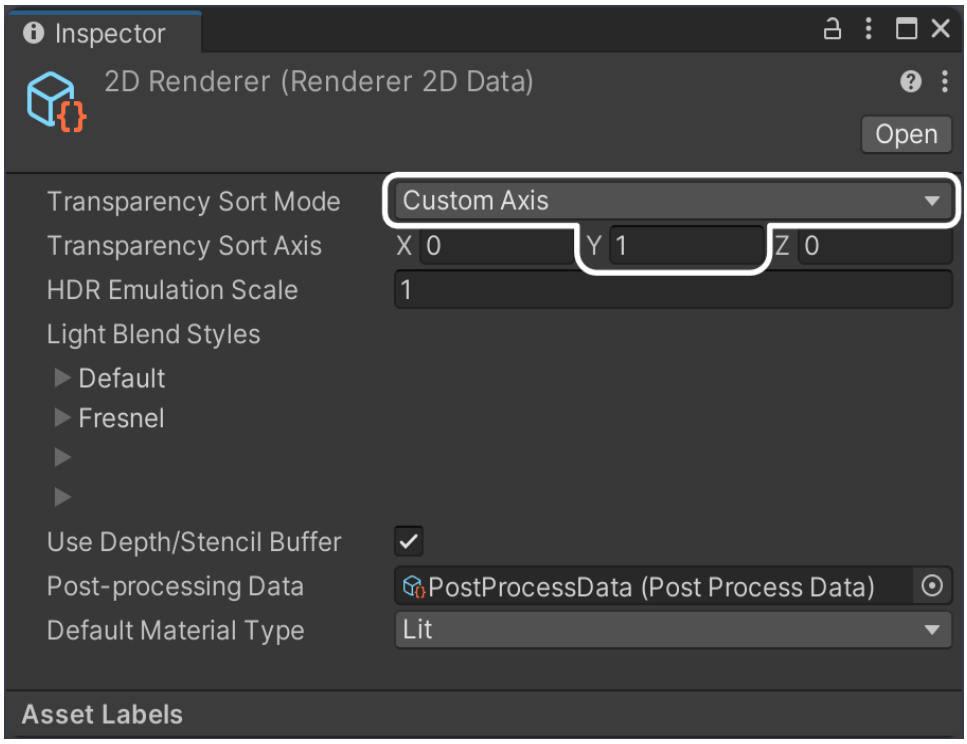
例えば、アイソメトリックゲームを作成していて、2D ライトを使用するトーチを設置する場合、トーチは周囲全体に影響を与えるでしょうか、それとも壁だけに影響を与えるのでしょうか。キャラクターは影響を受けますか。それともライトの前面に描画されるのでしょうか。ライトを受けるオブジェクト専用のソートレイヤーが必要でしょうか。ソートレイヤーを編集する際には、このような点を念頭に置き、2D ライトセクションでのライティング計画についてさらに学びましょう。

使用するソートレイヤーが多すぎると、詳細をすべて見失う可能性があるので注意してください。レンダラーをさらにソートする必要がある場合は、Order in Layer を使用してください。これにより、同じレイヤー上のオブジェクトを簡単にソートできます。

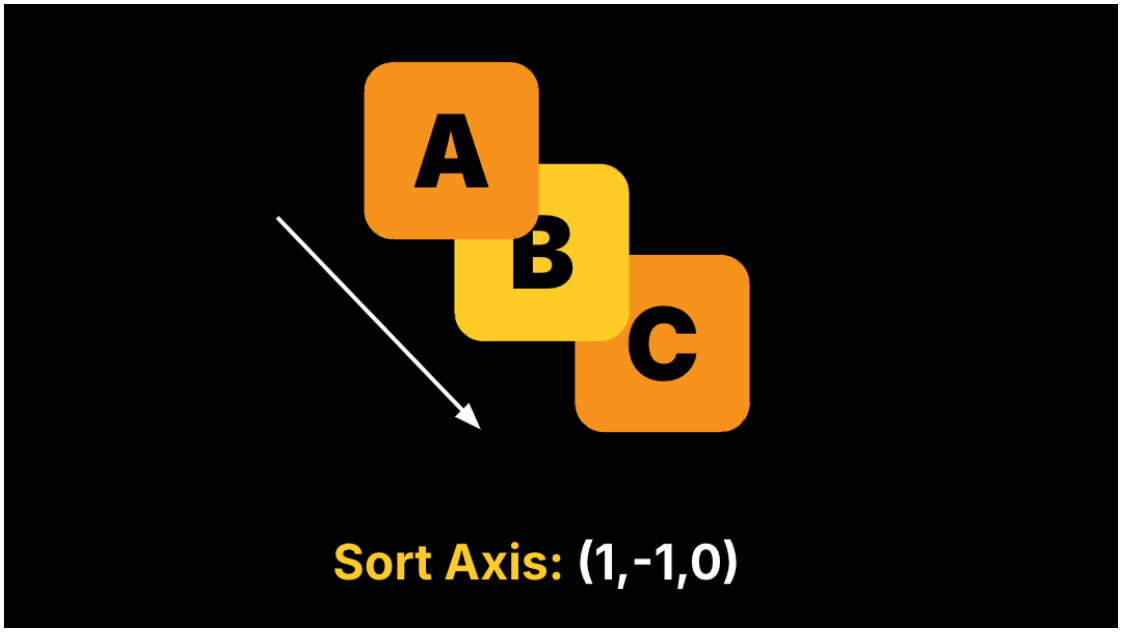
Transparency Sort Mode

平行投影カメラを使用する場合、ゲーム内に偽の 3D ビューを設定する必要がある場合があります。例えば、アイソメトリックなトップダウンゲームのための不等角投影や斜投影、またはバトルゲームのためのキャビネット投影などです。その場合、Distance to Camera (カメラとの距離) でレンダラーをカスタムソートする必要があります。

カスタム軸を選択してオブジェクトをソートします。トップダウンゲームやバトルゲームの場合、Y 軸でオブジェクトをソートし、背の高いキャラクターが他のキャラクターの下に描画されるようにして、遠くにいるように見せます。このオプションを編集するには、URP の設定時に作成された 2D レンダラーアセットを探して、Transparency Sort Mode オプションを Custom Axis に変更し、透明度ソート軸の値を 0, 1, 0 に設定します。



Y 軸でスプライトをソートして、値 (X:0, Y:1) 1 で Unity に対して下向きのベクトル (Vector2.downのように) に従うよう指定します。上部のスプライトが最初に描画され、下部のスプライトが後で描画されるため、下部のスプライトが前面に表示されます。



任意のベクトルをソート軸で表現することができ、このソート軸はスプライトが上から下へレンダリングされる方向となります。

ソートグループ

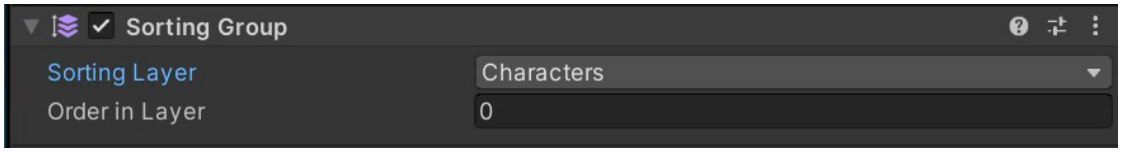
前の手順に従って Y 軸でソートしても、キャラクターのパーツや子オブジェクトが正しく表示されない場合があります。

同じソートレイヤーに存在するキャラクターのパーツが 2 つのキャラクターの間に混在しています。

これはバグではありません。ソートの動作に過ぎません。同じソートレイヤーに存在する場合、ソートではどのパーツがどのキャラクターに属しているかを認識しません。幸いにも、Sorting Group (ソートグループ) はソート優先リストの 4 番目の基準です。

Sorting Group とは、ソートを目的に、共通のルートを持つレンダラーをグループ化するコンポーネントです。同じソートグループ内のすべてのレンダラーは、同じ Sorting Layer (ソートレイヤー)、Order in Layer (レイヤーの順序)、および Distance to Camera (カメラとの距離) を共有します。

複数のスプライトで構成されるキャラクターやその他のオブジェクトを使用する場合は、このコンポーネントを親オブジェクト (階層内で最上位のオブジェクト) に配置し、他の 2D レンダラーと同様に Sorting Layer と Order in Layer を設定してください。この例は、Unite Now 技術セッション で確認することができます。



親ゲームオブジェクトにソートグループを作成して、このオブジェクトとその子オブジェクトが単一の要素としてソートされるようにすることで、ゲーム内のパーツと他のスプライトとの競合の可能性をなくします。

ソートグループで、キャラクターが正しく表示されるようにします。

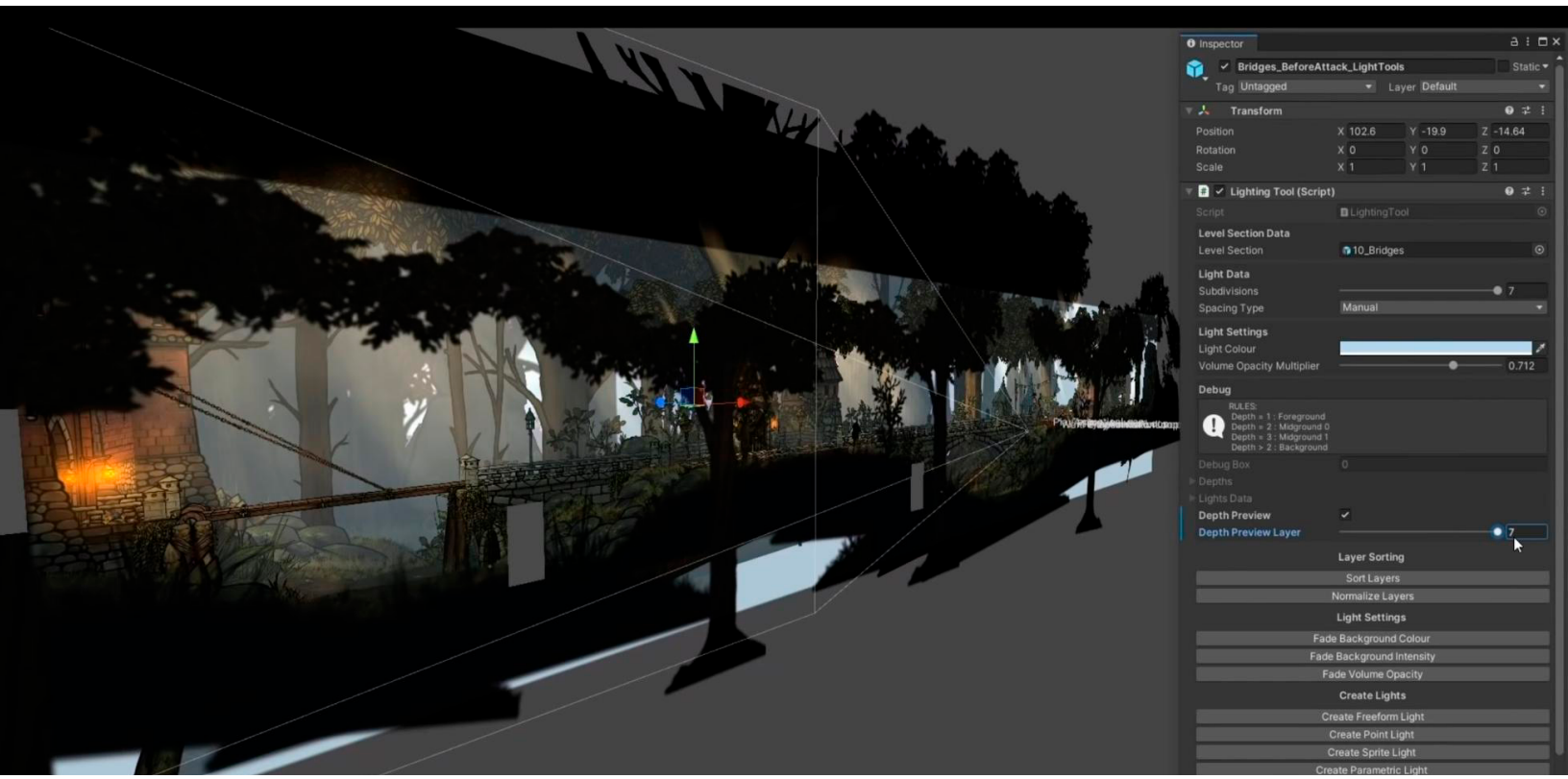
ソートグループをネストすることもできます。複数のスプライトで構成される武器をランダムに生成したい場合は、それらをソートグループに入れて、キャラクターの手に正しく表示されるようにします。

2D におけるカメラと視点

平行投影または透視投影カメラ

ほとんどの 2D ゲームでは、カメラを平行投影モードに設定することをお勧めします。これは 2D プロジェクトのデフォルトの選択肢です。なぜなら、1 次元の線は平行で、2 次元のみで作業するからです。平行投影モードは、ピクセルアートゲーム、パズル、2D アイソメトリック、トップダウン、プラットフォーマーなどのすべてのスタイルに対応しています。

2D ゲームに実際の奥行きはありませんが、視差効果を使用して深度の錯覚を作り出すことができます。視差効果により、カメラが移動する際に、背景と前景の複数のレイヤーが異なる速度でスクロールします。これは、人間の深度知覚を模倣することを目的としています。遠くの物体は近くの物体よりも遅く動いているように見えます。



Oddbug Studio による Tails of Iron の透視投影カメラ設定。Unity 2D ライト でゲームのライティングをどのように作成したかを学びましょう

平行投影モードがすべての 2D ゲームスタイルに適しているなら、なぜ透視投影モードを使用するのでしょうか。

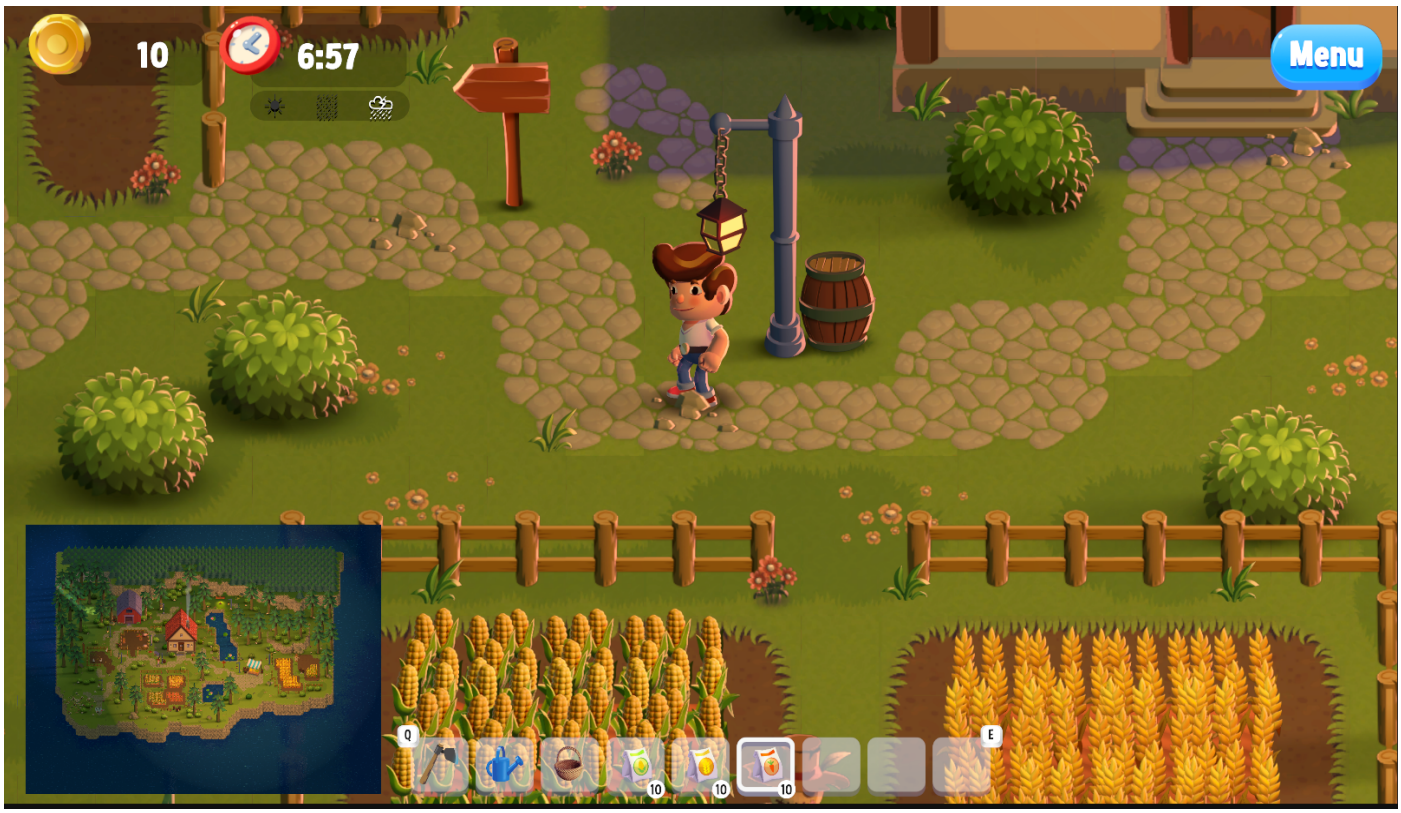
その理由の 1 つは、視差効果に即座にアクセスできるからです。名前が示すように、透視投影カメラは透視投影の視点を使用します。視差レイヤーのスクロールを処理するためにスクリプトを使用する必要はありません。代わりに、Z 軸上でカメラから遠くに要素を配置し、カメラからの相対的な距離に基づいて透視を考慮してそれらを拡大します。

選択したカメラはアート制作プロセスに影響を与えませんが、レベルの設定方法やゲームのルックアンドフィールを決定します。

つまり、平行投影モードは最も一般的な選択肢ですが、高度なスクロールや視差を使用する場合はその限りではありません。その場合、透視投影モードを選択することをお勧めします。

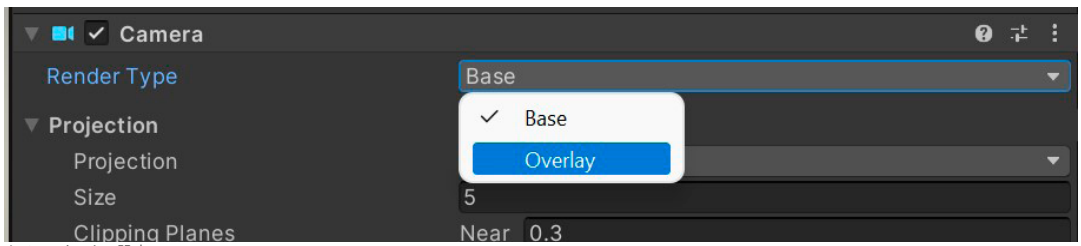
カメラスタック

URP の カメラスタック を使用すると、カメラの出力をレイヤー化し、最終的な画像に組み合わせることができます。カメラスタックを使用して、2D、3D、および UI オブジェクトを組み合わせることができます。



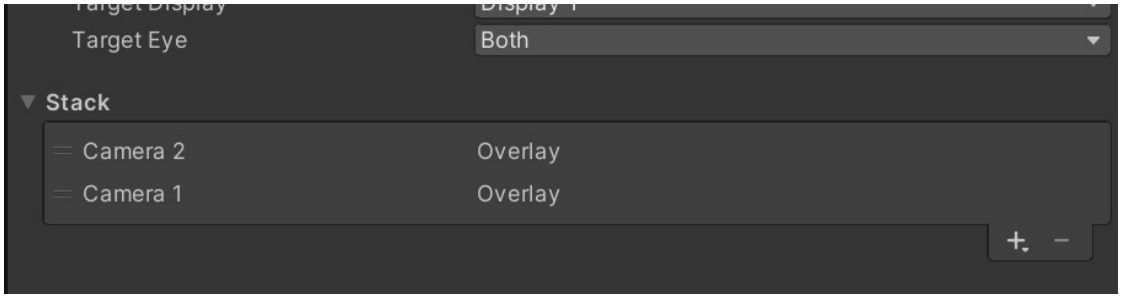
カメラスタックの使用例としては、メインカメラの上にマップの他のエリアを表示することが挙げられます。例えば、画面分割のように 2 人目のプレイヤーがマップの他の部分を探ることを可能にします。この例では、シーンの上空からのビューを提供しています。

スタックには、シーンに少なくとも 2 つのカメラが必要です。最初に描画される画像の基本カメラを選択し、追加のカメラは Overlay に設定する必要があります。



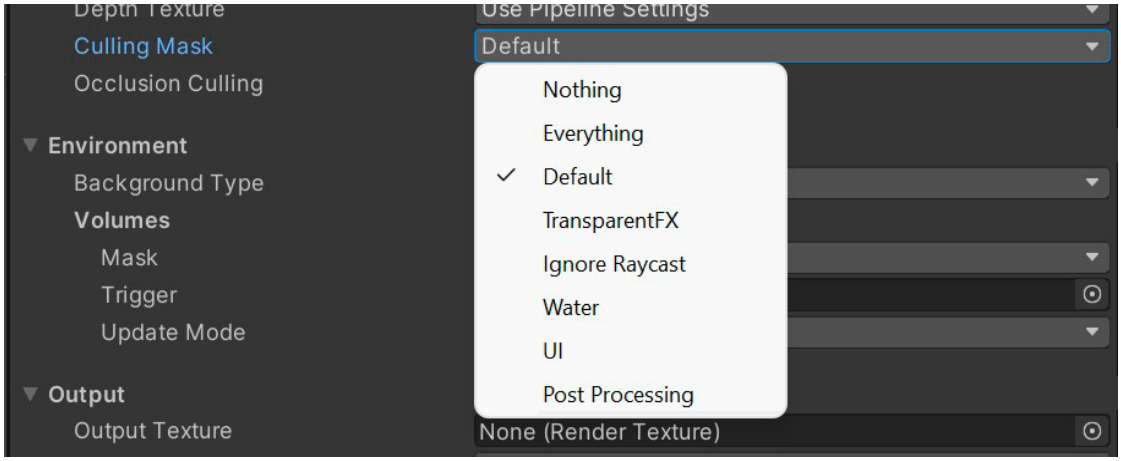
カメラスタックの設定

基本カメラは、どのカメラをオーバーレイとするか、そしてどの順番で重なるかを把握する必要があります。そのためには、Stack オプションに移動し、+ ボタンを使用して Overlay にするカメラを選択します。必要に応じて、オーバーレイカメラの描画順序をリスト内で上下にドラッグして変更します。



オーバーレイとして描画されるカメラの選択

レイヤーを使用して、各カメラによって描画されるオブジェクトを設定します。デフォルトでは、カメラはすべてのレイヤーを描画しますが、Camera コンポーネントの Culling Mask オプションで、カメラで描画されるレイヤーを選択することで変更できます。



カメラの Culling Mask オプションで描画するレイヤーの選択

ここで、カメラに描画させたいゲームオブジェクトを選択し、それらのレイヤーを変更して、先に設定したものと一致させます。そのようにして、異なるカメラによって描画されるようにオブジェクトが分類されます。例えば、3D 背景や 2D プラットフォームレイヤーなどです。

2D のための Cinemachine

カメラのタイプを選択したら、ゲームプレイに合わせて設定する必要があります。Unity の Cinemachine システムで、この設定を行う機能の他に、レベルの境界にカメラを制限したり、カメラの遷移やノイズを設定したりするなどの機能が提供されます。このセクションでは、2D ゲームのための Cinemachine のコア機能のいくつかを取り上げます。

始めに、Package Manager から Cinemachine をインストールしてください。次に、以下のステップを行って、プロジェクトのシーンビューに Cinemachine カメラを追加してください。

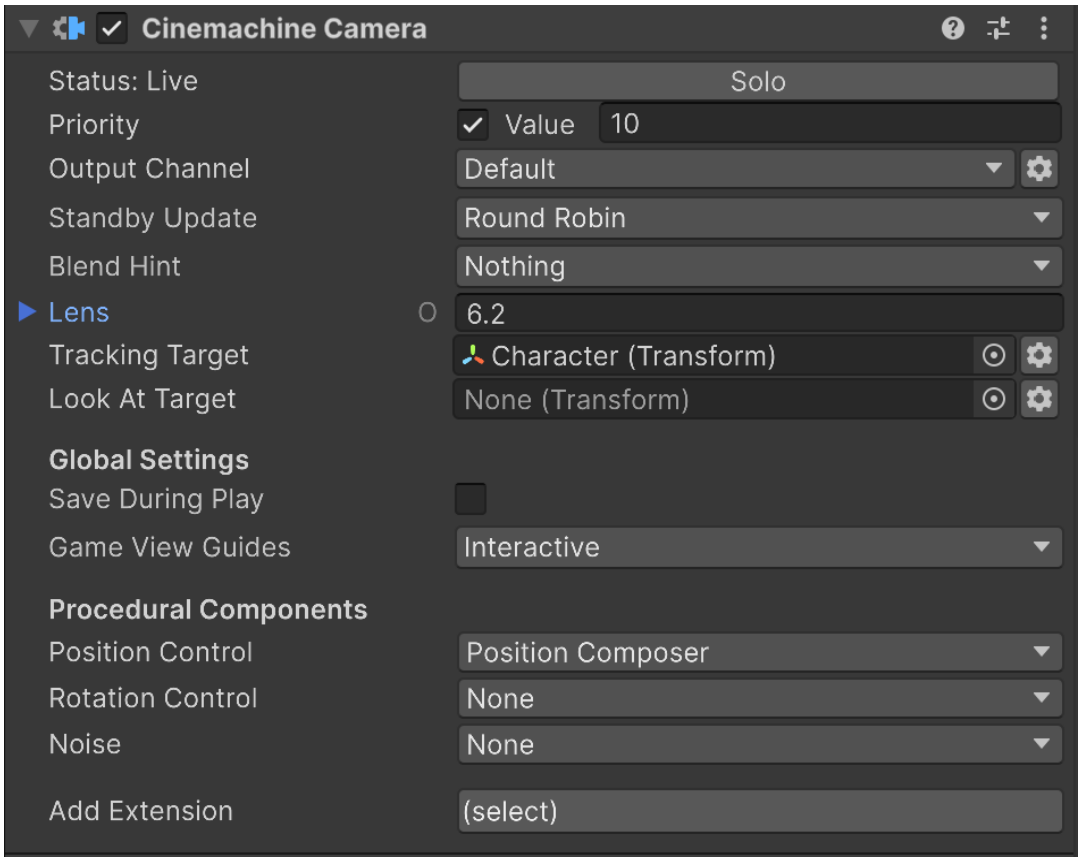
1. シーンビューで、シーン内を移動して、Cinemachine カメラでフレームを設定する視点を決定します。
2. Unity のメニューで、**GameObject** > **Cinemachine** > **Cinemachine Camera** を選択します。Unity では以下を使用して新しいゲームオブジェクトを加えます。
    - Cinemachine Camera コンポーネント
    - シーンビューカメラの最新の位置と方向に一致する Transform

シーンで最初に Cinemachine Camera を作成すると、Unity により自動的に Unity のカメラに Cinemachine Brain コンポーネントが追加されます。ただし、Unity のカメラにすでに含まれている場合は除きます。以下の方法で確認します。

1. Hierarchy で、Unity Camera を選択します (これは Camera コンポーネントを含むゲームオブジェクトです)。
2. Inspector で、ゲームオブジェクトに Cinemachine Brain コンポーネントが含まれていることを確認します。

Cinemachine Brain により、シーン内のすべてのアクティブなバーチャルカメラが監視されます。バーチャルカメラをキーフレームでアニメーション化したり、カメラ間をブレンドまたはスムーズに遷移させたり、その両方を組み合わせて、1 つのカメラをアニメーション化しながら別のカメラに遷移させたりすることができます。アニメーションはすべて Cinemachine Brain によって処理され、メインカメラに適用されます。これは、メインカメラを動かす強力なアニメーションシステムと考えてください。

2D バーチャルカメラを作成するには、**Cinemachine > Create 2D Camera** をクリックします。これにより、2D 環境用に設定されたバーチャルカメラが作成されます。シーンに最初のバーチャルカメラを追加すると、メインカメラに Cinemachine Brain コンポーネントも追加されます。



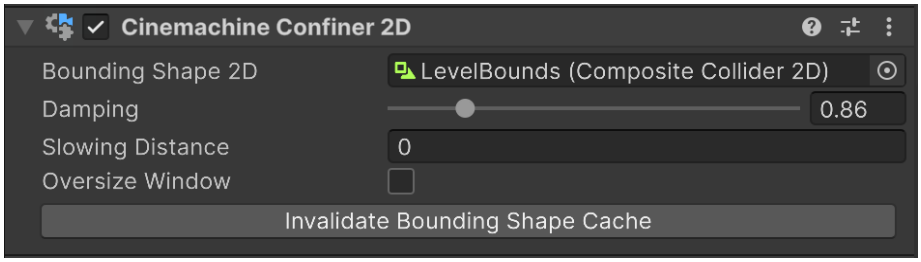
Cinemachine バーチャルカメラの作成

Cinemachine には追跡するオブジェクトが必要なので、Tracking Target (追跡ターゲット) フィールドにプレイヤーキャラクターを割り当ててください。2D 平面で操作していて、カメラがパンしたり傾いたりしないようにするには、Look At Target フィールドが空であり、Position Control が Follow または Position Composer に設定されていることを確認してください。最後に、プロジェクトの Lens プロパティを設定します。いくつかのオプションはメインカメラのプロパティから継承されることに留意してください。

Position Composer コンポーネントには、バーチャルカメラがターゲットを追跡する方法を変更するための便利なオプション (Offset、Damping、Dead Zone、Soft Zone など) があります。再生モード中にこれらのオプションを試してみてください。Save During Play オプションを有効にすると、変更が保存されます。

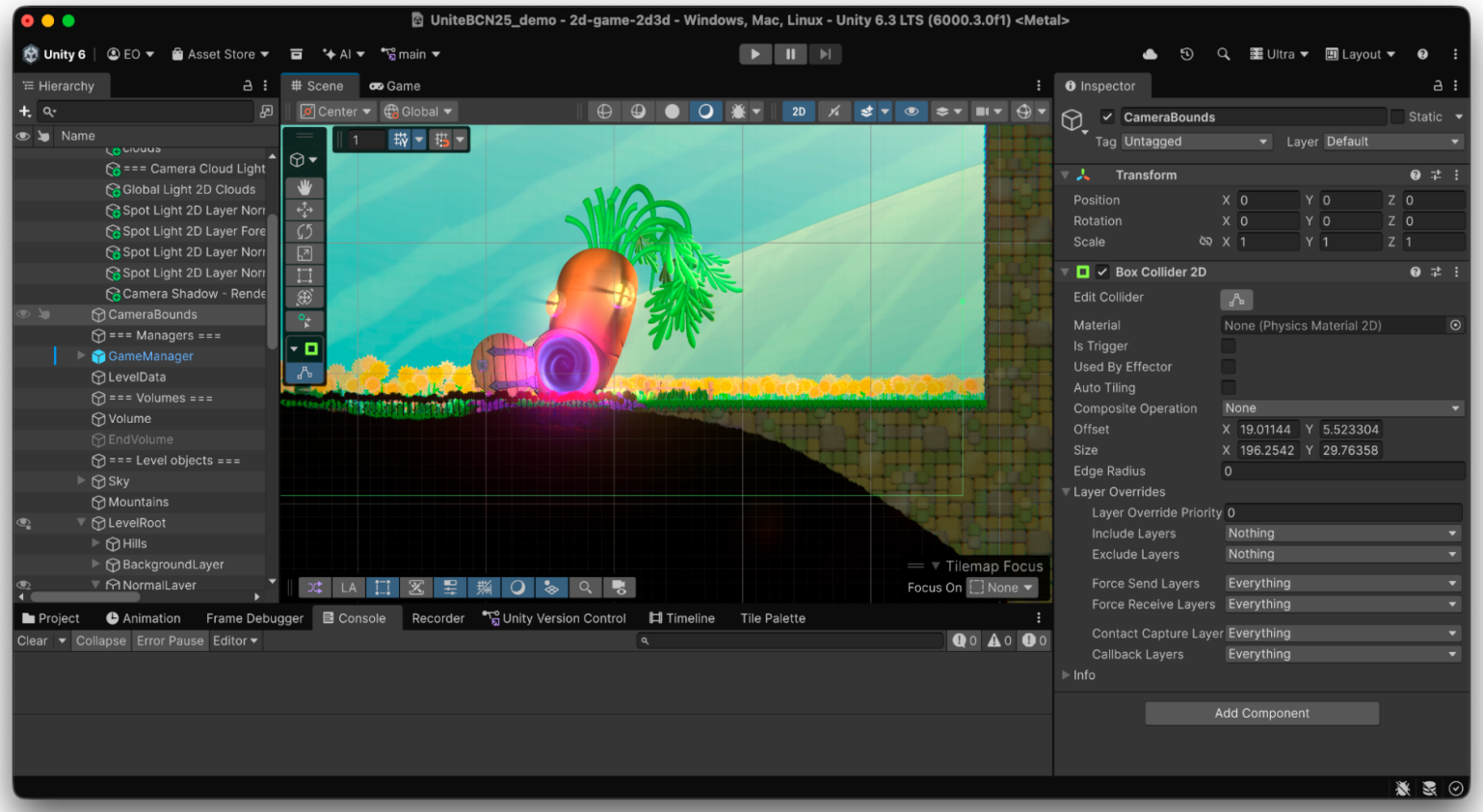
バーチャルカメラ用の拡張機能として Cinemachine Confiner 2D もあります。これを有効にするとカメラがレベルの境界を超えて移動することを制限し、プレイヤーに見せたい部分だけを表示できるようにします。これにより、レベルの不必要な部分を設計する必要がなくなります。

Cinemachine Confiner 2D を追加するには、ドロップダウンメニューの Add Extension から選択します。



Cinemachine Confiner 2D 拡張機能を追加

Cinemachine Confiner 2D では、Bounding Shape 2D として Collider 2D (Composite または Polygon) が必要です。空のゲームオブジェクトを作成し、Composite Collider 2D と Box Collider 2D を追加します。RigidBody 2D が自動的に追加されるので、その Body Type を Static に設定します。また、Box Collider 2D の Used by Composite オプションを有効にしてください。このゲームオブジェクトを Cinemachine Confiner 2D スクリプトの Bounding Shape 2D フィールドにドラッグします。カメラのサイズよりも大きくなるように Box Collider 2D のサイズの編集を忘れないでください。



CameraBounds という Collider 2D は、カメラが境界を超えて表示されることを防ぎます。

これで、カメラの錐台がコライダーのバウンディングボックスを超えることはありません。詳細については、Cinemachine Confiner 2D のドキュメントを参照してください。

スプライトの使用

スプライトは、すべての 2D ゲームにおける重要な要素です。2D テンプレートを使用したプロジェクトでは、ビジュアルアセットがデフォルトでスプライトとしてインポートされます。これらのアセットにはユーザーのアートが含まれ、シーンに追加して Unity の 2D ツールセットと一緒に使用できるようになっています。

スプライトアセット

スプライトは、平らな 3D メッシュにマップされた 2D テクスチャです。スプライトアセットで利用できるスプライトモードには以下があります。

- **Single:** これはデフォルトのモードで、画像には単一の画像要素のみが含まれます。
- **Multiple:** テクスチャソースファイルに同じ画像内の複数の要素が含まれている場合に選択します。そして、画像をさまざまなサブアセットに分割する方法を Unityが認識できるように、スプライトエディターで要素の位置を定義します。スライスされた各グラフィックスは、UI Toolkit で別々に使用できる個々のスプライトになります。PSD インポーターを使用してインポートされた PSD アセットは、デフォルトで Photoshop ファイルの各レイヤーにスプライトを表示します。
    - タイル状にすることができる画像や 9 スライススプライトのようにスライスする必要がある画像は、タイプが Multiple である必要があります。
- **Polygon**: 円形や正多角形の画像に最適なこのモードは、画像の形状にぴったり合ったアウトラインを設定し、アウトラインをより鮮明にするために役立ちます。例えば、これはパーティクルの効果に役立ち、オーバードローを減らすことができます。

さまざまなスプライトモードを使用したアセットの例

二次テクスチャとして使用されることを意図したスプライト (詳細は、二次テクスチャを使用した作業を対象とするライティングに関するセクションを参照) は、**Normal map** (法線マップ)、**Default** (デフォルト) または**Single Channel** (マスクマップ) としてインポートする必要があります。

Inspector には、以下のようなその他のスプライト固有の設定があります。

— **Pixels Per Unit:** グリッドユニットあたりで想定されるピクセル密度を示します (アセットの解像度の設定に関する前述のセクションに記載)。

— **Mesh Type:** スプライトを **Full Rect** を使用してクアッドの 3D メッシュにするか、**Tight** を使用してテクスチャの不透明なピクセルを包むタイトな 3D メッシュにするかを示します。スプライトエディターでは、メッシュはアウトラインと呼ばれ、ここで調整できます。

— **Extrude Edges**: テクスチャの不透明なピクセルとメッシュの間にマージンを追加するオプション。

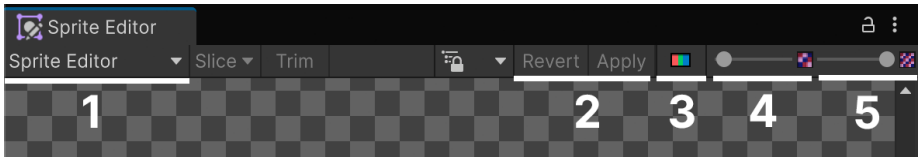
— **Pivot**: スプライトをスケール、移動、回転するためのメッシュ内の参照位置を示します。値は正規化されています。0 は下または左、1 は上または右、0.5 は中央です。Sprite Editor ウィンドウからも調整できます。

— **Generate Physics Shape**: 特にスプライトエディターで定義されている他の形状がない場合、物理演算にスプライトのアウトラインを使用します。

スプライトエディター

スプライトには、Unity の 2D ツールセット全体で共有される多くのプロパティがあります。シンプルなスプライトは、Unity へのインポートするとすぐに使用できます。その他のスプライトは、使用ケースに基づいて追加の調整が必要です。スプライトエディターには、スプライトアセットを選択した場合は Inspector ウィンドウから、または **Windows > 2D > Sprite Editor** を介して上部のメニューからアクセスできます。

スプライトエディターのメニューオプションについて簡単に紹介します。



1. Sprite Editor のメニューをクリックすると、スプライト編集のための追加オプションを示すドロップダウンリストにアクセスできます。
2. 変更を加えた際にスプライトへの変更を適用するか元に戻す必要がある場合は、これらのオプションのいずれかを選択してください。
3. 色とアルファのプレビュー表示モードを切り替えます。
4. ズームインおよびズームアウト (マウスを使用して拡大縮小することも可能)
5. このオプションを有効にすると、スプライトのさまざまなミップマップレベルのプレビューが得られます。

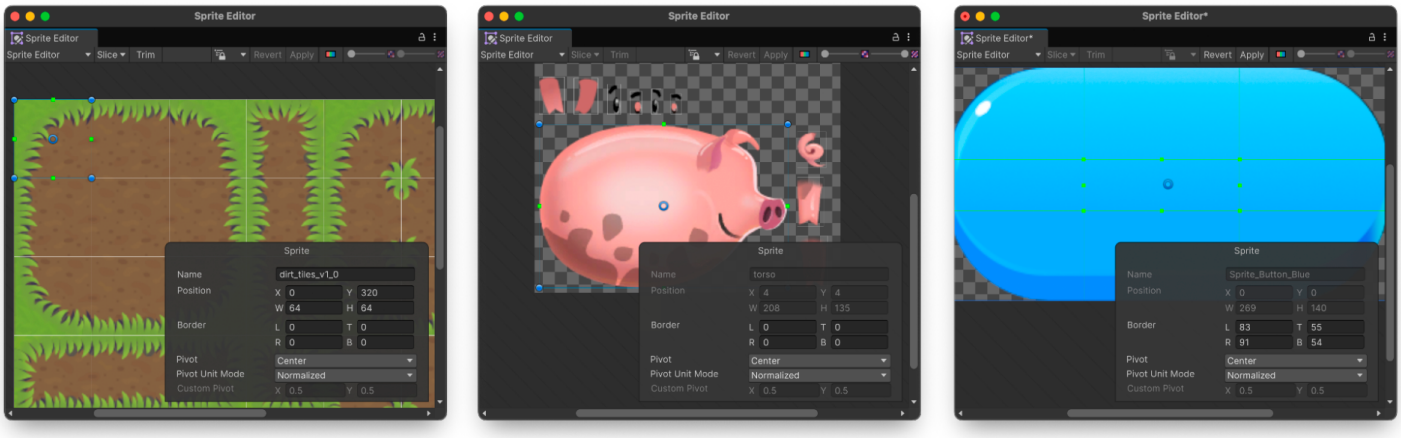
これらのオプションを詳しく見ていきましょう。

スプライトエディター

このメニューには、スライス、境界線の調整、および各スプライトのピボットを行うためのツールがあります。スプライトアセットで **Multiple** モードが選択されている場合、スライスオプションが利用可能になります。例えば、タイルシートをインポートする場合です。

マルチパーツキャラクターのパーツは、すべて同じシートにある場合は手動でスライスできますが、PSD インポーターを使用した推奨ワークフローでは、Photoshop ファイルの各レイヤーに対してスプライトを生成するため、スライスは必要ありません。

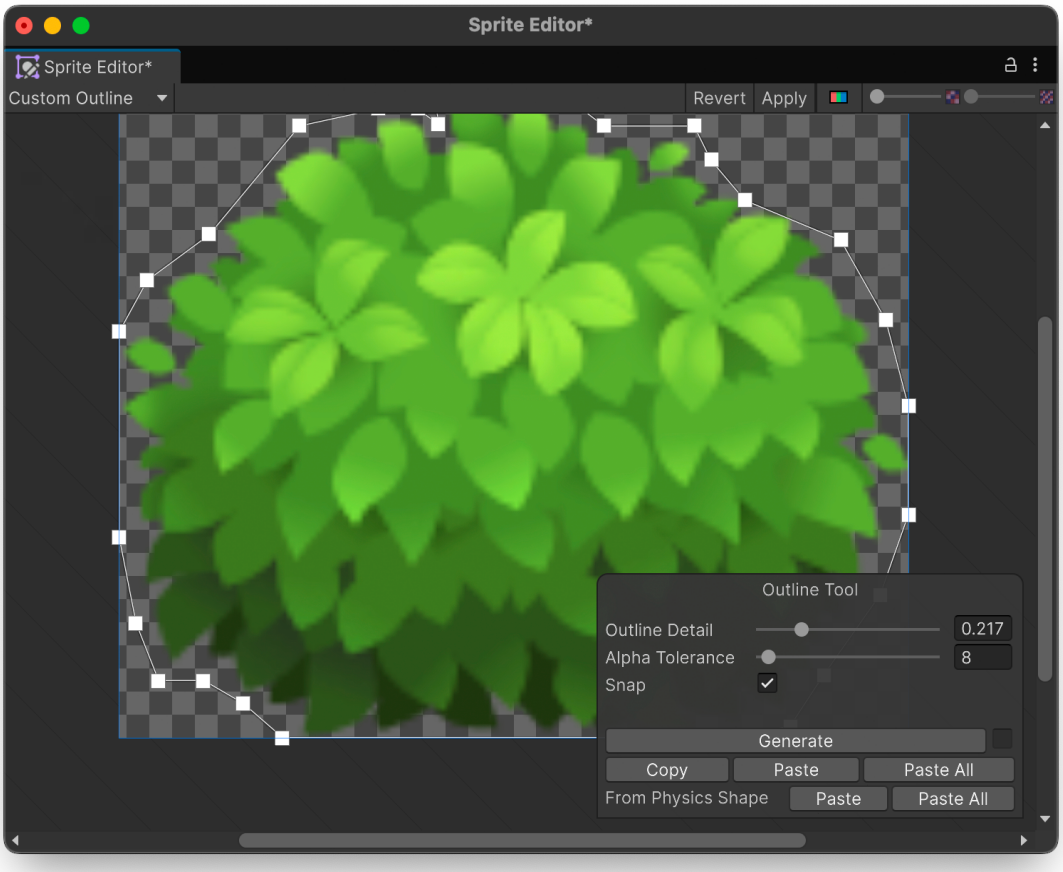
最後に、サイズが変更されても境界線を保持する必要があるスプライト (プラットフォームスプライトや UI ボタンなど) の場合は、9 スライス 機能を使用して、境界線と内部の繰り返し可能な領域を緑のハンドルで定義できます。ピボットハンドルを移動することで、各スプライトのピボットを調整できます。



左から順に、スライスされたタイルシート、アニメーション用のマルチパーツキャラクター、9 スライススプライト

カスタムアウトライン

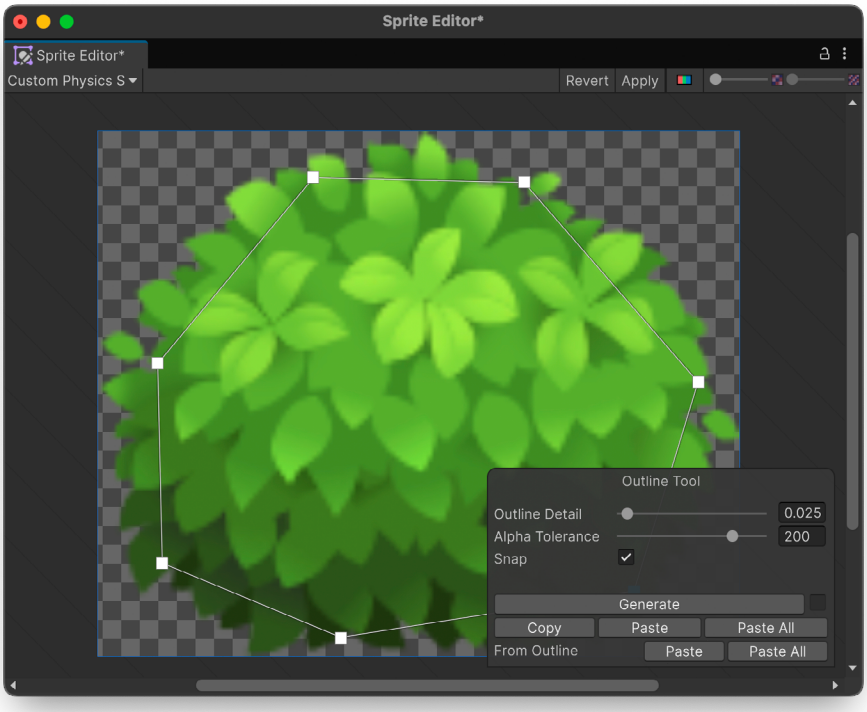
Custom Outline (カスタムアウトライン) オプションを使用して、スプライトの不透明なピクセルに合わせて調整されたタイトなメッシュを作成し、パフォーマンスを向上させます (アウトラインの外側のピクセルは無視され、レンダリングされません)。メッシュまたはアウトライン生成を微調整するには、パスの点を手動で移動して、複雑過ぎるアウトライン形状を作成せずに形状により適合させることができます。



Custom Outline オプションで生成および調整されたアウトライン

カスタム物理形状

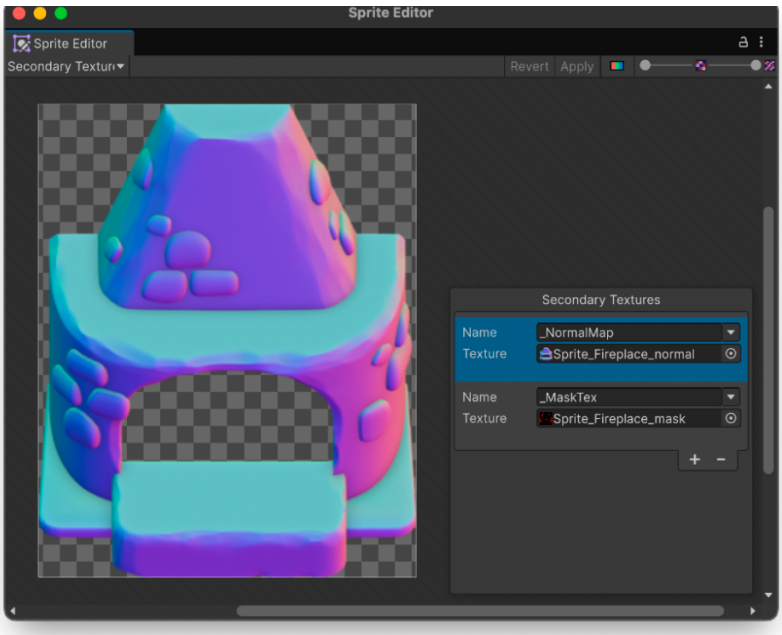
カスタムアウトライン機能と同じ生成オプションを共有していますが、Custom Physics Shape (カスタム物理シェイプ) モジュールの使用ケースはさまざまです。この機能を使用すると、スプライトの視覚的シルエットに厳密には従わない別の物理オブジェクトと衝突するスプライトの領域を定義できます。複雑な物理形状はパフォーマンスコストがかかるため、できるだけシンプルに保つように留意してください。

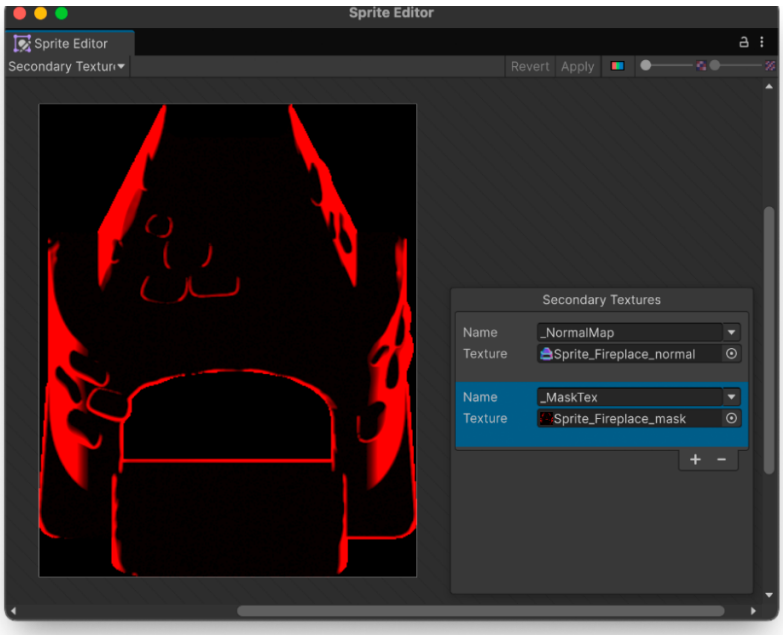


UI ビルダーで作成されたカスタム物理形状

二次テクスチャ

Secondary Textures (二次テクスチャ) モジュールを使用すると、2D ライトシステムまたは Shader Graph で使用される法線マップまたはマスクマップアセットを選択できます。二次テクスチャを追加することは必須ではありませんが、2D ライトと組み合わせることでゲームの外観に視覚的な向上をもたらすことができます。後述の 2D ライトとシャドウの章で二次テクスチャについて詳細を確認してください。

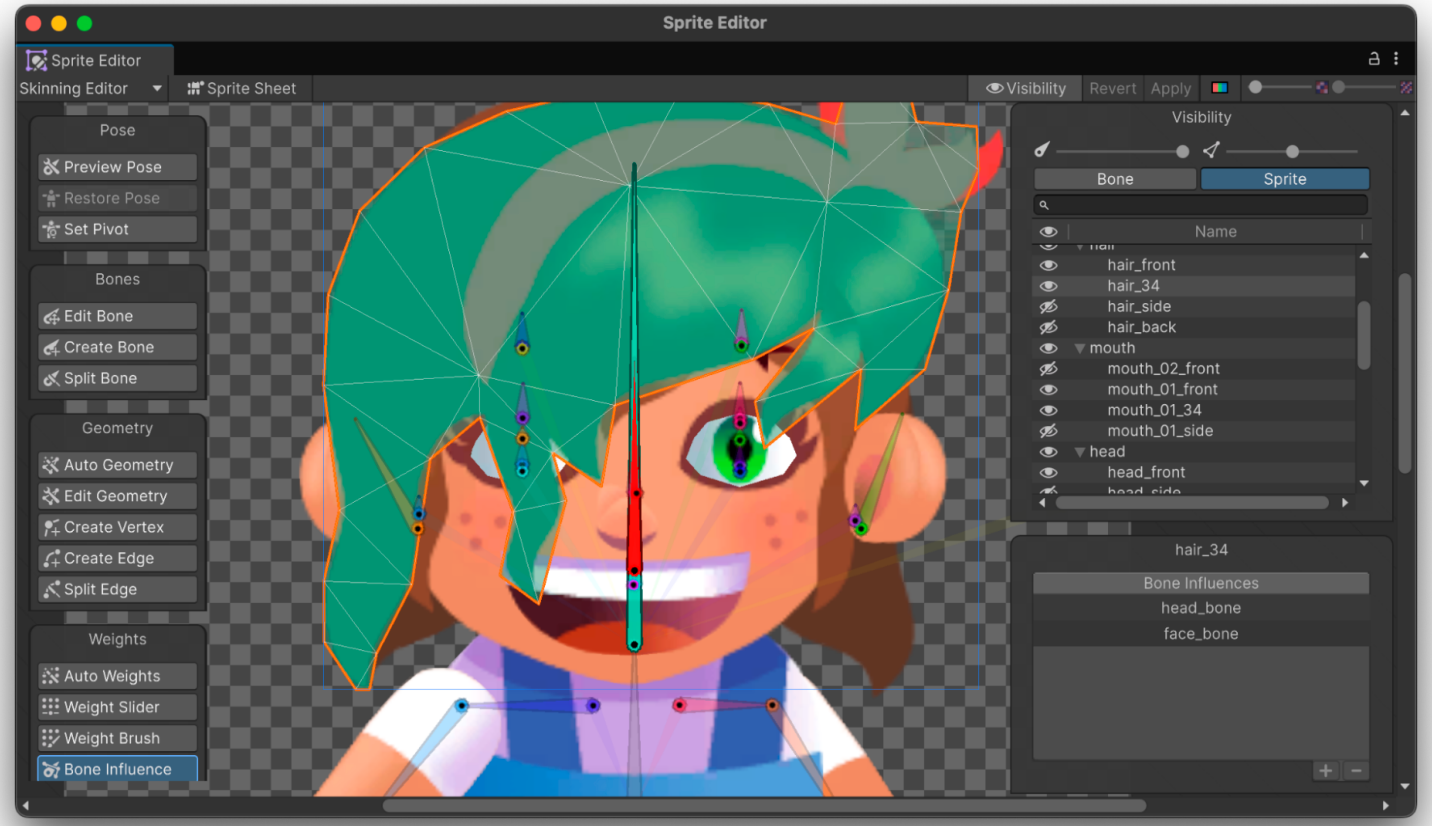




スプライトに追加された法線マップとマスクマップで興味深いライティング効果を作成します。

Skinning エディター

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用意されています。このセクションでは、各スプライトのジオメトリを作成し、それにボーンを関連付け、スプライトジオメトリの頂点ごとにボーンの影響 (ウェイト) を調整できます。設定が完了すると、Unity のアニメーションシステムを使用して、スプライトが曲がるアニメーションを作成できます。これは 3D キャラクターのスキンメッシュでの動作に似ています。詳細については 2D アニメーションの章で確認できます。

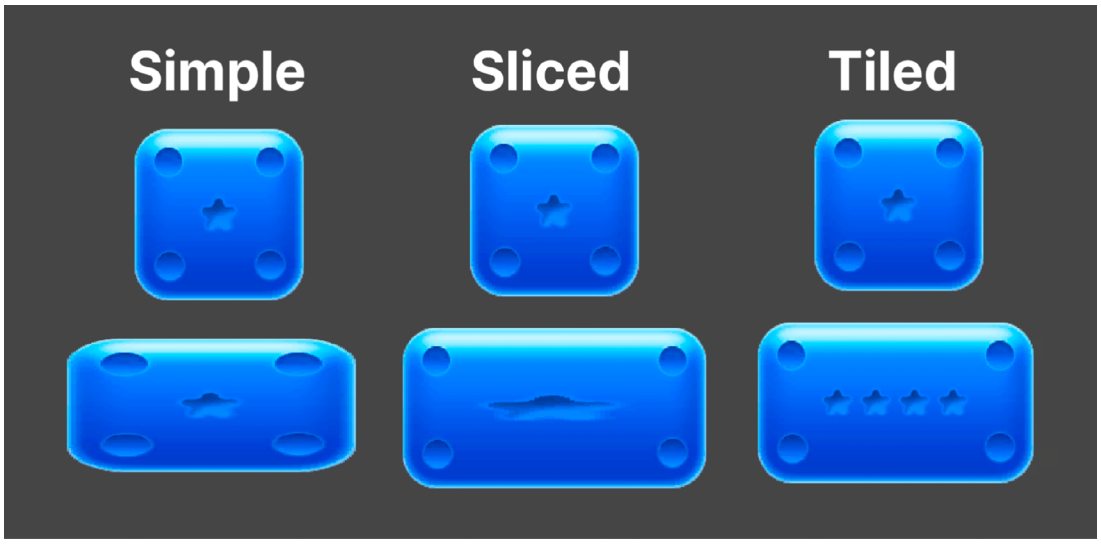


複数のパーツがあり、スプライトエディターでリグ付けされ、アニメーションの準備が整っているキャラクター

Sprite Renderer コンポーネント

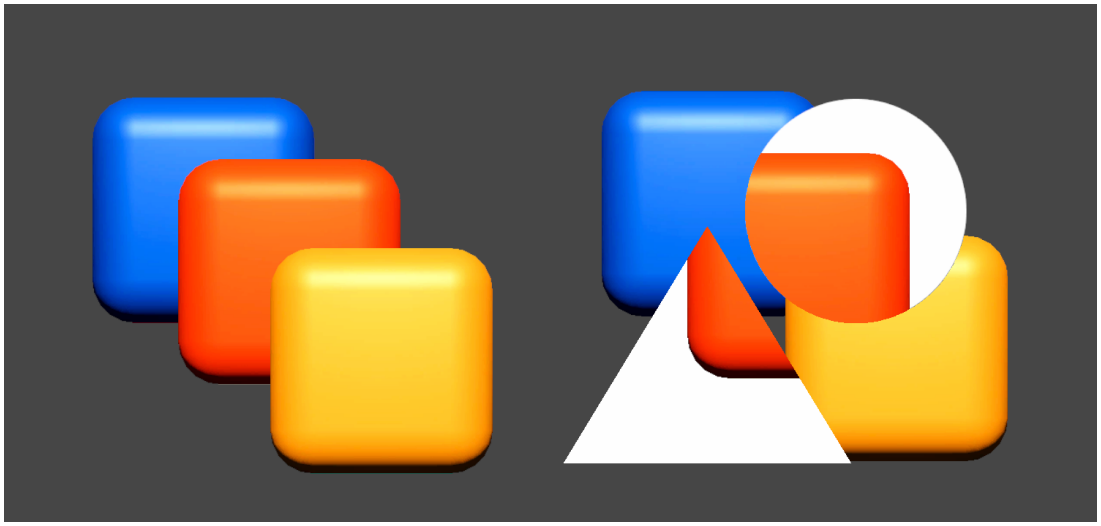
スプライトアセットを使用する準備が整ったら、アセットをシーンまたは階層ビューにドラッグするだけで使用を開始できます。通常の Transform コンポーネントとスプライトレンダラーを含む新しいゲームオブジェクトが作成されます。スプライトレンダラーで利用可能なオプションは以下の通りです。

- **Color:** スプライトに色を付けてゲームオブジェクトの簡単なバリエーションを作成したり、ゲームプレイ時に効果を追加したりできます。API を使用してコードから調整することができます。
- **Flip:** スプライトの向きを反転させて、スプライトを他の方向に素早く向けることができます。
- **Draw Mode**: スプライトの寸法が変更されたときのスケールの方法を定義します。Sliced、Tiled、Simple などのオプションから選択します。



同じスプライトをさまざまな描画モードでスケール

— **Mask Interaction:** スプライトを隠すことも、スプライトで隠すこともできます。マスクとして機能する他のスプライトの内側または外側にのみスプライトを表示したい場合は、Mask Interaction オプションを使用します。スプライトが スプライトマスク として機能するようにするには、Sprite Mask コンポーネントを追加する必要があります。これらのオプションの使用については、次のセクションを参照してください。



マスクインタラクションの使用前/使用後の 3 つのスプライト: 三角形と円のスプライトがスプライトマスクとして機能し、オレンジのキューブはマスクの内側でのみ表示され、黄色のキューブはマスクの外側でのみ表示されます。

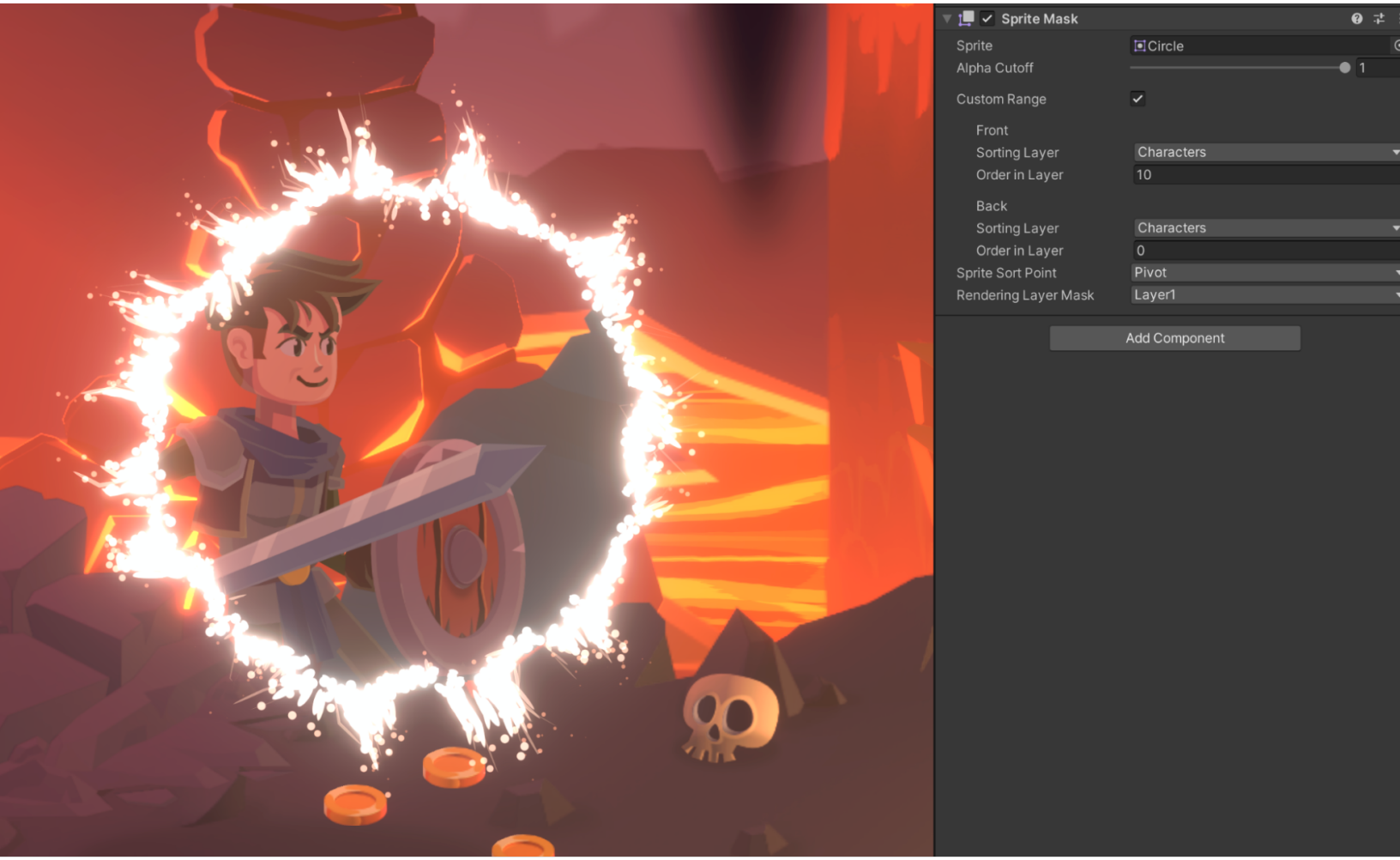
— **Sprite Sort Point**: スプライトとカメラ間の距離を計算する際に、スプライトの中心またはピボットポイントのいずれかを選択してください。

— **Material**: 新しく作成したスプライトのマテリアルを選択してください。デフォルトのマテリアルは **Sprites - Default** で、ソートシステムやライトシステムなどの 2D ツールと互換性があります。円形のアイコンをクリックすると、オブジェクトピッカーウィンドウが開きます。2D ライトを使用しない場合は **Sprite - Unlit-Material** などの他のマテリアルを選択できます。スプライトレンダラーと互換性のある他のシェーダーは、Shader Graph を使用して作成できます。

スプライトマスク

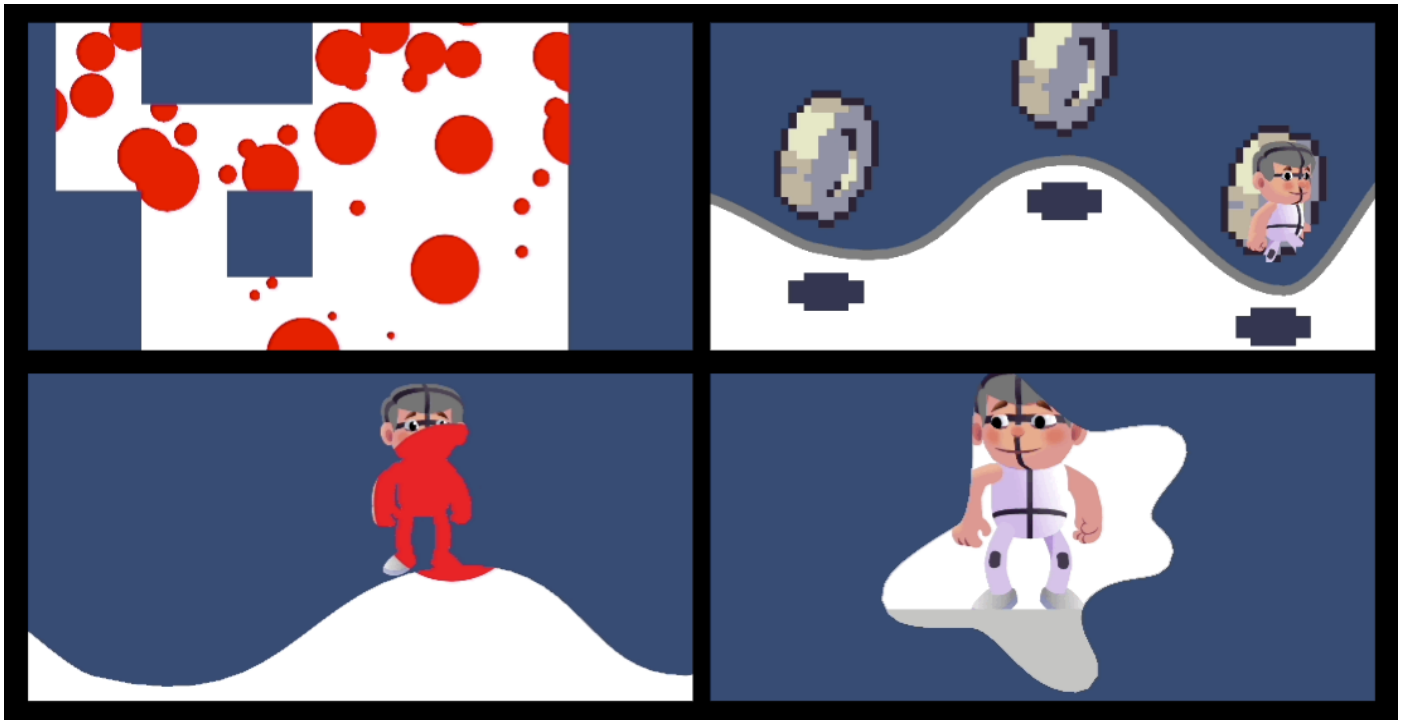
スプライトマスクを使用すると、スプライトマスクによる興味深い効果を生み出すことができます。例えば、画像の一部を隠してポータル効果を生み出したり、3D 効果のあるコレクションカードを作成したりできます。

マスク画像として使用されるスプライトは、アニメーションツールでアニメーション化することもできます。これにより、いくつかの興味深い効果を作成できます。例えば、ポータルを表示させ、その後大きくしたり形を変えたりすることができます。



