1-2 Unityについて理解しよう

ゲームについて理解したところで、ここでは本書で学習するUnityについて理解を深めましょう。

1-2-1 Unityとは

Unityは、2005年にMac上で動くゲーム開発プラットフォームとして誕生しました。その理念は「ゲーム開発の民主化」でした。

前述の通り、ゲーム開発には高いハードルが存在していました。おもしろそうなゲームのアイデアが浮かんだのに、作り方がわからず諦めてしまう、それはとても残念なことです。

そのような悩みを解決するために、「もっと多くの人がかんたんにゲームを開発できるようにしたい」という想いから生まれたのがUnityです。Unityはさまざまな機能を追加しながら進化を続け、現在では趣味のミニゲーム開発から商用の本格的なゲーム開発まで、幅広くカバーできる強力な開発プラットフォームとなりました。

図1.1 Unityホームページ

すべての業界に役立つ Unity

また最近ではゲーム開発だけに留まらず、アニメーションや医療、自動車産業など、ゲーム以外の分野でも活用されています。

ゲームについて理解したところで、ここでは本書で学習するUnityについて理解を深めましょう。

1-2-2 Unityにはどんな機能がある?

Unityには、ゲームを開発するために必要なさまざまな機能が搭載されています。

たとえば、重力の影響を自動で計算してくれる物理エンジンや、他のユーザが作った部品(Asset)を購入できるストアなどが用意されており、やり方次第では、プログラムを1行も書かずにゲームを制作することも可能です。

また、マルチプラットフォームに対応しており、Unityで作ったゲームを、専用機やPC、スマホなど、さまざまなプラットフォームで遊べるようにすることができます。

1-2-3 Unityは高くて手に入れられない?

Unityにはさまざまな機能が詰まっており、どのようなゲームでも開発できます。となると心配なのが、「Unityを使うにはどのくらいお金がかかるのか」という点ではないでしょうか。

先ほど述べたようにUnityの理念は「ゲーム開発の民主化」です。そして、料金プランもそれに沿ったものとなっています(**2-1-2**参照)。無料で使えるPersonalプランでもほとんどの機能が利用できますので安心してください。

Tips Unityを使うにあたって心がけておいた方が良いこと

筆者はUnityを使う前からプログラムを書いており、しくみを考えるのは得意な方です。そのため、さまざまな機能を自分で作ろうとしてしまいがちです。

これはこれでとても楽しいのですが、Unityに関しては、既存の機能やAssetをできるだけ活用することをオススメします。理由は既存の機能やAssetがとても強力で、使わないのはもったいないからです。

「有りものを使うなんて味気ない、全部自分で作りたい」と思う人もいるかもしれませんが、開発を続けていると必ずオリジナリティを出したい部分が出てきて、自ら勉強してカスタマイズするようになります。最初はこだわり過ぎず、既存の機能やAssetをフル活用してゲームを作っていきましょう。

2-3 Unityを動かしてみよう

インストールが完了したところで、皆さんはきっとUnityを動かしてみたくてウズウズしているかと思います。ここからはUnityを動かしながら、使い方の基本を学んでいきましょう！

2-3-1 プロジェクトを作成する

プロジェクトとは

Unityでは、ゲームを「プロジェクト」という単位で管理します。プロジェクトは、ゲームのスクリプト、キャラクターの画像、ステージのデータなど、ゲームを必要なファイルや情報で構成されています。

プロジェクトの作成

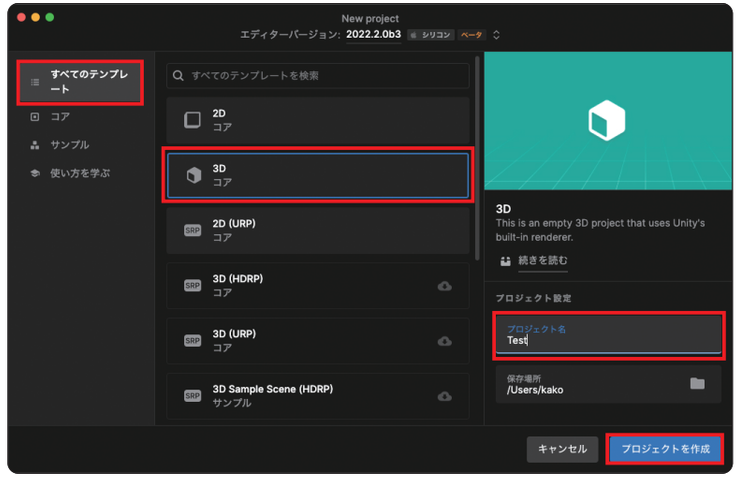
まずはプロジェクトを作成しましょう。

Unity Hubで左上の「プロジェクト」を選択し、「新しいプロジェクト」をクリックすると、プロジェクトの作成画面が開きます。

テンプレートでは、プロジェクトのテンプレート（雛形）を設定します。Unityの基本を学ぶための学習用テンプレートなどもありますが、今回は「すべてのテンプレート」を選択し、その中から「3D」を指定します。なお、テンプレートはプロジェクトの初期設定や初期パッケージが変わる程度で、「3D」を選択しても2Dゲームは作成できます。

