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の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

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。

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。

2-1 macOSにUnityをインストールしよう

ここでは、macOSにUnity HubおよびUnity本体をインストールする手順を説明していきます。

2-1-1 Unity Hubのインストール

本書ではUnity Hubを利用します。Unity HubはUnity本体ではなく、Unityのバージョンやプロジェクトを管理するためのツールです。

複数のゲームを開発している場合、ゲームごとに利用するUnityのバージョンが異なることがよくあります。またすでにゲームをリリースしていると、あとからUnityバージョンを変更するのが難しくなります。

Unity Hubを使用すると、複数のUnityバージョンの管理が非常に楽になりますので、とても重宝するツールです。

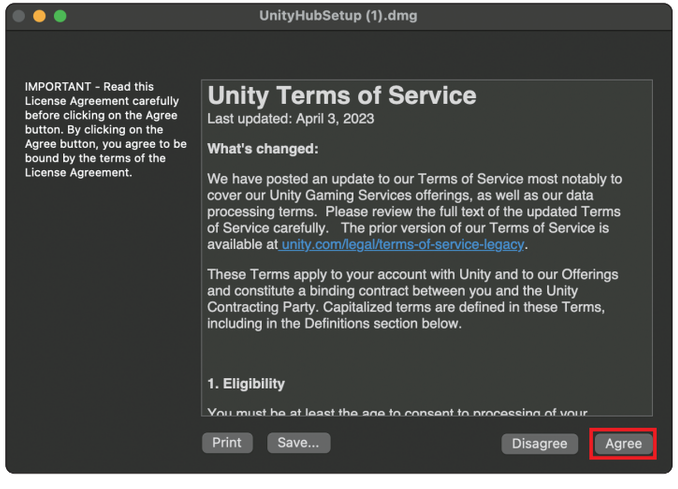
それでは、Unity Hubをダウンロード・インストールしていきましょう。

まずUnityダウンロードページ（https://unity.com/ja/download）にアクセスし、表示されたページの少し下にある「Mac用ダウンロード」をクリックします。Webサイトのダウンロード許可を聞かれた場合は、「許可」を選択してください。

図2.1 Unity Hubのダウンロード

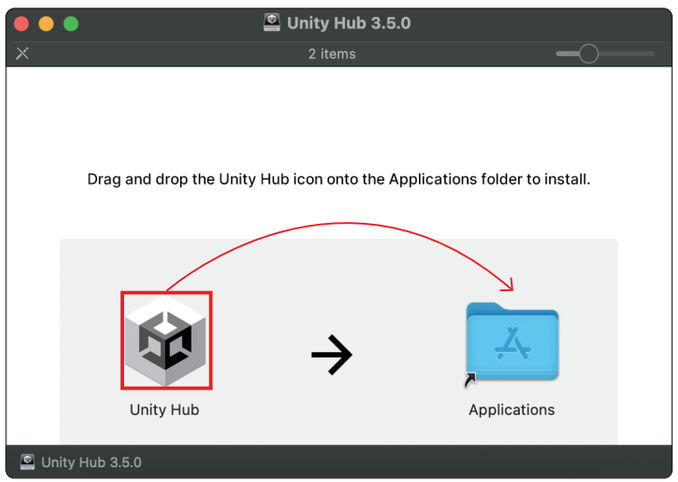
ダウンロードが完了したら、「UnityHubSetup.dmg」をダブルクリックします。英語で記述された利用規約が表示されますので、「Agree」をクリックします。

図2.2 Unity Hub利用規約



左側のUnity Hubアイコンを右側の「Applications」にドラッグします。Applicationsをダブルクリックし、中に「Unity Hub」があればインストールは完了です。

図2.3 Unity Hubのインストール



3-2 データの扱い方について学ぼう

スクリプトでは、キャラクター名、レベル、制限時間、獲得したアイテムなど、ゲームの内容に応じてさまざまなデータを扱います。まずはどのようにデータを扱うのか学んでいきましょう。

3-2-1 変数

変数とはデータの入れ物のことです。入れ物ですので、特殊な場合を除いて中身（値）はいつでも変更できます。変数には整数型・小数型など、さまざまな型が準備されており、変数を作る際は「どの型の変数なのか」を宣言します。