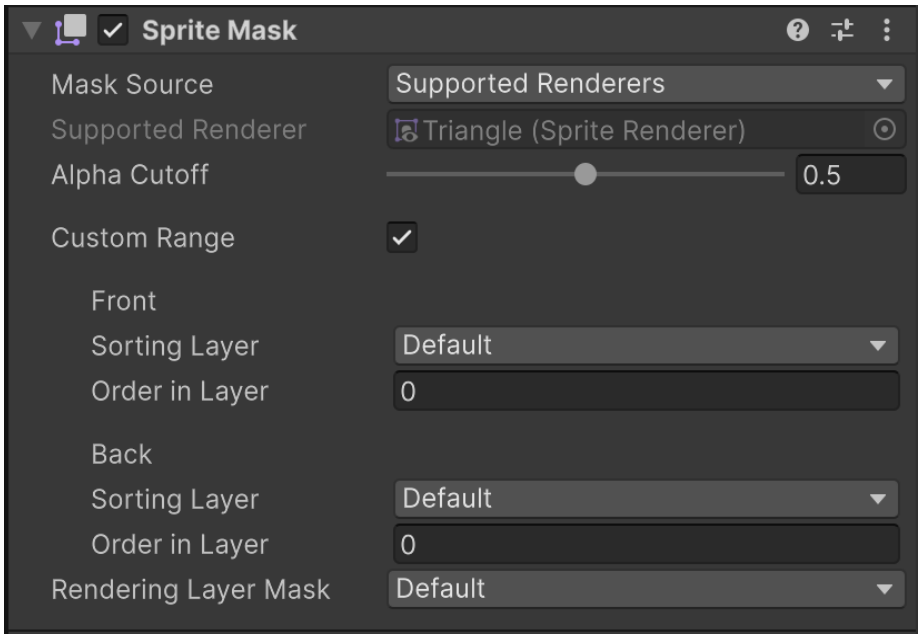
スプライトマスクとビルトインパーティクルシステムを使用して作成されたポータル効果

以前のバージョンの Unity では、スプライトマスクは、マスクとして使用される特定のスプライトによって定義されていました。これにより多くの効果が可能になりますが、Unity 6 ではマスクシステムが大幅に改善されました。



Unite Barcelona 2024 の2D workflows で示された例

Sprite Mask コンポーネントは、任意の 2D レンダラーを Mask Source として使用できます。これには、スプライト、タイルマップ、スプライトシェイプ、さらにはスケルトンにリグ付けされたキャラクターが含まれます。どのような 2D オブジェクトもマスクにすることができるため、アニメーション化されていても新たな創造的可能性が開かれます。



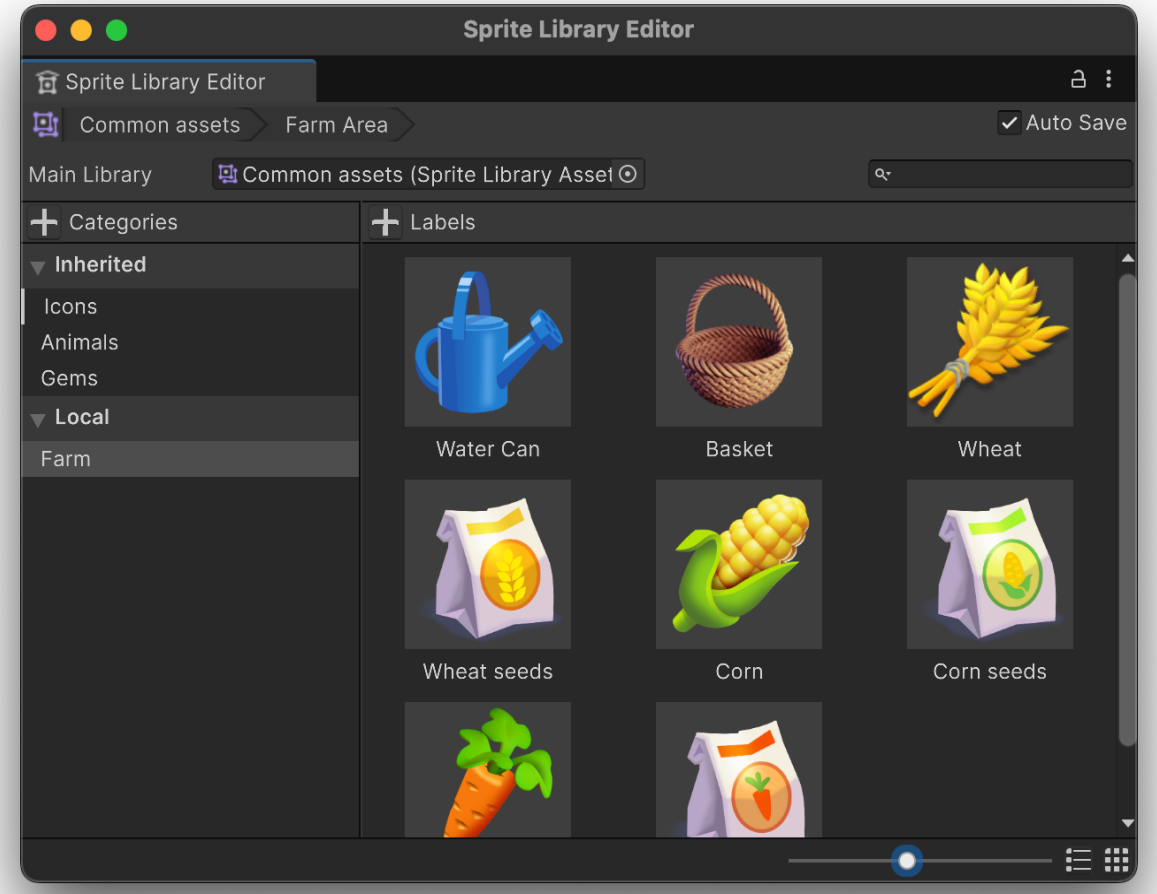
さらにコントロールを強化するために、レイヤーのソートやレイヤーの順序でマスクの影響を受けるスプライトを選択できます。

スプライトライブラリアセット

スプライトライブラリアセット を使用して、カテゴリごとにグループ化されたスプライトのコレクションを構築することができます。スプライトライブラリにより、カテゴリに基づいてエディターやランタイムでスプライトをスワップするための簡単で視覚的な方法が提供されます。これは、プロジェクトのスプライト数が膨大になる場合に非常に便利です (多数の手描きのアニメーションを含む 2D インディーゲームのスプライト数は数万に達することがあります)。

スプライトの種類を以下のカテゴリでグループ化できます。

- **ゲームプレイの目的:** 例えば、武器のカテゴリには、剣、斧、またはメイスとラベル付けされたスプライトが含まれる場合があります。
- **アニメーション**: キャラクターの左手のカテゴリには、さまざまなポーズで描かれた手や、帽子のカテゴリのようにさまざまな視覚的アクセサリーを持つ手が含まれる場合があります。
- **レベルデザイン**: このカテゴリには、背景の建物 (アパート、超高層ビル、オフィスなど) などの要素が含まれる場合があります。



Sprite Library Editor ウィンドウ

Sprite Resolver コンポーネント

スプライトライブラリとその関連コンポーネントである Sprite Resolver の主な用途の 1 つは、マルチパーツキャラクターのアニメーションを設定し、スプライトをスワップすることです。

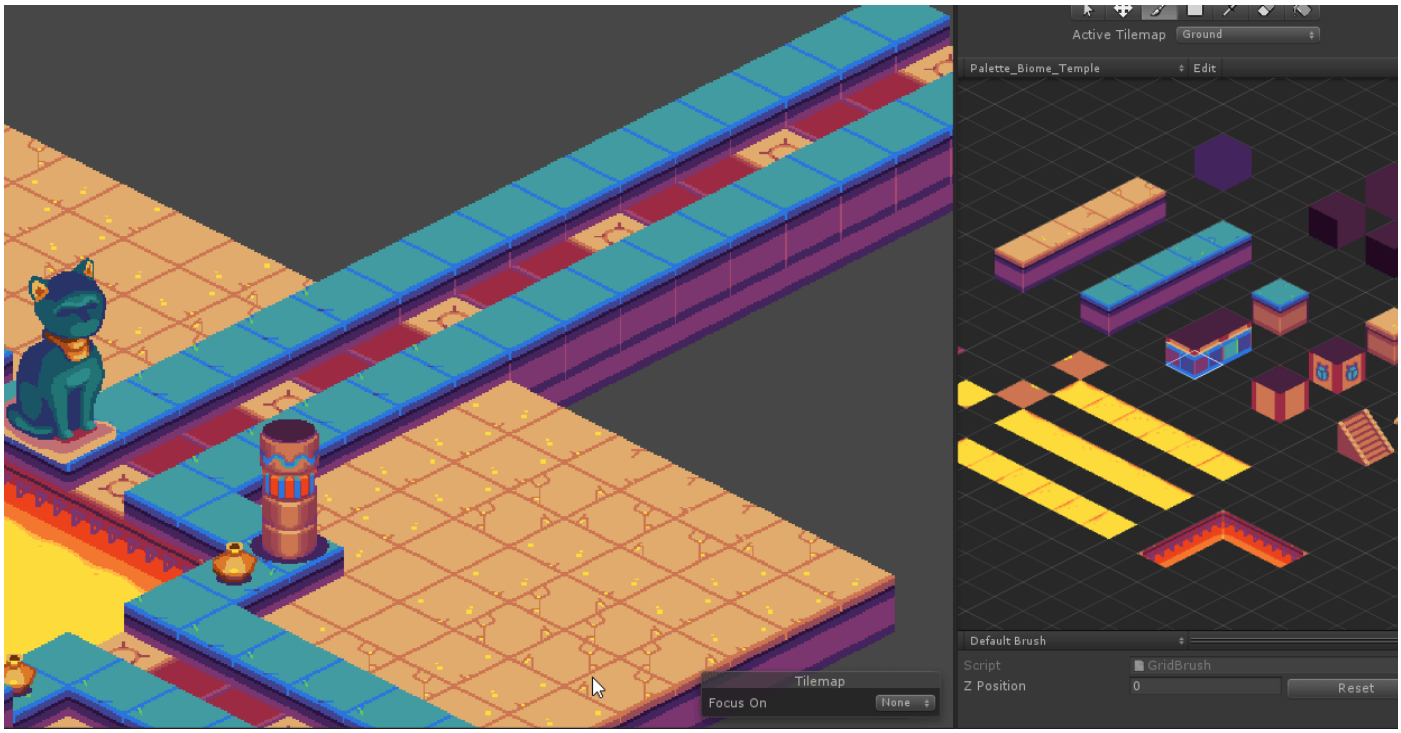
Unity 6 では、シーンビューに新しい**Sprite Swap オーバーレイ**が用意されています。このオーバーレイには、特定のカテゴリの選択されたスプライトと切り替え可能な、すべての関連ラベルが表示されます。

オーバーレイは、アニメーションやレベルデザインの段階で便利です。Project ウィンドウで検索することなく同じカテゴリのスプライトを選択できます。

スプライトスワップとスプライトライブラリアセットの詳細については、ドキュメントを確認し、開発サイクルの初期に使用を開始して、プロジェクトがスケールするにつれてライブラリを整理してください。

2D タイルマップ

タイルマップは、グリッド上に配置するタイルと呼ばれる小さなスプライトを使用してゲーム世界を作成する方法を可能にします。1 つの大きな画像がゲームの世界にレイアウトされるのではなく、レンガのような塊に分解され、レベル全体で繰り返されます。



Unity は アイソメトリック、六角形、長方形という 3 種類のタイルマップをサポートしています。

タイルマップの作成

タイルマップは、画面に表示されていないタイルを無効にすることで、メモリと CPU の消費の抑制に役立ちます。Unity では、ブラシツールを使用することで、グリッド上に効率的にタイルをペイントでき、ペイントルールを使用するようにスクリプト化することもできます。効率的なテストや編集を可能にするために、自動衝突生成機能も備わっています。

Bunny Blitz のタイルマップ

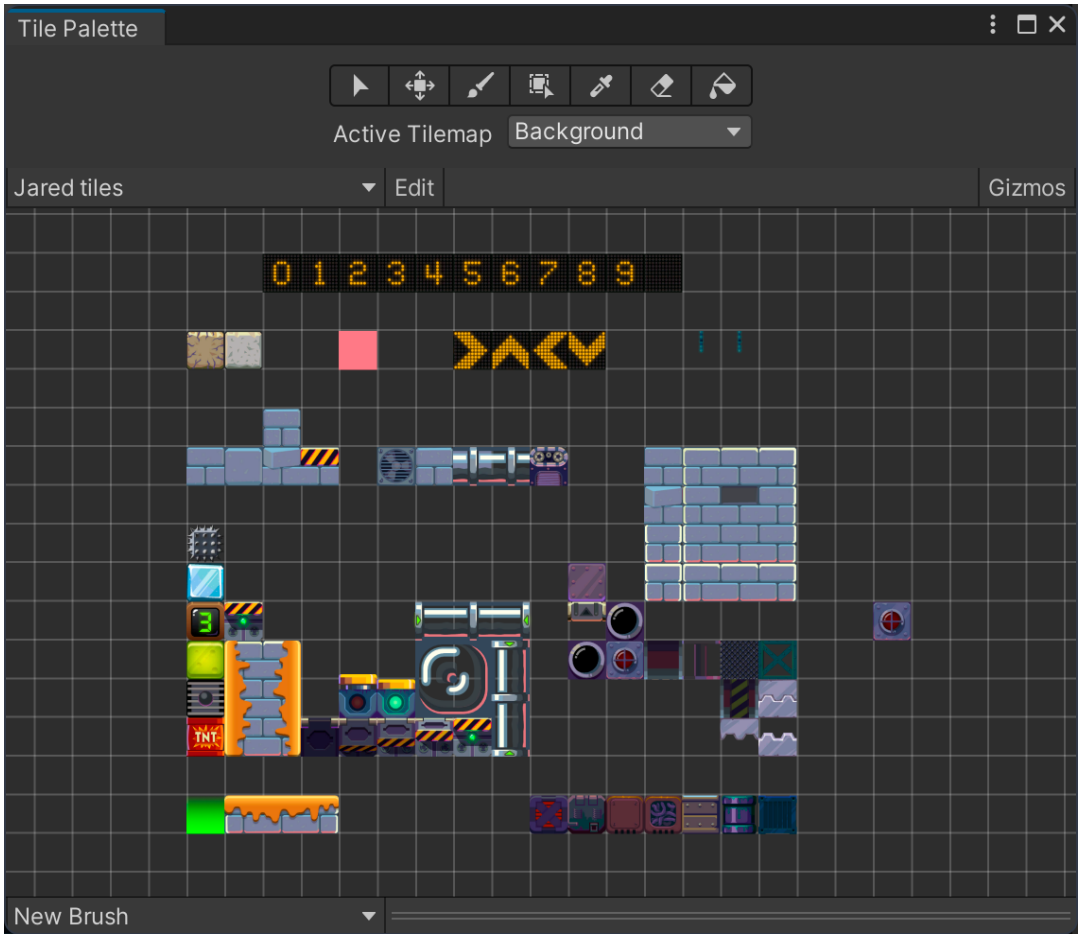
デフォルトでは、エディターには Tilemap パッケージが含まれていないため、Package Manager から 2D Tilemap パッケージをダウンロードするか、2D テンプレートからプロジェクトを開始する必要があります。

Tilemap パッケージと一緒に、2D Tilemap Extras もインストールする必要があります。これにより、独自のプロジェクトで使用できる再利用可能な 2D およびタイルマップエディタースクリプトが提供され、カスタムブラシやタイルを作成するための機能も提供されます。

パッケージに含まれるスクリプタブルブラシとスクリプタブルタイルについて、詳細を確認できます。

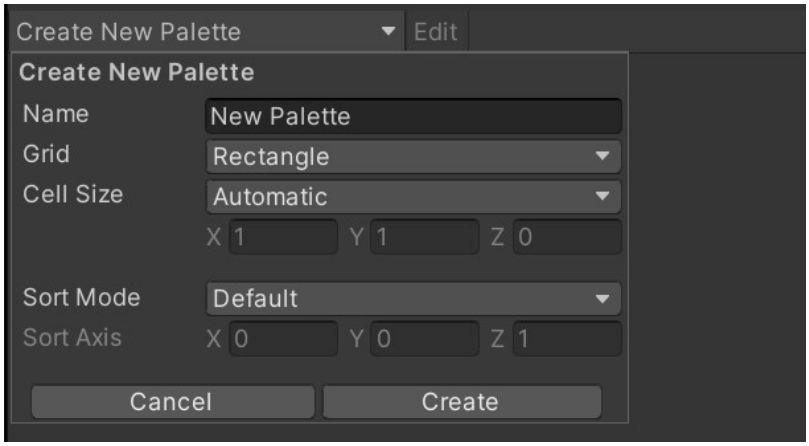
両方のパッケージをインストールしたら、**Window > 2D > Tile Palette** の順に移動して **Tile Palette** ウィンドウを開きます。

このウィンドウにより、タイルをペイントまたは編集するためのすべてのタイルとツールが提供されます。



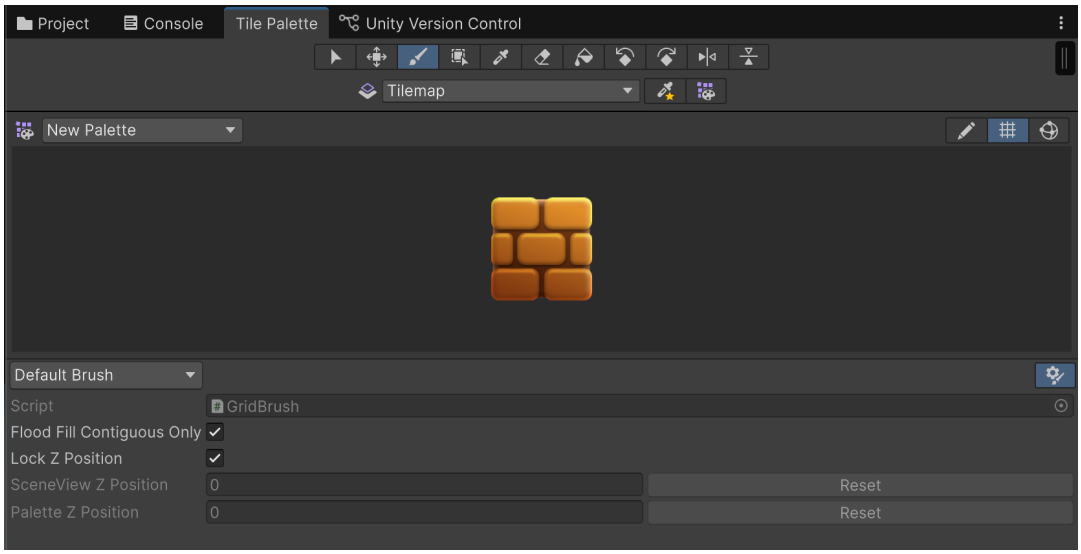
パレットがロードされた Tile Palette ウィンドウ

新規パレットを作成するには、**Create New Palette** ボタンをクリックします。オプションを示すドロップダウンウィンドウが表示されます。パレットに名前を付け、オプションを設定し、**Create** をクリックし、選択したフォルダーに保存します。



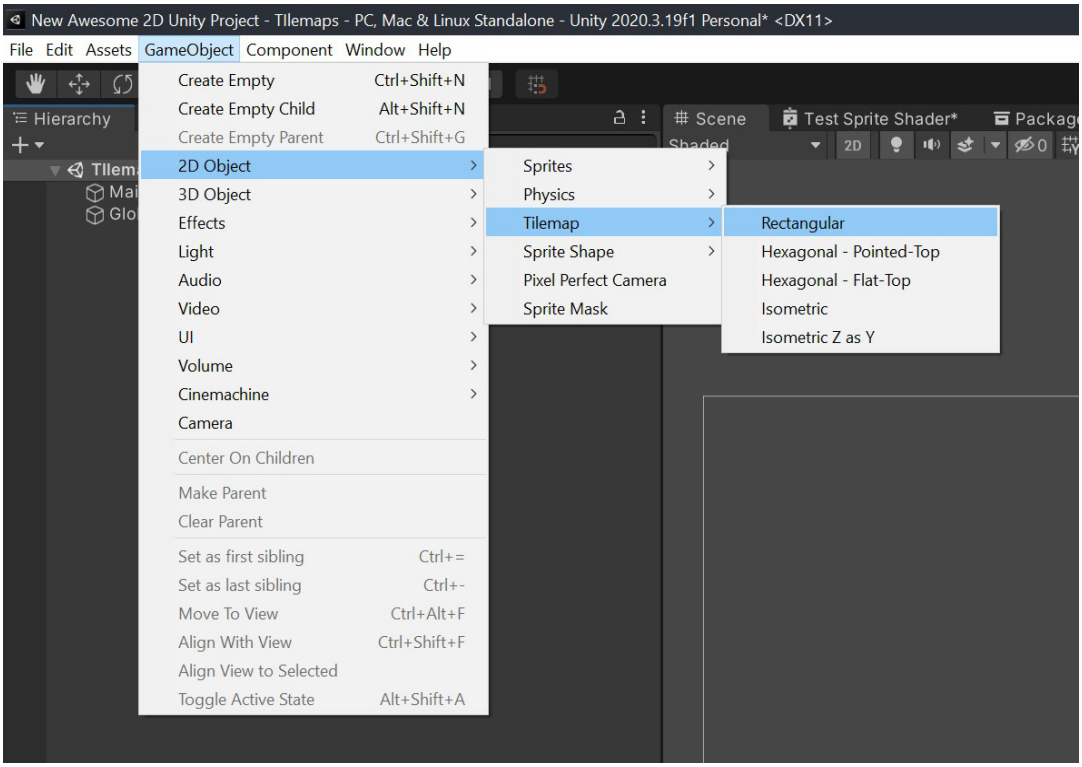
新しいパレットの作成

次に、パレットにタイルを加えます。スプライト、テクスチャ、またはタイルアセットを Palette ウィンドウにドラッグします (スプライトをドラッグした場合は、最初にタイルアセットを保存する必要があるかもしれません)。これでパレットのグリッドにスプライトが表示されます。



スプライトをパレットにドラッグすると、新しいタイルが作成されます。

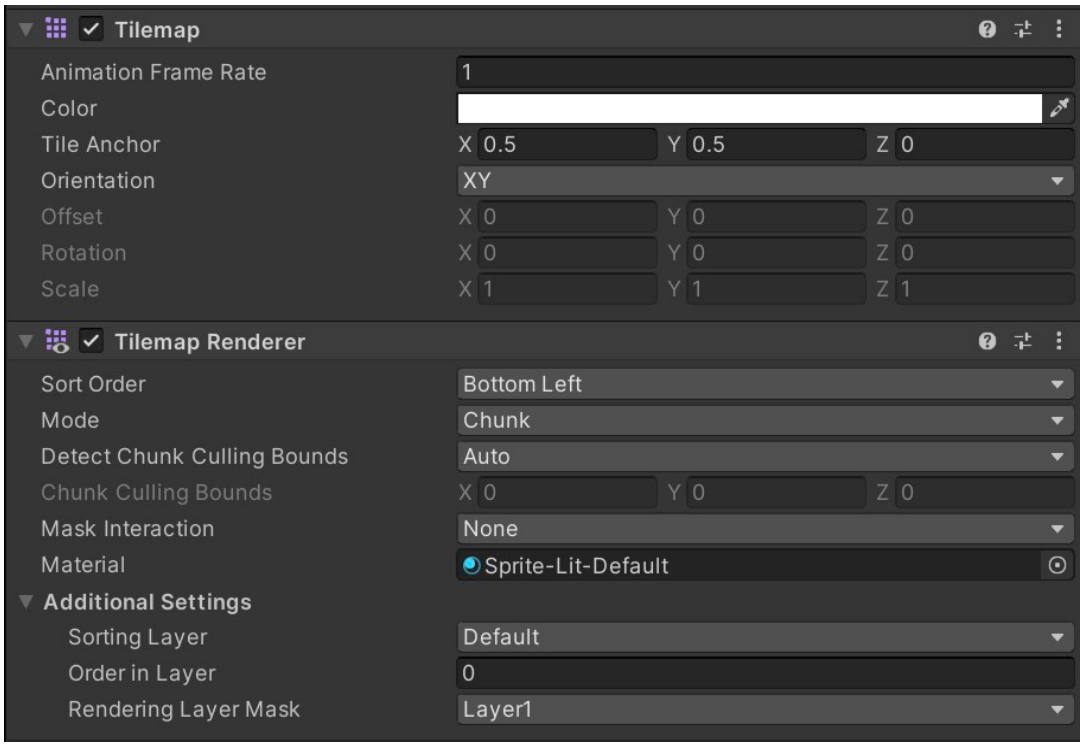
ここで、タイルをペイントするためのタイルマップを作成します。メニューアイテム **Gameobject > 2D Object > Tilemap > Rectangular** をクリックします。



タイルマップの作成

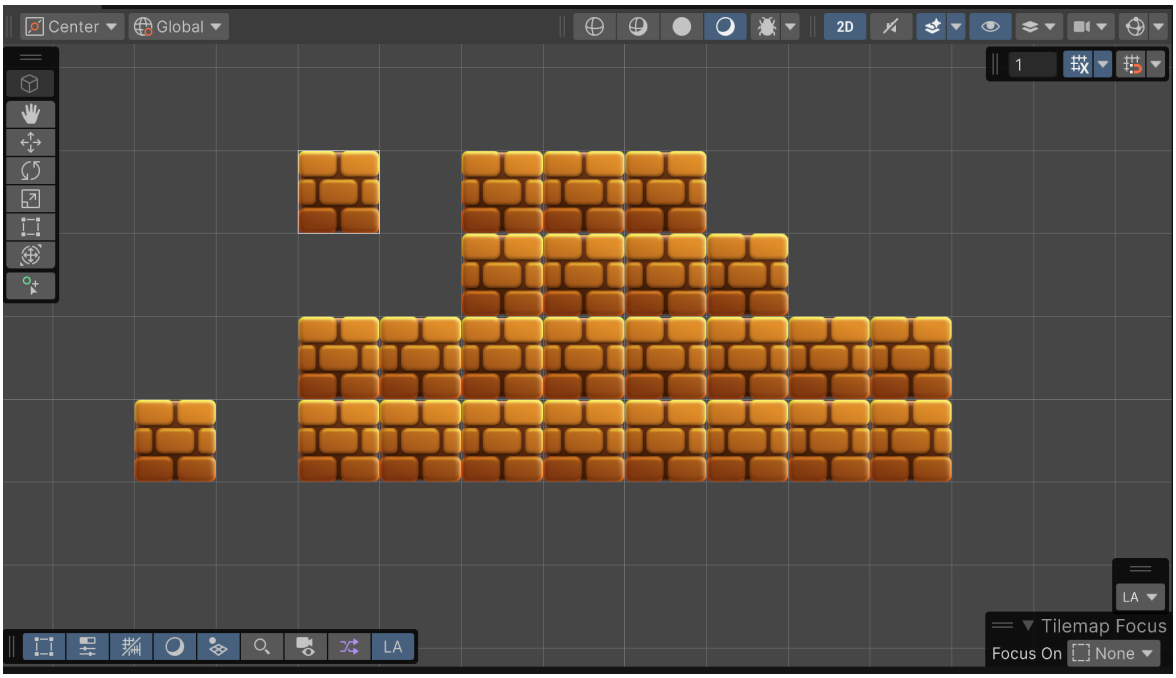
これにより、**グリッド** ゲームオブジェクトとその中にタイルマップゲームオブジェクトが作成されます。タイルマップゲームオブジェクトの名前をより説明的なものに変更します。グリッドゲームオブジェクトは複数のタイルマップを保持できます。シーンごとにセルサイズなどが異なる複数のグリッド設定することも可能です。今のところ、Grid コンポーネントのすべてのデフォルト設定をそのままにしておきます。

タイルマップゲームオブジェクトには、**Tilemap** と **Tilemap Renderer** という 2 つのコンポーネントがあります。設定はそのままにしてください。必要に応じて、ソートレイヤーの設定を変更して、レイヤー構造に合わせてください。

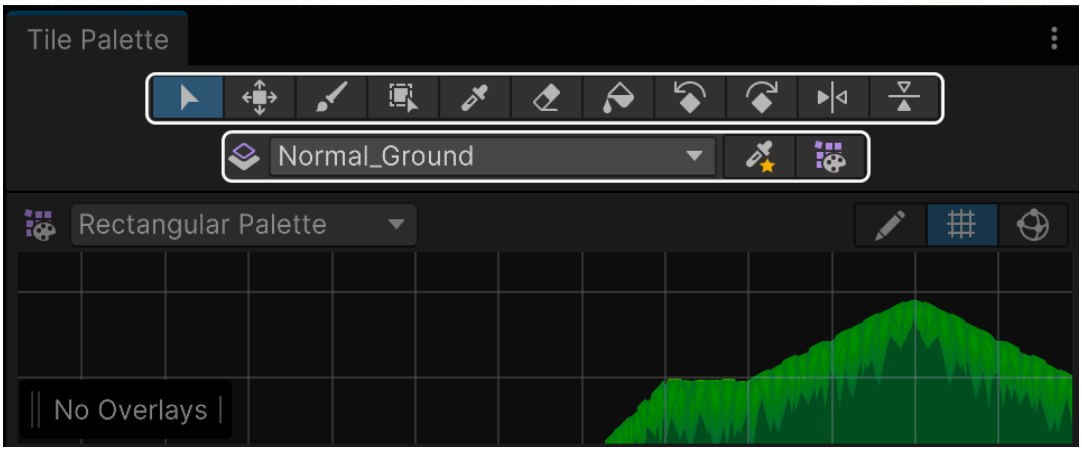


Tilemap コンポーネントと Tilemap Renderer コンポーネント

タイルマップとタイルがパレットに用意され、ペイントを開始する準備が整いました。Palette ウィンドウのツールバーでブラシツールをクリックし、ペイントするタイルを選択します。



タイルマップに描かれているシンプルなレンガタイル



タイルパレットのツール

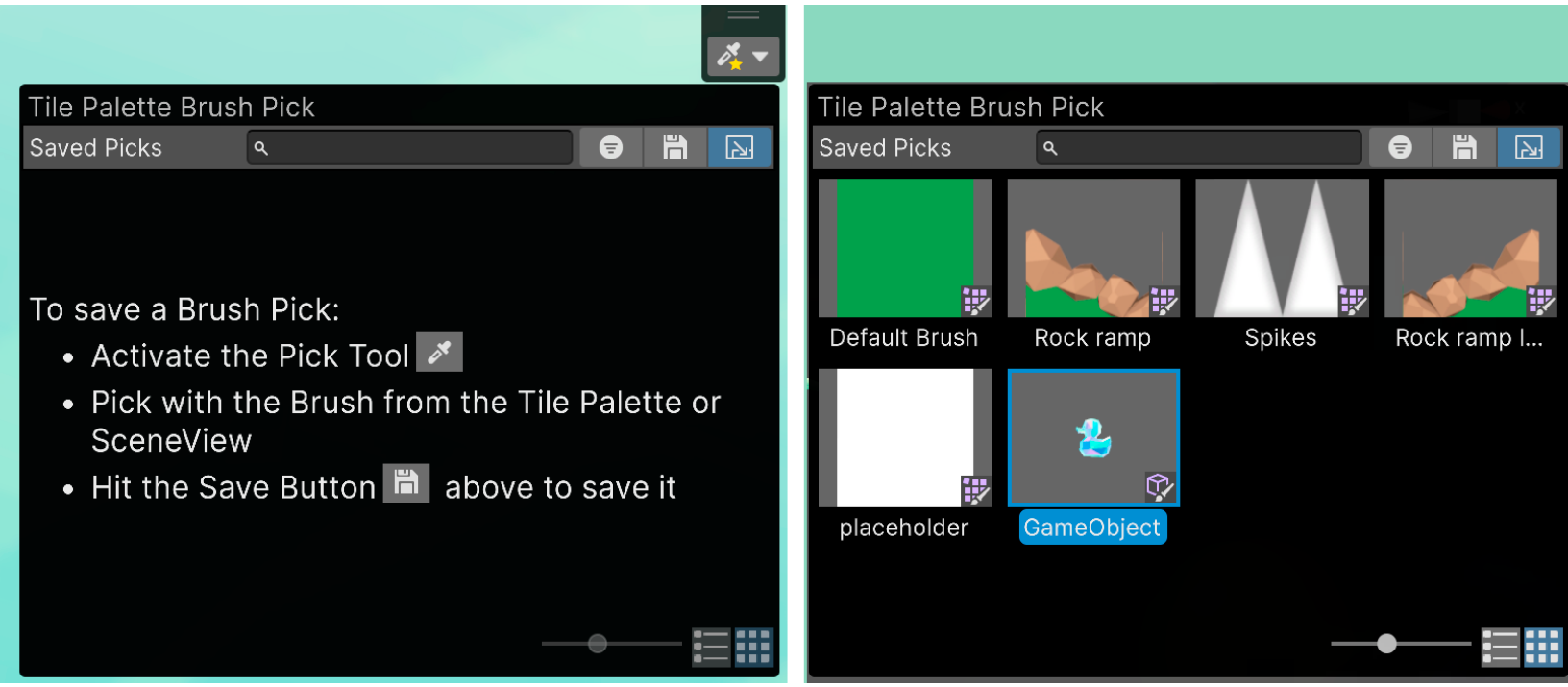
上の画像のトップバーメニューに並ぶさまざまなパレットツールを見てみましょう (キーボードのショートカットはかっこ内に記載されています)。

1. **選択 (S)**: クリックしてタイルを 1 つ選択するか、ドラッグして長方形の領域内のタイルを選択します。
2. **移動 (M)**: 選択したタイルを移動します。
3. **ブラシ (B)**: 選択したタイルとブラシを使用して、アクティブタイルマップ (アクティブタイルマップのドロップダウンから 1 つ選択) にペイントします。
4. **選択範囲の塗りつぶし (U)**: ドラッグして、選択したタイルで長方形の領域を埋めます。
5. **タイルサンプラー (I)**: タイルマップからタイルを選択し、アクティブに設定してペイントします。

6. **消しゴム (D)**: タイルマップからタイルを削除します。
7. **塗りつぶし (G)**: タイルで領域を塗りつぶします (領域は他のタイルで囲まれている必要があります。囲まれていない場合は、大きな矩形が描かれます)。
8. **左に回転 ([) と右に回転 (])**: ブラシ内のアクティブタイルを時計回りまたは反時計回りに 90 度回転させます。これは、使用頻度の少ないタイルに便利で、すべての向きのバリエーションがない場合に役立ちます。
9. **水平方向に反転 (shift + [) と垂直方向に反転 (shift +]):** 回転ツールと似た用途で、反転ツールを使用すると、タイルを反転させて、ブラシ内のアクティブタイルの簡単なバリエーションを作成することができます。

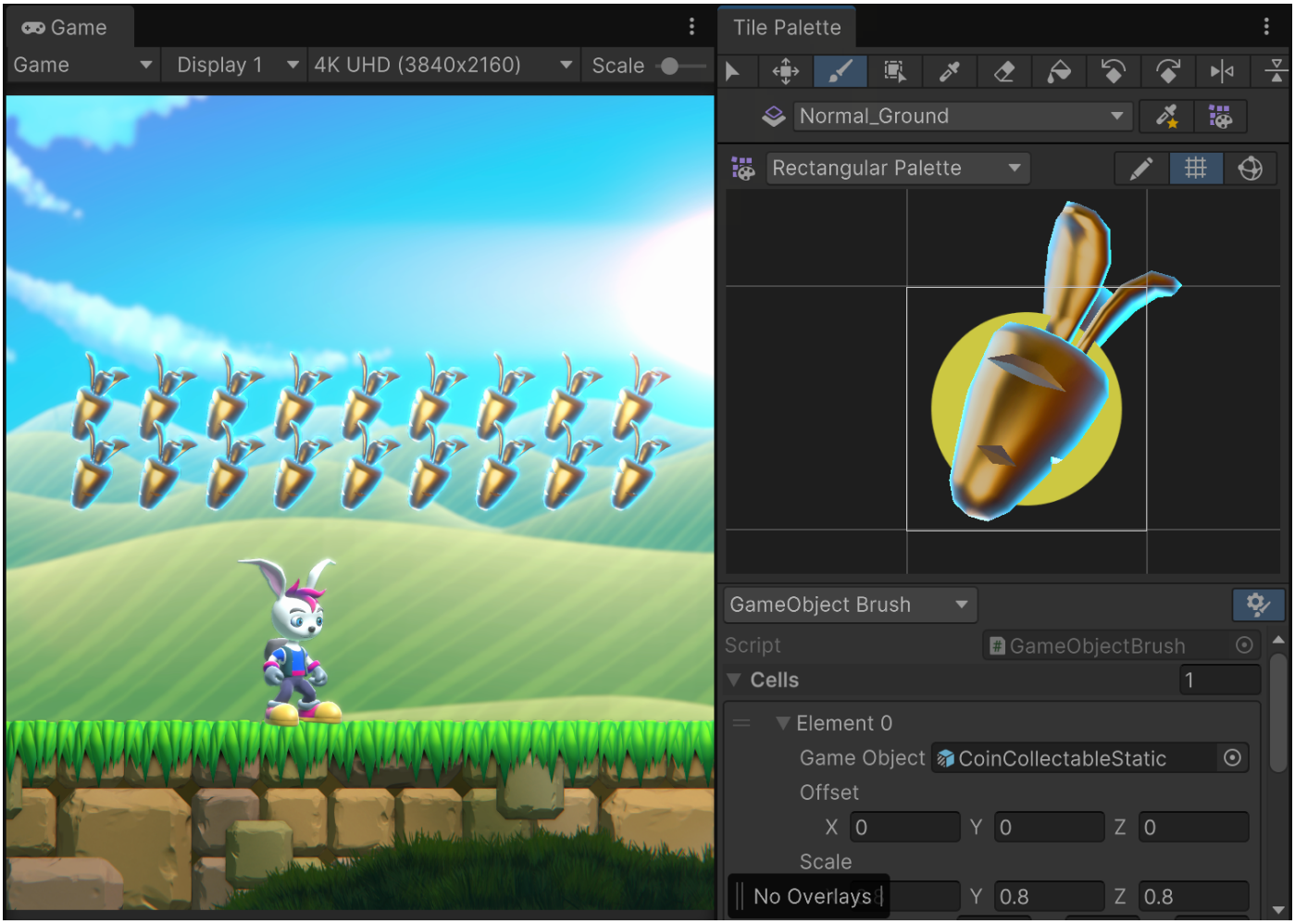
下部のバーのメニューに含まれるツールは以下の通りです。(左から)

1. **アクティブターゲット**: ペイントするタイルマップを選択します。
2. **タイルパレットブラシ選択 (‘)**: シーンビューでオーバーレイツールを有効または無効にします。これによりタイルを保存して、回転および反転したタイルやゲームオブジェクトを迅速に選択することができます。
3. **タイルパレットオーバーレイ (;)**: シーンビューでタイルパレットオーバーレイを有効または無効にします。



Tile Palette Brush Picks オーバーレイは、最も使用頻度の高いタイルを登録して迅速にアクセスするための便利な方法です。

これらのツールを使用すると、タイルを効率的にペイントおよび編集できます。さらに、タイルマップエクストラアセットにより、**Rule Tile** (ルールタイルの実際の例については以下のセクションを参照) や、タイルパレットにゲームオブジェクトを配置し、シーンにペイントするための **GameObject Brush** などの便利なスクリプトが提供されます。



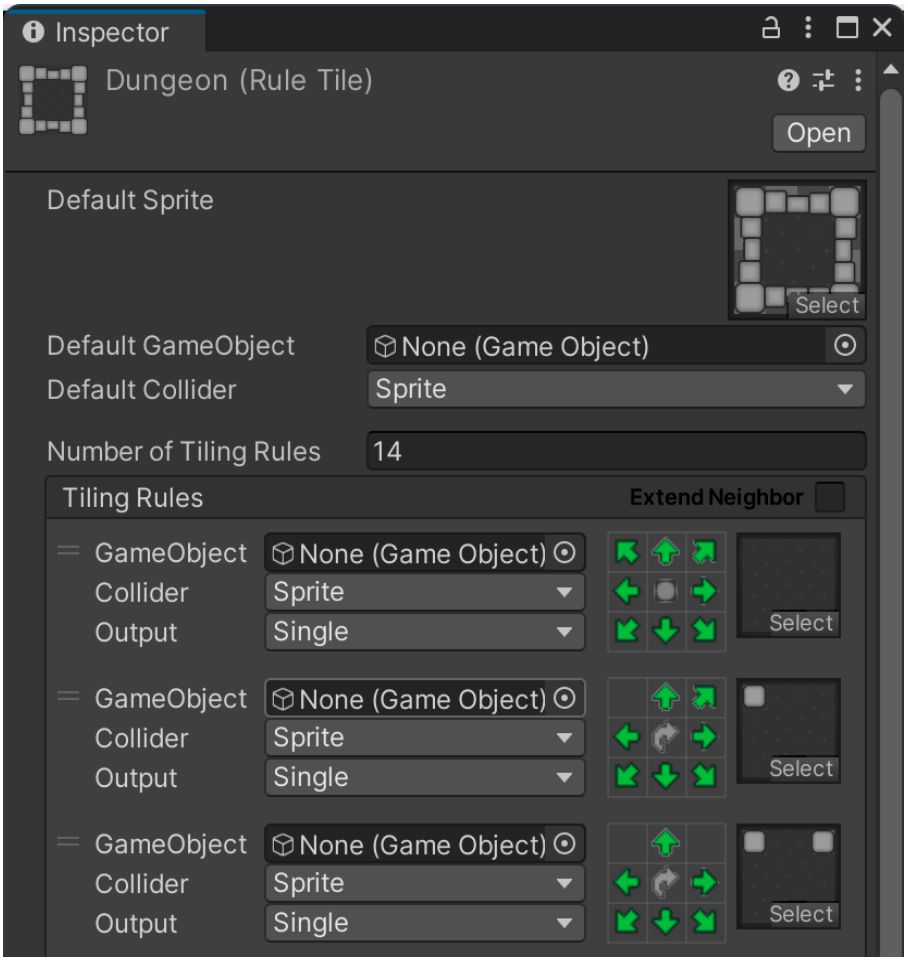
ゲームオブジェクトブラシは、タイルマップグリッドを使用してレベルに収集可能なオブジェクトを配置するために使用されます。

ルールタイル

タイルパレットツールを使用して、各タイルマップのグリッドにタイルを効率的に配置するために使用されます。ただし、パスによって形成される形状の辺や角を手動で配置すると、時間がかかり、エラーが発生しやすい場合があります。パスや他のタイルベースの形状に変更が加えられるたびに、手動作業を繰り返す必要があります。それを避けるために、Rule Tile (ルールタイル) 機能を使用して、形状の隣接タイルに基づいて正しい境界タイルをペイントする作業を行うことができます。

2D Tilemap Extras パッケージに含まれるルールタイルは、周囲を認識し、適切な画像を選択するスクリプト化されたタイルです。例えば、上に草があり、下に影がある地面タイルが挙げられます。

ルールタイルは、自動的に隣接タイルの位置と境界が必要な場所を認識します。適切なスプライトが選択されるので、境界線をペイントするために異なるタイルを選択する必要はありません。



Inspector 内の Rule Tile アセット

Rule Tile インスペクターにより、隣接タイルに基づいてどのスプライトを選択するかを決めるルールのリストが提供されます。各ルールの右側にはマトリックスとスプライトが表示されています。スプライトは、緑の矢印が指しているすべての方向に隣接タイルがある場合に使用されます。ルールタイル機能の詳細については こちら を参照してください。

Unity 6.3 における AutoTile

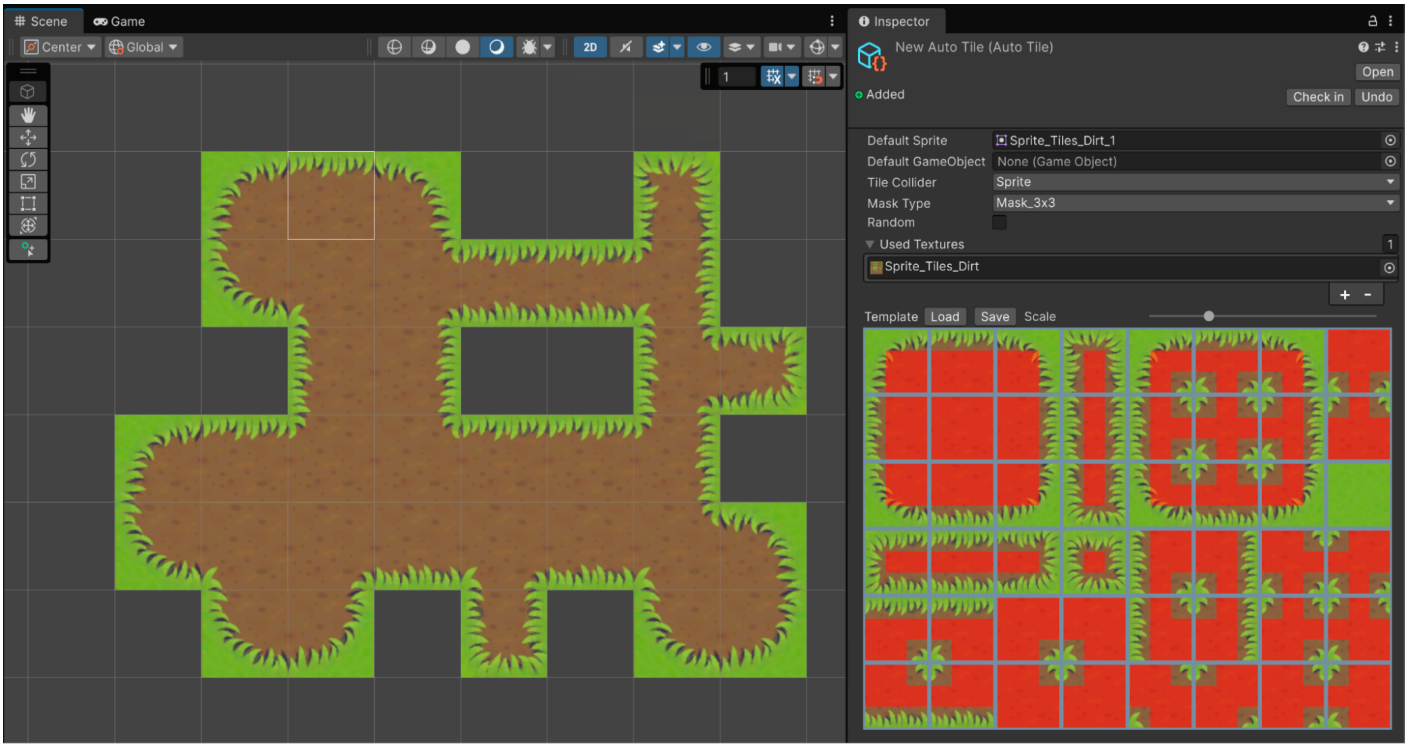
Unity 6.1 で導入された AutoTile (オートタイル) は、AutoTile アセット設定で定義したマスクに基づいて、適切な隣接タイルを表示するタイルアセットです。

ルールタイルと比較して、オートタイルはより直感的な設定が可能で、マスクを再設定することなくタイルシートのテンプレートバリエーションを作成することができます。

オートタイルを作成するステップは次の通りです。

1. プロジェクトにタイルシートをインポートします。
2. スプライトエディターでタイルシートをスライスします。以下の例では、タイルシートは 8x6 (8 列 x 6 行) です。変更を適用します。

3. Project ウィンドウで新しいオートタイルを作成します。
4. Inspector で、**Used Textures** フィールドにスライスされたタイルシートを追加します。
5. ペイントブラシでタイルの空洞部分を塗りつぶしてマスキングを開始します。タイルに別のタイルと同じマスキングがある場合はハイライトされ、マスキングが正しいことを二重に確認できます。
6. タイルマップでペイントしてオートタイルをテストします。



Happy Harvest のオートタイルの一例

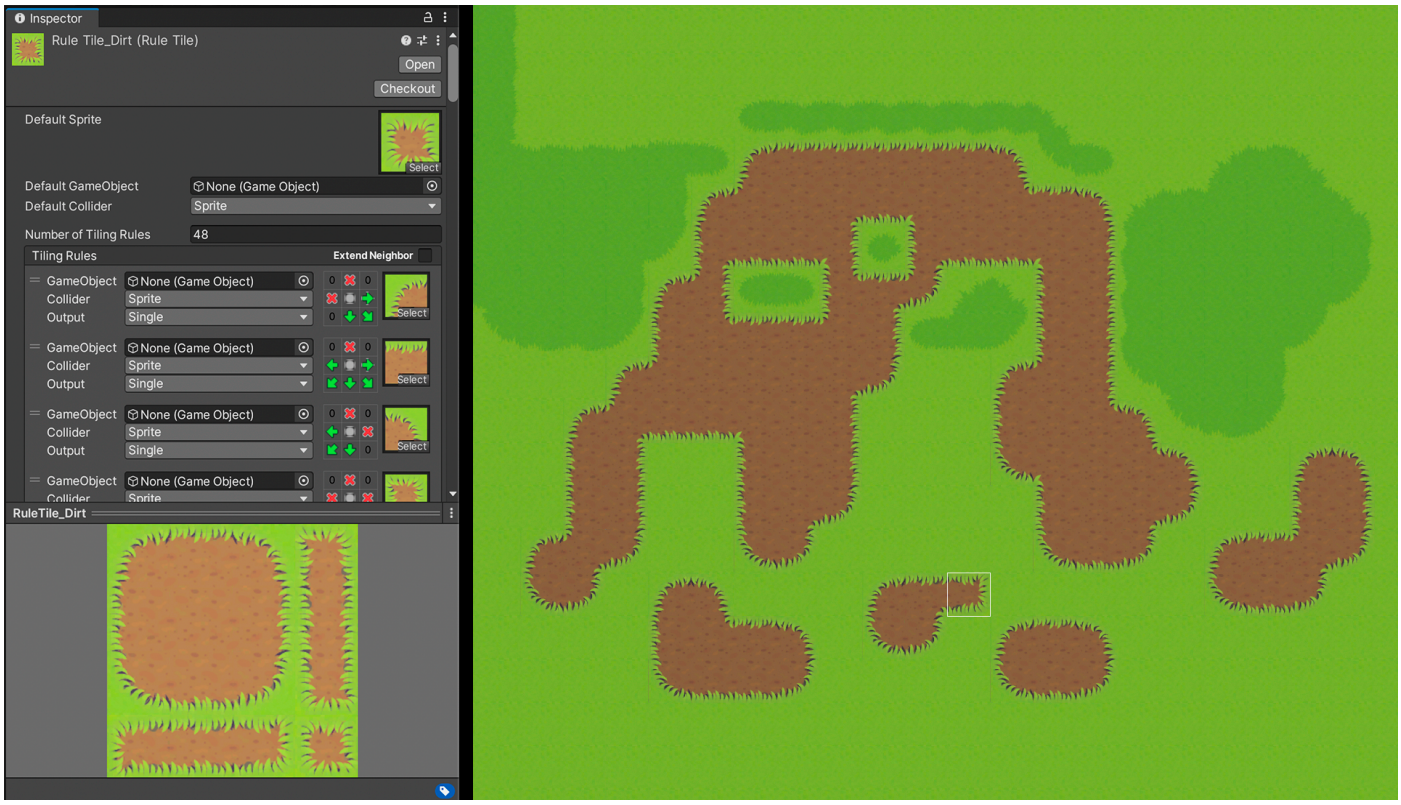
Used Textures フィールドにその他のテクスチャをロードできます。例えば、同じタイルシートのバリアントをロードし、**Random** オプションをチェックすると、同じ隣接基準に一致するタイルの間でランダムにタイルが選択されるので、大きなシーンで繰り返されるパターンの錯覚の問題を解決するのに役立ちます。

Unity 2D サンプルのタイルマップ

効率的な環境デザイン

ルールタイルは Happy Harvest 全体で広く使用されており、例えば草のある土のパッチを作成するために使用されます。サンプルの Project ウィンドウには、**Palette_Tiles** という名前のタイルパレットがあります。パレットの最初の行にはルールタイルがあります。プロジェクトをダウンロードして開き、Inspector でゲームオブジェクトを選択すると、アセットが Project ウィンドウに表示され、このアセットを選択してそのルールタイルの設定を確認できます。

注意: プロジェクトで適用できる場合、これらのタイルアセットを再利用したり、テクスチャの上にペイントして独自の用途に合わせることで、時間を節約できます。このサンプルに付属するスプライトを置き換えるか、ゲームのニーズに合った新しいルールタイルを作成することもできます。



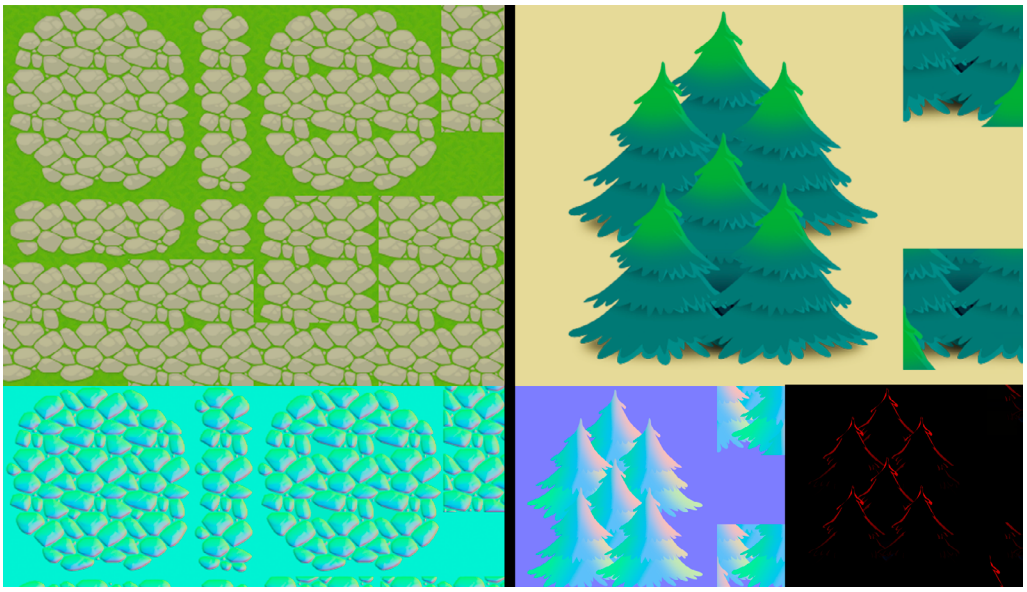
ルールタイルやオートタイルを設定すると、連続したパスを描くことがはるかに簡単になり、時間効率が良くなります。

このサンプルはオートタイル機能が導入される前に作成されましたが、前述のセクションで説明したように、同じ機能をオートタイルで再現できるはずです。

タイルマップ用の二次テクスチャ

Happy Harvest では、Tilemap フォルダー内のすべてのタイルマップには、同じ寸法とレイアウトを共有しながらも、ライティングを表示するためにペイントされた 法線マップとマスクマップテクスチャ が対応しています。

法線マップとマスクマップテクスチャは、スプライトエディターでタイルセットのメインテクスチャに追加されます。サンプルでは、マスクマップを使用してキャラクターや小道具のリムライトシルエット効果を作成しています。ただし、ほとんどのタイルは地面に使用されるため、リムライト効果は必要ありません。これが、このタイルセットのほとんどのテクスチャが黒い理由であり、リム効果のために作成された光を反射しないようにしています。建物の屋根に使用される建物タイルの場合は例外で、この場合は、これらのタイルの辺を強調する必要がありました。



(左) 石畳のためのより複雑なタイルシート。(右) 背景の小道具のためのよりシンプルなタイルシート。

タイルマップを使用したレベルとゲームプレイのデザイン

Happy Harverst のタイルはすべて同じサイズと形状で、同じグリッドゲームオブジェクトに含まれています。これにより、ゲーム内のタイルマップの数を少なくしておくことができます。**Terrain Manager** と呼ばれるコンポーネントがグリッドゲームオブジェクトに付属しており、ゲームプレイの可能性を広げています (後述のセクションを参照)。

グリッドゲームオブジェクトに含まれるタイルマップの例を以下に挙げます。

— **UnderwaterTiles:** これらは水中で見えるタイル、崖、池エリアの緑の地面に使用されます。

— **Water:** これらのタイルは、Splite-Lit-Material シェーダーを使用しており、水のアニメーションをシミュレートできるように ShaderGraph で作成されています。

— **GroundBase:** これは、道、草、泥、石のタイル用のタイルで構成された地面のアートワークで、高低差のあるタイルとの溝を埋めるものです。

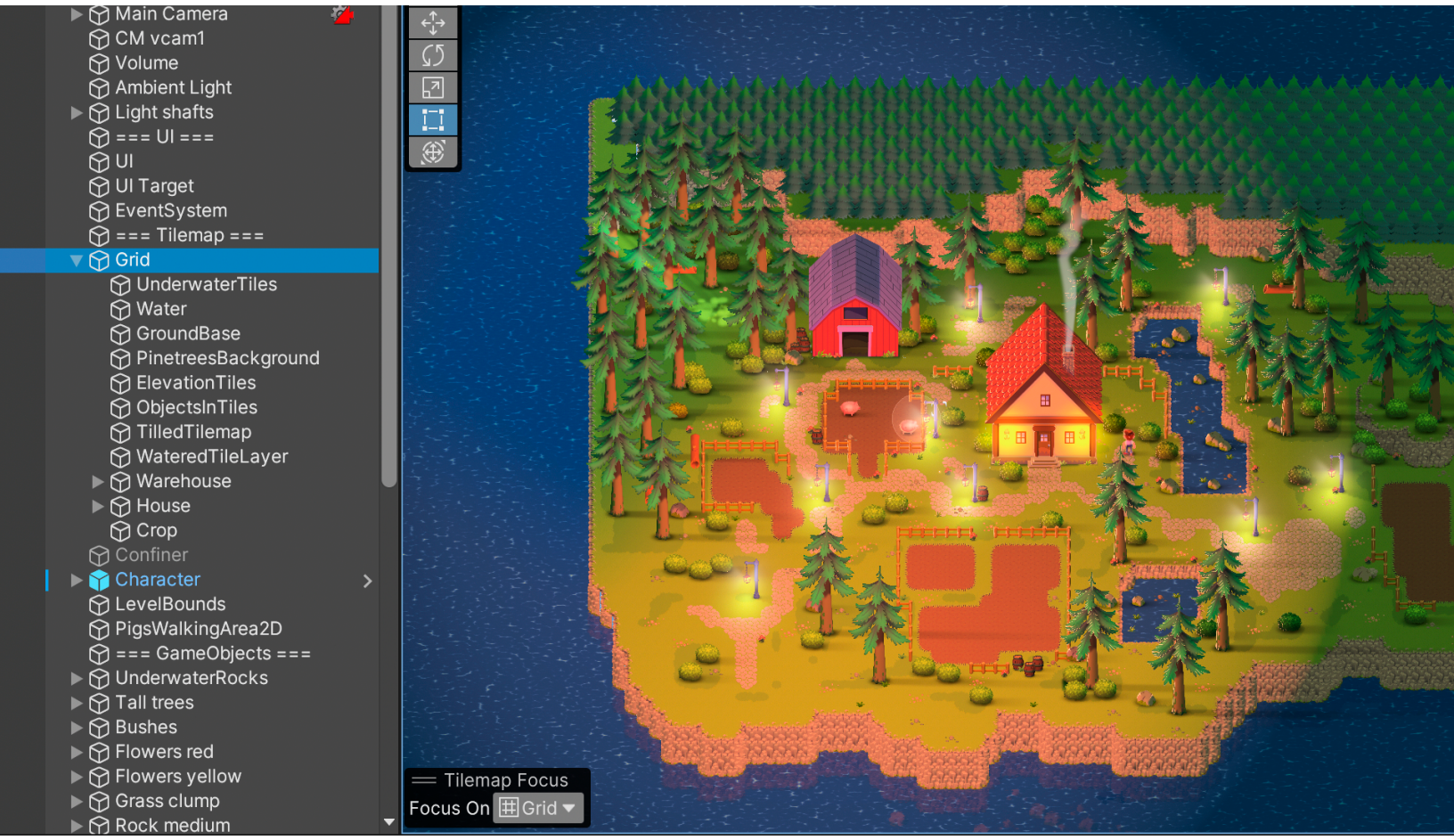
— **ObjectsInTiles:** これは、タイルパレットのゲームオブジェクトブラシでタイルマップに描かれたフェンスなどのゲームオブジェクトに使用されます。

— **TilledTilemap:** これらのタイルは、種を植えることができるタイルを検出するためのコードに使用されます。

— **WateredTileLayer:** 前述のタイルマップと同様に、これはゲームロジックによってタイルを湿ったように見せるために使用されます。タイルが乾くと、“湿った” タイルは削除されます。

— **Warehouse、House:** これらのタイルマップは、倉庫と家の作成に使用されます。タイルマップで建物を作成することで、アーティストは柔軟にそれらをより効率的に形状およびサイズを変更することが可能になりました。また、建物の一部で同じテクスチャを使用するため、テクスチャのスペースを節約することができました。

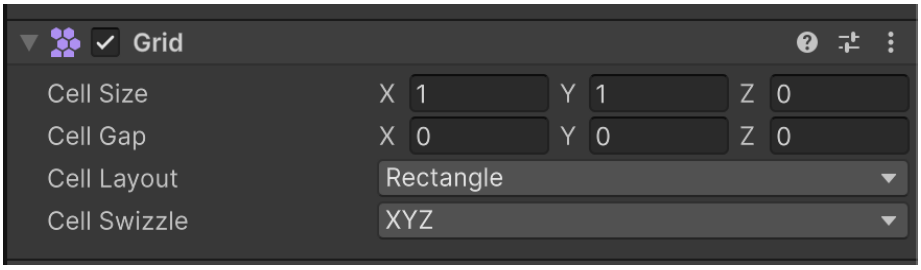
— **Crop:** これは、すべての植物を 1 つのゲームオブジェクトの下にまとめるために使用され、Terrain Manager でも使用されます。



タイルマップを含むグリッドゲームオブジェクト: Tilemap Focus ツールは、シーンビューで選択したタイルマップの内容のみを描画することによって、手元のタイルマップに焦点を合わせるのに役立ちます。

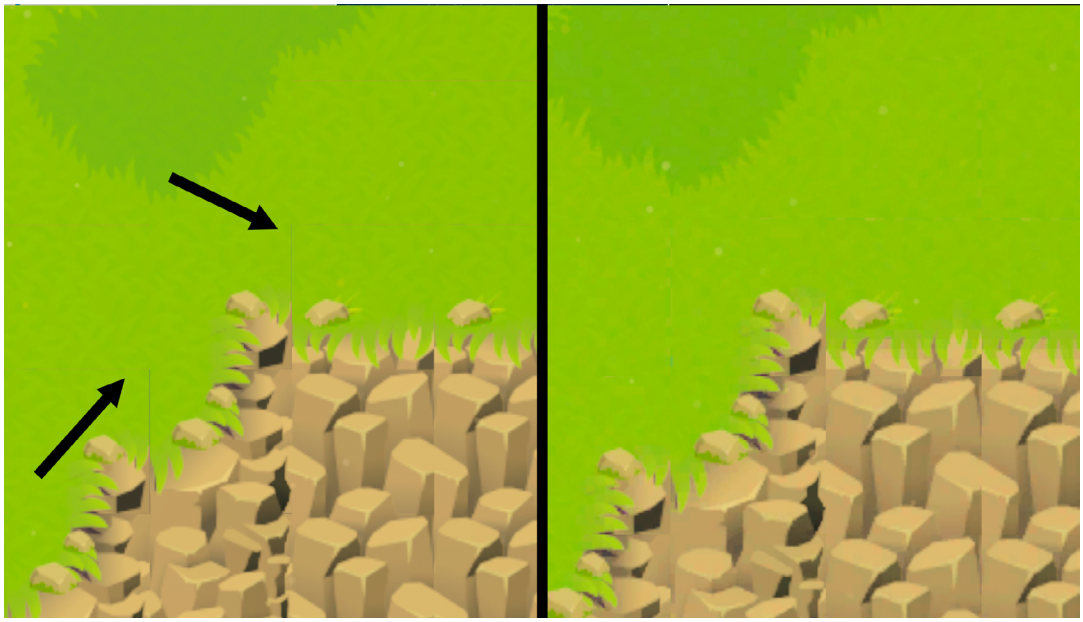
タイルマップのテクスチャのにじみを回避するためのヒント

アートやタイルマップに労力を注ぐのであれば、補間や辺のスムージングによるタイル間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が行わ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ない場合は、スプライトアトラス でスプライトのパッキングを行うことをお勧めします。これにより、タイルの端に透明な縁ができないため、継ぎ目が滑らかになります。さらに、タイルをわずかに拡大して隙間を避け、わずかな重なりを作ることに、タイルマップレンダラー内の内部ソートが役立つ場合があります。



グリッドコンポーネントのオプションで、隙間ができる可能性を軽減

デフォルトの設定はほとんどのスプライトで機能しますが、タイルマップではもう少し細かい調整が必要になる場合があります。スプライトをパックするときの回転の回避やアルファ拡張といった機能は、タイルの端をシャープに保つのに役立ちます。



左側の画像では、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。

ピクセルアートで作業している場合は、スプライトのフィルターモードを Point (フィルターなし) に変更すると、タイルが完全に一致し、隙間が一掃されます。

ゲームプレイ向けの Tilemap API

タイルマップは、グリッドベースのゲームに適しています。API は、視覚的なグリッドでタイルを読み書きする場合の簡単な方法であり、強力なレベルデザインツールとなるため、ゲームロジックの作成にも役立ちます。一般的に、グリッドベースのゲームプレイでは、コード内にのみ存在する多次元リストや辞書が使用され、何らかの視覚的な実装やオーサリングツールが必要となります。タイルマップは、これらすべてをすぐに利用できます。

Happy Harvest の Tilemap API

グリッドゲームオブジェクト内にある **Terrain Manager** スクリプトは、タイルの変更を処理し、タイルを使用して農場を追跡するための主要なマネージャーです。これは、グリッドベースのゲームプレイの設定に役立つTilemap API を使用しており、アイテムの二次元位置を簡単に特定できます。

この Monobehaviour クラスで、**GroundData** と **CropData** という 2 つの汎用クラスを作成します。これらには、タイルが湿っている時間の長さ、植物が成長する速さ、収穫せずにいた場合の枯れるまでの時間など、ゲームプレイに関連する変数が含まれています。

Terrain Manager スクリプトは、ゲームプレイ目的で使用されるタイルマップとタイルを参照します。

— **Ground Tilemap**: これは、プレイヤーが掘ったり種を植えたりできる場所を示す土のタイルのタイルマップを参照します。このタイルマップの外側のタイルを掘っても、何も起こりません。

— **Crop Tilemap:** これは、作物の親オブジェクトとして使用されるタイルマップゲームオブジェクトを参照します。この Tilemap API を使用すると、ランタイム中にこのタイルマップから作物タイルを配置、更新、削除できます。