プロジェクト名を「Test」に指定し、保存先を指定して「プロジェクトを作成」をクリックします。初回はプラグインのインストールが実行されるため、かなり時間がかかります。気長に待ちましょう。

図2.22 新規プロジェクトの作成



2-3-2 シーン、ゲームオブジェクト、コンポーネント、Asset

プロジェクトを作成すると、Unityエディタが自動的に開いて、SampleSceneというシーンが開かれた状態になります。

プロジェクトは、シーン、ゲームオブジェクト、コンポーネント、Assetなどで構成されています。

シーン（Scene）

シーン（Scene）とは、ゲーム中の場面を表します。たとえば、「タイトル画面シーン」「ステージ1 シーン」「ステージ2 シーン」といった形です。

1つのシーンにゲームのすべての要素を詰め込むことも可能です。ただし、後から変更を加えるのが大変になったり、不要な要素があるせいでゲームプレイ時のパフォーマンスに影響を及ぼすため、シーンは必要に応じて分けるようにしましょう。

ゲームオブジェクト（Game Object）

ゲームオブジェクト（Game Object）とは、ゲームを構成する要素で、シーンの中に配置されます。キャラクター、光源（周囲を照らすライト）、背景画像、UIなど、ゲーム中で登場するものは基本的にゲームオブジェクトとして存在しています。

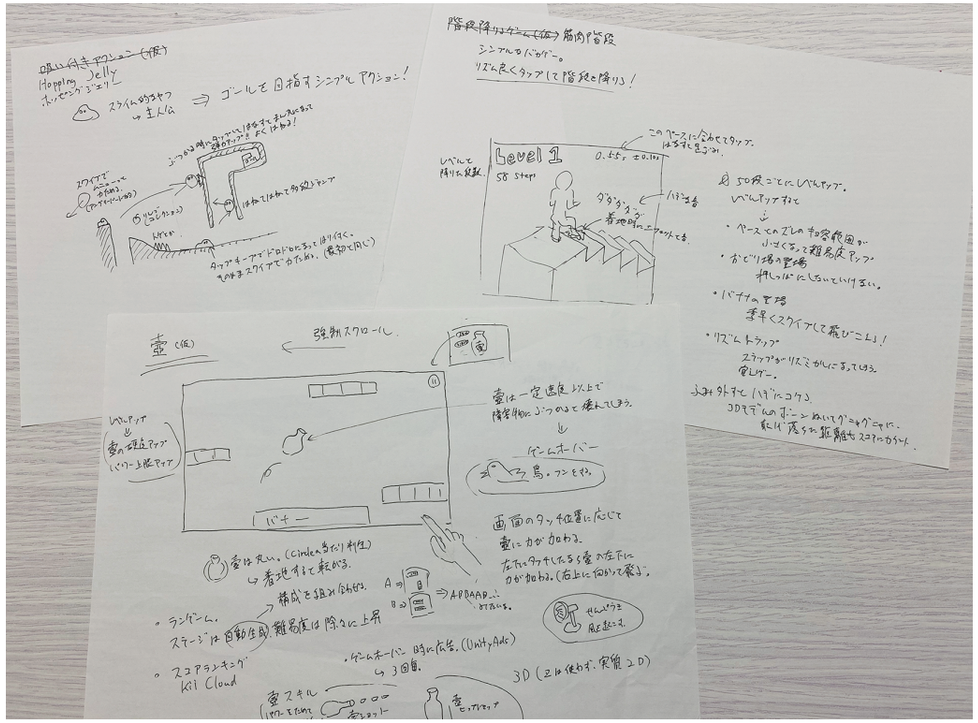
コンポーネント（Component）

ゲームオブジェクトには、コンポーネント（Component）という「ゲームオブジェクトのふ

参考として、以前筆者が書いた企画書を何枚か集めた写真を載せました。

1つのゲームにつき1枚の用紙にまとめ、ゲーム画面のイメージ＋各種ゲーム要素を大雑把に記載しています。自分用として雑に書いたため、他の人に見られるのは少し気が引けますが、こういった紙が1枚あるだけで、作ろうとしているゲームがイメージしやすくなるはずです。

図4.1 手書きの企画書サンプル



また、シンプルなゲームであれば企画書1枚だけでも開発は進められますが、画面遷移図やシステムの仕様書などがあると、さらに開発を進めやすくなりますので、必要に応じて書き足していくと良いでしょう。

4-6 ゲームの開発手順を確認しよう

企画書を作成することで、開発の道しるべはできました。開発前にあらかじめ手順をイメージしておくことで、さらに作業がスムーズに進められるようになります。

4-6-1 ゲームのコア要素を考えてみる

開発手順を考えるにあたって、開発するゲームの中で何が一番大事かを考えてみましょう。たとえば、以下の要素で構成されるゲームがあったとします。

- キャラクター育成機能
- アイテム強化機能
- 横スクロールアクション

一番大事なのは「横スクロールアクション」の要素です。アクション部分はプレイヤーがいちばん触れることが多い要素でゲームのコア部分となります。これがつまらないと、他の要素を作り込んでもゲームの人気は伸びづらいでしょう。