C#では、変数には型に一致する値のみ入れることが可能です。たとえば、整数型の変数に入れられるのは整数のみで、文字列を入れようとするとエラーになります。

以下にUnityでよく使う型を紹介します。

bool型

bool型は、true（真）またはfalse（偽）の2種類の値を格納できる、いちばんシンプルな型です。

```
bool value = true;
```

int型

int型は、-2147483648～2147483647の範囲の整数を格納できます。この範囲を超える整数を扱う場合はlong型を使います。

```
int value = 128;
```

float型

float型は、小数を格納できます。C#では、数値の末尾に「f」を付けることでfloat型であることを表します。桁数が多いと誤差が出る（値がほんの少しだけ意図せず変わってしまう）のが特徴で、誤差なく扱える有効桁数は6～9桁程度です。

Unityは座標などの各所にfloat型を使っています。そのため、しばしば誤差が発生して、Inspectorウインドウに表示されている値が変わることがあります。

```
float value = 0.25f;   float型は、数値末尾にfを付ける
float value2 = 5f;   小数点以下が無くてもOK
```

string型

string型は、キャラクター名やメッセージなど、任意の文字列を格納できます。C#では、文字列を「""」で囲います。

```
string value = "ABCあいう123";
```

配列型

配列型は、同じ型の複数の値を1つの固まりとして扱うための型です。配列に入れたデータ（要素）には「0」から順番に番号（インデックスと呼びます）が振られ、インデックスを使って値にアクセスし、データを読み取ったり変更したりできます。

要素の数を増やしたい場合は、新しい配列を丸ごと作り直して上書きする必要があります。

```
int[] values = new int[] {1, 2, 3};   「1, 2, 3」という3つのint型データが入った配列を作成する
Debug.Log(values.Length);   配列の要素が何個あるかをカウントする。3個入っているので、「3」が出力される
Debug.Log(values[0]);   0番目の要素を取得する。「1」が出力される
values[0] = 100;   0番目の要素を「100」に書き換える
Debug.Log(values[0]);   「100」が出力される
```

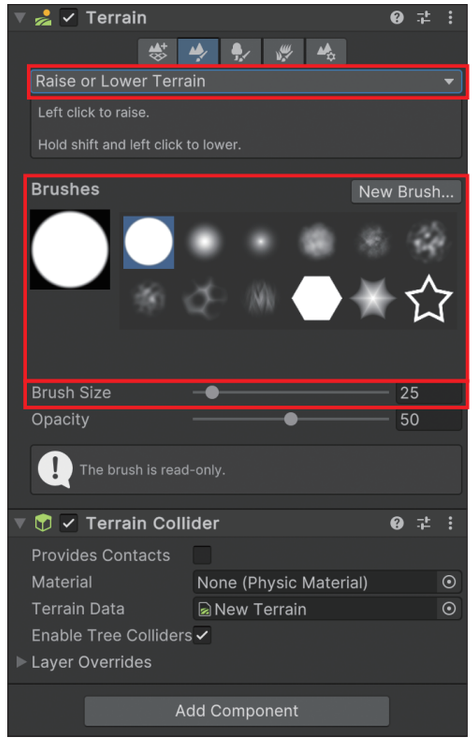
List型

List型は、配列型と似ていますが、要素の数を後から増やすことができます。

```
List<int> values = new List<int> {1, 2, 3};
1, 2, 3という3つのint型データが入ったListを作成する
Debug.Log(values.Count);   Listの要素が何個あるかをカウントする。3個入っているので、「3」が出力される
Debug.Log(values[0]);   0番目の要素を取得する。「1」が出力される
values[0] = 100;   0番目の要素を「100」に書き換える
Debug.Log(values[0]);   「100」が出力される
values.Add(999);   要素「999」を末尾に追加する
Debug.Log(values[values.Count - 1]);   「999」が出力される
```

Brushesから「好きな形のブラシ」を選択し、Brush Sizeで「ブラシのサイズ」を選択しましょう。Opacityはブラシの透明度で、塗ったときの起伏の変化度合いに影響します（100に近いほど地形が大きく変化します）。