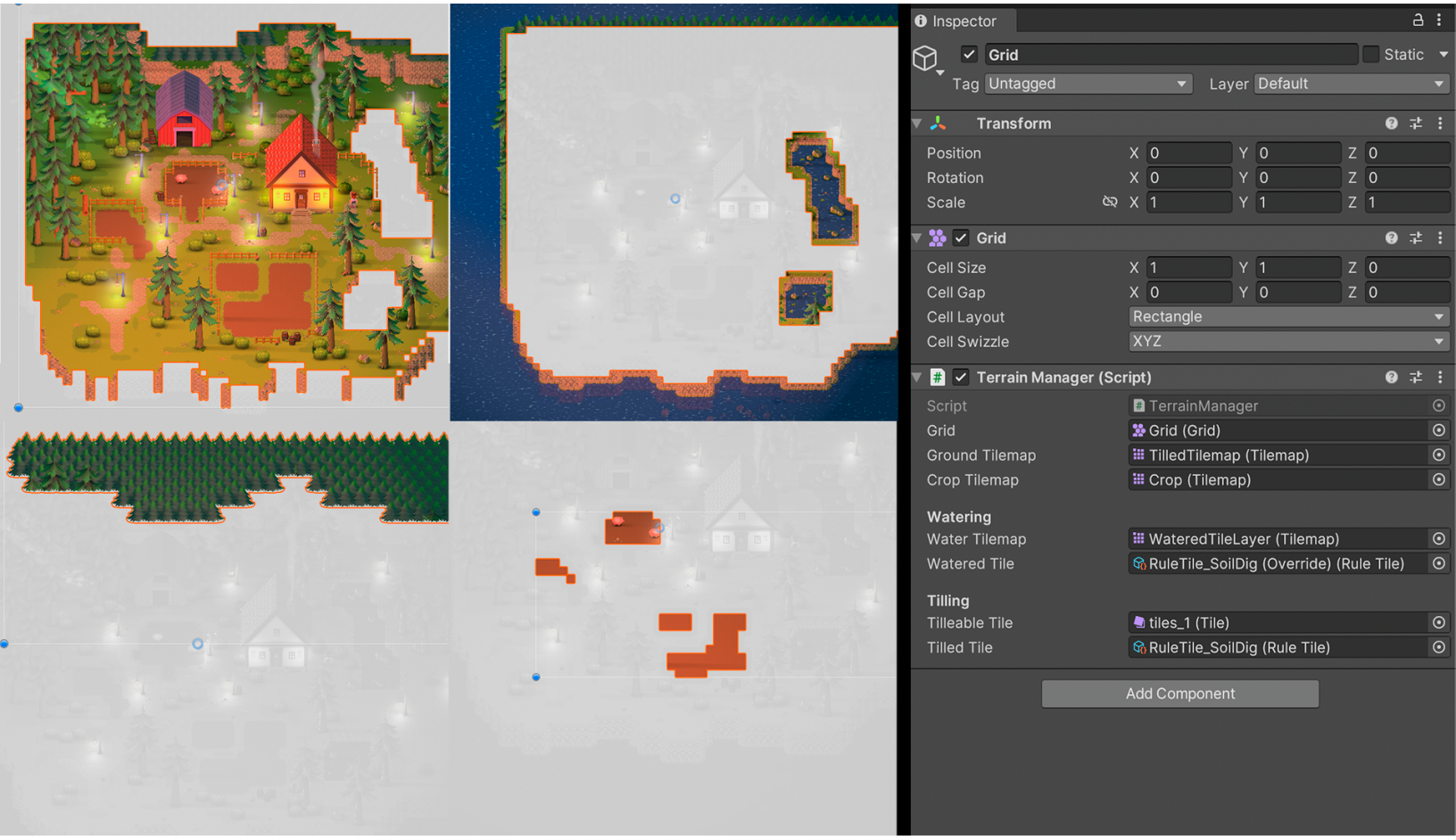
— **Water Tilemap:** これは、耕された土のタイルの上にペイントするために使用されます。水を模して、それらのタイルが植え付けや栽培の準備ができていることをプレイヤーに視覚的に伝えます。

— **Watered tile:** これは、水のグラフィックスを使用した ルールオーバーライドタイル です。新しいルールを設定せずに、ルールタイルのバリエーションを生み出すタイルを作成できます。これは、耕された土のタイルのルールをオーバーライドします。

— **Tilleable tile:** これは、掘削に使用できるタイルを識別するために使用されます。

— **Tiled tile:**これは、耕された土壌の外観をペイントするために使用されるルールタイルです。

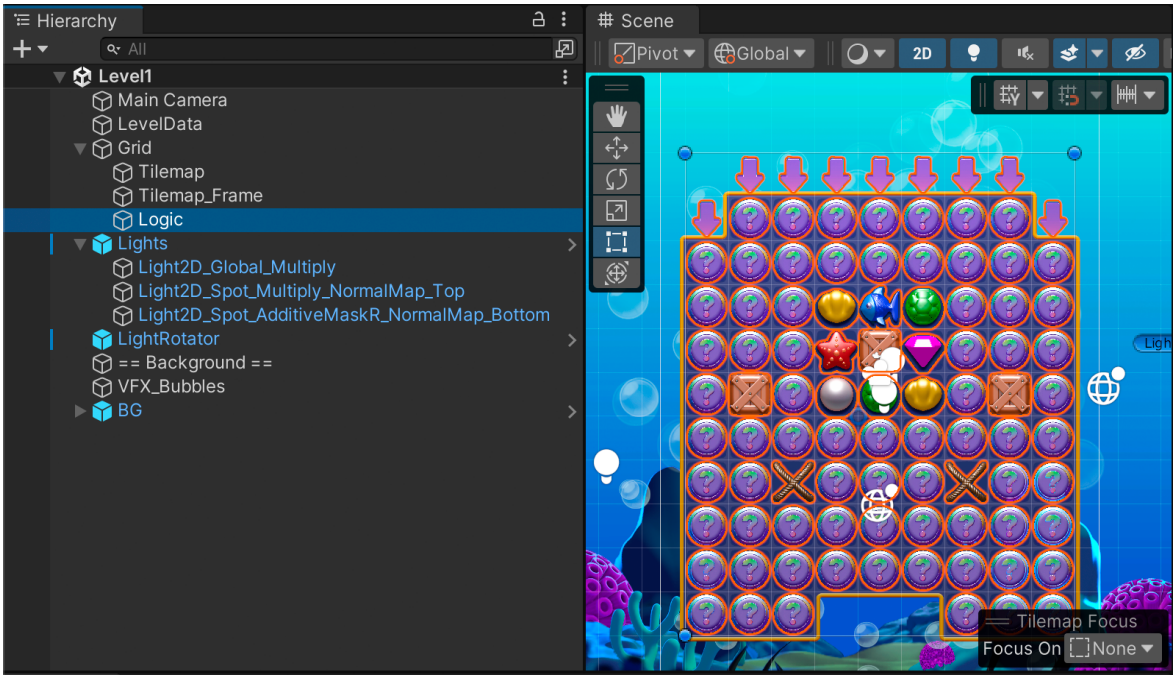
Terrain Manager のスクリプトには、Tilemap API を使用して VectorInt 位置形式でタイル情報を読み取り、タイルを適切に更新する機能が含まれています。これらの機能は、PlayerController がツール上で関数をトリガーする際に、さまざまなツールから呼び出されます。



Terrain Manager システム - ゲームプレイロジック向けのタイルマップを使用

Gem Hunter Match の Tilemap API

Gem Hunter Match は、主にモバイル向けに設計されたマッチ 3 パズルゲームですが、複数のプラットフォームで動作します。



Gem Hunter Match のレベルシーンの構造はシンプルです。

レベル構造を見ると、タイルマップが視覚アセットだけでなくゲームロジックにも使用されているのがわかります。その例は以下の通りです。

- **グリッドゲームオブジェクト:** このゲームオブジェクトには、レベルで利用可能な宝石の種類を決定する **Board** というスクリプトの他、VFX グラフベースの視覚効果への参照も含まれています。
- **子ゲームオブジェクト**: グリッドゲームオブジェクトにはいくつかの子ゲームオブジェクトが含まれています。子ゲームオブジェクトには以下が含まれています。
  - **Tilemap:** これは背景タイルと装飾を作成するために使用され、多くのピースの位置を追跡する便利な方法を提供する Tilemap API を備えています。
  - **Tilemap_Frame:** これは背景タイルの周りにフレーム装飾を作成するために使用されます。
  - **Logic:** これにはレベルデザインのためのプレースホルダータイルが含まれています。ここからプレースホルダー “ランダムジェム” タイル (疑問符で表示) のすべてが実際のゲームプレイのゲームオブジェクト (コードによりランダムに生成) に置き換えられます。ただし、パレット内の他のタイルを使用することで、特定のタイル位置に発生する宝石やアイテムの種類を選択することもできます。

タイルマップのベストプラクティスの詳細についてはこちらのビデオを参照してください。

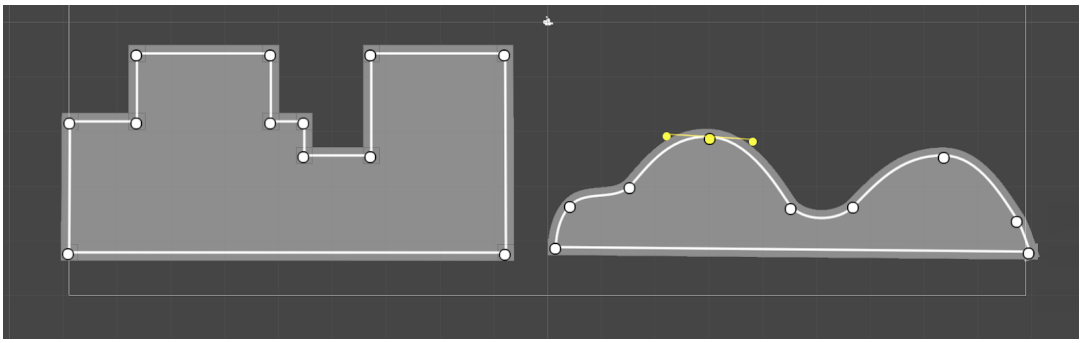
2D スプライトシェイプ

2D Sprite Shape (2D スプライトシェイプ) ツールの特徴は、形状のアウトラインに沿ったスプライトタイルです。アウトラインの角度に基づいてスプライトを自動的に変形させたり交換したりします。さらに、2D Sprite Shape ゲームオブジェクトに塗りつぶしテクスチャを割り当てることで、タイル状のテクスチャで塗りつぶされた形状を背景や他の大きなレベル構築用のプロップとして使用して作成できます。

新しい Unity 2D サンプルプロジェクトの Bunny Blitz には、背景のツタや丘のように、2D スプライトシェイプで作成された装飾要素があります。

2D スプライトシェイプで作成したパスは、グラフィックスソフトウェアで使用される既知のペンツールと同様に機能します。それらはベジェ曲線であり、シーン内で直接編集したり、オプションで閉じてタイル状のテクスチャで塗りつぶしたりすることができます。

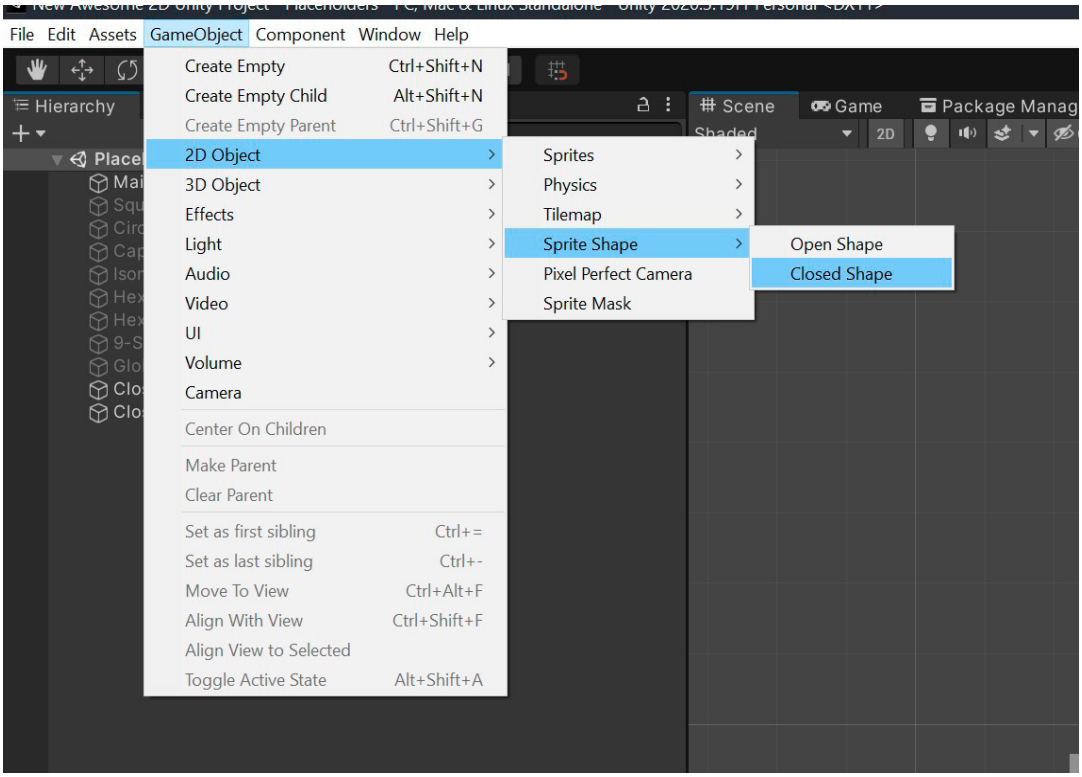
Sprite Shape は、Unity の便利なプロトタイプ機能であり、新しいスプライトの形状を効率的に作成および編集できます。



Sprite Shape では、Linear と Continuous という 2 つの接線モードを利用できます

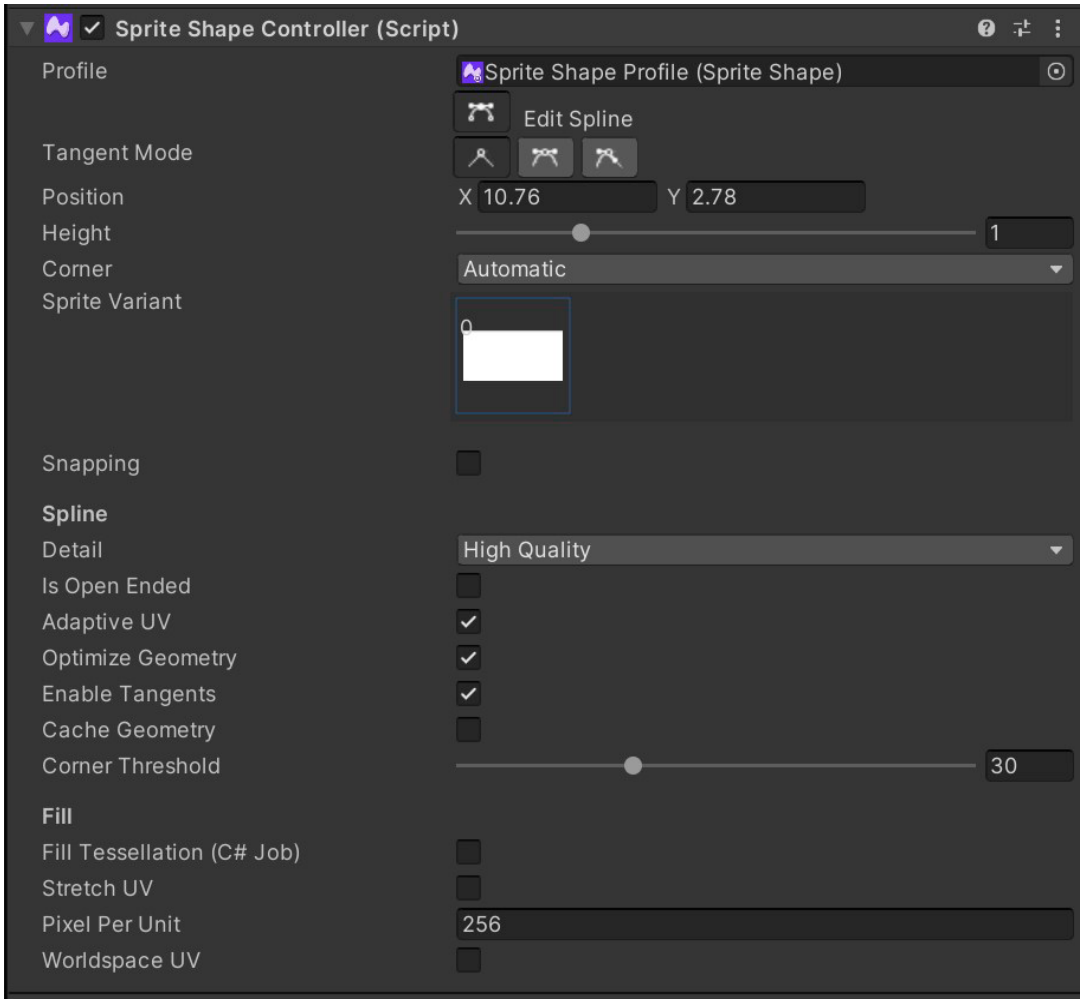
2D プロジェクトを 2D テンプレートで作成することを選択する場合、Sprite Shape はデフォルトでインストールされます。それ以外の場合は、Package Manager を使用してインストールできます。

新しい Sprite Shape を作成するには、GameObject メニューをクリックし、**2D Object > Sprite Shape** を選択し、Open Shape または Closed Shape のいずれかを選択します。



Sprite Shape の作成

Sprite Shape を変更するには、変更する Sprite Shape を選択し、Inspector の編集ボタンをクリックします。



Inspector ウィンドウでの Sprite Shape の変更

Sprite Shape の任意の部分を左クリックして、点を追加するか、削除キーで削除します。点を選択すると、以下の 3 つの **Tangent Mode** ボタンのいずれかを選択して、そのモードを変更できます。

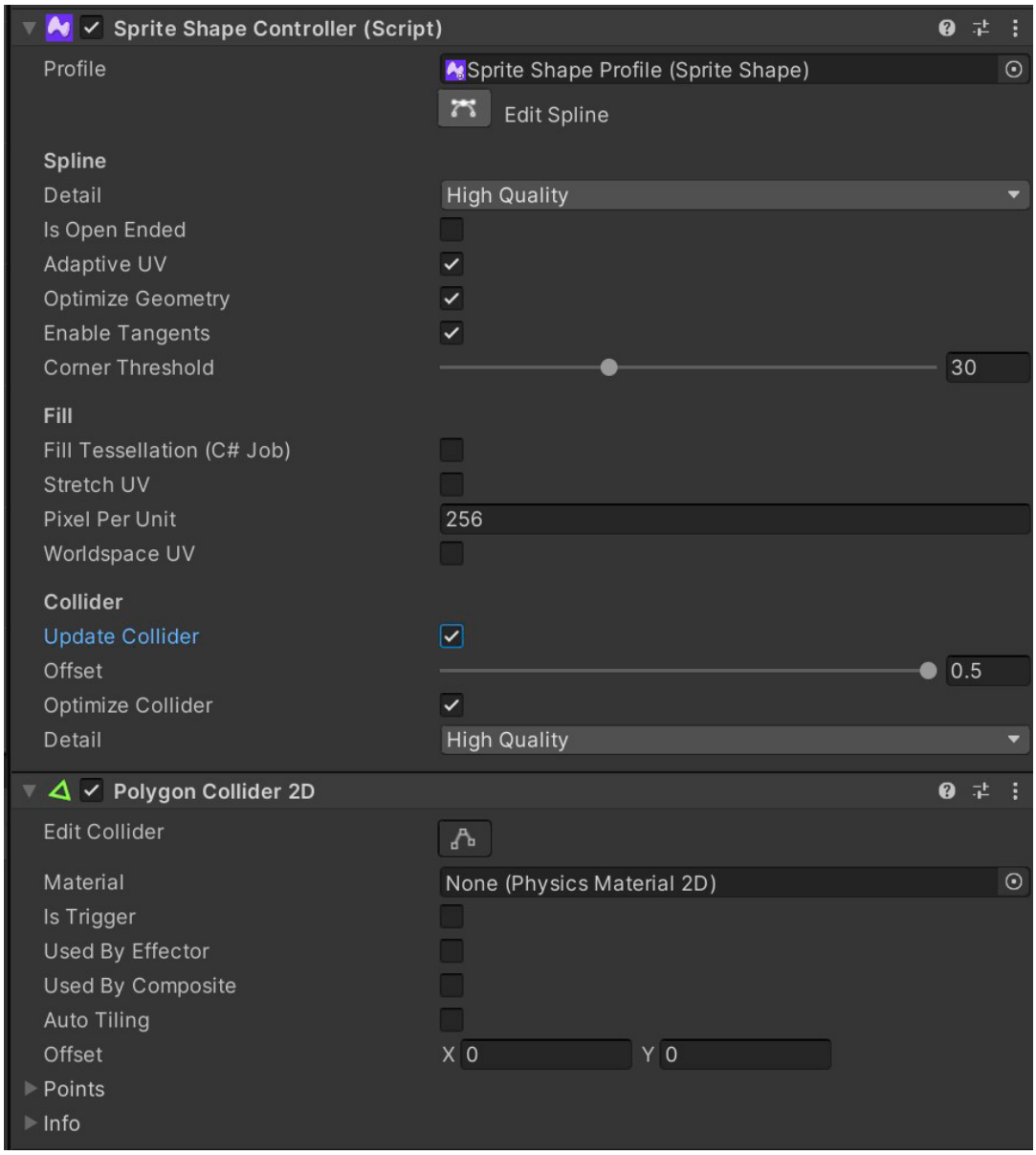
— **Linear**: 曲線は作成されず、点の両側に直線が残ります。

— **Continuous**: 反対方向を向いたハンドルで、点を中心に曲線を作成します。

— **Broken**: 独立して動くハンドルで、点を中心に曲線を作成します。

点を中心とする角の外観を選択することもできます。スナップツールは、点をグリッドにスナップするために便利なオプションです。

物理演算のために **Polygon Collider 2D** コンポーネントを追加します。プロトタイプ作成の目的であれば、デフォルトのプロパティで十分です。



Sprite Shape ゲームオブジェクトに Polygon Collider 2D コンポーネントを追加します。

これらのステップに従って、相互作用するスプライトシェイプを作成し、Inspector を使用して変更できます。

Sprite Shape のレベルデザイン

グリッドベースまたはタイルベースのデザインは、特にピクセルアートを使用している場合、レトロな 2D のビジュアルスタイルに適しています。横スクロールのプラットフォーマー、トップダウン RPG の他、建物や城などの非有機的な形状にも最適です。

グリッドベースのデザインにより、経路検索とレベル作成がより簡単になります。グリッドに限定すると、経路検索はよりシンプルになります。レベルを設計する際に、一定の距離を維持しやすくなります。例えば、キャラクターのジャンプが 3 ユニットの高さであれば、プラットフォームを配置する場所を簡単に計画でき、プレイヤーが瞬時にジャンプできるかどうかを想定することができます。

より有機的で、ブロックベースではないスタイルを望む場合は、2D スプライトシェイプを使用します。曲線を使用して、任意の形状を作成できます。地形、丘、草原、滑らかな表面はすべてスプライトシェイプに適しており、くっきりとしたピクセルアートスタイルに比べてシーンに現代的な外観を与えます。

スナップツールを有効にして、任意の角度や曲線を簡単に設定し、要素を正確に配置します。

(左) Skul: The Hero Slayer は SouthPAW Games が制作したタイルマップベースのゲーム。(右) Oddmar は Mobge LTD が制作したスプラインベースのゲーム。

スプライトシェイプで作成されたレベルや要素は、後で簡単に編集できます。ビジュアルスタイルに応じて、ゲーム内で両方のシステムを使用できます。

スプライトシェイプの塗りつぶしや境界線のマテリアルに、任意の Shader Graph のスプライトベースのシェーダーを使用することで、独自のスタイルを加えたり、さらに一歩進んで生成される形状を変更したりすることができます。Unity 6 以降、カスタムジオメトリ機能が Sprite Shape Controller で利用できるようになったので、カスタムスクリプトを使用してスプライトシェイプのジオメトリを生成または変更できます。

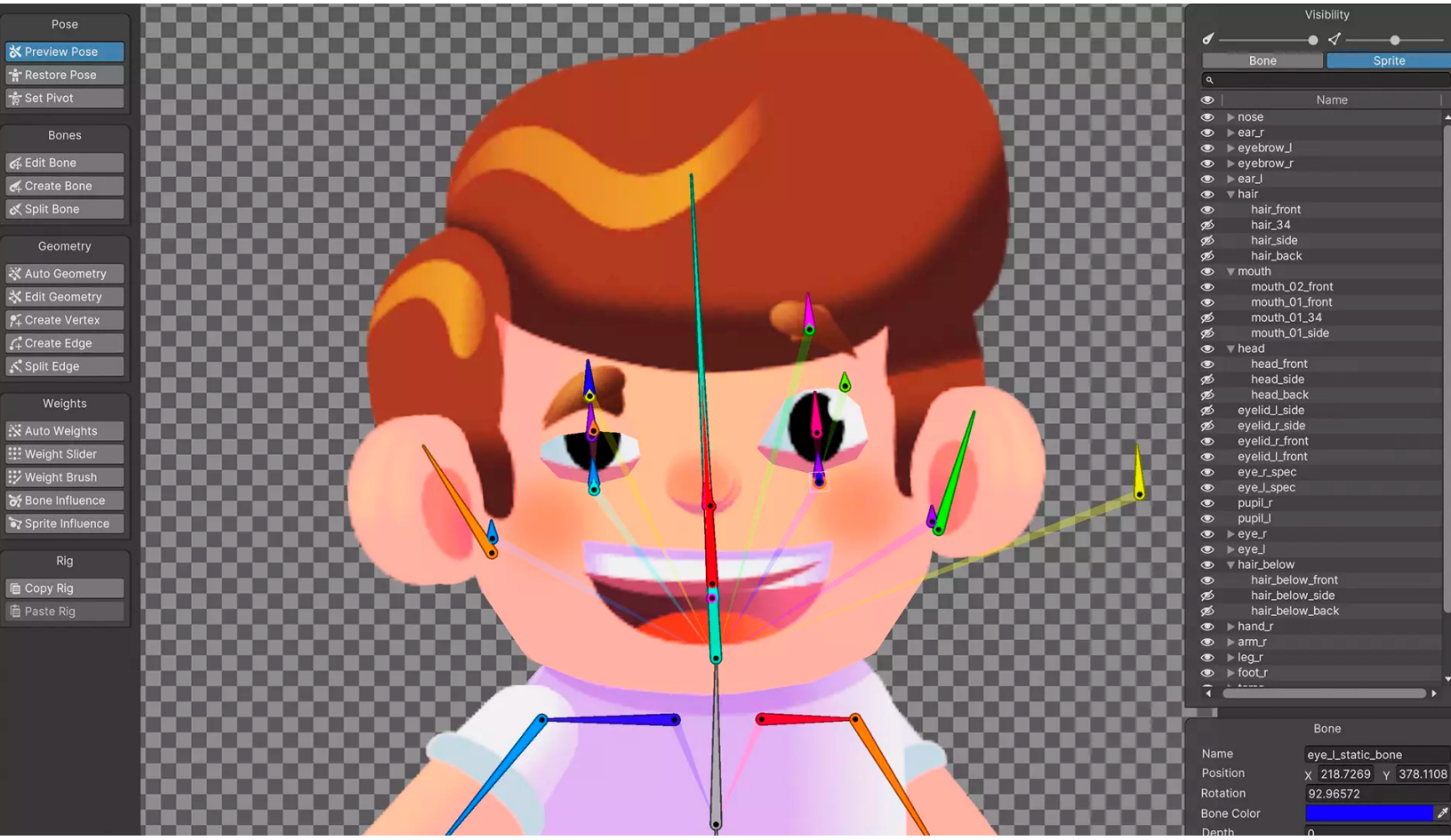
タイルマップとスプライトシェイプの両方とも、ランタイム中に変更して、新しいエキサイティングなゲームプレイの可能性を生み出すことができます。例えば、タイルマップは破壊が可能であり、スプライトシェイプは変形可能なので、多くの可能性があります。オプションの詳細については、Tilemap API と SpriteShape API のドキュメントを参照してください。

2D アニメーション：スケルタルアニメーション

Unity の 2D アニメーション機能を使用して、さまざまなジャンルやスタイルのキャラクターをアニメーション化します。Unity Asset Store から無料の 2D Animation Sample パッケージを入手して、詳細を見てみましょう。

2D アニメーションは、ゲームのアートを開発する上で最も時間がかかり、大変な部分です。アニメーションキャラクターを作成するには、アニメーションのタイミング、運動量、運動学などの知識が必要です。すべてのフレームを整えるには時間がかかることがあり、これらのフレームを保存して表示するには多くのメモリが必要です。キャラクターのタイミングや任意の部分を変更するには、すべてのフレームを再描画する必要がある場合があります。

歴史的に、3D アニメーションは 2D よりも単純でした。3D モデルを作成し、スケルトンを追加してリグを設定し、ボーンウェイトを設定し、ソフトウェアが補間するキーフレームを設定してアニメーション化します。キーフレームを編集することで調整を行います。



Sprite エディターで Happy Harvest のキャラクターの表情のリギングをテスト

Unity は、効率的な 2D アニメーション のツールセットを提供しており、例えば Photoshop からキャラクターアートワークを Unity に直接インポートするなど、さまざまな点で 3D アニメーションのプロセスを模倣しています。

デザイン、インポート、リグ

ゲームで最も重要な要素はプレイヤーのキャラクターです。キャラクターのデザインを考えるために十分な時間をかけてください。計画中に考慮すべき重要な面がいくつかあります。

視点

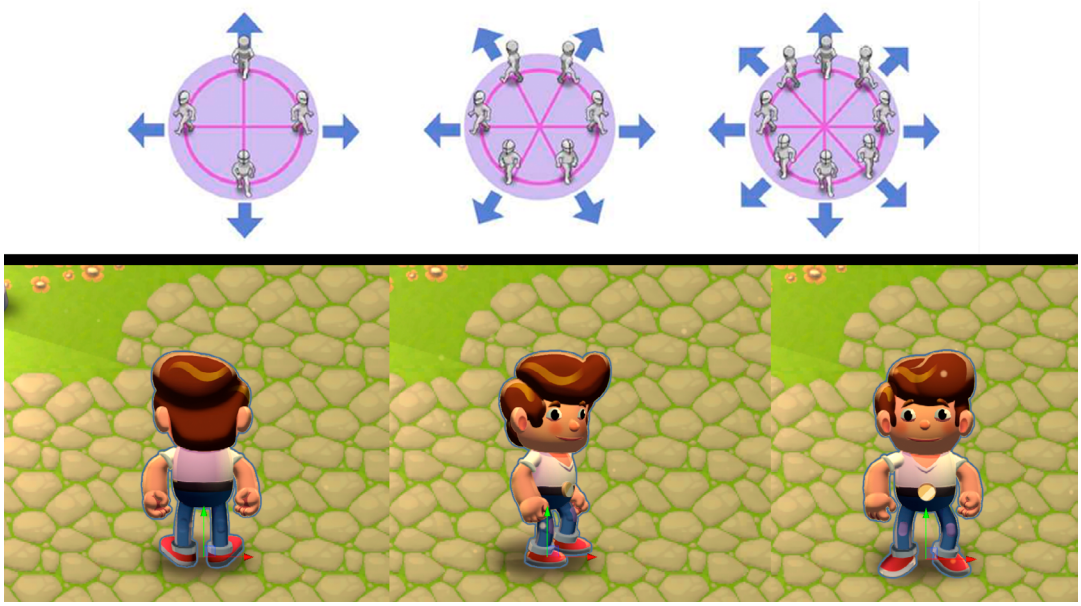
選択する視点は、ゲームキャラクターの見た目やアニメーションに影響を与えます。ほとんどのゲームビューでは、キャラクターは横顔で描画できます。3/4 ビューにわずかに回転することで、顔の特徴をより多く表示できます。

等角または上から 3/4 視点のトップダウンビューを使用する場合、カメラを少し上から傾けて顔の詳細を見えるようにすることで、似たような視点を作ることができます。この視点では、キャラクターは複数の方向を向いて描かれます。望む結果と予算に応じて、一般的に使用される 3 つのオプションのいずれかを選択できます。

— 4 方向

— 6 方向

— 8 方向



Happy Harvest - 2D のトップダウンまたは等角投影視点ゲームを作成する際のキャラクター回転のさまざまな例

どれを選んでも、右向きまたは左向きのアニメーションを反転させて反対方向を向かせる必要があるかどうかを決定する必要があります。この場合、以下のようにそれぞれ描画する必要があります。

— 3 方向: 右、上、下

— 3 方向: 右、右上、右下

— 5 方向: 右、右上、上、右下、下

アニメーションが反転する場合、キャラクターの手も一致するように反転します。キャラクターが右を向いて右手に剣、左手に盾を持っている場合、キャラクターが左を向くように反転すると、両方の武器の手を入れ替えます。この妥協を受け入れるか、すべてのアニメーションが正しい方向を向くようにするためにもっと時間をかけるかを決定してください。

従来の横スクロールの格闘ゲームを設計している場合、使用するのは 1 つの向きの方向だけで済みます。上下に移動するキャラクターは、横に動いているように見えます。

テスト後にキャラクターがゲームの環境で良く見えるかどうかは、最終的にはあなた次第です。現実的な視野角や視点を持つことよりも、構想が重要です。

キャラクターの再スキン

2D Animation パッケージの利点は、複数のキャラクター間で同じスケルトンとアニメーションを共有できることです。基本キャラクターを 1 つデザインし、リグを設定し、アニメーションを作成したら、そのスキンを簡単にスワップできます。スケルトン共有 という機能でこのプロセスを効率化します。

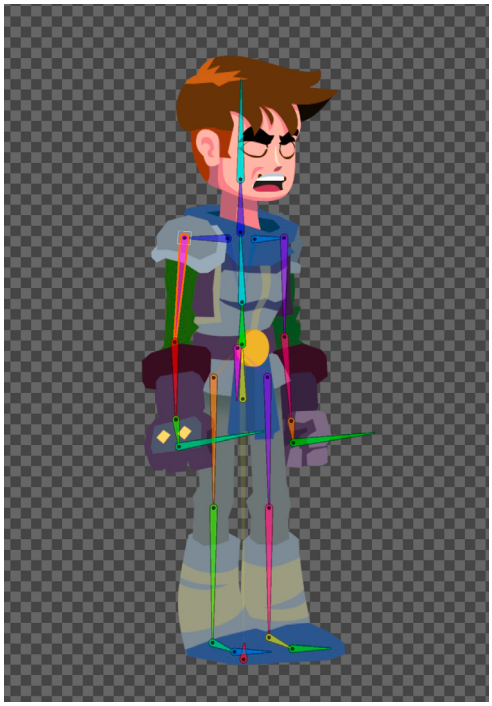
事前に計画することで時間を節約しましょう。同じスケルトンを共有するすべてのキャラクターのコンセプトを描き、そのスケルトンがすべてに合うかどうかを確認します。別のレイヤーにスケルトンを描き、それをすべてのキャラクターに重ねることで確認を行います。レイヤーの数はすべてのキャラクターに対して同じでなければならないことに留意してください。

画像編集アプリケーションでスケルトン描画を重ねて表示

一般的な 2D アニメーションのルール

後でリグ付けやアニメーション化を行うキャラクターをデザインする際に、従うべきよいルールをいくつか紹介します。

- 腕と脚を自然な位置に置いたキャラクターを描きます。体のパーツが曲がって描かれていると、アニメーションを作成する際に問題が発生する可能性があります。
- ゲームの PPU で推奨される解像度よりも少し高く設定してください。解像度は静止時には問題なく見えますが、画像を回転させたり引き伸ばしたりするとピクセル化が発生する可能性があります。
- 2D ライトの章でヒントを後述しますが、法線マップを使用する予定がある場合、スプライトのライトと影をペイントしないでください。代わりに、非方向性の影をペイントします。この技術はアンビエントオクルージョンと呼ばれ、スプライトの外観はよくなりますが、太陽光のようなディレクショナルライトは避けてください。



中立的なキャラクターポーズの例: 脚と腕は任意の位置にできますが、曲げずにまっすぐにしておく必要があります。曲がった手足を無理に曲げるとピクセルが伸びる原因になります。

— スプライトスワップ 機能を使用してスワップされた体のパーツのレイヤーは、適切にグループ化する必要があります。例えば、口の位置を示すすべてのレイヤーは、画像編集アプリケーション内で “口” という名前のグループに配置する必要があります。

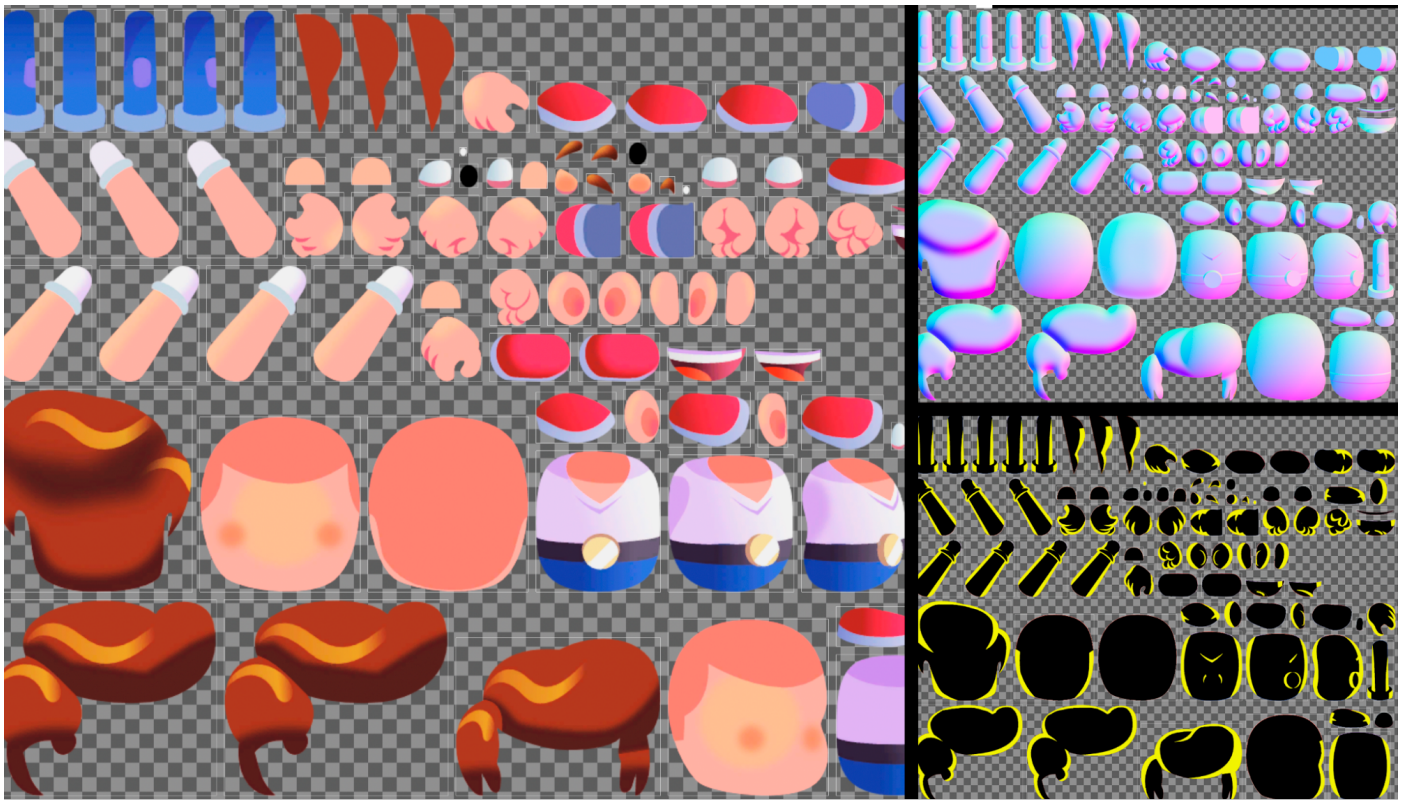
これでデザイン時に考慮すべきことが理解できました。お好みのグラフィックスアプリケーションを起動し、ペイントを始めましょう。

Unity にキャラクターをインポート

2D Animation パッケージによって、キャラクターのアートワークを Photoshop から Unity に直接インポートすることができます。PSD Importer パッケージでは、この目的のためにレイヤー化された PSD ファイルおよび PSB ファイルを処理する機能が提供されます。これにより、キャラクターのすべてのレイヤーをスプライトとしてインポートし、アプリケーションで描かれた通りに配置することができます。

Unity の 2D プロジェクトテンプレートを使用している場合、インポーターはすでにインストールされています。インストールされていない場合は、Package Manager を使用してインストールしてください。

エクスポートに必要なのは、ファイルを PSD または PSB 形式で保存することだけです。PSD は Photoshop やその他の制作ツールでは一般的です。



主要なキャラクターパーツがレイヤーで表示されたインポーター PSD のスプライトシート。右側は、ライティングのために二次テクスチャとして追加された PSD ファイルです。

Unity へのインポートは、他のアセットと同じです。ファイルを Assets フォルダーに保存するか、Project ウィンドウにドラッグします。

キャラクターの Inspector

インポートした PSD または PSB ファイルが選択されると、スプライトのインポート設定に似たオプションが表示されますが、2D アニメーションとリギング用のオプションが追加されています。設定すべき重要なオプションは以下の通りです。

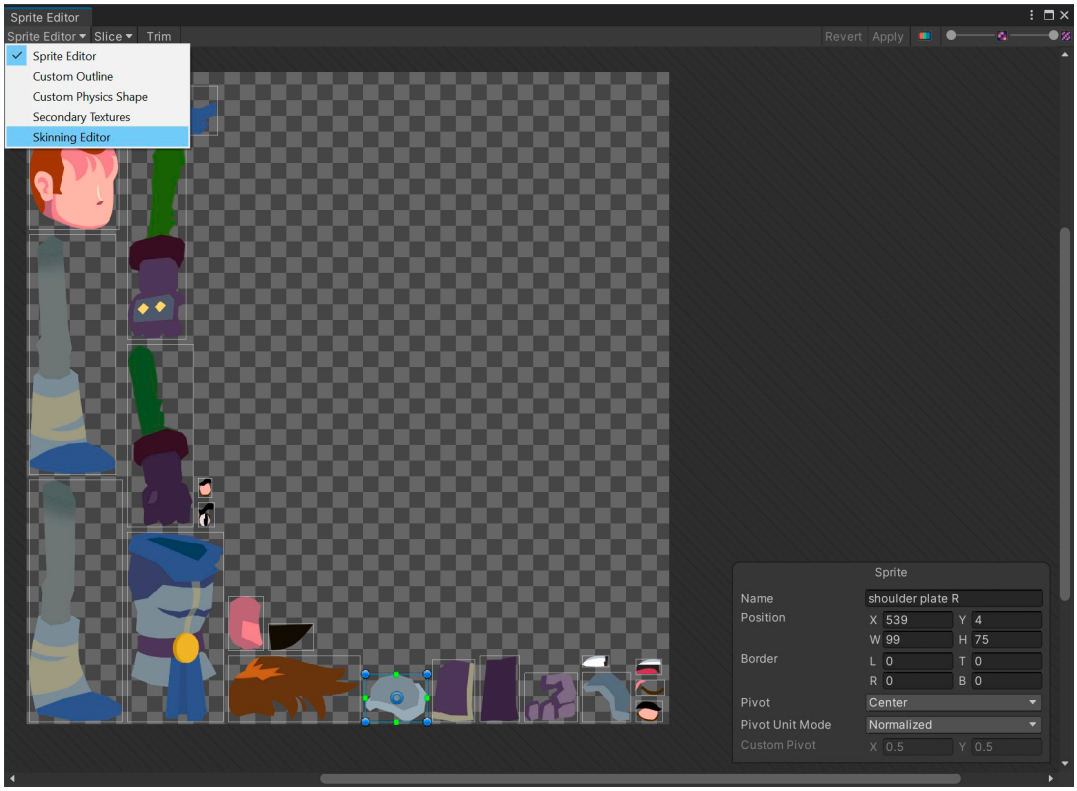
- **Import Hidden Layers:** このオプションで、非表示レイヤーを含むすべてのレイヤーを PSD または PSB ファイルから読み込みます。
- **Individual sprites (Mosaic):** この設定は、Texture Type が Multiple に設定されている場合にのみ利用可能です。インポートしたレイヤーからスプライトを作成し、テクスチャアトラス上に配置します。キャラクターをリギングする場合は、このオプションを有効にしてください。
- **Use as Rig:** このオプションでは、PSD または PSB ファイルと同じレイヤー階層と位置を持つキャラクタープレハブを生成します。これを有効にする必要があります。
- **Use Layer Group:** PSD または PSB ファイルからレイヤーグループを追加します。これを有効にすると、スプライトを入れ替えたパーツなど、キャラクターの一部をグループ化できます。

Layer Management セクションでは、レイヤーの階層を確認したり、インポートするレイヤーを手動で追加または削除したりすることができます。オプションを設定したら、Apply をクリックします。これにより、キャラクタープレハブが確定し、シーンにドラッグできるようになります。

Unity Asset Store から 2D Animation Sample パッケージを取得してください。

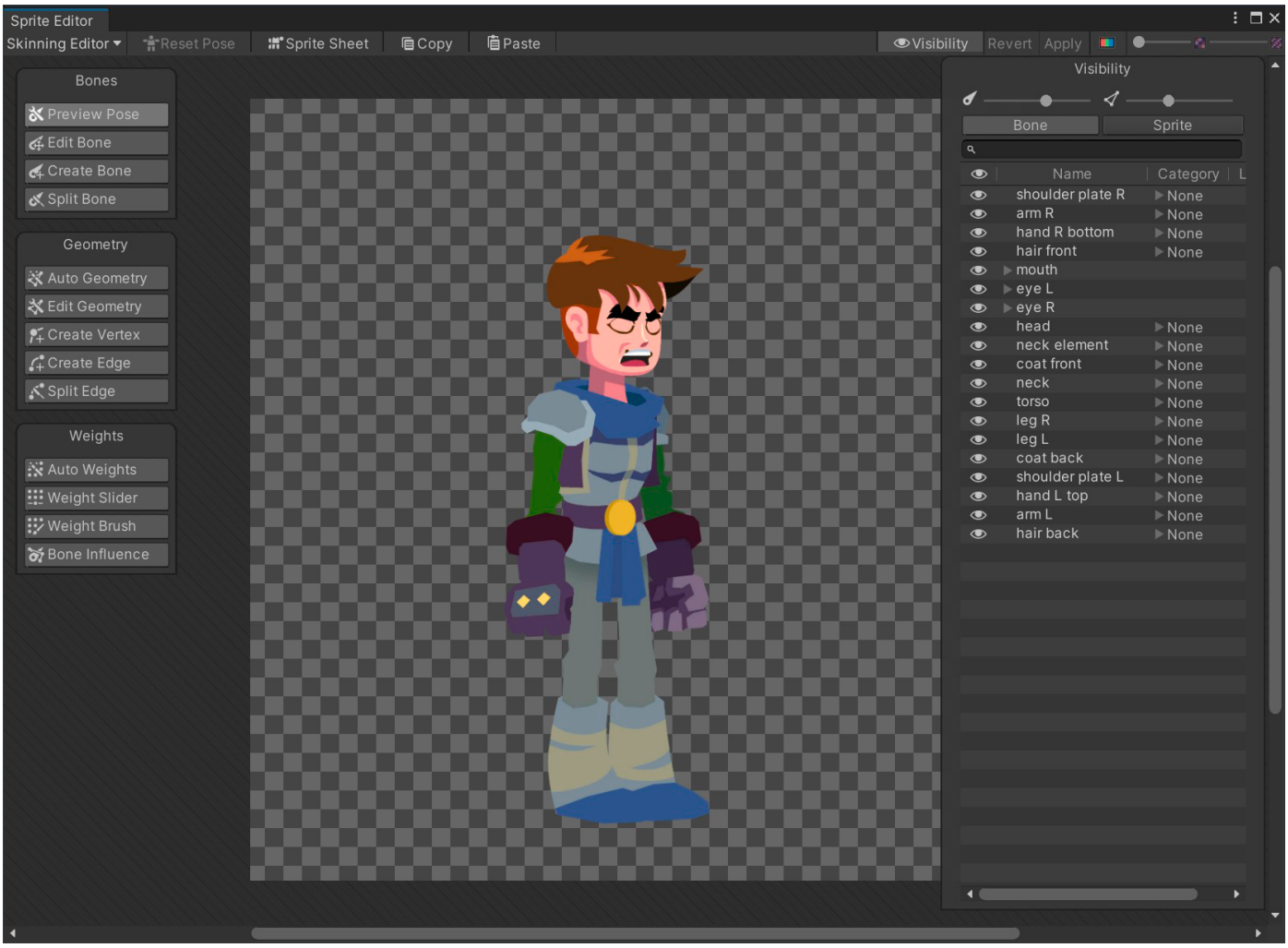
Sprite Editor でのリギング

キャラクターのリギングを開始するには、Inspector の Sprite Editor ボタンをクリックして、Sprite Import 設定にアクセスします。



Sprite Editor のウィンドウでツールを選択する方法

スプライトエディターのウィンドウの左上隅にあるドロップダウンメニューから Skinning Editor を選択します。



キャラクターをアニメーション化するオプションがある Skinning エディター

このウィンドウでは以下を実行することができます。

— ボーンの作成および編集

— スプライトジオメトリの作成および編集

— スプライトに対するボーンの影響の編集

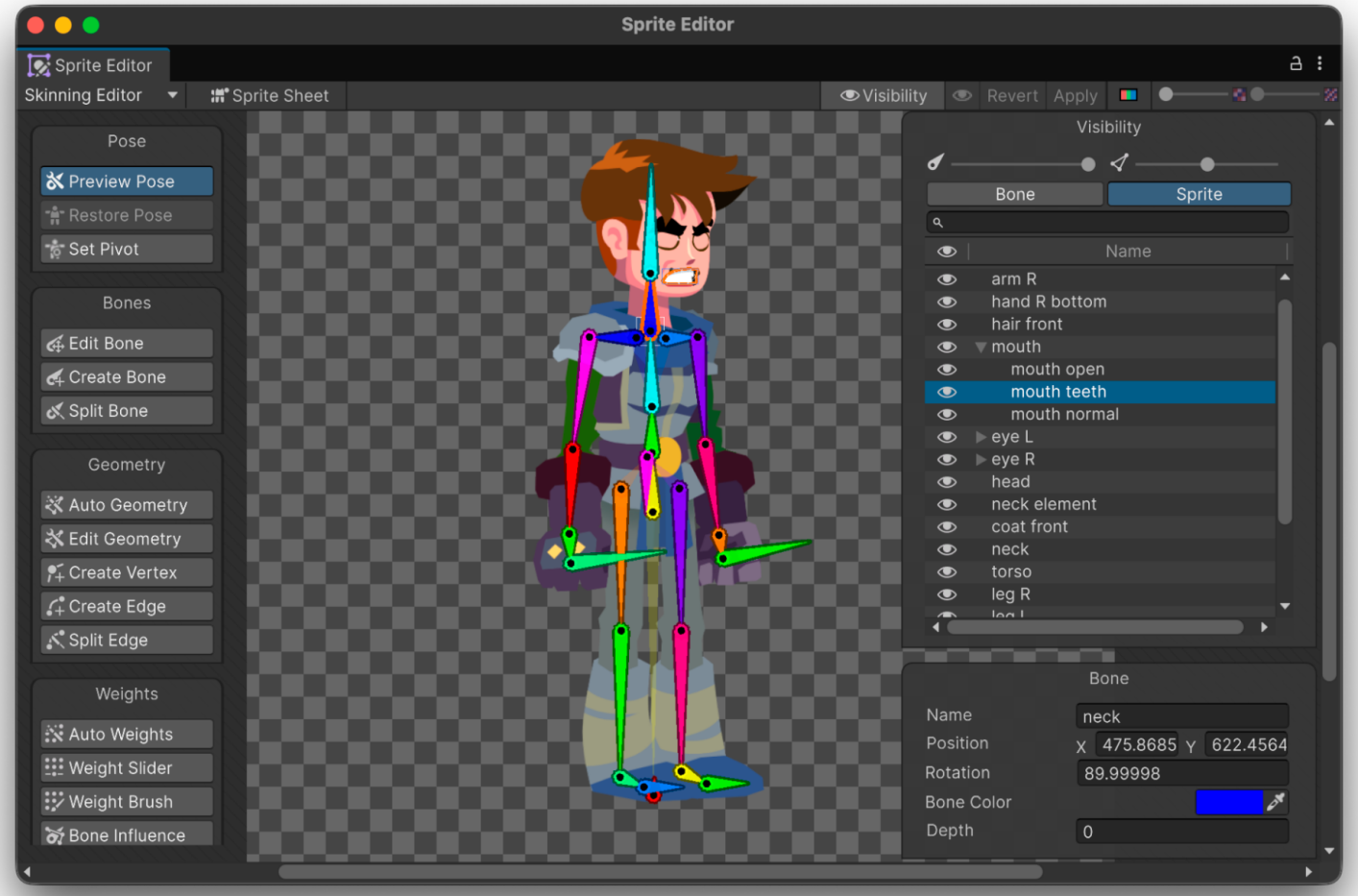
— スキンやスプライトのスワップで使用するためのカテゴリとラベルの作成

スケルトンの作成

キャラクターのスケルトンを作成することから始めます。**Create Bone** ボタンを選択し、メインウィンドウエリアを左クリックします。最初のクリックでボーンの中心が作成され、2回目のクリックでボーンの先端の位置がマークされます。このツールで、ボーン同士を連結し、ネスト状態にします。

ボーンの配置を変更したい場合は、右クリックしてから新しいボーンを配置したい場所を左クリックします。作成するボーンの親となる既存のボーンを制御するには、左クリックで既存のボーンを選択し、その後新しいボーンを作成します。

Edit Bone ボタンを使用してボーンを微調整できます。**Split Bone** ボタンを使用すると、ボーンを 2 つに分割できます。これは、手足を作るのに適した選択肢です。片脚のボーンを作成し、膝があるべき場所をクリックすると、ボーンは太ももとふくらはぎに分かれます。



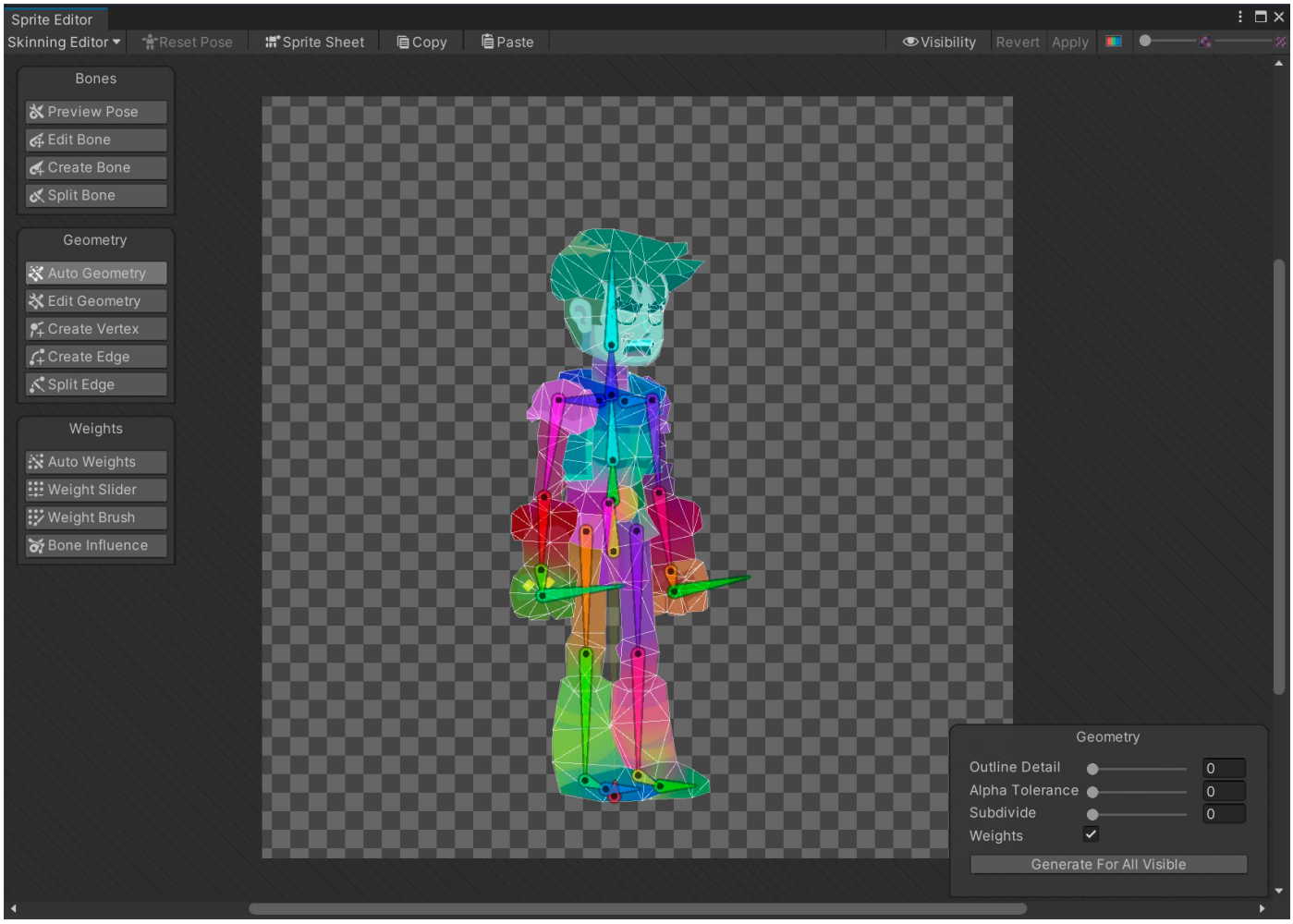
親ボーンと子ボーンを整理するために開かれた Bone タブ

ボーンリストビューでボーンの親を変更したり、名前を変更したりすることもできます。このビューを開くには、上部バーの右側にある Visibiilty ボタンをクリックし、次に Bone タブを選択します。ボーンの親を変更するには、リストビューでドラッグするだけです。アクティブなボーンの名前をクリックすることで、ボーンの名前を変更できます。ボーンに名前を付けることで、後で見つけやすくなります。階層が正しいことを確認するには、Preview Pose ボタンを選択していくつかのポーズを試します。ボーンの位置をリセットするには、ツールバーの Reset Pose ボタンを押します。

スプライトジオメトリ

スプライトをボーンに割り当てるには、ジオメトリを作成する必要があります。Auto Geometry ボタンを押して開始します。小さなポップアップウィンドウが開き、ジオメトリの作成方法を定義できます。

ジオメトリをできるだけシンプルに保つために、すべてのスライダーをゼロに設定するのが良いでしょう。Weights オプションを有効にして、ボーンをスプライトに自動的にバインドします。Generate For All Visible ボタンをクリックすると、すべてのスプライトのボーンウェイトが作成および設定されます。個々のスプライトに対してこれを行うには、スプライトをダブルクリックします。これは、特定のスプライトのジオメトリの微調整に便利です。



スプライトのジオメトリの自動生成

生成されたジオメトリより詳細に、頂点の数やジオメトリの曲がり方を完全に制御するには、メッシュを手動で編集する必要があります。以下のツールを使用してください。

— **Create Vertex:** 新しい頂点 (または点) を作成するか、既存の点を移動または削除します。

— **Create Edge:** 既存の 2 つの点の間に辺を作成するか、左マウスボタンを使用して新しい点を作成します。右マウスボタンで現在選択されている点の選択を解除し、他の頂点間に辺を作成できます。

— **Edit Geometry:** 頂点を移動します。

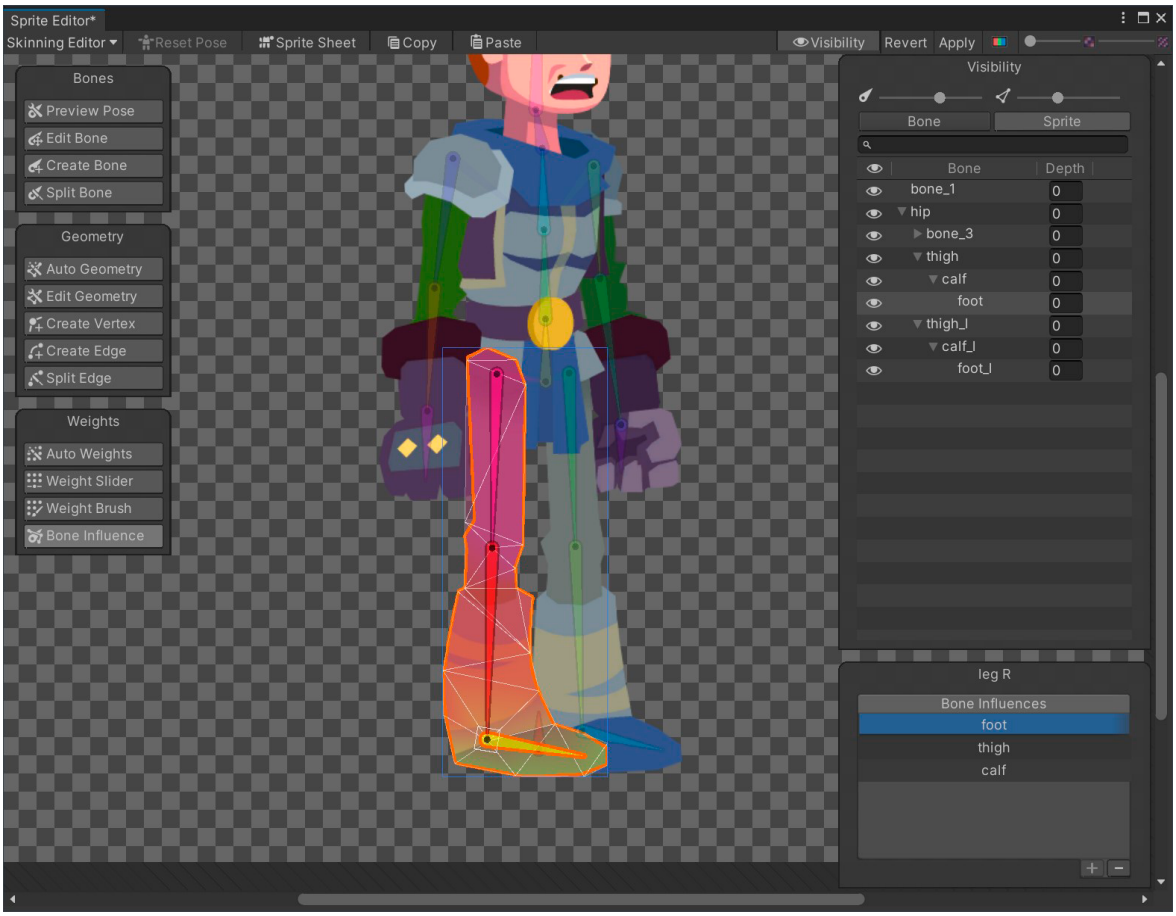
— **Split Edge:** 辺の中央に頂点を作成して分割します。

可能な限り、使用する頂点を少なくしてください。各頂点の位置はボーンの回転に基づいて計算される必要があるため、頂点の数が少ないほどパフォーマンスを効率化できます。また、頂点数が少ないほど、少ない点にウェイトを設定しやすくなるため、曲がったメッシュの見栄えがより良くなります。さらに、各ターゲットプラットフォームには理想的なゲーム頂点数があるので、ジオメトリを最適化するのが良いでしょう。

ウェイト

スプライトのジオメトリがきれいに整ったら、次は ウェイトの設定です。ウェイトでは、ボーンが各頂点に与える影響を 0 から 1 の範囲で定義します。0 はボーンが頂点に何も影響しないことを意味し、1 は頂点がボーンに接着されているかのように動くことを意味します。メッシュのウェイトをうまく調整することで、見栄えの良い曲がり方を可能にします。ウェイトの設定を誤ると、ゲームの視覚効果が崩れたり、スプライトが歪んだりします。

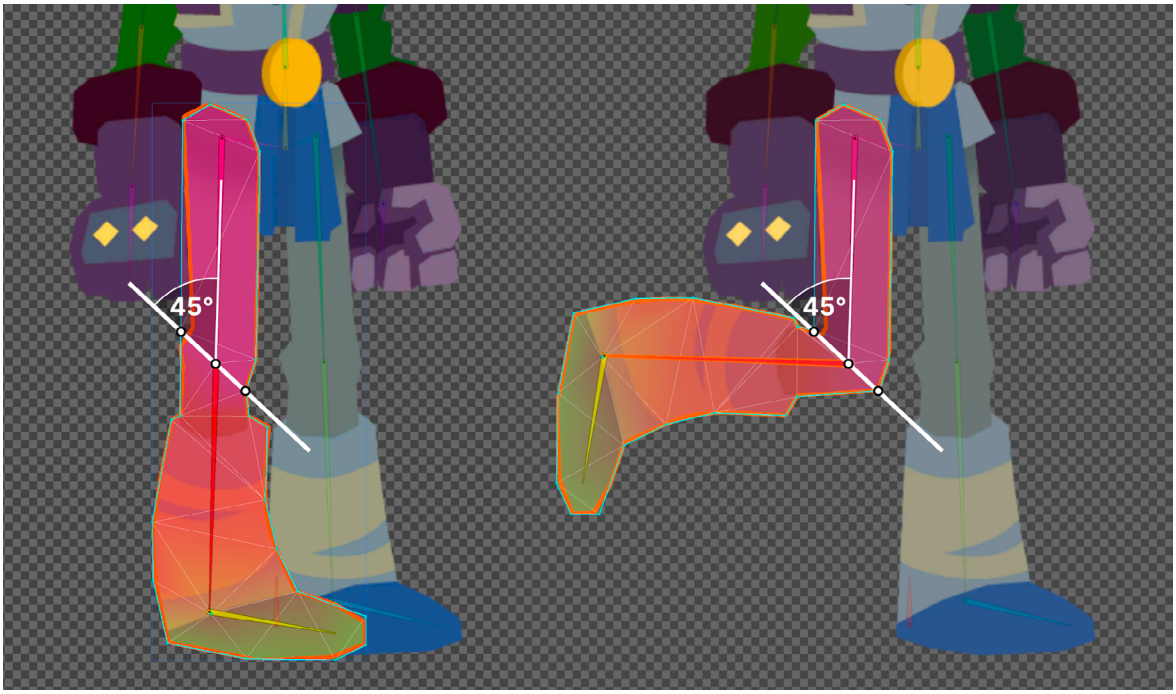
ウェイトの設定を始めるには、特定のスプライトに影響するボーンを定義する必要があります。Bone Influence ボタンをクリックし、スプライトを選択します。



スプライトに Bone Influences を設定

小さなポップアップウィンドウから、どのボーンが選択したスプライトに影響を与えるかを設定できます。スプライトに影響を与える必要のある関連するボーンだけを設定し、残りを削除します。

ここから、**Weight Brush** と **Weight Slider** を使用してウェイトを設定します。Weight Brush は、選択したボーンの影響を頂点にすばやく追加するためのツールで、マウスで頂点を塗るように操作します。Weight Slider はより正確な設定が可能で、1 つまたは複数の点を選択し、スライダーで各ボーンの正確な影響を決定できます。Weight Brush は迅速なウェイト設定に便利で、Weight Slider は肘や膝のようにボーンが曲がる部分の微調整に便利です。



手足の曲がり方の見栄えを最もよくする頂点の配置方法

ヒント: 肘や膝を設定する場合、2 本のボーンが交わる曲がりの中心を通る線に、内側と外側の頂点を揃えることで、最も自然に見える結果が得られます。この線は、ボーンを 45 度の角度で横切るように引く必要があります。これらの頂点は、上側のボーンの影響を受けます。ただし、キャラクターはそれぞれ異なるため、ウェイトを自由に変えて調整し、最良の結果を出してください。

作業を効率化するために、Dragon Crashers プロジェクトを開き、Prefabs および Prefab Variants のセクションで以下のキャラクタープレハブを見つけてください。

- Prefab_Character_Base
  - PV_Character_Witch (自動再バインドのために異なるスプライトライブラリを使用)
  - PV_Character_Knight
  - PV_Character_Wolfman
  - PV_Character_Skeleton

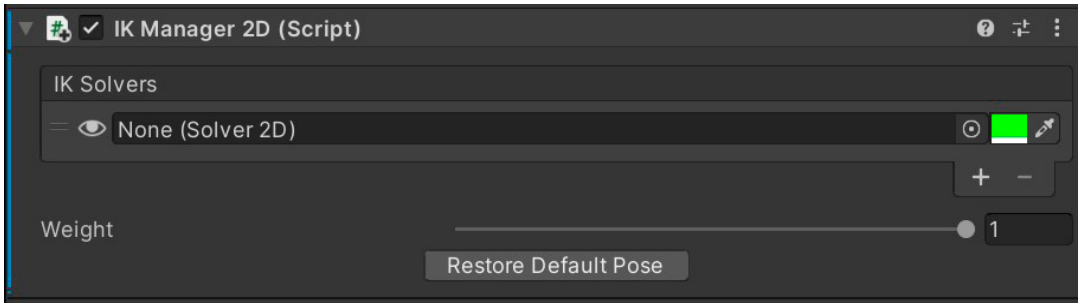
これらのプレハブは、同じ構造のキャラクター間で共有できるアニメーションのよい例です。

2D インバースキネマティクス

たまにしか意識しませんが、人間の体の動きは非常に複雑です。テーブルから水のグラスを取る場合、手はグラスが占める空間の点に移動しなければなりません。脳がこれらの計算をバックグラウンドで処理しているため、腕と前腕を回転させることをわざわざ考えなくてもすべてを実行します。