このようにゲームの要素に優先順位を付け、どこに注力すべきかを明確にしておくと作業が進めやすくなります。

4-6-2 プロトタイピング

注力すべき要素が明確になったら、その要素から開発をスタートします。

その他の要素はいったん置いておき、ざっくりゲームを遊べる状態にしてみましょう。このプロセスをプロトタイピングと呼びます。

プロトタイピングにはさまざまなメリットがあります。

たとえば、ゲームを実際に遊べるようになると改善のアイデアがたくさん湧いてきますし、人に遊んでもらって意見を聴くこともできるようになります。また、ゲームに欠陥がある場合（遊んでみたら全然おもしろくなかったり、ルールが破綻しているなど）も早い段階で発見できます。

5-2 地形を追加しよう

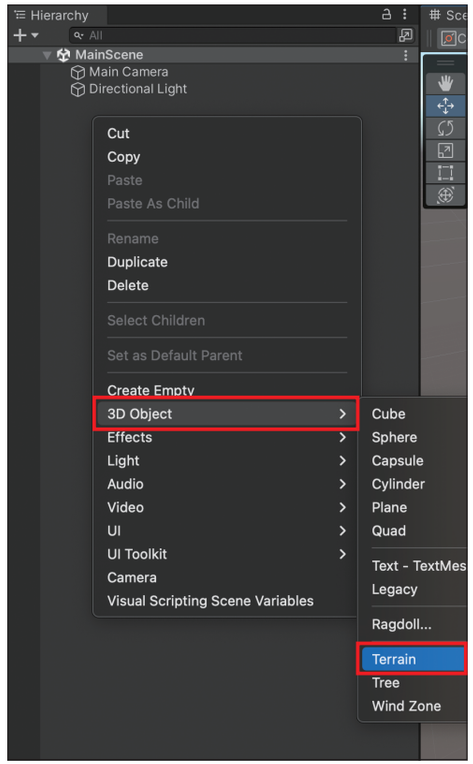
プロジェクトを作成して、Assetを準備したあとは、舞台となる地形を追加していきましょう。

5-2-1 Terrainの作成

5-1-3で4つのAssetのインポートが完了したら、Terrain（テレイン）でゲームの舞台となる地形を作成していきます。

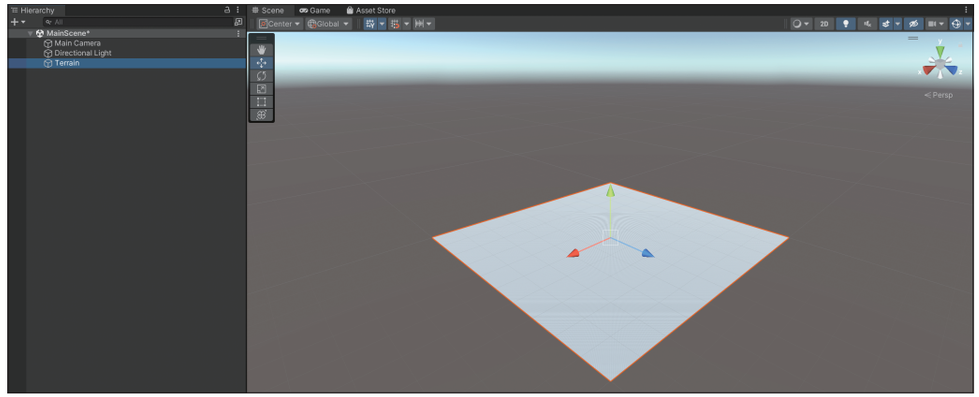
Hierarchyウインドウで右クリックし、「3D Object」→「Terrain」を選択します。

図5.13 Terrainの作成



Sceneビューにチェック柄の板が作成されました。Terrainのデフォルトサイズは1km四方とかなり大きいため、全体像を確認したい場合は、Hierarchyウインドウの「Terrain」をダブルクリックします。

図5.14 Terrainの全体像



5-2-2 Terrainの初期設定

初期状態のTerrainでは、地形の解像度が高い（＝細かく地形が設定できる）設定であるため、そのまま使用するとゲーム実行時の負荷が高くなります。

ハイスペックなPCであれば問題ありませんが、ここでは負荷を抑えるための設定を行います。地形の解像度や詳細の描画距離など、Terrain設定の変更によって負荷を低くすることができます。

解像度を下げて負荷を抑える

解像度などを下げて負荷を抑えるには、Hierarchyウインドウで「Terrain」を選択し、Inspectorウインドウで Terrainコンポーネントの一番右の（Terrain Settings）をクリックし、設定を変更します（表5.1）。

表5.1 Terrain Settingsの設定項目

設定項目	説明	設定値
Pixel Error	マッピング精度を設定する。この値を大きくすると精度は低くなる。精度を低くすると、遠くの地形は雑に描画されるようになる	100
Detail Density Scale	Terrainに配置される草などの密度に影響する。低いと草がまばらに表示され、高いとギッチリ詰まって表示される	0.1
Detail Scatter Mode	配置モード。「Coverage Mode」よりも「Instance Count Mode」の方が草の配置密度が濃くなる	Instance Count Mode
Heightmap Resolution	ハイトマップの解像度。Terrainでどれだけ細かく地形を変更できるかを設定する	257×257