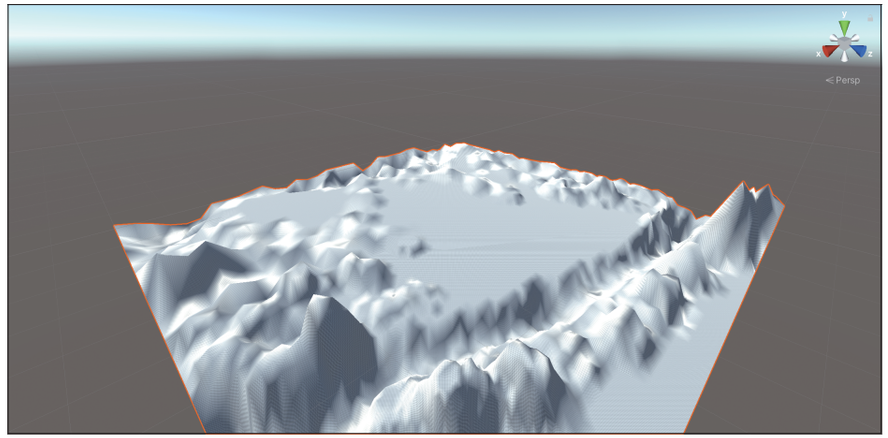
図5.18 地面を上げ下げする



ブラシの準備ができたら起伏を付けていきます。SceneビューのTerrain上で盛り上げたい部分を左クリックすると、地形が盛り上がります。掘り下げたいときは、Shift＋左クリックでOKです。

ドラッグすると連続して上げ下げが可能になります。Terrainの中央付近は後からプレイヤーキャラなどを配置しますので平地のまま残して、あとはご自身の好きなように地形を作成してみましょう。

図5.19 起伏を付けてみる



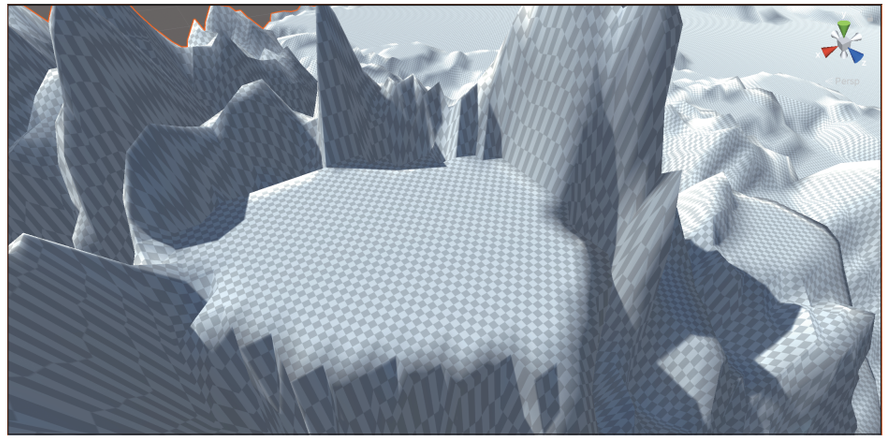
5-2-4 地面の高さを合わせる

Unityでは、高さの調整に使用できる機能が他にも用意されています。

Inspectorウインドウの Terrainコンポーネントの左から2番目の（Paint Terrain）のプルダウンで「Set Height」を選択します（5-2-2「地面を上げておく」と同じ）。

