Unity

プロ開発者向け UNITY スタートアップガイド

2020 LTS EDITION

ACCELERATE SOLUTIONS – GAMES → E-BOOK

目次

最初のステップ

ゲームを新しいプラットフォームやエンジンに移行させる作業は、はるかな旅路を行くようなものですが、取り組むだけの価値は十分にあります。Unity への移行によって、ゲームの成功を今日、明日、さらに将来にわたって実現するための基盤が整います。

Unity の強力な 3D エンジンがあれば、開発チームは最先端のツールを活用しながら、複数プラットフォームへの移植に取り組み、技術を絶えず更新あるいは改善して、ゲームの将来性を確保することができます。もちろん、事業としての収益性が犠牲になることもありません。

有益な探究でも壮大な冒険でも同じですが、物事はまず一歩踏み出すところから始まります（このガイドをダウンロードしたことがその一歩です）。

業界経験のある方なら、このガイドでも取り上げる設計とプランニング、開発とテスト、ローンチとメンテナンスにおいて、さまざまな壁に直面した経験があるはずです。Unity プラットフォームという新しい領域で、このような障害にうまく対処しながら業務を進めていくために役立つのが、本ガイドです。

これまでに Unity を使ったことがある方でも、ゲーム開発を以前より迅速に、効率的に、そして楽しくする仕組みを備えた新機能があることに気づくでしょう。

PC 版のアクション RPG でも、モバイル版の最新の物理演算パズルゲームでも、Unity エンジンは移行にまつわる面倒な作業を肩代わりしてくれます。その結果、開発者はインタラクティブで没入できるリアルタイム体験を作り出すという本来の仕事に集中できるのです。

Unity に初めて移行する場合にサポートが必要な場合は、Accelerate Solutions チームがカスタムの開発パッケージを用意し、エンドツーエンドで支援します。

Unity を活用して魅力的なゲームを制作し、さまざまなデバイス向けにリリースしているアーティスト、デザイナー、開発者のアクティブユーザーは、月間 150 万人に上ります。このガイドを読了したら、その仲間に入るための準備が整います。

ご自分のゲームを Unity エンジンで動かしてみませんか。さあ、始めましょう。

主な機能の概要

ビジュアルエディター

Unity エディターの包括的なインターフェースは、プロトタイピングとテストのサイクルを短縮し、時間をかけずにコンセプト実証から開発ビルドを制作するために役立ちます。

コンポーネントベースのアーキテクチャ

Unity シーンに含まれるあらゆる要素が**ゲームオブジェクト**です。このゲームオブジェクトを、再利用できるモジュール化パーツである**コンポーネント**と組み合わせて、さまざまな機能を実現できます。Unity のカメラは、単純にゲームオブジェクトとカメラコンポーネントを組み合わせたものです。ライトはゲームオブジェクトと Light コンポーネントの組み合わせです。こうしたコンポーネントベースのシステムは、シンプルで柔軟性と拡張性に優れています。

C#

Unity では、世界でも特に利用者数の多い業界標準のプログラミング言語である C# を使用します。C# はオブジェクト指向かつタイプセーフの言語であり、大規模なコミュニティが存在し、手厚いエンタープライズサポートを利用できます。

C をベースとした言語や Java に慣れ親しんでいる人であれば、それほど多くの学習をしなくても Unity のスクリプティングを始めることができます。Unity エディターがそのままではプロジェクト要件に合わなくても、独自のスクリプトを書いてコンポーネントを作成し、ビルトインのコンポーネントとシームレスに連動させることができます。

Unity は、Visual Studio、VS Code、JetBrains Rider などの各種 IDE をサポートしています。詳細については、下記の統合開発環境（IDE）サポートをご覧ください。

プレハブワークフロー

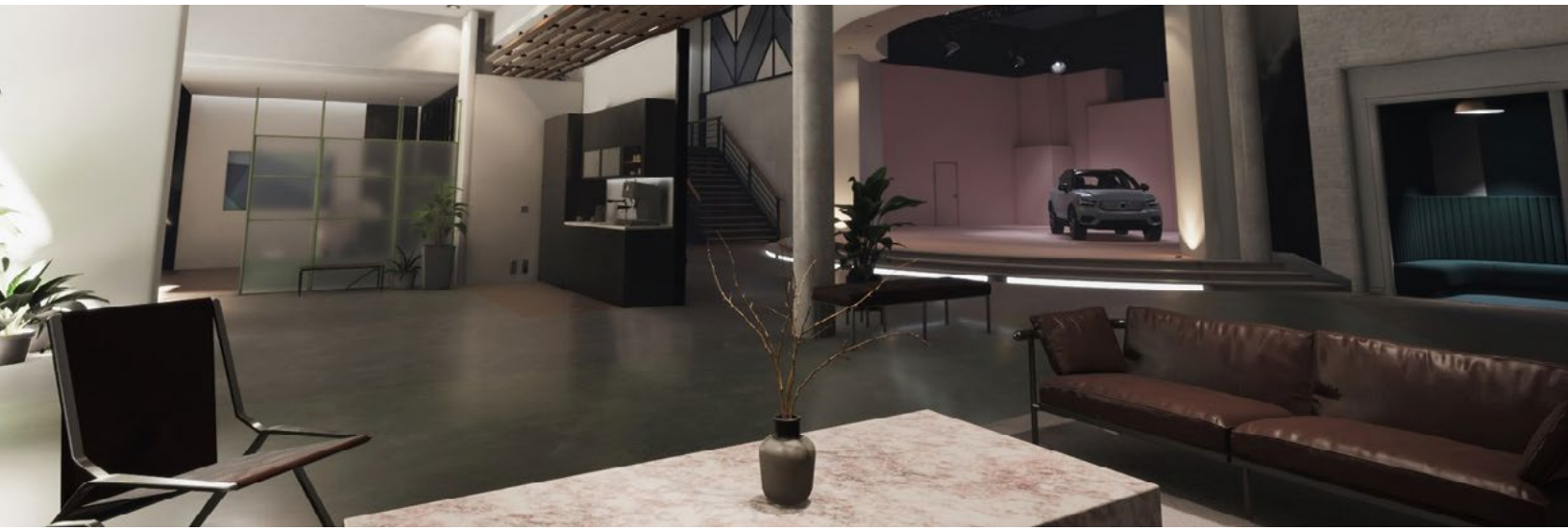
一度作成したものを再利用できれば、同じものを 2 回作成する必要がなくなります。時間を節約し、ランタイムのパフォーマンスを向上させることができます。**プレハブ**を使用して、ライブラリにアセットとしてゲームオブジェクトを設定し、ランタイムでインスタンス化することができます。このワークフローではバリアントが利用でき、「テンプレート」のプレハブの一部を「サブクラス」化してオーバーライドすることができます。また、小さいプレハブから複雑な入れ子構造のプレハブを作成することで、加えた変更をただちにシーンに反映できます。

Unity Hub

Unity Hub は、Unity のインストールやエンジンのバージョン、ライセンス、プロジェクトの管理に役立つフロントエンドのアプリケーションです。これによりプロジェクトを Unity の特定のバージョンに関連付けて、エンジンのアップデートで開発が滞ることや、ローカルで競合が発生することを防止できます。また、Unity Hub から、有用なトレーニング資料や活気のある Unity コミュニティにすばやくアクセスできます。

レンダーパイプライン

Unity では、既製のレンダーパイプラインを使い分けて、グラフィックスを画面に描画することができます。ユニバーサルレンダーパイプライン（URP）は幅広いプラットフォーム互換性とパフォーマンスの最大化を目的としており、HD レンダーパイプライン（HDRP）は PC やコンソール向けの忠実度の高いビジュアルを実現します。また、HLSL を使用してシェーダーを作成することも、ノードベースのシェーダーグラフツールでコーディングを行わずにシェーダーを作成することもできます。



HDRP は忠実度の高いビジュアルを制作するための機能。

マルチプラットフォーム開発

Unity は、「1 回ビルドすれば、どのプラットフォームにも展開できる」という理想に向けて、開発者の支援に取り組んでいます。私たちの目標は、皆様の製品をできるだけ多くの人に届けるとともに、皆様が業界の発展に取り残されず自社の IP を進化させていくお手伝いをすることです。Unity は、これまでに何千ものスタジオでこの目標を達成してきました。その一例として、『Subnautica』シリーズのクリエイター、Unknown Worlds のケースをご覧ください。

エディターでコンテンツを制作すれば、PlayStation®、Xbox®、Nintendo Switch ™、PC、モバイルなど、20 を超えるプラットフォームにわたって展開することができます。Unity で WebGL 用のビルドや、Oculus Quest® などの XR プラットフォーム用のビルドを作成することも可能です。

パッケージと拡張性

パッケージマネージャーを使用すると、エディターに頼らずに機能を探し、更新することができます。Unity ではコア機能を拡張するパッケージのコンテンツリポジトリが保持されます。

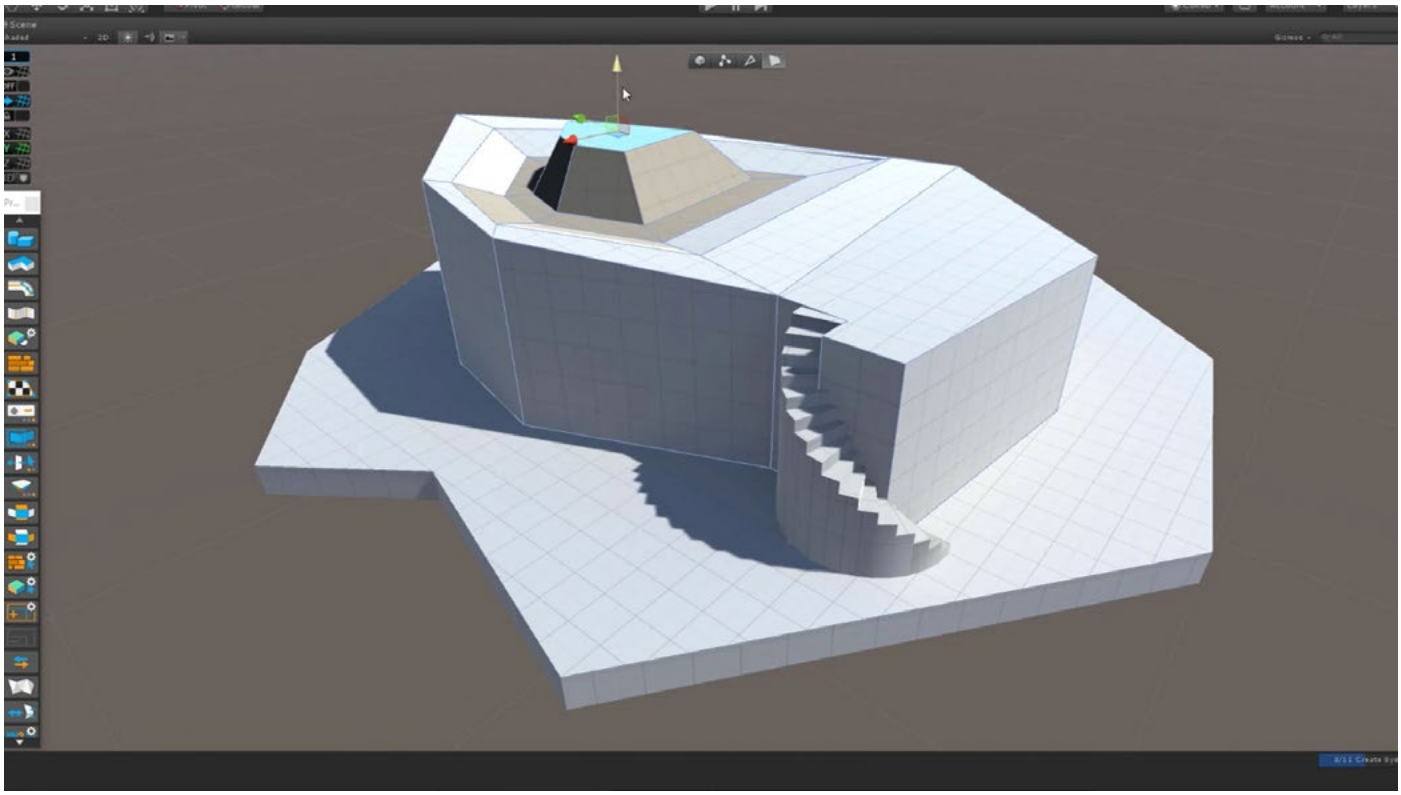
プロジェクトの無駄をなくして、必要なものだけをインポートしましょう。パッケージを使用すると、製品版を作成する前にプレビューや実験で機能を試すことができるため、将来のプロジェクトに取り入れる可能性のある新しいテクノロジーを事前に計画できます。

*Nintendo Switch は任天堂の商標です。

ワールド構築

Unity には、環境を構築するツールのスイートが含まれています。ProBuilder は、3D モデリングツールとレベルデザインツールのユニークなハイブリッドツールです。シンプルなジオメトリをモデリングして、ゲームレベルをホワイトボックス化できます。

2D 作業の際には、Unity の SpriteShape や Tilemap といったツールが 2D 環境のレイアウト作業に役立ちます。環境アーティストは、テレインを使用して自然なランドスケープを生成し、付属のブラシツールでペイントすることができます。



ProBuilder はレベルのプロトタイプ作成に役立つ。

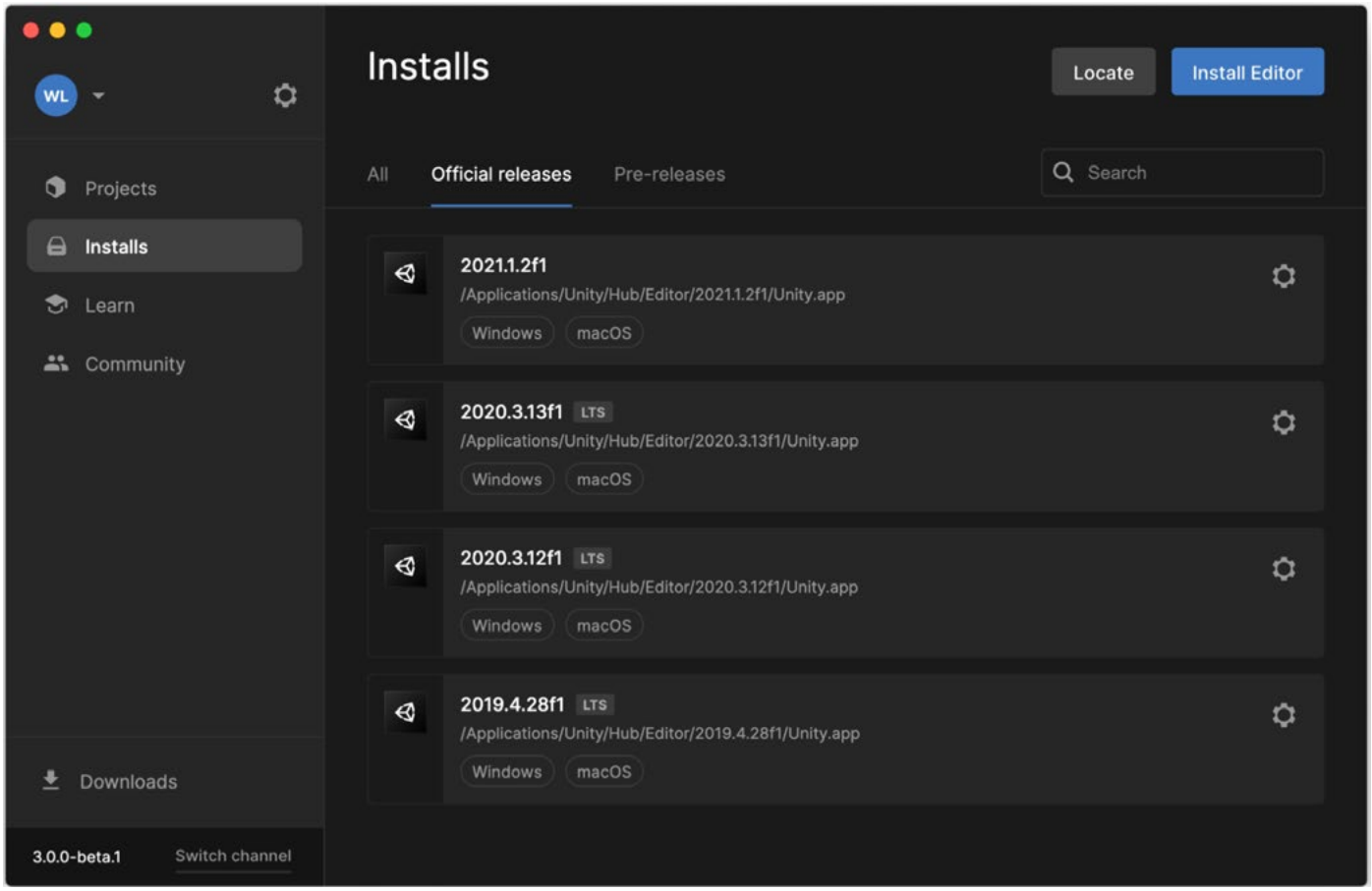
アクティブコミュニティ

Unity はさまざまな層に幅広く働きかけ、活発なユーザーベースを抱えています。Apple App Store や Google Play で配信されている上位 1,000 タイトルのモバイルゲームを見ると、その半数以上の制作やマネタイズに Unity コアプラットフォームが使用されています。月間のアクティブクリエイター数は 100 万人を超えているため、公式の Unity フォーラムやその他のオンラインコミュニティの大規模なナレッジベースを活用して、開発上の問題を調査することができます。

UNITY HUB

ここでは、Unity Hub という、Unity のプロジェクトとインストールをすべて管理するツールについて説明します。Unity Hub を使用して、1 つ以上のバージョンの Unity エディターをインストールできるほか、新しいプロジェクトを作成したり、既存のプロジェクトを開いたりできます。

Unity Hub を入手するには、Unity のウェブサイトにアクセスし、Unity Hub のダウンロードを選択します。インターフェースを使用して Unity をインストールし、個人向けまたはチーム向けのライセンスをアクティベートして、エンジンを起動します。Unity Plus と Pro ライセンスの場合、有効なシリアル番号も必要になります。



Unity Hub はプロジェクトやインストール済みバージョンの管理に役立つ。

Unity Hub で、「Projects」、「Installs」、「Learn」、「Community」の各画面を見ていきます。

「Installs」画面では、プレリリースと正式リリースの両方を含むさまざまなバージョンの Unity を選択してインストールできます。プラットフォーム固有のサポートや開発ツールを細かく管理する必要のあるモジュールだけを追加します。

製品開発には、最新版の長期サポート（LTS）バージョンの Unity を使用することをお勧めします。Unity 2020.3 LTS の特長は以下のとおりです。

— リリース間近のプロジェクトを進行するための安定した基盤

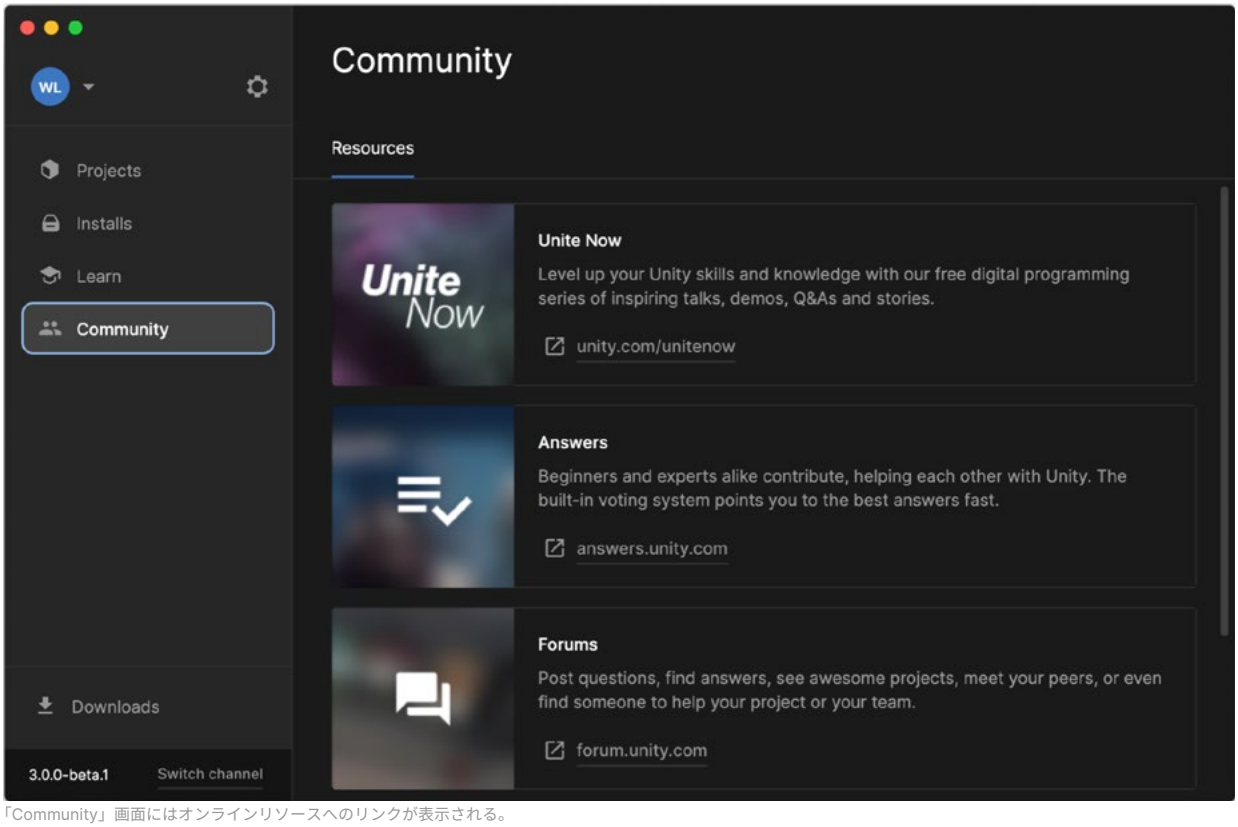
— 2022 年中頃までの隔週更新と、それ以降、2023 年 3 月までの月一回更新（初回リリース日から 2 年間）

LTS バージョンの更新では、エンジンの安定性向上を目的としたユーザビリティ修正のみが行われます。

アクティブなプロジェクトは、**「Projects」**画面に表示されます。各プロジェクトに特定のターゲットプラットフォームを関連付け、特定の Unity バージョンに対する開発をロックすることで、更新時に開発を中断させるバグが発生しないようにします。

「Learn」画面では、多数の教育リソースとチュートリアルを利用して、エンジンを使いこなすための学習に取り組むことができます。

「Community」画面では、Unity ブログや Unite Now トークなど、他の一般的なリソースを利用できます。技術的な疑問点がある場合は、質問をしたり、「Answers」ページや「Forums」ページの議論を参照したりすることができます。

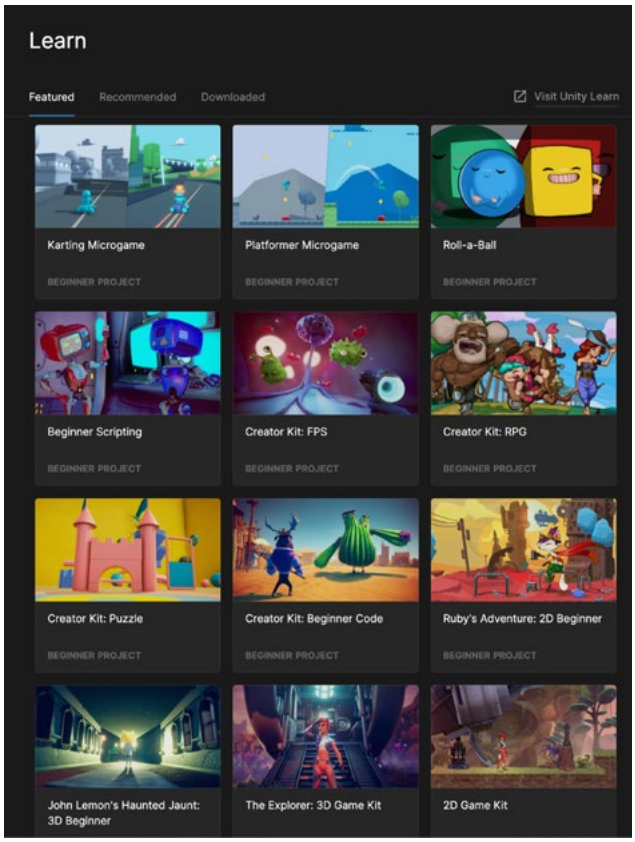


「Community」画面にはオンラインリソースへのリンクが表示される。

サンプルプロジェクト

開発環境に慣れる最適な方法は、おそらく、ゲームプロトタイプのバーティカルスライスを見て参考にすることです。そのような例はさまざまな場所で公開されており、Unity を知るための出発点となります。

プロジェクトを学ぶ
Hub の「Learn」画面から、さまざまなチュートリアルや学習リソースが利用できます。ダウンロードして Unity に直接インポートできるサンプルプロジェクトも数多く用意されています。



「Learn」画面でスタータープロジェクトを探す。

たとえば、クリエイターキットと Microgame は、文書化されたコードサンプル付きのミニテンプレートです。これらのプロジェクトの多くは、初心者が「コードフリー」で利用できるほか、経験のある開発者がスクリプトやアセットを解析して学習に役立てることもできます。「Learn」画面にあるプロジェクトの内容を調べて修正すると、Unity での開発で一般的に使用される制作テクニックを理解できます。

