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。

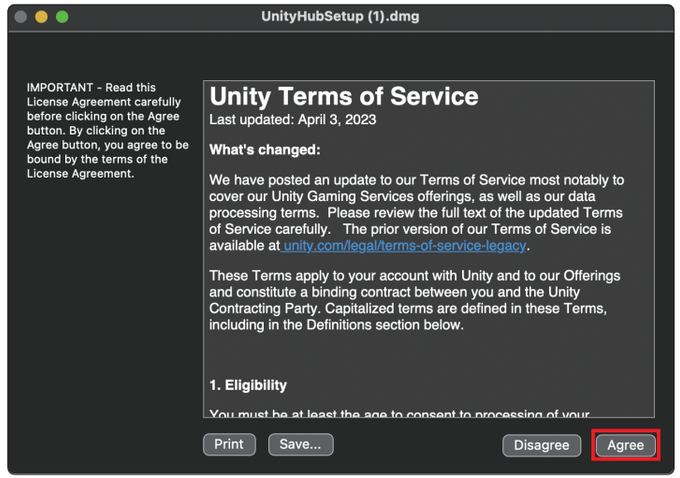
それでは、Unity Hubをダウンロード・インストールしていきましょう。

図2.1 Unity Hubのダウンロード

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用にダウンロード」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。

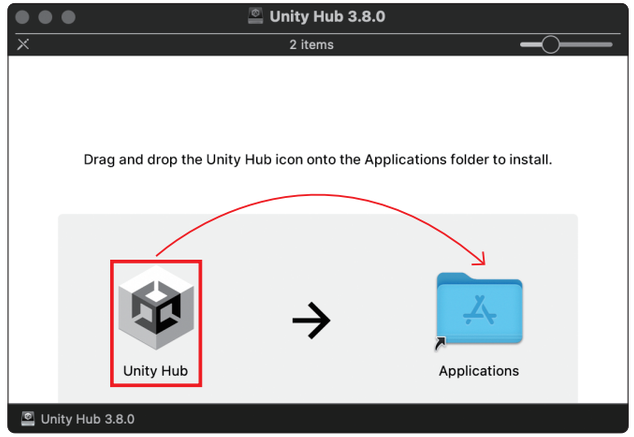
ダウンロードが完了したら、「UnityHubSetup.dmg」をダブルクリックします。英語で記述された利用規約が表示されますので、「Agree」をクリックします。

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



3-1 Unityでスクリプトを使おう

Unityの各機能やさまざまなAssetを利用すれば、スクリプトを書かなくてもゲームを作れます。しかし、自分でスクリプトを書けるようになれば、できることが格段に広がります。

3-1-1 スクリプトを作成する

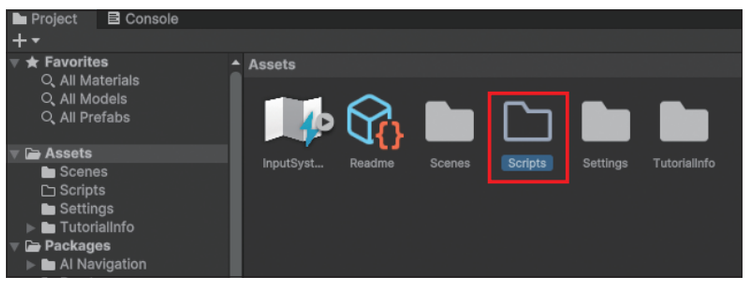
プログラミングは経験が無い人にとっては「難しそう」というイメージがあり、身構えてしまうかもしれません。しかし、基礎を習得しておけば決して難しいものではありません。

それでは、UnityでC#を使ったスクリプトを作成する手順を確認していきましょう。

Projectウインドウの Assetsフォルダで右クリックし、「Create」→「Folder」を選択してフォルダを作成します。フォルダ名は「Scripts」とします。