6-3 カメラがキャラクターを追いかけるようにしよう

次はキャラクターの動きに合わせてカメラが動くようにします。カメラを動かすスクリプトを自分で書く方法もありますが、Cinemachineを利用すれば、カメラをかんたんに制御できます。

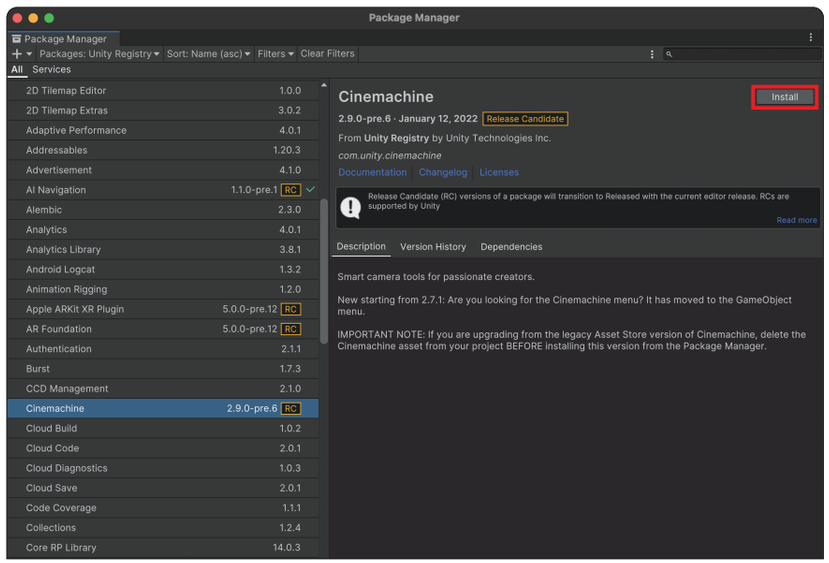
6-3-1 Cinemachineのインポート

Unity公式パッケージのCinemachineを使うと、カメラでキャラクターを追いかけたり、複数のカメラを自動で切り替えるなど、カメラに対してさまざまな制御が行うことが可能です。

Cinemachineをインポートするには、「Window」→「Package Manager」を選択してPackage Managerを開き、ウインドウ左側にあるプルダウンで「Unity Registory」を選択します。

Unity公式パッケージの一覧が表示されますので、「Cinemachine」を選択して「Install」ボタンをクリックします。

図6.13 Cinemachineのインストール



6-3-2 キャラクターを追尾するカメラを配置する

Cinemachineをインポートすると、GameObjectの作成メニューにCinemachineの項目が追加され、CinemaChineで制御する各種カメラを作成可能になります。

Hierarchyウインドウで右クリックし、「Cinemachine」→「Virtual Camera」を選択してカメラを作成しましょう。

Hierarchyウインドウに「Virtual Camera」という名前のオブジェクトが生成されていますので、これを選択します。

図6.14 Virtual Cameraの作成

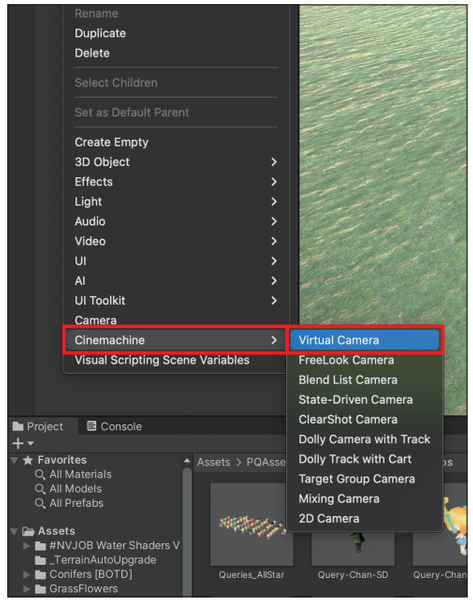
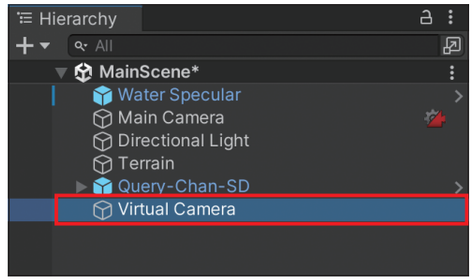


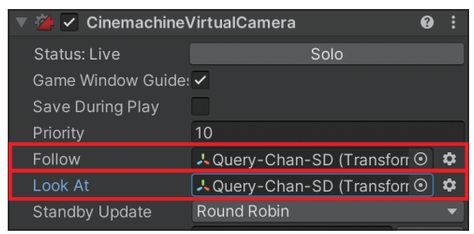
図6.15 生成されたVirtual Camera



Inspectorウインドウからキャラクターを追尾するための設定を行います。

CinemachineVirtualCameraコンポーネントのFollowに「追跡対象のゲームオブジェクト」を、Look Atに「注目対象のゲームオブジェクト」を指定します。今回はどちらも「Query-Chan-SD」としたいので、Hierarchyウインドウから「Query-Chan-SD」をドラッグ＆ドロップしてください。