ゲームで同じ動きをアニメーション化するには、腕と前腕の両方の回転を同時にアニメーション化する必要があります。自然な手の動きを作りながら、両方の回転を一致させるのは難しい作業です。2D インバースキネマティクス (IK) ツールは、2D Animation パッケージの一部であり、ボーンの回転を計算し、ボーンのチェーンをターゲット位置に移動させることができます。

まず、階層の一番上にあるオブジェクトに IK Manager 2D を追加する必要があります。このコンポーネントは、キャラクターのすべての IK ソルバー を管理する役割を担います。



IK Manager 2D コンポーネント

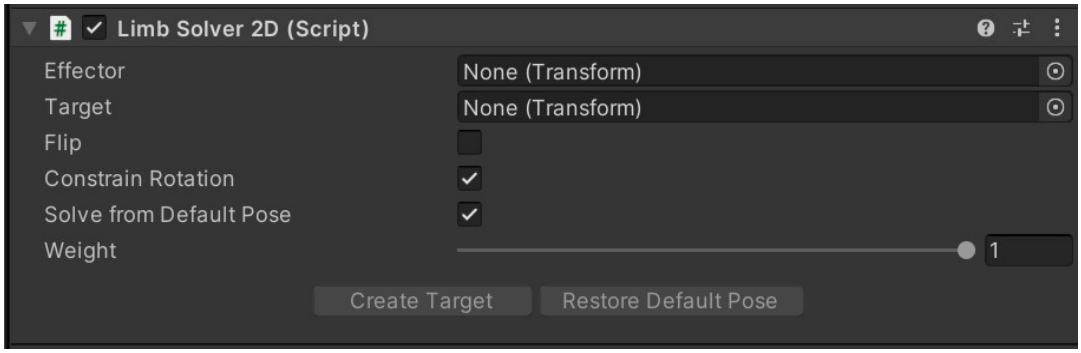
+ ボタンをクリックすると、新しいソルバーが追加されます。ソルバーは、対象の Transform に一致するようにボーンの回転を計算します。利用可能なソルバーは以下の 3 つです。

— **Limb:** これは、脚と腕に使用される標準的なソルバーで、最大 2 つのボーンとエフェクターを解決できます。

— **Chain (CCD) – Cyclic Coordinate Descent (循環座標降下法):** このソルバーは、アルゴリズムを実行する回数が増えるほど精度が向上し、より長いボーンのチェーンにも適しています。

— **Chain (FABRIK) – Forward And Backward Reaching Inverse Kinematics (前方後方到達インバースキネマティクス):** Chain (CCD) と同様に、この解決策はアルゴリズムが実行される回数が増えるにつれて、より正確になります。ボーンがリアルタイムで異なる位置に操作されると、迅速に適応します。

ヒューマノイドキャラクターでは、Limb ソルバーが最適な選択肢です。なぜなら、最も速く 2 つのボーンの手足に最適化されるからです。IK Manager 2D のリストに Limb ソルバーを追加することから始めます。これにより、Limb ソルバー 2D コンポーネントを含む新しいゲームオブジェクトが作成されます。このゲームオブジェクトに “Leg R LimbSolver2D” や “Arm L LimbSolver2D” のように説明的な名前を付けます。



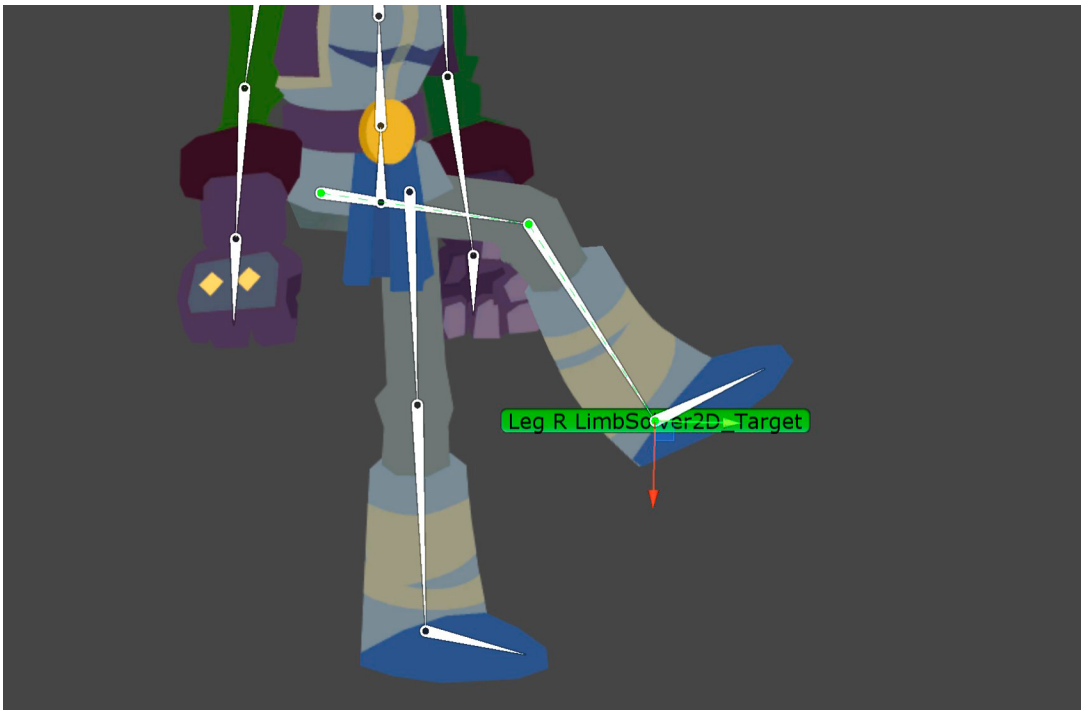
Limb Solver 2D コンポーネント

動作するには、ソルバーには Effector と Target という 2 つのゲームオブジェクトの Transform が必要です。Effector は、チェーンの最後のボーンの中に配置されます。例えば、指の先端のボーンです。そこから、Target の位置に到達しようとします。

まず、Effector を作成します。脚の IK を作成する場合には、太ももとふくらはぎ (またはすね) という 2 つのボーンがあります。ふくらはぎのボーンの子として Effector を配置するには、それを選択し、新しい空白のゲームオブジェクトを作成します。ゲームオブジェクトの名前を “Leg Effector” に変更し、Effector をボーンの先端に移動させます。

新しく作成したエフェクターを Limb Solver 2D の Effector フィールドに追加し、Create Target ボタンを押します。ターゲットゲームオブジェクトが Limb Solver オブジェクトの中に作成されます。

これで、Target オブジェクトの動きに脚が従うようになります。



2D IK の動き - ターゲットゲームオブジェクトを動かすと、太ももとふくらはぎの回転が自動的に計算され、ターゲットの位置に一致します。

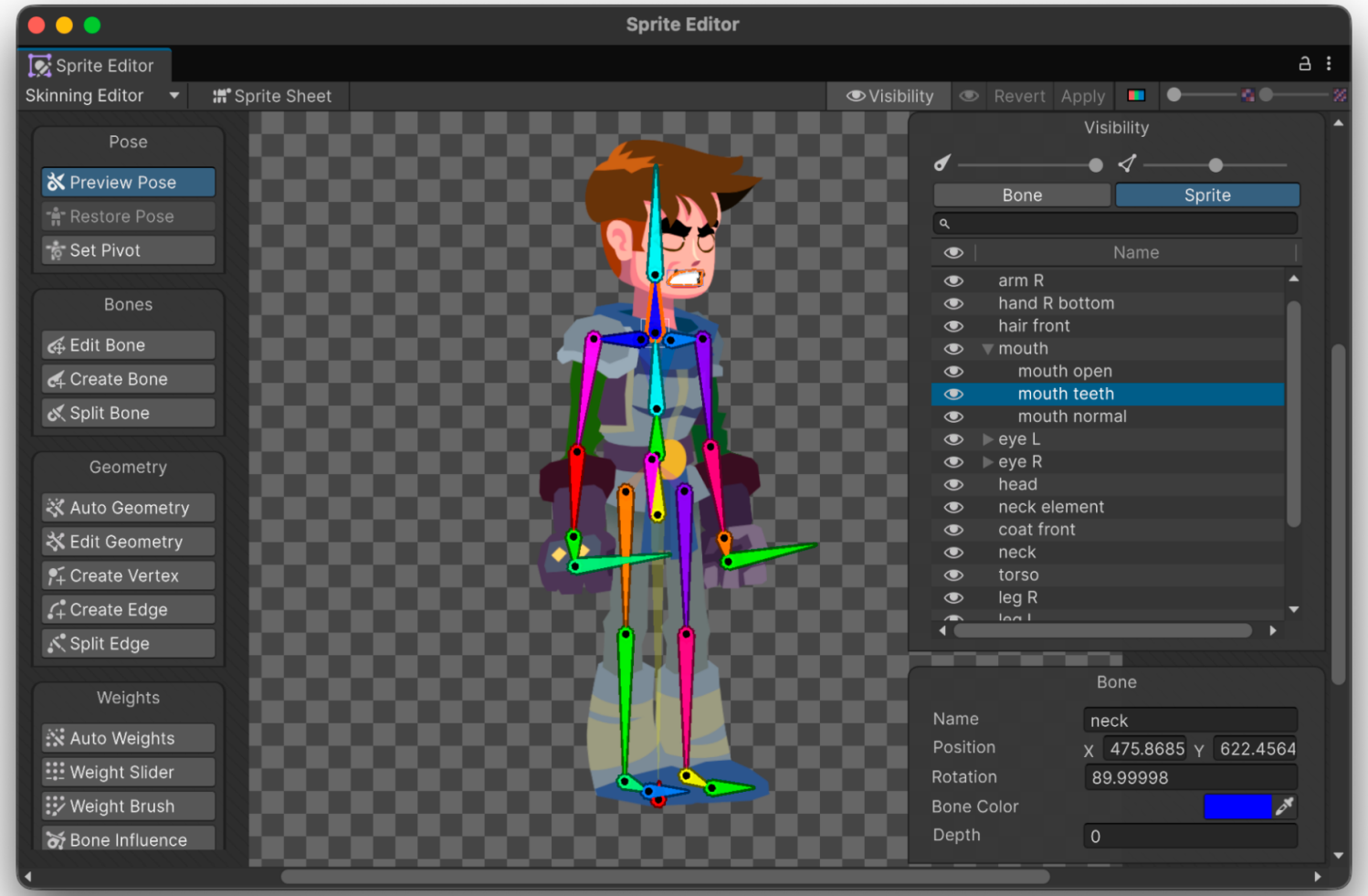
他の手足に IK を追加するプロセスを繰り返します。IKは、頭や首などの手足以外のボーンにも追加できます。これにより、キャラクターが周りの物を見まわすことができます。

これを習得すると、さらに上級者向けのユースケースも可能になります。例えば、キャラクターが銃を構えるように IK を設定したり、プロシージャルな歩行アニメーションを作成したりできます。

スプライトスワップとスキン

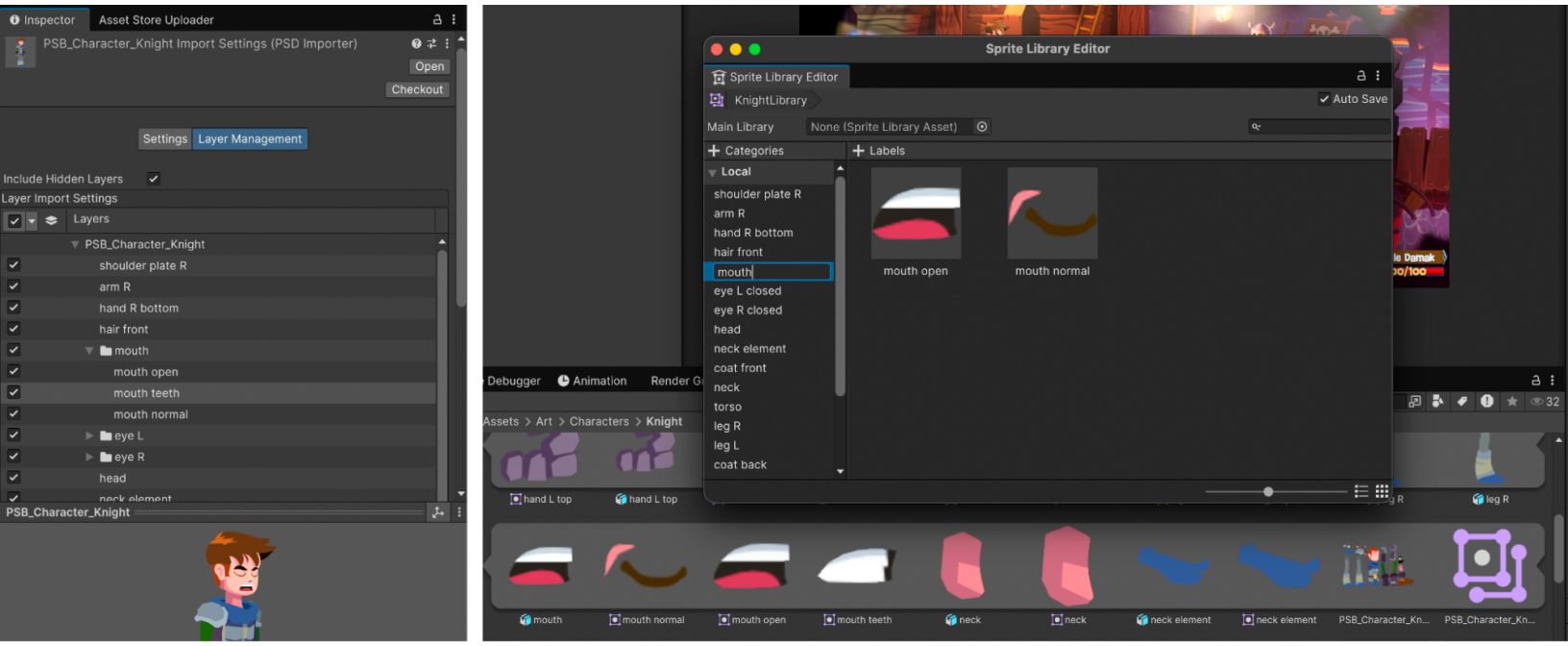
ボーンを回転しても、すべてのものをアニメーション化できるわけではありません。時には、その他にも顔の表情や手のポーズが必要です。この場合、スプライトスワップを使用して、1 つのスプライトを別のものと交換できます。

以前は、スプライトスワップは Skinning エディターから各キャラクターのスプライトにカテゴリとラベルを割り当てることで機能していました。Unity 6 では、カテゴリとラベルがスプライトライブラリの一部になりました。これは、47 ページ のスプライトスワップのセクションで説明されています。



Skinning エディターで Visibility 設定を有効にすると、キャラクターのスプライトとボーンを表示することができます。

Dragon Crashers の騎士キャラクターの 口 のスプライトを見てみましょう。口のカテゴリのバリエーションは、元の PSD ファイルまたは PSB ファイルのグループ化と一致する子要素であることがわかります。



(左) 2D PSD Importer によるインスペクター内の PSD ファイルのレイヤーが表示されています。(右) 騎士キャラクターを構成するスプライトのために Sprite Library が作成されています。

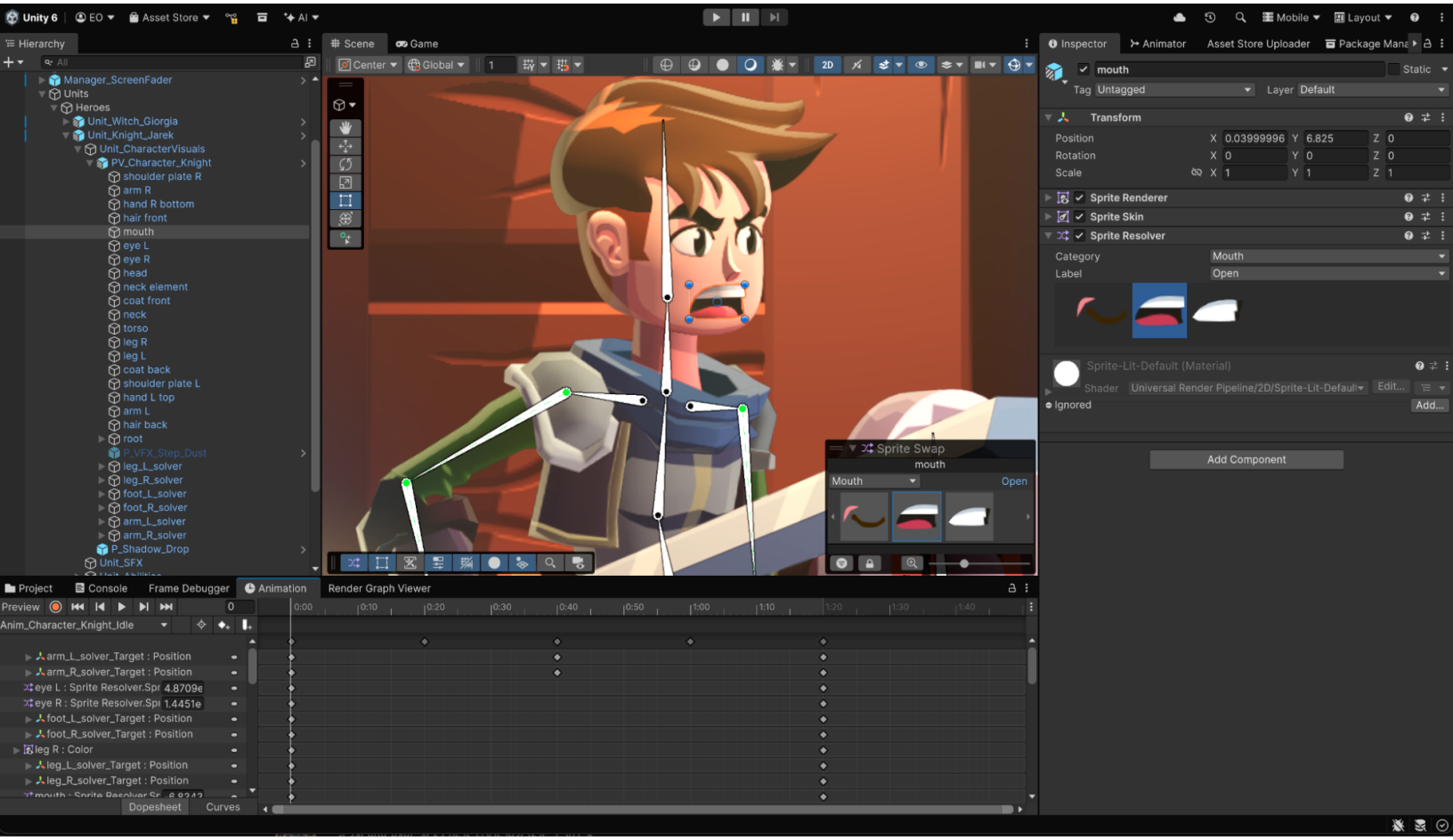
作成メニューから新しいスプライトライブラリを作成できます。キャラクターに関連する名前を付けてください。PSD ファイルを Sprite Library ウィンドウのカテゴリ列にドラッグアンドドロップします。

カテゴリは、PSD ファイル内の可視スプライトに基づいて自動的に作成されます。

カテゴリ名を変更する必要があるかもしれません。騎士キャラクターの例では、口のバリエーションのカテゴリは口と名付けられています。

PSD ファイル内の隠れたレイヤーだったため、すべての口のバリエーションが加えられたわけではありませんが、口のカテゴリに口のバリエーションをドラッグアンドドロップして追加できます。口の各バリエーションには、ラベルまたは名前が付けられます。

スワップされる予定のスプライトがスプライトリゾルバーに加えられ、クリックすることでスプライトスワップを実装できます。例えば、アニメーション化の際にリゾルバーを使ってスプライトを簡単に変更できます。



Dragon Crashers の騎士キャラクターの口のスプライトのスワップ

ここで、キャラクターの口を選択します。スプライトリゾルバーでカテゴリが選択されていない場合は、1 つ選択すると簡単にスワップできるバリエーションが表示されます。スプライトのスワップは、オーバーレイメニューからも可能です。

スプライトスワップを使用して、顔の表情、目や口のアニメーション化、リップシンクアニメーションの作成、手のジェスチャの変更を簡単に行うことができます。スプライトスワップによって、帽子や鎧などのキャラクター装備をスワップすることも可能です。

Sprite Swap オーバーレイ

2D アニメーションキャラクターは、カテゴリとラベルを備えたスプライトライブラリを利用することで、インスペクターやシーンのオーバーレイからスプライトを簡単に交換できます。

Unity 6 では、新しい**Sprite Swap オーバーレイ**がシーンビューで利用できるようになりました。このオーバーレイは、選択されたスプライトがスワップできるすべての関連ラベルを表示します。

階層のスワップ可能なすべての部分に対応できるようになります。マルチパーツキャラクターの任意の部分が選択されると、スワップ可能なスプライトを備えたすべての子オブジェクトがオーバーレイに一緒に表示されます。アニメーションウィンドウと組み合わせて使用することで、スプライトスワップアニメーションのワークフローをより効率的にすることができます。

スキン

Sprite Library アセットを使用して、キャラクターのスキンを作成できます。スキンは、アニメーションを保持しつつキャラクターの外観を変更するため、時間を大幅に節約できます。1 つの基本キャラクターにはすべてのスクリプトが含まれ、変更が行われると他のキャラクターにも適用されます。

以下は Dragon Carshers と Happy Harvest のサンプルのキャラクターにスキンサポートを設定するためのワークフローです。

1. 基本キャラクターを完成させます。リグを設定し、IK と Sprite Library を設定します。キャラクターからプレハブを作成します。任意の時点でキャラクターをアニメーション化できます。

2. 基本キャラクターの Skinning エディターに移動し、ツールバーの **Copy** ボタンをクリックします。これにより、ボーンとメッシュがウェイトと共にコピーされます。

3. このキャラクターのプレハブバリアントを作成します。これが新しい外観の新しいキャラクターになります。

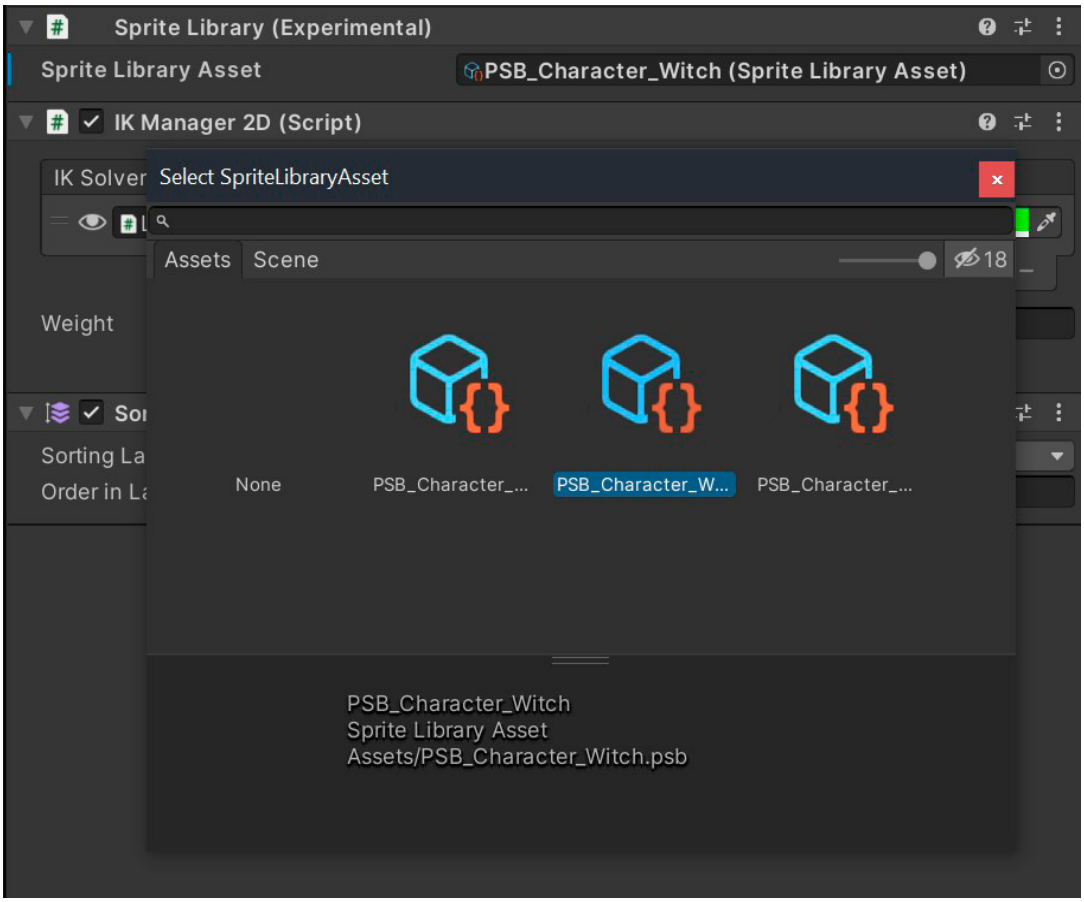
4. 新しいキャラクターの PSD ファイルまたは PSB ファイルをインポートし、Skinning エディターを開きます。

5. ツールバーの **Paste** ボタンをクリックして、ステップ 2 からコピーしたスケルトンを貼り付けます。ポップアップウィンドウで **Bones** と **Mesh** オプションが選択されていることを確認し、Paste ボタンをクリックして確認します。

6. ジオメトリを修正して、新しいスプライトに合わせます。ウェイトを再度確認します。

7. ステップ 3 で作成されたプレハブバリアントに移動します。Sprite Library コンポーネントに移動し、新しいキャラクターのために Sprite Library アセットをスワップします。

8. 新しいキャラクターを作成するには、ステップ 2 から 8 を実行します。



Sprite Library コンポーネントで Sprite Library Asset を選択します

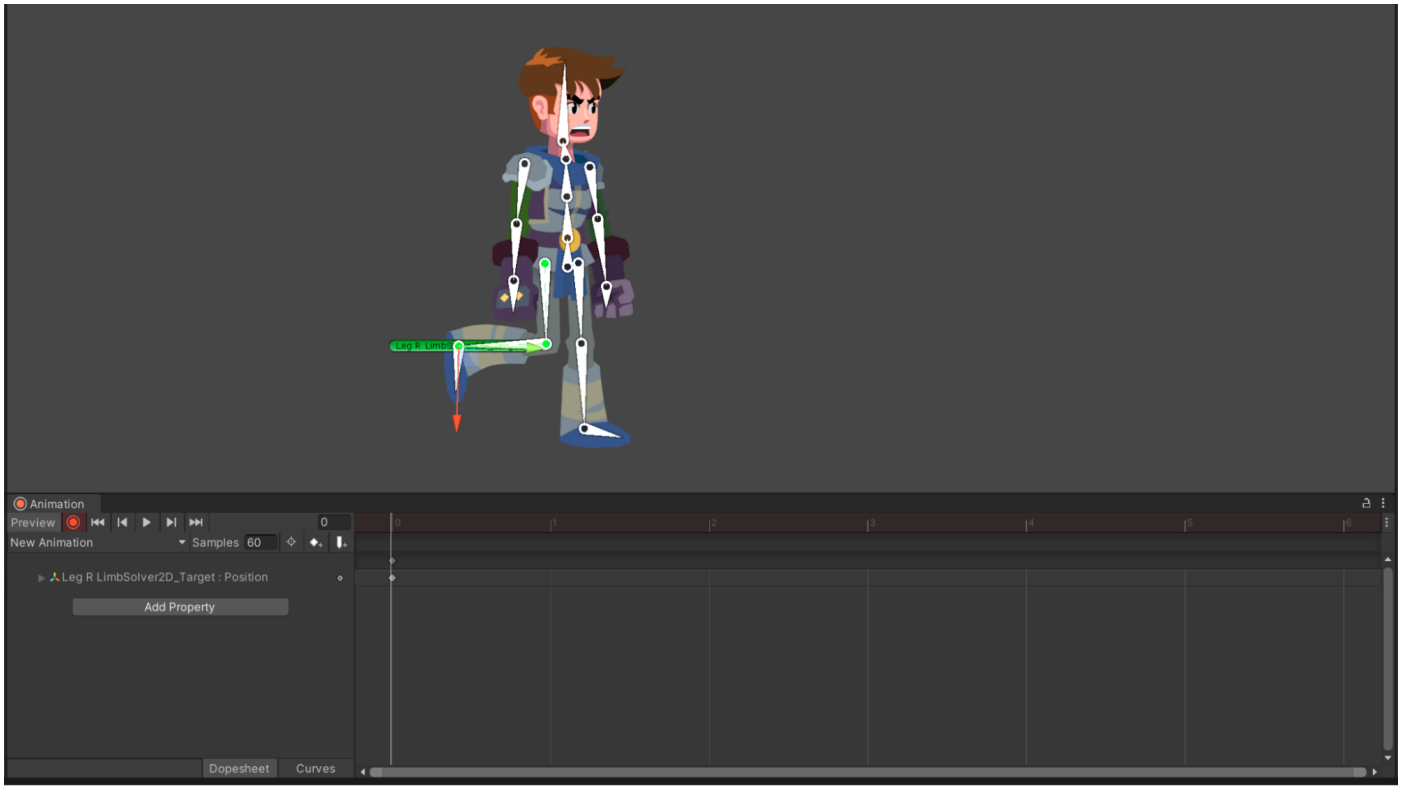
このワークフローに従うと、基本キャラクターのプレハブが 1 つできます。他のキャラクターはこのプレハブのバリアントになるため、基本プレハブと同じコンポーネントを備えています。基本キャラクターに変更が加えられた場合 (例えば IK やソートグループの追加や変更があった場合)、他のキャラクターはその変更を継承します。

アニメーションの基本

説得力のあるキャラクターを作成したい場合は、アニメーションが非常に重要です。

素晴らしいアニメーションを作成するには、アニメーションの原則とそれに適用するためのツールについて学ぶ必要があります。基本的なアニメーションの原則を学ぶことはしばしば軽視されますが、キャラクターの動きをよくするためにここで少し時間をかけること、ゲームに大きな利益をもたらします。

最初のアニメーションを作成ために、**Window > Animation > Animation の順に移動して Animation ウィンドウを開きます。**



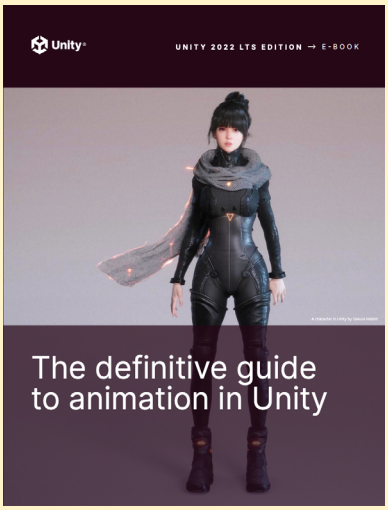
Animation ウィンドウ - アニメーション化の準備が整っているキャラクターが表示されています

キャラクターを選択します。**Create** ボタンをクリックし、最初の アニメーションクリップ を作成して、クリップに名前を付けることから始めましょう。アニメーションクリップは、オブジェクトの位置、回転、スケール、その他のプロパティが時間とともにどのように変化するかを記した直線的な記録のようなものです。また、すべてのアニメーションクリップを管理し、どのクリップが再生されるべきか、アニメーションがいつ変更またはブレンドされるべきかを追跡する アニメーターコントローラー も作成されます。

赤い **Record** ボタンをクリックしてアニメーション化を開始します。これで、キャラクターに加えたすべての変更がアニメーションクリップに記録されます。ボーンや IK ソルバーを動かしたり、スプライトリゾルバーでスプライトをスワップしたりして、アニメーションクリップに満足するまで再生できます。

アニメーションは、使用するツールの技術的理解を必要とするアートです。この e-book に加えて、Unity にはアニメーションに特化した詳細なガイドがあり、3D か 2D かにかかわらずキャラクターを本当に生き生きとさせるのに役立ちます。以下のリンクを参照し、Unity Asset Store の 2D Animation Sample パッケージ をチェックしてください。

Unity での 3D および 2D アニメーションの詳細について、これらのリソースを参照してください。



E-book: Unity でのアニメーション制作の完全ガイド

Unity での Humanoid アニメーションの扱い方

2D 物理演算

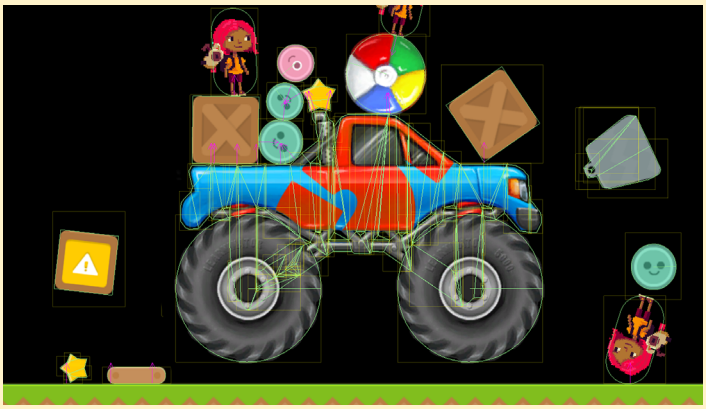
本書は主に、Unity を初めて使うアーティストやテクニカルアーティストを対象としており、視覚的な忠実度を高めるために利用できる 2D ツールを最大限に活用する方法を解説しています。

ただし、小規模なチームであっても大規模なチームであっても、ゲームプレイロジックの一部として 2D 物理演算を扱うことになる可能性は高いでしょう。では、Unity 6.3 における 2D 物理演算 の基本を見てみましょう。

3D プロジェクトの経験がある方にとっては、3D 物理演算と 2D 物理演算は実装において類似点があるものの、全く異なるシステムであるということに気を付けてください。3D オブジェクトの物理演算は、2D オブジェクトの物理演算と相互作用することはありません。

2D 物理演算を使用することで、システムが物理法則に現実的に (または非現実的にでも) 反応するような、生き生きとした世界を作ることが可能になります。キャラクター、車両、オブジェクト、チェーン、プラットフォームは、固体の地形、水、風の力、または構築したいものを含む環境と相互作用します。

2D 物理演算のすべての可能性を把握するために、サンドボックス 2D 物理サンプルを取得して、すべてのシステムが動作していることを確認することをお勧めします。物理システムなど、Unity でのゲームデザイン全般についてもっと学びたい場合は、ゲームデザインの e-book を無料で入手できます。この e-book は古い Unity バージョンに基づいていますが、その内容のほとんどは Unity 6.3 でも適用可能です。



2D 物理サンプル

E-book: The Unity game designer playbook (Unity ゲームデザイナープレイブック)

コライダー:　オブジェクトの相互作用可能な領域

物理世界で相互作用するためにオブジェクトの領域を定義する必要があります。Unity での 2D 開発では、ゲームオブジェクトに付属している 2D Collider コンポーネントを介してこの領域を加えます。形状がシンプルであればあるほど、パフォーマンスが向上します。以下の Collider コンポーネントが利用可能です。

Edge Collider 2D	—	スプライトの形状 やその他の形状に合わせて調整できる、線分で構成された辺のコライダー
Box Collider 2D	—	スプライトのローカル座標空間における定義された位置、幅、高さを備えた矩形
Circle Collider 2D	—	スプライトのローカル座標空間における定義された位置と半径を備えた円形コライダー
Capsule Collider 2D	—	滑らかで丸みのある外周を備え、頂点の角がないカプセル型コライダー。他のコライダーに沿った滑らかな動きを可能にし、鋭い角での引っかかりを防ぎます。
Polygon Collider 2D	—	物理演算によって複数の凸状のポリゴン形状を生成する領域を定義する、凹/凸のアウトライン
Custom Collider 2D	—	PhysicsShapeGroup2D API を介して PhysicsShape2D ジオメトリを割り当てると、コライダーの形状が設定されます。
Tilemap Collider 2D	—	同じゲームオブジェクト上の Tilemap コンポーネントのタイルに対してコライダー形状を生成します。Tilemap コンポーネントのタイルを追加または削除すると、Tilemap Collider 2D により LateUpdate 時にコライダー形状が更新されます。
Composite Collider 2D	—	固有の形状は持っていませんが、代わりに、使用するように設定された任意の Box Collider 2D、Polygon Collider 2D、Circle Collider 2D、または Capsule Collider 2D の形状を結合して使用します。

リジッドボディと他のリジッドボディやコライダー間の衝突を検出する場合、API OnCollisionEnter2D、OnCollisionStay2D、OnCollisionExit2D を使用すると、衝突イベントが発生したときに検出できます。

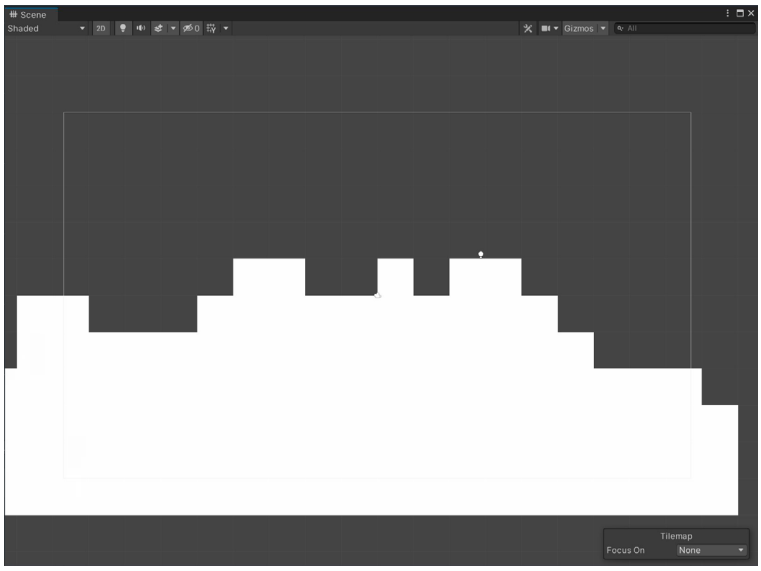
Is Trigger

Collider 2D のボックス **Is Trigger** にチェックを入れることで、物理的な動きを生じることなくこのコライダーで他のコライダーを検出します (オブジェクトに Rigidbody 2D コンポーネントが付属している場合は Rigidbody 2D コンポーネントを無視します)。これをセンサーとして考えてください。例えば、ステルスゲームでは、プレイヤーの存在を特定のエリアで検出するために、敵に **IsTrigger** を有効にした Collider 2D を設定することができます。コライダーが存在を検出すると、プレイヤーの追跡を開始できます。

その場合、C# で、OnTriggerEnter2D、OnTriggerStay2D、または OnTriggerExit2D イベントを使用して、トリガーとの衝突が発生したときのロジックを書くことができます。

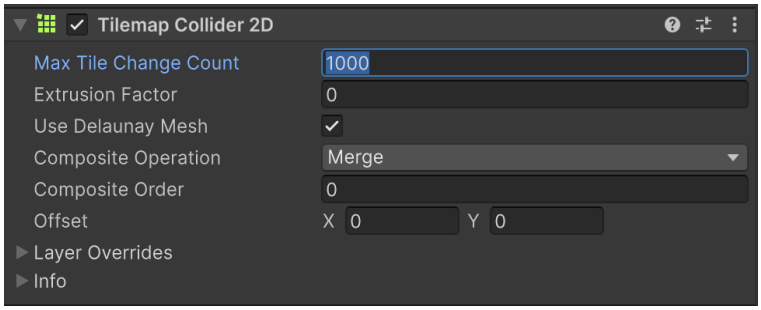
Tilemap Collider 2D の作成

長方形のタイルマップ用に単純な正方形のタイルで構成されたタイルマップの地面を作成することで、2D の物理演算を素早くテストできます。



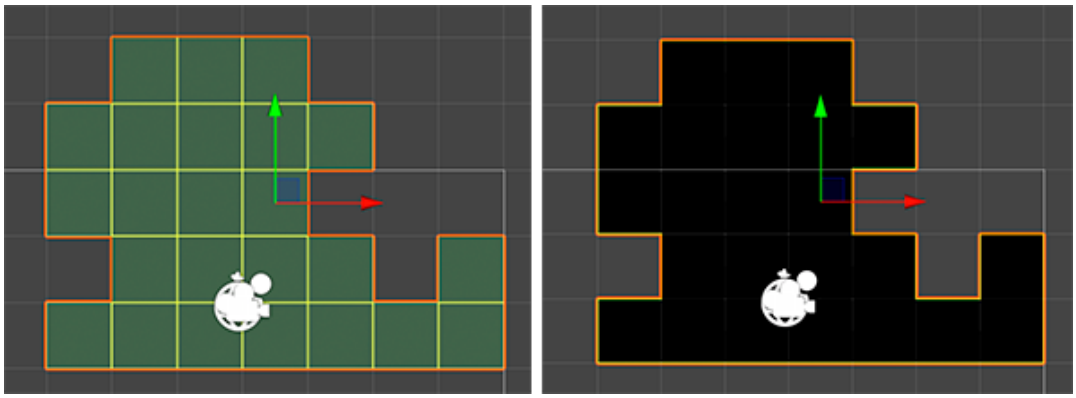
ペイントされたプレースホルダータイルを使用したタイルマップ

次に、物理演算のために　Inspector ウィンドウで Tilemapに Tilemap Collider 2D コンポーネントを追加します。これにより、タイルアセットで設定されたコライダータイプに基づいて、各タイルにコライダーが追加されます。



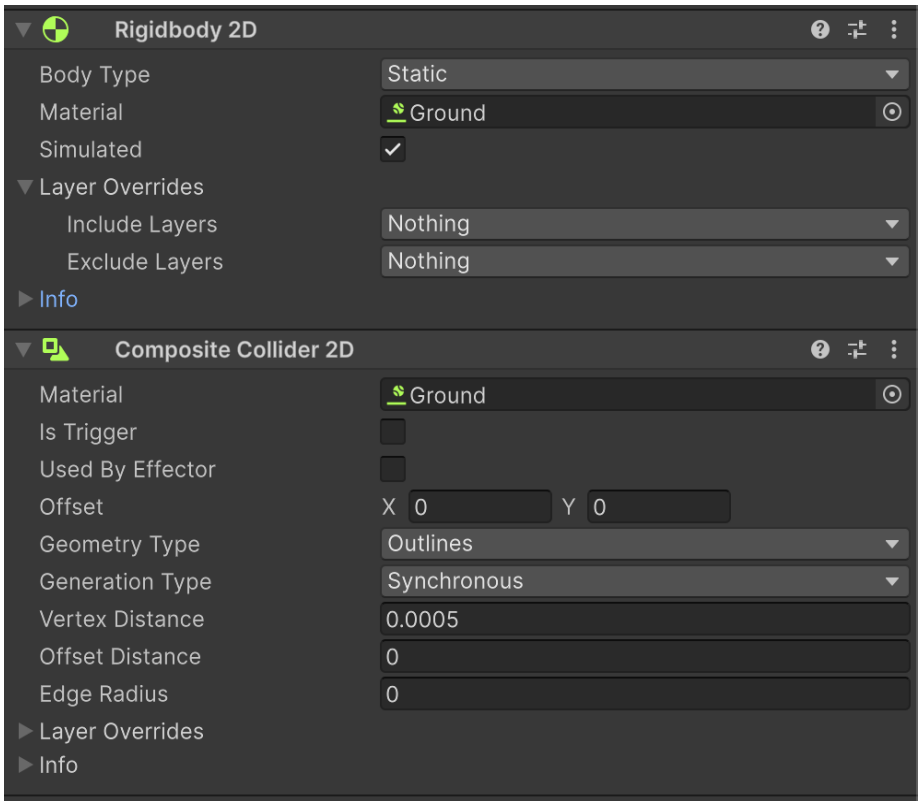
この例では、オプション **Use Delaunay Mesh** が有効になっています。これにより、コライダーメッシュ内で作成される形状の数と生成される小さな三角形の数が減少し、全体的な物理パフォーマンスが向上します。

同じゲームオブジェクトまたは親グリッドに Composite Collider 2D コンポーネントを追加して、コライダーを 1 つにまとめると、アウトラインとして設定されたジオメトリとの衝突動作が滑らかになります。タイルは個々のタイルとしてではなく、1 つの連続した地形のように動作します。この設定により、パフォーマンスも若干向上します。しかし、ランタイムにタイルを追加したり削除したりすることを計画している場合は、各タイルでコライダーを維持したほうがよいでしょう。



タイルをグループ化して、Composite Collider 2D とマージ操作を使用して 1 つの表面を形成できます。左の画像ではタイルが結合されていませんが、右の画像では結合されています。

Tilemap Collider 2D コンポーネントの Composite Operation 設定を Merge に設定し、自動的に追加された Rigidbody 2D タイプを Static (静的) に設定して、地面となるタイルマップが落下しないようにしてください。

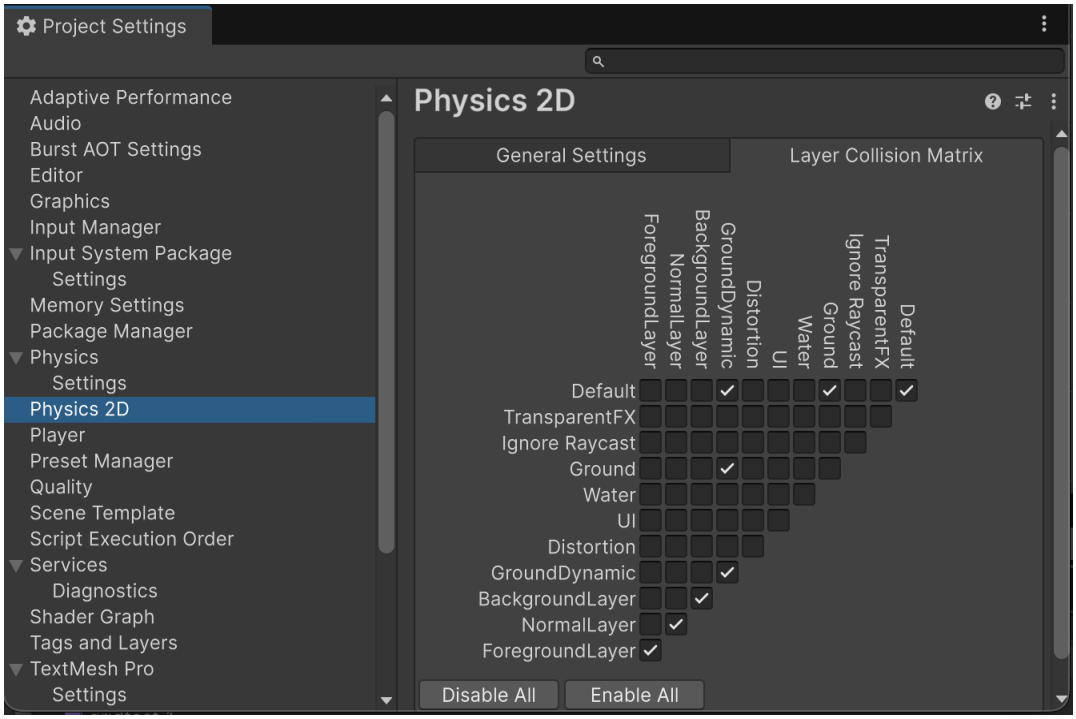


タイルマップゲームオブジェクトの Collider 2D コンポーネント

衝突マトリックス

Physics 2D によってゲームオブジェクトのレイヤーが使用され、オブジェクト同士が相互作用できるようになります。すべてのものがすべてのものと相互作用する必要はありません。例えば、プラットフォーマーゲームでは、敵が地面と衝突することは必要ですが、必ずしも他の敵と同時に衝突する必要はありません。その場合、その場合、敵が地面とは衝突する一方で、敵のレイヤー同士では衝突しないように設定を確認します。

これらの設定は **Project Settings > Physics 2D > Layer Collision Matrix にあります。**



Unity 6 の 2D Collider では、Inspector ウィンドウの Rigidbody 2D コンポーネントから、オブジェクトごとに特定の物理レイヤーを無視または考慮することを可能にします。

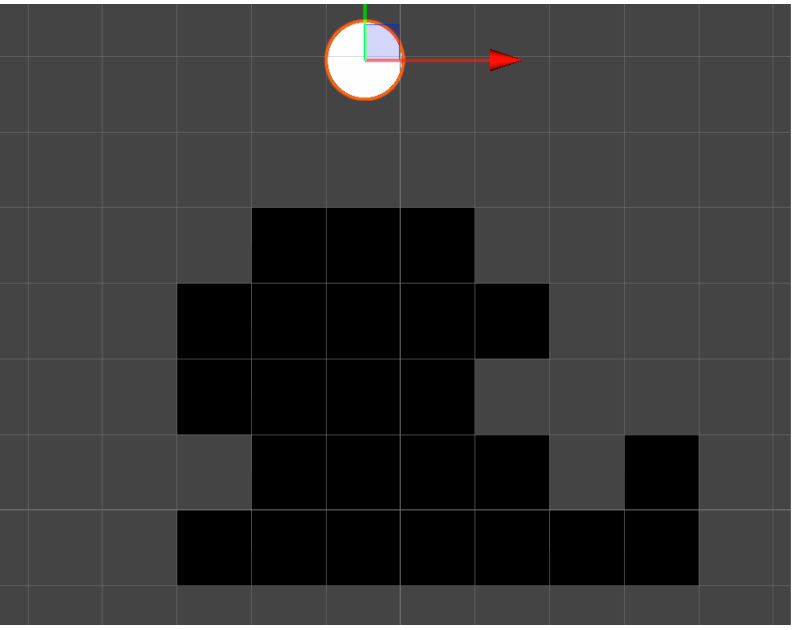
Rigidbody: 物理演算でオブジェクトを動かす

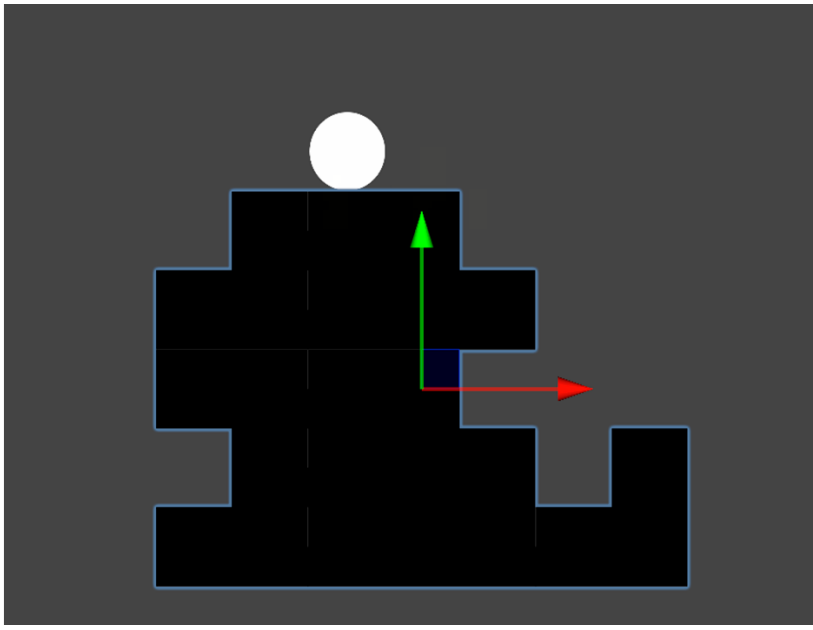
オブジェクトには定義された Collider 2D エリアがあるので、1 つ以上の Rigidbody 2D コンポーネントと物理的に作用させることができます。

Rigidbody は以下のように分類できます。

- **静的:** このオブジェクトは動かないものの、他のオブジェクトはこれに対して反応します。例えば、地面のオブジェクトは静的なリジッドボディになります。
- **キネマティック:** 他の衝突の影響を受けずに移動します。例えば、移動プラットフォームは、その上にオブジェクトがいくつあっても同じように移動します。
- **動的:** 質量、速度、力などの物理特性に基づいて移動し、接触している他のリジッドボディの影響を受けます。

前のタイルマップシーンを使用してこれをすぐにテストしましょう。**GameObject > 2D Object > Physics > Dynamic Sprite** の順に選択してシーンに新しいオブジェクトを作成します。この円形のオブジェクトをタイルマップの上に置き、再生モードに入ります。





動的なオブジェクトが静的なタイルマップの表面でリアルに跳ね返る様子

コードで Rigidbody 2Dを 動かす

2D 物理オブジェクトは、シーンにインスタンス化されると実際の物理オブジェクトのように動作しますが、たいてい、移動するプラットフォームや異なる方向に移動したりジャンプしたりするキャラクターが必要になります。

物理演算によって動くオブジェクトは、位置を変更するために Rigidbody2D APIs を使用する必要があります。GameObject Transform でそれらのオブジェクトを移動することは避けてください。代わりに、以下のように関数を使用してください。

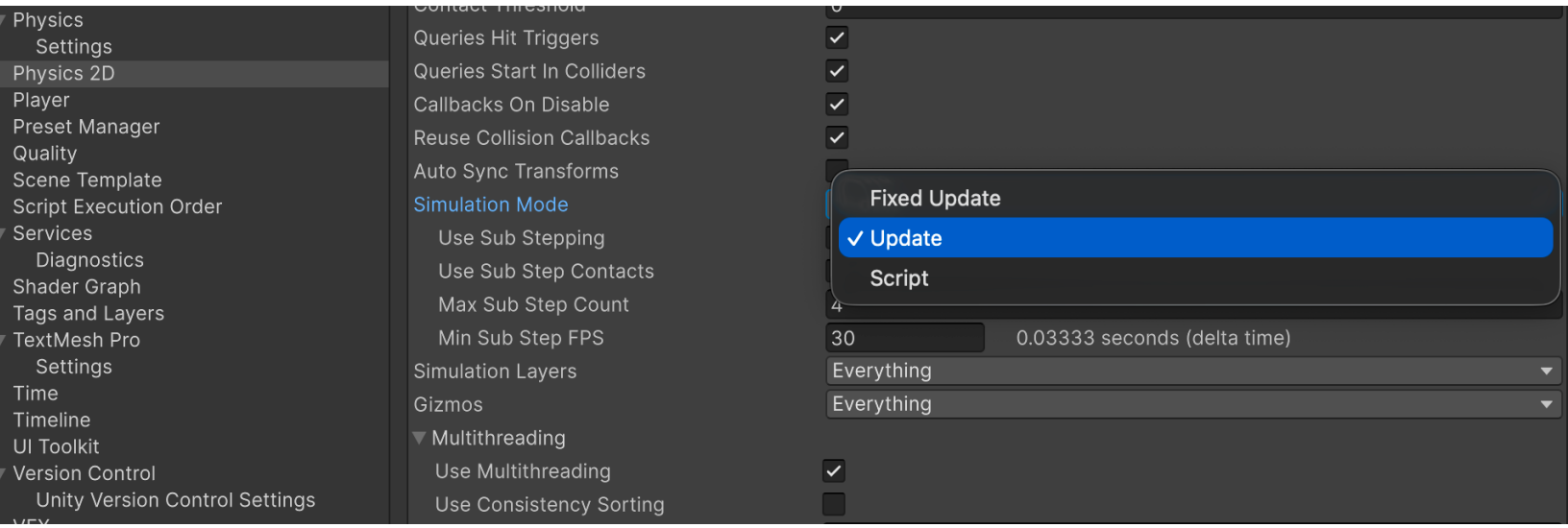
- AddForce または Slide メソッドを使用してキャラクターの動きを制御します。
- Impulse モードで AddForce メソッドを使用してオブジェクトを推進したり、キャラクターをジャンプさせたりします。
- 回転するプラットフォームのように、運動し続けるキネマティックリジッドボディを MoveRotation または MovePosition で動かします。
- position プロパティを変更することで、リジッドボディをある場所から別の場所にテレポートさせます。
- キャラクターが移動する速度をlinearVelocityプロパティを使用して評価します。

コードでは、Update または FixedUpdate サイクル でリジッドボディメソッドを使用する必要があります。プロジェクト設定では、一方からもう他方に変更できます。

— Update は毎フレーム発生し、より滑らかで正確なシミュレーションになります。

— FixedUpdate は、デフォルトで 0.02 秒ごとに設定された間隔で発生します。

どちらの場合でも、一定の動きを得るために値を Time.deltaTime で乗算することを忘れないでください。Time.deltaTime は、1 フレームを処理するのにかかる時間 (秒) です。これを使って運動関連の値を乗算すると、シミュレーションで一定の結果が得られます。



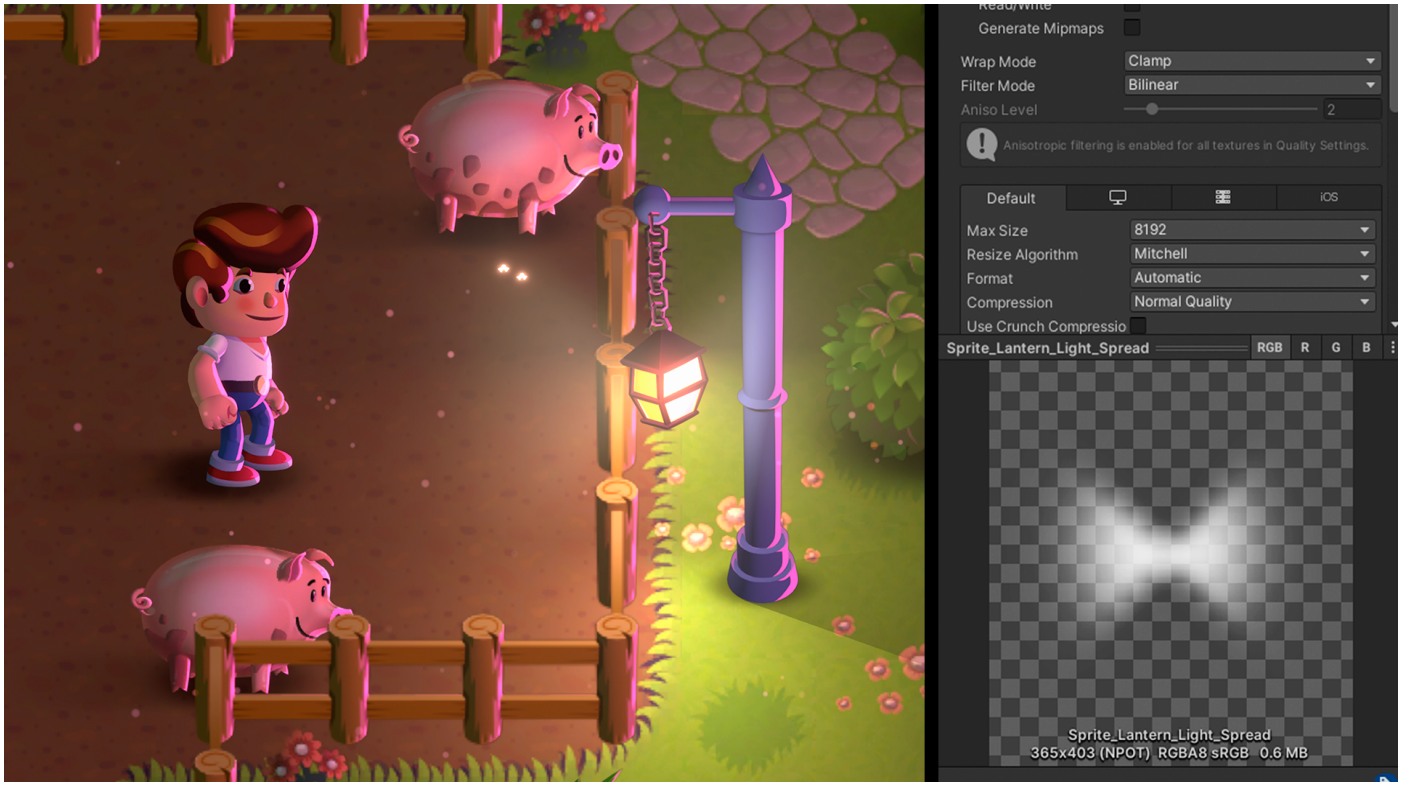
シミュレーションモードの設定

2D のライトと影

2D ゲームは、開発者の意図するように没入感のあるものにすることが可能です。デジタルの人形劇のように、2D ゲームには多くの可動部分やライティングなどのレイヤーがあり、描画が周囲の環境や光源に動的に反応するようにしたいことでしょう。

Unity の 2D ライトシステムには、ゲームのために豊かで多様なライティングや視覚効果を作成するための十分な機能があり、目指す雰囲気やムードの実現に役立ちます。

2D ライトの種類と設定



Happy Harvest の吊り下げランプにライトハローがあるスプライトを使用

Unity の高度な 動的な 2D ライティングシステムを使用し、スプライトに二次テクスチャのような技術を組み合わせることで、レベルのムードを動的に伝え、ゲームプレイを向上させることができます。

2D ゲームは、2D ライトシステムの適切な使用によってムードと没入感を大いに高めることができます。この画像は、Unity のサンプルプロジェクト Lost Crypt のものです。

2D ライトは、Light 2D コンポーネントが設定されたゲームオブジェクトです。これらは、Sprite Renderer、Sprite Shape Renderer、Tilemap Renderer、および Unity 6.3 以降の MeshRenderer や SkinnedMeshRenderer を備えた 3D ゲームオブジェクトと連携します (3D と 2D の複合に関するセクションを参照)。これらのライトでは、ソートレイヤーを使用し、各ライトは 1 つ以上のソートレイヤーに影響を与えることもできます。Target Sorting Layers ドロップダウンリストで、影響を受けるレイヤーを選択できます。同じソートレイヤーにあるライトのレンダリング順序を 制御するには、Light Order オプションを使用します。

2D のフリーフォームライトの使用例

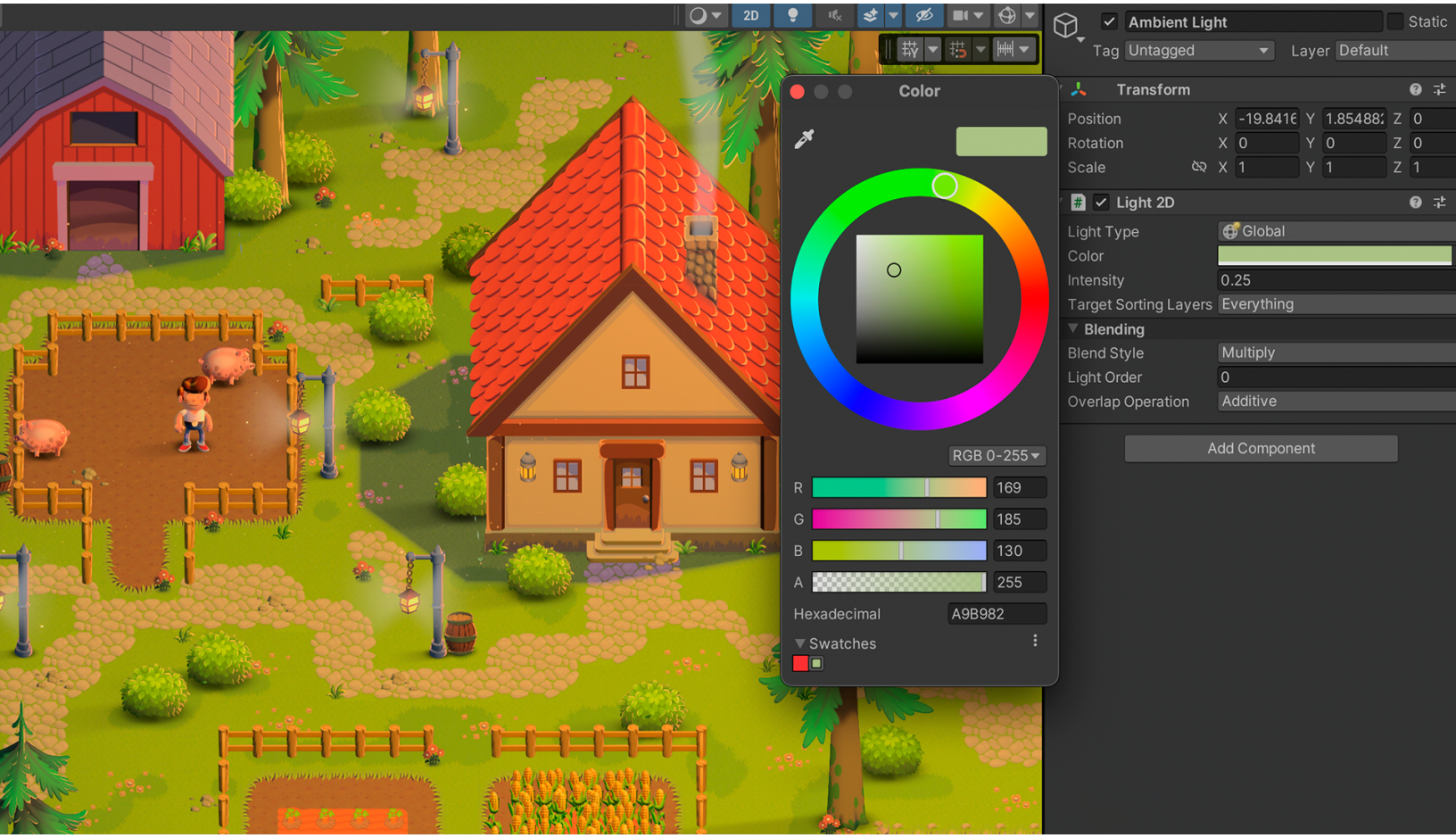
2D ライトには以下の 4 つの異なるタイプがあります。

— **Freeform:** これらのライトは、n 角形の形状にすることができます。このタイプのライトは、非有機的または様式化された環境で使用できます。これはたいていの環境 (溶岩だまりなど) を照らすのに適しており、ライトの形状をシミュレートしたり (天井の穴から差し込む神の光など)、ライトが投影される窓の形に合わせたりすることができます。

— **Sprite**: この形状は、任意のスプライトをライトのテクスチャとして使用することを可能にします。他のライトタイプでは実現できない特定の形状が必要な場合に便利です。適した使用例として、レンズフレア、光芒、ライトクッキー、ディスコボールライトのような光の形状投影、または壁に星を投影する赤ちゃん用ランプなどが挙げられます。

— **Spot:** このライトは、形状を円形または扇形にすることができます。このオプションは、スポットライトやトーチの火、キャンドル、車のライト、懐中電灯、ボリュメトリックライトなど、特定の位置を照らすのに適しています。

— **Global:** このライトは形状を持たず、代わりにターゲットのソートレイヤー上のすべてのオブジェクトを照らします。Global ライトは、Blend Style (ライトとスプライトの相互作用の方法) ごと、およびソートレイヤーごとに、1 つだけ使用できます。まず、基本的な環境光を加えるために使用します。



Global ライトはシーン全体に影響を与えるので、ゲームの世界の雰囲気を簡単に変えることができます。シーンに存在する Global ライトは 1 つだけにする必要があります。そのパラメーターを操作することで、例えば夜のライトのように、強度を下げてシーンに紫色の色合いを適用し、異なる環境条件を簡単にシミュレートできます。