このツールは、Heightに基準となる高さを指定して使用します。盆地や台地を作成する際にも必須のツールとなります。

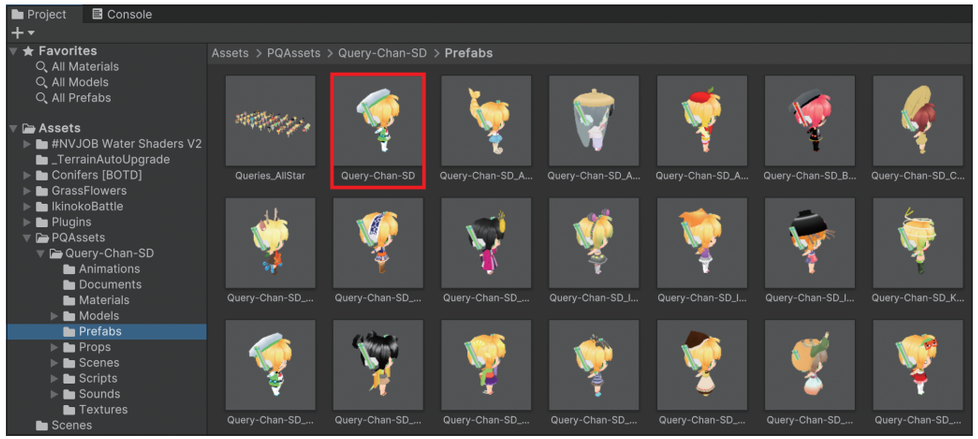
図5.20 Set Heightツールで斜面の一部を平坦にした図



6-1-3 Prefabを配置する

クエリちゃんの3DモデルのPrefabは、Projectウインドウの「Assets」－「PQAssets」－「Query-Chan-SD」－「Prefabs」フォルダに入っています（47都道府県にちなんだクエリちゃんも用意されています）。

図6.6 クエリちゃんのPrefab



ここでは、ノーマルのQuery-Chan-SDをSceneビューの任意の場所に配置します。

図6.7 クエリちゃんをSceneに配置

6-1-4 Shaderを変更して影を付ける

この状態でクエリちゃんの近くにカメラを配置してゲームを再生してみると、少し変なところがあることに気づくかもしれません。よく見てみると、クエリちゃんの影が表示されていません。

図6.8 クエリちゃんに影がない

これは、キャラクターの描画に影が表示されないShader（11-5-2参照）が使用されているのが原因です。影が無い代わりに負荷が低く色も鮮やかに表示されるなど、このShaderを使うメリットもありますが、今回はShaderを変更して影を付けてみましょう。

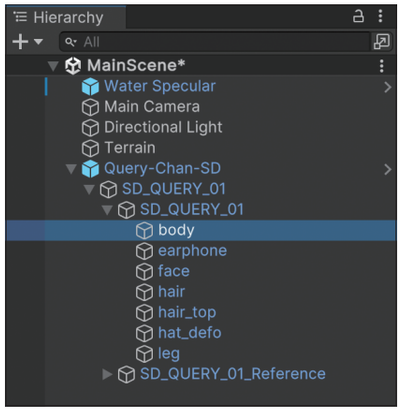
Shaderの変更個所

まずShaderを設定する場所を探します。クエリちゃんは可動部を含むため、ゲームオブジェクトが複雑に組み合わさっています。その中にある3DモデルとRendererを探します。

Hierarchyウインドウで Query-Chan-SDを展開すると、Query-Chan-SDの子オブジェクトが一覧表示されます。

その中にあるSD_QUERY_01、さらにその下にあるSD_QUERY_01を開くと、bodyやearphoneなどの子オブジェクトが確認できます。

図6.9 クエリちゃんの子オブジェクトと孫オブジェクト



7-2 一定範囲に入ると襲ってくるようにしよう

現時点ではプレイヤーと敵キャラクターがどれだけ離れていても、敵がプレイヤーを追いかける状態になっています。敵キャラクターの一定範囲内にプレイヤーが入った場合のみ、敵キャラクターがプレイヤーを追いかけるようにしてみましょう。

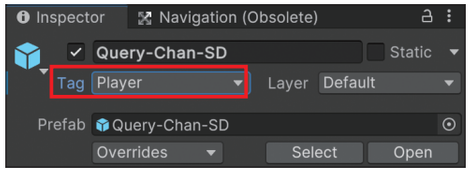
7-2-1 オブジェクトにタグを付ける

ゲームオブジェクトにタグを付けることで、スクリプトから「そのゲームオブジェクトがどのようなものであるか」を判定しやすくなります。今回はプレイヤーキャラクターかどうかの判定を行うために、Query-Chan-SDにPlayerタグを付けてみましょう。