図6.16 CinemachineVirtualCameraコンポーネントの設定（その1）



7-4 敵キャラクターに攻撃させてみよう

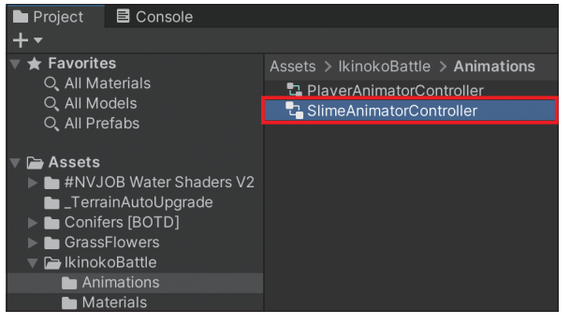
次は敵キャラクターが攻撃してくるようにしてみましょう。

7-4-1 アニメーションの設定

今回使用しているスライムの3Dモデルには、攻撃や被ダメージ時ののけぞりなど、各種アニメーションが同梱されています。Animator Controllerを作成して、スクリプトからアニメーションを制御できるようにしましょう。

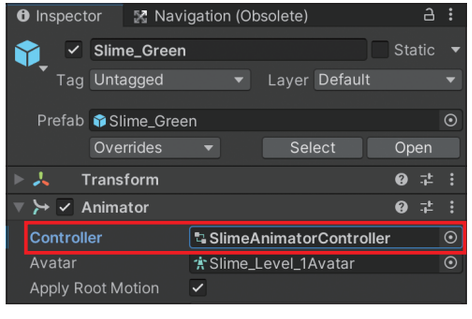
Projectウインドウの「IkinokoBattle」ー「Animations」フォルダを右クリックし、「Create」→「Animator Controller」を選択して、Animator Controllerを作成します。名前は「SlimeAnimatorController」とします。

図7.21 Animator Controllerの作成



Hierarchyウインドウで「Slime_Green」を選択し、Inspectorウインドウの Animatorコンポーネントにある Controllerに、作成した「SlimeAnimatorController」をドロップします。

図7.22 Animationコンポーネントの設定



クエリちゃんのときと同様の手順（6-6-3参照）で、SlimeAnimatorControllerに「Idle」と「Move」のEntityを作成し、Transitionでつなぎます。「Idle」と「Move」間のConditionsも、クエリちゃんと同じくMoveSpeedを使って設定しておきましょう。

IdleのInspectorウインドウのMotionを「slime_idle」に変更し、MoveのMotionを「slime_move」に変更します。Transitionでつなぐ部分については、6-6-3と同様に設定してください。

図7.23 Idle Entityの設定

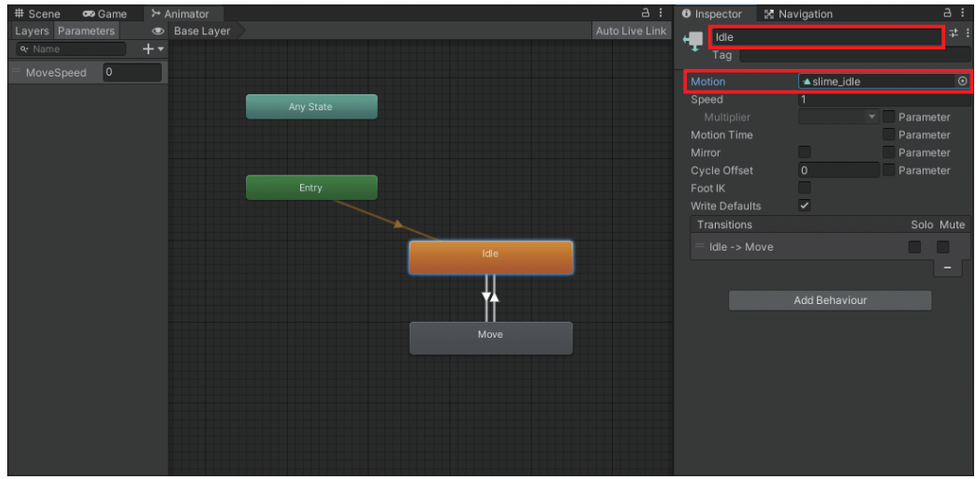
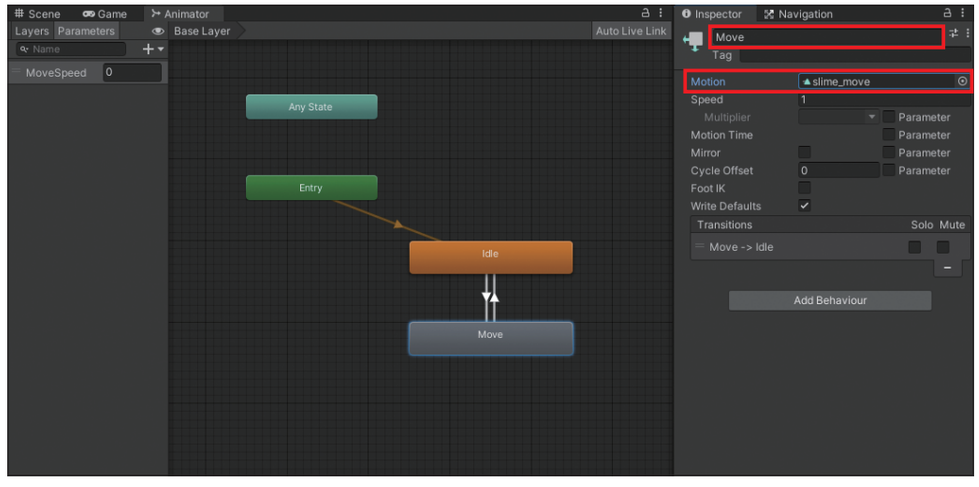