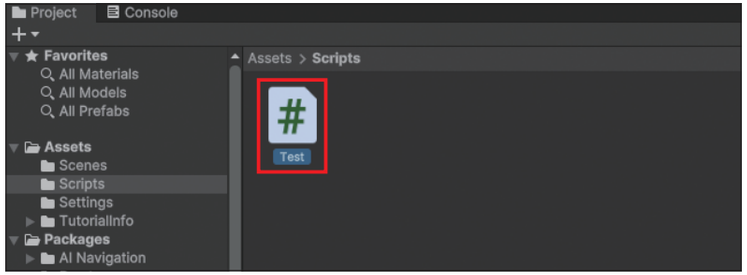
フォルダを作成しなくてもスクリプトは動作しますが、開発を進めるとスクリプトはどんどん増えていき、管理が大変になってきます。少しでも管理しやすくするため、適宜フォルダでまとめるようにしましょう（本書サンプルではScriptsフォルダに直接入れていますが、本格的なゲームを開発する際は、シーンや機能ごとにフォルダを分けることをおすすめします）。

図3.1 フォルダの作成



続いてスクリプトを作成します。Scriptsフォルダで右クリックし、「Create」→「Mono Behaviour Script」を選択し、スクリプト名は「Test」とします（リスト3.1）。ちなみに、スクリプト作成時に付けた名前はクラス名（リスト3.1の「class Test」の部分）にも自動で反映されますが、あとでスクリプト名を変更した場合は、クラス名には反映されません。名前がバラバラだと管理しづらくなりますので、スクリプト名とクラス名を一致させておきましょう。

図3.2 スクリプトの作成（Test.cs）



リスト3.1 Test.cs

```
using UnityEngine;

public class Test : MonoBehaviour
{
    // Start is called once before the first execution of Update after the
MonoBehaviour is created
    void Start()
    {

    }

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

    }
}
```

3-1-2 スクリプトをアタッチする

作成したスクリプトをゲームオブジェクトにアタッチしてみましょう。ゲームオブジェクトにアタッチすることで、スクリプトが実行されるようになります。

Hierarchyで右クリックし、Create Emptyで空のゲームオブジェクトを作成します。作成したゲームオブジェクトを選択し、Inspectorウインドウで「Add Component」ボタンをクリックし、検索BOXに「Test」と入力すると、先ほど作成したスクリプトが出てきます。

4-2 ゲームの方向性を決めよう

企画書を作る前段階として、ゲームの方向性を決めていきましょう。自分が制作するゲームがどのようなものか、まずは自分自身の頭の中ではっきりとイメージすることがその目的です。

4-2-1 ゲームの概要を思い浮かべてメモする

さて、皆さんが作りたいゲームはどんなものでしょうか。

まずざっくりと「どのようなジャンル」で「何が特徴」なのかを考えてみると良いでしょう。以下にいくつか例を挙げてみます。

- スライドパズルでダンジョン探索！「パズルRPG」
- 指一本で遊べるヒマつぶし！「階段駆け下りアクション」
- レトロな見た目で激しい弾幕！「ドット絵2D シューティング」
- 辺りを見回して謎を解け！「VR 謎解きゲーム」
- リズムに乗ってスピードアップ！「リズム＋レースゲーム」

ゲームイメージをメモする

どのようなゲームを作りたいのか、思い浮んだイメージをメモしていきましょう。

絵心が無くてもかまいません。落書きレベルで良いので、棒人間や○△□などのかんたんな図形を使って、イメージしたゲーム画面を絵として形にしていくことが重要です。

本書サンプルゲームの場合

本書のChapter 5から制作するサンプルゲームでは、ゲームジャンルを「お手軽3Dサバイバルアクション」としました。ゲームのウリは「ゲームを作っていきながら、Unityの基本機能を一通りきちんと学べること」にします。

概要は以下のような感じです。

- 3D アクションにサバイバル要素をドッキング！
- 武器を振ると時間が進むシステムを搭載。サクサク遊べる3Dサバイバルアクション！
- 材料を手に入れるには武器を振らないといけない。でも、武器を振ると時間が経ってしまう。時間が経つにつれ、敵の攻撃も激しくなってきます。効率良くさまざまな材料を手に入れて、アイテムを作りながら生き延びていこう！