Hierarchyウインドウで「Query-Chan-SD」を選択し、Inspectorウインドウ上部にある「Tag」のプルダウンをクリックします。

Tagはデフォルトの状態でいくつか準備されていますので、今回はこの中から「Player」を選択します。なお、プルダウン最下部の「Add Tags...」から新しいTagを追加することもできます。

図7.11 Tagの設定

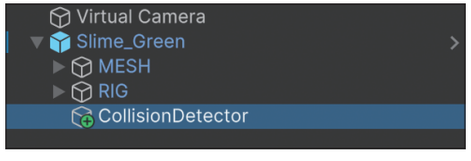


7-2-2 検知のためのColliderをセットする

一定範囲に入ったことを検知するためには、Colliderを使用します。

まずColiderをアタッチするために、HierarchyウインドウのSlime_Green上で右クリックして「Create Empty」を選択し、空っぽの子オブジェクトを作成します。名前は「CollisionDetector」としておきます。

図7.12 検知のためのゲームオブジェクトを準備



CollisionDetectorにSphere Colliderをアタッチし、「Is Trigger」にチェックを付けます。

デフォルトの状態ではCollider同士がぶつかると跳ね返りますが、Is Triggerにチェックを付けることによって、Collider同士がぶつかってもすり抜けるようになり、衝突判定だけが実行されます。これを利用して、範囲内に入ったかどうかを判定します。

この判定方法は、「攻撃が当たったどうか」や「ゴールにたどり着いたか」など、さまざまな場面で使用できるので覚えておきましょう。

また、今回はRadiusは「4」に設定します。Radiusを変えると、Colliderの範囲（Edit Colider ボタンを押すとSceneビューに緑色で表示されます）もリアルタイムで変わりますので、任意の範囲に調整しましょう。

図7.13 Sphere Colliderの設定

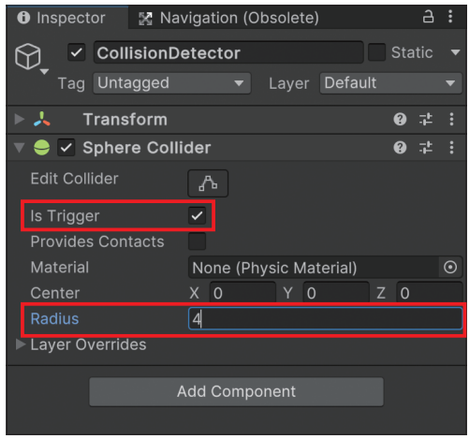


図7.14 Colliderの範囲表示

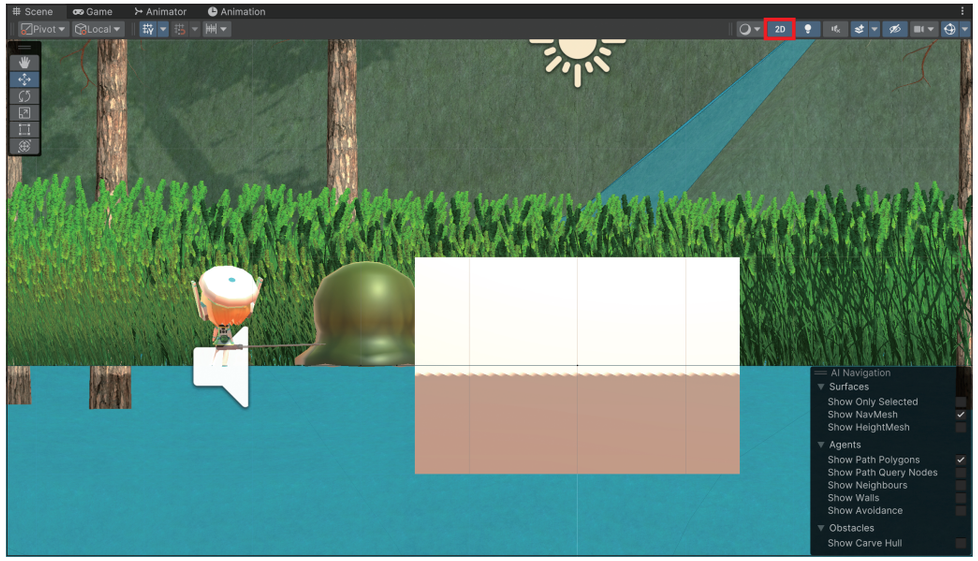
8-1 タイトル画面を作ろう

UnityのUI（ユーザーインタフェース）の基本的な機能を使って、タイトル画面を作成してみましょう。

8-1-1 新規シーンの作成

UI（User Interface、ユーザーインタフェース）を編集するには、Sceneビュー上部の「2D」をクリックして、2D用の視点にしておきます。これでUIを正面から見た状態に変更できます。