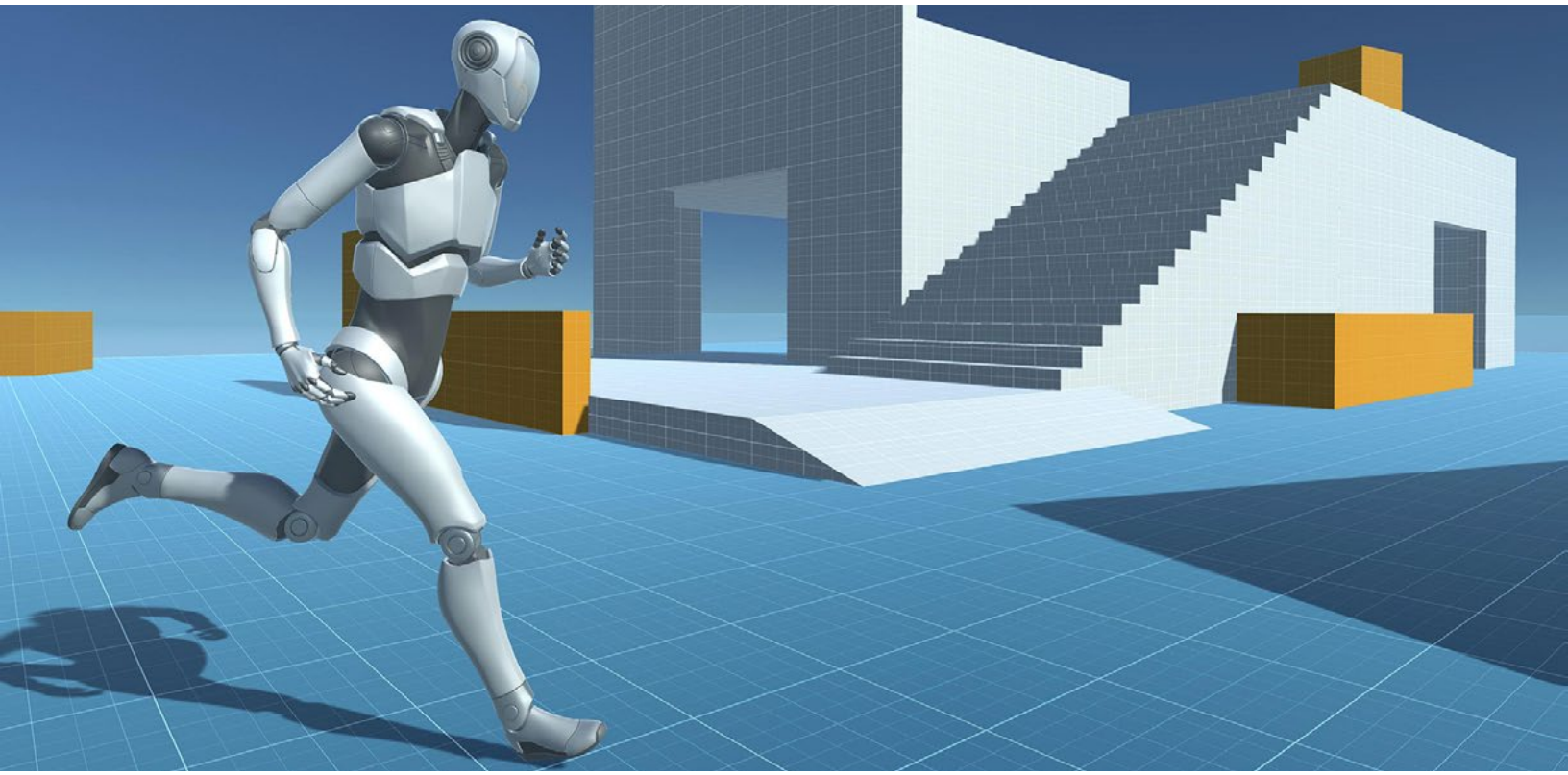
Unity では、変更を加えたプロジェクトを保存できます。そのプロジェクトは Unity Hub に表示され、後で作業を続けることもできます。変更を加える前の元のプロジェクトも保持され、「Learn」画面から利用することができます。

UNITY ASSET STORE のアセット

Unity Asset Store はコミュニティマーケットプレイスであり、無料アセットとプレミアムアセットが数多く提供されています。社内でツールやアートアセットを開発しない場合は、サードパーティ製の該当ツールやアセットをここで探すことができます。完成済みアートやプロトタイピング向けアート、エディター拡張機能やスクリプティングツール、あるいはすぐにでも製品化できる完結したプロジェクトにいたるまで、さまざまなアセットが用意されています。多くのアセットパッケージに Unity 開発に欠かせない部分をカバーしたテンプレートが多数含まれているため、開発作業を数日あるいは数週間短縮できます。

Asset Store では数万個のアセットが公開されており、日々さらにパッケージが追加されています。Unity から入手できるパッケージは、制作を始めようとしている人にとって格好の素材です。

— Starter Assets パッケージには一人称視点または三人称視点の軽量なキャラクターベースコントローラーが含まれており、無料で利用できます。これを使って、Cinemachine や Input System パッケージを使用してカメラをコントロールする方法を確認することができます。Starter Assets – First Person Controller と Starter Assets – Third Person Controller では、ゲーム世界のシーン内でキャラクターコントローラーのプロトタイプを作成する方法を学ぶことができます。



Starter Assets – Third Person Controller

— Lost Crypt – 2D Sample Project は、Unity の 2D 向けツールが連携して動作することを示す 2D 横スクロールデモです。このプロジェクトでは、2D アニメーション、2D スプライトシェイプ、2D タイルマップ、2D ライト、2D 用シェーダーグラフ、補助的なテクスチャ、ボリュームポストプロセッシングなどの機能が紹介されています。プロジェクトの詳細については、こちらのブログ記事で確認してください。

Lost Crypt による Unity の 2D 機能のデモ。

Unity 歴の浅い方は、Asset Store チームが選んだ新規ユーザー向けのお勧めもご覧ください。

Unity GitHub リポジトリ

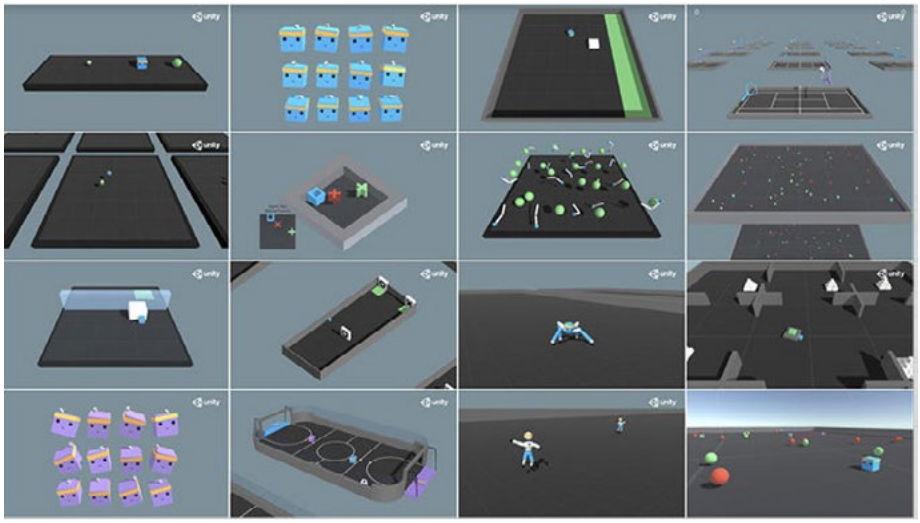
Unity Technologies の GitHub リポジトリには、特定の機能やトピックを実装したプロジェクトが複数用意されています。すべてのプロジェクトが最新バージョンの Unity で動作するわけではありませんが、Hub にアクセスして、適合する Unity バージョンをインストールできます。多くのリポジトリは実験的またはプレビューの段階であるため、新しい機能を評価して今後の制作に取り入れることができます。

注目のプロジェクトをいくつか紹介します。

— Boss Room は、小規模な協力型ゲームのサンプルプロジェクトで、新しい実験的な NetCode ライブラリと MLAPI をベースに構築されています。このプロジェクトは、1 階層のダンジョンを舞台に 4 種類のキャラクタークラスを使ってプレイするマルチプレイヤーゲームのフローの基盤となっている概念やパターンを紹介し、様式化された敵と、タイトルにもなっているボスと戦うゲームプレイを実現できるようになっています。

Boss Room

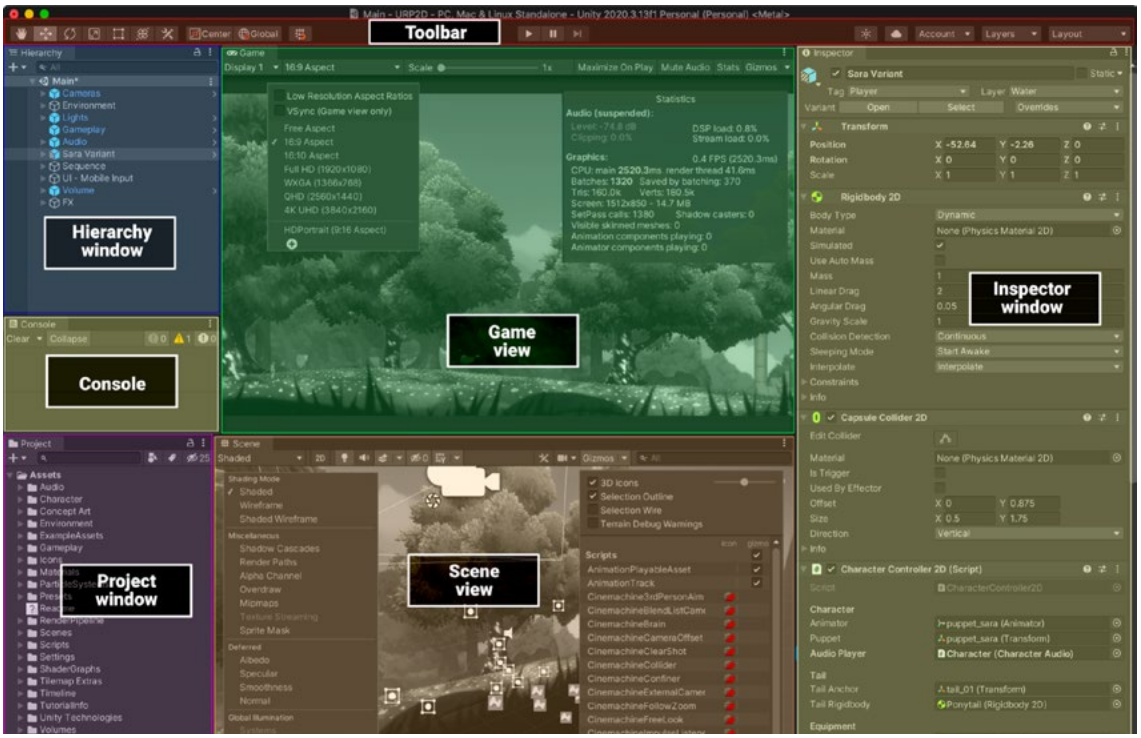
— Unity Machine Learning Agents（ML-Agents）ツールキットは、ゲームやシミュレーションで 2D、3D、VR/AR アプリケーション向けの知的エージェントをトレーニングできるオープンソースのプロジェクトです。研究者も既存の Python API を使用して、教科学習や模倣学習などの機械学習の手法をゲーム内アクターに適用することができます。



ML-Agents ツールキット

エディターの
インターフェース

Unity は柔軟なモジュラー方式のユーザーインターフェースを備えています。Unity エディターを使用したことがない場合は、こちらでメインウィンドウの簡単な説明をご覧ください。



エディターのメインウィンドウ

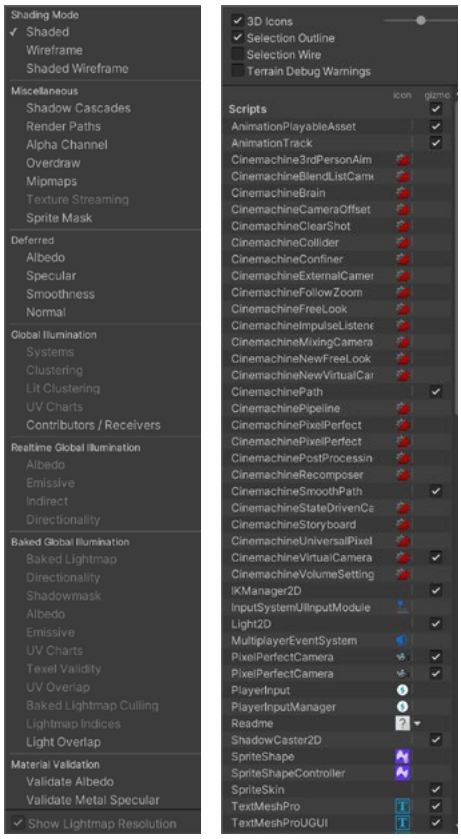
メインウィンドウ

Game ビュー：ゲームのメインカメラ、およびプレイヤーによるゲーム操作をプレビューできます。右上にあるレンダリングの「Statistics」ウィンドウには、グラフィックとオーディオの統計情報が表示され、アプリケーションの最適化に役立ちます。左上に表示される「Aspect Ratio」で、複数の画面解像度をプレビューしたり独自の目標解像度を定義したりできます。

レンダリングの統計情報

アスペクト比

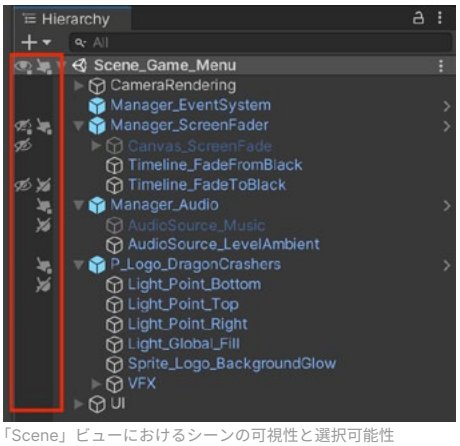
Scene ビュー：このビューは、各シーンを構成するオブジェクトを表示します。エディターでは、このビューでモデルやライト、物理ボディなどを配置する作業が中心になります。コントロールバーを使用してドローモードを変更でき、トラブルシューティングや簡潔な診断を行う際に便利です。ギズモメニューでは、アイコンや特定のゲームオブジェクトのオーバーレイグラフィックを設定できます。



ドローモードとギズモメニュー

「Hierarchy」ウィンドウ：このウィンドウには現在のアクティブなシーンに存在するオブジェクトと、各オブジェクト間の親子関係が表示されます。

「Hierarchy」ウィンドウには、「Scene」ビューの可視性と選択可能性を切り替えることができる便利なインターフェースが用意されています。この UI は「Game」ビューには影響しません。非常に多くのゲームオブジェクトを含む複雑なシーンで、要素を見やすく表示して操作することができます。



「Scene」ビューにおけるシーンの可視性と選択可能性の切り替え

「Inspector」ウィンドウ：このウィンドウには、選択したオブジェクトのプロパティと設定が表示されます。機能的なカスタマイズオプションが用意されています。フォーカスされたインスペクターを使用すると、特定のゲームオブジェクト、アセット、コンポーネントに対応する「Inspector」ウィンドウを開いて、ゲームプレイ変数やシーンオブジェクトをわかりやすく比較することができます。インスペクターを使用すると、毎回コードを調整しなくてもランタイムで要素を調整できるので、対話的なテストプレイが可能になり、ワークフローの迅速化が進みます。

「Project」ウィンドウ：Unity ではゲームを構成するすべてのアセット（プレハブ、テクスチャ、モデル、アニメーション、スクリプト）をディスク上のファイルとして保管します。「Project」ウィンドウでは、このようなファイルにすぐアクセスすることができます。ゲーム内リソースが納められたコンテンツディレクトリやライブラリのようなものと考えてください。「Search」バーから、個々のアセットや特定のタイプのアセットを検索できます。

ツールバー：エディターの最上部で、アプリケーションウィンドウのタイトルバーに接して表示されます。ツールバーのボタンでエディターの再生モードを調整して、リリースするアプリケーションがどのように動作するかを確認できます。ハンドツールはシーンをパンするときに使用し、「Move」「Rotate」「Scale」「Rect Transform」「Transform」の各ツールはゲームオブジェクトの操作に使います。

「Console」ウィンドウ：「Console」ウィンドウはデバック用のウィンドウで、Unity が生成するエラーや警告、その他のメッセージが表示されます（Debug クラスを参照）。「Console」ウィンドウに書き込まれる内容は、Unity によるものも開発者が作成したコードによるものも、すべてログファイルにも出力されます。

エディターの再生モード

「Game」ビューはアプリケーションのカメラからの視点でレンダリングされます。これはリリース用の最終版アプリケーションを表しています。再生モード中に行った変更は一時的なものであり、再生モードを終了するとリセットされます。再生モード中の変更は保存されないということに注意を促すために、エディターの UI は暗くなります。

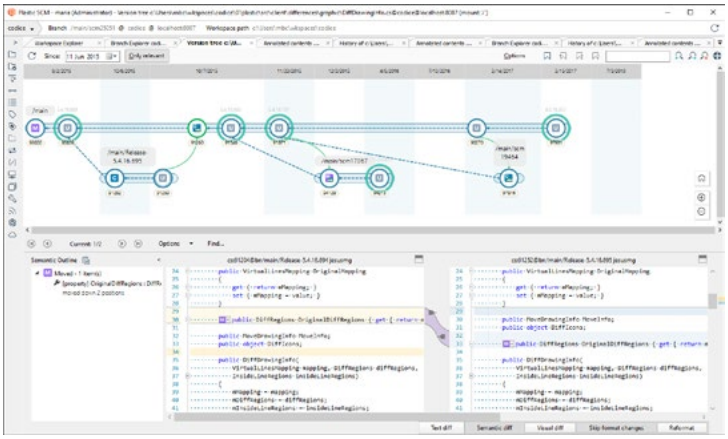
ランタイムでプレイヤーが見るものを制御するには、1 つ以上のカメラを使用する必要があります。詳しくは、Camera コンポーネントのページを参照してください。

インターフェース各部の詳しい情報については、Unity Learn の概要も参照してください。

バージョン管理

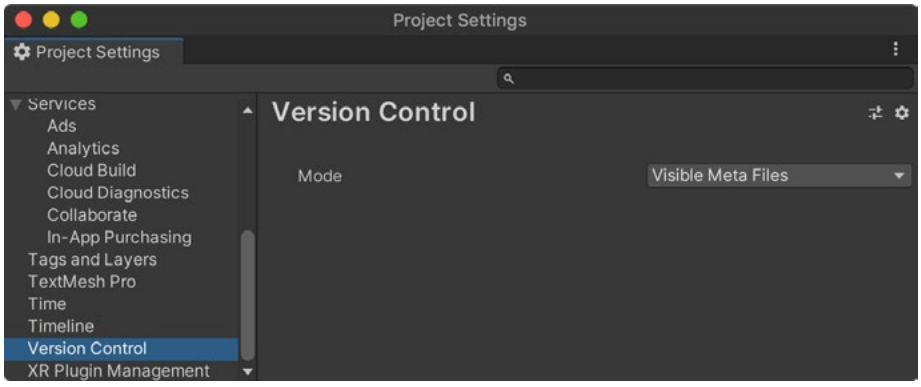
Unity は Plastic、Git、Perforce などのシステムに対応しており、ユーザーやチームに最適なソース管理ソリューションを選択することができます。

Unity のエディタには、Perforce と Plastic SCM という 2 つの代表的なバージョン管理システムが統合されています。Perforce サーバーと Plastic SCM サーバーのどちらかをプロジェクトに設定し、Unity と組み合わせて使用します。



PlasticSCM の合理的なインターフェースで簡潔なバージョン管理を実現。

ソース管理機能の使用を開始するには、**「Project Settings」>「Editor」**に移動します。**「Asset Serialization」の「Mode」**が**「Force Text」**に設定されていることを確認します。Unity では、ディスクに対するアセットの保存とロードを行うときにシリアル化を使用します。「Force Text」モードは、シーンファイルをテキストベースの形式で保管する、バージョン管理のマージに適したモードです。



「Project Settings」の「Version Control」

「Project Settings」>「Editor」>「Version Control」に移動します。使用するバージョン管理システムに合わせて、モードを**「Plastic SCM」**、**「Perforce」**、または**「Visible Meta Files」**（Git などの外部ソース管理用）に切り替えます。

Plastic SCM

Plastic SCM

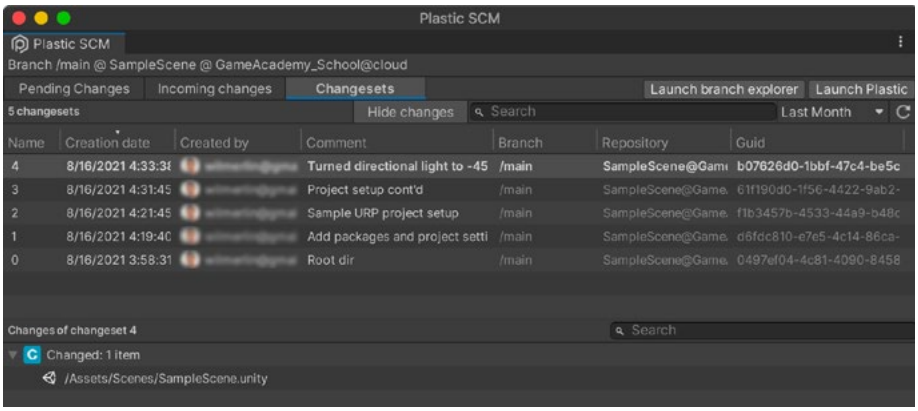
Plastic SCM は Unity でのゲーム開発用に推奨されるバージョン管理システムです。このソリューションは大きなバイナリファイル（500 MB 超）を快適に扱うことができるため、コードに対して想定するレベルと同じレベルの管理をアートアセットに対しても実施できます。

Plastic SCM によって、アートとプログラミングアセットの両方が確実にバックアップされていることを把握できます。加えて、直観的な視覚的インターフェースにより、ブランチングとバージョン管理が簡素化されます。

以下に Plastic SCM の主なメリットをいくつか示します。

- **スピード：**大規模なコードベースでブランチを作成する場合、Plastic は Perforce や Git よりもはるかに高速で処理できます。
- **ノンプログラマーのためのシンプルさ：**チームのアーティストは、Gluon モードで作業することができます。Gluon のシンプルなワークフローでは、アーティストがプロジェクトの一部をチェックアウトして作業し、変更をコミットしつつ、インターフェースの無関係な部分を削除できます。
- **クラウドホスティング：**Plastic では Plastic Cloud Edition と呼ばれる、完全分散型のチームを想定して開発されたネイティブソリューションを利用できます。チームで独自のサーバーを運用するのではなく、オンラインですべてをホスティングしたい場合は、Plastic Cloud Edition をお試しください。
- **分散型バージョン管理：**リモートオフィスごとに個別のリポジトリを設定することで、チームが常に「ローカルで」作業を進めることができます。これは大きな規模で行う Git に似たワークフローです。グラフィカルインターフェースを使用して変更のプッシュやプルを行い、リモートの競合を解決できます。
- **差分ウィンドウ：**Plastic は、変更、追加、または削除されたファイルを確認できる完結したインターフェースを備えています。さらに、2 つのバージョンのテクスチャアセットを比較する際に役立つ、独立した画像差分ツールも用意されています。

Plastic SCM を使用する際には、**「Plastic SCM」**ウィンドウを開くと（**「Window」> 「Plastic SCM」**）、変更リスト内にファイルが表示されます。



「PlasticSCM」ウィンドウ

「Pending changes」タブに、バージョン管理へのコミットが保留されているローカルの変更がすべて列挙されます。**「Incoming changes」**タブでは、新たに発生した変更と競合をすべて確認し、ローカルのプロジェクトを更新することができます。プロジェクトが変更されると、「Plastic SCM」ウィンドウの右上に「Incoming changes」通知が表示され、新しい変更が加えられていることがわかります。

「Version Control」ウィンドウの詳細については、バージョン管理インテグレーションに関するドキュメントを参照してください。さらに詳しい情報が必要な場合には、まずは Unite Now に収録されている『Version Control for games with Plastic SCM』ビデオを視聴するか、Plastic SCM ブックをお読みください。

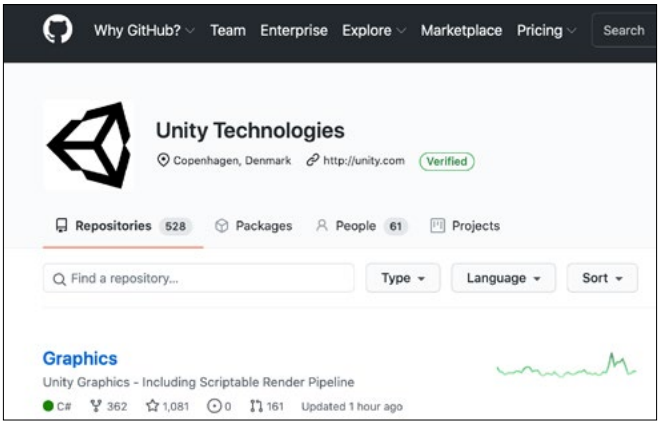
Git

Unity 開発者は Git などの外部ソース管理ソリューションを使用することもできます。ただし、その場合はプロジェクトの初期設定を手動で行う必要があります。

Git と Unity を組み合わせて使用するには、ローカルマシンに空のリポジトリを作成し、必要に応じて GitHub 経由でリポジトリとクラウドを同期させます。グラフィックやサウンドのリソースなど、サイズの大きいアセットのバージョン管理を効率的に行えるよう、Git LFS（Large File Support）を忘れずに導入してください。Unity により管理される .gitignore ファイルを利用して、Git リポジトリに格納すべきデータと格納しないデータを判別できます。

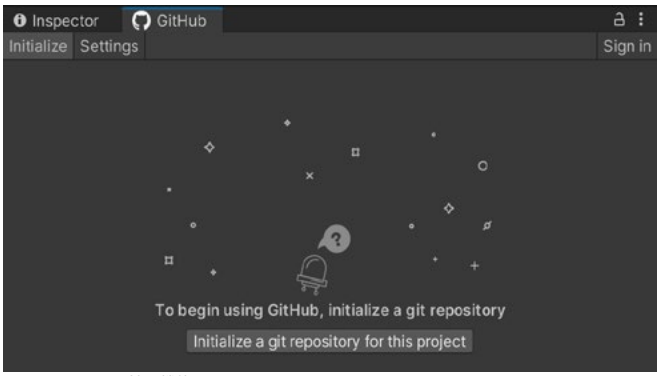
GitHub ホスティングサービスをさらに便利に利用するには、GitHub for Unity プラグインをインストールします。このオープンソース拡張を使用すると、Unity から離れることなく、プロジェクト履歴の閲覧、ブランチでの実験、変更のコミット、および作成したコードの GitHub へのプッシュを行うことができます。

また、GitKraken、SourceTree、GitHub Desktop などの高機能なビジュアルクライアントの使用も検討してみましょう。



GitHub を利用してリポジトリを作成。

『Introduction to Version Control』ビデオでは、Unity で Git を利用するための設定について順を追って説明しています。サンプルプロジェクトを通じて、ビジュアル GitHub クライアントを使用するための基礎知識を学ぶことができます。



GitHub for Unity 拡張機能

Perforce

Perforce でソース管理を行う場合は、Helix Core を使用してコード、アートワーク、ゲームエンジンなどのアセットを管理します。Helix Core は最大で 5 人のユーザー、20 のワークスペースまで無料**です。**

Perforce の優れた性能は、世界中にリモートチームが分散している場合でも十分に発揮されます。数多くの AAA やインディーゲームの開発スタジオが主要なソース管理ツールとして Perforce を使用しています。

使用する際は、以下を参考にしてください。

— Unity のバージョン管理ドキュメントに記載されている設定プロセスに従って手順を進めます。

— Perforce Helix Core に Unity を設定する方法を確認します。

— Perforce Helix Core の詳細については、Perforce のドキュメントを参照してください。

プロジェクトの整理

プロジェクトの成長に合わせて、チームやアプリケーションの要件に対応できるように、常に一定のレベルで整理された状態を保つ必要があります。ここでは、基本的なプロジェクトやシーン構造を確立する際に役立つ一般的なヒントを紹介します。