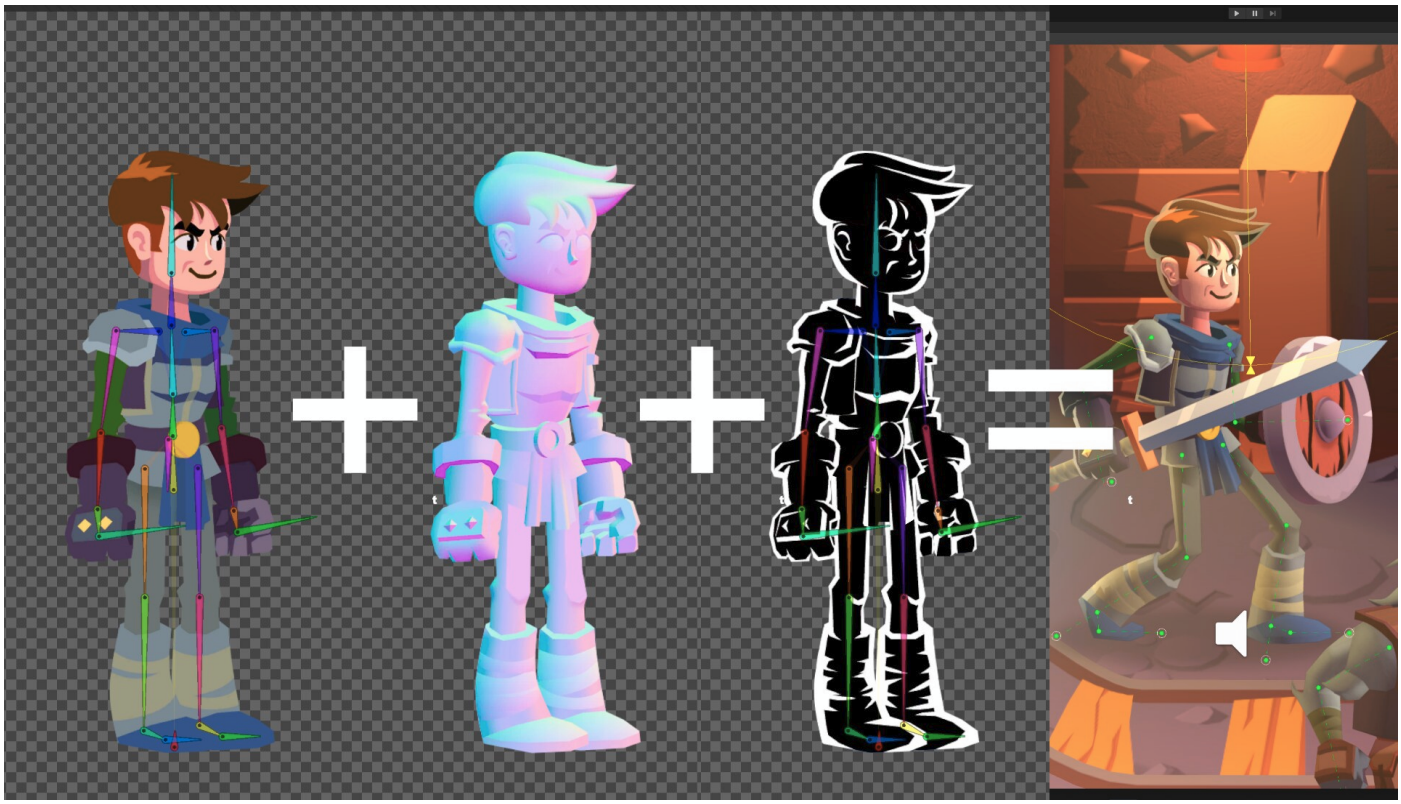
以下の設定は、各ライトタイプに利用可能です。

- **Blend style:** Blend Style で、特定のライトがシーン内のスプライトとどのように相互作用するかを決定します。シーン内のすべてのライトは、2D レンダラーで定義された 4 つの利用可能なブレンドスタイルのいずれかを選択する必要があります。使用できるブレンドモードは 3 つあり、それぞれのモードでスプライトがライトにどのように照らされるかを制御します。
  - **Additive:** 各ピクセルの RGB 値を基本レイヤーのピクセルの値に加算します。明度を加算します。
  - **Multiply:** 各ピクセルの (正規化された) RGB 値を基本レイヤーのピクセルの値に乗算し、色をブレンドします。
  - **Substractive:** 各ピクセルの RGB 値を基本レイヤーのピクセルの値から減算し、暗いトーンを作ります。

実際のブレンドスタイルの例 - (左から順に) Additive、Multiply、Subtractive

- **Light Order**: この値は、同じソートレイヤーを対象とする他のライトに対する、そのライトのレンダーキュー内での順番を決定するものです。
- **Overlap Operation**: このプロパティは、選択したライトが他のレンダリングされたライトとどのように相互作用するかを制御します。このプロパティでは以下の 2 つのモードが利用できます。
  - **Additive:** ライトは他のライトと加算的にブレンドされ、交差するライトのピクセル値が合算されます。これがデフォルトのライトブレンディング動作です。
  - **Alpha Blend:** ライトのアルファ値に基づいてライトが一緒にブレンドされます。これは、ライトが交差する場所の、一方のライトをもう一方のライトで完全に上書きする場合に使用できます。ただし、この場合のライトのレンダリング順序も各ライトの Light Order に左右されます。
- **Shadows**: このオプションを有効にすると、Shadow Caster 2D コンポーネントを備えたゲームオブジェクトがこの光源から影を投影します。
- **Volumetric**: れにより、ライトの減衰具合や投影される影の暗さを調整できます。
- **Normal maps**: これを有効にすると、スプライトの法線マップ情報を使用して、ピクセルの向きに基づいてライトの量が計算されます (二次テクスチャに関する詳細は以下を参照)。

二次テクスチャ



スプライトに二次テクスチャを追加

二次テクスチャは、法線マップとマスクマップ を使用して作成されます。これは、予算、時間、アートのリソースがある場合に、2D ライトと共に使用するオプションであり、ゲームの見た目を新たなレベルに引き上げることができます。キャラクターや背景に、ライトに反応するディテールがさらに追加され、その形状がより鮮明で立体的に見えるようになります。

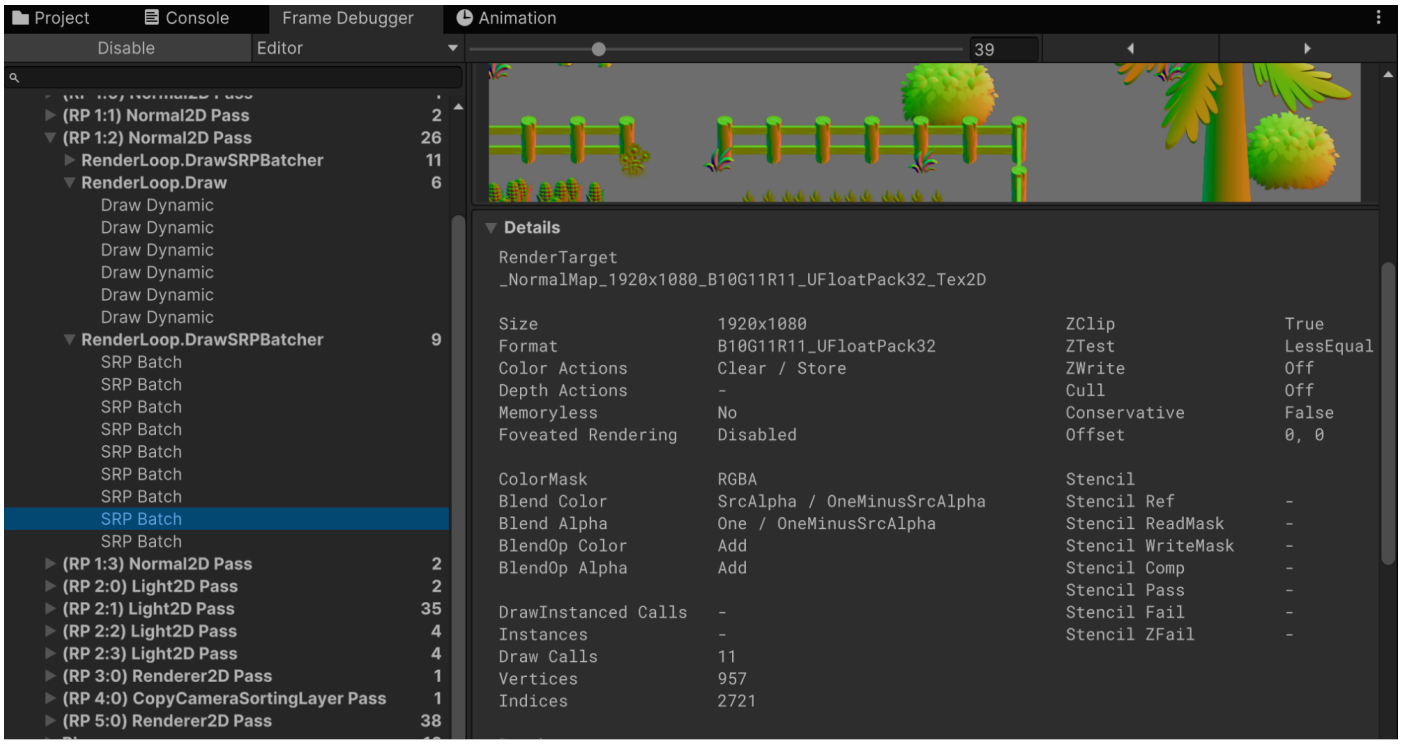
暖かいランタンの光に包まれた Happy Havest の幸せな子豚:(左) 法線マップを使用したライト、(右) 法線マップなしのライト

すべてのスプライトアセットには、マテリアルで使用される 二次テクスチャ をオプションで設定することができます。二次テクスチャは **Sprite Editor > Secondary Textures** を介して変更できます。デフォルトでは、法線マップとマスクマップの 2 種類のテクスチャを割り当てることができます。

マテリアルレベルではなくアセットレベルでテクスチャを割り当てることには、いくつかのもっともな理由があります。

利点の 1 つは、スプライトを使用するレンダラーは、スプライトと二次テクスチャが異なっていても、マテリアル (Sprite-Lit-Material) を共有できることです。つまり、それらをまとめて処理することで、より効率的にレンダリングできることを意味します。

スプライトシェーダーの場合、法線マップとマスクマップを加える必要があります。すべてのスプライトには、スプライト、法線マップ、マスクマップを備えた独自のマテリアルが必要です。そうなると、プロジェクトで使用するマテリアルは急速に数百にまで増加するでしょう。ただし、すべてのスプライトに追加のテクスチャを設定する場合は、使用できるスプライトマテリアルは 1 つだけです。マテリアルが 1 つであれば、ドローコールも数百回ではなく 1 回で済みます。



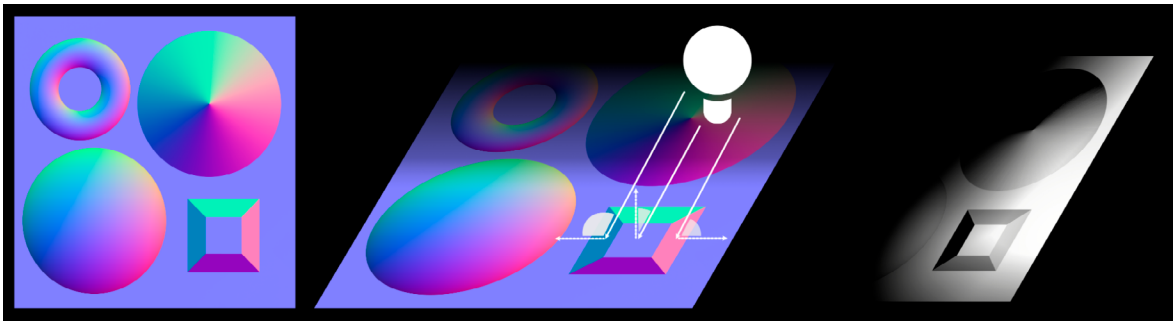
フレームデバッガーを使用すると、フレームのレンダリングのためのすべてのステップを観察できます。バッチ処理できないマテリアルが多いと、パフォーマンスに影響を与えます。

e-book の最後にある最適化セクションに、さらに多くのヒントが掲載されています。

法線マップ

適切に作成された法線マップは、スプライトが 3D に見えるかどうかを左右します。法線マップの各ピクセルは、メインテクスチャの角度に関するデータを格納しています。赤、緑、青 (RGB) のチャンネルには、X、Y、Z 座標の角度データが格納されます。法線マップを使用するすべてのライトには方向があり、法線マップを備えたテクスチャのピクセルは、この方向とピクセルの方向に基づいてシェーディング処理されます。これは現実と同じように機能します。ピクセルがライトの方向を向いている場合は照らされ、向いていない場合はライトを受けません。

RGB 値が法線マップの角度にどのように影響するかを見てみましょう。



法線マップでは、受け取るライトは、ピクセルが向いているベクトルと光源との間の角度の大きさに依存します (これを ドット積 とも呼びます)。角度が小さいほど、両方のベクトルがより整列し、ピクセルが受け取るライトが多くなります。この画像の右側では、法線マップが Unity の 2D ライトによって照らされています。

上の画像は、紫色のピクセルがカメラの方を向いている法線マップです。その RGB 値はそれぞれ127、127、255です。各色のチャンネルの値は 0 から 255 までなので、127 はほぼ中間値です。左側を向くためには (-90 度)、R の色値を 0 に設定する必要があります。右側を向くためには、R を 255 に設定します。真下または真上を向くためには、G チャンネルをそれぞれ 0 または 255 に設定します。

Happy Harvest における、アニメーションキャラクター、プロップ、タイルマップに適用された法線マップとマスクマップの例

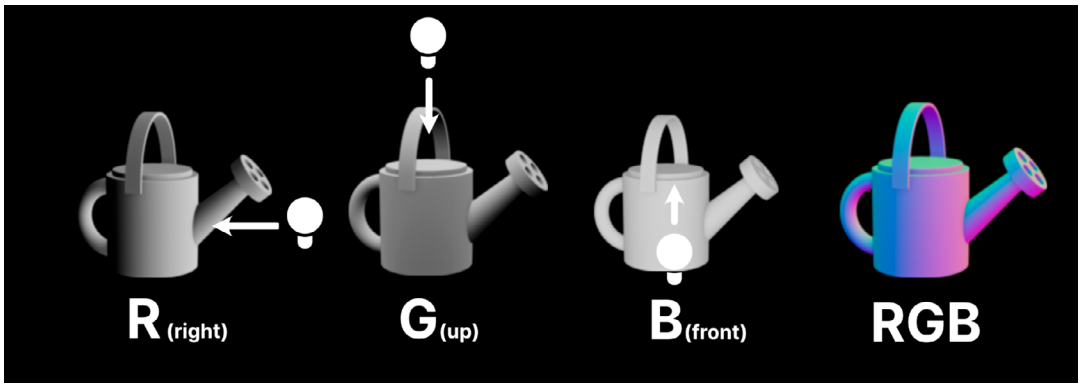
法線マップのためのペイント技術

法線マップの作成方法は、主に使用しているアートスタイルとスキルに依存します。3D マテリアルの法線マップを作成する技術は、一般的に 2D スプライトにも適用されます。以下のヒントを使用すると、外観の完全な制御に役立ちます。

ライトの方向を手動でペイント

法線マップをペイントする 1 つの方法は、異なる角度から照らされたスプライトを描画し、それを 1 つのテクスチャにまとめることです。スプライトは、R チャンネルの右側からのライトと、G チャンネルの上からのライトで照らされます。B チャンネルでは、スプライトは正面から照らされていますが、簡略化のために、2D スプライトで法線マップを使用する際にはこのチャンネルを省略できます。これは、2D の正面のライティングが全体のシェーディングにそれほど寄与しないためです。

ただし、このアプローチは時間がかかる可能性があり、X 軸と Y 軸で少なくとも 2 回シェーディングをペイントする必要があります。



右側と上から照らされた 2 つのシェーディンググレースケール画像を法線マップに結合する方法。正面のシェーディンググレースケールは任意です。

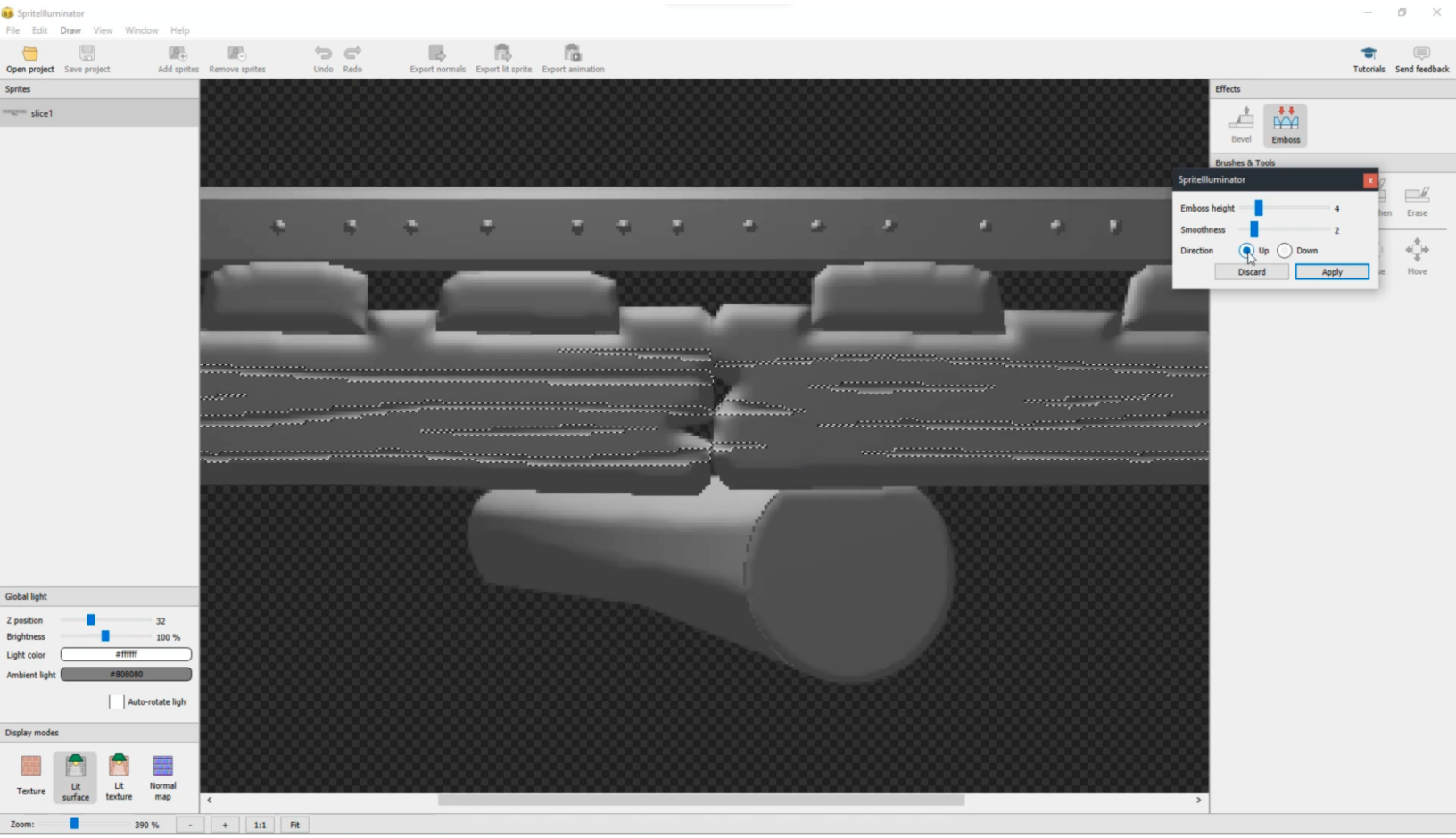
法線マップジェネレーター

別のペイントアプローチは、法線マップジェネレーターアプリケーションを使用することです。そのようなアプリケーションでスプライトを開くと、たった数回のクリックで法線マップを生成できます。ジェネレーターアプリケーションはスプライトの角度を考慮しないため、スプライト全体に使用するのは避けてください。また、これらのアプリケーションはオブジェクトも認識しません。代わりに、スプライトの色から形状を推定したり、画像編集アプリケーションのベベルやエンボス、あるいはレリーフ彫刻に似た一般的なフィルターを追加したりすることで形状を推定します。例えば、顔の角度は認識できませんが、角度の変化があるべき場所の推測を試みます。これは制限でもありますが、チェーン、ケーブル、ドラゴンの尾のようなベベルのあるスプライトセクションの法線マップの生成に役立ち、レンガ、石、木材のような面法線にも役立ちます。

セクションを法線マップジェネレーターにインポートし、値を調整し、エクスポートしてから、必要な部分と詳細を自分で追加します。

法線マップジェネレーターには、以下のようなものがあります。

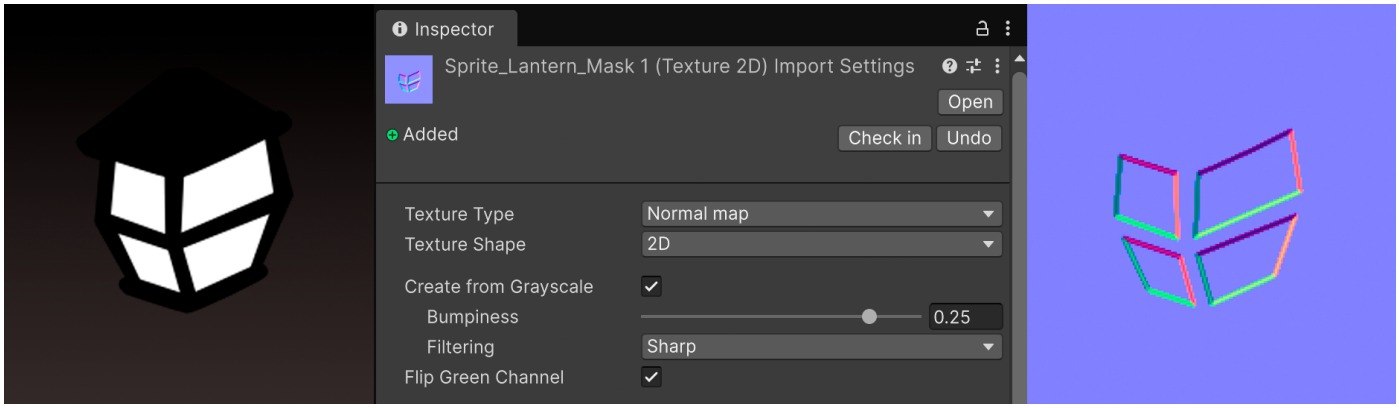
- SpriteIlluminator
- NormalPainter
- Krita の Tangent Normal Brush
- Laigter
- Sprite Lamp



SpriteIlluminator のエンボスツールを使用して法線マップに隙間を生成

Unity におけるグレースケールからの法線マップ

この技術に関して、Unity はグレースケールのハイトマップから法線マップを生成する方法を提供しています。グレースケールのハイトマップは、黒が最小の表面高度を、白が最大の高度を表すテクスチャです。法線 マップとして画像をインポートし、**Create from Grayscale** オプションを有効にする必要があります。この技術は、エンジンを離れることなく迅速に法線マップを生成するために便利です。



Create from Grayscale オプションを使用してテクスチャを法線マップに変換: **Flip Green Channel** オプションで、エンボス効果で盛り上がっているように見せたり、凹んでいるように見せたりすることができます。

この方法を使用すると、Inspector に **Bumpiness** (凸凹度) スライダーが表示されます。Unity ではピクセルの明度を使用し、高さの違いを法線マップの角度に変換します。角度の勾配はスライダーで制御されます。Bumpiness 値が低いと、ハイトマップの鋭いコントラストでさえも、より柔らかい角度や凹凸に変換されます。

色のサンプリング

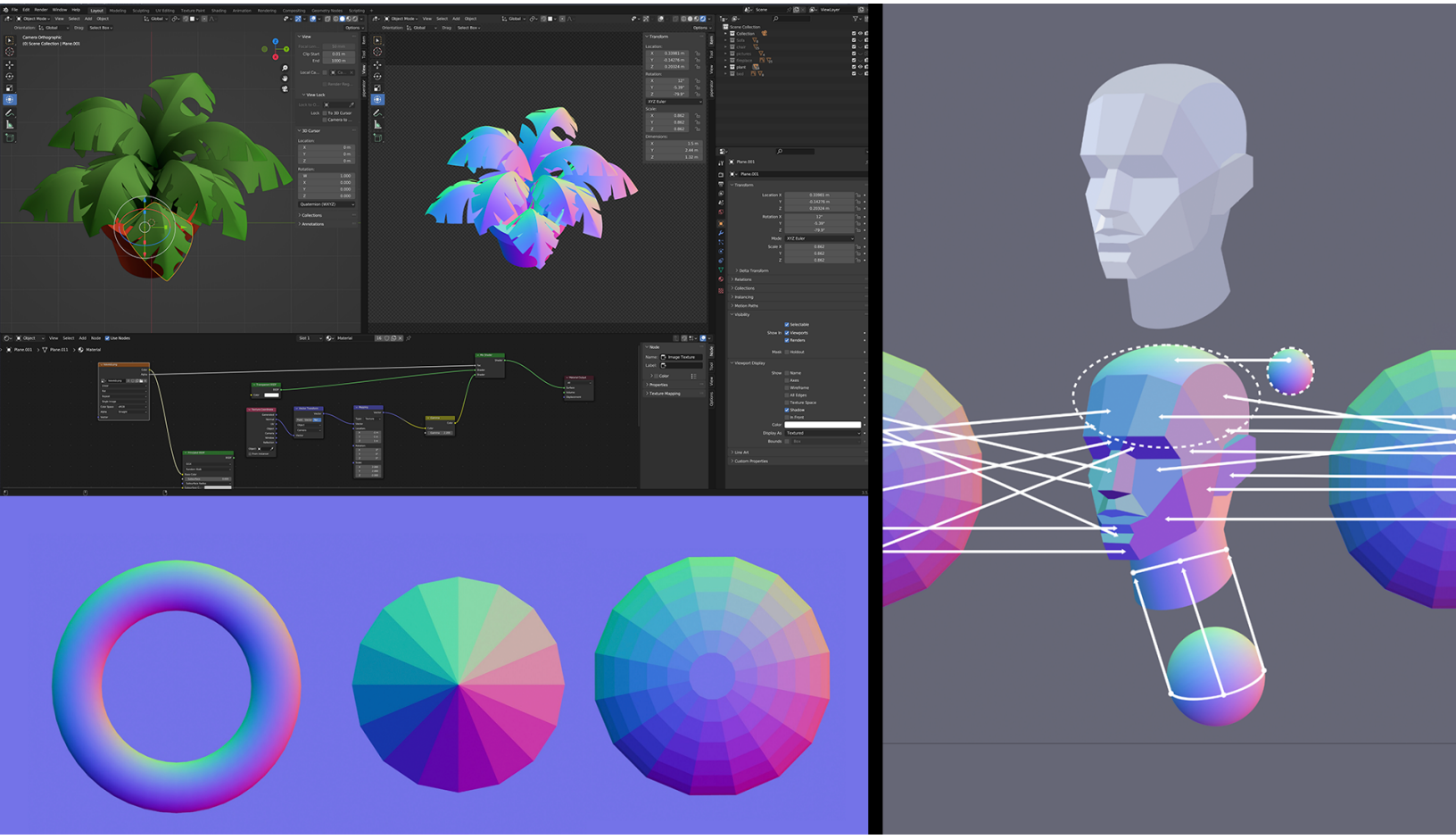
法線マップをペイントするには、異なる角度に使用する色を把握することから始めます。

まず、法線マップのパレットを取得して、オンラインの任意のテンプレートで表面の角度を表すために使用される色のサンプリングを行います。自分のペイントワークフローがどのようなものであっても、パレットをお気に入りのペイントアプリケーションにコピーし、カラーピッカーを使用して法線マップにペイントする色を選択するだけで済みます。

角度の色は 100% 正確である必要はありません。数度の違いは問題になりません。ただし、スプライトの全体的な形状は、信ぴょう性のあるものにしておくようにしてください。適切でない角度の色を使用すると、ライトが当たったときに形状全体が崩れてしまいます。

法線マップをペイントするのは最初は難しいかもしれません。なぜなら、優れた空間的想像力が必要です。まずは、頭部の基本平面のようなシンプルなものから始めるとよいでしょう。下の画像の右側にある簡略化された人間の頭部モデルは、ローポリ風です。用意されたサンプルは、このzipファイル をダウンロードして確認できます。

法線マップをペイントする際は、スプライトの一部である基本の 3D 形状を想像し、それぞれの部分の角度を視覚化してみてください。角度がわかれば、パレットスプライトのどの部分から色のサンプリングを行うかがわかります。



左上から順に: Blender で法線マップ生成のためのプロップの作成、テンプレートから色のサンプリングを行い頭の法線マップをペイント、形状のテンプレート

人間の頭の例では平面で作業していますが、柔らかいブラシでペイントする際もプロセスは似ています。硬い辺をぼかすことで、より自然な外観を実現できます。

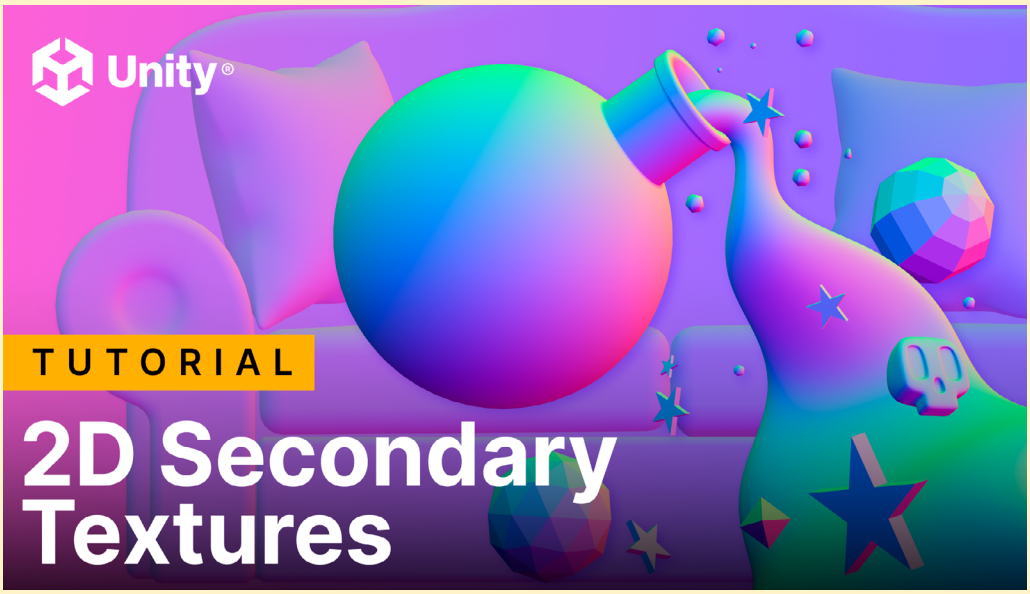
覚えておくと便利なショートカットをいくつか紹介します。球面的な形状がある場合、パレットから法線スフィアを貼り付けることができます。円柱の形状がある場合、スフィアの一部を取り出し、貼り付けて引き伸ばすか、グラデーションを作成することができます。

法線マップの一部をコピーして貼り付けたり、回転させたりするとシェーディングが壊れることに注意してください。しかし、これを利用することもできます。例えば、凹面の球形状が必要な場合は、単にスフィアを 180 度回転させて穴を作ります。

これは、本書のサンプルにある法線マップを、エンジン内の白いスプライトに適用した場合の表示例です。

最適な法線マップ生成の方法を選択してください。ゲーム用に多くのアセットが作成される可能性が高いので、最も目立つオブジェクトに焦点を当て、ゲームの他の部分は簡素化してください。

以下の動画でさまざまなペイント技術を説明しています。



セカンダリテクスチャ - Lit スプライトと 2D VFX

2D ライティング用のスプライトの準備

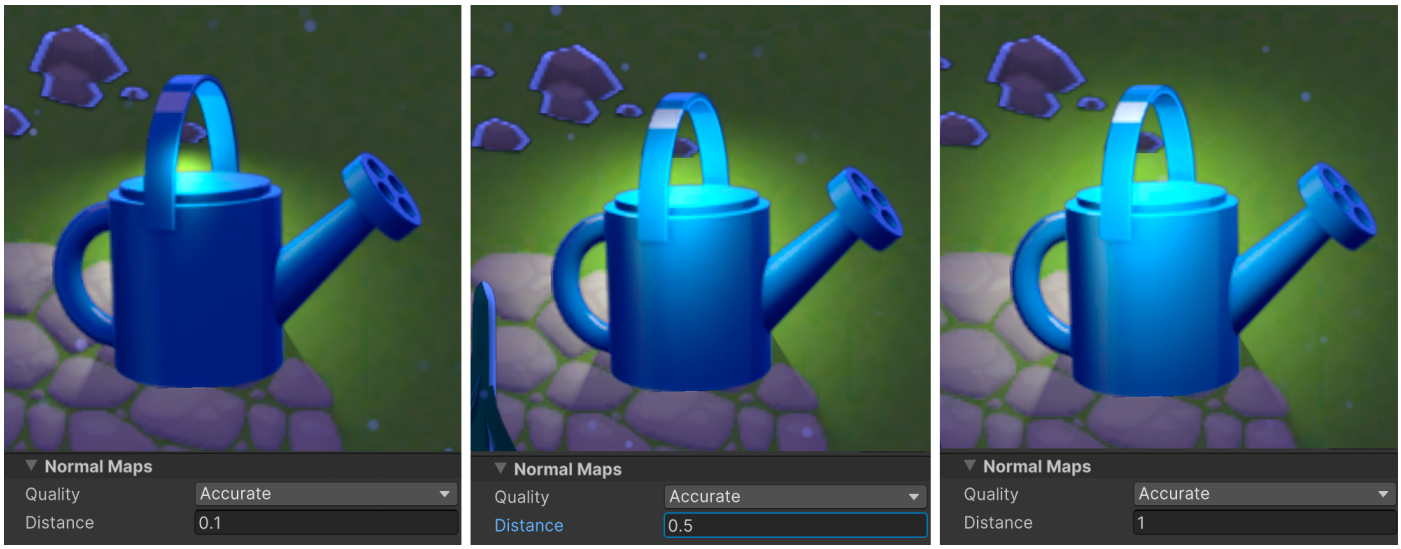
基本色のスプライトに注意してください。ゲームで 2D ライティングを広範囲に使用する予定があり、法線マップを最大限に活用したい場合は、スプライトにライトと影をペイントしないでください。

2D ライティングは、すでに影がペイントされたスプライトに使用すると見栄えが良くありません。法線マップにライトをペイントすることになるので、最終的には作業量が 2 倍になります。代わりに、非方向性の影をペイントすると、太陽光などのディレクショナルライトを避ける限り、スプライトはより見栄えがよくなります。

2D ライトをオフにすると、スプライトに色の情報 (アルベド) が含まれますが、ライトや影の情報が含まれていないため、フラットに見えます。

法線マップの有効化

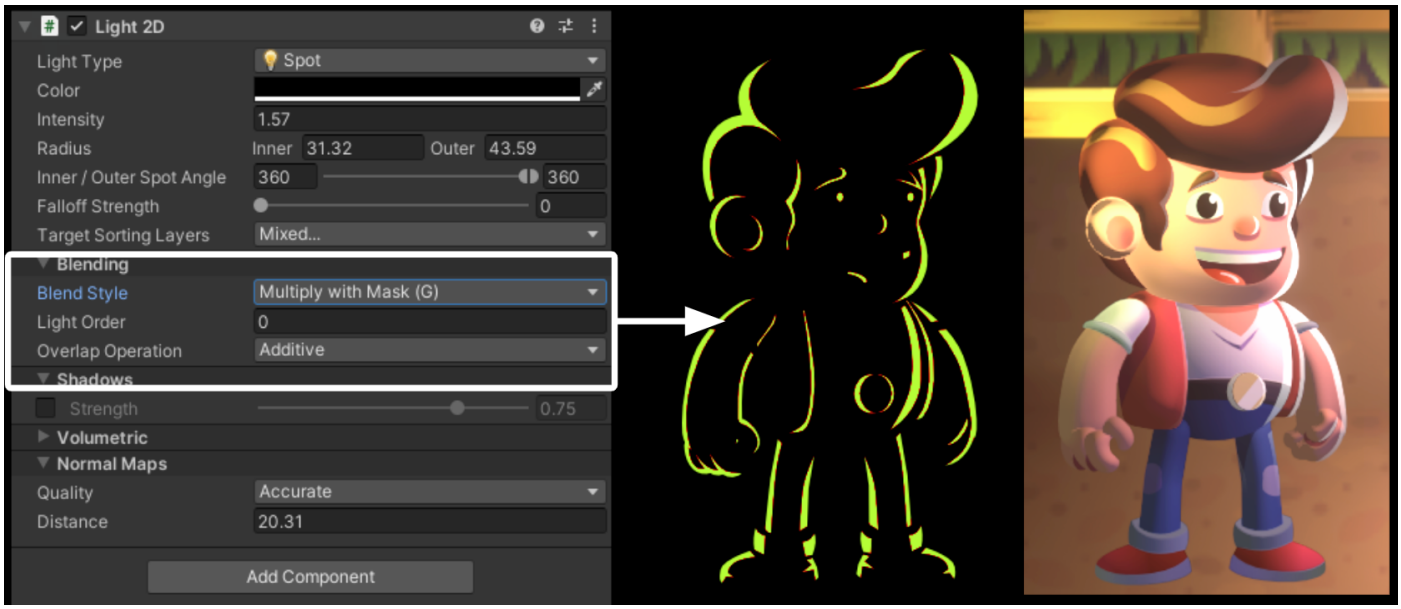
Unity のサンプル Happy Harvest や Dragon Crashers では、ボリュームの錯覚を作り出すために、ライトと法線マップが至る所で使用されています。スポットライト、ポイントライト、フリーフォームライトと共に法線マップを使用できます。スプライトで法線マップを使用するには、2D Light の法線マップ を有効にする必要があることを忘れないでください。ライトとスプライトサーフェスの間の距離を制御することもできます。効果を増減することもできます。



2D ライトの法線マップサポート設定と、距離に対するさまざまな設定の効果

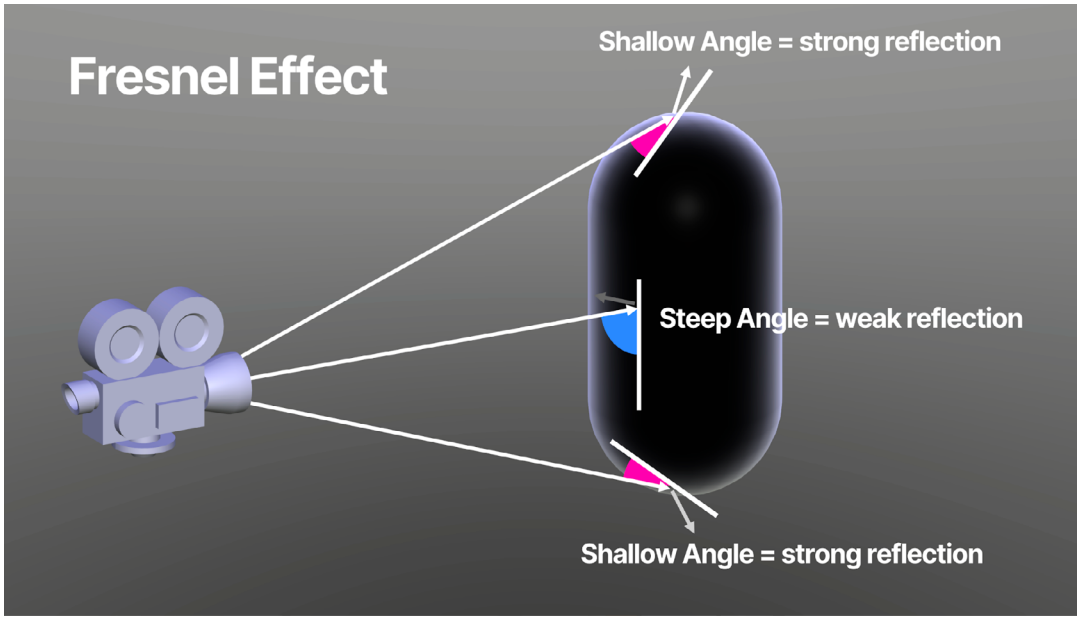
マスクマップ

マスクマップは、リムライトなどのその他のライティングに使用できます。独自のテクスチャを加え、Shader Graph シェーダーで名前を指定して参照することで、その他の視覚的効果に利用できます。



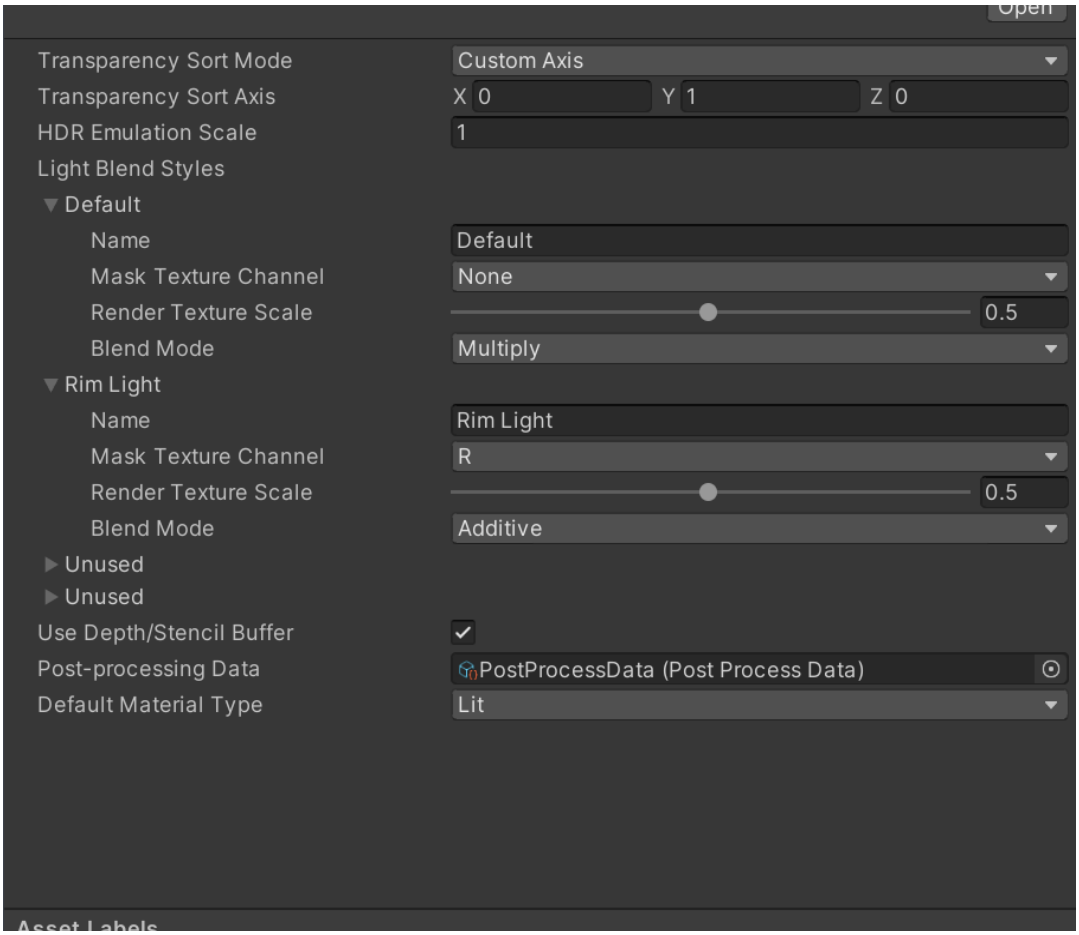
“ヒーロー” のライティングスタイルは、Happy Harvest の農夫のキャラクターを目立たせるために作成されました。この効果では、キャラクターのハイライトのペイントに使用される、緑のチャンネルのマスクマップにのみ影響を与えるブレンドスタイルを使用しています。

リムライティングは、キャラクターの輪郭を強調するために使用される効果です。これは、物体の後ろからライトが来ることと、光の散乱の自然な特性をシミュレートします。これをフレネル効果と呼びます。現実世界では、ライトが物体の表面に当たる角度が広いほど、物体からの反射光が多くなります。



フレネル効果の図解

湖の水は遠くから見るとより反射しますが、下を見るときに水の中に自分の足を見ることができるのはこのためです。これが、物体のエッジがより反射的である理由でもあります。2D グラフィックスでは、マスクマップとブレンドスタイルを使用してこの効果のシミュレーションが可能です。



2D レンダラーのデータアセットで Light Blend Styles のオプションを設定

フレネル効果の作成方法

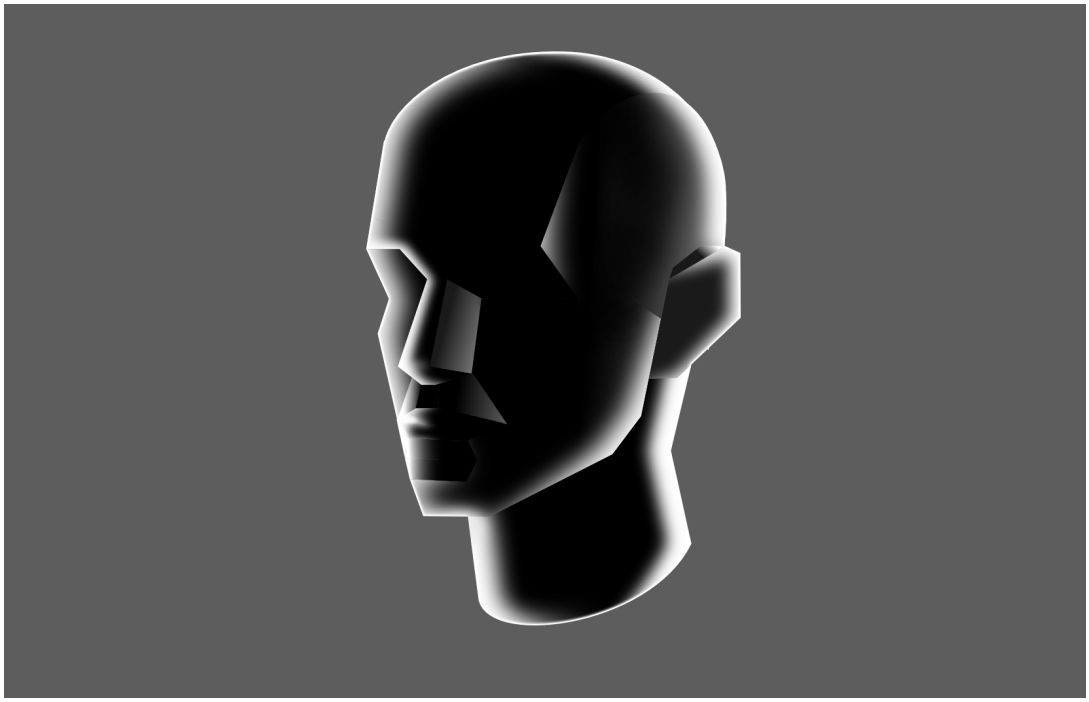
上記の人間の頭のサンプルをダウンロードしたら、Unity でスプライトを開き、以下のステップに従ってフレネル効果を作成してみてください。

1. 2D Renderer Data アセットには、**Light Blend Styles** オプションの下にさまざまなブレンドスタイルのための 4 つのオプションがあります。最初のオプションはデフォルトなのでそのままにしておき、代わりに 2 番目のオプションを操作してください。“リムライト” や “フレネル” など、自分に合った名前を付けてください。
2. **Mask Texture** チャンネルを R に設定します。ライトは Mask Map テクスチャの赤色を使用します。
3. **Blend Mode** を **Additive** に設定します。これにより、ライトが既存のライティングの上に追加され、明度が増します。

4. フレネルライトはマスクマップの 1 つのチャンネル (この場合は R) しか使用しませんが、簡素化のためにマスクマップを白と黒でペイントしましょう。

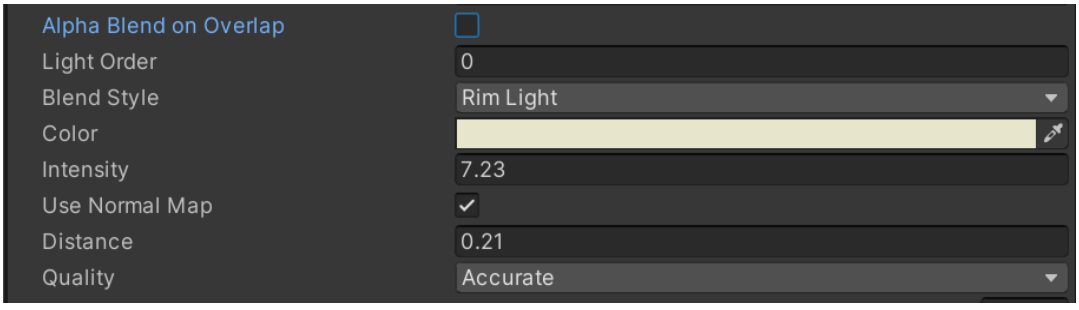
    — ライトを反射する部分は白く、反射しない部分は黒くなります。フレネル効果はオブジェクトのエッジに影響を与えるので、基本スプライトをコピーし、黒く塗り、その後エッジを白くハイライトします。それは、背後から明るいライトが当たっている物体のように見えるようになるはずです。

5. 作業を早めるために、オブジェクトに内側のグロー効果を追加し、その上に詳細を描き加えます。残念ながら、このプロセスを早めるアプリケーションはないので、ここではあなたのペイントスキルに頼る必要があります。

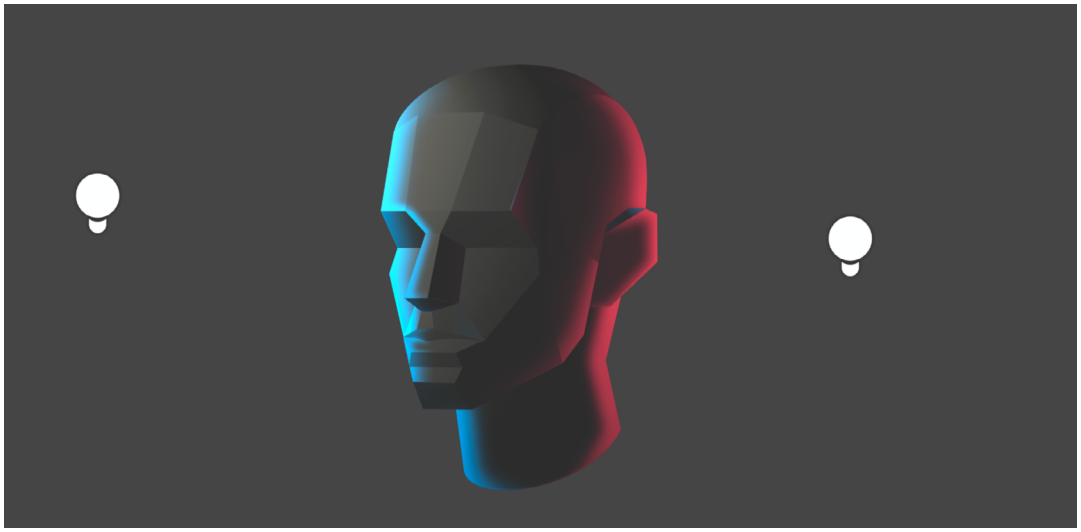


サンプル からの基本平面ヘッドスプライトのマスクマップ

6. ライトがマスクマップに影響を与えるようにするために、その Blend Style を先に作成したスタイルに変更します。



ライトの Blend Style を Rim Light に変更

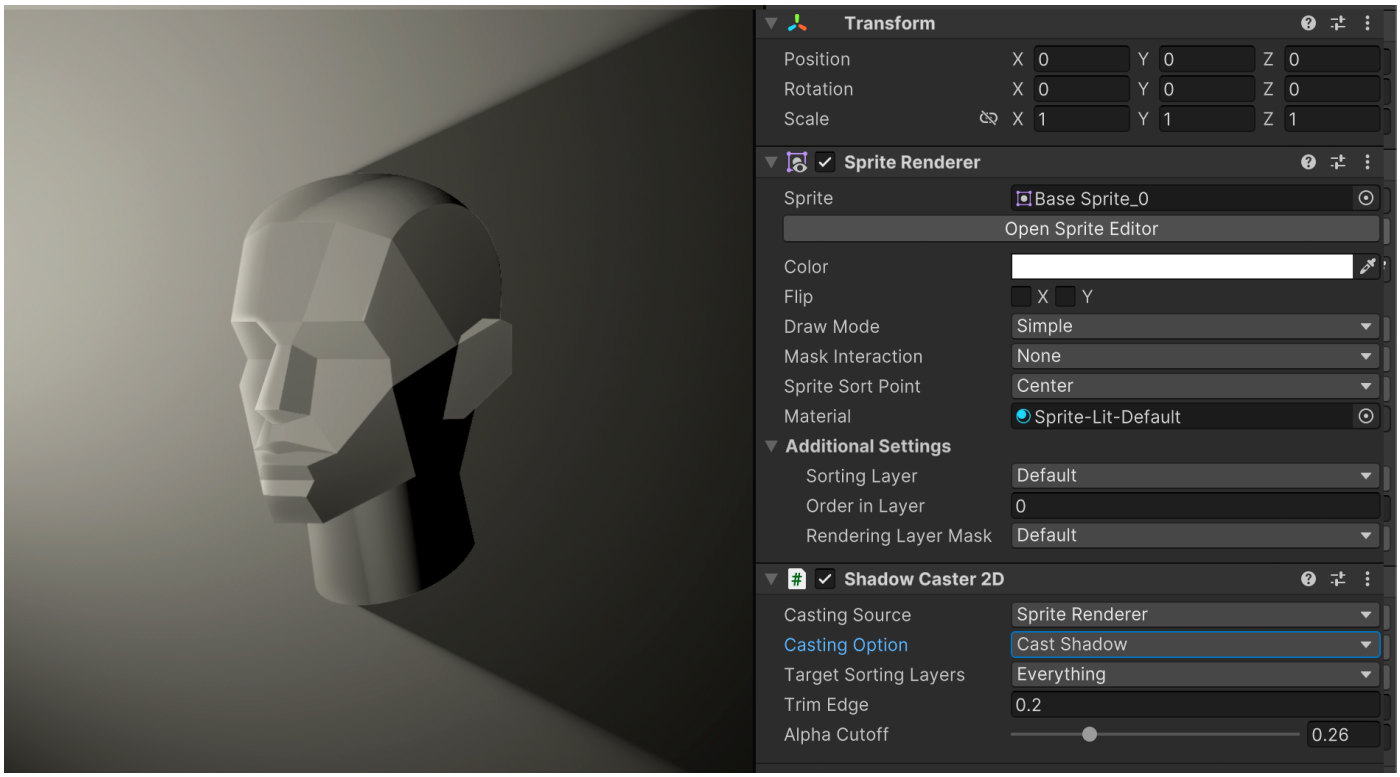


Unity エディターでのスプライトのリムライティング効果

7. フレネルライトは、法線マップオプションが有効で、距離が低い値に設定されているときに最も効果的に機能します。これにより、オブジェクトの反対側がハイライト表示されることを防ぎます。

2D シャドウ

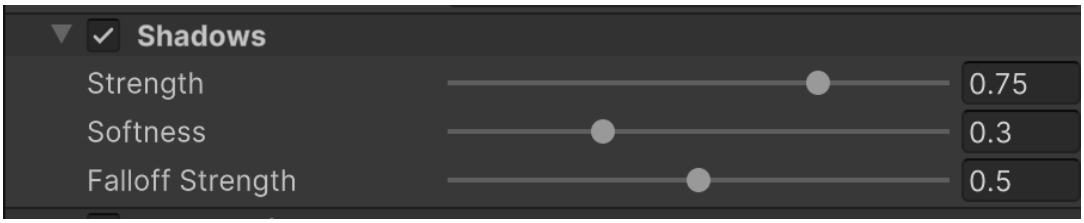
ゲームオブジェクトが 2D ライトから影を落とすようにするには、オブジェクトに Shadow Caster 2D コンポーネントを加えます。



Shadow Caster 2D コンポーネント

Unity 6 では、デフォルトで影のキャスト元はスプライトレンダラーであり、スプライトアセットに自動生成されたアウトラインが使用されます。アウトラインのトリミングとアルファ値をカットオフするためのその他の設定があります。

また、Shadow Caster の形状エディターに切り替えて、そのジオメトリを編集し、オブジェクトのシルエットに合わせたり、作成した他の形状に合わせたりすることもできます。影を作成するためにライトにもいくつかのオプションを設定する必要があります。



2D ライトで影を作成するためのオプション

Strength オプションで、影の中にある他のオブジェクトに対するライトの不透明度を決定します。1 に設定されている場合、ライトは影の領域内を一切照らしません。

Softness オプションは、シャドウキャスターからの距離に基づいて影の辺をぼかし、影の外観をより自然にします。**Falloff Strength** は影のエッジの強度を調整します。

2D のライトと影の視覚的技術

このセクションでは、ゲーム内のライティングシナリオを再現するための創造的な方法を見ていきます。これらの例は、Happy Harvest プロジェクトのスクリプトとアセットから引用したものです。

無限の影

影を投影するオブジェクトにより、ライトの領域が制限されている場合や、街灯や暖炉のような強く集中した光源がある場合に適用される無限の影が生成されます。

シャドウキャスターは通常、横スクロールゲームのオブジェクト全体またはキャラクター全体ですが、等角投影またはトップダウンの視点ではこの効果は非現実的に見えるでしょう。奥行き感を改善する方法の 1 つとして、地面に接しているオブジェクトの基部に沿った形のシャドウキャスターを作成することが挙げられます。Happy Harvest の以下の例では、Shadow Caster 2D コンポーネントが、キャラクターゲームオブジェクトの足のボーンに設定されているので、足にのみ影響を与えます。

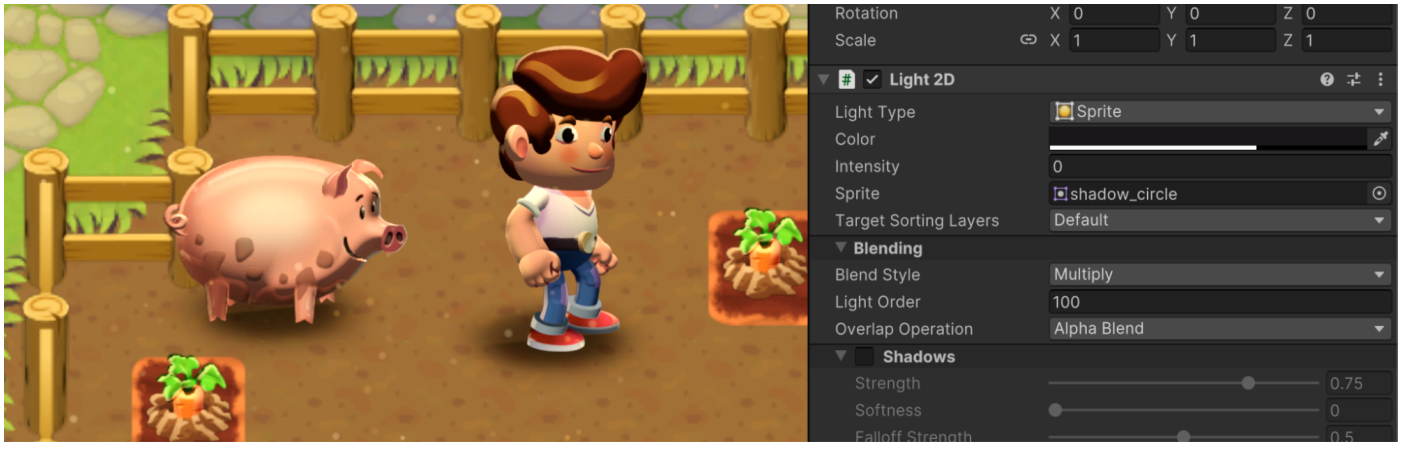
左の画像ではシャドウキャスターはキャラクターの足のボーンに設定されているためうまく機能しますが、右の画像では一般的な環境ライティングであるため、適切に機能しません。

この技術は強い光源や制限された光源ではうまく機能しますが、開放的な空間で太陽光源からライトが当たると、上の木々の画像のように、無限の影の効果は奇妙で不自然に見えてしまいます。

Unity の 2D ライトシステムには、柔軟性に富んでおり、他のタイプのライトや影を再現するために創造的な方法で活用できます。

ぼんやりした影

動くキャラクターを環境になじませるために役立つ昔ながらの技術として、影を投げかけるオブジェクトの子オブジェクトとして、ぼんやりとしたシンプルな影のスプライトを加える方法があります。ここでのコツは、Light 2D コンポーネントの **Light Type** を **Sprite** に設定し、**Blend Style** を **Multiply** に設定することです。ライトの色が暗い場合、下のエリアを暗くし、**ネガティブライティングと呼ばれる効果を生み出します。**



影を表現するための安価で効果的なテクニックとして、キャラクターの下に 2D ライトをぼんやりとした影として使用する方法があります。

トップダウンゲームでは、“太陽光” からの無限の影の投影が奇妙に見えることがあり、木のような静的オブジェクトの標準的なぼんやりとした影も同様です。Happy Harvest では、時間に応じて回転し伸びるような、長くぼんやりとした影を作成しました。その結果、より柔らかな影が生まれ、アートディレクションにより忠実に沿ったものになりました。

DayCycleHandler スクリプトの UpdateShadow 関数が、この影を回転させます。このスクリプトは “マネージャー” として機能し、関数を使ってループ処理を行い、影のサイズと回転を更新します

他のぼんやりとした影と同様に、伸びていてぼんやりとした影はスプライトに基づくライトです。Happy Harvest をダウンロードして開くと、**Trees** という親ゲームオブジェクト内の任意の子オブジェクトを調べるとこれを確認できます。

Happy Harvest の木や茂みの長く伸びたぼんやりとした影

昼夜サイクル

2D シーンには 3D シーンのようなディレクショナルライトはありません。しかし、X および Y 位置からスプライトを照らす光源を使用し、効果を強化するために法線マップまたはマスクマップを使用することができます。これにより、日が進むにつれて太陽が動くような効果を作成でき、これはトップダウンおよびシミュレーションゲームの重要な要素です。低スペックのプラットフォームを対象にしている場合、大型のライトを使用することはコストがかかることにも注意が必要です。

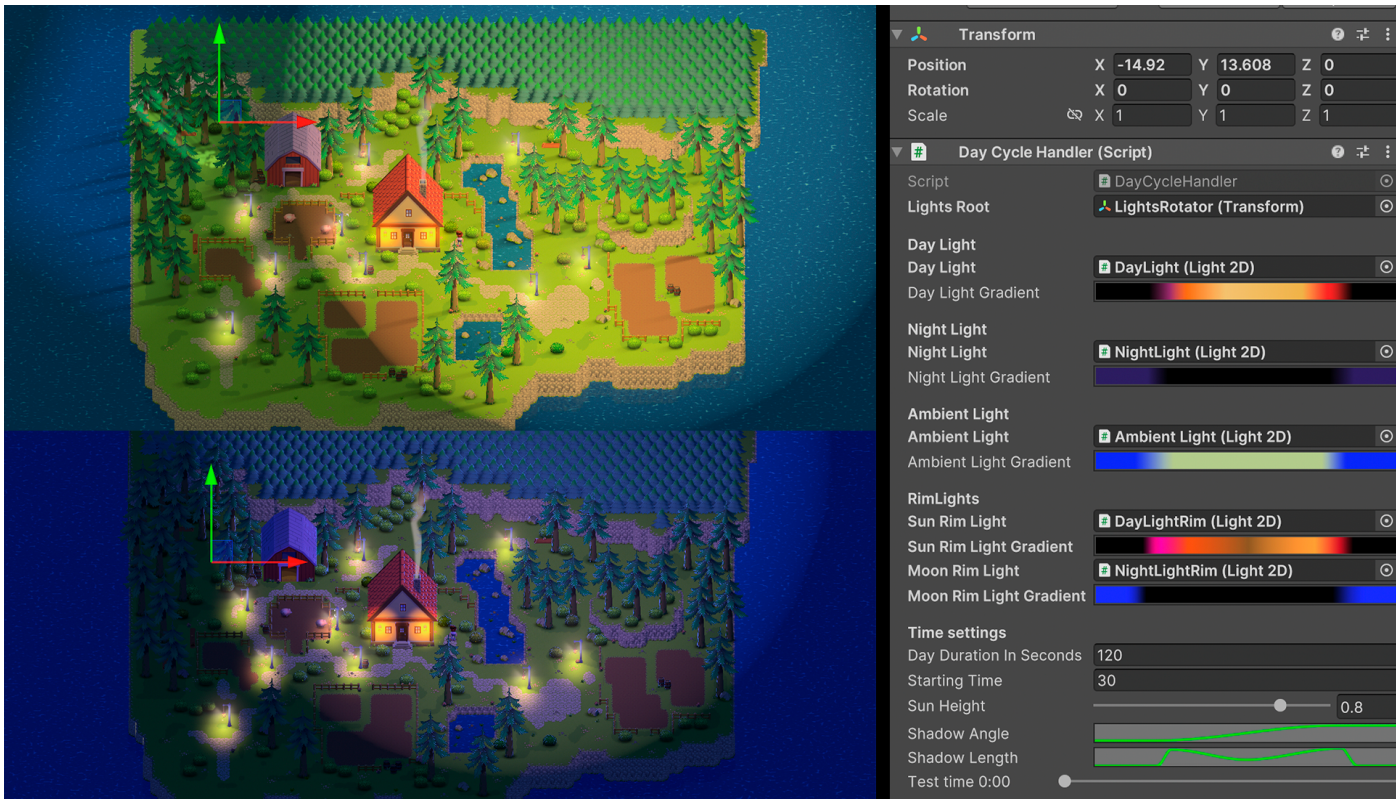
Happy Harvest では、大型のスポットライトがキーライトとして使用されています。それはメインカメラに設定されているため、常に画面に表示され、太陽の動きをシミュレートするスクリプトと共に回転します。

プロジェクト内の子ゲームオブジェクトである **LightsRotator** には、4 つのライトが取り付けられています。

- **NightLight**: 月の光の向きをシミュレートします。
- **DayLight**: 太陽の光の向きをシミュレートします (NightLight の反対の位置)。
- **NightLightRim:** NightLight の月のディレクショナルライトに似ていますが、キャラクターとプロップのリムライトにのみ使用されます。
- **DayLightRim:** DayLight の太陽光のディレクショナルライトに似ていますが、キャラクターとプロップのリムライトにのみ使用されます。

これらのライトの動きを制御するスクリプトは、日中の色の変化も制御します。特定の時間に各ライトの色を設定するためにグラデーションが使用されました。これらのグラデーションは、**DayCycleHandler** ゲームオブジェクトに添付された **DayCycleHandle** スクリプトで見ることができます。

有限の影の効果を制御するためのパラメーターとして、**Shadow Angle** と **Shadow Length があります。** それぞれのフィールドでは、アニメーションカーブが日中の影の時計回りの角度を示します。長さのパラメーターは、特定の瞬間における影の長さを定義します。例えば、太陽が沈むときには長い影が必要で、“太陽” がシーンを垂直に照らすときには短い影が必要です。Test Time スライダーを動かして、Shadow Angle と Shadow Length の設定を更新する必要がある場合もあります。



Happy Harvest の大きなスポットライト。一方は太陽、他方はも月を表しています

フリーフォームライトの操作

トップダウンゲームでは、大きくて高い建物は、シャープなエッジのある正確な影を投影する必要があります。Happy Harvest では、フリーフォームライトを使用して建物の影を作成しました。それらは、建物が地面に投影するであろう影を再現していますが、2D では扱うべき深度情報がないため、これが必要な近似です。

建物の影を作成するために使用されるフリーフォームライトでは、Multiply ブレンドモードとより暗い色を使用するネガティブライティング技術を使用しています。

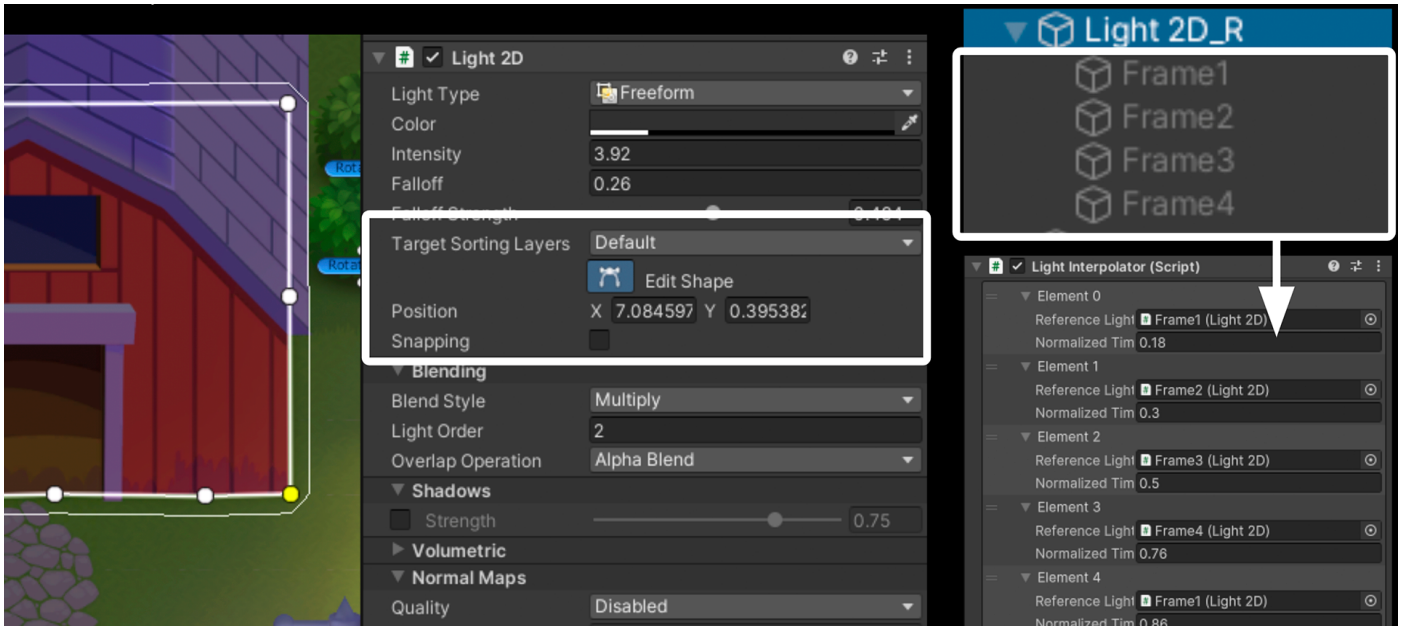
サンプル画像に見られるような明確な影を表現する際の課題は、それらを昼夜のサイクルとどのように調和させるかという点です。影が “太陽” のさまざまな位置に反応するようにするために、フリーフォームライトのベクトルポイントを異なるリファレンスシャドウの間で補間する Light Interpolator スクリプトが作成されました。