図8.1 Sceneビューを2Dに変更



「File」→「New Scene」（ショートカットは Command ＋ N）を選択して新しいシーンを作成します。Scene Templateは「Basic (Built-in)」を選択してください。

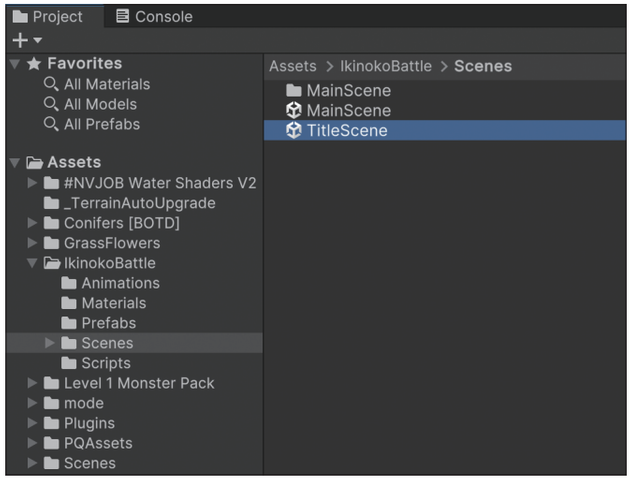
作成したら Command ＋ S を押し、名前を「TitleScene」として保存しておきます。

シーンも1ヵ所にまとめた方が管理しやすいため、Projectウインドウで「IkinokoBattle」の下に「Scenes」フォルダを作成し、Chapter 7までに使用したMainSceneと一緒に入れておくと良いでしょう。

開いているSceneファイルの移動はできないため、MainSceneを開いている場合は、別のシーンを開いてから、MainSceneファイルを移動してください。

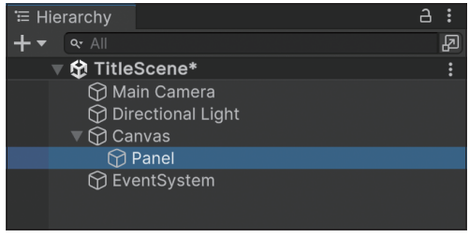
また、「Assets」－「Scenes」－「MainScene」フォルダにはベイクされたNavMeshが入っています。MainSceneを移動する際はMainSceneフォルダも一緒に移動してください。

図8.2 新規シーンの作成と移動



シーンの移動が完了したら、TitleSceneで作業を進めていきます。Hierarchyウインドウ上で右クリックし、「UI」→「Panel」を選択すると、Canvasとその子オブジェクトのPanelが生成されます。

図8.3 Canvasの作成



8-1-2 Canvasとは

Canvasはその名の通りUIを配置するキャンバスのことで、この中にさまざまなUI部品を配置していきます。

9-1 BGMやSEを追加しよう

BGMやSE（効果音）のアリ・ナシでゲームの印象が大きく変わります。ここでは音声ファイルの扱い方を解説します。

9-1-1 Unityで再生可能な音声ファイル

Unityでは、WAV（拡張子.wav）、MP3（拡張子.mp3）、Vorbis（拡張子.ogg）など、さまざまな音声ファイルを再生することが可能です。

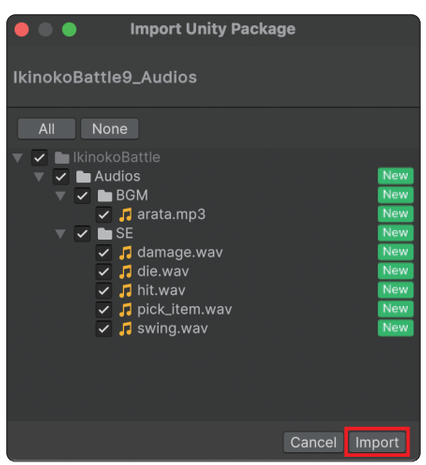
音声ファイルをProjectウインドウにドラッグ＆ドロップすると、Audio Clipとしてインポートされます。