「Project」ウィンドウ

「Project」ウィンドウには、現在のプロジェクトに関連するファイルがすべて表示されます。これはコンテンツディレクトリであり、アプリケーションのアセットや他のプロジェクトファイルが表示されます。

Unity では、ソースファイルが個別の .meta ファイルと併せてプロジェクトに直接保存されます。この .meta ファイルには、関連付けられたアセットに関するエンジン固有データおよびエディター固有データが含まれています。

また、Unity に各アセットをインポートする際には、エンジンがランタイムで使用する際に最適な形式でインポートされます。このように処理されたアセットは Library フォルダーに格納されてキャッシュとして機能します。ソース管理に追加する必要はありません。

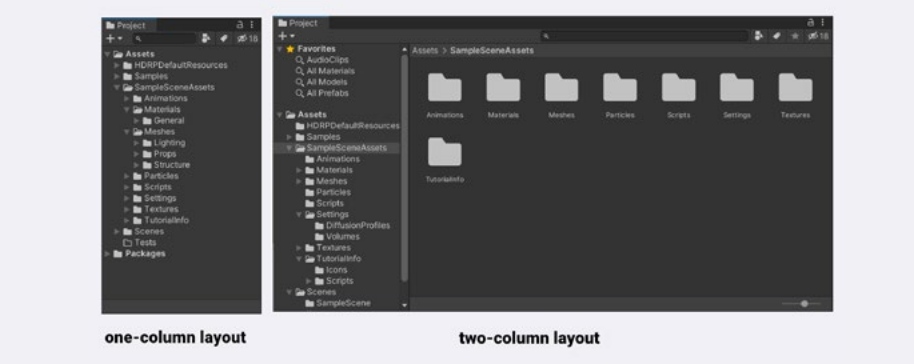
「Project」ウィンドウには、操作に役立つ UI 機能がいくつかあります。

— **右クリック**で表示されるコンテキストメニューから、よく使われる操作（アセットの作成とインポート、ディスクのフルパス表示など）を行うことができます。

— プロジェクトが大型化しても、**「Search」**フィールドを使用してアセットを検索することができます。特定のタイプのアセットを探す場合は、「t:」構文を使用してタイプで絞り込みます（たとえば、**t:Material** と入力すると、プロジェクトに含まれるすべてのマテリアルアセットが表示されます）。この機能は大規模プロジェクトを扱うときに役立ちます。

— よく使用するフォルダーを、インターフェース上部にある**「Favorites」**フィールドにドラッグアンドドロップできます。**「Save Search」**ボタンを押して、「Favorites」に検索条件を保存することも可能です。

— ウィンドウ自体のレイアウトも変更できます。ウィンドウ右上の**「More Items」**（⋮）メニューを選択し、**「One Column Layout」**または**「Two Column Layout」**を選択します。2 列レイアウトでは、追加のペインに各ファイルの視覚的なプレビューが表示されます。

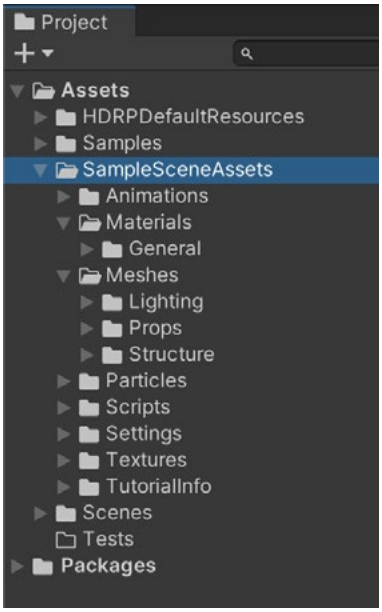


1 列レイアウトと 2 列レイアウト

「Project」ウィンドウの中にあるのが **Assets フォルダーです。**このフォルダーには、ゲームの制作に使用するアセットが含まれます。テンプレートを使用してプロジェクトを開始すると、いくつかの一般的なアセットに対応するサブフォルダーが用意されます。フォルダーのほとんどはユーザー定義のものですが、Unity ではいくつかのフォルダー名が特定の目的のために予約されています。予約されている名前については、「Special folder names」を参照してください。

以下に、プロジェクトの整理によく使われるサブディレクトリを挙げています。チームの好みやプロジェクトの都合により別の名前を使用することもあるでしょうが、重要なのは一貫性を保つことです。スタイルガイドを作成して、それに準拠するようにしてください。

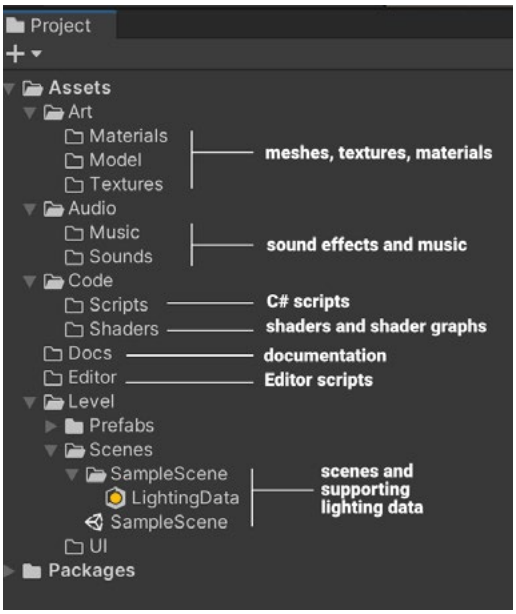
Animations	このフォルダーには、アニメーションを表現するモーションクリップとそのコントローラーファイル、ゲーム内のシネマティクスに関するタイムラインアセット、プロシージャルアニメーション用のリギング情報が格納されます。
Audio	オーディオクリップなどのサウンドアセットのほか、エフェクトと音楽をブレンドするためのミキサーが格納されます。
Editor	このフォルダーには、Unity エディターで使用するために作成されたスクリプトツールが格納されます。ターゲットビルドには表示されません。
Fonts	ゲームで使用されるフォントが格納されます。
Gizmos	ギズモアイコンがあると、「Scene」ビューや「Game」ビューでゲームオブジェクトを視覚化するときに便利で、特にメッシュを持たないゲームオブジェクトの場合に役立ちます。このアイコン用の画像ファイルが Gizmos フォルダーに格納されます。
Materials	サーフェスシェーディングのプロパティを記述するアセットが格納されます。
Meshes	外部の DCC アプリケーションで作成されたモデルが格納されます。
Particles	ParticleSystem や Visual Effect Graph で作成された、Unity でのパーティクルシミュレーションが管理されます。
Prefabs	既製のコンポーネントを使用した再利用可能なゲームオブジェクトが格納されます。これをシーンに追加して、レベルやゲームプレイを制作することができます。
Scripts	ユーザーが開発したゲームプレイ用コードがすべて、ここに格納されます。
Scenes	Unity では、プロジェクトの小さな機能部分がシーンアセットに格納されます。これがゲームレベルやレベルの一部に対応していることがよくあります。
Settings	HDRP と URP 両方のレンダーパイプラインの設定が Assets フォルダーに格納されます。
Shaders	グラフィックパイプラインの一部として GPU 上で実行されるプログラムが格納されます。
Textures	マテリアルやサーフェシング用のテクスチャファイル、ユーザーインターフェース用の UI オーバーレイ要素、ライティング情報が含まれたライトマップなどの画像ファイルが格納されます。
ThirdParty	Asset Store などの外部ソースからアセットを入手した場合、プロジェクトの他のアセットと区別して、ここに格納します。これによりサードパーティ製のアセットやスクリプトを更新しやすくなります。サードパーティ製アセットは変更できないセット構造を備えていることがあります。



HDRP テンプレートが適用された SampleScene に複数のアセットフォルダーが含まれている。

「Supported Asset Types」マニュアルページには、よく使用されるアセットについての詳細が記載されています。Template プロジェクトや Learn プロジェクトは、フォルダーを効率的に整理する方法の例として使用できます。必ずしも上記のフォルダー名に従う必要はありませんが、これはプロジェクトの規模が大きくなったときに拡張しやすい、使い勝手のよいフォルダー構成と言えるでしょう。

もちろん下図のように、プロジェクトのニーズやチームの意向に合わせて自由にフォルダー構造を展開してもかまいません。



プロジェクトのニーズに合わせてフォルダーを構成してもよいが、構造内での一貫性は保つこと。

フォルダー構造と命名規則

プロジェクトを整理する方法は 1 つではありませんが、一般に以下のようなベストプラクティスに沿って整理することをお勧めします。

- **命名規則とフォルダー構造をドキュメント化する**。スタイルガイドやプロジェクトテンプレートがあると、ファイルを探したり整理したりしやすくなります。
- **選択した命名規則を遵守する**。選択したスタイルガイドやテンプレートから逸脱しないようにします。命名規則を変更する必要がある場合は、解析を実施して影響を受けるアセットを特定し、スクリプトを使用してそれらすべてのアセットの名前を一括で変更します。
- **ファイル名やフォルダー名にスペースは使用しない**。Unity のコマンドラインツールでは、パス名にスペースが含まれていると問題が発生します。
- **テスト領域やサンドボックスには、分離した環境を割り当てる**。本番用でないシーンを置いて実験するための独立したフォルダーを作成します。ユーザー名付きのサブフォルダーを作成して、チームメンバー別に作業領域を切り分けることができます。
- **ルートレベルにフォルダーを追加しない**。一般に、コンテンツファイルは「Assets」フォルダーに保存します。どうしても必要な場合以外は、プロジェクトのルートレベルに追加のフォルダーを作成しないでください。

シーン

シーンは、Unity のコンテンツを扱う場所です。シーンにはゲームのオブジェクトが含まれ、メインメニューや個別のレベルなどの作成も行うことができます。シーンごとに環境、障害物、装飾を配置することで、ゲームの 1 レベルを大まかに構成します。これにより、アプリケーションを部分ごとに設計して作成できるため、モジュール性が保たれます。

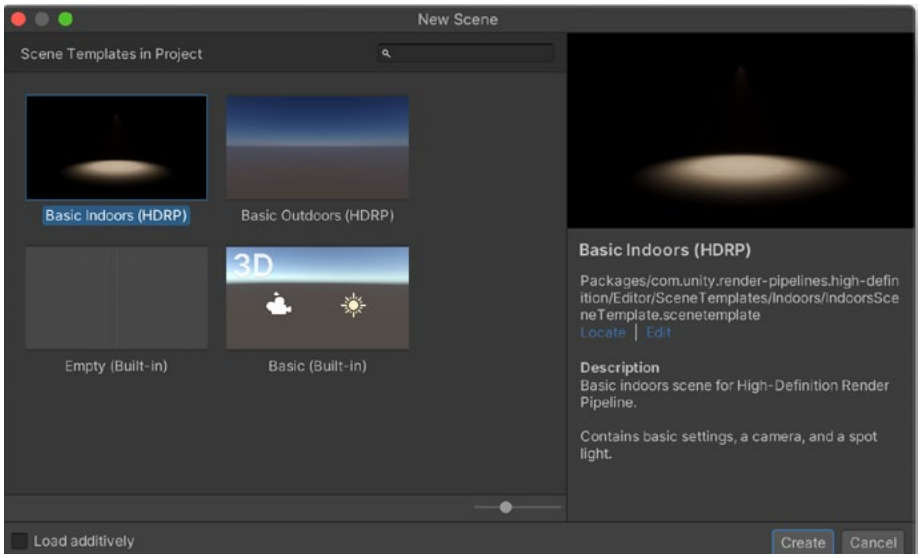
シーンファイル自体は、ディスクに保存されるアセットです。「Asset Serialization」のモードを「Force Text」に設定すると、シーンがテキストファイルとして保持されます。これ以外のモードであれば、デフォルトでバイナリとして保存されます。

シーンは多くの場合、ゲームの 1 レベル、または 1 レベルの一部を表します。デモやシンプルなゲームではシーンが 1 つしかないこともありますが、ほとんどの商業ゲームでは 1 レベルごとに 1 つのシーンが使用され、それぞれに固有の環境、キャラクター、UI などが与えられます。

1 つのプロジェクトでいくつシーンを作成してもかまいませんが、シーンの構成がパフォーマンスに大きな影響を及ぼすことがあります。シーンの構成とパフォーマンスに関する詳しい情報については、Unity が作成したモバイルパフォーマンス最適化についてのガイドを参照してください。

ゲームのさまざまな部分を表現してアプリケーションを形にするには、シーンを作成し、ロードし、保存する必要があります。典型的な「シーンフロー」では、イベントを契機に別のシーンのロードが行われます。たとえば、メニューシーンでユーザーがインターフェースをクリックしたときに、メインのゲームプレイシーンがロードされます。なお、シーンを連続的にロードすることもできます。つまり、シーンの要素を分割して 1 つずつ追加する形でロードすることが可能です。

新しいシーンを作成するときに、シーンテンプレートを選んで使用することができます。たとえば、HDRP の「3D Sample Scene」には複数のテンプレートが付属しています。独自のシーンテンプレートを定義すると、ワークフローの効率が向上し、チーム全員が同じアセットと設定でシーンの作成を始められるようになります。



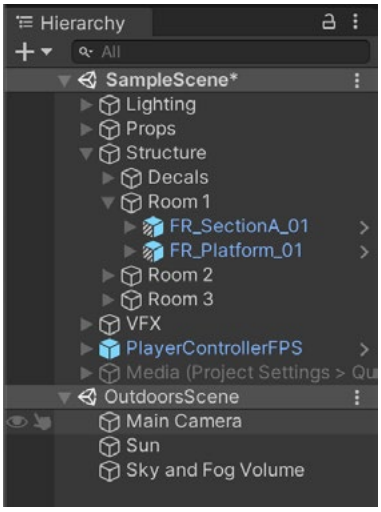
HDRP のシーンテンプレート

Unity には、スクリプトでシーンのロードや管理を行う場合に使用する SceneManagement API があります。詳細については、シーンフローに関する Learn チュートリアルを参照してください。

「Hierarchy」ウィンドウ

「Hierarchy」ウィンドウには、現在ロードされているシーンのアセットに含まれるゲームオブジェクトがすべて表示されます。具体的には、モデル、カメラ、プレハブなどです。ゲームオブジェクトをドラッグするだけで、親子関係を変更できます。

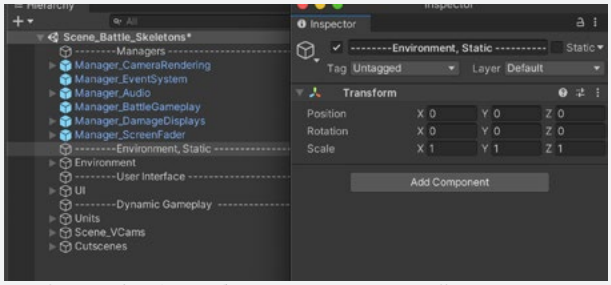
「Scene」ビューのオブジェクトを追加または削除すると、「Hierarchy」ウィンドウでもそれらのオブジェクトが追加または削除されます（逆も同様です）。「Hierarchy」ウィンドウでは、ランタイムにロードされる複数のシーンを表示できます。各シーンに、それぞれのゲームオブジェクトが含まれます。



HDRP テンプレートが適用された SampleScene に複数のアセットフォルダーが含まれている。

シーンとヒエラルキーに関する一般的なヒント

— **空のゲームオブジェクトに名前を付けてスペーサーとして使用する。**オブジェクトが見つけやすくなるよう、シーンは入念に整理します。あらゆるゲームオブジェクトに言えることですが、オブジェクトは最小限に抑えてください。整理上の必要性と効果のバランスをとることが重要です。不要な親子関係を作って整理することは控えてください（下記の**「適切な親子関係を使用する」**を参照）。



キャプション：空のゲームオブジェクトはスペーサーとして使用できる。

— **メンテナンスのプレハブと空のゲームオブジェクトをワールド原点に配置する。**Transform を指定せずにオブジェクトを配置する場合、(0,0,0) の位置に置きます。こうすることで、コードが簡潔になり、ローカル空間からワールド空間への変換、またはその逆方向の変換に伴う問題が減少します。

— **ワールドのフロアは y = 0 に置く。**これにより、オブジェクトをフロアに配置しやすくなります。ゲームロジック、AI、物理演算では、ワールドを xz 平面に沿った 2D 空間として扱います。

— **動的と静的のオブジェクトを分ける。**動くオブジェクトをランタイムで生成する場合は、それらを空のプレースホルダーオブジェクトの下に整理することを検討します。同様に、動かないレベルジオメトリはそのヒエラルキー内の別の位置に格納します。この方法は、適切なライティングテクニックをジオメトリに適用する際に役立ちます（たとえば、ライトマッピングとプローブライティングを分けて格納します）。

— **適切な親子関係を使用する。**互いに関係のあるオブジェクトは機能別にグループ化します。感覚的に理解しやすいヒエラルキーを作成します（たとえば、車のボディが親、タイヤが子となるような親子関係を作成します）。ヒエラルキーは少ない方がパフォーマンスを高められるため、なるべく不要な親子関係を作成しないようにします。シーンのヒエラルキーについては、これらのガイドラインに従ってください。

命名規則

ゲームオブジェクトの命名規則は特にありませんが、実際のプロジェクトでは以下の基準や手法を検討することがお勧めです。

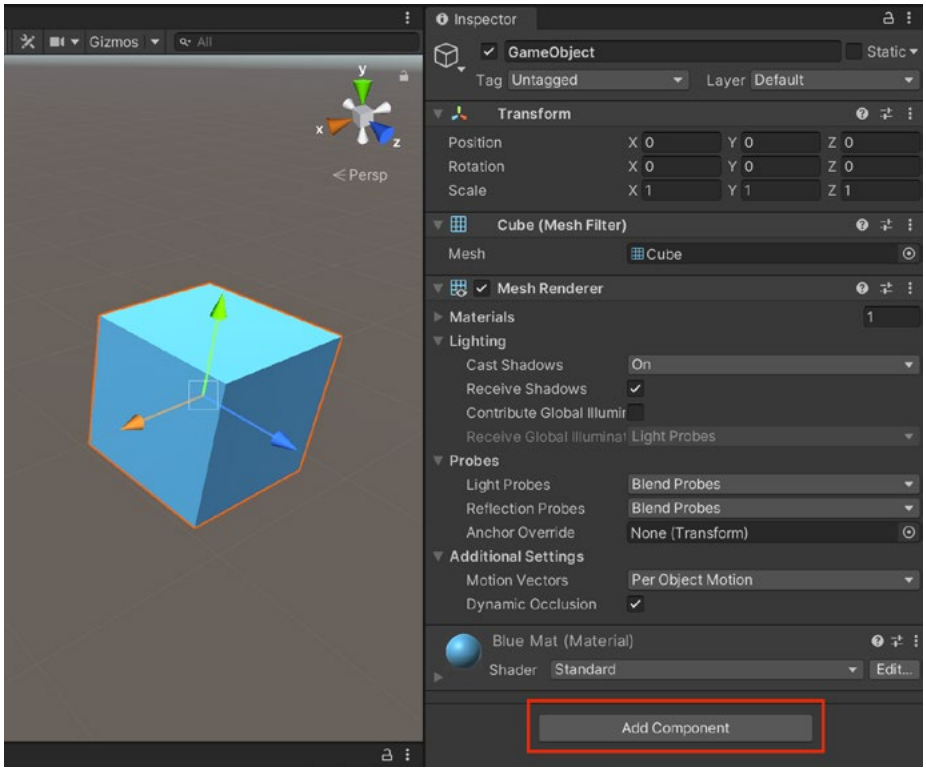
基準	例
わかりやすい名前を使用する。省略語を使用しない。数か月間は忘れることのない名前にします。他の人にも意図が伝わり、発音可能で、思い出しやすいものを選びましょう。混乱のもとになるので、省略語は使わないように注意します。	**largeButton、LargeButton、leftButton** 良くない例： lButton
キャメルケースまたはパスカルケースを使用する。オブジェクト名にスペースを使用することは避けます。キャメルケースまたはパスカルケースを使用することで、読み取りやすくなります（調査によれば、タイピングの正確さも高まります）。	**OutOfMemoryException、dateTimeFormat** 良くない例： Outofmemoryexception、datetimeformat
アンダースコアやハイフンの使用は控える。アンダースコアやハイフンの使用は基本的に控えましょう。ただし、特定の状況下では便利なこともあります。冒頭に名前を配置し、アンダースコアでつなげることで、アルファベット順に並べやすくなります。オブジェクトのバリエーションを示すのにも使用できます。	有効化状態： **EnterButton_Active、EnterButton_Inactive** マップ： **Foliage_Diffuse、Foliage_Normalmap** 詳細レベル： **Building_LOD1、Building_LOD0**
順序を示すには、番号でサフィックスを付ける。一連のシリーズになっていないものには番号は付けないようにします。	パスの場合は以下のようなノード名にする： **Node0**、**Node1**、**Node2**、etc.
デザインドキュメントの名前に準拠する。	デザインドキュメント内で場所の名前が **HighSpellTower** や **RedDragonLair** というものだった場合は、これらのスペリングを正確に使用する。

これらの命名基準はチームに合わせて適宜検討し、採用する場合は*一貫して*その基準を遵守してください。

ゲームオブジェクトとコンポーネント

Unity のゲームプレイとインタラクティビティは、**ゲームオブジェクト**と**コンポーネント**で構築されます。この 2 つが Unity プロジェクトの基本的な構成要素であり、インターフェースを介して、またはスクリプトを使用して作成します。

ゲームヒエラルキーの構成要素はすべてゲームオブジェクトです。ゲームオブジェクトは基本的に空のコンテナであり、それ自体に機能はありません。キャラクターから環境やパーティクルエフェクトまで、ゲーム内のさまざまな物事を表現することができます。



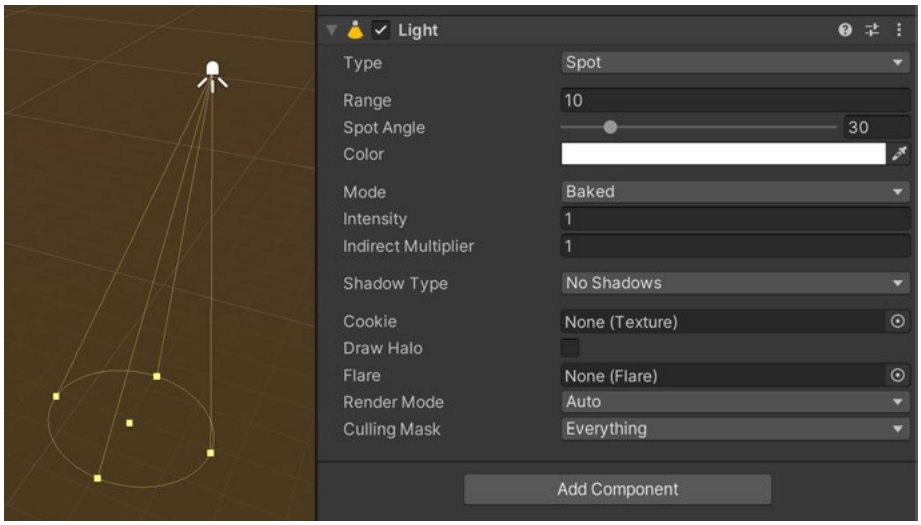
コンポーネントを追加すると機能も変化する。

ゲームオブジェクトにゲーム内の機能を持たせるには、前もって対象のゲームオブジェクトに特別なコンポーネントを追加する必要があります。1 つのゲームオブジェクトに複数のコンポーネントを格納することができます。多くの場合、同じゲームオブジェクト内のコンポーネントが相互に協調したり、他のゲームオブジェクト内のコンポーネントと連携したりして機能します。

作成する機能に応じて、さまざまなコンポーネントを組み合わせてゲームオブジェクトに追加します。スクリプトを使用して独自のコンポーネントを作成することもできます。

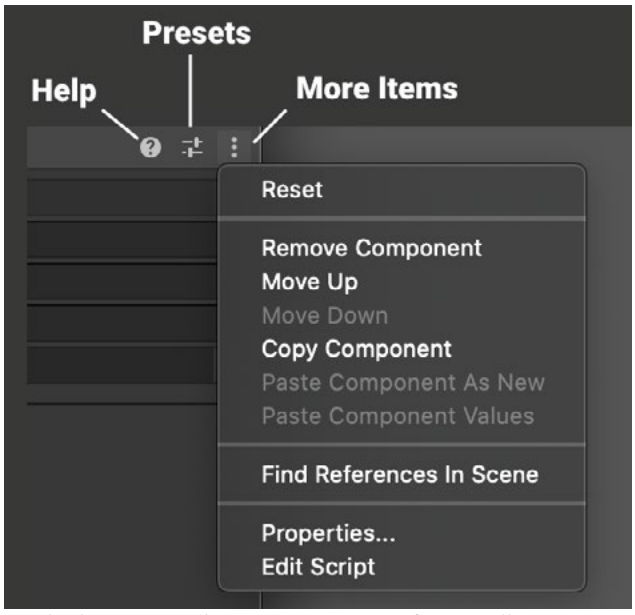
インスペクター

コンポーネントには編集可能なプロパティ（変数）がいくつかあり、「Inspector」ウィンドウで変更することができます。カスタムスクリプトを使用して、これらの変数を変更することもできます。



ライトの範囲、色、輝度は「Inspector」ウィンドウで変更できる。

インスペクターで値の入力やリセットを行うことができます。「Help」アイコンからはドキュメントをすばやく表示でき、プリセット機能で手早く値を保存したりロードしたりすることができます。**「More Items」**（⋮）メニューを使用して、コンポーネントを並べ替えたり、値をコピーして貼り付けたりすることができます。



ヘルプ、プリセット、その他のアイテムメニューでコンポーネントを管理

Transform コンポーネント

Unity のすべてのゲームオブジェクトに、位置、回転、スケールを保持する Transform コンポーネントが格納されています。このコンポーネントを削除することはできません。子であるゲームオブジェクトのインスペクターに表示される Transform の値は、親の Transform を基準としたローカル座標で表示されます。

Transform のスケールによって、モデリングアプリケーションのメッシュサイズを基準として、Unity のメッシュサイズが決定されます。これは、ワールド空間の 1 単位を 1m と想定する物理演算エンジンにとって重要なことです。

多くの 3D アプリケーション同様、Unity は内部的に回転をクォータニオンとして表現しますが、インスペクターでは編集しやすいように、それと同じ角度を表すオイラー角が表示されます。

トランスフォームを使用するためのヒント

- **子を追加する前にトランスフォームの値を消去する。**トランスフォームを子として追加する際は、親のトランスフォームの位置や回転を「0,0,0」に設定します。既存の値は Unity によって補完されるので、親子関係を変更した後も子オブジェクトは同じワールド空間上の位置に維持されます。

  親がトランスフォームされなかった場合、子のローカル座標はグローバル座標と同一になります。これにより、設定が簡単になります。

- **Tool モードや Pivot モードを正しく使用する。**移動、回転、スケーリングを行う際は、ツールバーのツール設定に注意してください。Unity には Pivot モードと Center モードの切り替え機能があります。これを使用すると、ギズモが Pivot モードと Center モードに切り替わります。同様に、ローカル座標系とグローバル座標系も切り替えることができます。