4-2-2 ゲームを制作する理由を考える

次に、みなさんがそのゲームを制作しようと思った理由を考えてみましょう。

たとえば、以下のような理由が考えられると思います。

- 好きなゲームがあり、それに似たゲームを作りたかったから
- 最近遊んだゲームに不満があって、もっと良いものが作れると思ったから
- 何となく思いついて、すぐに作れそうだったから
- 新しく技術を学んで、それを使ったゲームを作りたかったから
- ヒットしそうな、おもしろいアイデアだと思ったから
- 絵や物語を書いているうちに、ゲーム化したくなったから
- 既存のゲームに飽きて、これまでにないゲームを作りたかったから

Tips ゲームの目的と目標を分けて考える

ゲームを頑張って作っても、単調だったりすぐにマンネリ化すると、プレイヤーは飽きてしまいます。たくさんのゲームが日々リリースされていますので、一度飽きてしまったプレイヤーはほぼ戻ってこないと考えて良いでしょう。

飽きやマンネリ化を少しでも避けるため、「プレイヤーの最終目的」と「最終目的への道しるべとなる小さな目標」を分けて考え、実装することをおすすめします。

ゲームクリアにつながる「さらわれたお姫様を助け出す」や「魔王を倒して世界に平和を取り戻す」などは、ゲームの「目的」です。この目的が単体で存在するだけでは、プレイヤーが途中で飽きてしまいます。

そこでたとえばRPGの場合は、ゲームの節々で「ダンジョンを攻略する」「ボスを倒す」「次の街を目指す」といった目標を散りばめ、「小さな目標を順番に達成していき、魔王を倒すという最終目的を達成した」という流れを作ると、プレイヤーの離脱を抑えることができます。

また1回1分程度でプレイできて、ハイスコアを目指すだけのシンプルなゲームであっても、「一定のスコア獲得で新要素を開放」「コインを集めて新要素を開放」「アチーブメント（条件を満たすと獲得できる称号）」といったプレイの目標になり得るしくみを入れておくと、繰り返し遊んでもらいやすくなります。

5-1 プロジェクトを作成しよう

ここでは、本書サンプルゲームのプロジェクトを作成して、地形を作成するために必要なAssetをそのプロジェクトに追加していきます。

5-1-1 プロジェクトの作成

Unityでは、ゲームをプロジェクト単位で管理しています。まずは本書で作成するゲーム用のプロジェクトを作成しましょう。

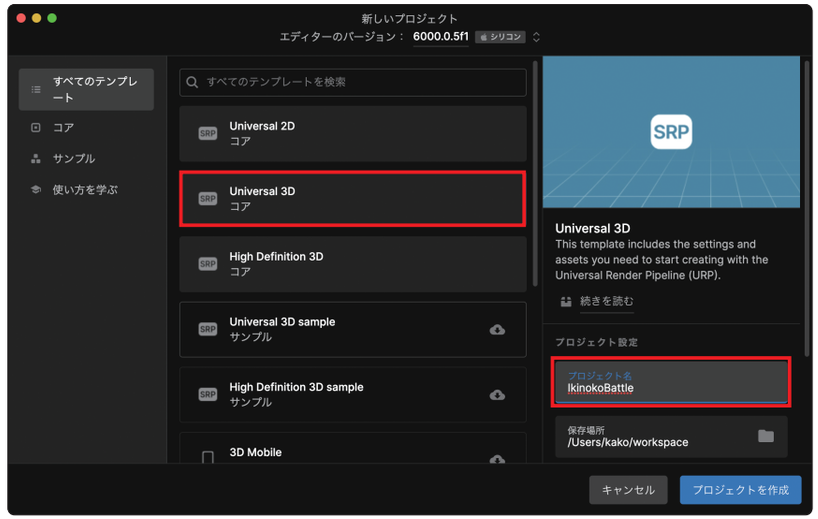
Unity Hubで「プロジェクト」を選択して「新規作成」ボタンをクリックします。以下のように入力して「プロジェクトを作成」ボタンをクリックします。