サンプルの Hierarchy ウィンドウには、**Light_2D_Warehouse という名前のゲームオブジェクトがあります。**このゲームオブジェクトに取り付けられているのは 4 つのフリーフォームライトです。それぞれが建物の上、右、下、左に太陽があるときに建物が投影する影を再現しています。このスクリプトでは、API を使用して異なるベクトルポイントを滑らかに補間します。

倉庫の下にできる、よりはっきりとした影。一日を通して移動する。

最初に上の影が作成され、その後、下の影と建物の両側の影を作成するために修正されます。各影の点数が同じになるようにして、各影を作成する際にそれらの点間の遷移を考慮することが重要です。

影が作成されると、それらは Light Interpolator コンポーネントスクリプトに追加され、日中の各影の時間的なウェイトを示す正規化パラメーターが設定されます。スクリプトの **Preview Time** 機能を使用すると、エディターでどのように見えるかを事前に視覚化できます。



サンプルで使用される補間関数のスクリプトによって、フリーフォームライトのベクトルの位置が補間されます。

Sprite Custom Lit シェーダー

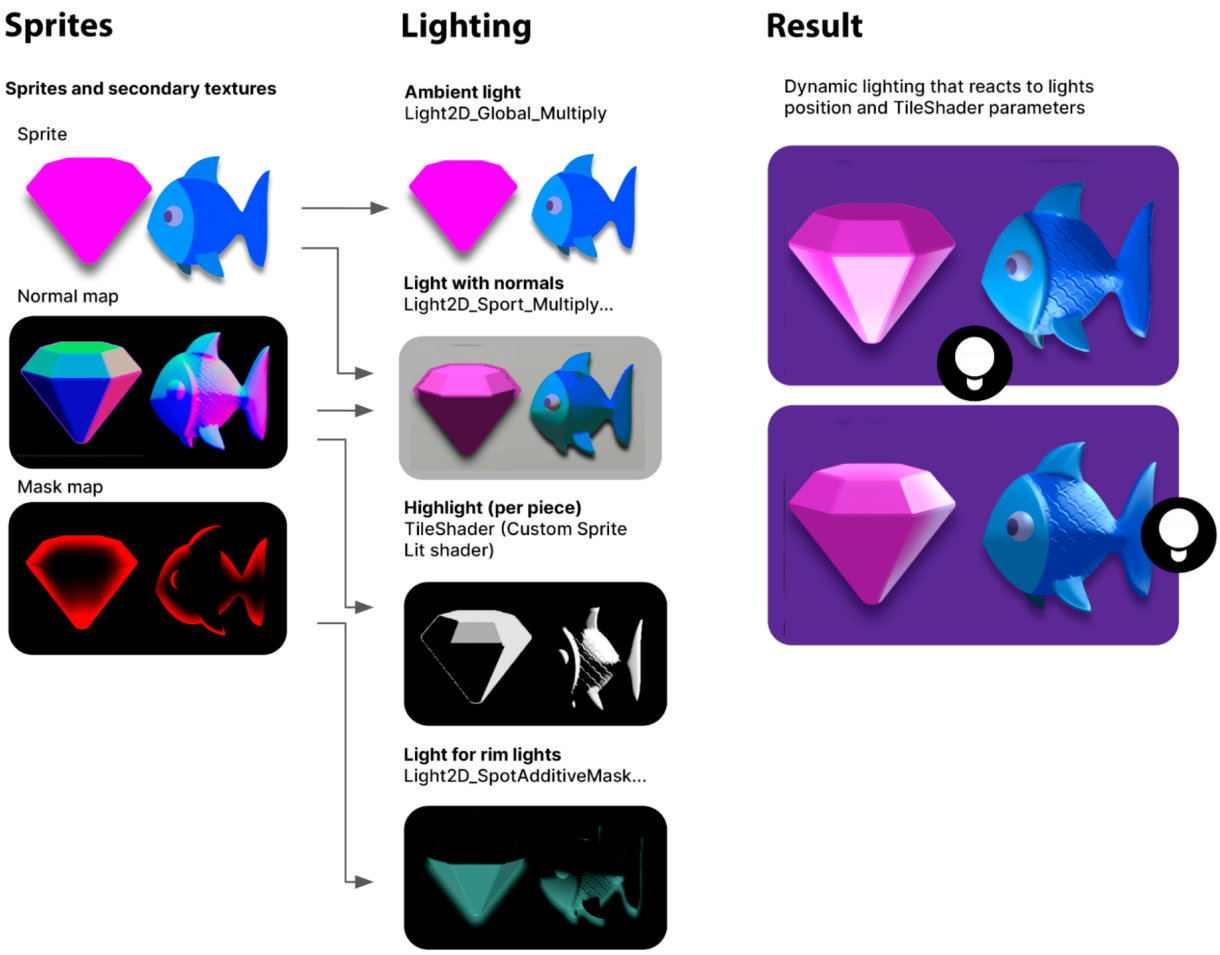
2D ライトシステムの標準的な使用法は、オブジェクトどうしや周囲の環境を照らすように適用することですが、独自のライティングシステムを作成するにはどうすればよいでしょうか。Shader Graph で利用可能な **Sprite Custom Lit** シェーダーを使用すると、2D ライトシステムによって生成されたライトマップを読み取り、スプライトで使用するためにこれらのマップに必要な効果を適用することができます。

Gem Hunter Match のこのサンプルでは、Sprite Custom Lit シェーダーを使用して、効率的に数百のピースを照らしています

その好例として、Gem Hunter Match のサンプルを見てみましょう。ゲームボードの各ピースに正確にライトが当たっています。これは、次のように組み合わせることで実現されています。

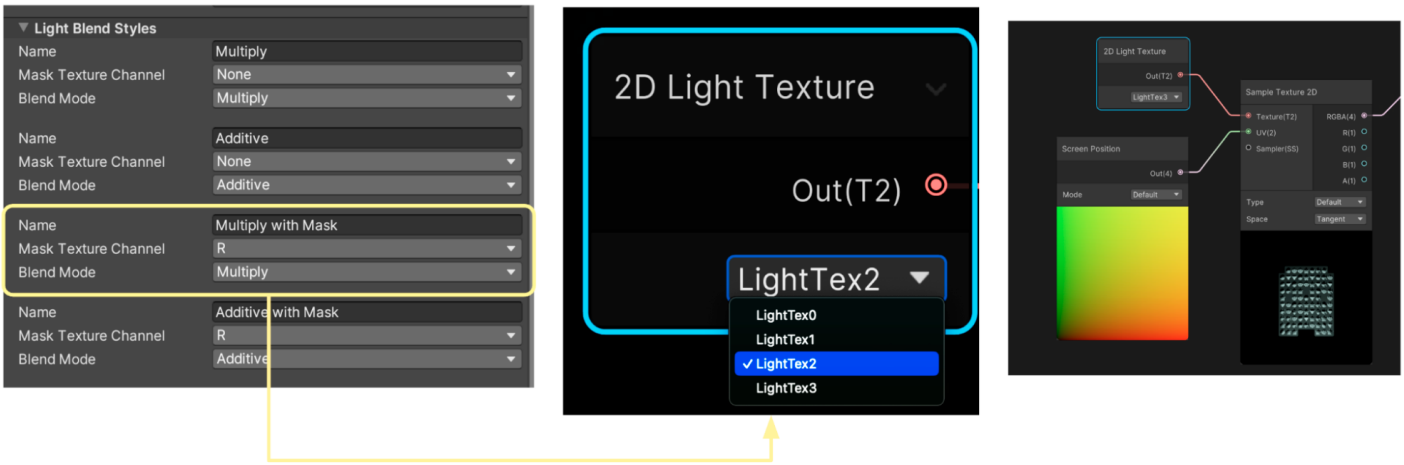
- 法線マップと光源の架空の位置を使用して、各ピースのカスタム照明を行います。Dot Product (ドット積) ノードを使用することで、ピクセルが受け取るライト量を計算できます。

- 2D ライトテクスチャにおけるライト情報のブレンド: リムライトやフレネル効果のために使用されるスポットライトと、ワールド空間の光源のための別のスポットライト。



スプライトの準備: この画像は、Gem Hunter Match のボードピーススプライトが法線マップとマスクマップを使用した動的なライティング効果のためにどのように準備されたかを示しています。

2D Light Texture ノードから各 Blending Style に対応する 2D Light Texture を詳しく見て、これを使用してこのシェーダーでオブジェクトにカスタム 2D ライティングを適用します。



Gem Hunter Match におけるカスタムスプライトライティングやその他の創造的なアイデアについて、以下のビデオで詳しく知ることができます。

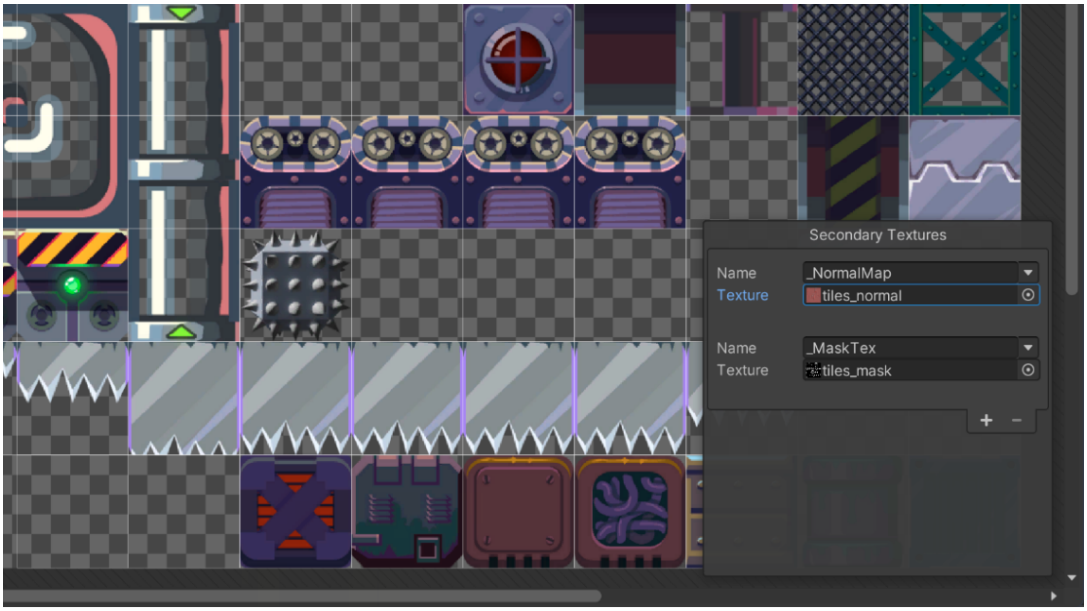
パート 1 と パート 2

2D レンダラーでの 2D ライトの使用

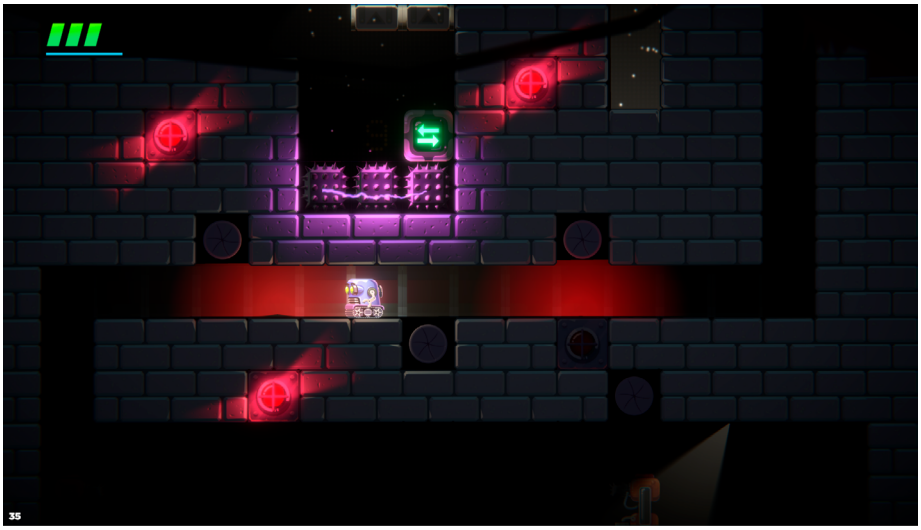
2D ライトはスプライトの法線マップやマスクマップと一緒に使用できますが、異なるレンダラーでも使用できます。では、見てみましょう。

2D タイルマップ

タイルマップで 2D ライトを使用するには、スプライトエディターの二次テクスチャモジュールを使用して、タイルとして使用されるスプライトに法線マップとマスクマップテクスチャを割り当てます。二次テクスチャは 2D ライトシステムによって自動的に使用されます。



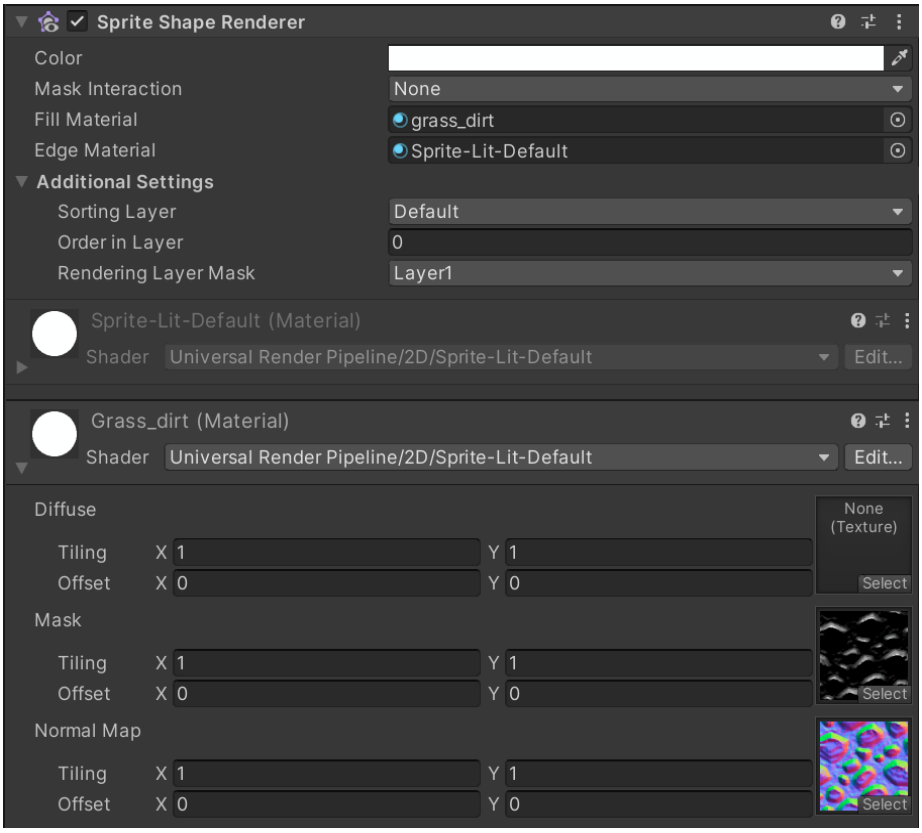
二次マップのあるタイルマップ



タイルマップ上で法線マップとマスクマップを使用した 2D ライト

2D スプライトシェイプ

Sprite Shape Renderer では、オブジェクトごとに 2 つの異なるマテリアルを使用します。1 つは **Fill Material** フィールドに割り当てられ、もう 1 つは **Edge Material** フィールドに割り当てられます。Edge Material はスプライトアセットの二次マップを使用します。Fill Material には、スプライトの代わりにタイル用に **Wrap** モードを **Repeat** に設定したテクスチャを使用します。二次テクスチャを設定するために、新しい Fill Material を作成して **Material Property Block** を有効にする必要があります。



法線マップを使用したスプライトシェイプのゲームオブジェクトの例

2D アニメーションキャラクターと二次テクスチャ

2D PSD Importer での二次テクスチャの設定は、法線マップに関する 1 つの小さな違いを除いて、同様に機能します。

2D PSD インポートキャラクターの法線マップとマスクマップを作成する最も簡単な方法は、基本キャラクターの .PSD ファイルから作業することです。

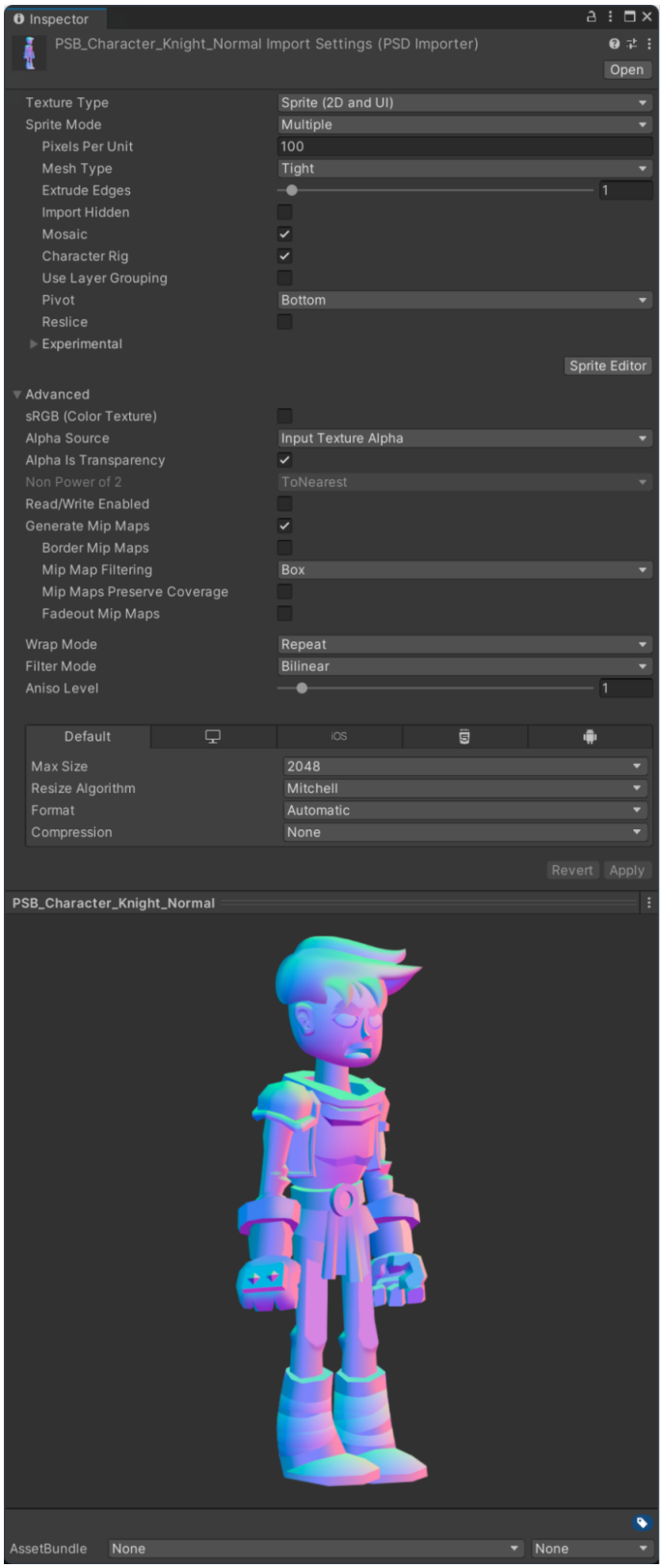
アニメーションキャラクターに法線マップを追加するために以下のステップを完了してください。

1. 基本キャラクターの .PSD ファイルを複製します。複製したファイルの名前に _normal などのサフィックスを追加して名前を変更します。

2. このファイルを任意の画像編集ソフトで開き、各レイヤーに法線マップをペイントします。ファイルを保存します。

3. 法線マップを含む PSD が Unity にインポートされたら、Texture Type を Sprite に設定し、Advanced 設定で sRGB (Color Texture) オプションのチェックを外す必要があります。法線マップには色の sRGB データは含まれておらず、角度の値のみが含まれています。

4. この PSD ファイルを基本キャラクターの二次テクスチャとして割り当てます。マスクマップを作成するには、このプロセスを繰り返し、複製したファイルに別のサフィックスを付け、ステップ 3 を飛ばします。



アニメーションキャラクターに法線マップを追加

3D を 2D として行うレンダリング

多くの Unity 製ゲームでは、ゲームプレイや環境に 2D と 3D の要素を組み合わせています。これは、3D プロジェクトで 2D グラフィックスを表現するシステムを構築することや、カスタムレンダリングで 2D に 3D を取り込むことなどが含まれています。チームは、ゲームの芸術的テーマ、制作パイプライン、パフォーマンス目標に応じて 2D と 3D を組み合わせることができます。特定の使用例として、次のようなものが挙げられます。

— さまざまな角度から表示されるアニメーション化されたキャラクターは、例えばアイソメトリックゲームで 3D で作成する方が簡単です。

— アニメーション、アクセサリー、またはカスタマイズの再利用は、3D キャラクターを使用することでより簡単になるかもしれません。

— 奥行き感を高めるために、2D キャラクターを使って 3D 背景を作成することができます。

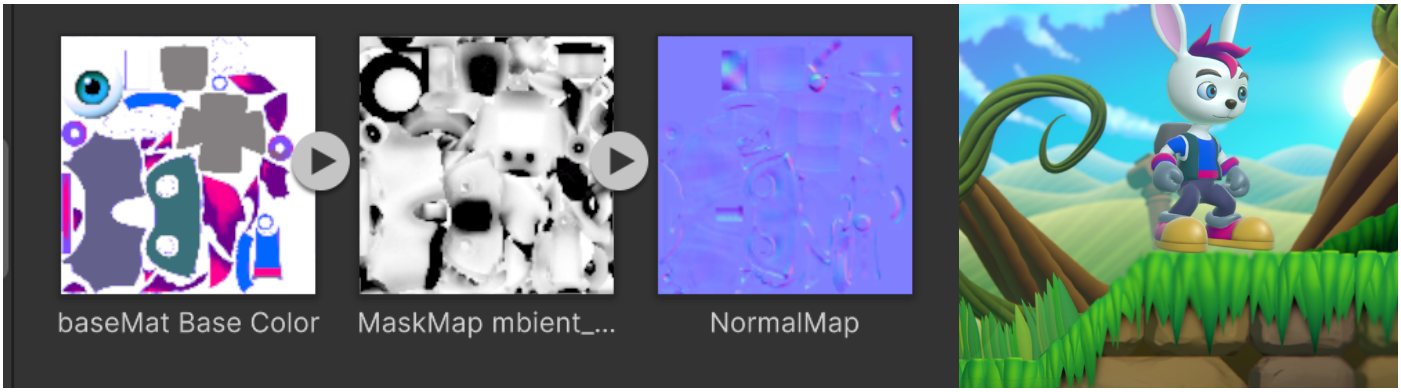
Unity 6.3 LTS では、3D アセットを 2D プロジェクトに取り込むための新しいワークフローが提供され、以下のようなメリットがあります。

— すべての 2D ツールが利用可能

— 3D アセットおよび 2D アセット全体で 2D ライトシステムとシェーダーマテリアルを使用することで、より統一感のある外観を実現

— 2D ワークフローに慣れた開発者にとって 3D の自然な統合が可能

— スプライト、タイルマップ、スプライトシェイプ、3D アセットを簡単にソート



Bunny Blitz の 3D キャラクターに使用されるテクスチャは、2D スプライトに追加できる二次テクスチャと同じです

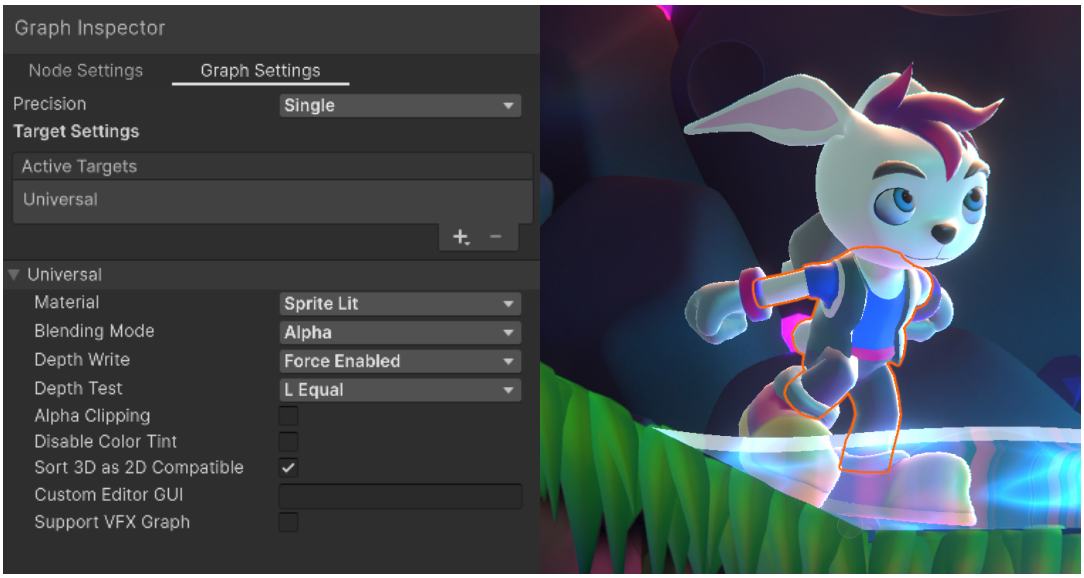
互換性

MeshRenderer と SkinnedMeshRenderer は、URP の 2D レンダラーを使用して 2D プロジェクトで使用できる互換性のある 3D レンダラーです。

互換性のあるマテリアル

Mesh2D-Lit-Default と **Mesh2D-Unlit-Default** が含まれており、3D アセットが 2D ライトによって照らされるようにします。これらのマテリアルに割り当てることができるテクスチャは、2D スプライト用のものと同じで、メインテクスチャ、法線マップ、マスクマップです。

さらに、**Sort 3D as 2D** 設定が有効な Shader Graph スプライトマテリアルは、3D アセットで使用でき、2D ライティングも適用されます。



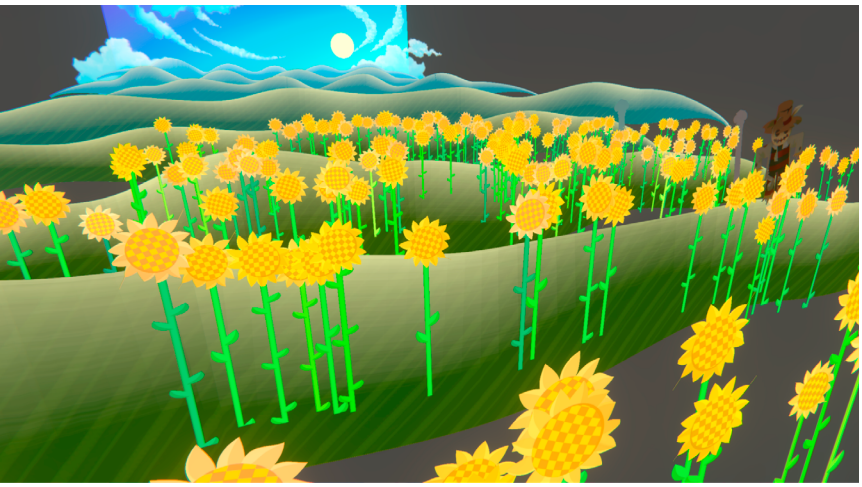
Sort 3D as 2D モードを有効にした Sprite Lit シェーダー

3D および 2D アセットのソート

プロジェクト内のオブジェクトのソートを計画する方法は以下の 2 つです。

1. Z 軸に要素を配置し、透視投影効果を利用したい場合
    - 3D オブジェクトと 2D オブジェクトを異なる平面に配置するだけで、正しくソートされます。
    - **注意:** 3D オブジェクトに 2D ライトを照射したい場合は、それらにはどのソートレイヤーのライトから光を照射するかを示す **Sorting Group** コンポーネントを設定する必要があります。
    - この技術を使用する利点の 1 つは、深度バッファに書き込むことで、被写界深度のようなポストプロセスエフェクトが利用可能になることです。

2. 2D プロジェクトのソートレイヤーシステムを使用してオブジェクトを配置したい場合
    - 3D オブジェクトを 2D ワールドに取り込み、Sorting Group コンポーネントを追加してください。**3D as 2D モード** をチェックすると、クリッピングを避けるためにフラット化され、他のスプライトのように動作します。



(左) Z 軸に配置された要素がある 2D シーンの画像。(右) Happy Harvest サンプルの画像。ソートレイヤーを使用して Bunny Blitz の 3D のウサギのキャラクターをスプライトにブレンドしています。

3D と 2D のマスキング

スプライトマスクは 3D と 2D の両方に互換性があり、任意のオブジェクトを任意の 2D レンダラーでマスクすることができます。

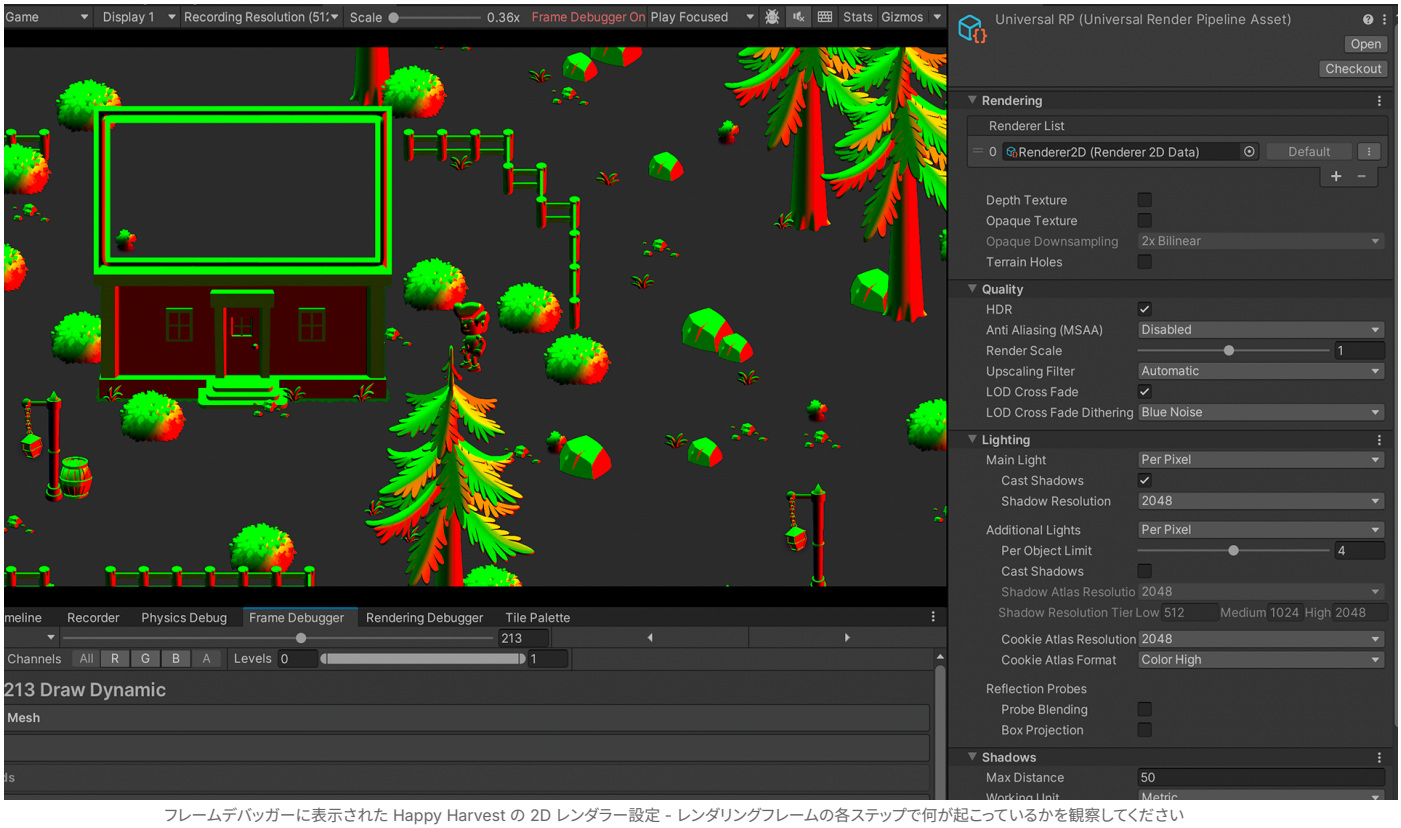
Mesh Renderer や Skinned Mesh Renderer、または 2D タイルマップ、スプライトシェイプ、スプライトレンダラーを含む任意の 2D レンダラー内で、**Mask Interaction** オプションに注意してください。マスク内、マスク外、またはマスクとして機能する任意の 2D レンダラー内でのみ表示されます。

Unity 6.3 LTS の新しい API を使用すると、ランタイムに任意の 3D または 2D オブジェクトのマスクインタラクションを変更できます。

ウサギはポータルの楕円形スプライト内でマスクされ、スプライト内でのみ表示されます。

Unite Barcelona 2025 の 基調講演 や 2D セッション で、新しい 2D/3D ワークフローについてさらに学ぶことができます。

2D 最適化のヒント



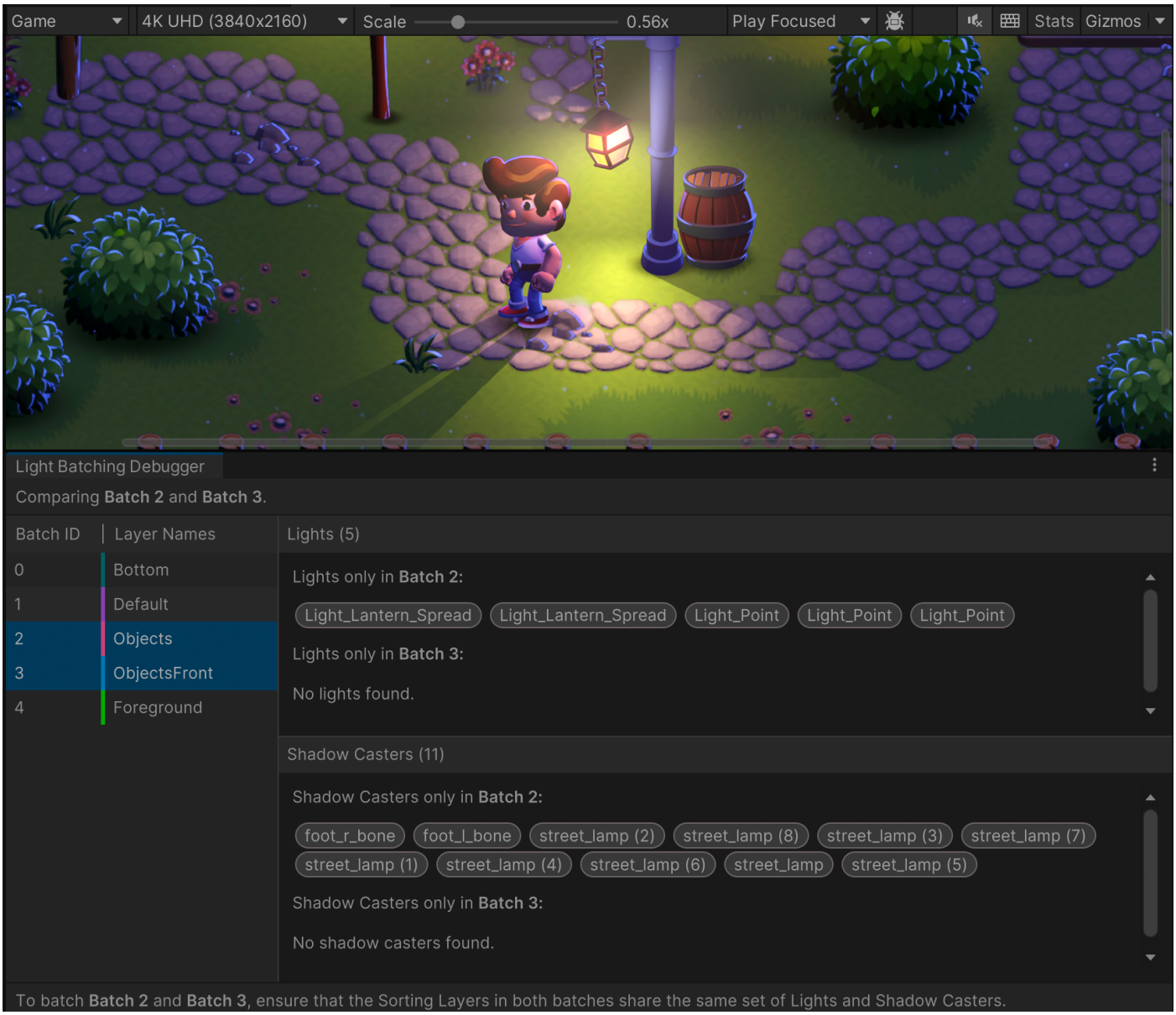
フレームデバッガーに表示された Happy Harvest の 2D レンダラー設定 - レンダリングフレームの各ステップで何が起こっているかを観察してください

2D ライトのための一般的な最適化

特にモバイルプラットフォームを対象にする場合、2D ライティングを使用する際に、ゲームにライトを追加するコストが一般的に懸念されます。常に、サポートされている最低スペックを含む実際の対象ハードウェアでテストしてください。そして、プロジェクトのパフォーマンスを向上させるために、一般的な最適化の以下のヒントを試してください。

- フィルレートをできるだけ低く保ってください。大型のライト 1 つは、小型のライト複数個よりもパフォーマンスが劣る場合があります。
- ライトはバッチ処理可能な場合に最高のパフォーマンスを発揮します。連続するレイヤーでライティング設定が同じライトはすべて一緒に描画できます。
- レンダースケールをできるだけ低く保ってください。レンダースケールによってライティングをレンダリングする際に使用されるテクスチャサイズを調整し、テクスチャサイズを小さくすることでレンダリングされるピクセル数を減少させます。
- 画面上に影を落とすライトの数を最小限に抑えてください。影を描画すると、無視できないパフォーマンスコストが発生します。
- 画面上のソートレイヤーと、さまざまなブレンドスタイルの数を最小化してください。ブレンドスタイルの描画を切り替える際にもパフォーマンスコストが発生する可能性があります。
- プロジェクトのニーズに合わせて、Max Light Render Textures と Max Shadow Render Textures の数を調整してください。数値が高いほど (限界まで) パフォーマンスが向上します、ただし必要なメモリも増加します。メモリとレンダリングのニーズに合った適切な数値を見つける必要があります。
- 法線マップを使用する場合は、ライトの標準ライティングのQuality を Accurate ではなく Fast に設定してください。
- 2D シャドウは、少数のライトにのみ使用してください。

2D ライトバッチングデバッガーの使用

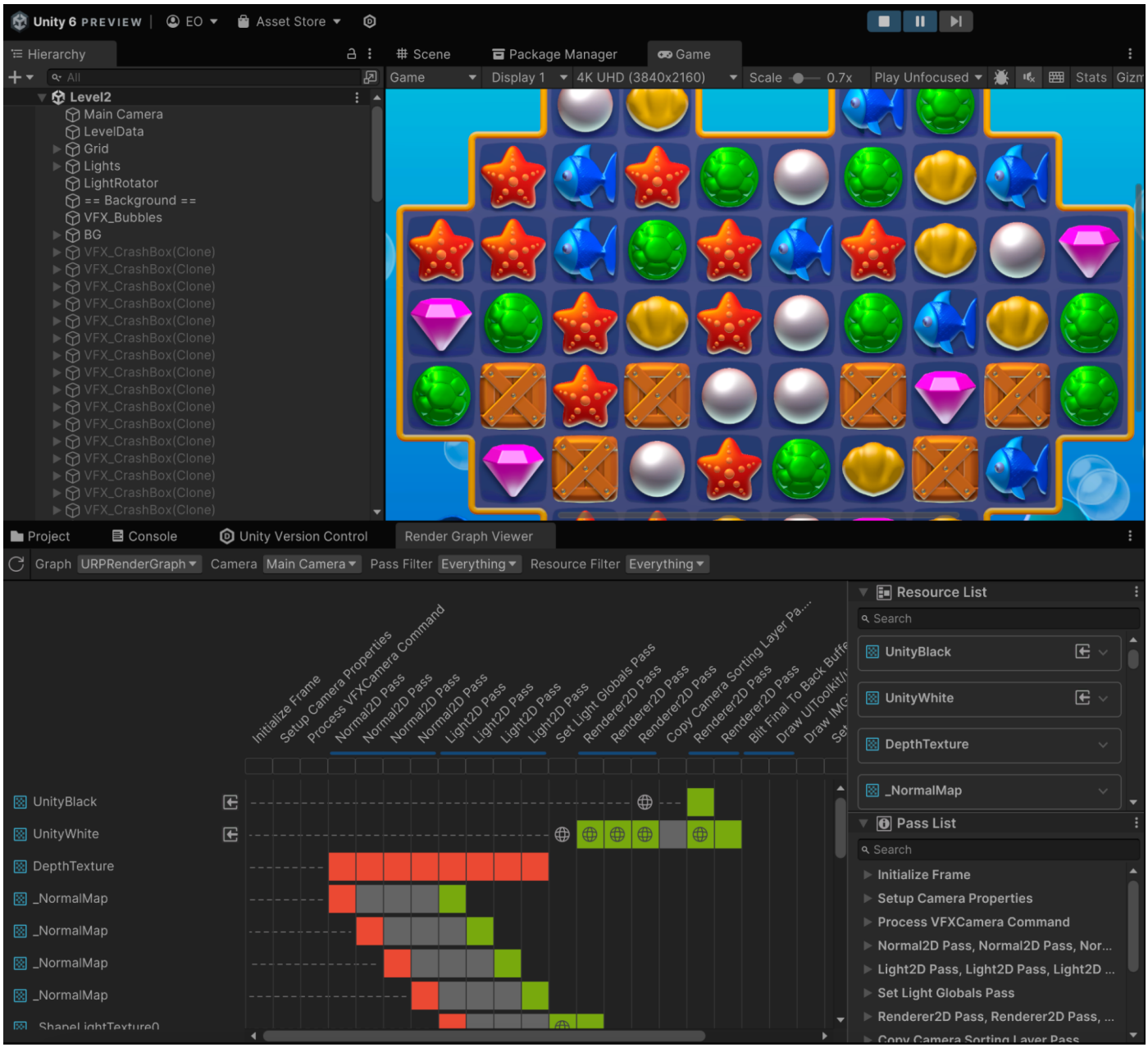


— 2D Light Batching Debugger を使用して、Unity がシーン内で対象とするソートレイヤーに従って 2D ライトと Shadow Caster 2D コンポーネントをどのようにバッチ処理するかを視覚化します。

— デバッガーによって隣接するバッチが比較され、各ソートレイヤーを対象とするライトやキャスターがハイライトされ、Unity がソートレイヤーをバッチ処理できるように追加または削除する必要があるものが表示されます。

— 全体として、デバッガーはライトと影がソートレイヤーにどのように影響するかについてより良い判断を下すことで、ゲームのパフォーマンスを向上させるために役立ちます。

レンダーグラフによる 2D レンダラーの使用

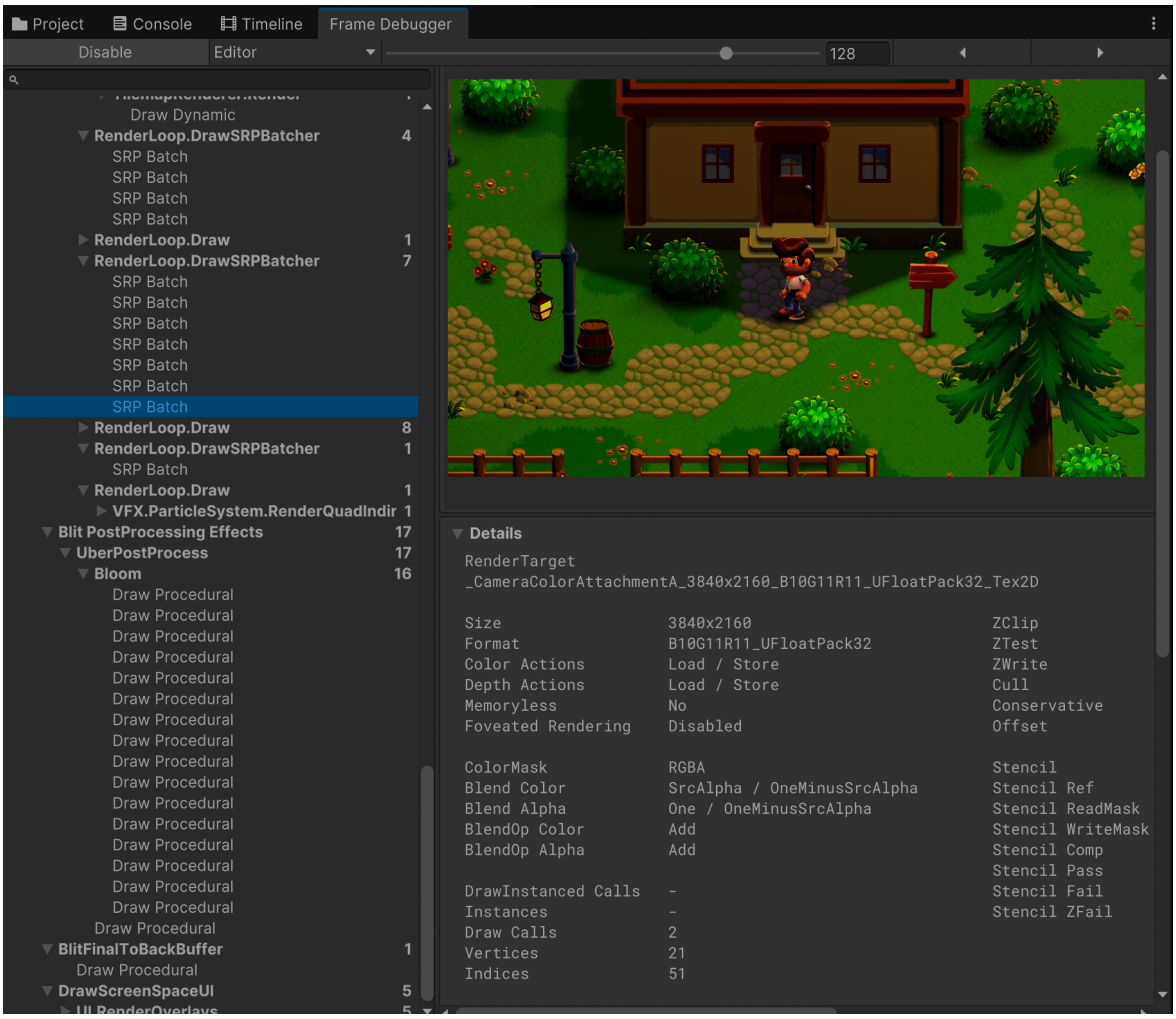


— レンダーグラフは、レンダーパスの数、および各レンダーパスが使用するメモリと帯域幅を最小限に抑えるように、レンダーパスを自動的に最適化します。

— フレームで使用しない余分なリソースの割り当てを回避します。

— 複数のレンダーパスを単一のレンダーパスにマージします。

SRP バッチャーの使用

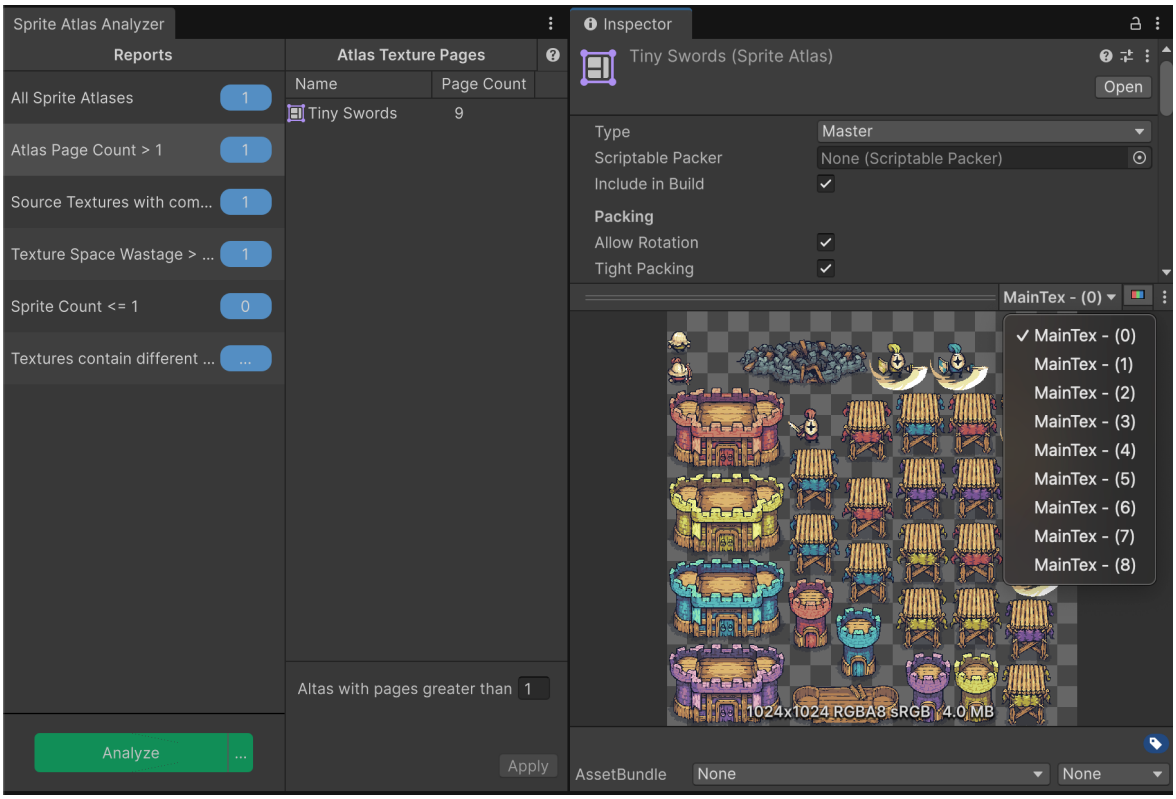


— SRP バッチャーは、以前は CPU に負荷をかけていたタスクを GPU メモリに移動し、ゲームプレイのために CPU の負荷を軽減します。

— メッシュが密なスプライトやスキンスプライトの場合、通常、SRP バッチャーによりパフォーマンスが向上します。

  — **注意**: 完全な矩形 (クワッド) または頂点が非常に少ないスプライトの場合、動的バッチ処理 (SRPバッチャーの無効化) が推奨されます (APIを使用してオブジェクトごと)。

スプライトアトラスアナライザーの使用



Sprite Atlas Analyzer ウィンドウの Atlas Page Count を確認

Unity 6.3 LTS の新機能である Sprite Atlas Analyzer ウィンドウは、スプライトアトラスのパフォーマンスの問題を特定することに役立ちます。プロジェクト内のすべてのスプライトアトラスに含まれる 2D テクスチャアセットの詳細情報が提供されます。

使用するには、パッケージを インストール し、**Window > Analysis > Sprite Atlas Analyzer** に移動して起動します。ウィンドウが開いたら、**Analyze** ボタンをクリックしてプロジェクト内のスプライトアトラスの内容の分析を始めます。

Sprite Analyzer には、以下の 6 つの ビルトインレポート が用意されています。

- All Sprite Atlases (すべてのスプライトアトラス)
- Atlas page count (アトラスページ数)
- Source textures with compression in Sprite Atlas (スプライトアトラス内の圧縮されたソーステクスチャ)
- Texture Space Wastage (テクスチャスペースの無駄)
- Sprite count (スプライト数)
- Textures contain different secondary texture count in Sprite Atlas (スプライトアトラス内のさまざまな二次テクスチャ数を含むテクスチャ)

詳細については、Unite 2025 の 2D ツールに関するセッションの Sprite Atlas Analyzer セクション を参照してください。

タイルマップの最適化

Happy Harvest の画像 - 右側では、レンダリングデバッガーでオーバードローモードが有効になっており、明るい領域は重なり合うピクセルを示し、暗い領域は重なりが少ない場所を示しています。

- シーンに多くのスプライトが含まれている場合は、タイルマップの使用を検討してください。これにより、多くの (オーバーヘッドがある) Sprite Renderer を単一の Tilemap Renderer に置き換えることができます。
- バッチ処理のためにレベルコライダーを Static に設定します。
- シーンで一緒に配置される可能性の高いスプライトをスプライトアトラスに配置します。
- スプライトエディターの Custom Outline オプションを使用してスプライトのメッシュを簡素化します。
- (Edit Spline ボタンがオンのときに表示される) Cache Geometry オプションを有効にしてスプライトシェイプジオメトリをキャッシュします。

アニメーションの最適化

- Burst パッケージをインストールして 2D アニメーションのパフォーマンスを向上させます。
- スケルトンを作成する際は、必要以上に多くのボーンやスプライトを使用しないようにしてください。
- 使用する頂点をできる限り少なくして、スケルトンメッシュを簡素化します。
- アニメーターで Culling を有効にします。パフォーマンス向上のため、アニメーションキャラクターがオフスクリーンの場合はこのオプションは無効になります。
- 背景の鳥やハエ、小動物など、小さなオブジェクトや大量に存在するキャラクターにはスケルタルアニメーションを使用しないようにしてください。代わりに、フレームごとのアニメーションやシェーダーなどのさまざまな技術を使用してください。
- フレームごとのアニメーションを使用する際は、フレーム数やテクスチャサイズをできるだけ小さく保つようにしてください。

VFX およびポストプロセスエフェクト

— ポストプロセスエフェクトの使用はできるだけ抑えてください。

— 可能な限り、Bloom、Chromatic Aberration、Color Grading、Lens Distortion、Vignette などの負荷の少ないエフェクトを使用してください。

— Film Grain と Panini Projection はよりコストがかかります。

— Bloom の High Quality Filtering を無効にします。

— オブジェクトプールを使用すると、VFX アセットのインスタンス化と破棄を避けることができます。ゲームオブジェクトを単純にアクティブ化および非アクティブ化します。

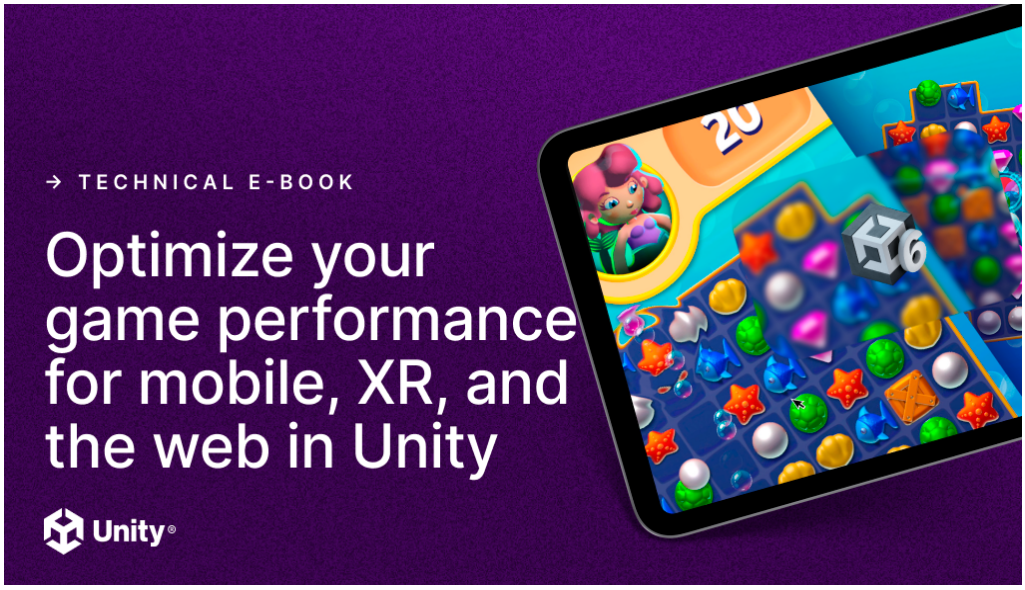
— Shader Graph では、可能な限り半精度を使用してください。

Unity のパフォーマンスツールの使い方

Unity 6 e-book は、Unityプロジェクトのための高度なパフォーマンス最適化スキルの開発に役立ちます。

70 ページ以上にわたる本ガイドには、Unity でのアプリケーションのプロファイリング、メモリの管理、消費電力の最適化に関する外部および社内の Unity 専門家からの高度な知識とアドバイスがまとめられています。

ガイドをダウンロード



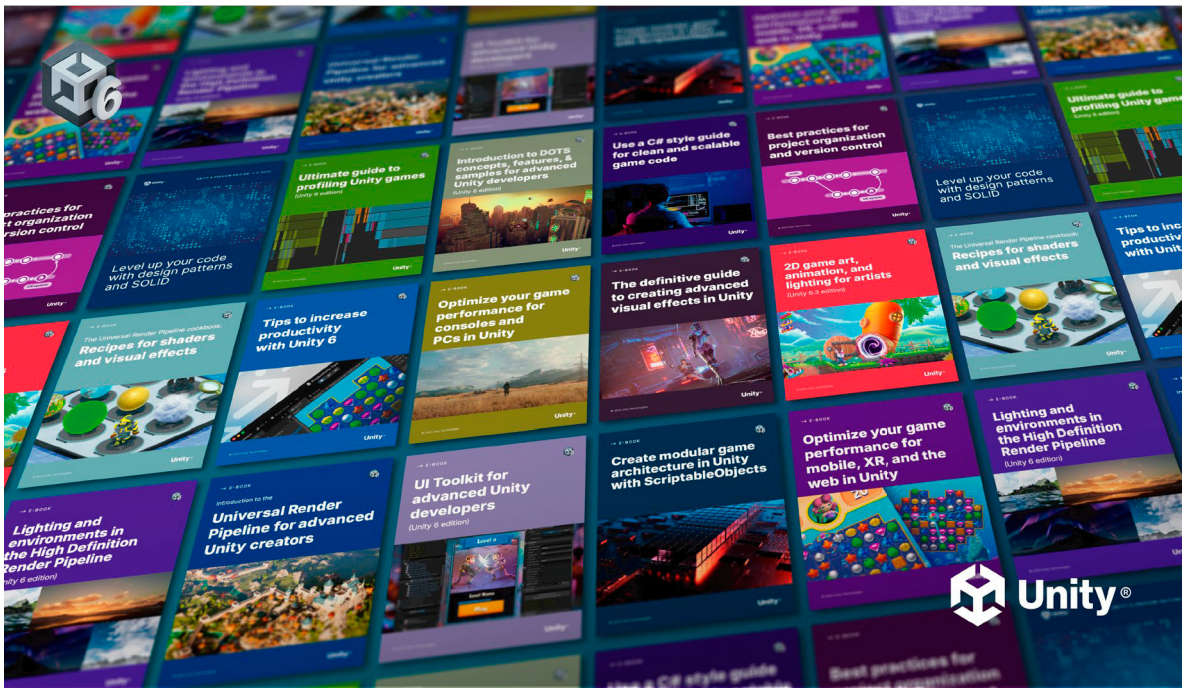
このガイドは、幅広いデバイスで高いパフォーマンスを発揮するアプリケーションを立ち上げる手助けをし、ローンチ時およびその後の成功の機会を最大限に引き出すことができます。

ガイドをダウンロード

コードのアーキテクチャやアートアセットを最適化する方法を学び、対象のハードウェア上でスムーズに動作し、制限内でゲームを立ち上げる方法を学びましょう。

ガイドをダウンロード

まとめ



Unity のベストプラクティスのハブ では、上級 Unity 開発者およびクリエイター向けのさまざまな e-Book をダウンロードできます。業界のエキスパート専門家と Unity のエンジニアやテクニカルアーティストが作成した 30 以上のガイドから、必要なものを選べます。

さらに、Unity Blog、Unity Discussions、Unity Learn、および #unitytips ハッシュタグを通じて、より多くのヒントやニュースを見つけることができます。

付録：2D 視覚効果

(左) Unity デモ Dragon Crashers – 2D Sample Project の画像、(右) Archanor VFX による Epic Toon FX の画像。パッケージは両方とも Asset Store で入手可能。

視覚効果（VFX）は、見栄えの良いゲームにおいて重要な要素であり、プレイヤーの体験にとっても不可欠です。VFX は、環境の危険や回復ゾーンなどのゲームイベントを伝えるだけでなく、アクションシーンの最後に大きな炎の爆発が起こるなど、プレイヤーのうまく行った操作に対して　視覚的に報酬を与えるための役割も果たします。

Unity で 2D 視覚効果を作成するには、いくつかの異なる方法があります。例えば、火の効果を作成するには、次のような方法があります。

- 炎をフレームごとにアニメーション化します。
- Shader Graph で作成したシェーダーを使用してスプライトをアニメーション化します。
- 炎のようなパーティクルを生成するには、CPU 上で動作する ビルトインパーティクルシステム、または GPU のパワーを活用して数百万個ものパーティクルを生成する VFX Graph のいずれかを使用できます (どちらのシステムも同じプロジェクトで使用できるため、特定の効果に最適なものを選択できます)。

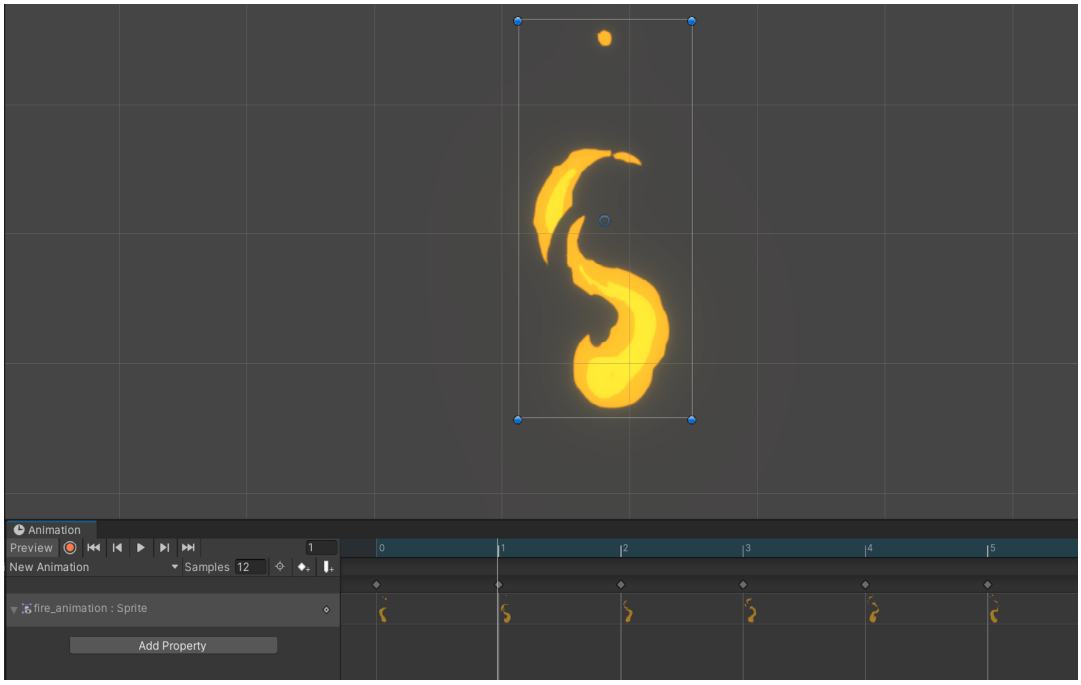
Unity の 2D プロジェクトで特殊効果を作成する前に考慮すべきいくつかの重要な点があります。

- **カメラ**: 透視投影カメラを使用する場合、効果は 3D ゲームのように三次元空間で発生します。平行投影カメラを使用する場合、システムのレンダラーを設定して適切な ソートレイヤー を使用する必要があります。カメラの視点については こちら を参照してください。
- **ポストプロセス**: ユニバーサルレンダーパイプライン (URP) を選択した場合、2D プロジェクトに ポストプロセスエフェクト を適用できます。例えば、URP の Volume フレームワークを介してパーティクルの効果にブルーム効果を加えることができます。
- 効果はさまざまな方法で実現できます。例えば、効果を表すゲームオブジェクトは、カスタムシェーダーを使用したスプライトや複数のパーティクルシステムなど、さまざまなシステムを使用するいくつかの子ゲームオブジェクトを持つことができます。
- **アニメーションクリップ**: アニメーションクリップ は、滑らかなスプライトベースのアニメーション効果を調整するために役立ちます。Unity のスケルタルアニメーション システムを (控えめに) 使用して、スプライトアニメーションの一部を制御することもできます。
- **Unity Asset Store**: プロジェクトに視覚効果を加えるための有利なスタートを切るには、Unity Asset Store にアクセスすることをお勧めします。多くの既製の 2D 効果が用意されています。

このセクションでは、VFX Graph に焦点を当て、Dragon Crashers、Happy Harvest および Gem Hunter Match サンプルの特定の効果を参照しながら、これらのすべての方法を扱います。

2D VFX のための Unity ツールセット

フレームごとのアニメーション化



炎のフレームごとのアニメーション化

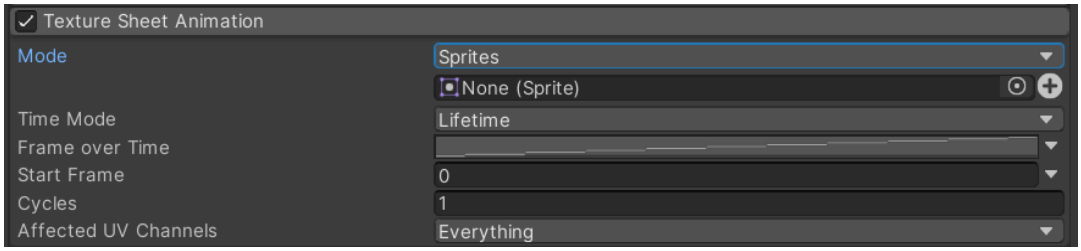
フレームごと (またはフリップブック) のアニメーションを使用すると、Unity ですばやく効果を加えることができます。フレームをスプライトとしてインポートし、Animation ウィンドウで Sprite プロパティを時間の経過とともにアニメーション化するだけです。また、すべてのフレームスプライトを選択してシーンにドラッグすることもできます。これにより自動的にアニメーション化されます。

フレームのインポートは高速ですが、アニメーション化するために各フレームを描画する作業は高速ではなく、このプロセスには、あなたが使えるよりも多くの時間とスキルが必要になることがあります。時間を節約するために、他のアプリから VFX をフレームとしてエクスポートすることもできます。

時間を節約する別の方法として、Project ウィンドウからすべてのアニメーションフレームアセットを選択し、階層ビューまたはシーンにドラッグすることができます。その後、以前に選択した画像のシーケンスを使用するアニメーションを含む新しいゲームオブジェクトが作成されます。