ツール設定

- **オブジェクトを床に座らせるか立たせるかする必要がある場合、そのピボットは中央ではなく底面に配置する。**これにより、キャラクターやオブジェクトをフロアに正確に配置しやすくなります。3D プロジェクトでは、xz 平面での見下ろし視点を使用した作業が簡単になります。

- **すべてのメッシュが同じ方向を向くようにする。**特定の方向を向くことがあるキャラクターやオブジェクトなどのメッシュの場合、一貫性に注意しておくとランタイム時の計算がよりシンプルになります。

— **最初からスケールの正確性に注意しておく。**アートアセットは、インポート時にスケールファクターが 1、x、y、z スケール軸が 1 の状態になるように作成しましょう。

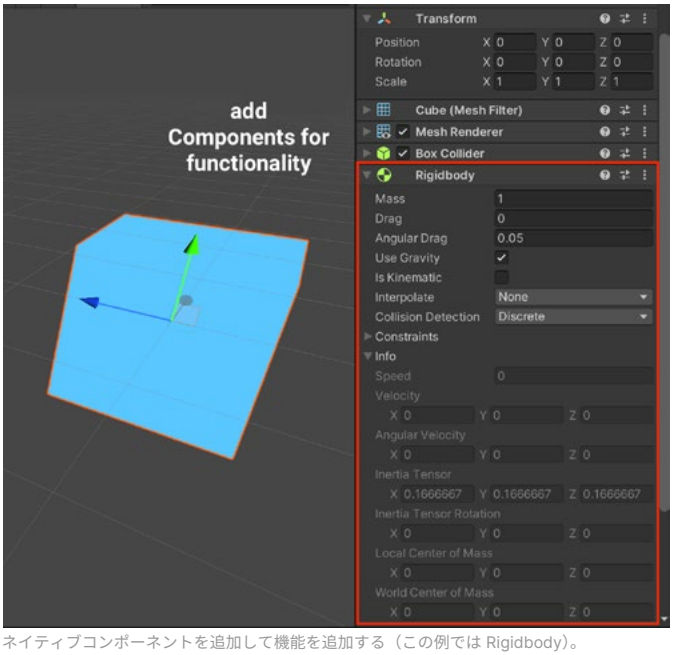
参照オブジェクト（**「GameObject」>「3D Object」>「Cube」**）ですばやくスケールを比較できます。スケーリングが必要な場合は、ルートトランスフォームでスケールが不均一にならないようにしてください。そうしないと、子トランスフォームを追加したときに予期しない結果が生じることがあります。

— **物理演算シミュレーションで Rigidbody コンポーネントを使用する場合は、1 メートルを 1 単位としてモデリングします。**そうしない場合は、代わりにモデルのインポート設定でメッシュスケールファクターを使用してください。速度は相対的に認識されるため、大きなオブジェクトはゆっくりと落下するように見えます（例：ワールド単位で同じ速度で移動するネズミとゾウがいた場合、人間の目は体のサイズと比較して速度を認識するため、異なる速度で移動しているように見えます）。

しかし、速度が違うように見えても、物理演算シミュレーション自体は正しく行われています。Rigidbody コンポーネントの参照ページで、メッシュスケールが物理演算に与える影響をご確認ください。

ビルトインコンポーネント

「Components」メニューで、選択したゲームオブジェクトにコンポーネントを追加することができます。たとえば、Rigidbody を 3D キューブに追加する場合（**「Add Component」>「Physics」>「Rigidbody」**）は、インスペクターに Rigidbody のプロパティが表示されます。



ネイティブコンポーネントを追加して機能を追加する（この例では Rigidbody）。

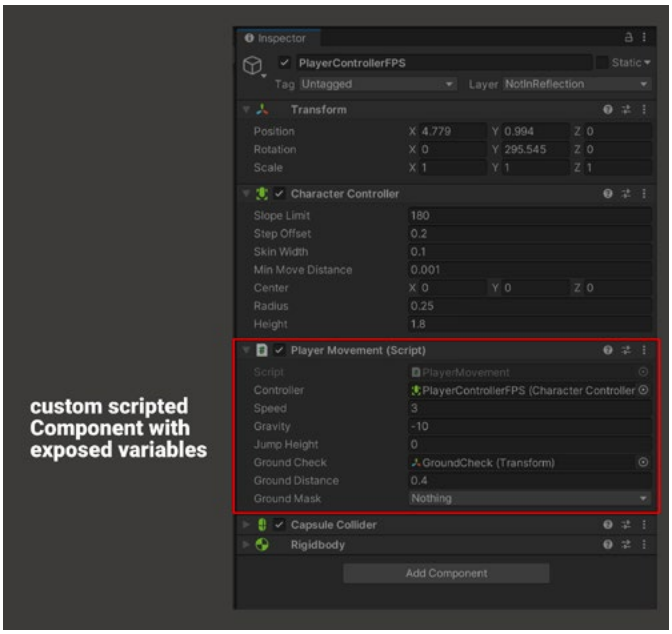
空のゲームオブジェクトが選択されている状態で**「Play」**をクリックすると、Unity の物理演算エンジンによってゲームオブジェクトに重力が作用して落下します。つまり、空のゲームオブジェクトに Rigidbody によって機能が追加されたのです。

コンポーネントを組み合わせることで、独自のゲーム体験を作り出すことができます。ビルトインコンポーネントを利用することで、サウンド、レンダリング、AI ナビゲーションなどを追加できます。エディターを使いこなすにつれ、使える機能はさらに増えるでしょう。

カスタムコンポーネント

ビルトインコンポーネントでゲームオブジェクトにさまざまな機能を追加できるとはいえ、どうしても独自の動作を追加しなければならないこともあります。そんなときに使用するのが Unity スクリプティング API です。カスタムスクリプトコンポーネントを使うと、ゲームイベントのトリガー、衝突のチェック、物理演算の適用、ユーザー入力への反応など、非常に多くの事柄を実行できます。

スクリプトを作成してゲームオブジェクトにアタッチすると、ビルトインコンポーネントと同様に、そのゲームオブジェクトのインスペクターにスクリプトが表示されます。その値はゲームデザインをテストしながらエディター上で調整できます。プロジェクト内で保存してコンパイルすると、そのスクリプトはコンポーネントとなります。



カスタムスクリプトコンポーネントの例

ゲームオブジェクトにアタッチされたスクリプトは、MonoBehaviour という名前のビルトインクラスから継承されます。**MonoBehaviours** の相互操作の作成は、Unity でのゲームプレイの作成においては非常に重要です。

Unity スクリプティング API では、UnityEngine と UnityEditor で利用可能なすべてのクラスが詳細に網羅されています。まずはマニュアルをよくお読みいただいたあと、必要に応じて詳しい情報をご確認ください。

プレハブ

Unity のプレハブシステムを使用すると、ゲームオブジェクトを再利用可能なアセットとして作成、設定、保管することができます。コンポーネント、プロパティ値、子のゲームオブジェクトのすべてが、そのままプレハブとして保持されます。プロジェクトのプレハブアセットは、シーンに新しいプレハブインスタンスを作成するためのテンプレートとして利用できます。

特定の設定が施されたゲームオブジェクトを再利用するには、そのゲームオブジェクトをプレハブに変換します。

プレハブの使用

ヒエラルキー内で多数のゲームオブジェクトのプレハブインスタンスを使用することができます。プレハブを使用して、複製が必要なものを何でも複製できます。キャラクター（プレイヤーや NPC）でも、小道具でも、レベルジオメトリの一部でもプレハブ化できます。一般的なプレハブの使用例には、以下のようなものがあります。

- **環境アセット：**特定のレベルで同じ樹木や建物を何度も再利用する場合は、それらのメッシュをプレハブに変換します。
- **ノンプレイヤーキャラクター（NPC）：**同じタイプのキャラクターが、ゲームの複数のレベルで何度も登場することがあります。その場合は、オーバーライドを使用し、動作、外観、音声を変えて個体差をつけます。
- **弾丸やパワーアップアイテム：**元はシーンに存在していなかったゲームオブジェクトをランタイムでインスタンス化する場合は、プレハブを使用します。たとえば、レーザーガンを発射するたびにパーティクルエフェクトをインスタンス化します。
- **プレイヤーのメインキャラクター：**プレイヤーのプレハブをゲームの各レベルの開始地点に配置できます（ロードされた個別のシーンに追加することも可能）。

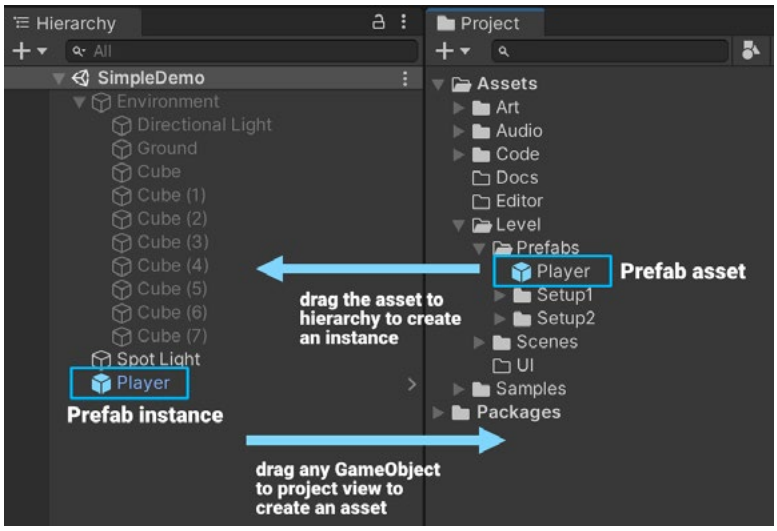
変更を加えると自動的にプレハブが同期されるため、アセットを変更した後にそれをシーンに反映させるプロセスが簡略化されます。

プレハブアセットとインスタンスの作成

プレハブアセットを作成するには、「Hierarchy」ウィンドウから「Project」ウィンドウにゲームオブジェクトをドラッグします。そのゲームオブジェクト（コンポーネントや子ゲームオブジェクトを含む）が新しいアセットとして「Project」ウィンドウに表示されます。

プレハブアセットの目印は、青いキューブのプレハブアイコン、またはゲームオブジェクトのサムネイルです。どちらになるかは、「Project」ウィンドウのレイアウトが 1 列か 2 列かによって決まります。

先の手順と同様に、「Project」ウィンドウのプレハブアセットを「Hierarchy」ウィンドウまたは「Scene」ビューにドラッグすると、プレハブインスタンスが作成されます。



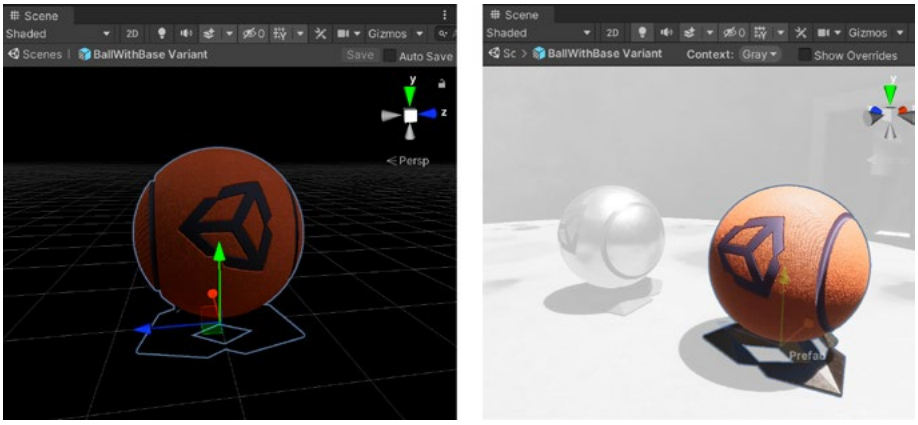
プレハブアセットとインスタンスの作成

ディスク上のプレハブアセットに加えられた変更は、シーン内のインスタンスに自動的に反映されます。アセット内のすべてのインスタンスで同じ編集を繰り返す必要はありません。

ただし、すべてのプレハブインスタンスを同一に保つ必要もありません。プレハブインスタンスの設定を個別にオーバーライドして、他のインスタンスや元のアセットとは異なる設定を持たせることができます。

プレハブモード

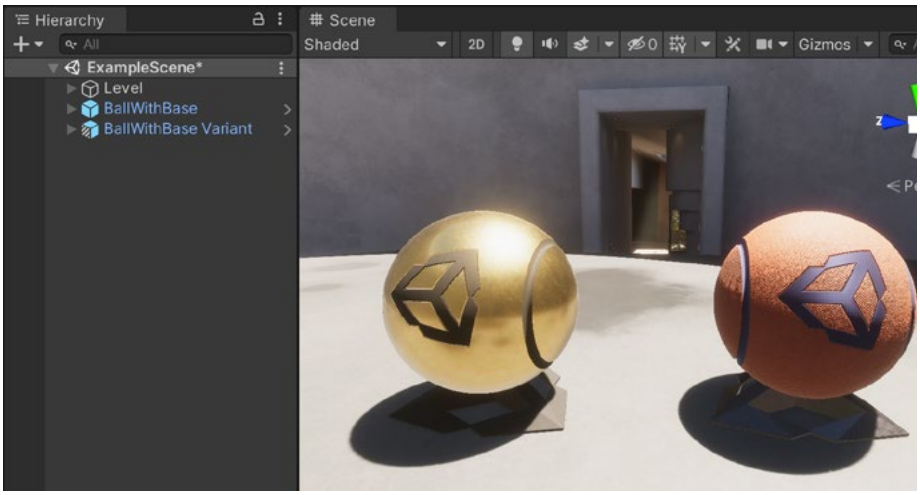
プレハブアセットを編集する際に、専用の**プレハブモード**で開くことができます。プレハブアセットのコンテンツを個別表示して単独で編集するモードや、シーンにある他のゲームオブジェクトとの関係がわかるように表示して編集できるモードがあります。プレハブモードで加えた変更は、そのプレハブの全インスタンスに反映されます。



Prefab モード（左は分離状態、右はコンテキスト内）

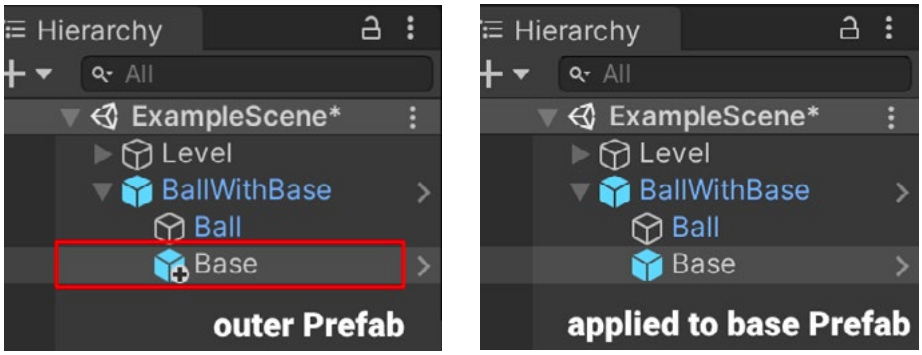
プレハブバリアントと入れ子構造のプレハブ

プレハブのバリアントを作成して、必要な変更を加えたプレハブをデザインすることができます。原型となるアセットから特徴を継承して、変更したい部分だけをオーバーライドすると考えてください。



同じベースプレハブのバリアント

プレハブ内で別のプレハブを入れ子にして、複雑なオブジェクトヒエラルキーを作成すると、複数レベルでの編集を行いやすくなります。入れ子構造のプレハブインスタンスには、ヒエラルキー内でアイコンにプラスの印が付きます。この印は、外側のプレハブのインスタンスをオーバーライドしていることを示します。



入れ子構造のプレハブ。「外側」のプレハブにはプラスバッジが表示される

ソースのプレハブアセットを単独で編集すると、入れ子構造のプレハブに変更が反映されます。

パッケージマネージャー

Unity がオプションのアドオンパッケージとして提供している機能を利用すると、さらに柔軟な開発が可能になります。これはコアエディターの外部で機能するため、より迅速なリリースサイクルにも対応することができます。パッケージ管理システムを活用すると、ユーザーは特定のプロジェクト要件に必要なツールだけをインストールできるようになるため、ビルドのサイズを抑えることができます。パッケージ自体の機能はさまざまです。プロジェクトの各部を強化するためのアセット、シェーダー、テクスチャ、プラグイン、アイコン、スクリプトを組み合わせて格納できます。

「Package Manager」ウィンドウを使用して、利用可能なパッケージを管理します（**「Window」** > **「Package Manager」**）。

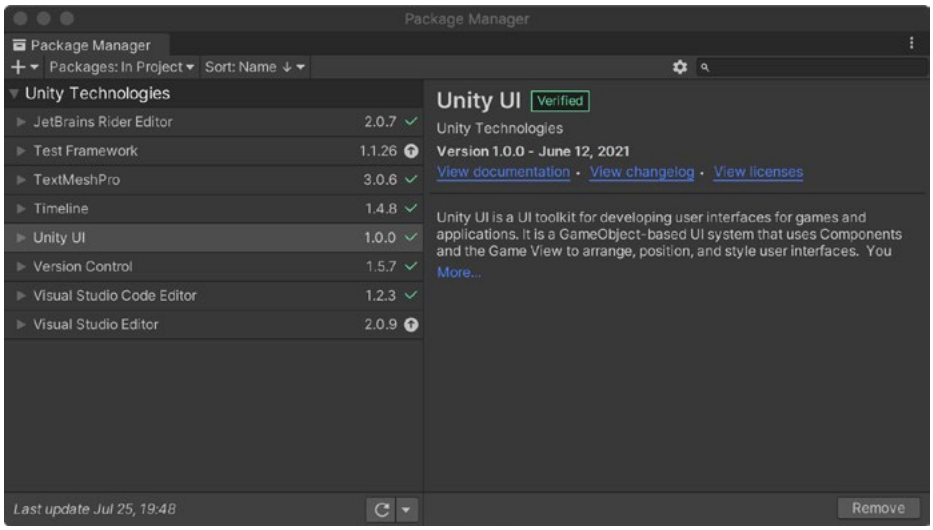
ビルトインの、または Unity Registry に含まれている Unity 製パッケージに加えて、ディスク、GitHub URL、Asset Store からカスタムパッケージをインストールすることもできます。

— Git URL またはローカルパスから直接パッケージをインストールするには、 + ボタンをクリックします。

— 「Packages」ドロップダウンメニューで、リストの表示内容を変更します（「Unity Registry」、「In project」、「My Assets」、「Built-in」）。一方で「Sort」メニューを使用すると、名前または日付でパッケージを並べ替えることができます。

— **「My Assets」**の追加フィルタリングオプション機能を使用して、Asset Store パッケージを絞り込むことができます。

— **「Search」**フィールドを使って、名前でパッケージを検索できます。プロジェクトで使用しているパッケージの管理には、「Download」、「Import」、「Update」の各ボタンが使用できます。



パッケージマネージャー

経験豊富な開発者は、PackageManager API を活用して独自の共有ライブラリを作成することもできます。それについては「Creating Custom Packages in Unity」の動画で簡潔に紹介されています。

プログラミング

Unity ではコードを使用することで、ゲームの各部を直接カスタマイズまたは制御したり、チームで使用する視覚的なツールを作成したりできるだけでなく、Unity 自体の動作を変更することもできます。Unity で使用される C# は、ソフトウェア業界で広く採用されている現代的なオブジェクト志向の言語です。

カスタムコンポーネントと MonoBehaviour

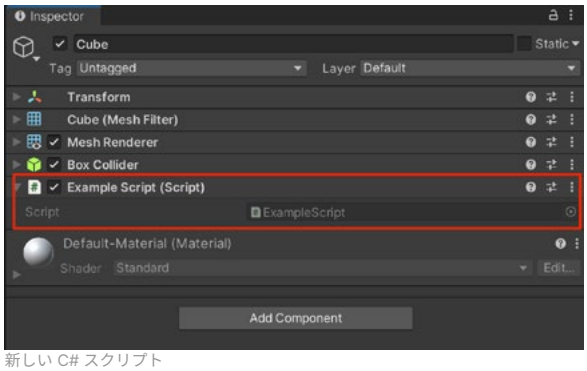
どの種類のゲームオブジェクトにも一連のデフォルトコンポーネントが付属しています。たとえば空のゲームオブジェクトにも、最初から Transform コンポーネントが含まれています。こうしたデフォルト設定をカスタムコンポーネントで拡張し、作成したゲームロジックと、チームがゲームシーンに使用しているオブジェクトを直接結び付けることができます。また、デザイナーがコンポーネントの設定を利用して値や動作を微調整できるようにすることも可能です。

こうしたことを行うには、スクリプトを作成し、コンポーネントとしてゲームオブジェクトに追加します。それぞれのスクリプトは、MonoBehaviour というビルトインクラスから派生します。

MonoBehaviour クラスは、新しいコンポーネントタイプを作成するための青写真のようなものと考えてください。スクリプトコンポーネントをゲームオブジェクトにアタッチするたびに、この青写真に従って、そのコンポーネントの新しいインスタンスが定義されます。

スクリプトは Unity 内で直接作成されます。**「Assets」**メニューを使用（または**右クリック**）して、**「Create」>「C# Script」**の順に選択すると、新しい C# スクリプトがディスク上に生成されます。使用するクラス名に合わせてファイル名を付けます。ヒエラルキー内でこのファイルをオブジェクトにドラッグアンドドロップすると、インスペクターに ExampleScript がコンポーネントとして表示されます。

Unity の現在のスクリプトテンプレートは、最初に **Start** と **Update** という 2 つの関数を持つクラスが記述されています。



新しい C# スクリプト

ノート：ファイル内のクラス名を変更し、ファイル名は変更せずに、スクリプトをカスタムコンポーネントとしてアタッチした場合、そのスクリプトは正常に動作しなくなる可能性があります。Unity では、MonoBehaviour から継承される **ExampleScript** という名前のクラスが自動的に設定されます。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ExampleScript : MonoBehaviour
{
    // Start is called before the first frame update
   void Start()
   {
   }