- テンプレートを「Universal 3D」
- プロジェクト名を「IkinokoBattle」

Unityには3種類のレンダーパイプライン（描画方式）があり、今回選択した「Universal 3D」テンプレートでは、URP（Universal Render Pipeline）を使用します。

URPは、軽量かつ表現力豊かな描画が可能な万能レンダーパイプラインです。プロジェクトのレンダーパイプラインに対応したAssetを使わないと正しく描画されない場合があるため、「このプロジェクトではURPを使っている！」と覚えておいてください。

図5.1 プロジェクトの新規作成



プロジェクトが作成されると、SampleSceneが開きます。Projectウインドウの「Assets」ー「Scenes」の中にある「SampleScene」の文字をクリックし、シーン名を「MainScene」に変更して作業を進めましょう。

5-1-2 Asset Storeとは

3D世界の基礎となる地形は、Terrain（テレイン）を使って作成します。

Terrainは3D地形データの作成ツールで、山や谷などの起伏・木々の生い茂る森などをさまざまなブラシを使ってお絵かき感覚で作成することができます。

初期状態のプロジェクトにTerrain用の素材は含まれていません。そこでUnityが持つ強力な機能の1つであるAsset Storeを使ってゲームの素材を準備しましょう。Asset Storeとは、Unityで使えるAssetを購入・ダウンロードすることができるオンラインストアです。

Asset Storeでは、画像・音楽・3Dモデルなどの部品から、Unityのエディタ拡張機能・ゲームのプロジェクト丸ごとに至るまで、さまざまなAssetが配布・販売されています。またパブリッシャーの登録を行うことで、自分が制作したAssetを販売することもできます。

Asset Storeは、Unityの大きなメリットの1つです。というのも、Asset Storeで素材を購入・ダウンロードすることによって、短期間でかつ高品質なゲームの開発が可能になるためです。

素材を自分で作成することはとても楽しい作業ですが、目的が「ゲームを作って世に出すこと」であれば、Asset Storeから素材を調達して開発時間を節約するようにしましょう。

6-1 キャラクターをインポートしよう

プレイヤーキャラクターの作り方を理解すると、さまざまなゲームに活用できます。ここからは本書で用意したサンプルプロジェクトを使って、キャラクターの3Dモデルをインポートし、そのキャラクターに影を付けるまでの手順を説明します。

6-1-1 サンプルプロジェクトの準備とAssetのインポート

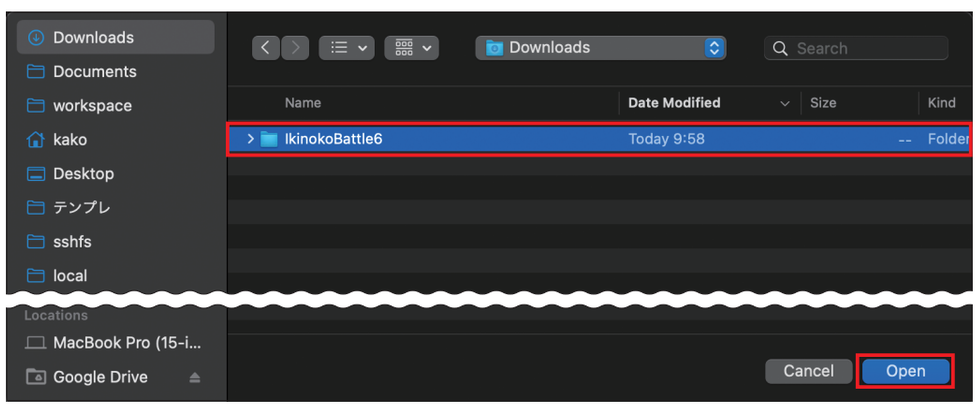
まずはサンプルプロジェクトを準備し、Chapter 5で使用した各種Assetをインポートします（サンプルプロジェクトの代わりに、前の章で作成したプロジェクトを使ってもOKです。その場合、6-1-1は読み飛ばしてください）。

◎ サンプルプロジェクトの準備

P.4を参照して、技術評論社のサポートページ（https://gihyo.jp/book/2024/978-4-297-14514-9/support）からIkinokoBattle6.zipを入手します。