図7.24 Move Entityの設定



8-2 ゲームオーバー画面を作ろう

続いて、少し演出を入れつつゲームオーバー画面を作ってみましょう。

8-2-1 ゲームオーバー画面シーンの作成

ここから、ゲームオーバー画面のシーンを作成していきます。

「File」→「New Scene」を選択して（ショートカットは Command ＋ N）で新規シーンを作成し、名前を「GameOverScene」として保存します。

次にHierarchyウインドウで右クリックし、「UI」→「Panel」を選択してパネルを作成します。

Canvasの設定も必要となりますが、8-1-2～8-1-3と同じ流れですので、参照して設定を行ってください。

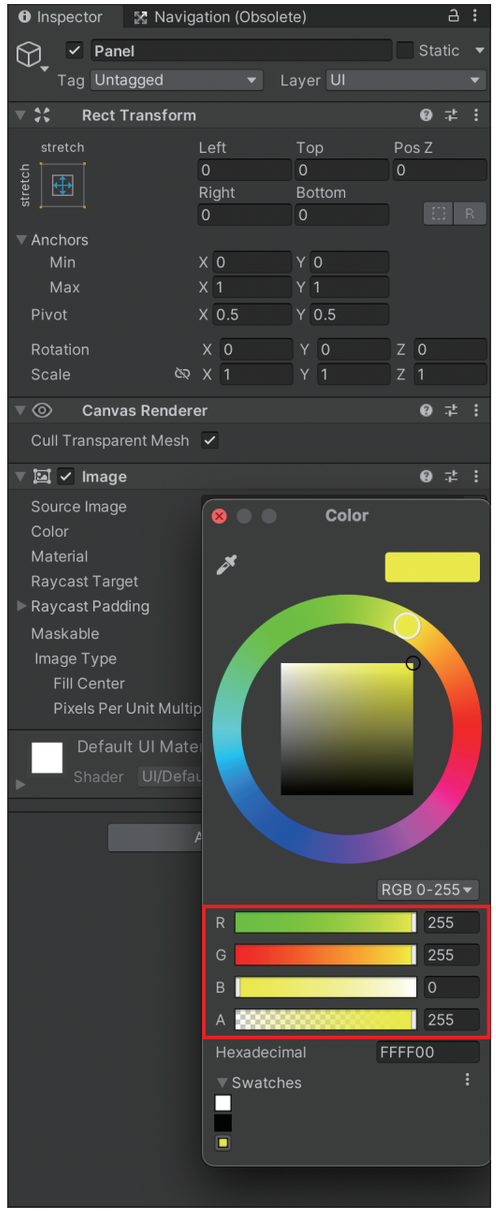
8-2-2 UIに影を付ける

UIや2DのSpriteには、コンポーネントで影を付けることができます。デザインのアクセントになったり、同系色の背景に溶け込みづらくすることもできますので、試してみましょう。

影をわかりやすくするため、Panelの背景色を黄色に変更しておきます。

Hierarchyウインドウで「Canvas」ー「Panel」を選択し、Inspectorウインドウの Imageコンポーネントの Colorを変更します。プルダウンで「RGB 0-255」を選択し、Rを「255」、Gを「255」、Bを「0」、Aを「255」に設定することで、不透明の黄色を指定できます。

図8.24 Panelの色を設定



次にHierarchyウインドウの「Canvas」ー「Panel」で右クリックし、「UI」→「Legacy」→「Text」でTextを作成します。名前は「GameOver」にしておきましょう。

10-1 パフォーマンスを改善しよう

ゲームの機能を実装したあとは、動作チェックが必要不可欠です。きちんと動かない部分はその都度直していくとして、ゲームの負荷が高すぎて画面がカクカクしてしまう場合は、パフォーマンスの調整が必要です。

10-1-1 フレームレートを設定する

フレームレート（FPS、Frames Per Second）とは画面が1秒間に更新される回数のことです。アクションゲームなどの動きが多いゲームで、動きを滑らかに見せたい場合は、フレームレートを高く設定しておく必要があります。

フレームレートの設定を行うには、「Edit」→「Project Settings...」を選択し、ProjectSettingsウインドウで「Quality」を選択します。