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

  {
  }

}
```

Unity はゲームプレイを開始する前、そして Update 関数を初回呼び出しする前に、**Start** 関数を呼び出します。したがって、Start 関数は変数を設定し、環境設定を読み込み、他のゲームオブジェクトとのつながりを構築するための理想的な場所です。

Update 関数は、フレームごとに実行されるコードを処理します。たとえば、移動、アクションのトリガー、ユーザー入力への応答などのコードです。

Update と **Start** の 2 つ以外に、MonoBehaviour のイベント関数はありません。このようなビルトインメソッドは、決まった順番で実行されます。MonoBehaviour のイベント関数をオーバーライドする際に、どのようなゲームプレイを構成し、どのようなメインゲームループを作成するかを考えることになります。

Unity でのコーディングの基礎に慣れるには、こちらで Unity のドキュメントを参照してください。

オブジェクトの初期化

プログラミング経験が豊富な方にとっては、オブジェクトの初期化にコンストラクターが使用されないことが意外かもしれません。オブジェクトの構成はエディターによって管理されるので、ゲームプレイの開始時には実行されません。スクリプトコンポーネント用のコンストラクターを定義するのは Unity 通常の運用方法ではないため、問題が発生する可能性があります。

MonoBehaviour のライフサイクルと構造

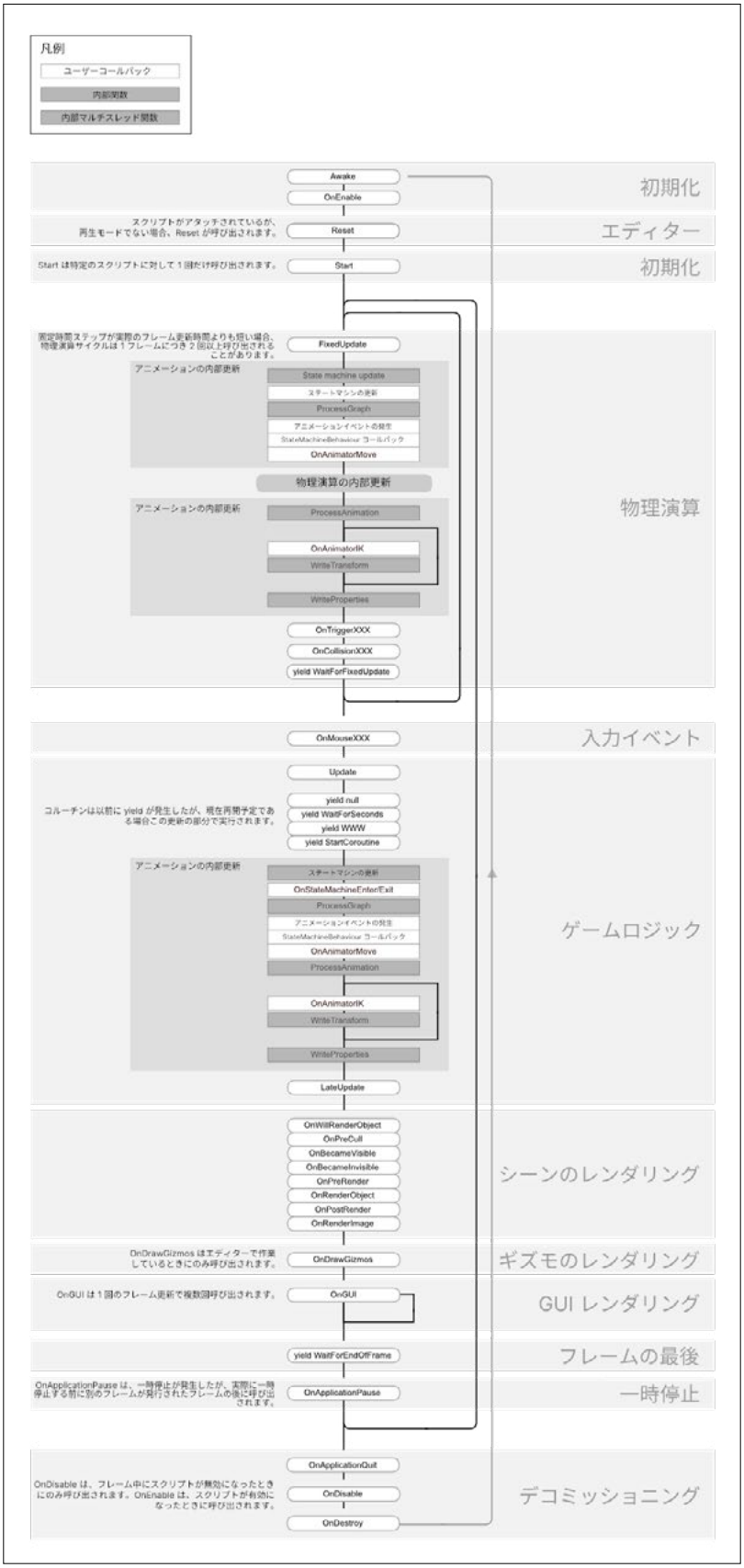
ゲームエンジンには、ユーザー入力の処理、ゲームステートの更新、画面へのレンダリングをエンドレスでループする機能が必要です。Unity では、この **PlayerLoop** がゲームエンジンの核となるローレベルのクラスです。このクラスは、初期化とフレームごとの更新を処理するいくつかのサブシステムを制御します。

PlayerLoop とやり取りできるよう、ユーザーのスクリプトは MonoBehaviour 基底クラスから派生します。そのため、このクラスの動作を学ぶことがゲームプレイを生み出すために重要となります。こちらのフローチャートは、MonoBehaviour のイベント関数がスクリプトのライフタイム全体にわたってどのように実行されるかを示しています。

MonoBehaviours を扱う際にゲームループに欠かすことのできない要素として、以下のようなものがあります。

- 最初のシーンのロード
- エディター
- 最初のフレームが更新される前
- フレームとフレームの間
- Update の更新順
- アニメーションの更新ループ
- レンダリング
- コルーチン
- オブジェクトが破棄されたとき
- 終了時

経験豊富なユーザーであれば、独自の PlayerLoop と PlayerLoopSystems を作成することができます。しかし、まずは Monobehaviour クラスや下記の一般的なクラスをよく理解することをお勧めします。



MonoBehaviour のライフサイクル

スクリプティングのヒント

Unity のスクリプティングバックエンドは .NET Framework を基盤としています。Unity での開発では、C# の機能を活用したスクリプティングが可能です。

- **名前空間**：Unity で使用されるクラスには一意の名前が必要です。複数人のプログラマーが同じプロジェクトで作業する場合、「Controller」のような同じ名前があると競合のもとになります。これを避けるために名前空間を使用し、データ型を整理してください。必要に応じて using ディレクティブを適用することで、名前空間のプレフィックスを短縮できます。

  例えば、プレイヤー用と敵用に名前空間を作成するとします。その際、Player.Controller と Enemy.Controller という名前を付ければ、同じプロジェクト内でも共存させられます。

  詳細については、名前空間のマニュアルページを参照してください。

- **コルーチン**：アクションをトリガーして、複数のフレームにわたって実行したい場合があります。例えば、オブジェクトが A から B にゆっくりと移動する場面や、色をゆっくりとフェードアウトさせる場面を想像してください。通常、関数は同じフレーム内で完了し、結果を返してしまうので、ロジックを複数フレームにわたって実行させることは困難です。

  コルーチンは、実行を一時停止して Unity に制御を返した後、次のフレームで中断したところから続行できる関数のようなものです。特定の時間にわたってゲームロジックを適用させたいときに、コルーチンが役立ちます。例えば、指定した時間だけ実行を一時停止させる処理には主にコルーチンが使われます。別のコルーチンの実行を待機したり、条件が整うのを待つための while ループとコルーチンを組み合わせたりすることもできます。コルーチンは、MonoBehaviour の Update イベント関数のように動作しますが、それだけでなく、更新の間隔も調整できるのです。

  詳細については、コルーチンのマニュアルページを参照してください。

- **属性**：クラス、プロパティ、関数の上位に配置して特定の動作を示すことができるマーカーです。

  例えば、数値のフィールドをインスペクターのスライダーに変更するには、Range 属性を使用するか、フローティングツールチップをインスペクターのフィールドに追加します。C# では属性名を角括弧で括ります。

  属性の完全なリストは、「スクリプティング API」でご確認ください。

一般的なクラス

Unity でスクリプティングを開始したら、重要度の高いビルトインクラスを確認する必要があります。このリストは網羅的なものではありませんが、この機能を知るための出発点となるでしょう。クラスの完全なリストと詳しい情報については、スクリプティング API を参照してください。

クラス	説明
GameObject	シーンに存在できるオブジェクトの種類を表します
MonoBehaviour	すべての Unity スクリプトの派生元となるベースクラス
Object	Unity がエディター内で参照できるすべてのオブジェクトのベースクラス
Transform	スクリプトを使用して、さまざまな方法でゲームオブジェクトの位置、回転、スケールを操作したり、ゲームオブジェクトに階層式の親子関係を付与したりできます
Vectors	2D、3D、4D の点、線、向きの表現と操作ができるクラス
Quaternion	絶対位置または相対位置の表現、作成、操作ができるクラス
ScriptableObject	大量のデータを保存するのに使用できるデータコンテナ
Time	時間の計測と制御、プロジェクトのフレームレートの管理ができるクラス
Mathf	三角関数、対数関数などを含む一般的な演算ユーティリティのコレクション
Random	一般的に使用されるさまざまな種類のランダム値を簡単に生成できます
Debug	エディターで情報を可視化でき、プロジェクトの実行中に起こっていることの把握と調査に役立ちます
Gizmos と Handles	シーンビューやゲームビューの線や形状を作成したり、インタラクティブなハンドルやコントロールを作成したりできます

メモリ管理

Unity は業界標準の言語である C# をサポートしています。C# は Java や C++ と共通点のある言語です。また、C# はメモリの割り当て、割り当て解除、メモリリークの解消などのメモリ管理を自動で処理してくれる「マネージド言語」です。

C++ などの言語では、プログラマーが適宜、関数を呼び出して、ヒープメモリのブロックを割り当てたり割り当てを解除したりする必要があります。一方、C# は自動メモリ管理機能により、明示的なメモリ割り当てやその解除が必要な言語に比べて、コーディングの労力が少なくて済み、メモリリーク（メモリ割り当てが解除されていない状態）が発生する可能性も大幅に少なくなっています。

値型と参照型

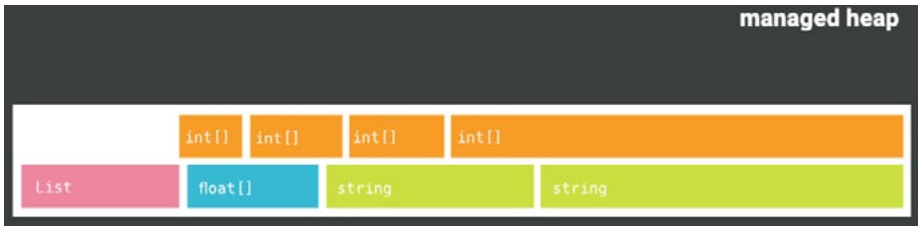
Unity では関数を呼び出すときに、そのためのメモリ領域を予約し、関数のパラメーターの値をコピーします。整数、浮動小数点数、ブール値などの*値型*は、数バイトしかメモリを占有しません。Unity は値型を直接格納し、パラメーター受け渡しの際にその値型をコピーします。

別のデータ型（オブジェクト、文字列、配列など）は*参照型*です。参照型のデータはより多くのメモリを占有し、定期的にコピーすると非効率になる場合があります。その代わりに、Unity ではこれらのデータをヒープメモリに格納し、ポインターを介してアクセスします。したがって、構造体（*値型*）のみが必要な場合は、クラス（*参照型*）を使用するよりも、構造体を使用したほうが同じデータを効率的に保持することができます。

明示的にメモリを割り当てたり、それを解除したりする必要はありませんが、マネージヒープメモリについて理解し、それがゲームアプリケーションのパフォーマンスにどのような影響を及ぼすのかを知っておくことは*必要*です。

ガベージコレクション

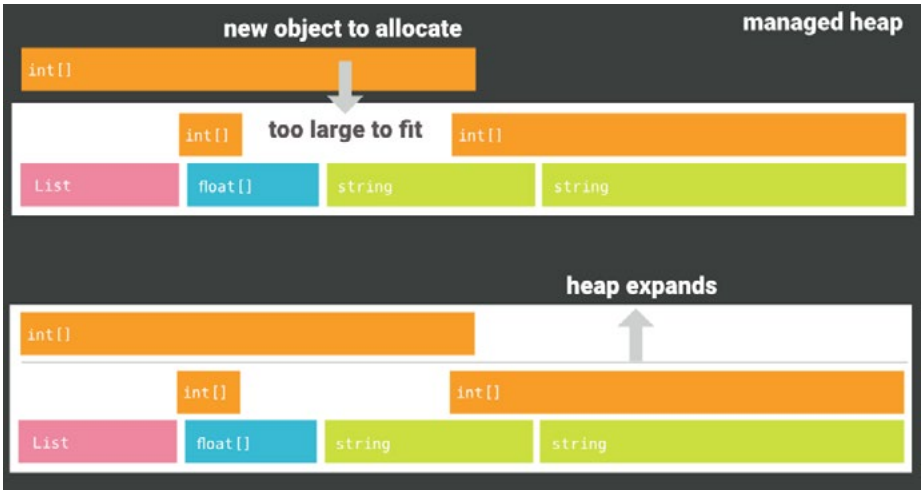
ヒープメモリのブロックは、使用されていて有効な参照を格納していれば「ライブ」です。



マネージメモリアロケーターが自動的にヒープメモリを割り当てる。

ゲームの実行中に、メモリブロックへの参照が失われることがあります（ゲームオブジェクトが破棄された、変数が再割り当てされた、など）。メモリブロックへの参照がすべて失われると、**マネージメモリアロケーター**によって安全にメモリを再利用できます。

アロケーターは定期的にライブのメモリブロックの間にある空き領域を探索します。使われていないメモリ領域を見つけて解放する処理をガベージコレクションと呼び、「GC」という略称が使われることもあります。ゲームアプリケーションが新しいメモリブロックをリクエストすると、アロケーターがこれらの未使用ブロックから割り当てるメモリを調達します。



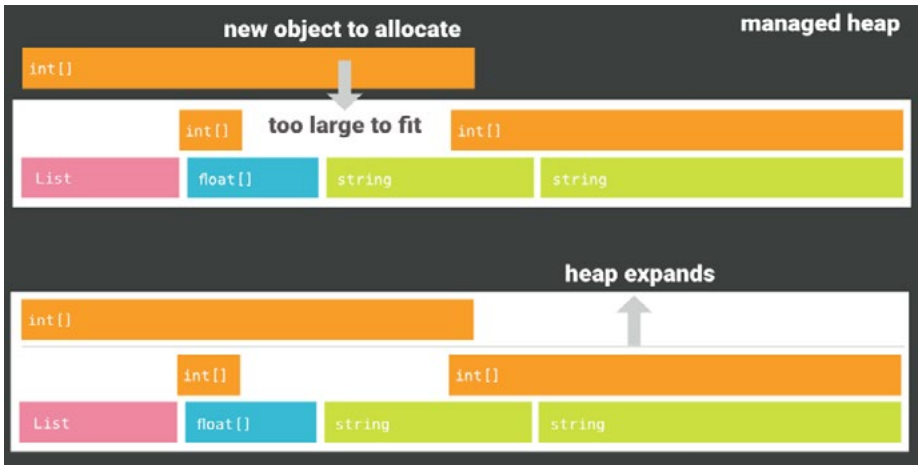
ガベージコレクションにより、スクリプトコードの実行が一時停止され、使用していないメモリが開放される

Unity は Boehm–Demers–Weiser ガベージコレクターを実装しています。Unity は**ガベージコレクション**の処理中にユーザーのプログラムコードの実行を中断し、ガベージコレクターが処理を完了すると単純に通常の実行を再開します。

この中断が原因となって、ユーザーのアプリケーションの実行に遅延が発生することがあります。遅延の程度は、ガベージコレクターが処理を行う必要のあるメモリの量と、ゲームのターゲットプラットフォームの種類によって異なります。1 ミリ秒に満たないこともあれば、数百ミリ秒に及ぶこともあります。

ゲームのようなリアルタイムアプリケーションにとって、この遅延は大問題になりかねません。ガベージコレクションで生じた中断は GC スパイクと呼ばれ、ゲームプレイのスタッターを引き起こすことがあります。ガベージコレクションはほとんどの場面で目につかないとはいえ、コレクションのプロセスはシーンの背後でかなりの CPU 時間を必要としています。

また、Boehm GC アルゴリズムではコンパクションが行われないことにも注意してください。メモリ内で既存のオブジェクトを移動させてオブジェクト間のギャップを埋める操作が行われないため、メモリの断片化が起きることがあります。割り当てようとする新しいオブジェクトが既存のギャップに収まらない場合、アロケーターがヒープのサイズを拡張して合わせる必要性が生じることがあります。ヒープ拡張は、パフォーマンスに影響を及ぼす可能性があります。



ヒープは拡大することがあるので注意

Unity では、必要以上の頻度でガベージコレクターをトリガーしないようにしてください。トリガーしすぎると、ランタイム中にアプリケーションのフリーズやスタッターを引き起こすことがあります。こちらのブログ記事で最適化のヒントとコツを確認し、ガベージコレクションの影響を抑えるために活用してください。

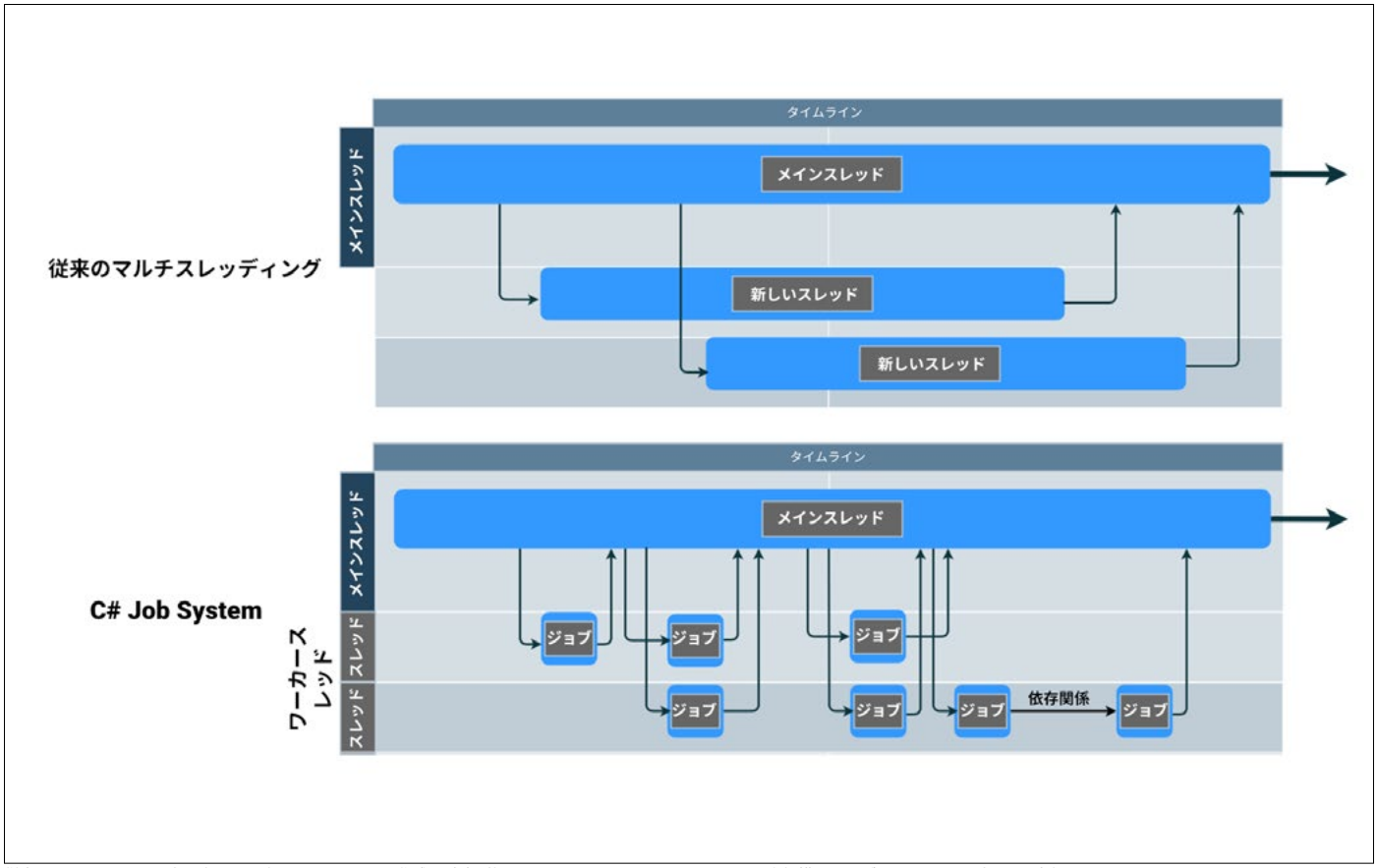
Unity は、複数のフレームに GC を分割する **Incremental Garbage Collector** もオプションで提供しています。この機能は現在実験段階であり、詳細についてはこちらのブログ記事をご覧ください。

メモリ管理とガベージコレクションの詳細については、自動メモリ管理についてとマネージヒープについての Unity ドキュメントを参照してください。さらに、Learn サイトの「Memory Management in Unity」ガイドもお読みください。

マルチスレッディング：C# Job System と Burst Compiler

最新の CPU には複数のコアが搭載されていますが、その性能を引き出すには、アプリケーションで**マルチスレッドコード**を活用する必要があります。Unity の Job System を使用すると、大きなタスクを小さなチャンクに分割し、余力のある CPU コアで並列実行できます。これにより、パフォーマンスを大幅に高めることができます。

マルチスレッドプログラミングでは、*メインスレッド*である 1 つの CPU 実行スレッドが、別のスレッドを作成してタスクを処理するという手法がよく使用されます。このような追加の*ワーカースレッド*は、自身の作業が完了したらメインスレッドに同期します。



従来のマルチスレッドプログラミングでは、スレッドの作成と破棄が繰り返される。C# Job System では、小規模なジョブはスレッドのプールで実行される

少数のタスクを長時間にわたって実行する場合は、この手法によるマルチスレッディングがうまく機能します。しかし、ゲームアプリケーションでは通常、1 秒あたり 30 ～ 60 フレームで短いタスクを数多く処理する必要があるため、この手法は効率的ではありません。

そのため、Unity では C# Job System という少し異なるマルチスレッディング手法を使用します。この手法では、短いライフタイムのスレッドを多数生成するのではなく、作業をジョブと呼ばれる小さな単位に分割します。

このジョブがキューに入れられ、ワーカースレッドの共有プールでスケジュールされた実行のタイミングまで待機します。JobHandle を使用して依存関係を作成することで、ジョブが正確な順番で実行されます。安全システムによって競合状態を回避できるよう、ジョブはデータのコピーに対して実行されます。その後、ネイティブコンテナがメインスレッドに結果を返します。

Job System を補完する手段として、Burst コンパイラーがあります。Burst は LLVM を使用して、IL/.NET バイトコードを最適化されたネイティブコードに変換します。Burst を使用するには、パッケージマネージャーで Burst パッケージを追加するだけです。Burst を導入すると、Unity 開発者は引き続き C# のサブセットを使用できるという利便性を享受しつつ、パフォーマンスを向上させることができます。

Unity のスクリプティングバックエンド

Unity のスクリプティングバックエンドは、Mono と IL2CPP（Intermediate Language To C++）の 2 つです。以下のように、互いに異なるコンパイル手法を使用します。

- **Mono** では実行時コンパイル（JIT コンパイル）を使用し、ランタイムにオンデマンドでコードをコンパイルします。
- **IL2CPP** では事前コンパイル（AOT コンパイル）を使用し、実行前にアプリケーション全体をコンパイルします。

IL2CPP は Unity が開発したスクリプティングバックエンドで、一部のプラットフォームのプロジェクトを作成するときに、Mono の代わりに使用できます。パフォーマンスを高めてビルドサイズを削減できますが、多くの場合、ビルド時間は長くなります。

IL2CPP を使ってプロジェクトを作成する場合、Unity は選択したプラットフォームのネイティブバイナリファイル（.exe、apk、.xap など）を作成する前に、スクリプトからの IL コードを変換し、C++ コードに組み立てます。

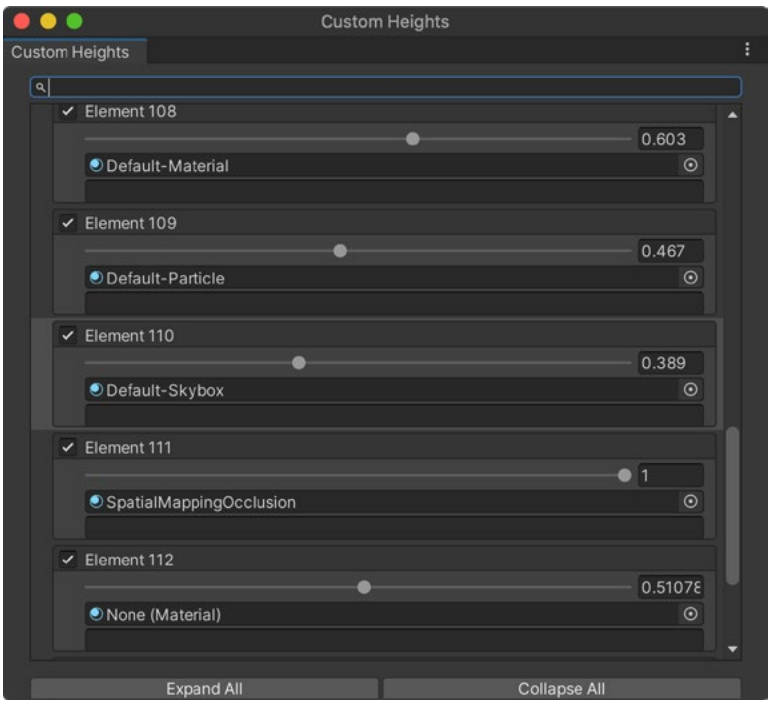
IL2CPP は、iOS と WebGL 用にビルドするときに唯一利用できるスクリプティングバックエンドであることに注意してください。

IL2CPP の使用方法について詳しくは、Unity IL2CPP のブログシリーズと、IL2CPP を使用したプロジェクトのビルドに関するページをご覧ください。

エディタースクリプト

作業効率を高めるため、チームやプロジェクトのニーズに合わせて開発環境をカスタマイズしたいというニーズにも対応します。

特殊なワークフローが必要な場合には、独自にカスタマイズしたインスペクターやウィンドウを使用してエディターを拡張できます。これはインスペクターやシーンなどのビルトインウィンドウと同様に操作できます。カスタムのプロパティドロワーを使用して、プロパティの表示方法を定義することもできます。



カスタムエディターウィンドウ

Odin Inspector および Serializer