次にUnity Hubを起動して「プロジェクト」タブを選択し、「開く」ボタンをクリックします。フォルダの選択ウインドウが表示されますので、解凍したIkinokoBattle6フォルダを選択して「開く」（Windowsの場合は「フォルダの選択」）をクリックします。

図6.1 IkinokoBattle6フォルダを選択



プロジェクトにはUnityエディターバージョンが設定されていて、該当するバージョンのUnityがインストールされていない場合は「見つからないエディターバージョン」と記載されたダイアログが表示されます。ダイアログにはインストール済みのUnityバージョン一覧も表示されますので、その中から任意のバージョンを選択してプロジェクトを開いてください（途中でバージョン変更の確認ダイアログ等が表示されますので、「Continue」を押して続けます）。

Unityエディタのバージョンを変更すると、自動的にプロジェクトの最適化が行われるため、クリックしてから開くまでに少し時間がかかります。プロジェクトを開いたら、Projectウインドウの「Assets」ー「Scenes」ー「MainScene」を開いておきましょう。

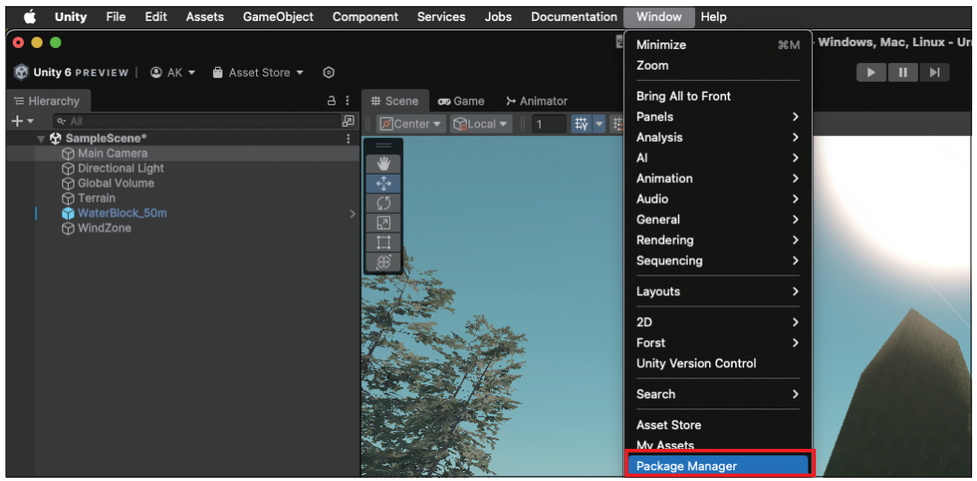
◎ 各種Assetのインポート

次に、Chapter 5で使用した5つのAssetをインポートします。手順はすべて同じですので、5-1-3を参照して作業を行ってください。

- Terrain Textures Pack Free
- Conifers [BOTD]
- Grass Flowers Pack Free
- Simple Water Shader URP
- Wispy Skybox

入手済みのAssetのインポートを行う場合は、Package Managerから行います。「Window」→「Package Manager」を選択すると、Package Managerウインドウを開きます。

図6.2 Package Managerを開く



7-1 敵キャラクターがプレイヤーを追いかけるようにしよう

ここでは、Asset Storeから敵キャラクターで使用するAssetをインポートし、プレイヤーを追いかける動きを付けてみましょう。

7-1-1 敵キャラクターのインポート

Asset Storeから敵キャラクターをインポートします。本書ではクエリちゃんにマッチする、かわいい敵キャラクターの3Dモデルを使うことにします。

「Window」→「Asset Store」を選択してAsset Storeを開き、画面上部の検索ボックスに「level 1」と入力して「Level 1 Monster Pack」を検索します。