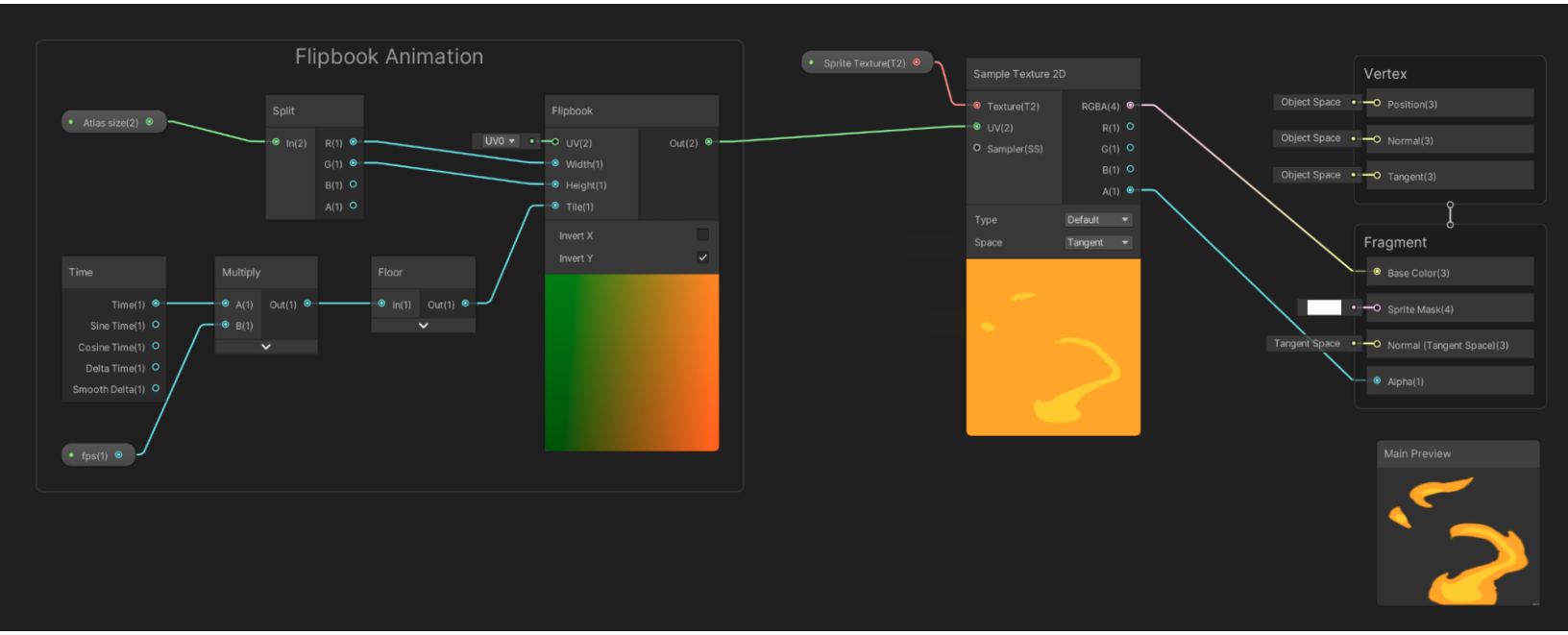
フリップブックアニメーション技術は非常にパフォーマンスが高くなるため、画面上に多くのインスタンスの効果を表示したいときに使用してください。高解像度のフレームの多くは、より多くのメモリを必要とすることに留意してください。

フレームごとのアニメーションは、パーティクルシステム でも使用可能です。ここでは Texture Sheet Animation (テクスチャシートアニメーション) と呼ばれています。



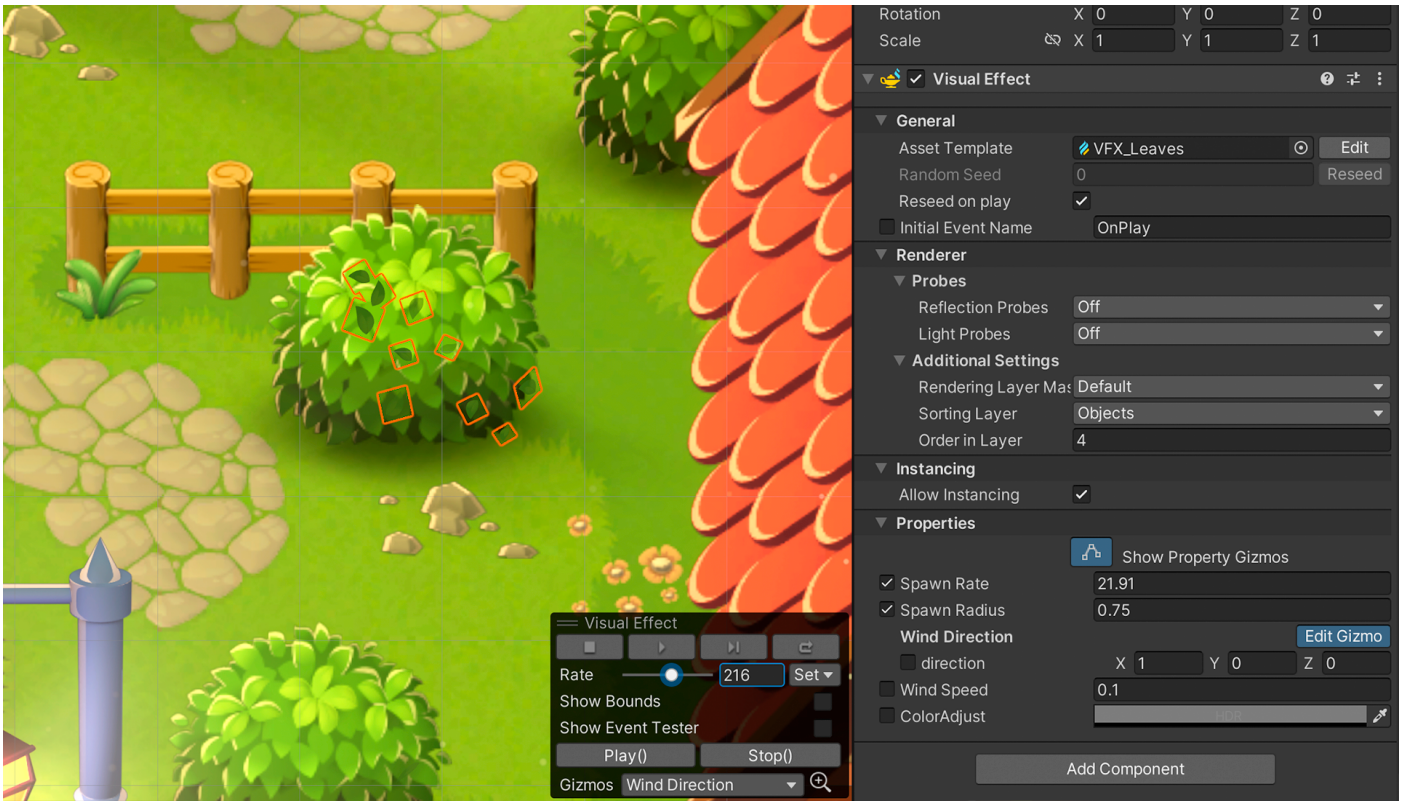
パーティクルシステムでのフレームごとのアニメーションの設定

Shader Graph では、テクスチャの UV 位置を時間経過とともにアニメーション化することで、フリップブックアニメーションを使用することもできます。これにより、フレームが動いているような錯覚を生み出すことができます。



Shader Graph におけるフリップブックアニメーション

2D 用の VFX Graph



Happy Harvest では、キャラクターが通り過ぎるときに葉が茂みから落ちる効果を作成するために VFX Graph が使用されています

VFX グラフ はノードベースのエディターです。このエディターを使用すると、テクニカルアーティストや VFX アーティストは、シンプルな一般的なパーティクルの動作から、パーティクル、ライン、リボン、トレイルなどを含む複雑なシミュレーションまで、ダイナミックで GPU によって加速されたパーティクル効果をデザインできます。

VFX Graphシステムは HDRP と URP の両方に対応しており、Unity の 2022 LTS 以降のバージョンで Shader Graph を使用する 2D レンダラーと互換性があります。

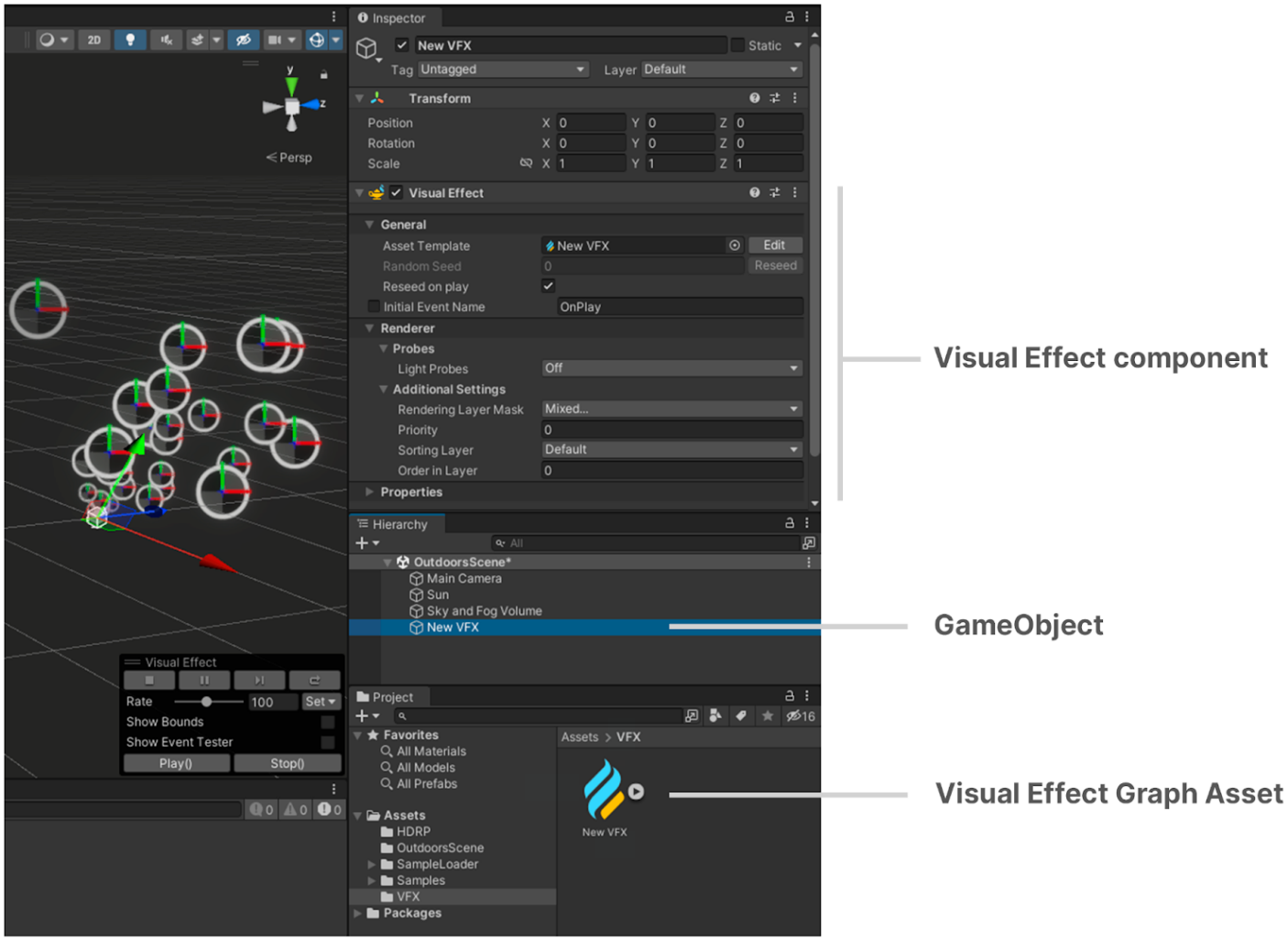
プロジェクトに新しいグラフを作成するには、VFX Graph のドキュメントを参照することをお勧めします。VFX Graph の基本的なアウトラインを見てみましょう。これにより、2D ゲームおよび 3D ゲームで効果を作成するための強力なツールである理由を理解するために役立ちます。

VFX Graph のビジュアルエフェクトは、次の 2 つの部分で構成されています。

— シーン内のゲームオブジェクトに付属しているビジュアルエフェクト (VFX) コンポーネント

— プロジェクトレベルに存在するビジュアルエフェクト (VFX) グラフアセット

Unity は各 VFX Graph を Assets フォルダーに保存するため、各アセットをシーン内の Visual Effect コンポーネントに接続する必要があります。ランタイム時には、異なるゲームオブジェクトが同じグラフを参照できることに注意してください。

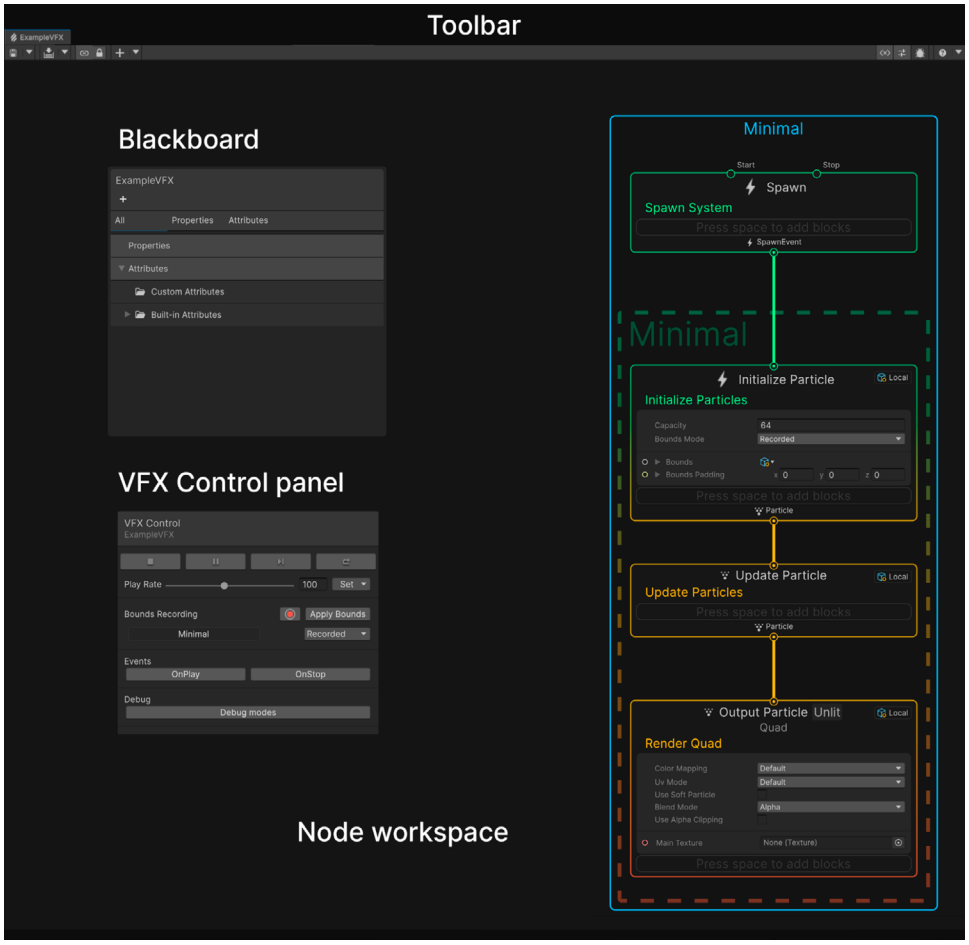


VFX Graph アセットと Visual Effect コンポーネント

VFX Graph ウィンドウ

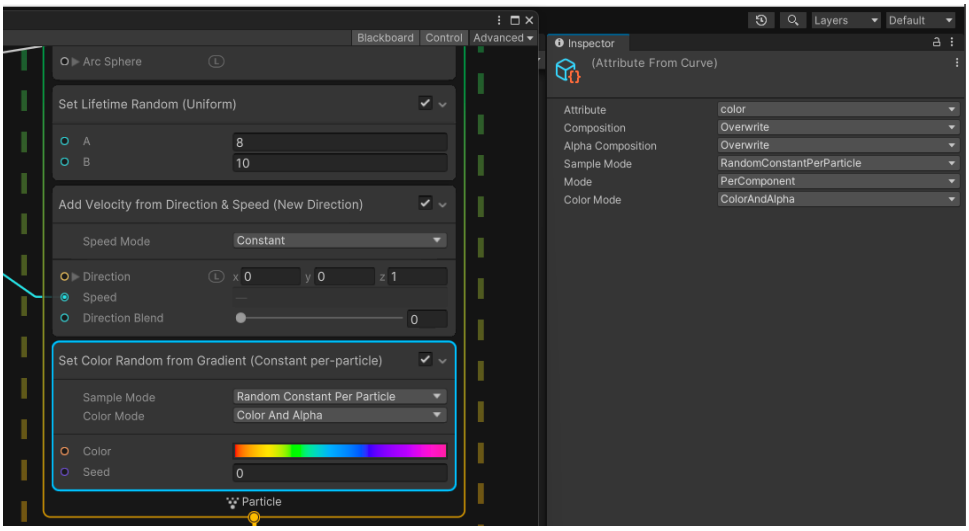
ウィンドウのレイアウトには以下が含まれます。

- **Toolbar**:グローバル設定へのアクセスとパネルの切り替えを行います
- **Node workspace**:VFX Graph の作成と編集を行います
- **Blackboard**: グラフ全体で再利用可能な属性とプロパティを管理します
- **VFX Control Panel**: 付属しているゲームオブジェクトの再生を変更します



VFX Graph ウィンドウ

エディターのレイアウトに Inspector 用のスペースを空けておいてください。グラフの一部を選択すると、パーティションオプションやレンダリング状態などの特定のパラメーターが表示されることがあります。



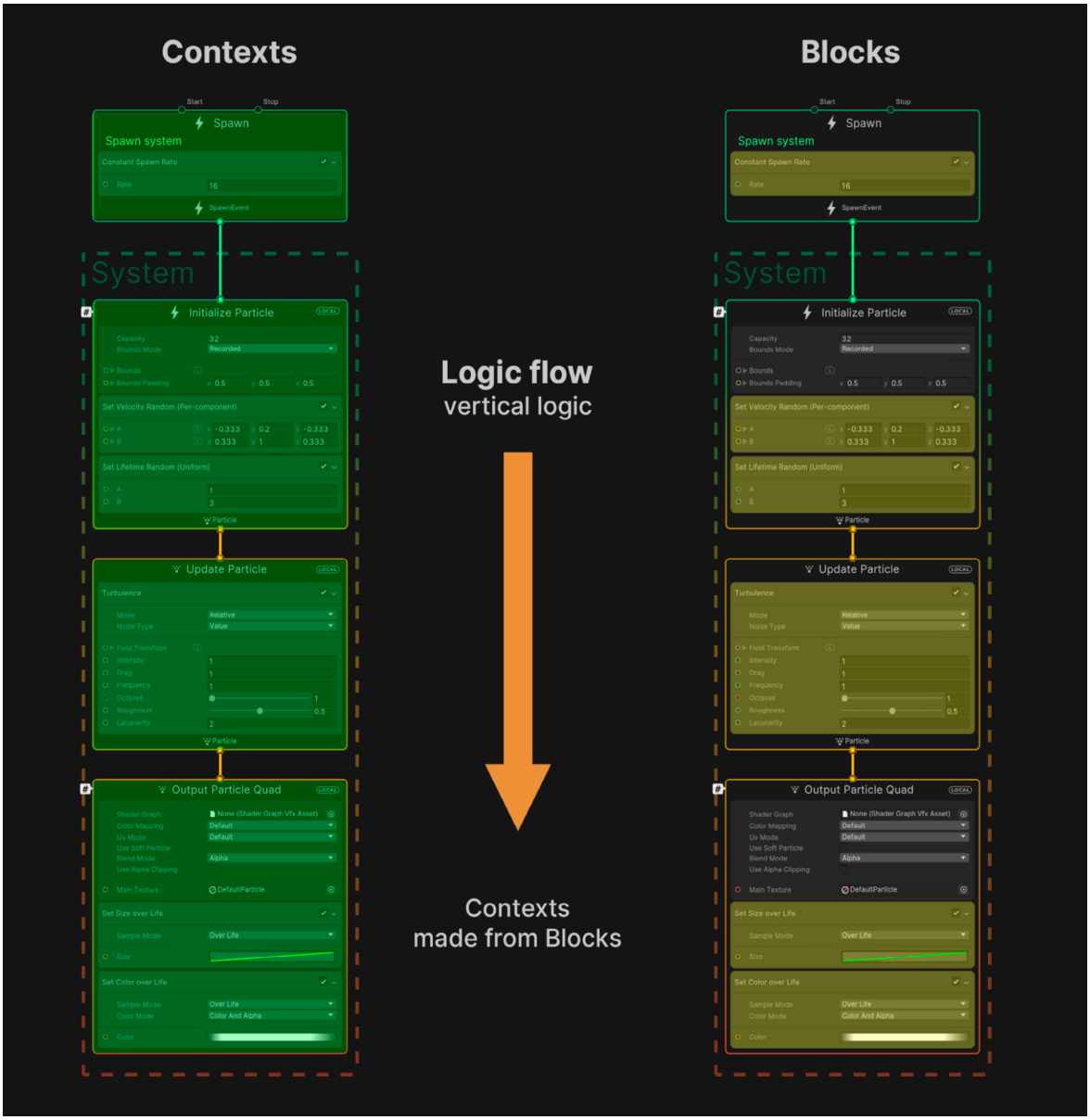
Inspector を使用した特定のパラメーターの変更

グラフロジック

ビジュアルエフェクトは、ウィンドウのワークスペース内のノードのネットワークから構築する必要があります。VFX グラフは、この章で後述される Shader Graph のような他のノードベースのツールに似たインターフェースを使用します。

VFX Graph には、システムと呼ばれる 1 つ以上の垂直スタックがあります。システムは、グラフのスタンドアロン部分を定義し、いくつかのコンテキストを格納します。システムは、システムを構成するコンテキストを囲む点線によって表されます。

各コンテキストは個々のブロックからなり、パーティクルやメッシュの属性 (サイズ、色、速度など) を設定できます。複数のシステムを 1 つのグラフ内で連動させ、最終的なビジュアルエフェクトを作成できます。



グラフの垂直ロジックは下方向に流れます。

VFX Graph 内の 出力コンテキスト は、効果がどのようにレンダリングされるかを定義します。出力パーティクルクアッド のようなメッシュ用の出力コンテキストにおり、Shader Graph を追加するオプションが提供されます。Lit または Unlit 出力のいずれから始めても問題ありません。ライティングモデルは Shader Graph に適応するように変更されます。以下に示すデフォルトのシェーダーを使用するか、独自の Shader Graph ベースのシェーダーを使用できます。

2D における VFX グラフのミニヒント

URP の 2D プロジェクトでは、以下の 3 つのスプライトベースの Shader Graph のいずれかを使用できます。

— Lit: 2D ライトシステムからライトを受けます。

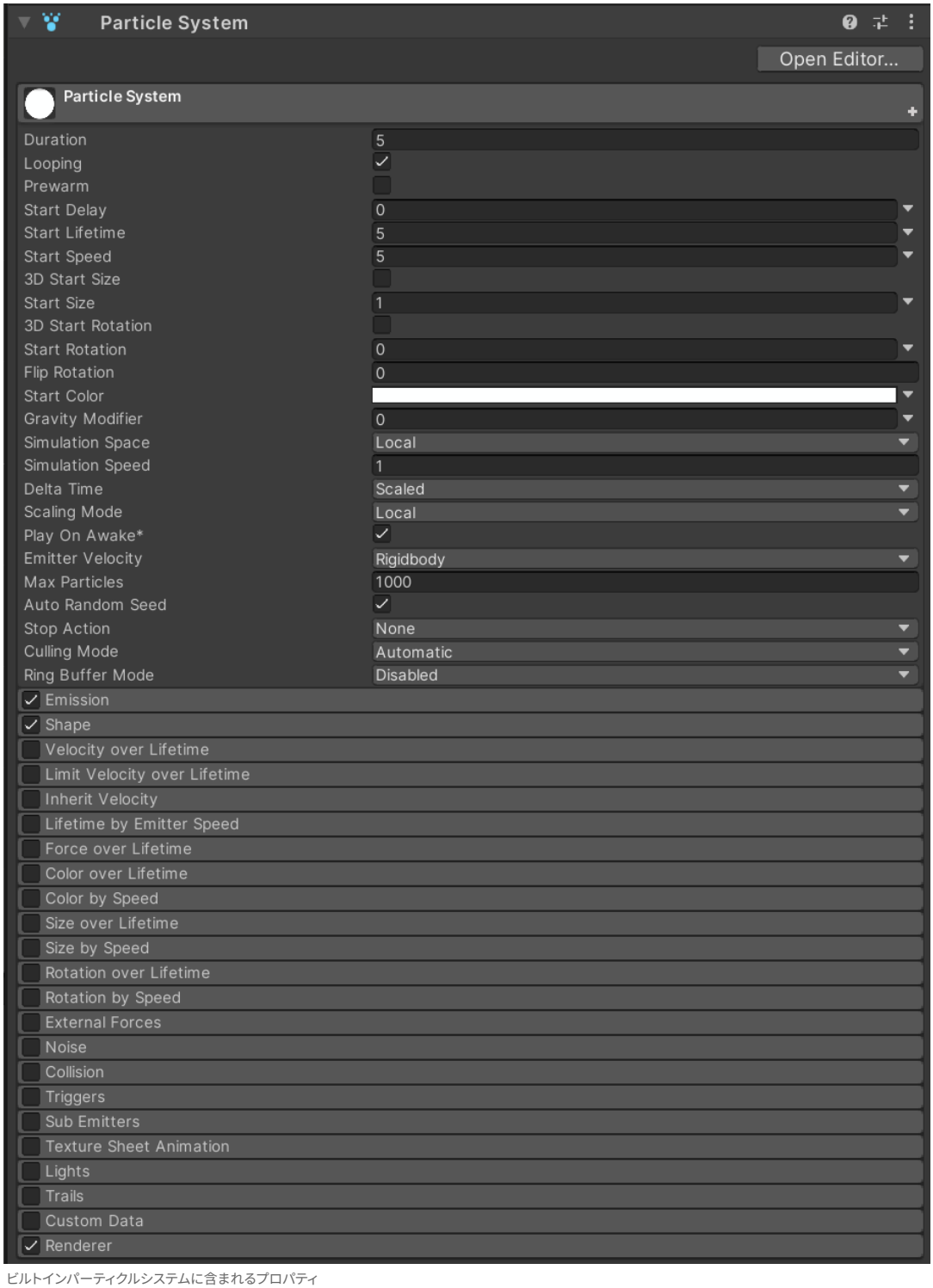
— Unlit: 2D ライティングシステムの影響を受けません。

— Custom: カスタム 2D ライティングモデルを実装する場合に使用します。

Shader Graph 内の SRP ターゲットで**Support VFX Graph** オプションを確認するようにしてください。

Shader Graphで公開されたプロパティは、VFX Graph に表示されます。必要に応じてそれらをオーバーライドできます。例えば、VFX Graph パーティクルシステムが時間の経過とともにシェーダーの色を変化させるように制御したい場合などです。

ビルトインパーティクルシステム



ビルトインパーティクルシステムに含まれるプロパティ

ビルトインパーティクルシステムを使用すると、単一の視覚効果を達成するために、多くの小さな画像やメッシュを表示およびアニメーション化することを可能にします。サイズ、速度、色、回転などのパーティクル

プロパティによって、特定の事前定義されたルールとランダム化を使用して時間の経過とともにアニメーション化できます。これにより、火、爆発、煙、魔法の呪文などの動的効果を作成できます。

メニューオプション **GameObject > Effects > Particle System**を選択して、新しいパーティクルシステムを作成します。

パーティクルシステムのメインモジュールには、システム全体に影響を与えるグローバルプロパティが含まれています。これらのプロパティのほとんどは、新しく作成されたパーティクルの初期状態を制御します。重要なプロパティには次のものが含まれます。

- 継続時間:システムがどのくらいの時間実行されるか
- ループ処理:システム全体が永遠にループするかどうか
- スピード:パーティクルの初期速度
- 開始サイズ:パーティクルの初期サイズ
- Color: パーティクルの最初の色
- 重力:パーティクルに適用される重力

ビルトインパーティクルシステムに含まれるすべてのプロパティやモジュールについて学ぶには、ドキュメントのパーティクルシステムモジュールコンポーネントリファレンスセクションを参照してください。

パーティクルシステムは、互いにネストすることもできます。ネストすると、システムが同時に再生され、非常に複雑な効果を実現できます。例えば、爆発を作成するために、煙用のシステム、小さな火花用のシステム、飛んでいる破片用のシステムを組み合わせることができます。

パーティクルシステムは、ゲームのほぼすべての VFX を作成できる非常に強力なツールです。HDR の Unlit シェーダーとブルームのポストプロセスを使用して、効果に光を加えることもできます。すべてのプロパティをテストしてください。最もシンプルなパーティクルでも、ゲームを洗練させることができます。

ゲーム制作プロセスを大幅に加速化するために、Asset Store で事前制作されたパーティクル VFX をチェックしてください。

Shader Graph

VFX を作成するもう 1 つの方法は、シェーダーを使用することです。シェーダーはグラフィックスプロセッサーのための一連の指示であり、例えば、ピクセルの色や頂点の位置を計算することができます。

Shader Graph を使用すると、視覚的にシェーダーを構築できます。コードを書く代わりに、グラフフレームワーク内でノードを作成して接続します。Shader Graph により、変更を反映する即時フィードバックが提供されます。シェーダー作成に不慣れなユーザーにもとても簡単です。

Shader Graph を使用すると、次のことを実現できます。

- UV をワープさせてアニメーション化する
- 表面の外観を連続的に変更する

— 画像編集アプリケーションに似た画像フィルターを追加する

— オブジェクトの表面を、ワールドの位置、法線、カメラからの距離などの情報に基づいて変更する

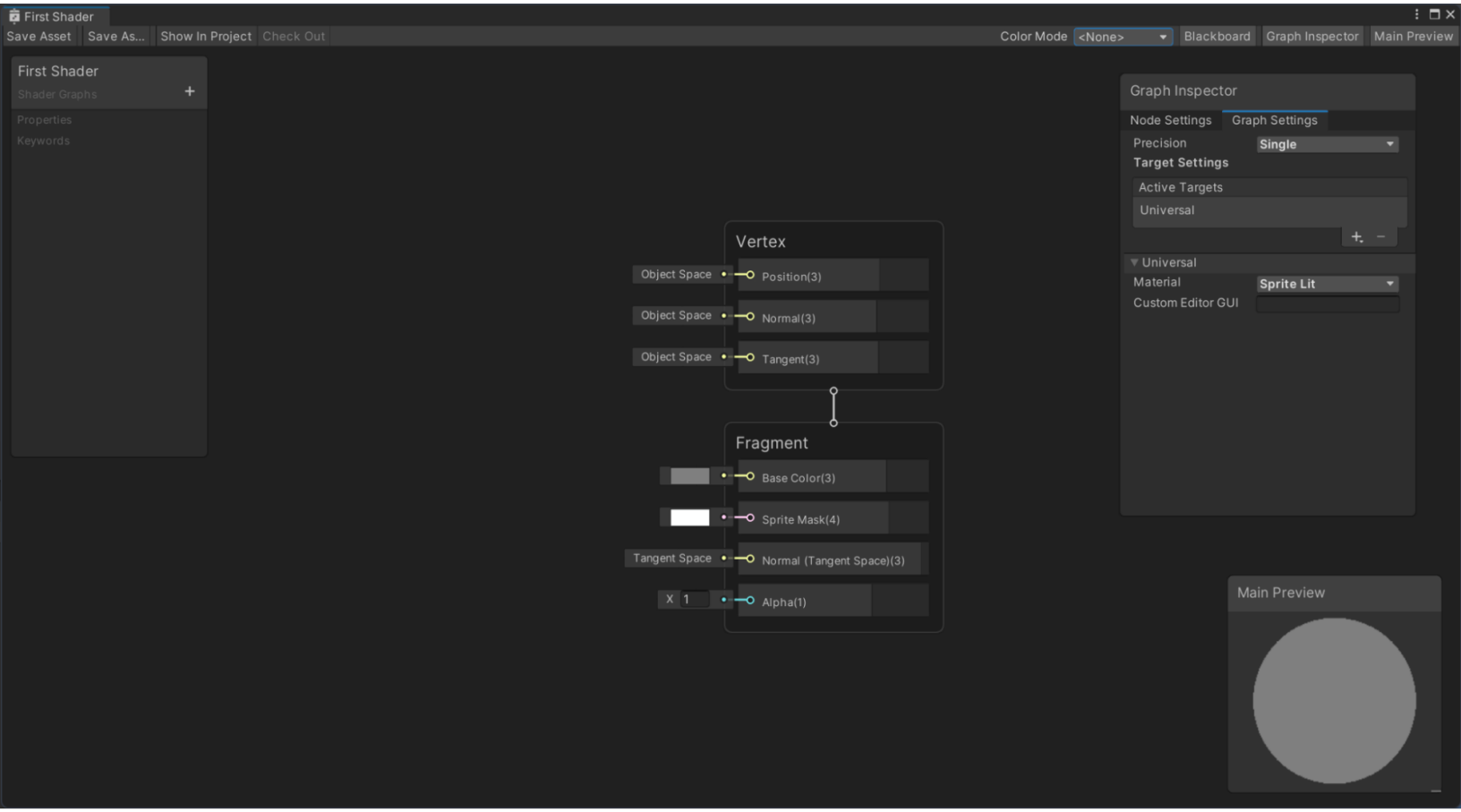
— Material Inspector にプロパティを公開することで、シーンのコンテキストでシェーダーのビジュアルを調整する

シェーダーグラフの使用

2D 用の Shader Graph の使用を開始するには、Project ウィンドウに移動し、右クリックして**Create > Shader > Universal Render Pipeline > Sprite Lit Shader Graph** (または**Sprite Unlit Shader Graph**) を選択します。

スプライトがライトの影響を受けるようにするには、Sprite Lit Shaderを使用します。これはほとんどすべてのキャラクターやオブジェクトに当てはまります。火、光源、電気、魔法、ホログラム、または幽霊のようなキャラクターなど、照明のないまたはエミッシブな表面には Sprite Unlit を使用します。

シェーダーを作成するには、まずファイルに名前を付けます。これにより、最初のシェーダーが作成されます。次に、ファイルをダブルクリックして編集モードに入ります。



Shader Graph ウィンドウ

これにより、Shader Graph ウィンドウが開きます。中央部分には、ワークスペースと呼ばれるマスタースタックがあります。これは Shader Graph の最終出力であり、シェーダーの最終的な外観を決定するものです。シェーダーごとにマスタースタックは 1 つだけである必要があります。

マスタースタックには、頂点およびフラグメント（またはピクセル）関数に対応する 2 つのコンテキストが含まれています。

左側には Blackboard が表示されます。これには、Inspector およびキーワードで表示されるプロパティが含まれています。

右上には、2つのタブで構成される Graph Inspector があります。Node Settings では、選択したノードの設定を変更できます。Graph Settings では、グラフ全体の設定を変更します。

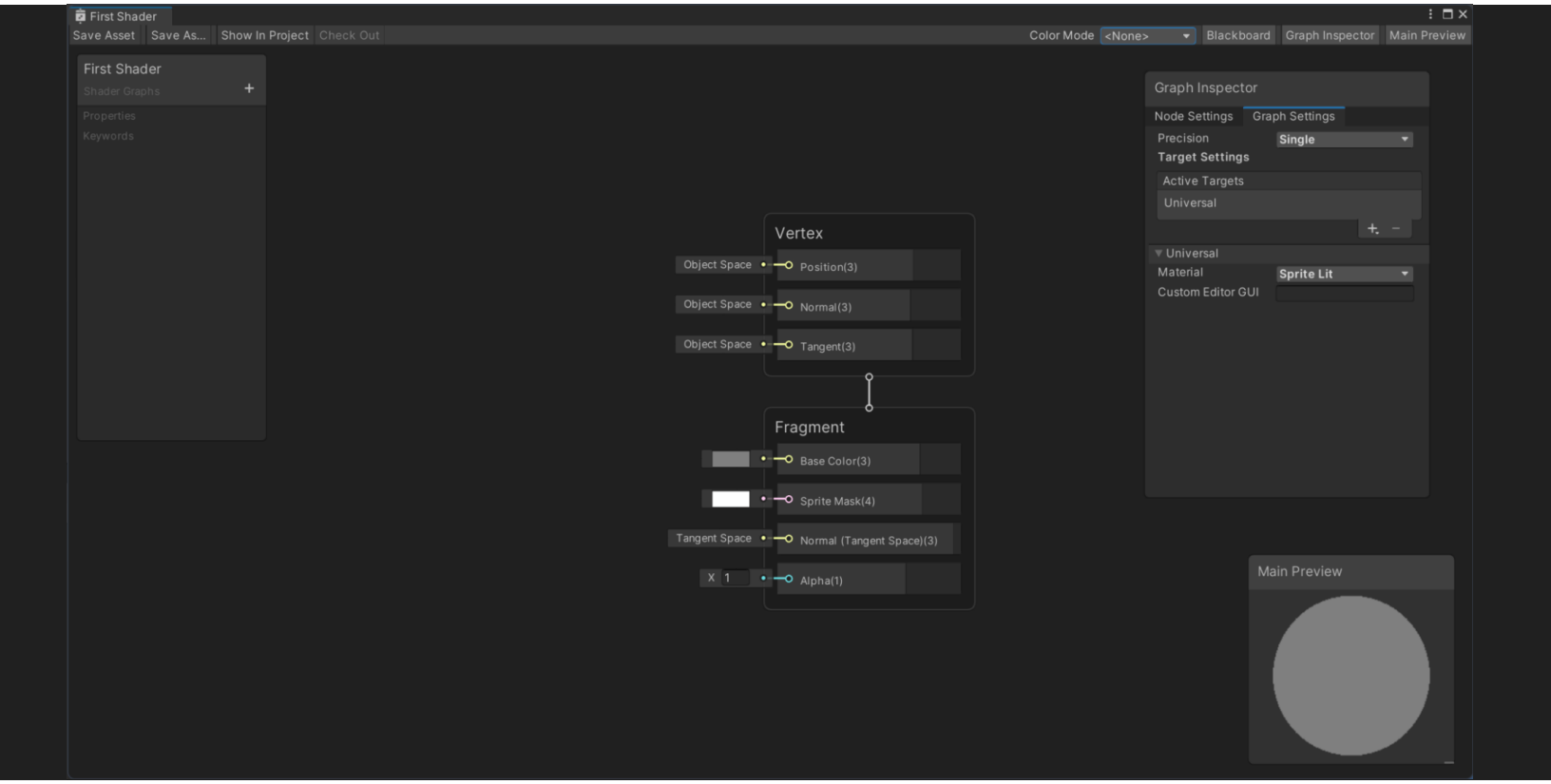
下部の Main Preview では、最終的なシェーダーを表示できます。

Shader Graph のノードは、入力、効果、および相互作用を保持するビルディングブロックです。グラフは左から右に進むため、ノードには左側に入力スロットがあり、右側に出力があります。

2D Shader Graph を作成するためのステップはすべてこちらで確認できます。Shader Graph のドキュメントはこちらです。Shader Graph の各部分とそのロジックの設定方法についての詳細が確認できます。

以下のセクションで、Dragon Crashers と Gem Hunter Match の Shader Graph のいくつかの例について参照してください。

シェーダーを作成するには、まずファイルに名前を付けます。これにより、最初のシェーダーが作成されます。次に、ファイルをダブルクリックして編集モードに入ります。



Shader Graph ウィンドウ

これにより、Shader Graph ウィンドウが開きます。中央部分には、ワークスペースと呼ばれるマスタースタックがあります。これは Shader Graph の最終出力であり、シェーダーの最終的な外観を決定するものです。シェーダーごとにマスタースタックは 1 つだけである必要があります。

マスタースタックには、頂点およびフラグメント (またはピクセル) 関数に対応する 2 つのコンテキストが含まれています。

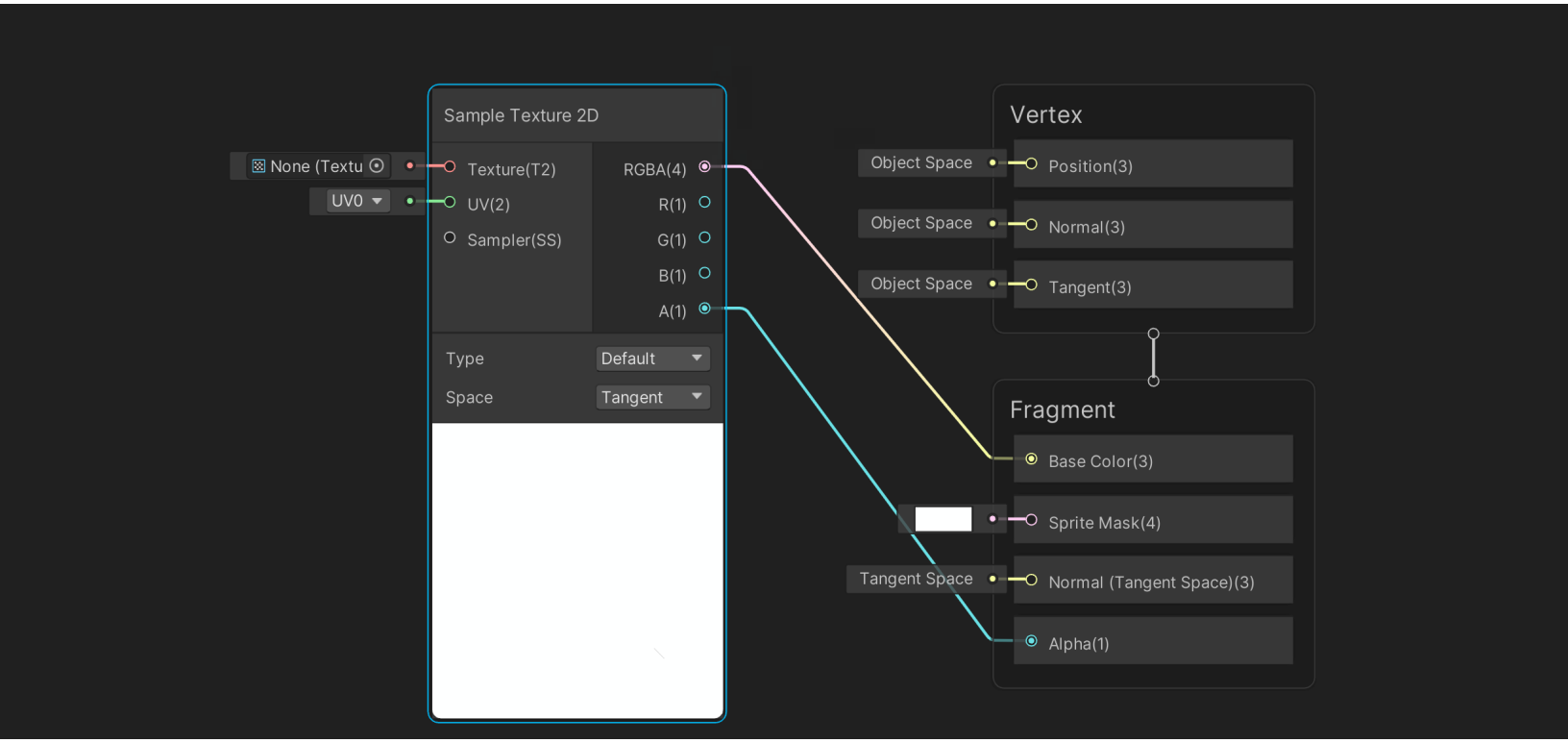
左側には Blackboard が表示されます。これには、Inspector およびキーワードで表示されるプロパティが含まれています。

右上には、2つのタブで構成される Graph Inspector があります。Node Settings では、選択したノードの設定を変更できます。Graph Settings では、グラフ全体の設定を変更します。Inspector から、精度モードを切り替えることができます。これは、低スペックのデバイスでは半分 に設定することをお勧めします。シェーダーのマテリアルを、Lit と Unlit の間で切り替えるすることもできます。

下部の Main Preview では、最終的なシェーダーを表示できます。

Shader Graph のノードは、入力、効果、相互作用を保持するビルディングブロックです。グラフは左から右に進むため、ノードには左側に入力スロットがあり、右側に出力があります。

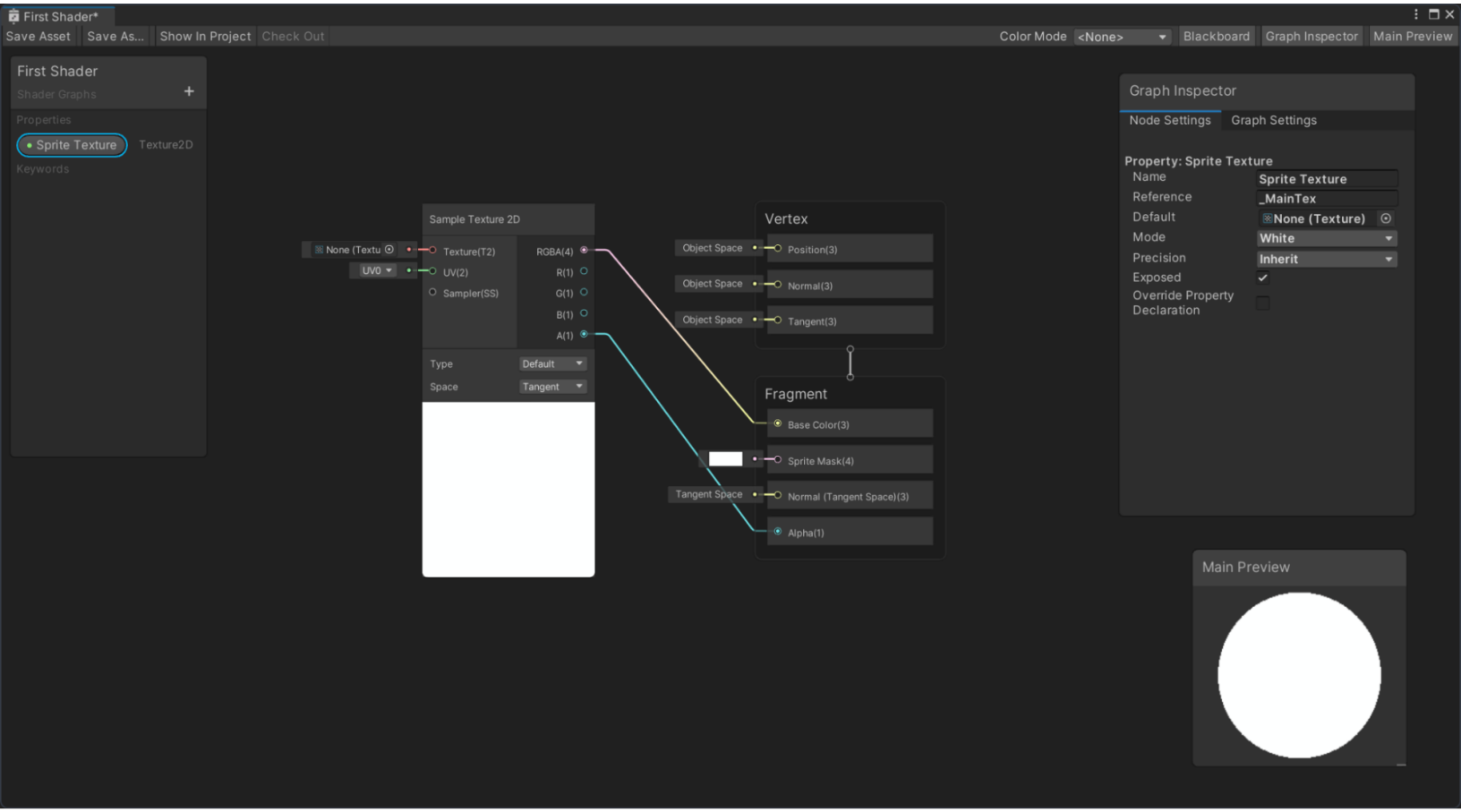
新しいノードを追加するには、ワークスペースの任意の場所を右クリックし、コンテキストメニューから **Create Node** を選択します。タイプ別にグループ化されたすべてのノードのリストが表示されます。**Input > Texture > Sample Texture 2D** を選択します。



Texture 2D ノードの例

Sample Texture 2D ノードのRGBA(4) スロットを、Master Stack のBase Color(3) の入力スロットに接続します。A(1) スロットを Alpha(1) に接続します。ここで、Master Stack のフラグメントコンテキストは、サンプルテクスチャ 2D ノードに提供するテクスチャの色とアルファを使用します。

このテクスチャをシェーダーの外部から参照するには、プロパティが必要です。



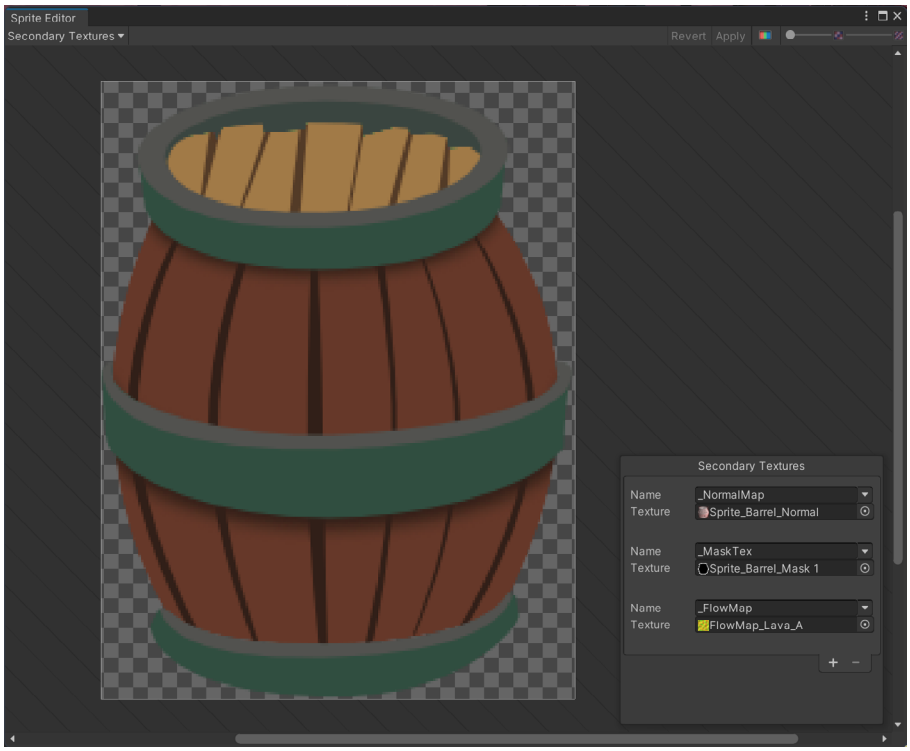
公開されたプロパティは、インスペクタービューに表示され、グラフで使用されるパラメーターを変更できます。

Blackboard にあるプラスボタンをクリックし、テクスチャ 2D を選択します。新しく作成したプロパティを選択すると、Graph Inspector にオプションが表示されます。プロパティの名前を変更できますが、より重要なのは Reference オプションです。これを _MainTex に変更します。

テクスチャには、すべての Sprite Renderer に設定されているスプライトを使用します。Sprite Renderer で、現在のマテリアルのシェーダー内の _MainTex プロパティを確認し、そのスプライトテクスチャをシェーダー内のテクスチャとして設定します。最後に、このプロパティを Blackboard からワークスペースにドラッグし、Sample Texture 2D ノードの Texture(T2) 入力スロットに接続します。ツールバーの Save Asset ボタンをクリックすると、シェーダーがコンパイルされます。

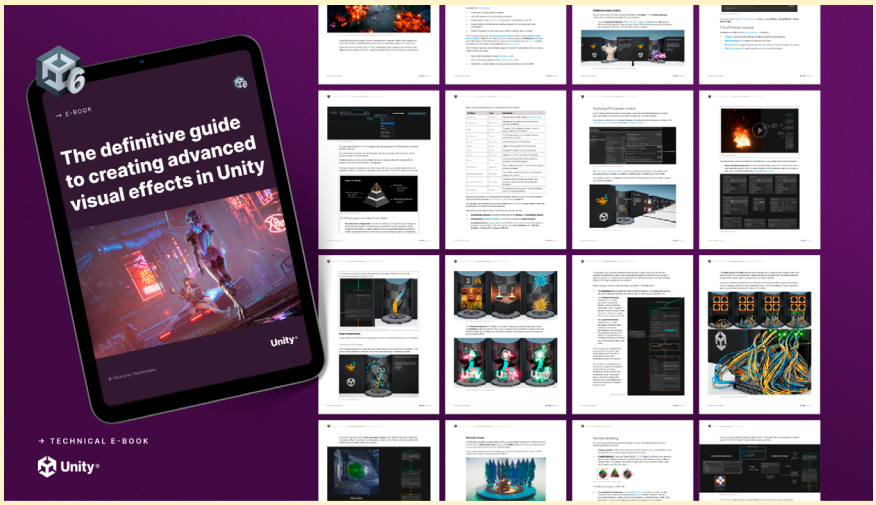
Shader Graph は、Unity の他のシェーダーと同様に使用できます。Project ウィンドウで右クリックし、**Create > Material** を選択し、リストからシェーダーを選択します。すべてのプロパティがカスタマイズ可能になります。

作成したマテリアルをスプライトに適用することで、シェーダーをテストできます。もちろん、これはテクスチャだけの最もシンプルなシェーダーです。自由に実験し、他のノードを試してみましょう。あるいは、このシェーダーに法線マップとマスクマップ (スプライトマスク) を追加してみることもできます。ヒント:Sample Texture 2D タイプ は、法線マップを正しく表示するために Normal に設定する必要があります。スプライトエディターの Secondary Textures ウィンドウで法線マップとマスクマップテクスチャのリファレンス名を確認できます。それでは頑張ってください!



樽のスプライトの二次テクスチャ名前を確認し、Shader Graph で名前で参照できる独自のテクスチャを追加できます。

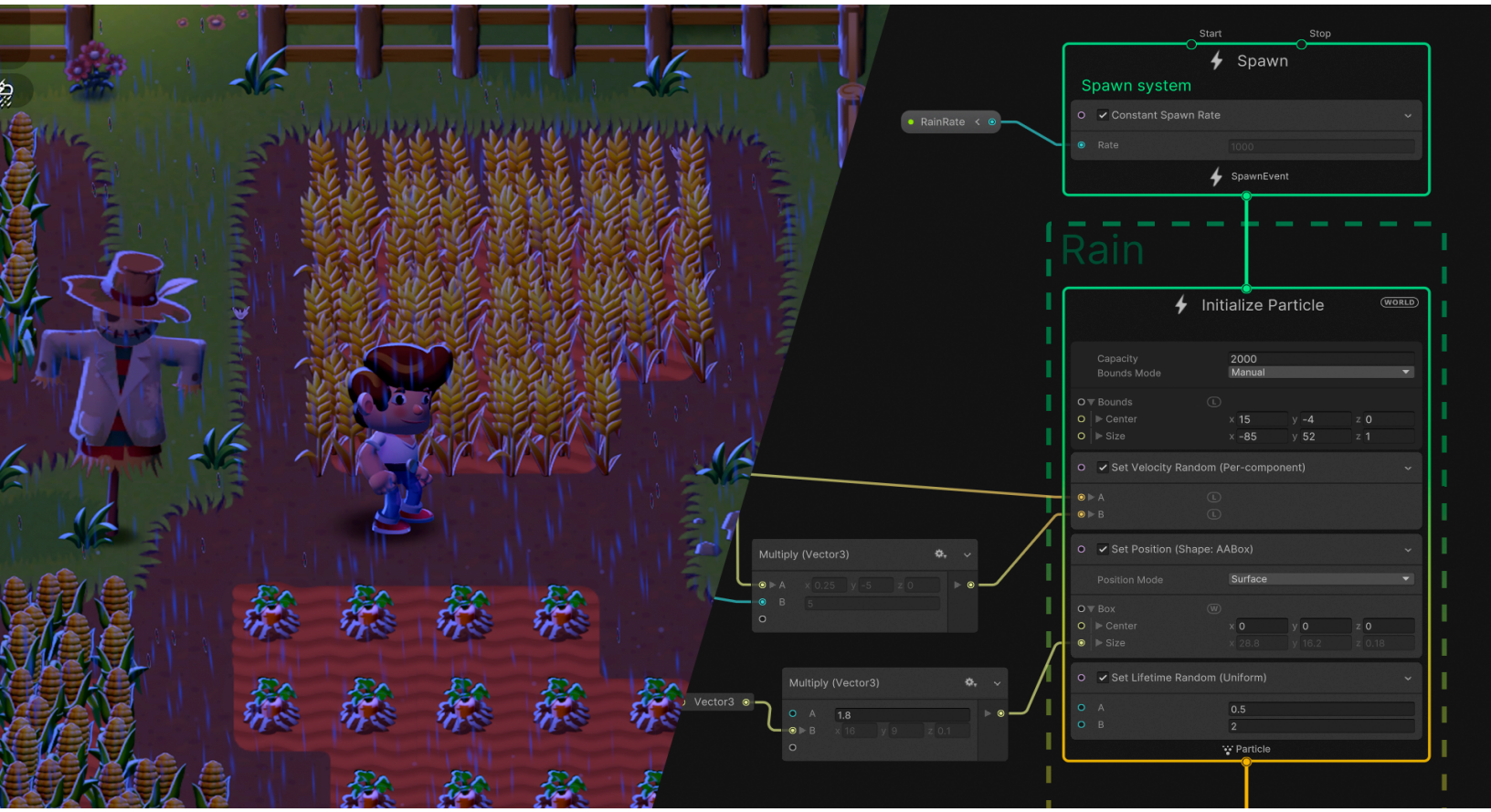
VFX Graph の多くの機能の詳細な解説については、Unity 6 で高度な視覚効果を作成するための Unity の包括的なガイドをダウンロードしてください。



e-book をダウンロード

Unity サンプルの VFX

Happy Harvest の雨



Happy Harvest で VFX Graph を使用して雨の効果を設定

雨はゲームにおける一般的な粒子ベースの効果です。ここでは、VFX Graph を使用して Happy Harvest で雨を作成するための主なステップを示します。

1. (階層内で **Visual Effect Rain** と名付けられた) 雨の粒子は、地面の上のソートレイヤーでオブジェクトの後ろにレンダリングされます。これは、雨粒が地面に接触したときに水しぶきの効果が生成され、実際の雨がどのように反応するかをシミュレートするためです。街灯柱のような垂直面に雨粒が見える場合、不自然に見える可能性があります。

   現実の世界では、光は雨粒を含むすべての環境に影響を与えます。Happy Harvest では、雨の効果用の VFX Graphで、出力コンテキスト内で Sprite Lit ベースの Shader Graph を使用して雨の粒子を照らします。雨に対応する Lit Shader Graph がないと、一日を通して変化する光を反映して色調を変化させることができません。

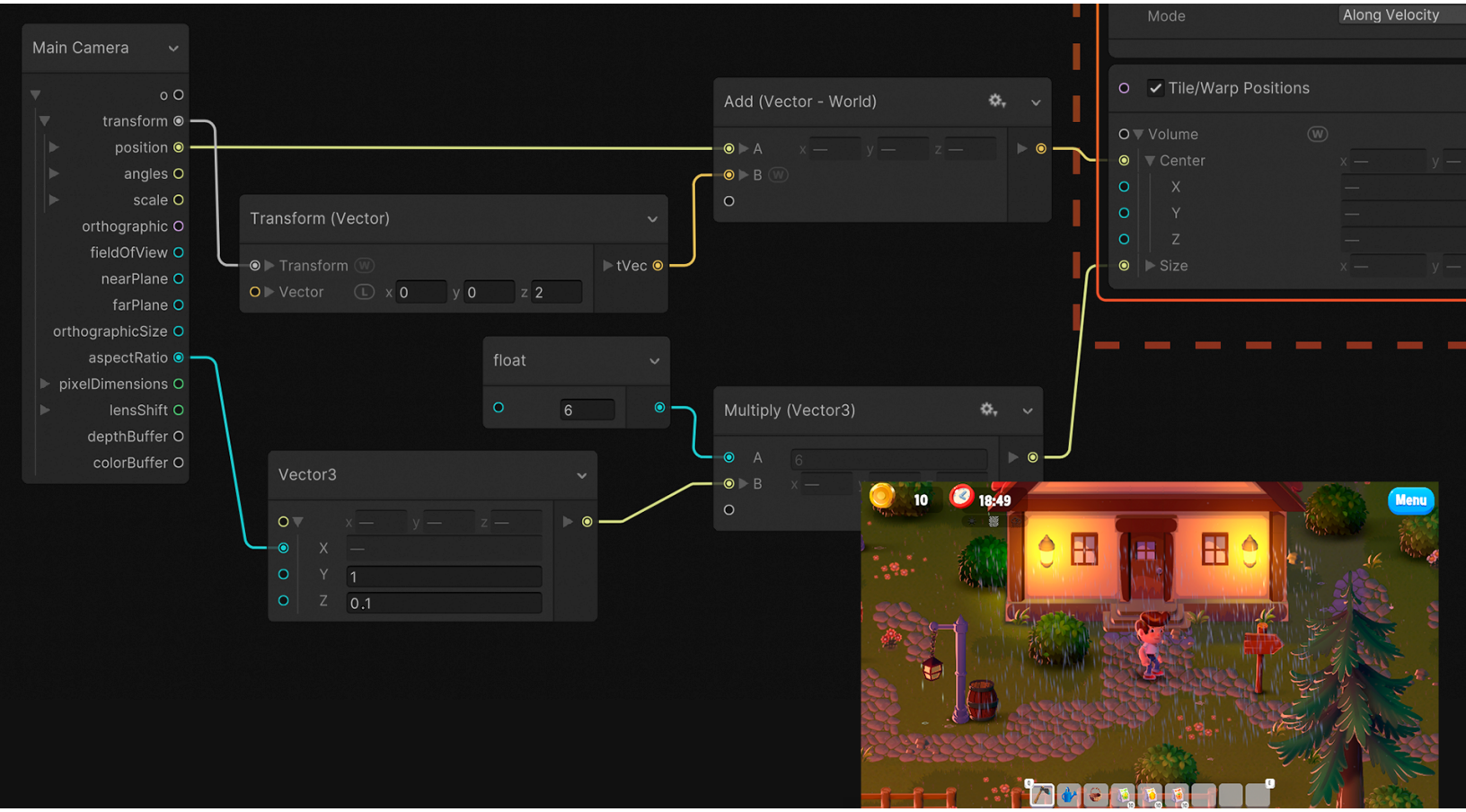
   フレームごとの水しぶき: 雨の粒子が消滅すると、Trigger On Die ブロックが GPU イベントをトリガーし、シンプルな水しぶきアニメーションを再生する単一のフリップブックパーティクルを生成します。Happy Harvest では、テクスチャ内の異なるフレームを表示するために Flipbook Shader Graph UV ブロックを使用します。また、同じ効果を得るために VFX Graph 内の Flipbook Player ブロックを使用することもできます。

2. 雨粒は前景に表示されます (階層内では **Visual Effect Rain Foreground** という名前です)。カメラレンズに近い位置にあることをシミュレートしているため、地面に当たっても水しぶきは発生しません。

3. 前述の効果のバリエーションとして、閃光を発する雷の効果を有効にして使用されます (**Visual Effect Rain Foreground Thunder**)。雷には、VFX Output Event Play Audio コンポーネントを使用してサウンドエフェクトが加えられます。このコンポーネントには、Output Event Handlers のサンプルスクリプトが付属しています。出力イベントコンテキストによって、グラフ内でこのコンポーネントがトリガーされます。

   雷の閃光: この効果を作成するためのシンプルで効果的な方法は、カメラの中心位置で初期化された VFX Graph ベースの単一パーティクルを生成し、画面全体を埋めるために十分な大きさにすることでした。出力ノードの Color Over Lifetime ブロックは、時間の経過に伴うグラデーションを評価することでパーティクルの色を変更します。そのグラデーションでは、透明度の変化で雷の閃光の点滅をシミュレートします。

カメラタイルの最適化



カメラに合わせたパーティクルの AABox サイズ乗数の調整

雨は Happy Harvest の全シーンに影響を与えますが、カメラを通して見えるのはその一部だけなので、計算されたパーティクルの多くは不要です。Inspector で **Frustum Culling** オプションを有効にすることで、それらのパーティクルのレンダリングを避けることができます。または、デモで使用される技法であるカメラタイルを使用します。

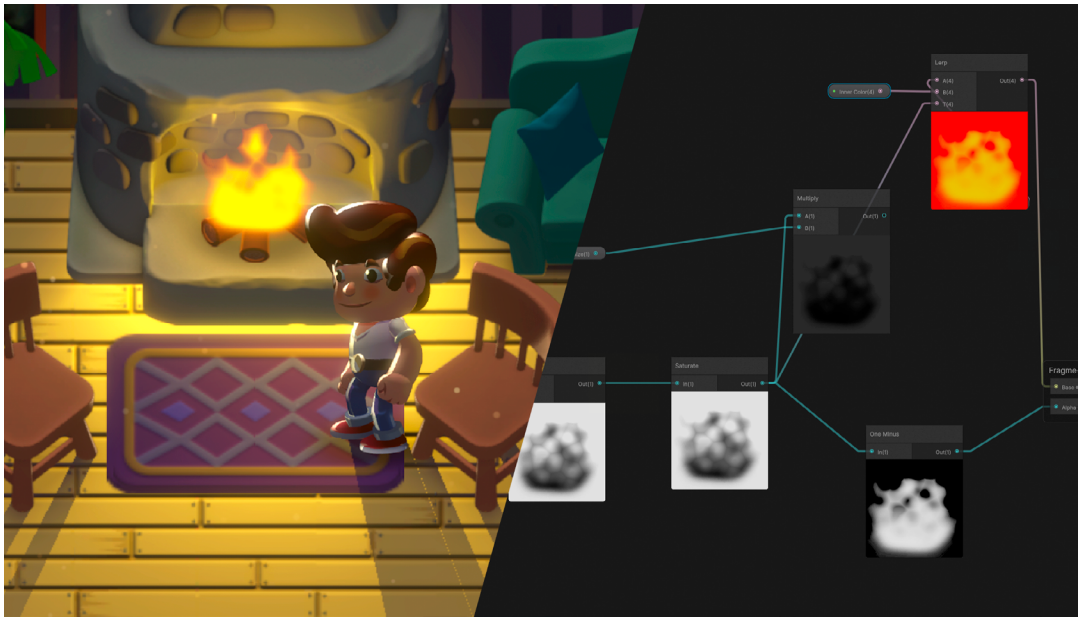
カメラタイルを使用すると、シーンビューに一致するボックス内にパーティクルを作成してレンダリングできるため、パーティクルがシーン全体にシミュレートされているかのような印象を与えることができます。カメラの外側ではパーティクルが生成されないため、追加の設定をしなくても、生成されるパーティクルの量と効果の容量の割り当てを削減できます。

Happy Harvest でカメラタイルを使用するには、位置設定形状である AABox を使用して、雨の発生位置をカメラの寸法に合わせるように設定します。カメラの形式 (サイズを手動で入力するか、MainCamera を直接使用するか、aspectRatio スロットを使用) は、シーン内のカメラ設定に合うようにサイズ乗数によってスケールされます。AABox は、カリングによって生じるポップ効果を避けるため、実際のカメラサイズよりもわずかに大きく設計されています。Shape Spawn モードは Surface に設定されています。これは、2D プロジェクトではボリューム全体に生成する必要がないためです (Z 軸は使用されません)。

この設定が完了したら、システムの出力に Tile/Warp Positions ブロックが必要になります。これはカメラの Transform にリンクされ、カメラの位置と寸法に一致するようにします。これは、システムの Initialize Context でも同様の操作を行う必要があります。

この技術は 3D における雨の効果にも適用できますが、奥行き感を出すためには、Shape Spawn モードを Volume に設定し、Z 軸の値をゼロ以外にする必要があります。

心地良い火



農夫が忙しい一日の終わりに VFX Graph で作られた火を楽しんでいます。

ゲームのもう 1 つの一般的な効果として火があります。Happy Harvest では、農家の家の庭と暖炉に心地よい焚き火があります。火の効果は、炎、火花、煙 という 3 つのパーティクルのセットで構成されており、これらはすべて、**VFX_Fire** という同じグラフから生成されます。

炎のアニメーション化

炎の滑らかなアニメーションは、Shader Graph のシェーダーレベルで実現されており、GPU とシェーダーの力を活用することで何が達成できるかを示しています。

プロジェクト内の **ShaderGraph_Fire** ファイルを開くと、単一のスプライト (**Mask texture2D**) を使用してマスクを作成し、炎が楕円形になることがわかります。アニメーションとスプライトは、ノイズと炎と背景の色合いを含みスクロールする “ボロノイ” パターンによって作成されます。アルファマスクは同じ炎の形状を使用しますが、Saturate Operator を使用することで、よりはっきりとしたアウトラインを作り出します。
火花や煙の効果は、炎に洗練されたディテールを加えるもので、標準の Lit Shader Graph を使用します。Velocity、Scale、Initial Position の各ブロックに変更を組み合わせて加えることで、たった 1 つのスプライトだけで生き生きとした効果を生み出すことができます。火花と煙の効果に関するグラフには、自分のプロジェクトでこれらの効果を再現するために役立つ注釈が含まれています。

水タイルのアニメーション

水に加えられた効果は、デモの明るい漫画スタイルにマッチさせるために役立っています。

水面は、漫画のようなコースティクス効果や透明度のある波模様を用いて表現されており、デモ全体のアートスタイルに合います。水のシェーダーは、デモの環境のほとんどが作られているタイルマップとよく機能するように最適化されています。