Unity Asset Store で購入できるサードパーティの Unity 公認ソリューションパートナーツールである Odin Inspector および Serializer を使用すると、EditorWindow API の処理時間を短縮できます。Odin の提供する 100 以上のビルディングブロック属性を使用すると、手動でカスタム GUI コードを記述したり管理したりすることなくカスタムエディターを作成できます。

Odin を使用してカスタムエディターウィンドウを作成する際には、OdinEditor Window クラスを継承し、フィールド、プロパティ、メソッドに属性を入力するだけです。

Odin で定義できるプロセスの一例は次のとおりです。

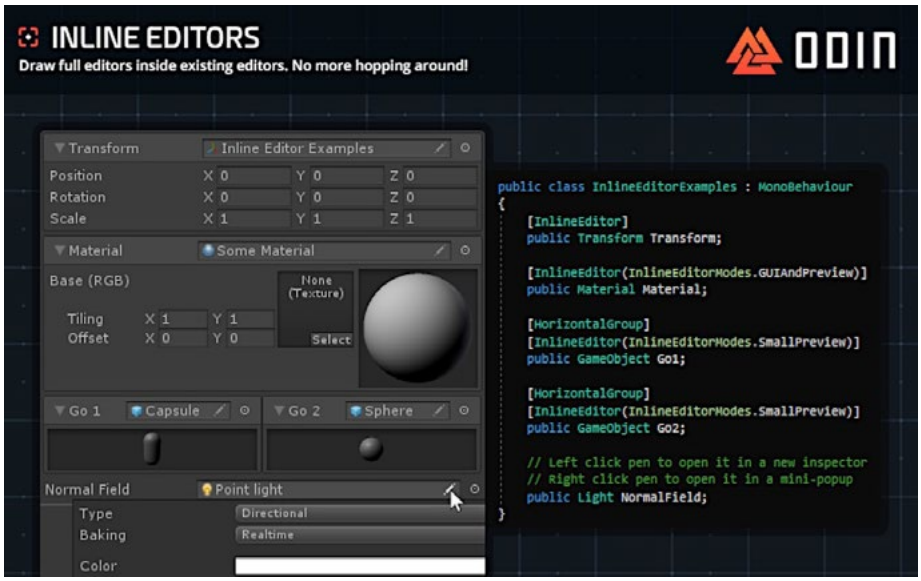
- TabGroup や ToggleGroup などのグループ属性でレイアウトをカスタマイズする
- ネイティブの Unity インスペクターでは通常は利用できない辞書などのフィールドをシリアライズする
- ボタン属性をメソッドに追加して「Inspector」ウィンドウに簡単にボタンを作成する
- テストやデバッグ目的で静的メンバーを修正する（引数を含む静的メソッドをインスペクターから直接呼び出すなど）

— 属性を使用して「Inspector」ウィンドウ用のカスタムエディターを作成する

— 属性式を使用して属性内で直接 C# コードのスニペットを記述し、定型コードを削減する

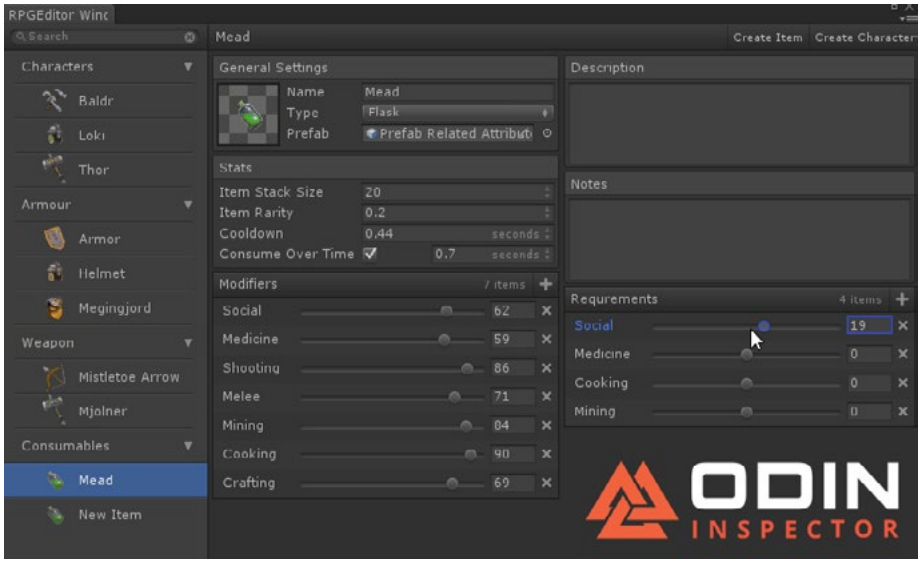
— Required、ValidateInput、ChildGameObjectsOnly などの属性でユーザー入力を検証する

たとえば、スクリプトを使用して次のようなインスペクターを作成できます。



Odin Inspector での作成例

Odin を使用して作成したエディターウィンドウの例は次のとおりです。



Odin で作成した RPG エディターウィンドウ

Odin Inspector は、Unity Asset Store の Personal および Enterprise エディションの両方で利用できます。

統合開発環境（IDE）のサポート

Unity は複数の IDE をサポートしており、お好みの開発環境で作業できます。Visual Studio は、Windows 版と macOS 版の Unity ではデフォルトでインストールされます。「Preferences」（**「Unity」>「Preferences」>「External Tools」>「External Script Editor」**）でスクリプトエディターを選択します。

Unity はデフォルトで以下の IDE をサポートしています。

- **Visual Studio** は、Windows 版と macOS 版の Unity のデフォルトの IDE です。Windows 版の Unity には、Visual Studio 2019 Community も含まれます。macOS 版 Unity には、Visual Studio for Mac が含まれます。
- **Visual Studio Code**（Windows、macOS、Linux）は軽量かつ無料のカスタマイズ可能なオープンソースコードエディターで、速度とカスタマイズ性に定評があります。VS Code を Unity で使用する方法の詳細については、「Unity Development with VS Code」をご覧ください。
- **JetBrains Rider**（Windows、macOS、Linux）は ReSharper を基盤にしており、ReSharper のほとんどの機能が含まれています。詳細については、Unity 向け JetBrains Rider に関するドキュメントをご覧ください。

ご利用のテキストエディターが、サポートが組み込まれている上記のもの以外の場合は、Unity での開発用にテキストエディターのカスタマイズが必要になる場合があります。たとえば、数多くのコミュニティメンバーが Unity で Sublime Text を使用するためのパッケージ（プラグイン、拡張機能、アドオン）を作成しています。

スクリプトテンプレート

カスタムコンポーネントの作成を開始すると、新しい C# スクリプトを作成するたびに同じ変更を適用することがあります。たとえば、自動的に Update イベント関数を削除したり、デフォルトの名前空間を追加したりする必要性を感じることがあるでしょう。タスクに合わせて自分でいくつかのキー操作を保存し、スクリプトテンプレートを設定できます。

Unity では、**ScriptTemplates** リソースに保存されているテンプレートを使用します。

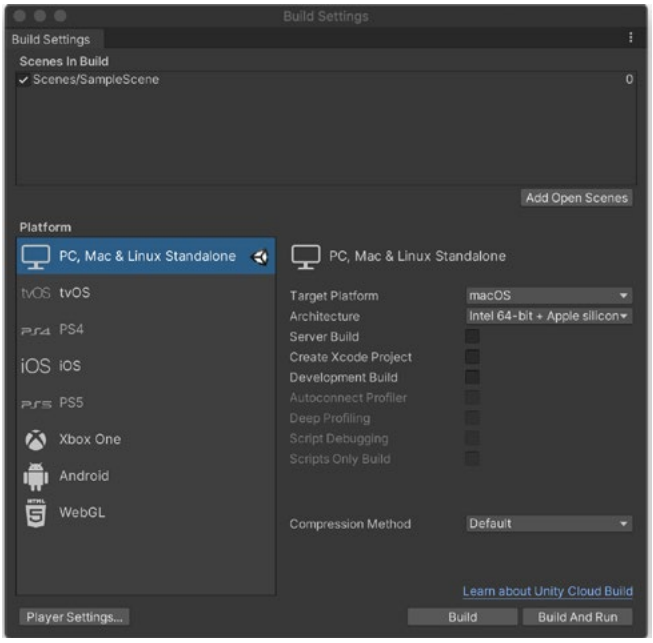
- **Windows：**
  C:\Program Files\Unity\Editor\Data\Resources\ScriptTemplates
- **Mac：**
  /Applications/Hub/Editor/[version]/Unity/Unity.app/Contents/Resources/ScriptTemplates

必要に応じてこれらのテンプレートファイルを開いて編集してから、Unity エディターを再起動して変更を適用します。変更前のテンプレートと変更後のテンプレートの両方のバックアップをかならず保管しておいてください。

ビルドと公開

Unity の強みの 1 つは、複数のプラットフォームに展開できる機能です。アプリケーションのビルドの公開設定を調整するには、**「File」>「Build Settings」**に移動します。

関連するシーンを**「Scenes in Build」**に追加します。アプリケーションのエントリポイントとなるシーンのインデックスは 0 になります。



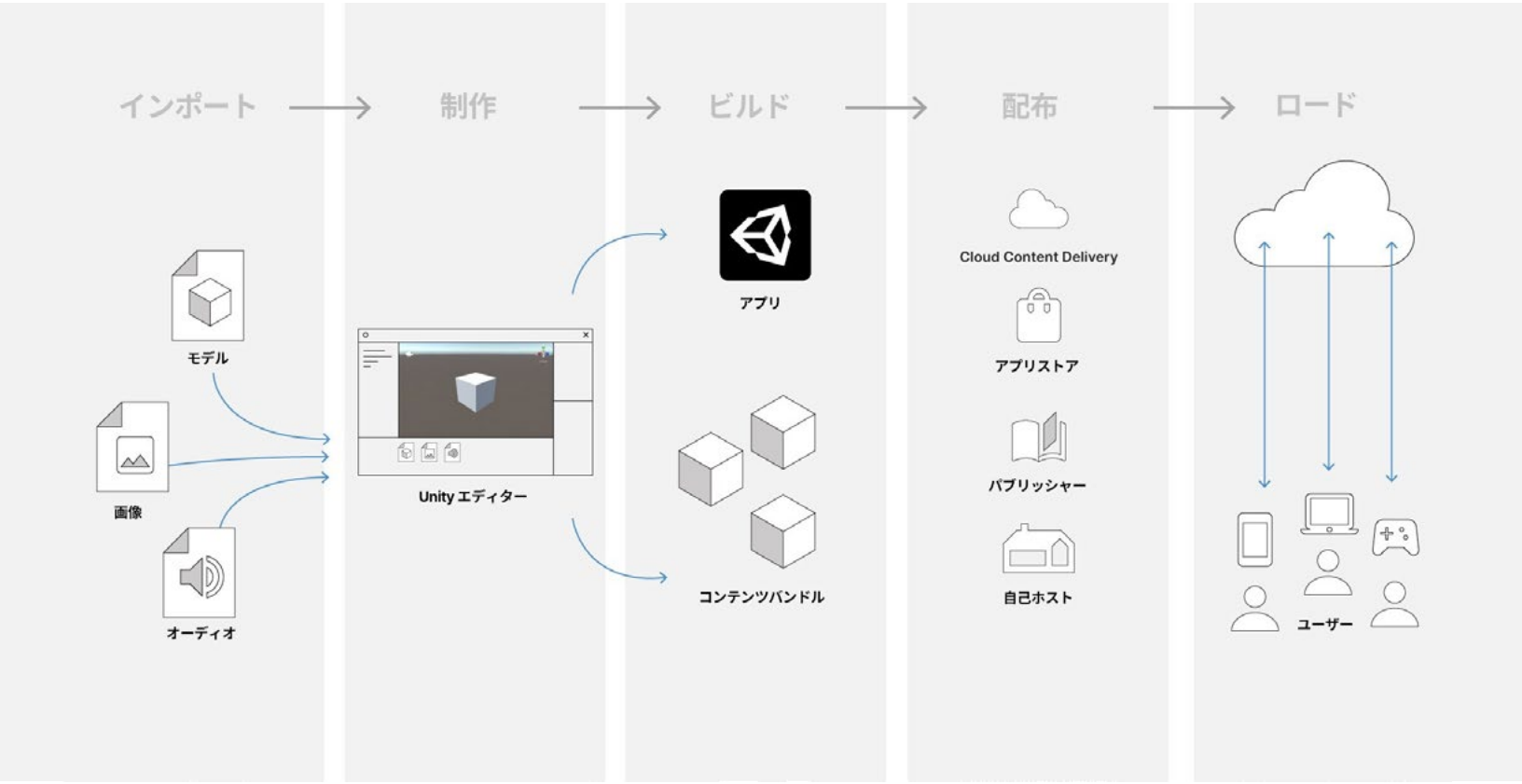
Build Settings

利用できるプラットフォームは左下に表示されます。**「Install with Unity Hub」**オプションを使用して、利用できる他のターゲットプラットフォームのサポートをインストールします。Unity Hub の**「Installs」**画面に直接アクセスし、**「Add Module」**から任意のバージョンの Unity のサポートを追加することもできます。PlayStation、Nintendo Switch、Xbox などのコンソールでは、該当するプラットフォームパブリッシャーから提供される追加モジュールが必要になります。

アプリケーションをビルドする際に、クロスプラットフォームに関する考慮事項を評価する必要があります。たとえば、モバイルデバイスはストレージやメモリが少なく、CPU の処理能力も低いため、自動メモリ管理中のガベージコレクションによるスパイクの影響も大きくなります。グラフィックス負荷の大きいコンソールゲームは、iOS や Android に移植することはできません。少なくとも、プラットフォームに合わせた大幅な最適化が必要になります。

また、複数のプラットフォームを対象としたビルドを行う際は、入力要件にもご注意ください。タッチスクリーンや加速度センサー付きのモバイルデバイスの場合は、コントローラーやキーボード、マウスで入力を行うプラットフォームとは違う設定が必要です。新しい **Input System** を使用すると、複数のデバイスによるユーザー入力を処理できます。

プラットフォーム依存のコンパイル機能によって、プリプロセッサーディレクティブでスクリプトを分割し、ターゲットプラットフォームに合わせることができます。これを、#if コンパイラディレクティブや ConditionalAttributeClass と組み合わせることができます。プラットフォーム固有のリリースの UI やオブジェクトを追加または削除する場合（iOS アプリケーションから終了ボタンを削除するなど）、プラットフォーム依存のコンパイルによってそのロジックを簡略化できます。



ワークフローの構築と公開

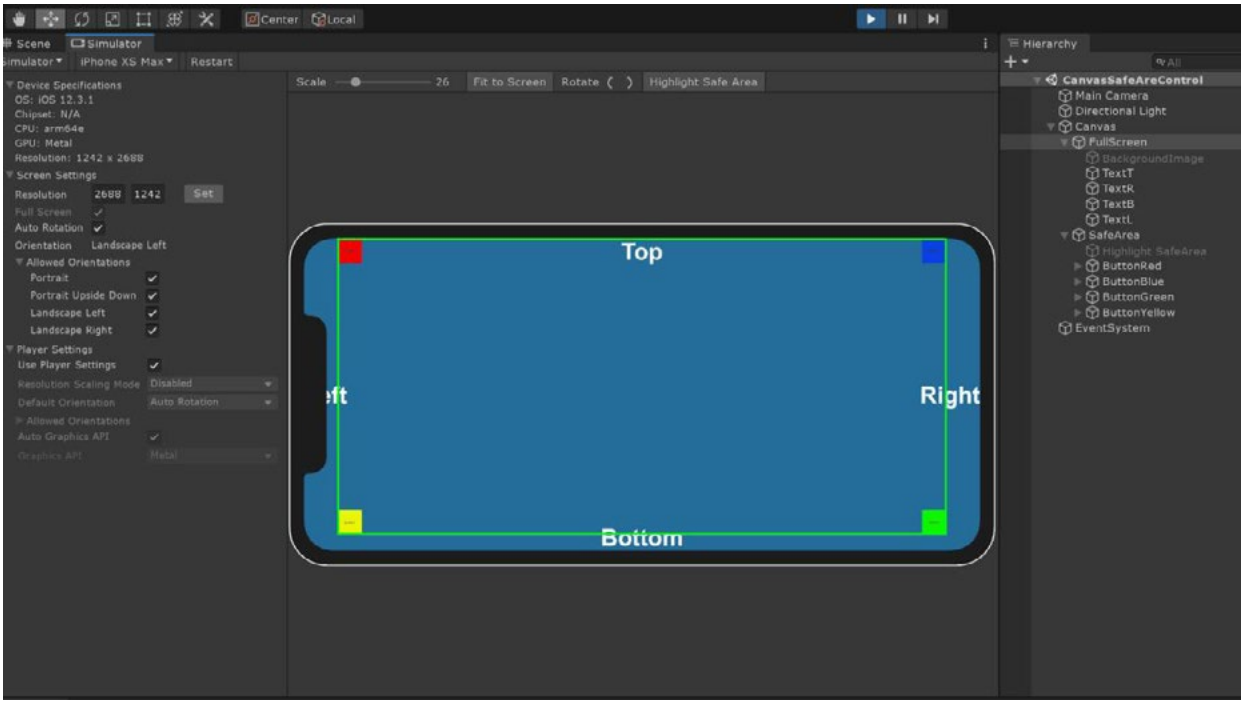
Device Simulator

モバイル向けにビルドを行う場合は、パッケージマネージャーを使用して Device Simulator をインストールすることをおすすめします。これにより、シミュレートされたモバイルデバイス上にアプリケーションを表示し、スクリーンの形状や解像度、向きをプレビューできます。

Device Simulator のビューを使用する手順は次のとおりです。

- ゲームビューの左上にドロップダウンメニューがあります。このメニューから、ゲームビューと Device Simulator ビューを切り替えることができます。
- 上部のメニューから、**「Window」>「General」>「Device Simulator」**を選択します。

Device Simulator



Device Simulator の主な用途は、アプリケーションのレイアウトを表示して基本操作をテストすることです。

なお、Device Simulator のビューでは、プロセッサーの処理速度や利用可能な RAM など、デバイスのパフォーマンスやレンダリング特性がシミュレートされないことに注意してください。

利用方法については、『Simulate your Game with Device Simulator in Unity』をご覧ください。

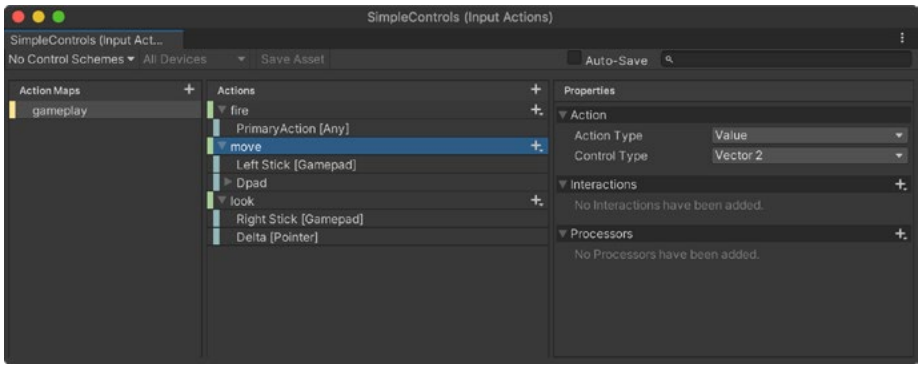
入力

Unity には主な入力処理方法が 2 つあります。パッケージマネージャーの Input System と、ビルトインの Input クラスです。

Input System

Input System は、使いやすさと柔軟性、複数のデバイスやプラットフォームにわたる一貫性を重視しています。ビルトインの Input クラスの代替手段になります。

Input System はデバイスから直接入力内容を取得できますが、効果を発揮するのは、プレイヤーインタラクションを促進するために一連の間接的なアクションを作成する際です。その後、バインディングやアクションのコレクションである actionMap を作成できます。このアクションマップによって、対象のアクションを実際のデバイスに関連付けることができます。



Input System はパッケージマネージャーで入手できます。

より多くのデバイスに入力を適応させる場合は、特定のデバイスのボタンやキーボードのキーなどをハードコーディングするのではなく、アクションマップを追加で作成します。コールバックとアクションを 1 度設定すれば、その後はマッピングを追加することで、より多くのデバイスをサポートできます。

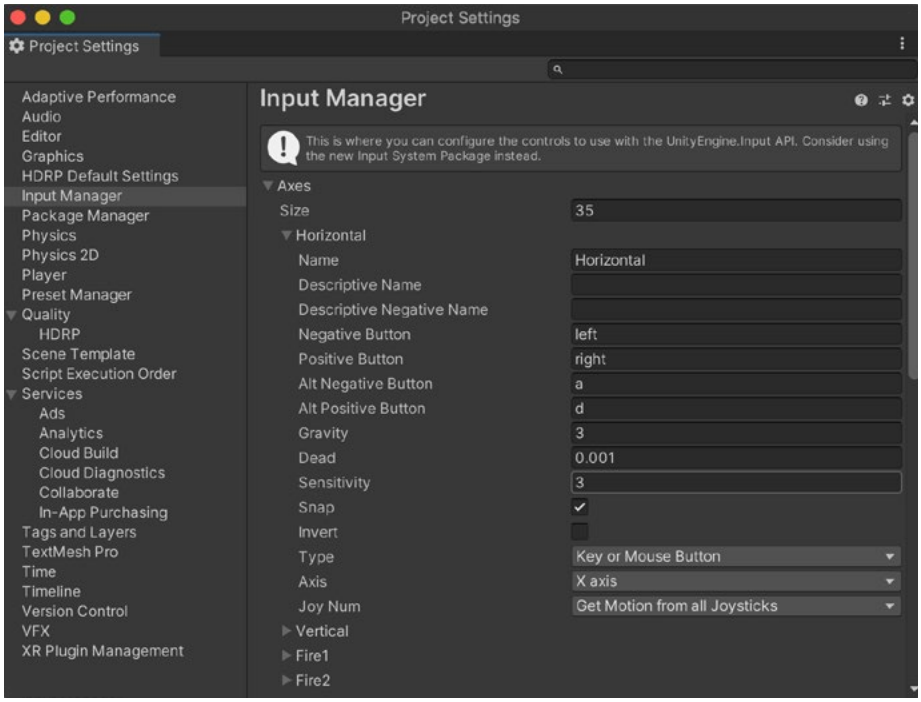
アクションマップは、ゲーム内のコンテキストに応じて入力を切り替える際にも役立ちます。たとえば、車両を運転しているときと、歩いているときまたは走っているときで、プレイヤー入力の動作を変えることができます。

Input System のクイックスタートガイドに目を通し、パッケージマネージャーを使って入手できるサンプルをいくつかインストールしてください。

ビルトインの Input クラス

代替手段として、オリジナルのビルトインの Input クラスを使用することもできます。この方法では、Input Manager（**「Edit」>「Project Settings」>「Input Manager」**）を利用して、キーやボタン、その他の入力デバイスの仮想軸を設定できます。モバイルデバイスでのマルチタッチや加速度センサーのデータにも対応しています。

Input System パッケージ（上述）を有効にすると、このビルトインの Input Manager は無効になる点にご注意ください。



ビルトインの Input Manager

入力用の便利なクラスやメソッドの例は次のとおりです。

Input	従来型のゲーム入力で設定された軸を読み取り、モバイルデバイスでマルチタッチや加速度センサーのデータにアクセスするために使用します。
Input.GetAxis	axisName で特定された仮想軸の平滑値を返します（キーボードやジョイスティックデバイスの場合は -1 ～ 1）。マウスデバイスの場合は、現在のマウスのデルタ値に軸感度をかけた値になります。
Input.GetAxisRaw	Input.GetAxis と同様ですが、スムージングフィルターがありません。
Input.GetButton	buttonName によって特定された仮想ボタンが押されている間に true を返します。
Input.GetButtonDown	buttonName によって特定された仮想ボタンをユーザーが押しているフレーム中に true を返します。
Input.GetKey	name によって特定されたキーをユーザーが押している間に true を返します。
Input.GetKeyDown	name によって特定されたキーをユーザーが押し始めたフレーム中に true を返します。
Input.touches	最後のフレーム中のすべてのタッチ操作のステータスを表す読み取り専用のオブジェクトリスト。
Touch	画面の指でのタッチ操作のステータスを表す構造。
KeyCode	キーの押下、マウス、ジョイスティックのオプションをすべてリストアップします。

ビルトインの入力システムの概要については、入力に関するマニュアルのページをご覧ください。

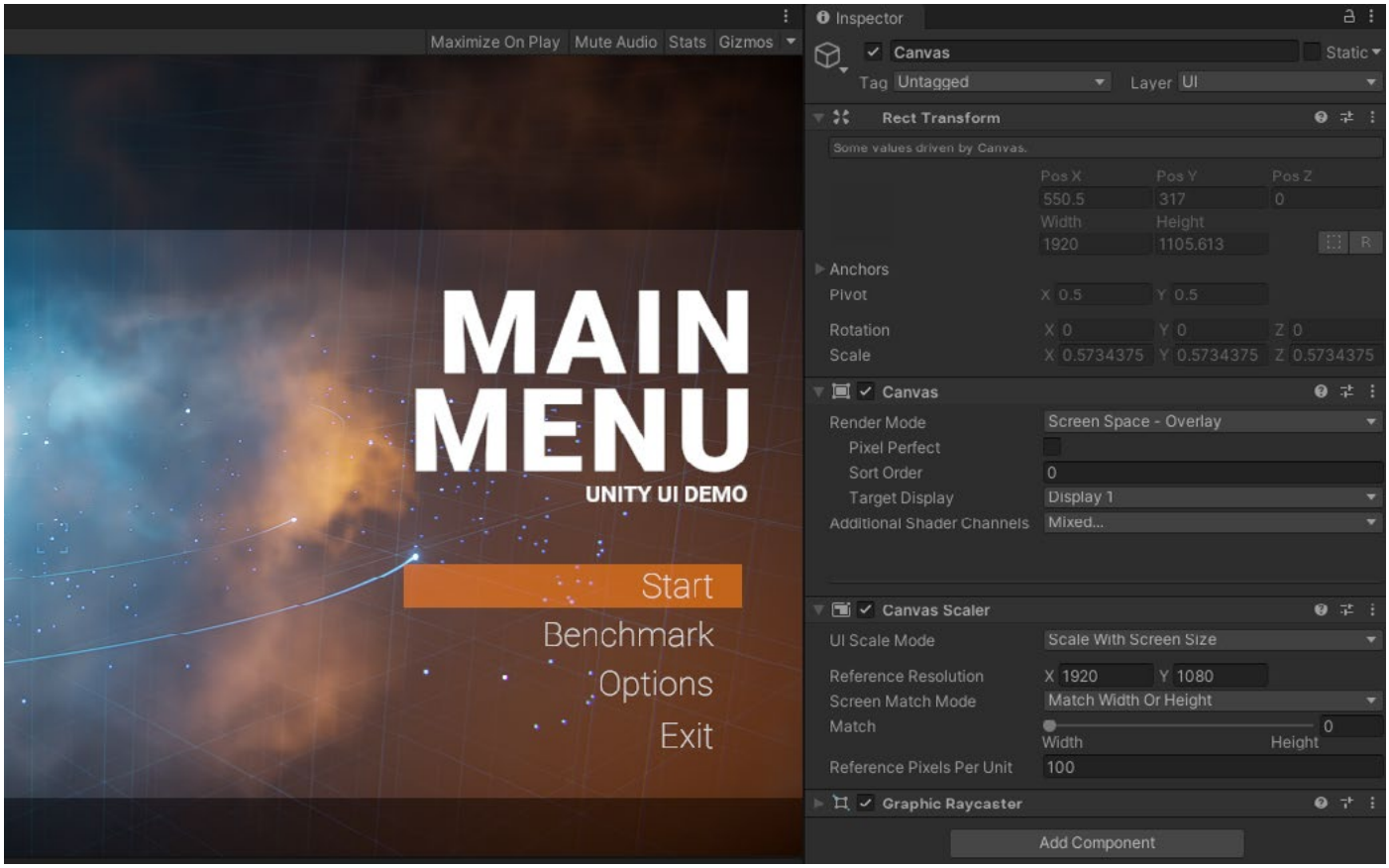
ユーザーインターフェース

Unity では、ゲーム内ユーザーインターフェースとエディタースクリプト用に、複数の UI システムが用意されています。ユースケースに合わせて、いずれかを使用することも、複数使用することもできます。

Unity GUI

Unity UI（UGUI と呼ばれることもあります）では、ゲームオブジェクトベースのアプローチで UI 要素を編集、配置できます。Unity UI は、ゲームやアプリケーション用ユーザーインターフェースを開発するためのシステムです。コンポーネントやゲームビューを使用して、ユーザーインターフェースの調整、配置、スタイル設定を行います。Unity UI を使用して、Unity エディターのユーザーインターフェースを作成または変更することはできません。

Unity UI ガイドには、UI 要素のレイアウト方法やインタラクションの作成方法がまとめられています。



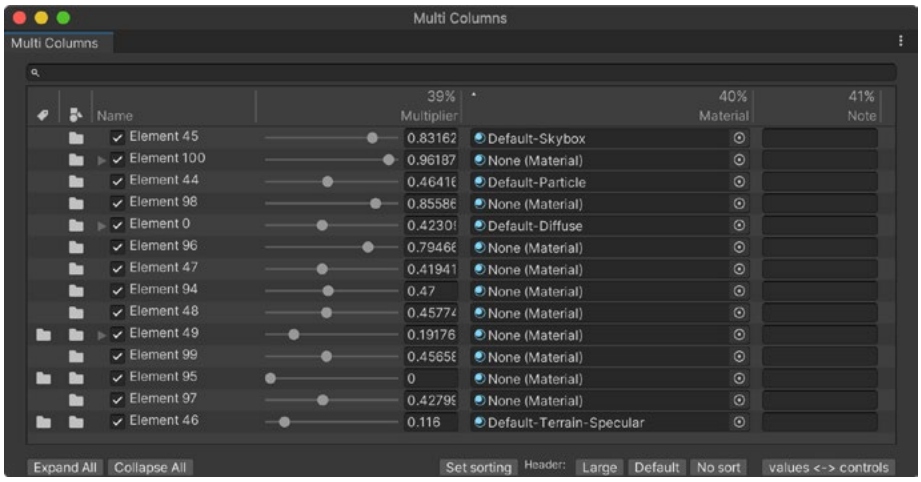
Unity の UI 要素はゲームオブジェクトベースのアプローチ

即時モード GUI

即時モード GUI（IMGUI）は、カスタムのインスペクターやエディターウィンドウの作成に役立ちます。IMGUI 要素を作成するには、OnGUI という特殊な MonoBehaviour 関数内にコードを記述します。基本的にこのシステムは、通常のゲーム内ユーザーインターフェースを想定したものではありません。

このコードは「即時モード」でユーザーインターフェースを表示するもので、毎フレーム実行されます。OnGUI コードがアタッチされるオブジェクト以外に、永続的なゲームオブジェクトはありません。このコードでは、単一の関数呼び出しによって描画、処理される GUI コントロールを生成します。

IMGUI が必要な場合は、こちらのスクリプティングガイドをご覧ください。



即時モード GUI で作成されたエディター UI

プロファイリングと最適化

素晴らしいゲームをデザインしたら、次のステップはそれを高速化することです。Unity には、ターゲットプラットフォームで利用可能なリソースを最大化するのに役立つ一連のプロファイリングおよび最適化ツールが含まれています。

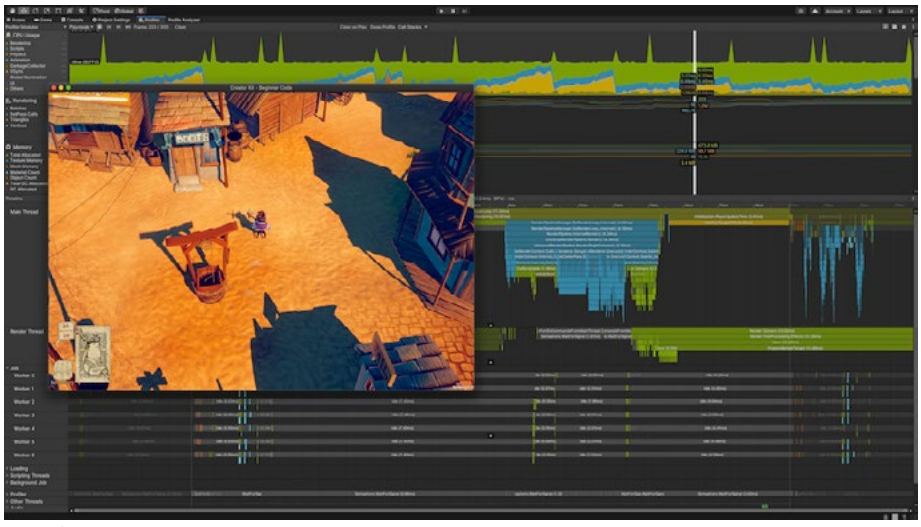
Unity プロファイラー

Unity プロファイラーでは、アプリケーションのパフォーマンスに関する情報を確認できます。最適な結果を得るためには、ビルドを公開するエンドプラットフォームでこれを実行します。そうすることで、どのようなタイミングでアプリケーションのパフォーマンスに影響が生じるのかを正確に把握できます。

プロファイラーは、エディターの再生モードでも機能します。このモードでは精度が犠牲になるため、手早くチェックしたいときは便利ですが、最終的な評価には適していません。

プロファイラーは、CPU、GPU、メモリ、レンダラー、物理演算、オーディオなど、アプリケーション内の改善の余地がある領域を特定するのに役立ちます。そうした領域のイテレーションを行うことができます。コードやアセット、シーン設定、カメラのレンダリング、ビルド設定がアプリケーションの実行にどのように影響しているのかなどを突き止めることができます。

プロファイラーには、一連のグラフとして結果が表示されるため、スパイクが生じた場所を可視化できます。



Unity プロファイラー

Unity はさまざまなプロファイラーマーカーを提供しており、アプリケーション内で時間がかかっている要素に関するデータを確認できます。ProfilerRecorder では、そうしたマーカーを使用して、有用なフレームのタイミングを取得することもできます。

特に一般的なマーカーは次のとおりです。

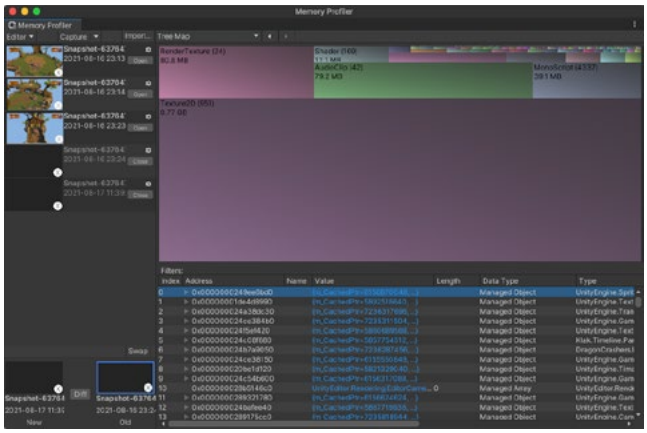
プロファイラーマーカー	概要