Level 1 Monster Packは、かわいい敵キャラクターの3Dモデルとアニメーションが同梱された無料で利用可能なAssetです。

検索結果で表示される「Level 1 Monster Pack」を、**5-1-3**を参照してダウンロード・インポートを実行します。

図7.1 Level 1 Monster Packの詳細

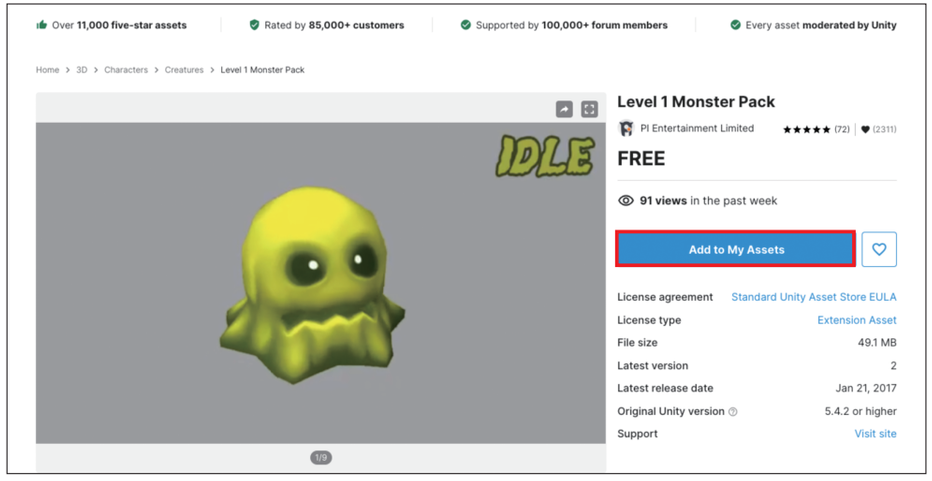
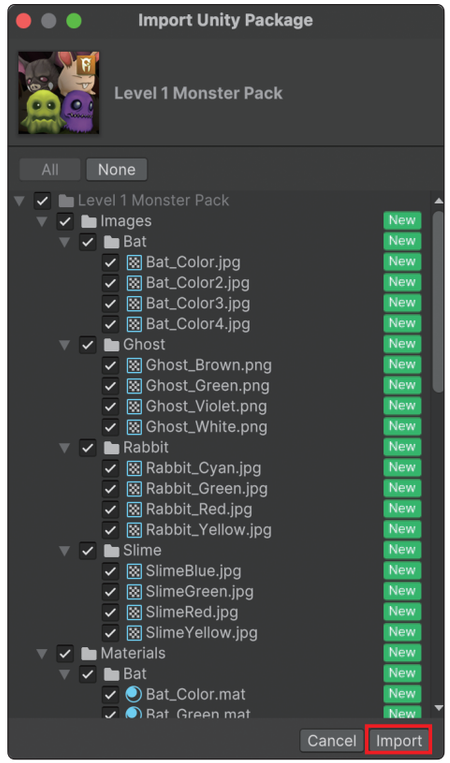


図7.2 Level 1 Monster Packのインポート



7-1-2 追いかけるのは意外と難しい

敵キャラクターにプレイヤーキャラクターを追いかけさせようとした場合、真っ先に浮かんでくる方法は「プレイヤーの方向に向かって敵キャラクターを移動させること」ではないでしょうか。