9-1-2 Audio Clipのプロパティ

Audio Clipでよく使う設定がいくつかありますので、把握しておきましょう。

サンプル配布ページ（https://gihyo.jp/book/2023/978-4-297-13741-0/support）から、BGMとSEの音声ファイルを含むパッケージ「IkinokoBattle9_Audios.unitypackage」をダウンロードし、ダブルクリックした開いたImport Unity Packageダイアログで「Import」ボタンをクリックしてインポートします。

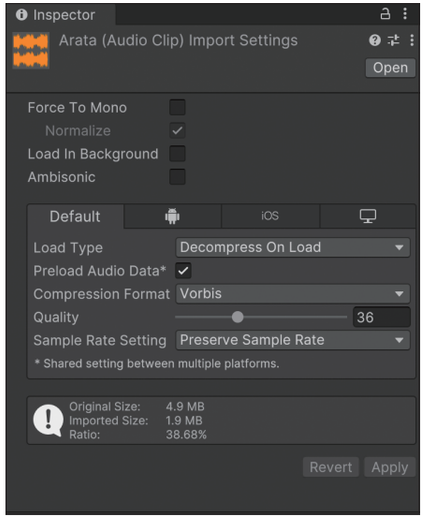
図9.1 BGMとSEのインポート



Unityにインポートした音声ファイルはAudio Clipとして扱われます。

Projectウインドウで「IkinokoBattle」－「Audios」－「BGM」－「arata」を選択し、Inspectorウインドウで「Audio Clip」のプロパティを確認してみましょう。

図9.2 Audio Clipのプロパティ



Load Type

Load Typeはゲーム実行時に音声ファイルをどう読み込むかを設定するプロパティです。ゲームのパフォーマンスに影響を与えます。

音声ファイルのサイズによって、どのLoad Typeを設定すればよいかは変わってきますので、適宜使い分けましょう（表9.1）。

表9.1 Load Typeの種類

Load Type	説明
Decompress On Load	音声ファイルを読み込む際にデコードしてメモリ上に保持する。パフォーマンスは良くなるが、デコードしたデータはサイズが大きくメモリ容量を圧迫するため、元々のサイズが小さいSEなどで使用する
Compressed In Memory	音楽データを圧縮されたままの状態でメモリに保持し、再生時にデコードする。メモリ消費は抑えられるが、再生開始時の負荷は大きくなる。Decompress On LoadとStreamingの中間にあたる設定
Streaming	音楽データをメモリに展開せず、随時デコードしながら再生する。再生時にメモリをほとんど消費しない代わりに、再生中に負荷がかかり続ける。BGMなどサイズが大きい音声を再生する場合に使用する

Quality

Qualityは音声ファイルの品質に影響するプロパティです。値が大きくすると音質が良くなり、小さくすると音質が悪くなります。この設定は音声ファイルのサイズに影響を及ぼします。

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

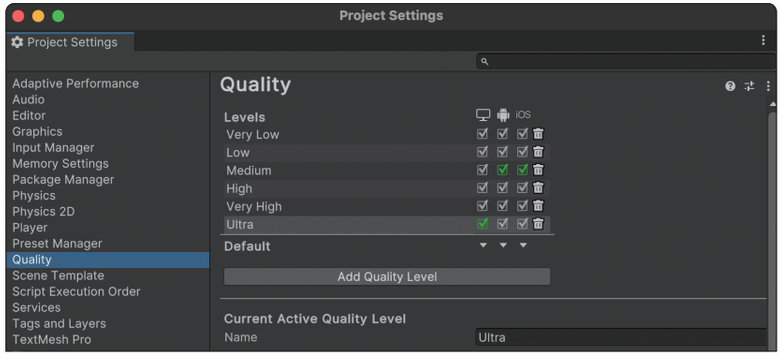
Qualityでは、AndroidやiOSなど任意のプラットフォーム画質設定を行います。チェックボックスが緑色になっているのがDefaultの設定です。