メインスレッドのベースマーカー	メインのゲームループ、PlayerLoop、EditorLoop（エディター内でプロファイリングを行う場合）のサンプルを含むメインスレッド上のマーカー
スクリプト更新マーカー	MonoBehaviour.Update メソッド（Update/FixedUpdate/LateUpdate）とコルーチンのサンプルを含む
レンダリングおよび VSync マーカー	CPU が GPU のデータ処理に時間を費やしている場所や、GPU の処理終了を待っている可能性がある場所のサンプルを含む
バックエンドのスクリプティングマーカー	Mono または IL2CPP のスクリプティングバックエンドアクティビティをハイライトし、ガベージコレクションやガベージの割り当ての問題のトラブルシューティングに役立つ
マルチスレッディングマーカー	消費された CPU サイクルは測定せず、スレッド同期と Job System 関連の情報をハイライトするサンプルを含む
物理演算マーカー	Physics.Contacts、Physics.TriggerEnterExits、Physics.UpdateBodies などをサンプリングする高レベルの物理演算プロファイラーマーカーを含む
パフォーマンスに関する警告	プロファイラーによって、パフォーマンスが重視されるコンテキストで避けなければならない特定の呼び出しが検出されたときに表示される

「Profiler」ウィンドウアクセスするには、メニューから**「Window」>「Analysis」>「Profiler」**を選択します。このウィンドウの概要については、「Profiler」ウィンドウに関するドキュメントをご覧ください。

メモリプロファイラー

メモリプロファイラーは、Unity エディターと Unity プロジェクトの中で、メモリ使用量を削減できる領域を特定できるツールです。ネイティブメモリとマネージドメモリのアロケーションの概要のほか、メモリに保持するそれらの間の参照を把握できます。

メモリプロファイラーを使用することで、メモリのスナップショットを取得、検査、比較して、メモリのリークやフラグメンテーションを検出できます。

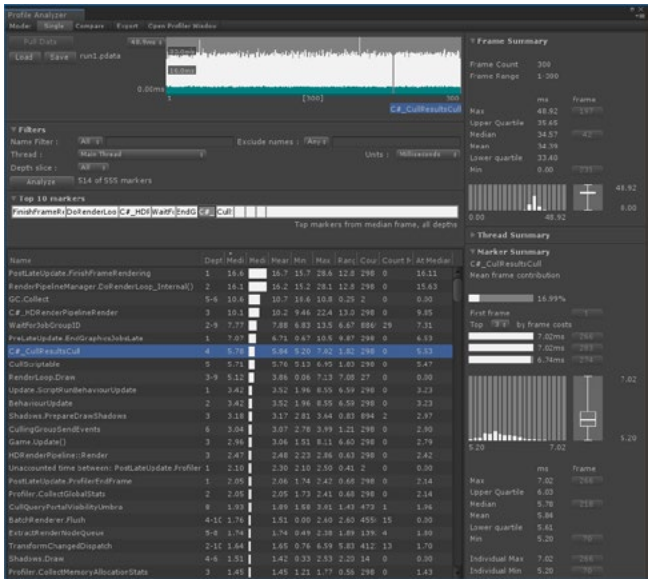


メモリプロファイラーのメインビュー（「Tree Map」ビュー）

Profile Analyzer

Profile Analyzer は、Unity プロファイラーの一連のフレームからフレームとマーカーのデータを集計して可視化し、その動作を把握するのに役立ちます。Profile Analyzer を使用することで、2 つのデータセットを並べて比較し、Unity プロファイラーの既存の単一フレーム分析を補完できます。

Profile Analyzer の使用方法の詳細については、このウィンドウに関するドキュメントをご覧ください。



Profiler Analyzer

Unity でのモバイルプラットフォーム向けの最適化について、より詳しく知りたい方は、ブログ記事の「Optimize your mobile game performance」をご覧ください。Unity エキスパートによるモバイルゲーム向けの最適化のヒントをまとめた詳細な e ブックもダウンロードできます。

デバッグとテストプレイ

Unity は調整やデバッグに適した便利なツールです。いつでもエディターの再生モードに入り、エディターでアプリケーションを実行しながらあらゆるゲームプレイ変数を確認できます。

再生モードでは、ゲームビューを使用して、リリース用の最終的なアプリケーションをプレビューできます。Unity シーンをリアルタイムで変更、テストできるほか、任意のタイミングで一時停止することも、コードを 1 行ずつステップ実行することもできます。

テストプレイを円滑化するため、次のようなチートを作成することもできます。

- レベル、キャラクター、アイテムなどをアンロックする
- 要素やゲームプレイを無効にする
- GUI を切り替える
- 無敵状態にする
- 時間、お金、コレクタブルなどを増減させる

ゲームプレイのデバッグ

テストプレイに厳密な決まりはありませんが、以下のことを考慮しておくのがお勧めです。

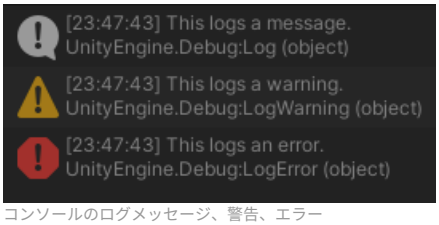
- プレイヤーのワールド座標を出力するためのショートカットを実装しましょう。ステージの特定の場所で特定のバグが発生しているかどうかを確認する際に便利です。
- チームの規模が小さい場合は、チームメンバーごとにテストプレハブを作成し、コミットされていないファイルから設定とデバッグオプションを読み取ります。こうすることで、うっかりテストオプションをコミットしたり、本番環境のシーンに変更したりすることがなくなります。
- テストシーンやサンドボックスシーンには、すべてのゲームプレイ要素を保持します。例えば、すべての敵や操作可能なオブジェクトなどを保持したままシーンを作成するということです。こうすることで、ゲームを通しでプレイしなくても簡単に機能をテストすることができます。
- エディターが便利に使えるようになる方法で記述します。静的メソッドには MenuItem 属性をアタッチして、Application.isPlaying を確認できるようにしてから、ロジックを実行します。

デバッグに関するその他のヒント

Unity には、エディターで実行中に情報を可視化できる Debug クラスが含まれています。これを利用することで、メッセージや警告を「Console」ウィンドウに出力し、Unity が生成するエラーや警告、その他のメッセージを表示できます。

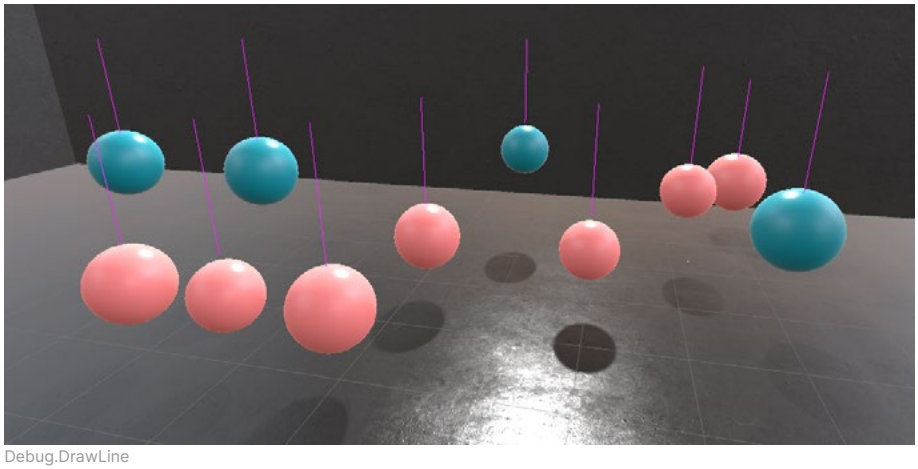
Debug を使用して、シーンビューとゲームビューに可視化ラインを描画したり、スクリプトからエディターの再生モードを一時停止したりできます。

- インスペクターで特定の値をチェックする必要があるものの、アプリケーションを手動で一時停止するのが難しい場合は、Debug.Break による実行が便利です。
- コンソールのメッセージに Debug.Log、Debug.LogWarning、Debug.LogError を使用し、さまざまな重大度を付けてフォーマットすることができます。



コンソールのログメッセージ、警告、エラー

- **Debug.Log** を使用する場合は、コンテキストとしてオブジェクトを渡すことができます。コンソールでメッセージをクリックすると、「Hierarchy」ウィンドウでゲームオブジェクトがハイライト表示されます。
- リッチテキストを使用して、**Debug.Log** ステートメントをマークアップします。これがコンソールのエラーレポートの改善に役立ちます。
- 物理演算に関するトラブルシューティングを行う際には、Debug.DrawLine と Debug.DrawRay がレイキャスティングの可視化に役立ちます。


Debug.DrawLine

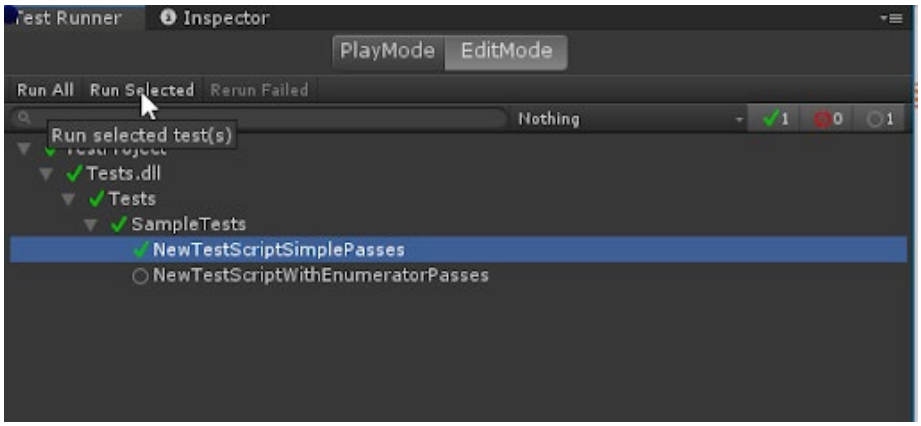
Unity Test Framework（UTF）

Unity Test Framework（旧 Unity Test Runner）を利用すると、自動テストを作成し、コードが適切に実行されているか確認できます。プロジェクトの開発中に、論理的なコードスニペットの単体テストを作成してテスト駆動開発を適用できます。

UTF では、編集モードと再生モードの両方でコードをテストできます。スタンドアロン、Android、iOS などのターゲットビルドでテストを実行することもできます。UTF は、.NET 言語向けのオープンソースの単体テストライブラリである NUnit ライブラリを拡張します。

Test Framework を開くには、**「Window」>「General」>「Test Runner」**の順に選択します。手順に沿って作業フォルダーとアセンブリの定義を設定してから、単体テストをテストアセンブリに追加します。このプロセスは、バグを迅速に隔離し、テスト可能な方法でコードを作成する方法を把握するために役立ちます。

Test Framework の詳細については、ブログ記事の「Performance Benchmarking in Unity」と、Unity Test Framework に関するドキュメントをご覧ください。



Unity Test Framework

パーティクルエフェクト

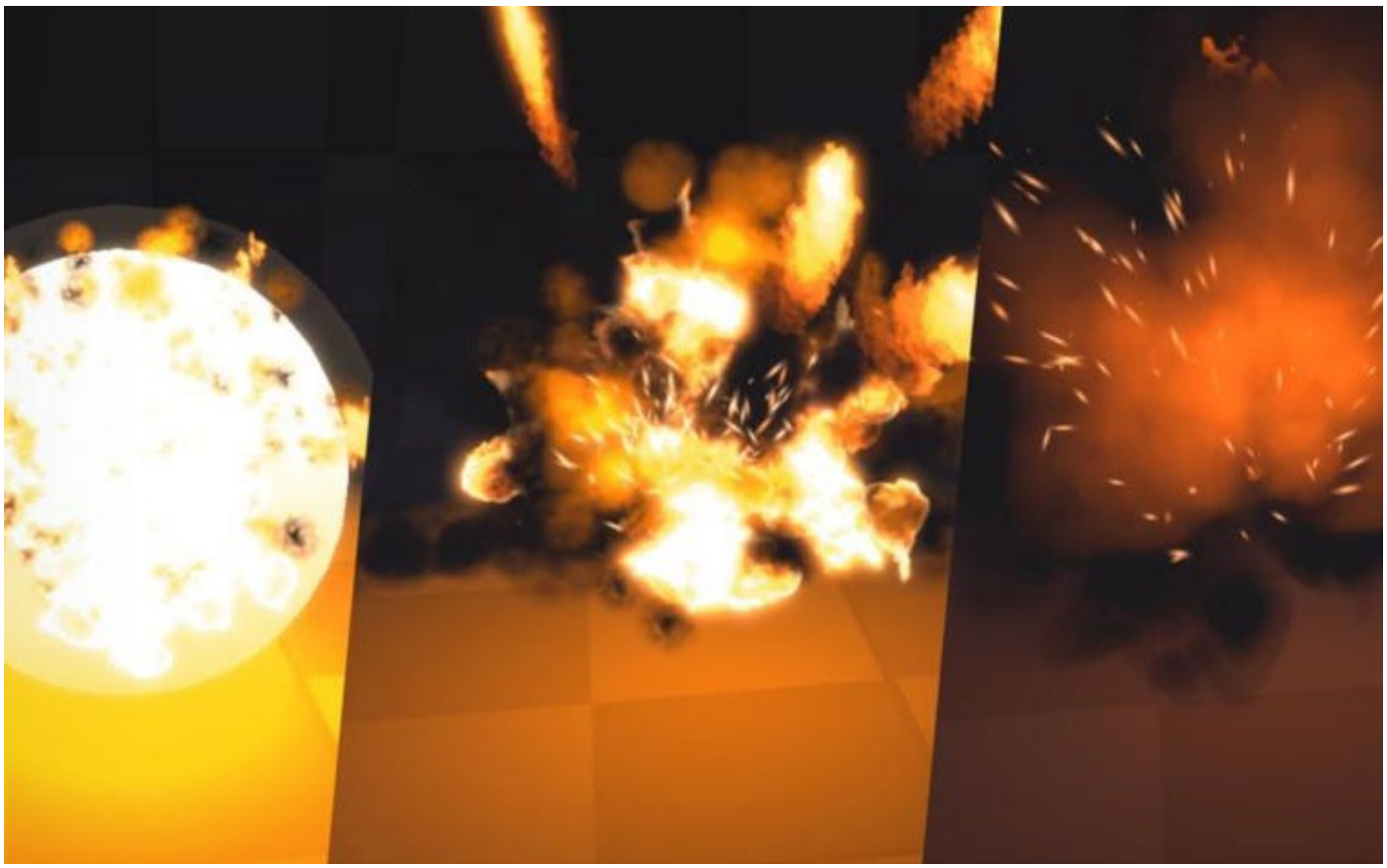
Unity では、エフェクトのニーズ（煙、液体、炎）に合わせて、2 種類のパーティクルシミュレーションソリューションを利用できます。

— パーティクルシステムでは、CPU 上で数千のパーティクルのシミュレーションを行うことができます。パーティクルシステムコンポーネントのモジュールを使用して、事前定義済みの動作を表現できます。各エフェクトを作成する際は、多くの場合、複数のゲームオブジェクトと ParticleSystem を重ねます。

ParticleSystem は、ビルトインレンダーパイプラインと URP をサポートしています。ParticleSystem クラスにアクセスして、スクリプトでシステムとその個々のパーティクルを定義できます。

パーティクルシステムでは、Unity の基盤の物理演算システムや、シーン内のあらゆるコライダーを使用できますが、フレームバッファにはアクセスできません。

たとえば、ビルトインの ParticleSystem では、Asset Store から Particle Pack をダウンロードできます。



パーティクルシステムを使用したシンプルなエフェクトシミュレーション

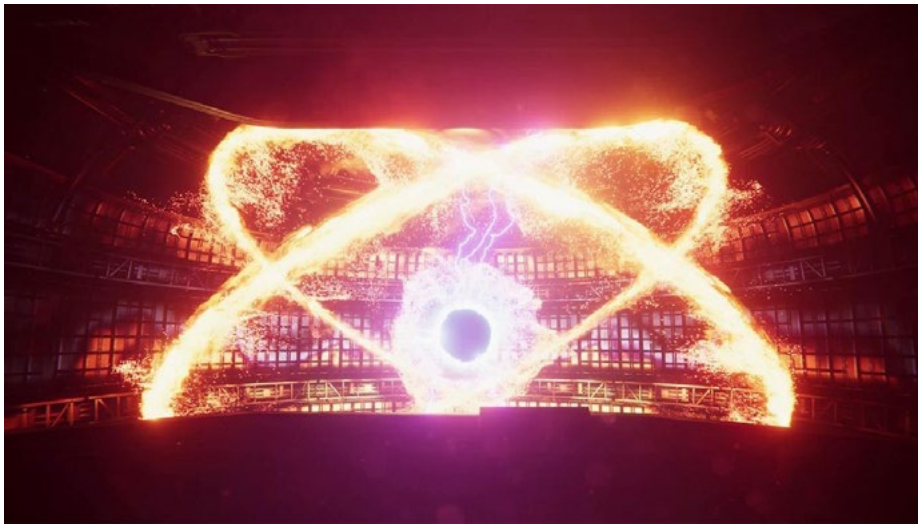
— **Visual Effect Graph** は、コンピュートシェーダーを使用して GPU で計算を実行します。これにより、色や深度バッファとも相互作用する数百万のパーティクルのシミュレーションを行うことができます。

ワークフローには、詳細にカスタマイズできるグラフビューが含まれます。

Visual Effect Graph は、基盤の物理演算システムにはアクセスできませんが、ポイントキャッシュやベクトル場、符号付き距離場などの複雑なアセットを扱うことができます。

Visual Effect Graph は、コンピュートシェーダーと HDRP をサポートしているプラットフォームでのみ機能します (現在 URP のサポートはプレビュー中)。イベントインターフェースを使用してカスタムイベントを送信し、アタッチされたデータを渡してグラフで処理できます。Visual Effect コンポーネントでは、再生コントロール API も利用できます。

Visual Effect Graph の制作例を示す GitHub プロジェクトが 2 つ用意されています。それが、Visual Effect Graph のサンプルプロジェクトと、宇宙船のデモです。この 2 つのプロジェクトを見ると、Visual Effect Graph を使用して、無数の高品質なエフェクトを作成する方法がわかります。



Visual Effect Graph で画面上に作成した無数のパーティクル

2 つのシステムのどちらかを選ぶ際は、デバイスの互換性を常に考慮します。大半の PC やコンソールはコンピュートシェーダーをサポートしていますが、多くのモバイルデバイスはサポートしていません。現在のところ、目的のプラットフォームがモバイルであれば、ビルトインの ParticleSystem をお勧めしています。ターゲットプラットフォームがコンピュートシェーダーをサポートしていなくても、Unity なら、プロジェクトで両方のタイプのパーティクルシミュレーションを利用できます。

物理演算

Unity には、3D プロジェクト用の NVIDIA PhysX エンジンと 2D プロジェクト用の 2D エンジンが実装されています。Unity のビルトインの物理演算エンジンは、Rigidbody のインタラクション、ジョイント、フォースに対応した業界標準のソリューションを提供します。レイキャスト、シェイプキャスト、オーバーラップテスト、トリガー / 衝突処理のメソッドには、物理演算 API を使用します。

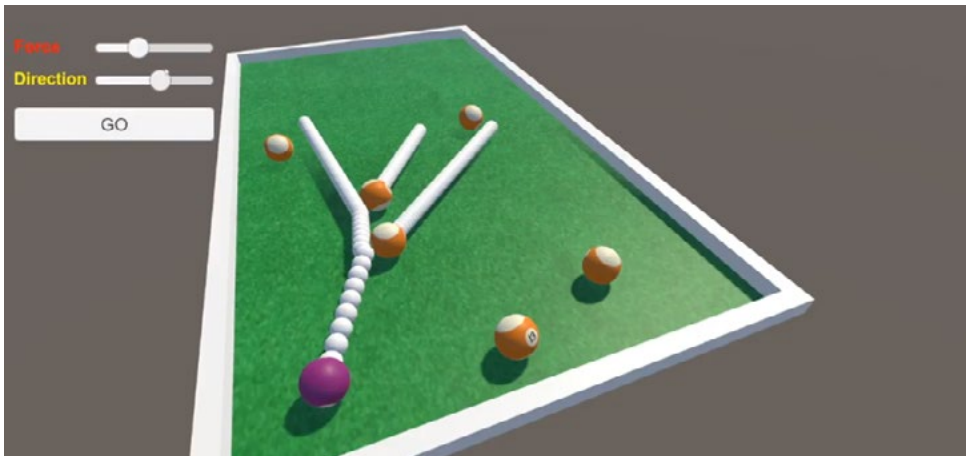
3D 物理演算

Unity のビルトインの物理演算エンジンには、以下の便利なコンポーネントが付属しています。

- **Rigidbody**：物理演算エンジンでオブジェクトの動きを制御できます。Rigidbody は重力により落下します。提供されるメソッドを使用して、フォースやトルクを適用できます。コライダーコンポーネントがアクティブな場合、基礎となるゲームオブジェクトが物理演算の衝突に応答します。
- **コライダー**（Box Collider、Sphere Collider、Capsule Collider、Mesh Collider）：物理演算の衝突の境界を定義するプリミティブボリュームまたはメッシュボリュームを表します。
- **ジョイント**（Fixed Joint、Hinge Joint、Spring Joint、Character Joint）：2 つの Rigidbody コンポーネントを連結して、さまざまなモーションの制約（破壊可能なジョイント、1 つの回転軸、弾性のあるジョイント、ボールとソケットなど）をモデル化できます。
- **CharacterController**：Rigidbody を利用することなく、衝突によってコンストレインされた動きを処理できます。

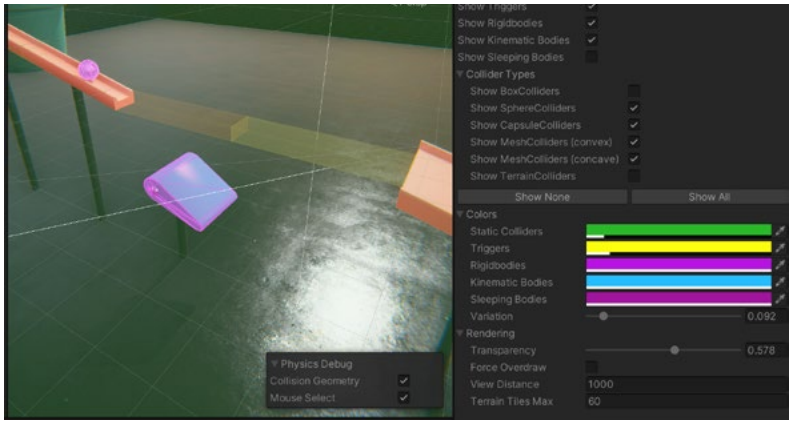
Rigidbody コンポーネントは、高速で動く物体の投機的連続型衝突判定とスイープに基づく連続型衝突判定を備えています。

複雑な物理演算コンテキストの管理や対処には、マルチ物理シーンを使用します。例えば、ゲームオブジェクトの衝突や軌道を予測するために、複数の物理シーンのシミュレーションを行うことができます。



複数のシーンを使用して衝突や軌道を予測する

Unity には、デバッグの可視化ウィンドウ（**「Window」>「Analysis」>「Physics Debugger」**）もあります。このウィンドウでは、物理演算コンポーネントにカテゴリ別に色を付けたり、切り替えたりできます。そのため、シーン内のコライダーなどの物理演算ベースのシナリオのトラブルシューティングをすばやく行うことが可能です。



Physics Debugger ウィンドウ

物理演算の更新は、固定時間ステップで呼び出される MonoBehaviour の FixedUpdate メソッド内で適用されます。Physics クラスには、物理インタラクションに使用する静的メソッドがいくつか用意されています。また、Collider クラスには、衝突やトリガーを処理する物理演算イベントがいくつか用意されています。以下に、一般的な物理インタラクションに使用するメソッドをいくつか紹介します。

クラス / メソッド	**説明**
Monobehaviour.FixedUpdate	**「Edit」>「Settings」>「Time」>「Fixed Timestep」**で定義される固定フレームレートでの物理演算の更新メッセージ
Physics.Raycast、Physics.Linecast、Physics.BoxCast、Physics.CapsuleCast、Physics.SphereCast	コライダーがレイ、線、形状のどれになるか定義するメソッド
Physics.OverlapBox、Physics OverlapSphere、Physics OverlapCapsule	コライダーが指定の形状になっているかテストするメソッド
Collider.OnCollisionEnter、Collider.OnCollisionStay、Collider.OnCollisionExit	このオブジェクトのコライダーに別のコライダーが接触したときと、接触状態を継続したとき、離れたときに送信されるメッセージ
Collider.OnTriggerEnter、Collider.OnTriggerStay、Collider.OnTriggerExit	このオブジェクトのコライダーに別のトリガーが接触したときと、接触状態を継続したとき、離れたときに送信されるメッセージ（トリガーでは衝突は発生しませんが、メッセージは送信されます）

2D 物理演算

Unity は、2D 物理演算の処理に最適化された個別の物理演算エンジンを備えています。2D 物理演算コンポーネントのほとんどは、同等の 3D 物理演算コンポーネントを単純に「フラット化した」ものです。

対応するコンポーネントは、Physics2D、Rigidbody2D、Collider2D、Joint2D といったように、名前に「2D」が付加されています。それ以外の点については、同等の 3D クラスと同じように動作します。

ユースケースの例については、PhysicsExamples2D プロジェクトをぜひダウンロードしてください。また、こちらの関連する紹介ビデオをご覧ください。

ビルトインの 3D および 2D 物理演算の実装の詳細については、物理演算に関するドキュメントを参照してください。3D 物理演算およびその設定を最適化するためのヒントについては、物理演算のパフォーマンスの最適化に関するビデオをご覧ください。

オーディオ