しかし、この方法には問題点があります。

敵キャラクターとプレイヤーの間に何も無ければ問題ありませんが、段差や障害物が間に存在すると、敵キャラクターはそこで詰まってしまいます。

それでもゲームとして成立するかもしれませんが、できればもう少しスマートに移動させたいところです。

そのような場合に使えるのが、目標地点までの経路探索を行うNavMeshというしくみです。

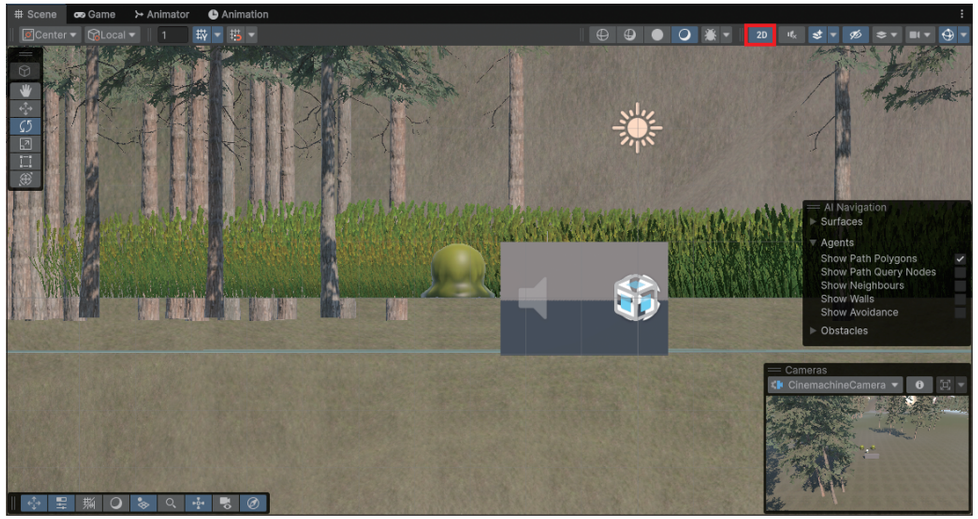
8-1 タイトル画面を作ろう

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 (URP)」を選択してください。

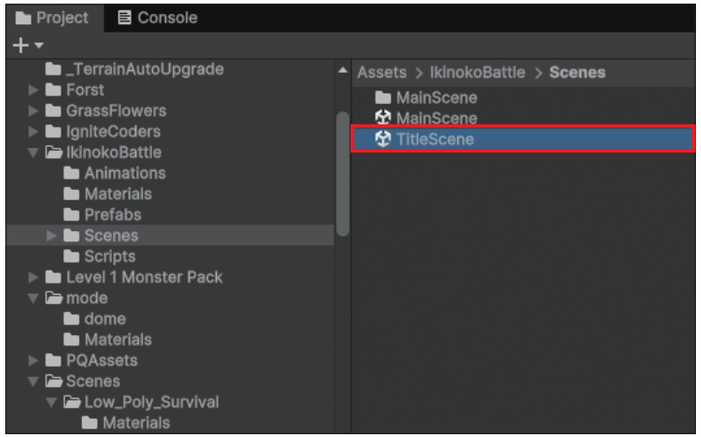
作成したらCommand＋Sを押し、名前を「TitleScene」として保存しておきます。

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。

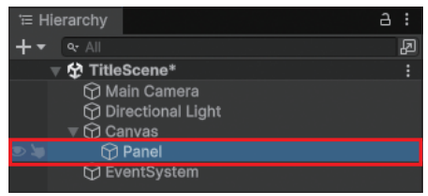
また、「Assets」ー「Scenes」ー「MainScene」フォルダにはベイクされたNavMeshが入っています。MainSceneを移動する際はMainSceneフォルダも一緒に移動してください。

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。

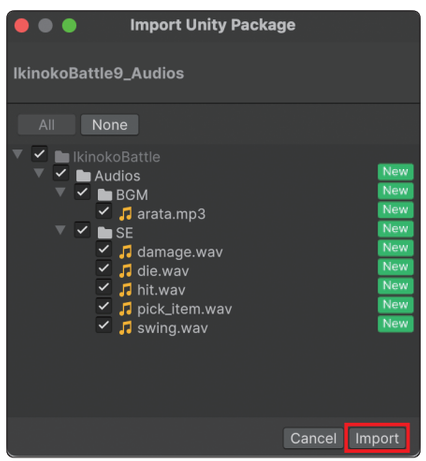
音声ファイルをProjectウインドウにドラッグ＆ドロップすると、Audio Clipとしてインポートされます。

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/2024/978-4-297-14514-9/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。

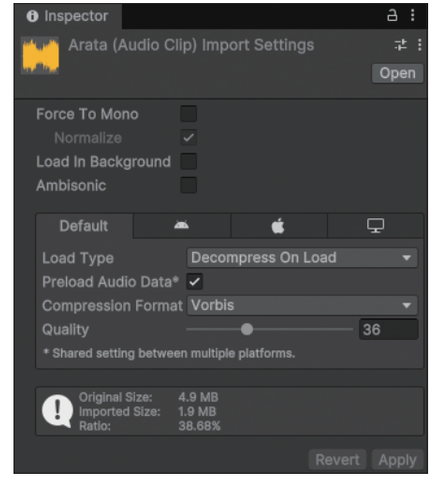
図9.1 BGMとSEのインポート



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。

他のプロパティの詳細については、公式マニュアル（https://docs.unity3d.com/ja/current/Manual/class-AudioClip.html）を参照してください。