画質品質の設定

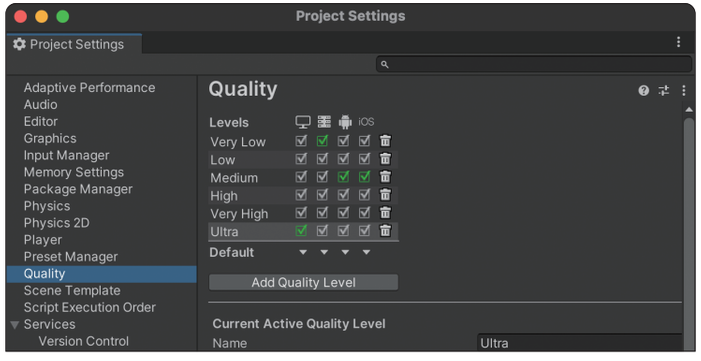
Qualityでは、AndroidやiOSなど任意のプラットフォーム画質設定を行います。チェックボックスが緑色になっているのがDefaultの設定です。

Texture Quality（テクスチャの品質）やAnti Aliasing（アンチエイリアスのかけ方）など、パフォーマンスに大きく影響する設定がたくさん用意されています。必要に応じて調整しましょう。

各パラメータの詳細は公式マニュアル（https://docs.unity3d.com/ja/current/Manual/class-QualitySettings.html）を参照してください。

Default右側にある▼でゲームに適用される設定を切り替えることが可能で、それだけでもパフォーマンスが大きく変わります。

図10.1 画質品質の設定



垂直同期の設定

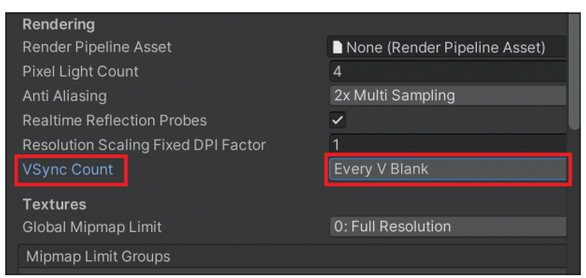
Qualityを下にスクロールすると、VSync Countの項目があります。ここでは、垂直同期を設定します。垂直同期とは、ディスプレイのリフレッシュレートとフレームレートを同期することで、この設定によって画面の描画が安定させることができます。

VSync Countに設定できる値は、表10.1の3種類です。

表10.1 垂直同期の設定（VSync Count）

設定	説明
Don't Sync	垂直同期を行わず、フレームレートは可能な限り高くなる。スクリプトで任意のフレームレートを指定する場合はこれに設定する（詳細は後述）
Every V Blank	垂直同期を行う。ディスプレイのリフレッシュレートが60Hzである場合は、フレームレートも60になる。ちなみに、一般的なディスプレイはリフレッシュレートが60Hzのものが多い
Every Second V Blank	垂直同期を半分の周期で行う。ディスプレイのリフレッシュレートが60Hzである場合は、フレームレートは30になる。動きの滑らかさは落ちるが、負荷を抑えたい場合に設定する

図10.2 垂直同期の設定



12-1 ビジュアルスクリプティングについて知ろう

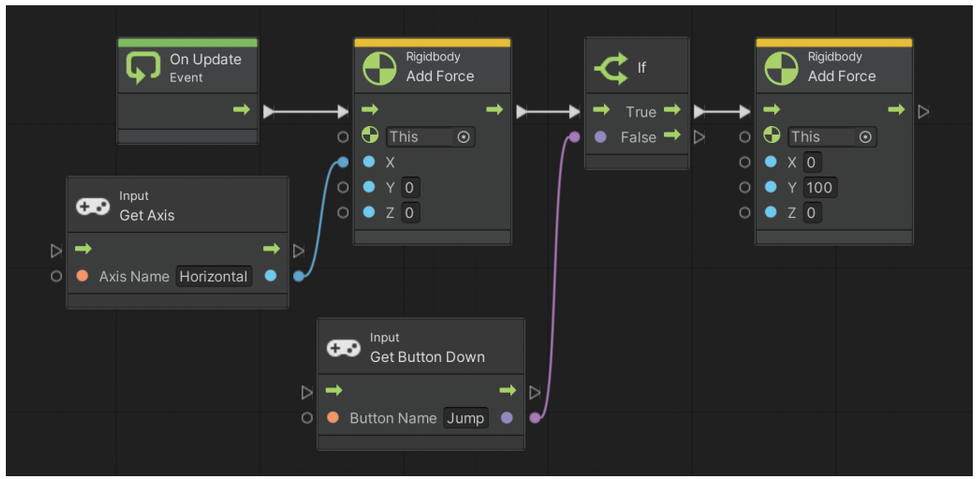
ここでは、ビジュアルスクリプティングとは何かについて解説し、導入手順や基本知識についてまとめています。

12-1-1 ビジュアルスクリプティングとは

これまでUnityでスクリプトを記述するには、C#でプログラミングするのが基本でした。これはプログラミングが苦手な方からすると高いハードルを感じてしまうものでした。