Unity のオーディオシステムは、3D 空間でサウンドを再生するための洗練された機能を備えています。また、ゲームプレイ中や、保存、送信の際に使用するオーディオを、利用可能なマイクで録音することもできます。

オーディオコンポーネントは、現実世界のオーディオコンポーネントを模倣します。AudioSource は、ゲームオブジェクトにアタッチすると、AudioClip と呼ばれるオーディオファイルを再生できるようになります。再生すると、（通常はメインカメラにアタッチされている）シーン内のどこかにある AudioListener がこのクリップのサウンドを拾います。

オーディオシステムはこのようにして、ソースとリスナーが互いに動いた場合のドップラー効果を再現できます。エコーなどのエフェクトは、オーディオフィルターを使用して実現されます。

AudioMixer は、さまざまなオーディオソースをブレンドし、エフェクトを適用して、マスタリングを行います。

キャプション：AudioMixer

以下は、よく使用されるオーディオコンポーネントクラスです。

クラス	説明
AudioClip	Unity にインポートされたオーディオファイルは、スクリプトから AudioClip として利用できます。この方法を使って、ランタイム時にオーディオシステムからエンコードされたオーディオデータにアクセスすることができます。AudioClip のメタ情報は実際のオーディオデータがロードされる前でもロードできます。
AudioSource	ゲームオブジェクトにアタッチされている AudioSource は 3D 環境でサウンドを再生します。
AudioListener	AudioListener は周囲のすべてのサウンドを拾うマイクのようなデバイスです。リスナーはシーンごとに 1 つあります。
AudioMixer	オーディオミキサーグループは基本的に、音量の減衰やピッチ調整を適用できる信号チェーンです。オーディオミキサーグループを使用すると、オーディオ信号にエフェクト処理を施したり、マスタリングを行ったりできます。

レンダーパイプラインとグラフィックス

グラフィックスなくしてゲームは完成しません。シーンのライティングに取り掛かる前に、さまざまなレンダーパイプラインから 1 つを選ぶ必要があります。レンダーパイプラインとは、シーンのコンテンツを画面上に表示する一連の処理を実行するものです。

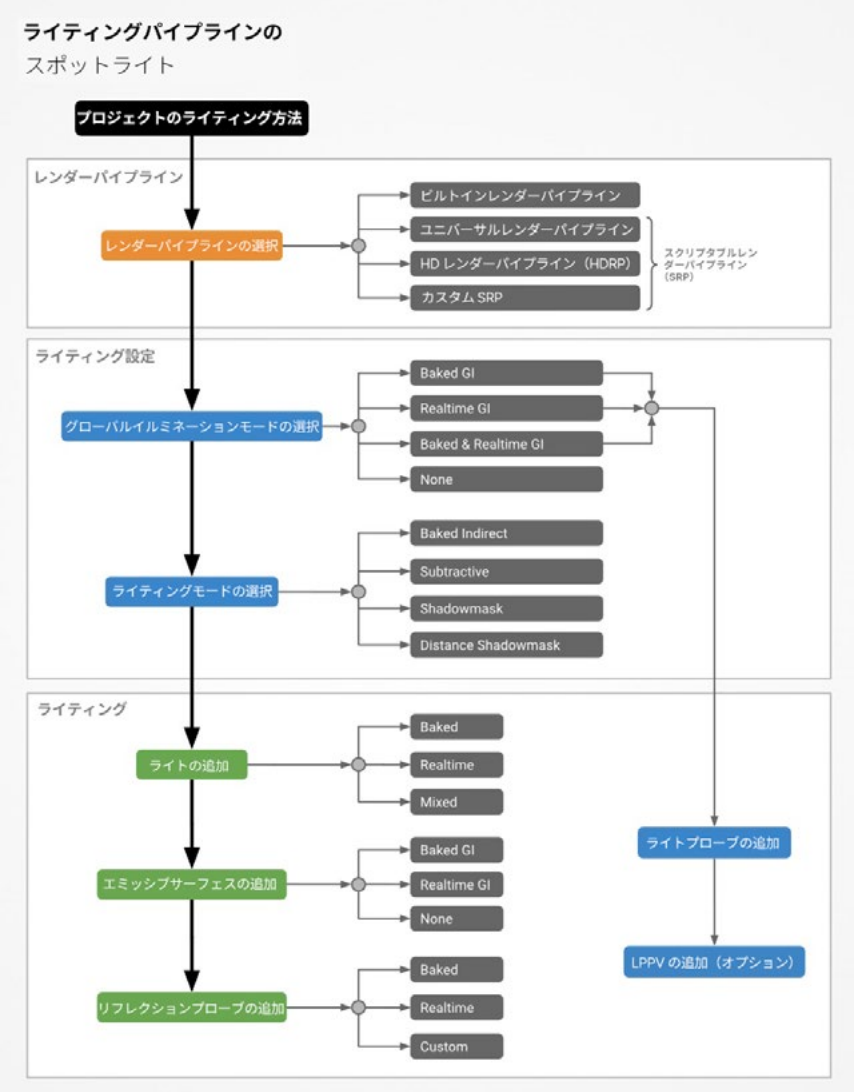
3D パイプライン

Unity には、3 つの既成のレンダーパイプラインが用意されています。それぞれ機能やパフォーマンス特性が異なります。独自のレンダーパイプラインを作成することもできます。

- **ビルトインレンダーパイプライン**は汎用のレンダーパイプラインで、カスタマイズできる範囲は限定的です。これがデフォルトのレンダーパイプラインです。
- **ユニバーサルレンダーパイプライン（URP）**は、既成のスクリプタブルレンダーパイプライン（SRP）の一種です。URP では、アーティストにとって使いやすいワークフローで、最適化されたグラフィックスを作成できます。1 つの Unity プロジェクトで、モバイル、デスクトップ、コンソールなど、複数のプラットフォームをターゲットとする場合は、URP の使用を検討してください。

  URP は、ビルトインレンダーパイプラインでは利用できないグラフィックス機能やレンダリング機能を備えています。パフォーマンスの維持と引き換えに、ライティングとシェーディングの計算コストを下げます。Unity でサポートされているすべてのプラットフォームをサポートし、それらに対して適切にスケールします。モバイル、Nintendo Switch、または Oculus Quest 向けに制作する場合は、URP を選択します。
- **HD レンダーパイプライン（HDRP）**も既成のスクリプタブルレンダーパイプラインの一種で、PC、Xbox、PlayStation などのハイエンドなハードウェア向けです。Unity で利用できる最高品質の物理ベースのライティングとレンダリングをサポートします。写実的な表現を求めている場合は HDRP を選択します。

  HDRP では、さまざまなライトの種類（ピラミッド / ボックスのスポットライト、リアルタイムのチューブ、リアルタイムの長方形）とより高度なリフレクション（ブレンディングを使用したプローブ、平面反射、スクリーンスペースリフレクション）を利用できます。ハイエンドなシェーダーとエフェクトでマテリアルを調整したり、パイプラインデバッグツールを使用してレンダラーのトラブルシューティングを行ったりすることが可能です。



プロジェクトの計画時には早期にレンダーパイプラインを選んでおく

HDRP と URP はどちらも以下を備えています。

- シェーダーグラフ：HLSL でコードを記述することなく、ビジュアルノードエディターを使ってシェーダーを制作できるツール。開発者は、シェーダーグラフを利用することで、シェーダー関連の作業の一部をアーティストやテクニカルアーティストに振り分けることができます。
- サンプルシーン：ライティング設定、マテリアル、シェーダーの設定方法の例を示します。

  既成のレンダーパイプラインアセットを複数利用すると、設定がそれぞれのパイプラインに合わせて既に最適化されているグラフィックス品質レベルをすばやく切り替えることができます。
- 最新の Post-Processing Stack：URP と HDRP はそれぞれのパイプラインに合わせて最適化された独自のポストプロセッシングシステムを備えています。そのため、アーティストが使いやすいインターフェースで全画面のフィルターを適用できます。

— ビジュアライゼーションツールを備えたレンダーパイプラインデバッグユーティリティ：ライティング、シェーディング、シャドウなどの問題の解決に利用できます。

URP と HDRP の基盤であるスクリプタブルレンダリングパイプラインは、C# スクリプトを使ってレンダリングコマンドのスケジュールや設定を行うことができる薄い API レイヤーです。パイプラインのほぼすべての面をカスタマイズできる柔軟性を備えています。SRP に基づいた独自のカスタムレンダーパイプラインを作成することもできます。

ビルトインレンダーパイプラインは、URP や HDRP よりもカスタマイズ性は劣りますが、幅広いプラットフォームをサポートしています。ビルトインレンダーパイプラインを使用するには、異なるレンダリングパスを設定し、コマンドバッファとコールバックで機能を拡張する必要があります。

『The Heretic』のショートフィルムに表れている HDRP のハイエンドのグラフィック処理能力

利用可能なパイプラインの詳細な比較については、Unity のレンダーパイプラインを参照してください。

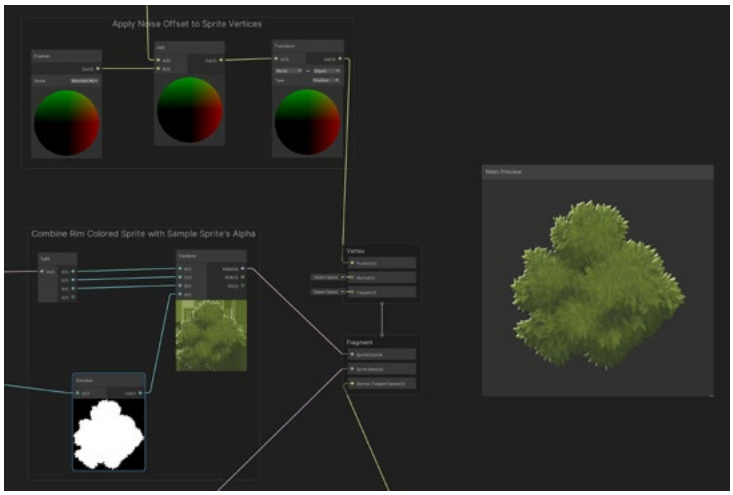
2D パイプライン

URP は、**2D Renderer** も備えています。スプライトベースのゲームを制作する場合のために、2D 体験の制作に役立つよう特別に設計されたツールが用意されています。

— **2D Lights** では、さまざまな形状（フリーフォーム、スプライト、パラメトリック、ポイント、グローバル）を利用できます。2D Lights は、スプライトエディターで補助的なテクスチャとして適用すると、法線マップおよびマスクテクスチャと相互作用できます。これは、高度なライティングエフェクトの作成に役立ちます。

『*Lost Crypt*』サンプルプロジェクトの 2D ライト

— シェーダーグラフには、Lit と Unlit スプライトマスターノードがあります。それにより、ノードの入力が効率化されるため、適切な Sample Texture2D ノードを渡して、2D でレンダリングすることができます。



2D シェーダーグラフ

— **2D Pixel Perfect パッケージ**には、Pixel Perfect Camera コンポーネントが含まれています。このカメラは、ピクセルアートをさまざまな解像度で鮮明な状態に保ち、モーションを安定した状態にします。

まずは、2D サンプルプロジェクトの *Lost Crypt* や *Dragon Crashers* など、利用可能な 2D デモのいずれかを確認することをお勧めします。そうすれば、2D ツールを組み合わせてリアルタイムな体験を実現する方法の糸口がつかめるはずです。

ワールド構築

ゲームには、いくつかのレベルが必要です。そこで役立つのが以下のツールです。

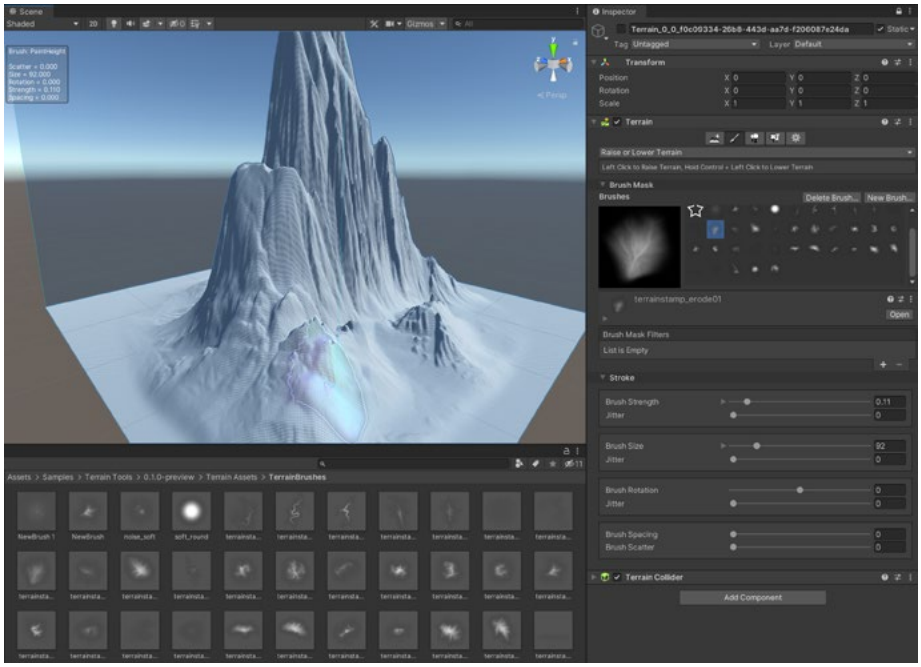
ProBuilder と Polybrush

Unity には、合理化された 3D モデリングとレベルデザインツールである ProBuilder というアドオンパッケージが付属しています。ProBuilder を使用すると、構造物のプロトタイプ作成を短時間で行うことができます。また衝突ジオメトリやトリガーゾーン、ナビメッシュのカスタム作成も可能です。ProBuilder は、シンプルなジオメトリの作成やゲームレベルのグレーボクシングに最適化されています。

Polybrush を使うと、メッシュのスカルプト、テクスチャのブレンド、頂点色のペイント、レベルへのオブジェクトの散布が可能になります。ProBuilder と組み合わせることで、Polybrush は包括的なレベルデザインツールスイートとなります。お好みの DCC ツール（Maya、Blender など）を使ってラウンドトリップして、モデルをさらに調整することができます。

Terrain ツール

3D の複雑な自然環境を実現するために、Unity エディターにはビルトインの Terrain ツールが用意されています。Terrain ツールは Unity 2021.2 以降でパッケージとして提供されており、ゲームにランドスケープを追加できます。エディターでは、複数のテレインタイルの作成、ブラシベースのツールを使ったランドスケープの高さや外観の調整、ランドスケープへの草木の追加が可能です。

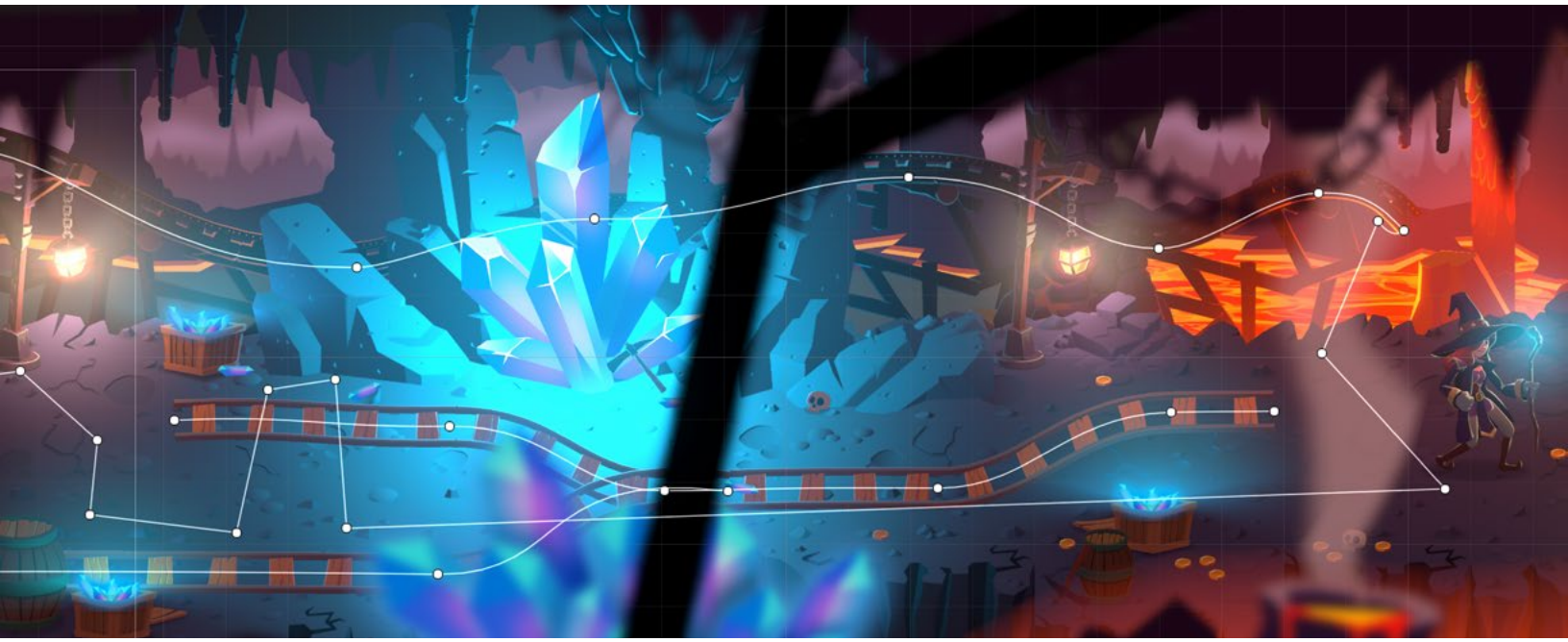


Unity Terrain tools

2D タイルマップと Sprite Shape

2D ゲームを制作する場合は、タイルマップを使用して、六角形タイルや等角タイルで構成された巨大なグリッドベースの世界を作成します。タイルマップとは、2D のステージを作成するためにタイルアセットを格納して処理するシステムです。タイルパレットを設定すれば、さまざまなブラシを使用して 2D タイルマップをペイントすることができます。

Sprite Shape では、視覚的で直観的なワークフローを使用して、豊かな自由形状と有機的な 2D 環境を作成できます。このツールでは、設定した角度範囲に基づいて、スプラインパスに沿ってスプライトを動的にタイル表示することができます。スプラインに沿ってタイル表示されたスプライトは自動的に変形させることができ、アウトラインの角度に基づいてスプライトを切り替えることもできます。



SpriteShape ツールで有機的な 2D 環境を制作できる

UNITY ASSET STORE

Unity Asset Store では、テクスチャ、アニメーション、モデル、プロジェクト全体のサンプル、チュートリアル、エディター拡張ツールなどのさまざまなアセットを入手できます。Unity Technologies およびそのコミュニティメンバーが作成したアセットが活発に公開されているマーケットプレイスです。パッケージマネージャーがあれば、アセットパッケージを Unity プロジェクトに直接ダウンロードできます。

Asset Store にオンラインでアクセスするには、お持ちの Unity アカウントにログインする必要があります。2D、3D、スクリプトなど、バラエティ豊かなアセットをご覧ください。

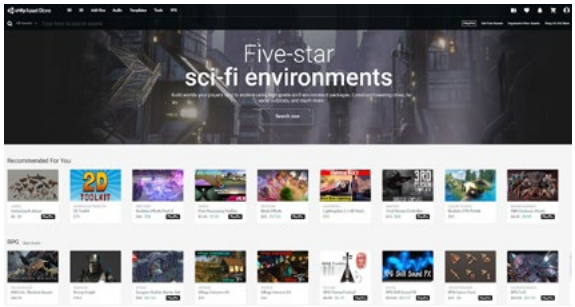
- **「テンプレート」**セクションでは、さまざまなチュートリアルやスターターパックをダウンロードできます。
- **「オーディオ」**では、サウンドファイルのライブラリを入手できます。プロジェクトになじむ環境音、音楽、ボイス、効果音を取り入れてユーザーエクスペリエンスを豊かにしましょう。
- **「2D」**と**「3D」**のセクションでは、環境、小道具、キャラクターのアセットが豊富に提供されています。さまざまなアートスタイルやジャンルのアセットでステージが充実します。
- **「ツールとエディター」**のページは、開発時間に余裕がないときに便利です。
- AI やビジュアルスクリプティングを通じた機能強化も検討できます。

アセットのインポート

無料または有料のアセットを入手すると、お持ちの Unity アカウントに表示されます。パッケージマネージャー（**「Window」>「Package Manager」**）から**「My Assets」**で検索することで、アカウントに関連付けられたアセットを現在のプロジェクトにダウンロードまたはインポートできます。

パブリッシャーになる方法

Asset Store でパブリッシャーになることで、制作物（3D モデル、エディターの拡張機能、オーディオなど）を販売し、ゲーム開発中の新たな収入源とすることができます。開始する際は、Asset Store パブリッシャーアカウントを作成して、販売審査ガイドラインを確認し、Asset Store プロバイダー契約を遵守してください。これで「Sell Assets」ページでコンテンツを販売できるようになります。



Asset Store のサードパーティ製アセットのマーケットプレイス

学習リソース

Unity の詳細なドキュメントと学習リソースを活用して、開発スキルを深く掘り下げることができます。

ドキュメント

Unity のことを最も包括的に把握できるのは、Unity マニュアルとスクリプティング API です。これらのマニュアルは事実上すべての機能とツールを網羅しており、継続的に更新もされています。

パッケージにはそれぞれのドキュメントを集めたマイクロサイトが存在します。最新バージョンを確認するには、Unity マニュアルからアクセスするか、「Package Manager」ウィンドウから使用しているバージョンのパッケージのドキュメントにアクセスしてください。

Unity Learn

Unity Learn では、750 時間以上にも及ぶオンデマンドやライブの学習リソースを利用できます。Unity エキスパートや仲間のクリエイターにライブでアクセスし、質問をしたり、ヒントを得たり、実際のプロジェクトに一緒に取り組みましょう。

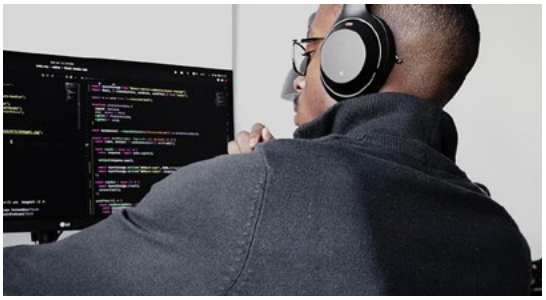
Unity ブログ

Unity ブログには、旬のニュース、最新情報、ソフトウェアリリースに関する記事に加え、最新鋭テクノロジーや、コミュニティイベント、ゲームクリエイターとの対談などを取り上げた短い読み物記事が掲載されます。アーティスト、プログラマー、テクニカルアーティスト、デザイナーなど、あらゆる職種でのスキル向上に役立つ、コミュニティ発のヒントやインサイトをご活用ください。

プロフェッショナルトレーニング

Unity プロフェッショナルトレーニングなら、200 時間を超えるオンデマンドなトレーニングコンテンツにアクセスできます。年間サブスクリプションを購入すれば、厳選されたコース、軽量な動画コンテンツ、評価テスト、コース課題にアクセスして、認定を得ることができます。

これらの教材は、Unity の経験豊富なインストラクショナルデザイナーが Unity のエンジニアや製品チームと協力して制作しています。これは、チームが常に Unity の最新テクノロジーに関する最新のトレーニングを受けることができることを意味します。開始するにはコースのカタログをご覧ください。



Unity プロフェッショナルトレーニング

次のステップ

このガイドでは、Unity に移行することのメリットをお伝えしました。ここでさらに Unity チームからのサポートやガイダンスを受けていただくと、移行がより簡単になります。どんな熟練した登山家も、エベレストには 1 人で登れないのと同じです。

その成功を確実にするために、当社には Accelerate Solutions チームが控えています。

さまざまなニーズに応じたソリューションパッケージでコンサルティングから全体開発まで対応し、Unity への移行、新しいプラットフォームへの移植、パフォーマンスの最適化、開発の高速化などを支援します。

サポートを受けずに開発される場合でも、移行に限らないサポートサービスをご用意しています。ゲームの品質を数か月間から数年間でさらに向上させたい場合などにお勧めです。

プロフェッショナルトレーニングは、Unity プラットフォームの完全マスターを目指すトレーニングです。技術的なトピックを深く掘り下げ、ゲームの最適化やワークフローの合理化を実現し、クリエイティビティを新たな高みに引き上げることができます。

Success Plan をご利用いただくと、移行と将来の開発をより確実に行うことができます。Unity のエキスパートが一人ひとりに合わせたガイダンスで丁寧にサポート。Unity コンテンツにも独占的にアクセスできるので、戦略的または技術的な障害を最小限に抑えることができます。

移行を開始し、コラボレーションサービスについての詳しい情報提供をご希望の方は、当社エキスパートとのチャットにご参加ください。頼れるガイドとして、皆様の移行プロセスに加われる日を楽しみにお待ちしております。


Unity®