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

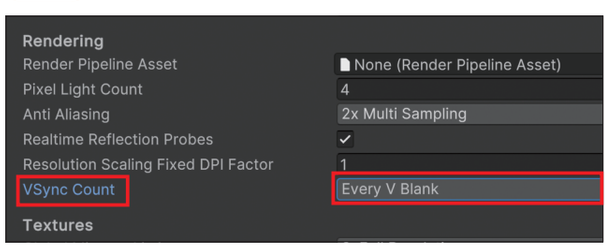
Qualityを下にスクロールすると、VSync Countの項目があります。ここでは、垂直同期を設定します。垂直同期とは、ディスプレイのリフレッシュレートとフレームレートを同期することで、この設定によって画面の描画が安定させることができます。

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のものが多い一方で、ゲーミングディスプレイでは240Hzに達するものもある
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



13-2-3 UnityのDebugモードを使う

スクリプトの状態を詳しく確認したい場合は、UnityのDebugモードを使ってみましょう。Debugモードにすると、スクリプトのprivateフィールドをInspectorウインドウ上で確認することが可能です。

Debugモードを実行するには、Inspectorウインドウ右上の（⋮）をクリックし、「Debug」を選択します。

Inspectorウインドウの表示内容が変わり、privateなフィールドが見られるようになりました。

図13.6 Debugモードの設定

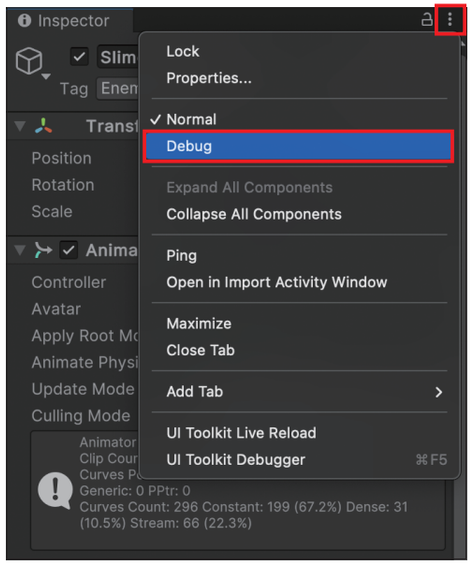
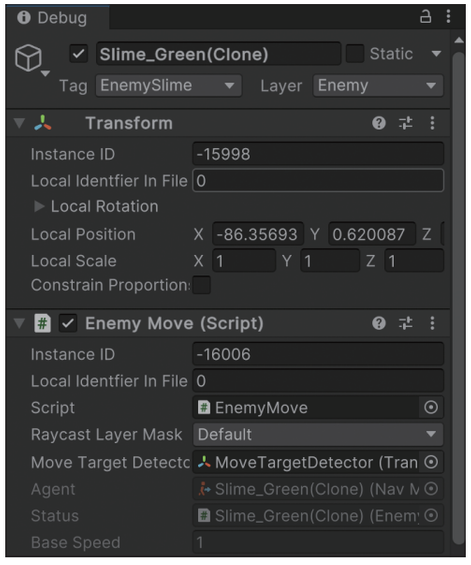


図13.7 Debugモードでの表示



13-2-4 デバッグ機能を実装する

独自のデバッグ機能を準備しておくことも有効な手段の1つです。

プレイ中に任意の情報を表示するのに加え、テスト用のチート機能（たとえば、キャラをLv.99にするボタンなど）を準備しておくと、調査の効率がアップします。

ただし、テスト用のチート機能の扱いには注意が必要です。

「リリース版で機能をOFFするのを忘れて、チートし放題だった……」といった悲劇を回避するには、Unityのプラットフォーム依存コンパイル（https://docs.unity3d.com/ja/current/Manual/PlatformDependentCompilation.html）が有効です。この機能を利用すると、「Unityエディタでのみデバッグ機能を有効にし、リリース版には含めない」といった制御が可能です。

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