ビジュアルスクリプティングでは、プログラムを書く代わりに画面上で部品を配置していき、それを繋ぎ合わせてスクリプトを作成します。目に見える形で組み立てていくため、プログラミングの知識が浅くても、処理の流れが把握しやすいという特徴があります。

図12.1 ビジュアルスクリプティングの例



Unityのビジュアルスクリプティング（https://unity.com/ja/products/unity-visual-scripting）は、元々Asset Storeで販売されていた「Bolt」という有料のビジュアルスクリプティングアセットでした。2020年5月にUnityによって買収され、現在は誰でも無料で利用可能になっています。

なお、ビジュアルスクリプティングを利用する場合でも、プログラムの基礎知識（変数やif文など）およびUnityの基礎知識（ライフサイクルやコンポーネントなど）は必要です。本書でもChapter 3で一通り解説していますので、プログラミング未経験者の方は、あらかじめ目を通しておくようにしてください。

12-1-2 ビジュアルスクリプティングの導入方法

ビジュアルスクリプティングは、Unity 2021以降ではデフォルトで組み込まれているため、特別な設定を行わなくても利用可能です。

それ以前のバージョンを利用している方は、Asset Store（https://assetstore.unity.com/packages/tools/visual-scripting/bolt-163802?locale=ja-JP）からダウンロード・インストールを行ってください（5-1-3参照）。

図12.2 Asset StoreのBoltページ

13-3 処理が実行されない原因を探ろう

ログ出力やステップ実行を仕込んで情報を見える化を実現しても、処理が実行されなければ何もわかりません。このような場合は、処理の呼び出し元をたどったり、Unityのイベント関数や設定が正しいかどうかを確認してみましょう。

13-3-1 関数の呼び出し元をたどっていく

処理が実行されないということは、その関数の呼び出し元に原因があります。

呼び出し元を特定するには、Visual Studioで関数の上で右クリックし、「すべての参照を検索」を選択します。

関数を使っている個所が一覧で表示されますので、これを繰り返して関数の呼び出し元をたどっていき、呼び出し元にブレークポイントを仕込んでみましょう。

もしブレークポイントで一時停止した場合は、問題はほぼ解決したようなものです。あとはステップ実行を進めていけば、何が原因で処理が呼ばれないのかを特定できます。

13-3-2 イベント関数の記述ミスを疑う

関数の呼び出し元をたどってもダメだった場合は、別の原因を疑ってみましょう。

たとえば、Unityのイベント関数は、大文字小文字が間違っているだけで呼ばれなくなります。イベント関数の記述間違いではエラーが発生しないため、気付きづらく厄介なトラブルです。

イベント関数を記述する際は、関数名を手打ちせずにIDEの補完機能を使うなどして、間違える可能性を減らしておきましょう。

また、各IDEではUnityのイベント関数に目印が付きます。たとえば、Visual Studioでは、関数の上部に「Unity メッセージ」と表示されます。イベント関数が呼ばれない場合はチェックしてみましょう。

図13.8 Visual Studioでイベント関数に表示されるメッセージ

```
Unity メッセージ|0 個の参照
private void OnCollisionEnter(Collision collision)
{
    ...
}
```


13-3-3 スクリプトがアタッチされているかチェックする

当然ですが、ゲームオブジェクトにスクリプトがアタッチされていなかったり、ゲームオブジェクトが非アクティブだったりするとイベント関数は呼ばれません。意外とやってしまうので、改めて確認してみましょう。

13-3-4 衝突判定ではトラブルが起こりがち

Unityの衝突判定は、慣れないうちはよくトラブルが発生します。以下に衝突判定のイベント関数が呼ばれない場合のチェックポイントを挙げましたので、参考にしてみてください。

Rigidbody・Collider・イベント関数の組み合わせが正しいかチェック

Rigidbody・Collider・衝突を検知するイベント関数は、2D用と3D用で分かれています。Rigidbody2DとSphereColliderのように、間違った組み合わせだと正しく動きません。組み合わせが正しいか、チェックしてみましょう。

ColloderのIs Triggerとメソッドの組み合わせが正しいかチェック

ColliderのIs TriggerがOFFであればOnCollisionが付く関数、ONであればOnTrigger が付く関数が呼ばれます。組み合わせが間違っていると動きませんので、気を付けましょう。

レイヤーが正しいかチェック

ゲームオブジェクトは、所属するレイヤーを設定することが可能です。物体が衝突するかどうかはレイヤーの設定で決まるため、レイヤーが意図したものになっているか確認しましょう。

併せて、どのレイヤーとどのレイヤーが衝突する設定になっているかチェックしましょう。詳しくは、7-4-3の衝突するレイヤーの設定を参照してください。

Colliderの範囲が合っているかチェック

Sceneビューで対象のゲームオブジェクトを選択し、Colliderの範囲（緑色の線）が正しく表示されているかチェックしましょう。

RigidbodyまたはRigidbody2Dがアタッチされているかチェック

衝突判定を発生させるためには、ぶつかるゲームオブジェクトの少なくとも片方にRigidbody（2Dの場合はRigidbody2D）をアタッチする必要があります。

ゲームオブジェクトを物理演算ではなくスクリプトで動かしている場合は、気付きづらいので、注意しましょう。