デフォルトの 2D Lit マテリアルである **Water** は、タイルマップの Tilemap Renderer にあります。これは、水のシェーダーを含むカスタムマテリアルに置き換えられました。タイルマップ内のすべてのタイルは同じマテリアルを使用してレンダリングされるため、このタイルマップは水タイルにのみ使用されます。

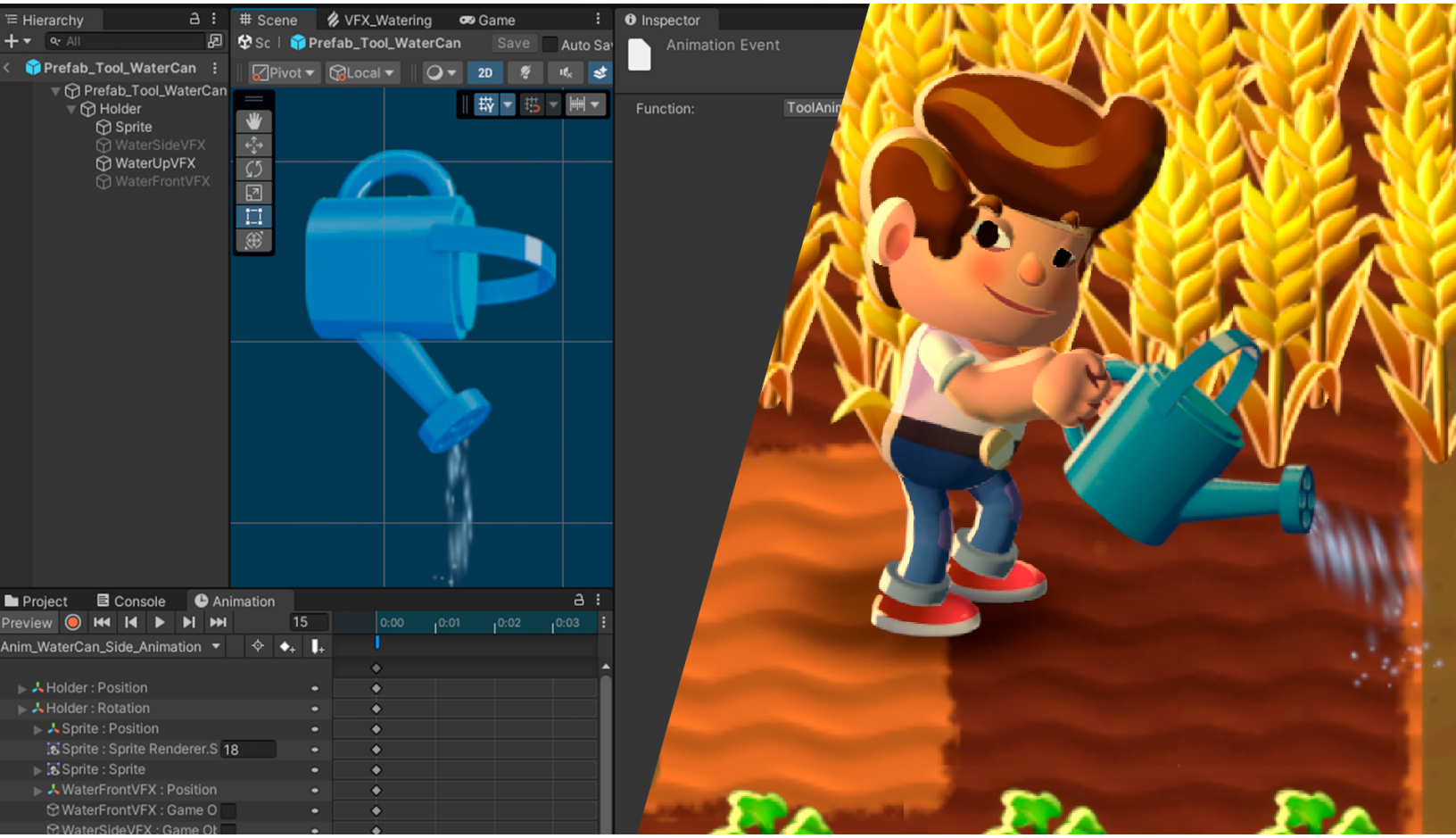
ShaderGraph_Water グラフでは、水の効果は、引き伸ばされた反転ボロノイノイズパターンから生成され、コースティクスをシミュレートします (水のコースティクスでよく用いられる手法です)。この振動する動きは、タイルとオフセット UV の時間経過によるもので、そこにノイズ効果が適用されています。

屈折効果

屈折効果は、Lerpノードを使用して表面テクスチャにブレンドされます。屈折では、カメラソートレイヤーテクスチャが使用されます。これは、定義されたソートレイヤーまでのカメラレンダリングのスナップショットであり、前のソートレイヤーのすべての要素がそのテクスチャに表示されることを意味します。Happy Harvest では、レンダリングは **Objects** ソートレイヤーまで行われます。これは、キャラクターのような要素に使用されるものです。カメラソートレイヤーのテクスチャの UV は、Normal と Hight ノードで生成された法線マップによって歪められます。この法線マップは、表面テクスチャにも使用されているボロノイテクスチャを使用します。

Lost Crypt 2D デモを使用して、このチュートリアルを参照しながら、2D の他の水の効果に関して段階を踏んで学んでください。

ゲームプレイにおける VFX



じょうろのプレハブには、キャラクターの向きに基づいてスプライトを変更する3つのアニメーションクリップが付属しています。

これまで説明してきた効果は、Happy Harvest のゲームプレイとは直接関係ありません。これらは環境の一部であり、デモで起こっていることに関わらず機能します。しかし、ほとんどのゲームでは、環境内やキャラクター内で何らかのアクションが発生したタイミングに基づいて、一部の効果をトリガーする必要があります。Happy Harvest で、トリガーされた効果を見てみましょう。

アイテムを使用するときの効果

いくつかの効果は、じょうろのようなキャラクターのツールに付随しています。じょうろのプレハブ、**Prefab_Tool_WaterCan** には、キャラクターが向いている方向 (上、下、横) に基づいてスプライトを変更する 3 つのアニメーションクリップがあります。これらのアニメーションには、Tool Animation Event Handler コンポーネント内の関数を呼び出すアニメーションイベントがあります。

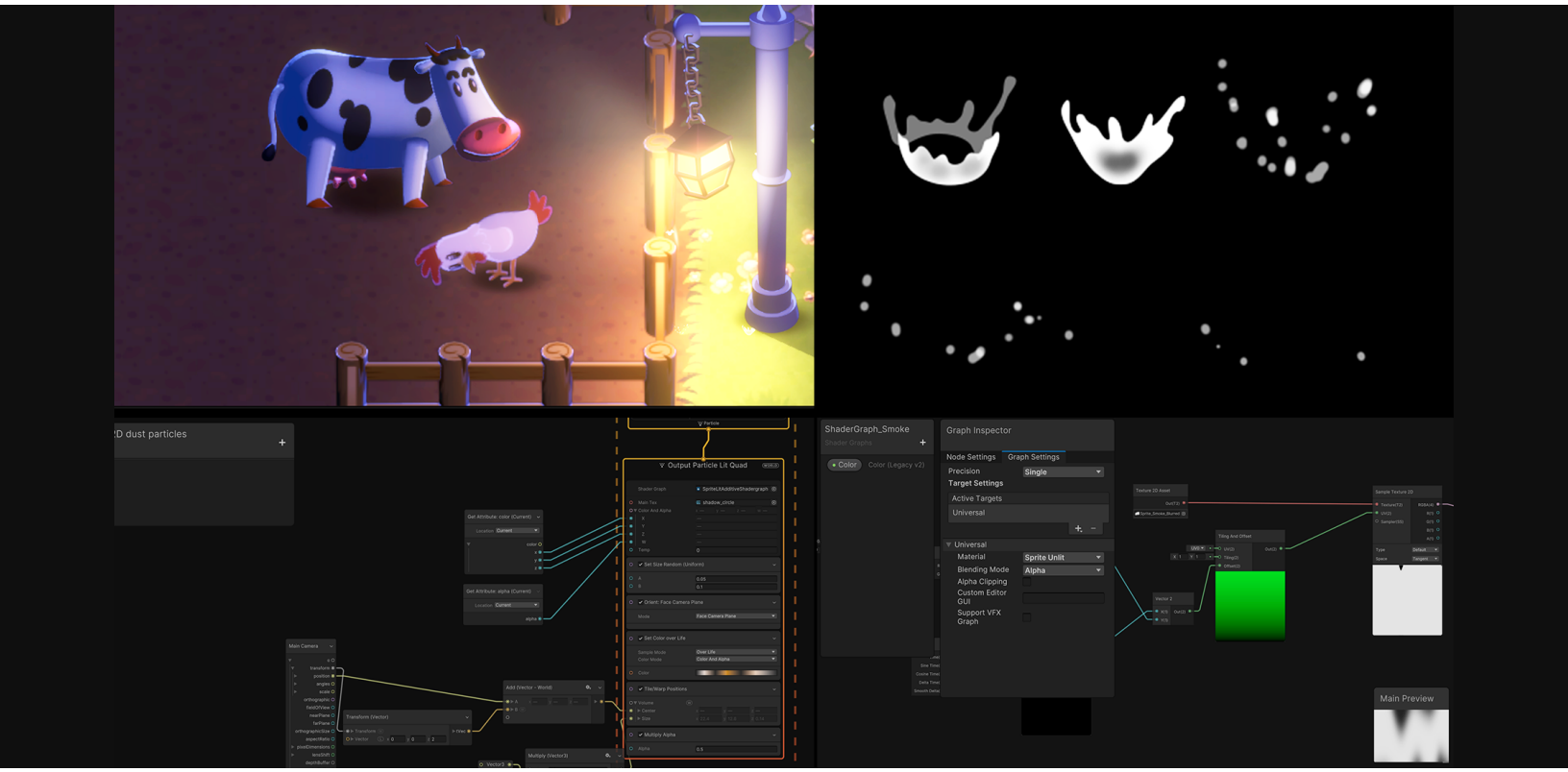
このスクリプトは、Inspector から参照される各位置に対応する VFX Graph のインスタンスを有効または無効にします。

タイルを繰り返し処理するときの効果

いくつかの効果は、掘削や収穫のようにタイル上で発生します。これにより、キャラクターのアニメーションや位置変更に問題が発生することを防ぎます。セルを耕す時が来ると、耕す効果がそのセルに作用し、収穫する時が来ると、そのセルで収穫効果を呼び出されます。

耕す効果がトリガーされると、**TillingPuff_prefab** 効果を **TerrainManager** コンポーネントから **Grid** ゲームオブジェクトに参照します。タイルの中心に移動し、その後再生します。作物収穫のようなその他の効果で、**Data/Crops** スクリプタブルオブジェクトから参照される VFX グラフを呼び出します。**TerrainManager** スクリプトの初期化時に、VFX プールディクショナリが作成されます (スクリプト内の変数 **m_HarvestEffectPool**) 。これにより、必要に応じて再生するためにいくつかの VFX プレハブがインスタンス化されます。

違いを生むちょっとした工夫

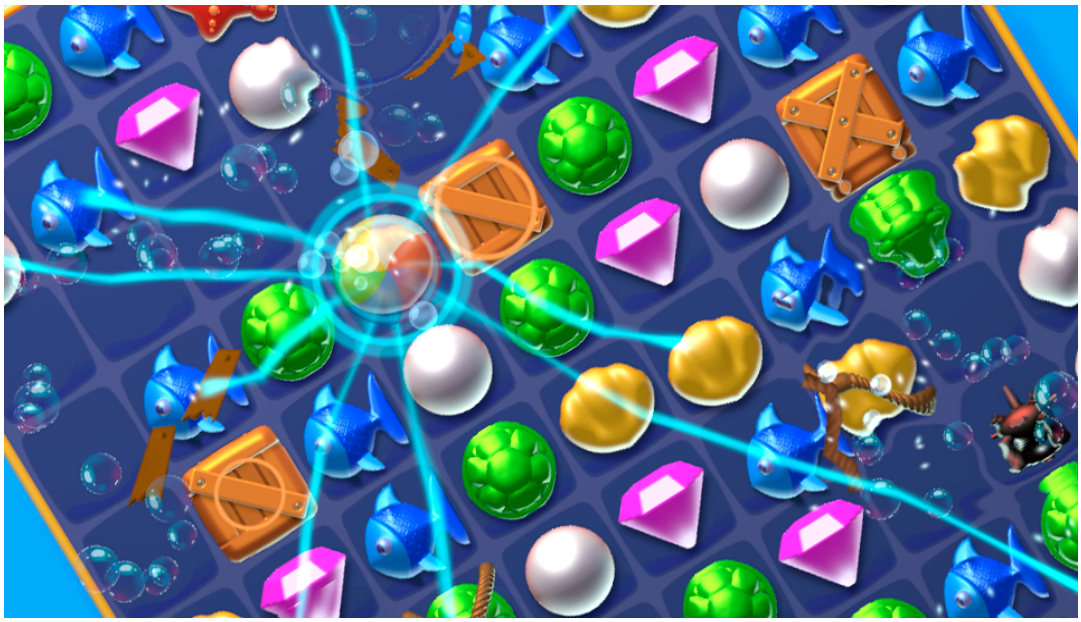


Happy Harvest の細かい効果によって、デモが洗練され、探索がより楽しいものになります。これらの小さな効果の例として以下が挙げられます。

- **VFX_DustParticles**: 環境に周囲のほこりを広げます。
- **VFX/Water**: 微妙な水の波と飛沫を起こします。
- **VFX_Leaves**: プレイヤーが通り過ぎるときに、茂みから葉が落ちるようにトリガーします。
- **Character**: キャラクターに付随する煙のパフ効果をトリガーします。これはビルトインパーティクルシステムで作成されます。
- **P_VFX_Moths**: スポットライトの近くを飛ぶ蛾を発生させます。これもビルトインパーティクルシステムで作成されています。

上記のリストの VFX Graph による各効果には、各ノードの目的を説明する注記が含まれています。

Gem Hunter Match の VFX



Gem Hunter Match の VFX Graph および Shader Graph ベースの視覚効果には、それぞれのグラフに説明コメントが含まれており、各システムが 2D ライトとどのように連携しているかを理解するのに役立ちます。

Gem Hunter Match の特殊効果には以下のものが含まれます。

- **unlit のパーティクルを表示する VFX Graph ベースの効果:** この効果には出力パーティクルクアッドを使用します。
- **カスタムシェーダーを使用する VFX Graph ベースのパーティクルエフェクト**: これらはシェーダーグラフを使用してパーティクルを描画します。例えば、**VFX_CellHoldDistortion** という効果は、シェーダーで **Support VFX Graph** オプションが有効な出力に 2D シェーダーを使用します。出力ノードに 2D Shader Graph を追加すると、出力は以下のいずれかになります。

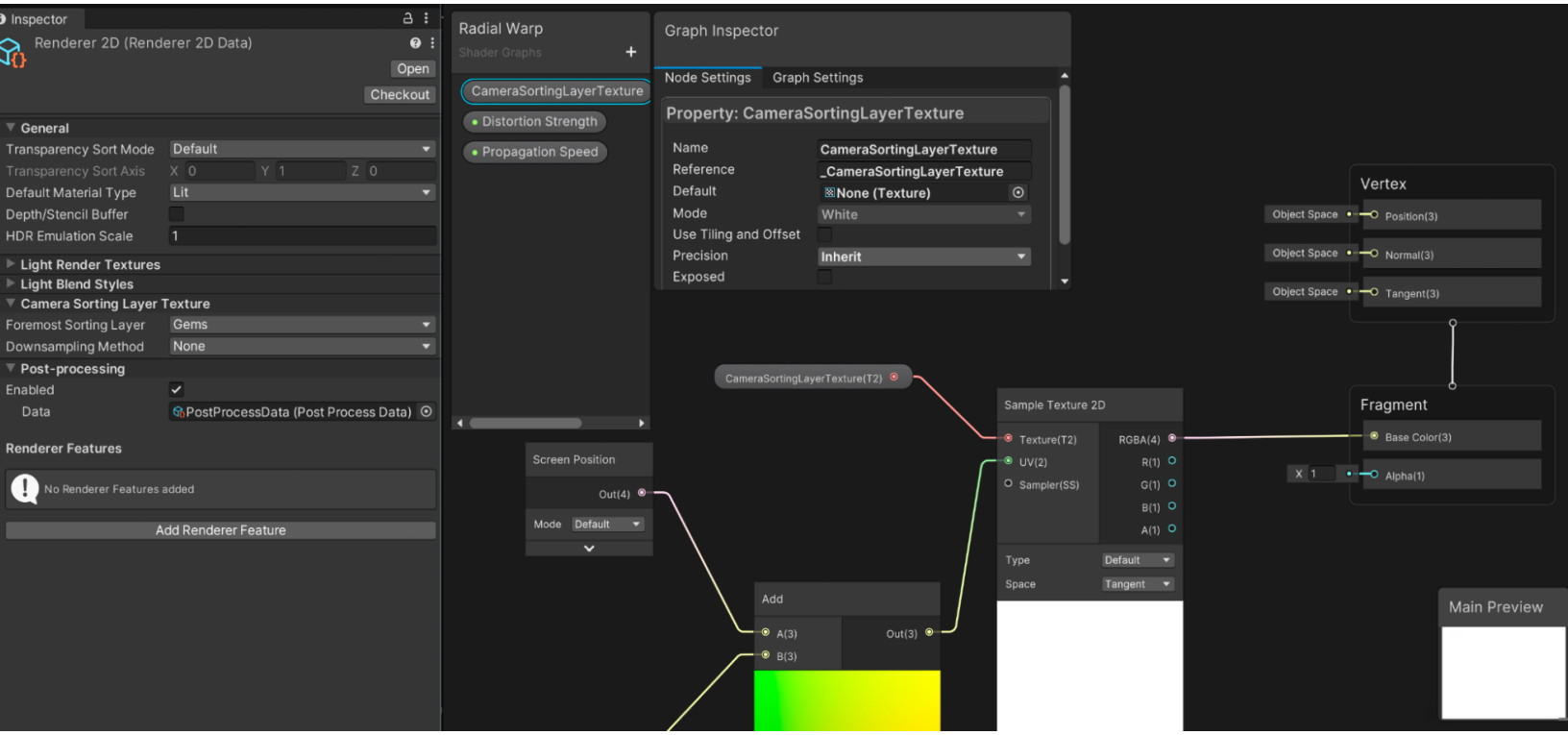
— **出力パーティクルスプライト unlit クアッド:** Shader Graph のマスターノードが Sprite Unlit の場合、パーティクルに効果を適用できますが、2D ライトには反応しません。

— **出力パーティクルスプライト lit クアッド**: Shader Graph のマスターノードが Sprite Lit の場合、2D ライトに反応します。

洗練された見た目の効果とは、通常、GPU イベント で特定のタイミングで発生するいくつかの Spawn ノードと滑らかにリンクして構成された一連のエフェクトです。独自のパーティクルの効果を作成する際は、各 VFX Graph の Inspector で正しいソートレイヤーを使用するようにしてください。

カメラソートレイヤーテクスチャ

Radial Warp シェーダーでは、URP 2D Camera Sorting Layer Texture 設定を使用します。この便利な機能を使うと、URP 2D Renderer の設定で指定されたソートレイヤーまでで生成されたグラフィックスにアクセスでき、そのデータを Shader Graph で使用して効果を適用できます。Happy Harvest サンプルでは、カメラソートレイヤーテクスチャを使用して水の屈折効果を作成し、Dragon Crashers では煙の歪みを作成します。このサンプルでは、衝撃波をシミュレートする歪みを適用するためにこれを使用し、マッチング時に視覚的な魅力を高めています。

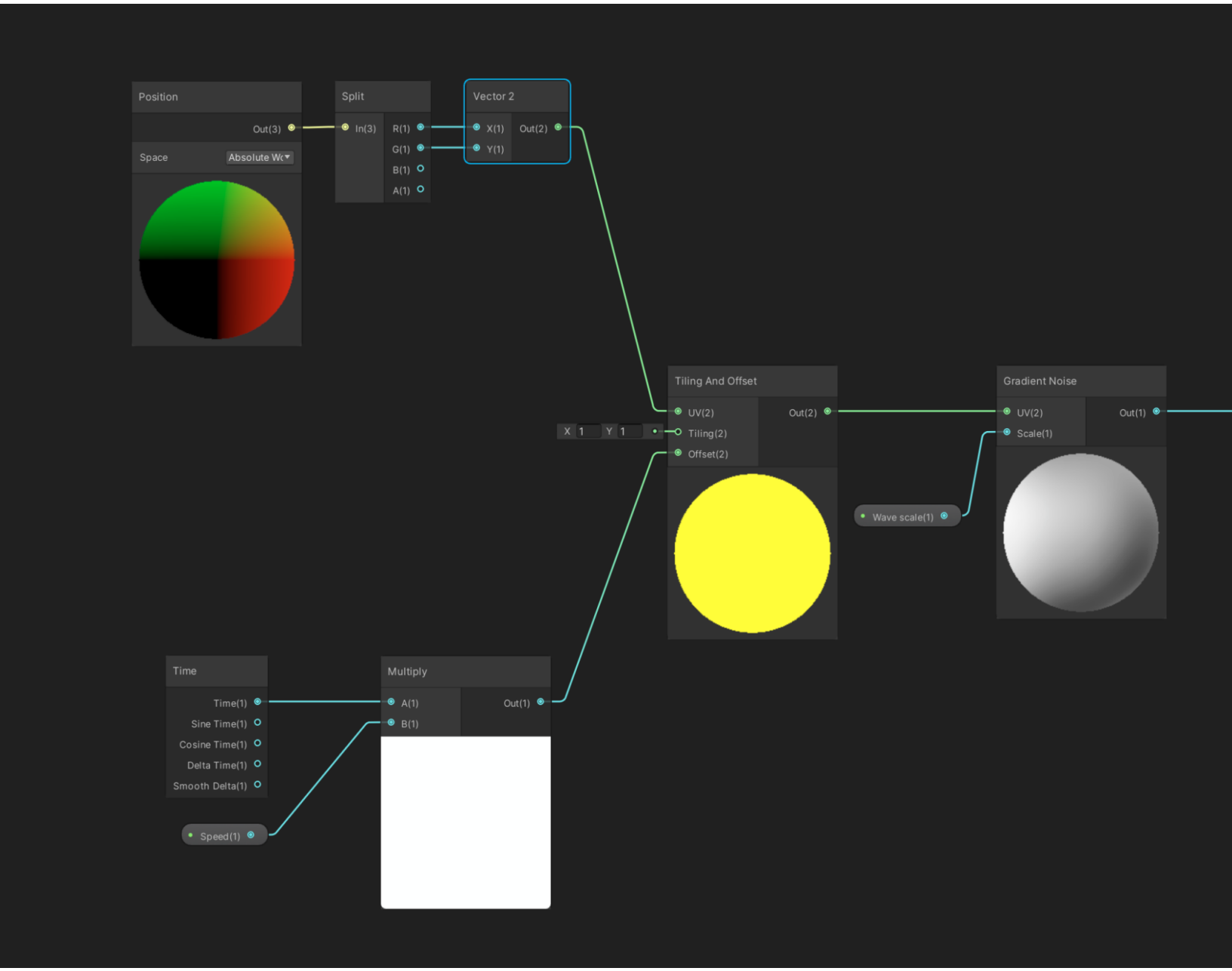


左端の Renderer 2D Data Asset と Radial Warp シェーダーは、画面上の画像をシェーダーの基本色のテクスチャとして読み込み、UV で変形を適用します

Dragon Crashers の Shader Graph の例

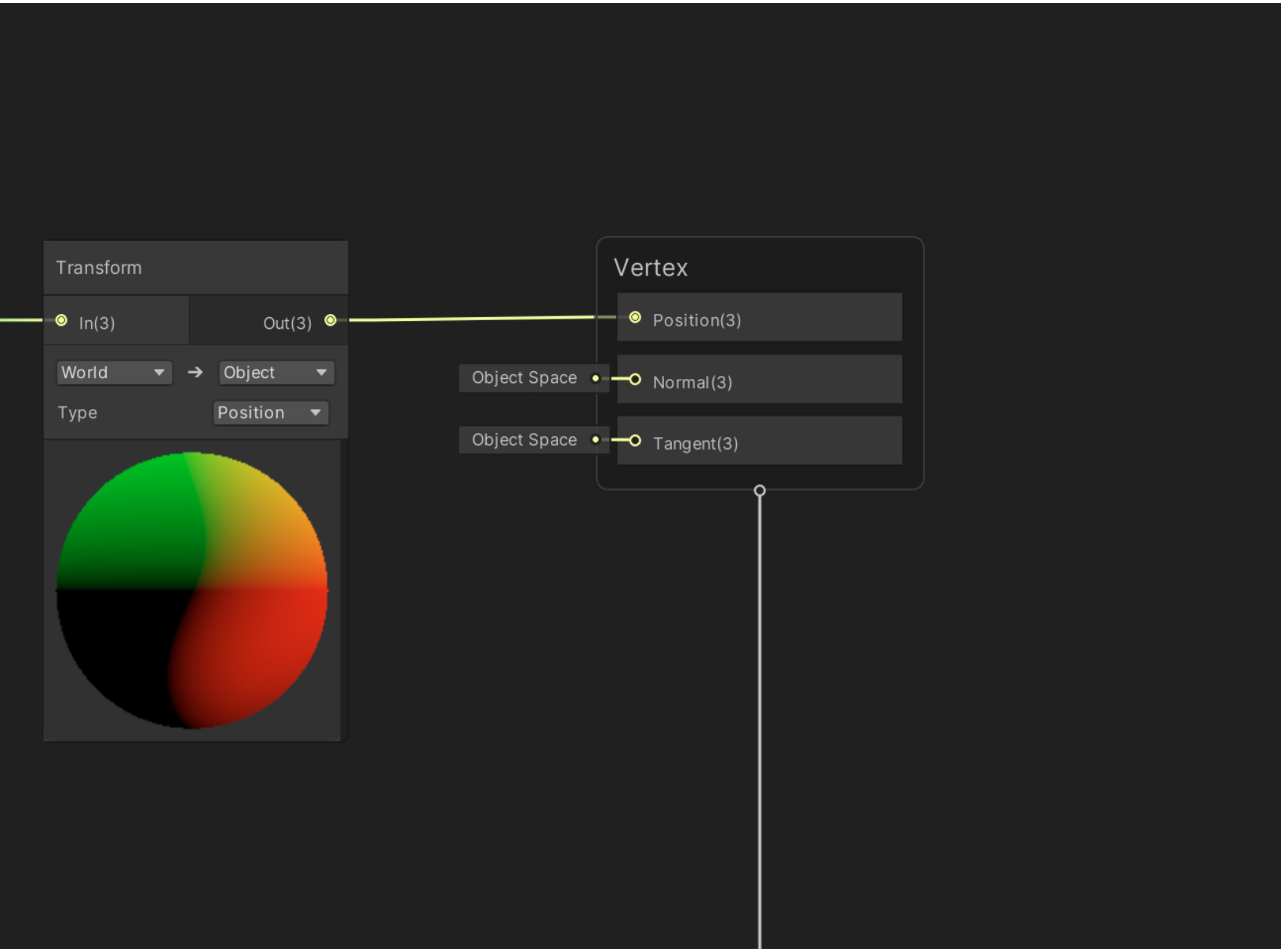
頂点ディスプレイスメント

頂点ディスプレイスメントシェーダーは、メッシュのジオメトリ内の頂点の位置に影響を与えます。頂点をさまざまな方法で変位させることができますが、最も簡単な方法は動くノイズを適用することです。このシェーダーは、葉やぶら下がるつる、ロープ、旗などの波打つオブジェクトに適しています。この Shader Graph ファイルの例は、Dragon Crashers プロジェクトで見つけられます。ファイル名は ShaderGraph_Sprite_Lit_Waving です。



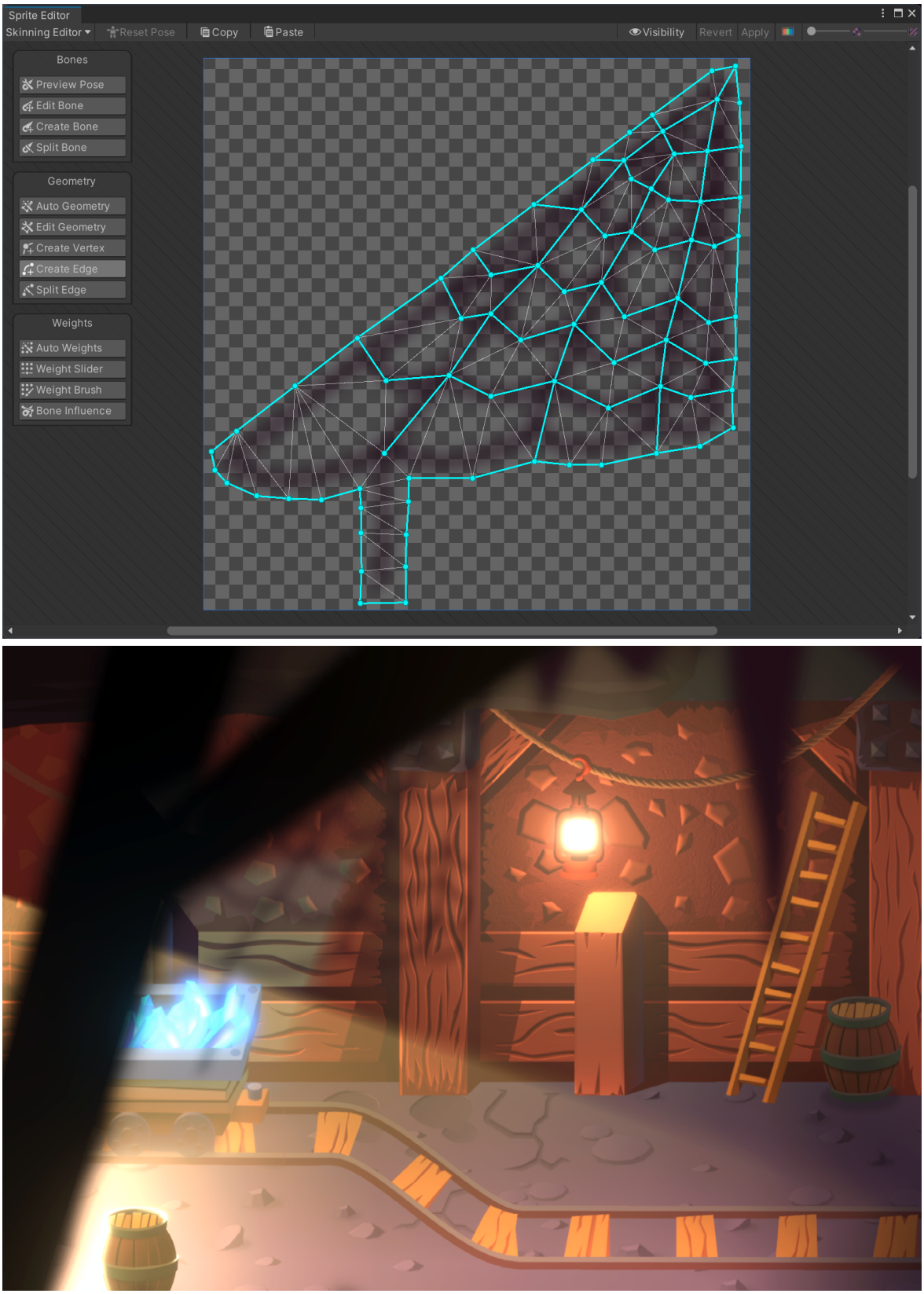
Waving シェーダーの主な部分

波のアニメーションを実現するために、グラデーションノイズが波のマスクとして使用されます。その UV オフセットは、Speed という名前の Float で乗算された Time ノードによってアニメーション化されます。これらはすべて、頂点コンテキストの頂点の位置に影響を与えます。



Shader Graph の頂点コンテキストにより頂点の位置が修正され、フラグメントマスターでピクセル情報がスタックされます。

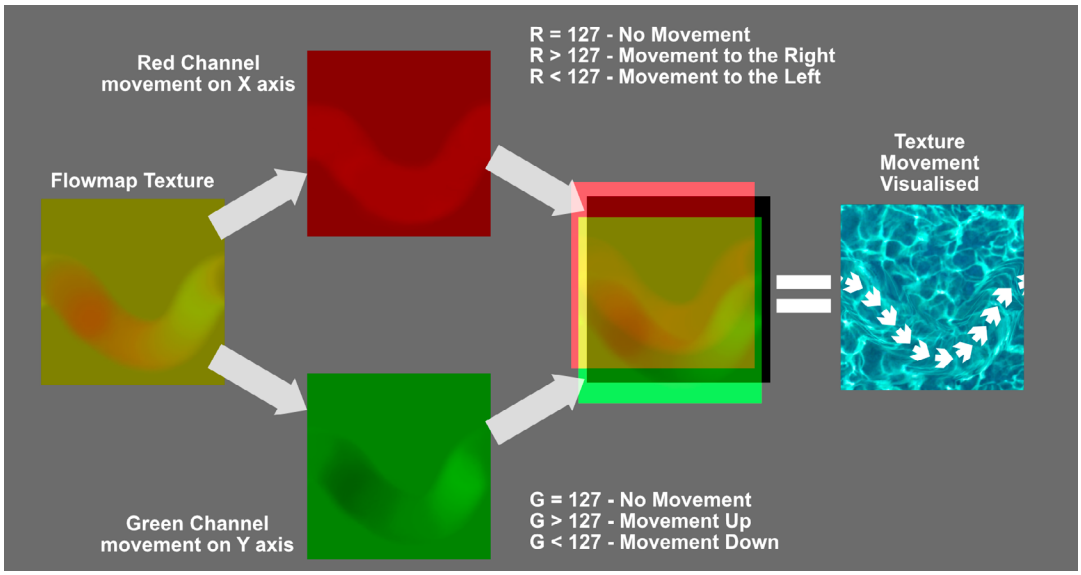
Waving シェーダーの影響を受けるスプライトには、十分な頂点数がある必要があります。頂点数が不十分だと、アニメーションが粗く見えます。頂点を編集するには、Skinning エディターの Geometry ツールを使用します。



Cobweb シェーダーの実例 (クリックしてアニメーションを確認)

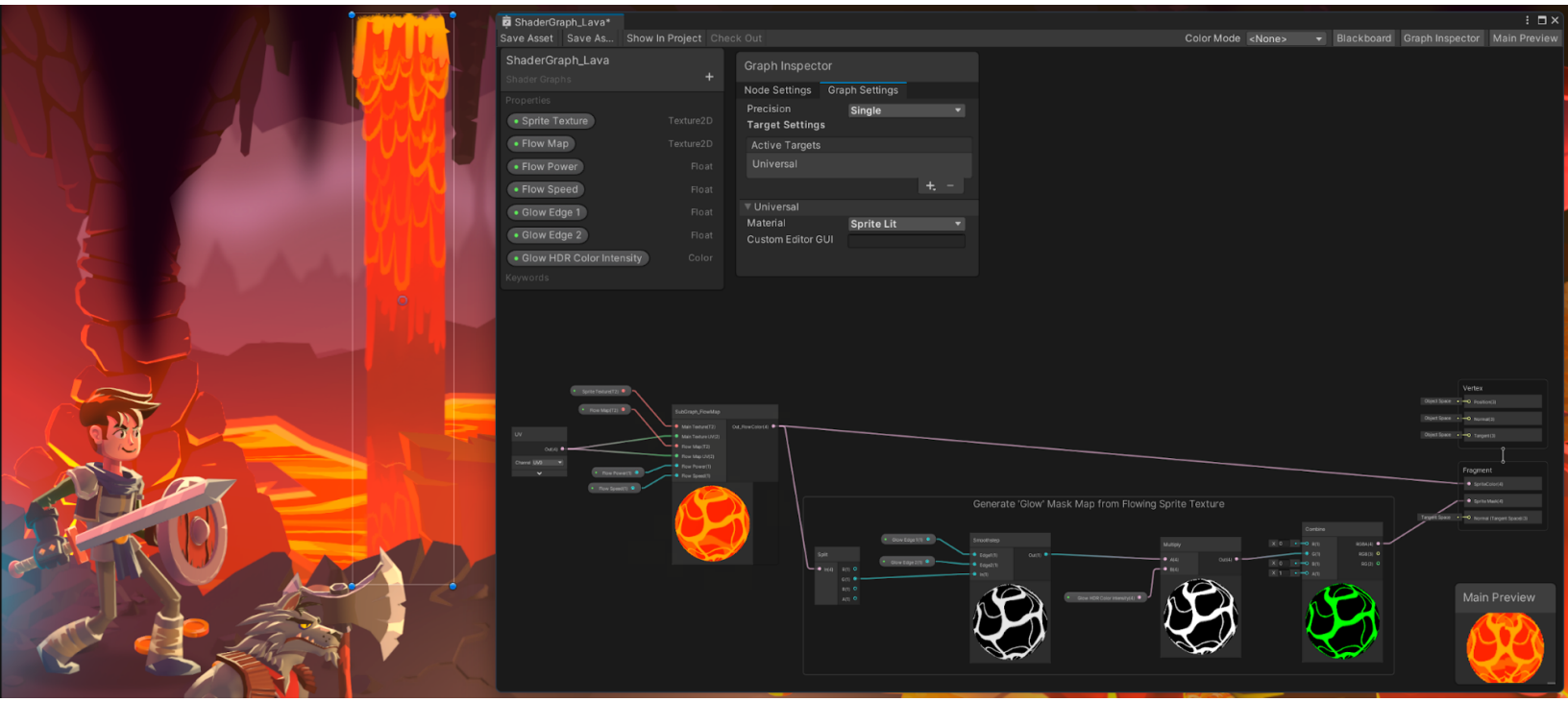
フローマップ

フローマップは、方向情報を保存するテクスチャです。Dragon Crashers デモの中にフローマップシェーダーがあり、名前は ShaderGraph_Lava です。このシェーダーでは、フローマップテクスチャを使用して、メインテクスチャの UV 座標の方向を制御します。赤と緑の色は、各フレームでピクセルが従う XY 方向を示すために使用され、メインテクスチャのピクセルが “流れる” ようになります。



フローマップテクスチャの色の説明: 赤色は X 軸上のピクセルの動きを制御し、緑色は Y 軸上の動きを決定します。シェーダーは、矢印で視覚化された方向にメインテクスチャのピクセルを移動させます。

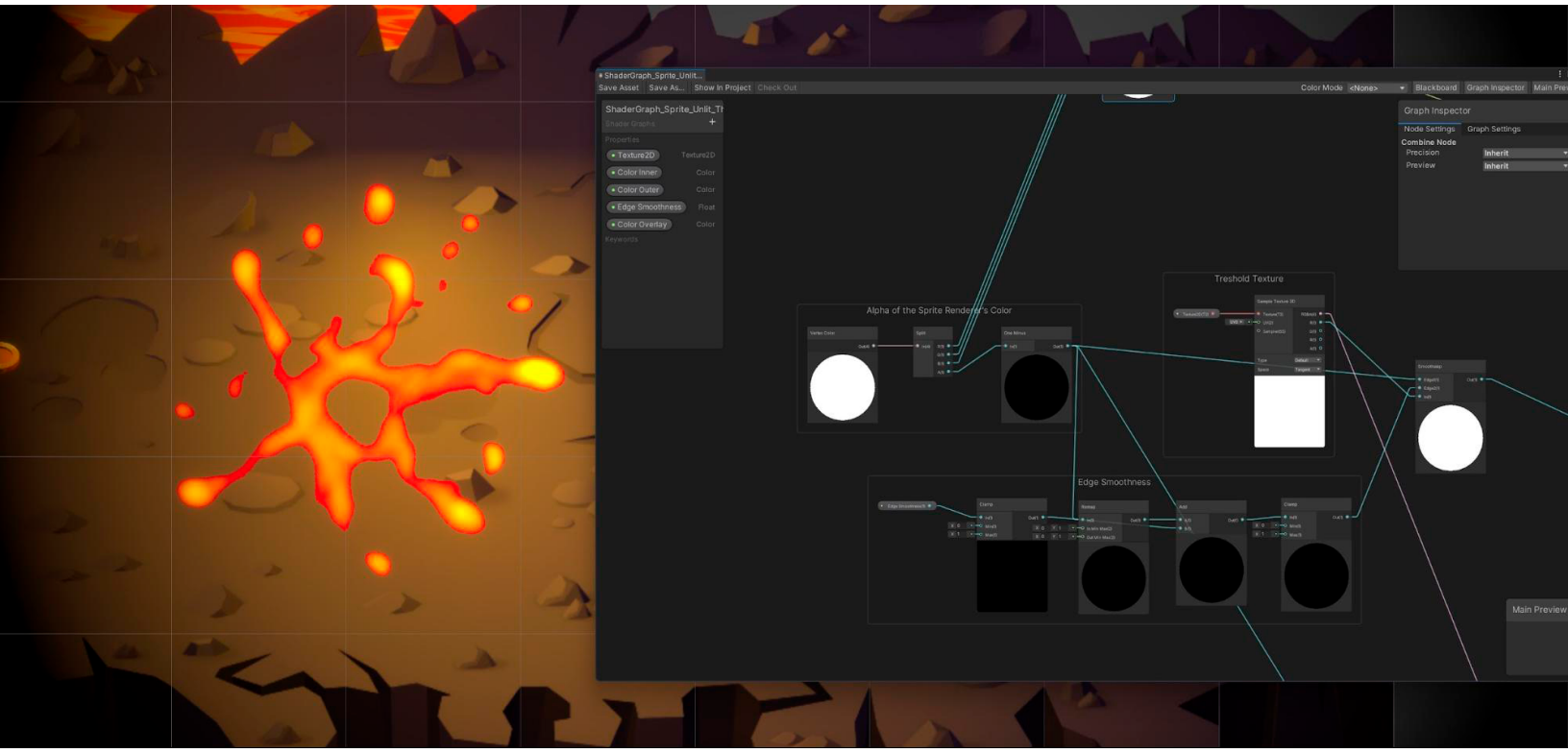
SubGraph FlowMap を開いて、この効果を実現する方法を学びましょう。



手描きのフローマップは、溶岩の流れに粘性の効果や漫画風の効果を与え、プロジェクトのアートディレクションに合致します。フローマップ作成に使用できるツールには、Flow Map Generator / Visualizer や Flow map painter が含まれます。

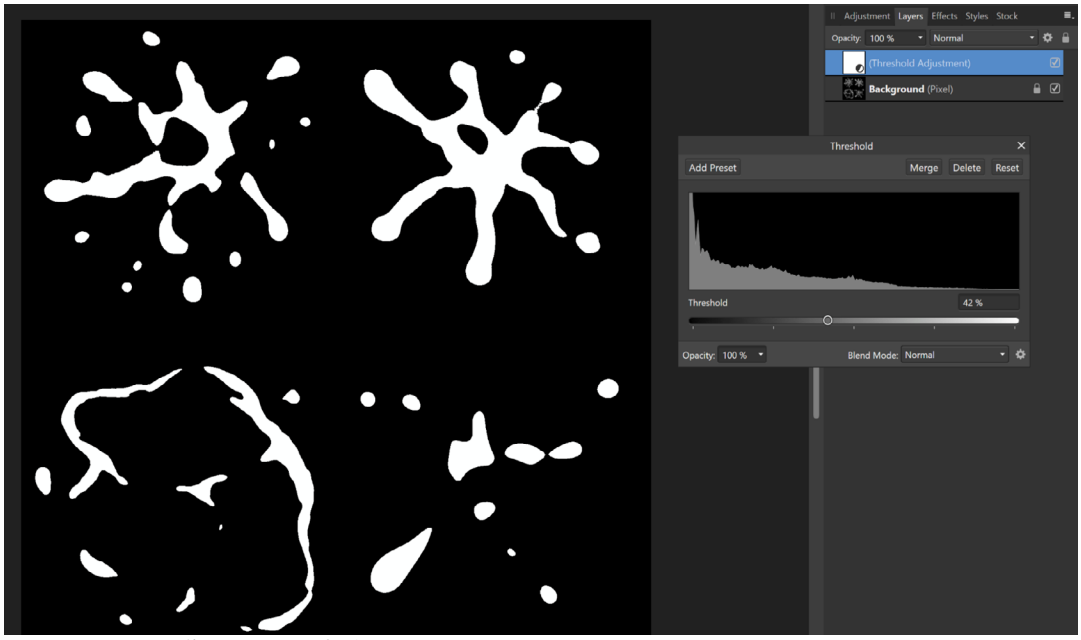
流体プロップアニメーションのためのしきい値アニメーション

別のシェーダーアニメーション技術として、アニメーションアルファクリッピングと呼ばれるものを試すことができます。これにより、単一のテクスチャから滑らかなアニメーションを作成します。これは、各フレームで特定の範囲のピクセルをアルファ値に基づいて表示することによって作成します。テクスチャはシングルチャンネルであるため、効果は簡単に達成できます。Dragon Crashers で、ShaderGraph_Sprite_Unlit_ThresholdAnim という名前の例を見つけてください。

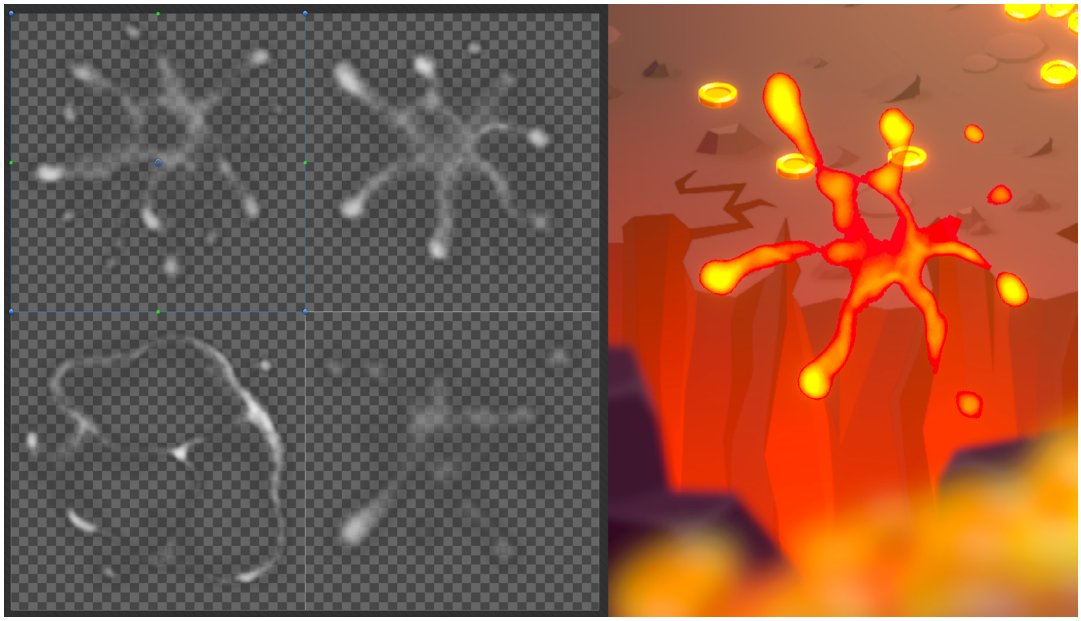


この効果を達成するために、Smoothstep ノードが使用されます。テクスチャの赤いチャンネルは In(1) スロットに接続され、頂点色のアルファでエッジと Smoothstep 効果の滑らかさの評価が制御されます。頂点色はスプライトレンダラーの Color プロパティによって制御されるため、スプライトの不透明度を変更すると、テクスチャがアニメーション化されます。これにより、シェーダーはパーティクルでも使用可能になります。

Affinity Photo を使用する場合、このシェーダーのテクスチャを白黒でペイントし、値のスライダーを動かしながら Threshold Adjustment を使用してアニメーションをプレビューできます。



Affinity Photo でのしきい値アニメーションのプレビュー

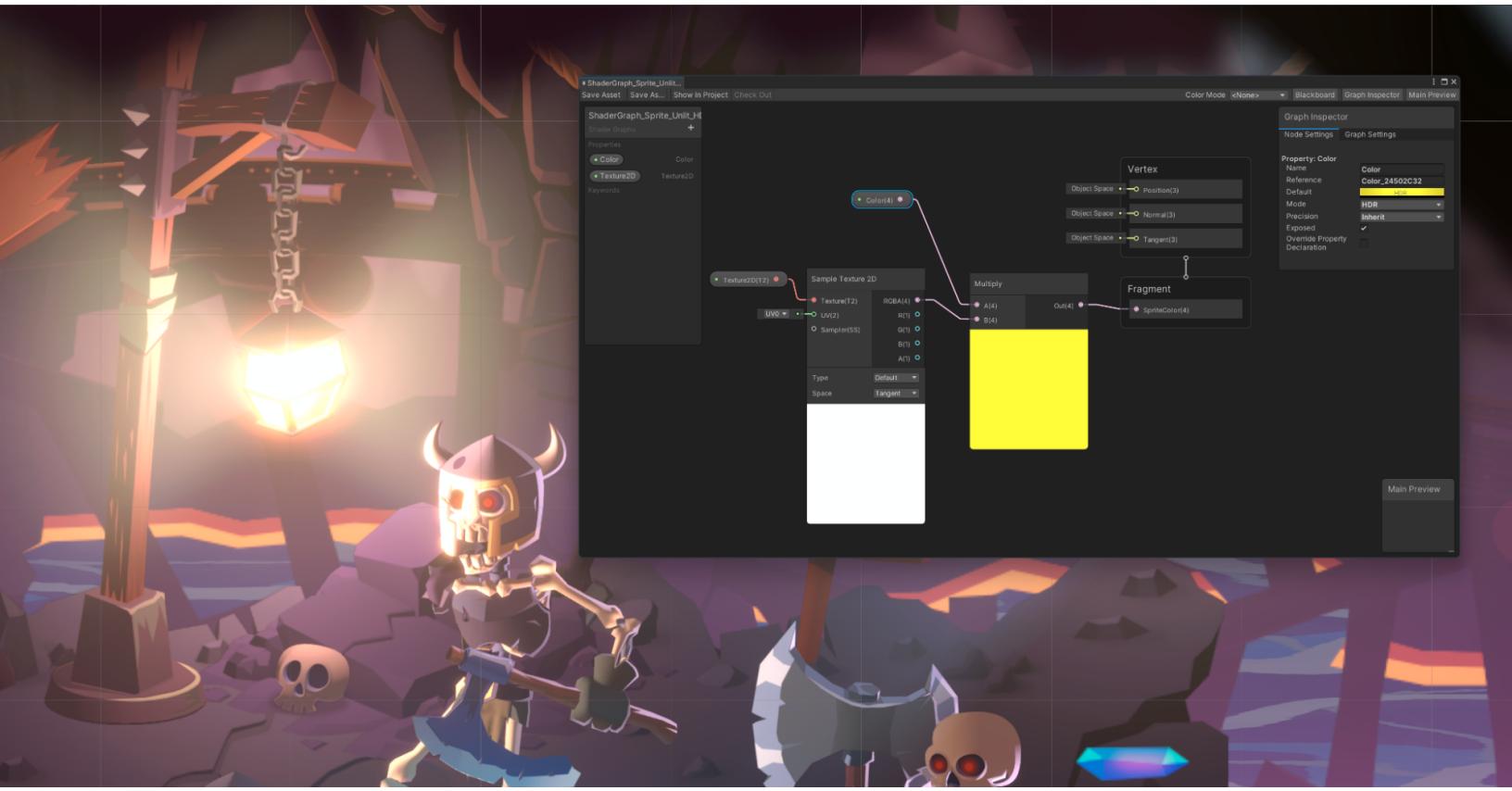


溶岩効果に使用されるテクスチャと、ゲーム内での外観 (アニメーション化された場合の外観は こちら を参照)

この効果では、フレームごとのアニメーションを必要とせずに、流体や飛び散りの外観を実現します。このアニメーションについては、Surface Tension のコミュニティ投稿で詳しく学ぶことができます。

ライトの周囲にグローを追加

Dragon Crashers のもう 1 つの効果として、ランタンからの強い光が挙げられます。これは、HDR 強化スプライトシェーダーと Bloom ポストプロセスエフェクトを組み合わせることによって作成されます。



Dragon Crashers のHDRシェーダー (ShaderGraph_Sprite_Unlit_HDRTint)

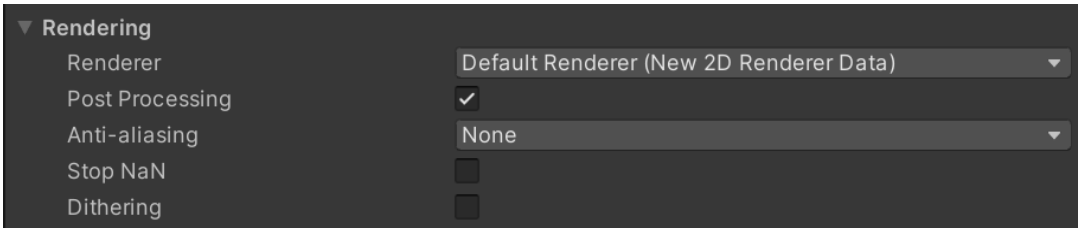
この効果を作成するために、HDR カラーモードを使用するカラーノードでテクスチャを乗算します。Material では、HDR 値を 1 を超えるように上げ、メインカメラで Bloom ポストプロセスエフェクトを有効にします。

ポストプロセス

ポストプロセスは、最終的な画像フレームに効果を追加することによって、ゲームの外観を向上させます。これらの効果は、物理的なビデオカメラの外観をシミュレーションすることも、完全に様式化することもできます。URP には ポストプロセス機能 が含まれているため、別のパッケージをインストールする必要はなく、効果の設定は簡単です。

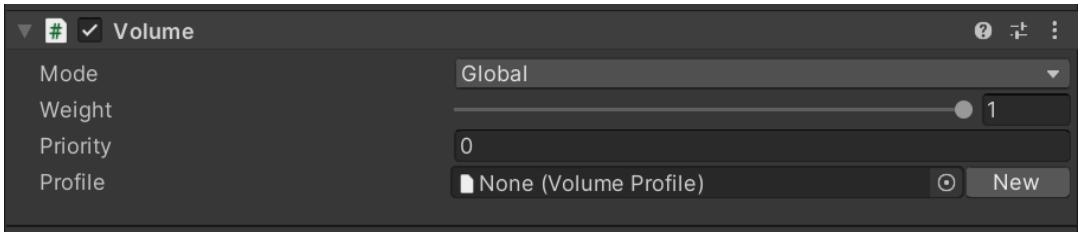
(左) ポストプロセスなしのカメラ出力の画像。(右) ポストプロセスが加えられた同じ画像。効果は違いを示すために誇張されています。ポストプロセスエフェクトは慎重に適用してください。特にモバイルゲームでは注意が必要です。

まず、メインカメラを選択し、Post Processing オプションを選択します。



メインカメラで Post Processing を有効化

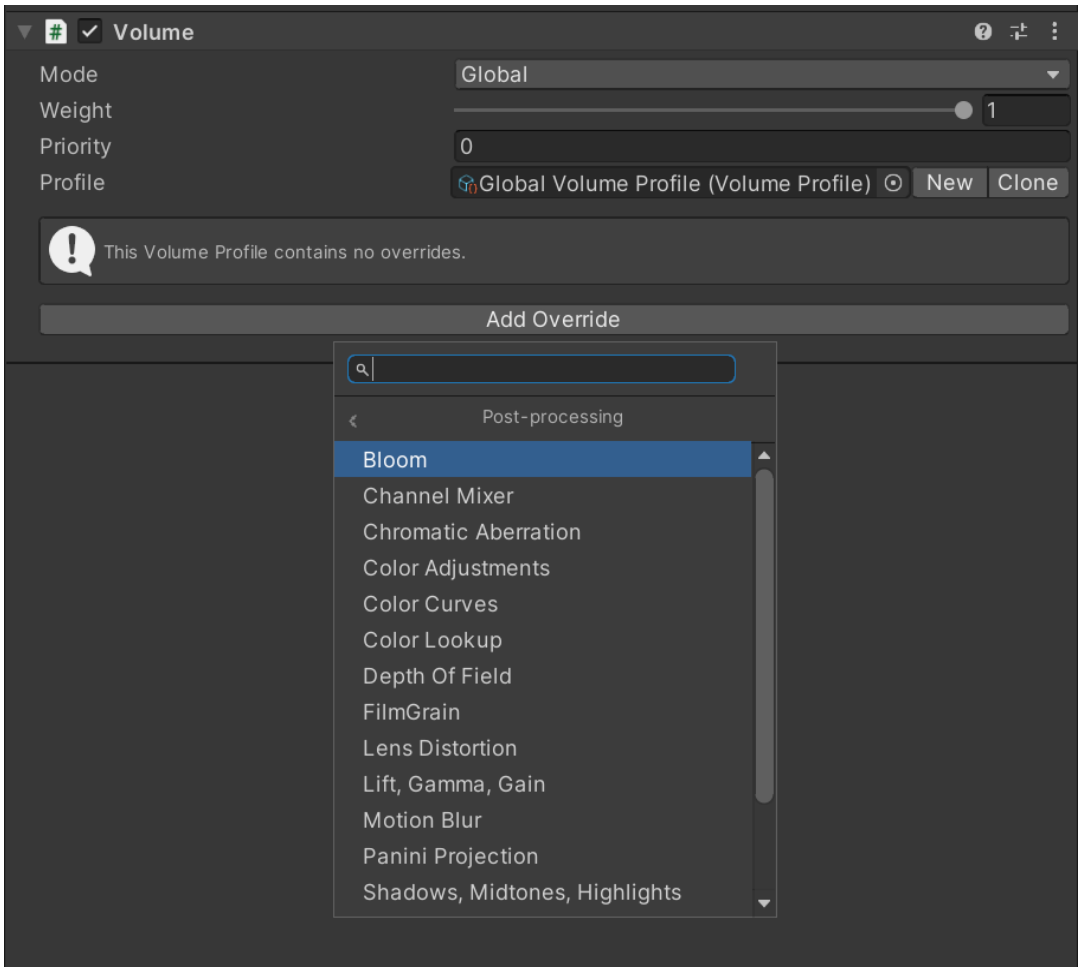
ポストプロセスは ボリューム システムを使用しているため、新しいポストプロセスボリュームを追加するには、**GameObject > Volume > Global Volume を選択します。**



ボリュームプロファイルの定義

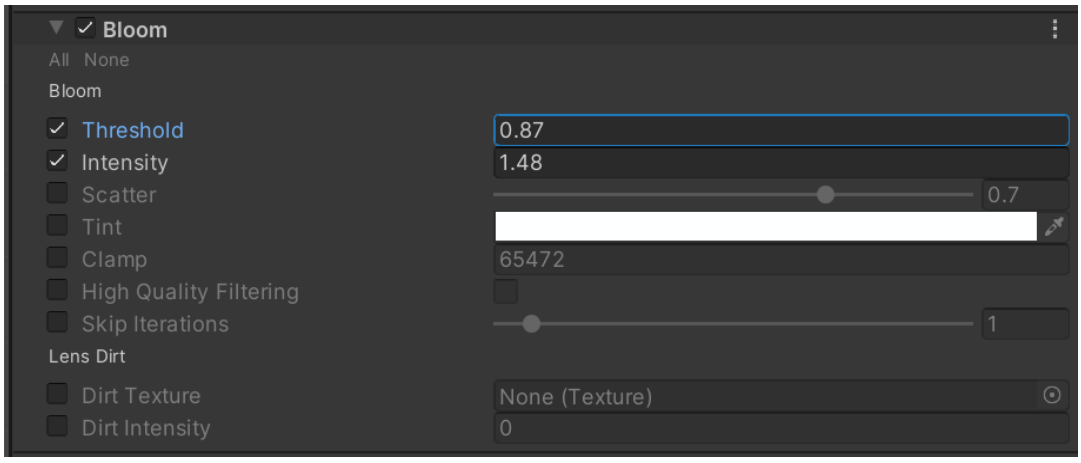
New ボタンをクリックして Volume プロファイルを作成します。Volume プロファイルで、ポストプロセスエフェクトを保存し、ボリュームやシーン間で簡単に共有できます。

ポストプロセスエフェクトを加えるには、Add Override ボタンをクリックし、リストから希望の効果を選択します。



Bloom ポストプロセスエフェクトの選択

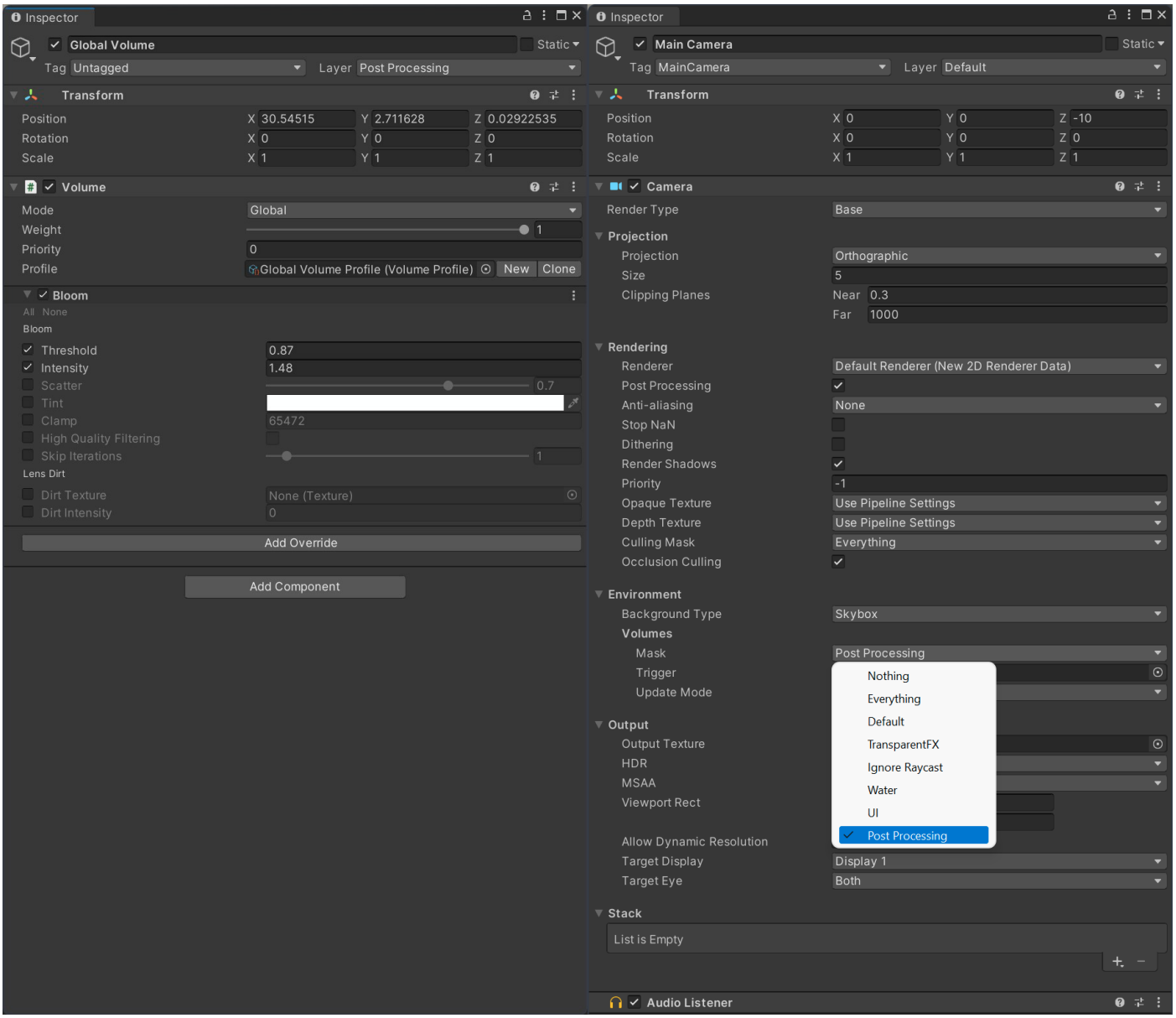
まず始めのブルームを選びましょう。



ブルーム効果のプロパティの定義

効果のプロパティを調整するには、隣接するチェックボックスを選択します。

確認:　Global Volume のゲームオブジェクトレイヤーがメインカメラの Mask　オプションで選択されたレイヤーと一致するようにしてください。



カメラの Post Processing チェックボックスが有効になっていること、そして Volume のゲームオブジェクトのレイヤーがカメラのボリュームマスクのドロップダウンリストで選択されていることを確認してください。

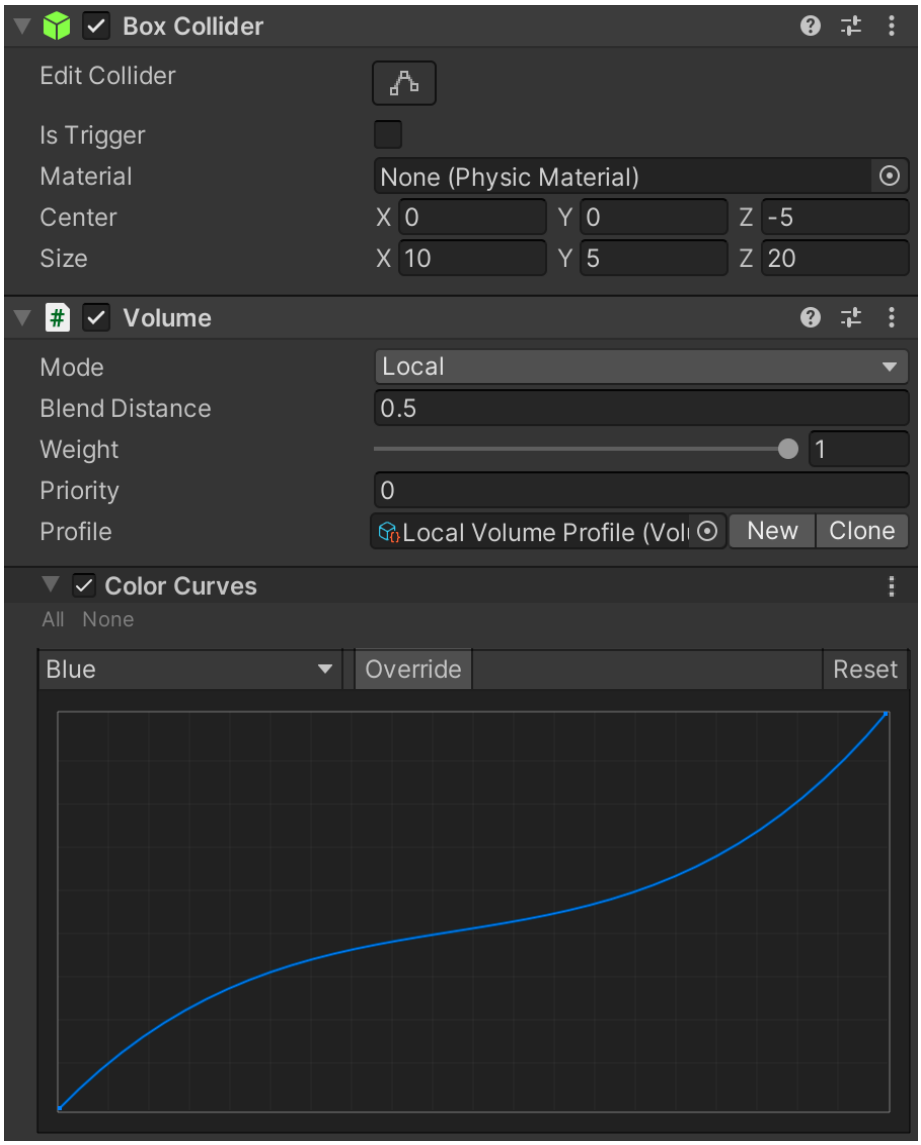
すべてが正しく機能していれば、ゲームビューにブルーム効果が表示されるはずです。

グローバルボリュームが使用されているため、ボリュームオブジェクトがシーンで有効なときは、効果が常に表示されます。

ローカルボリューム

レベル全体にポストプロセスエフェクトを追加できますが、プレイヤーが建物に入るときなど、レベルの一部に対して効果を設定することも可能です。

そのためには、異なるポストプロセスプロファイルが付属した別の Volume オブジェクトを作成する必要があります。



Volume オブジェクトに 3D Box Collider を追加

Volume の Mode を Local に設定し、次に 3D Box Collider をボリュームオブジェクトに追加します。カメラが Box Collider の境界に入るとポストプロセスが適用されるため、カメラオブジェクトを入れるために十分な大きさで Z 軸に沿っていることを確認してください。デフォルトではコライダーが小さすぎるため、シーンビューでサイズを再確認してください。

Blend Distance の値を増加させることで、カメラがボリュームゾーンに入ったときに効果が急にオンになることを防ぐことができます。


Unity®