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

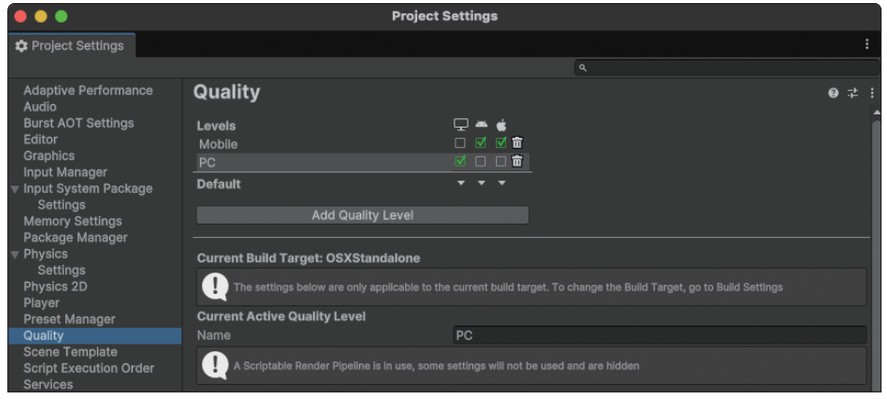
Qualityでは、AndroidやiOSなど任意のプラットフォーム画質設定を行います。チェックボックスが緑色になっているのがDefaultの設定です。

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

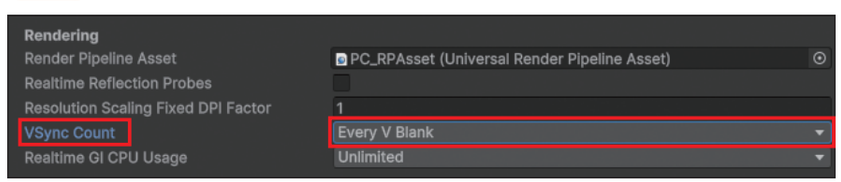
Renderingカテゴリの中にVSync Countの項目があります。ここでは、垂直同期を設定します。垂直同期とは、ディスプレイのリフレッシュレートとフレームレートを同期することで、この設定によって画面の描画が安定させることができます。

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



13-1 エラーを確認しよう

ゲームが実行できなくなったり、キャラクターが動かなくなるなどのトラブルが発生したときは、まずどのようなエラーが発生しているかを確認してみましょう。

13-1-1 エラーの内容をチェックしよう

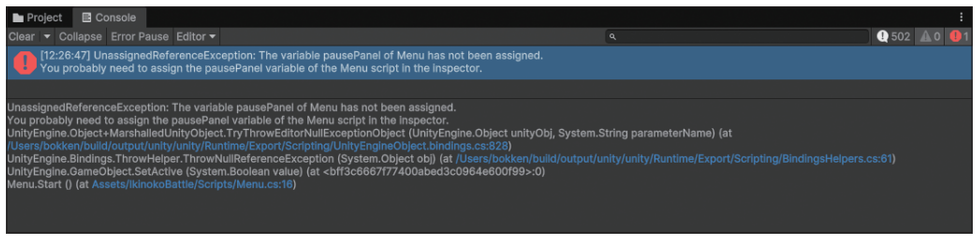
エラーが発生した場合、エラーの内容がConsoleウインドウ（Command＋Shift＋C）に表示されます。まずConsoleウインドウを開いて、どのようなエラーが発生しているのかを確認しましょう。

基本的にエラーログは表13.1に挙げた構造になっています。

表13.1 エラーログの構造

項目	説明
時間	エラーの発生時間。エラーログを含め、各ログに必ず表示される
エラーの種類	スクリプトの構文エラーなどの場合は、エラーの種類だけで対処方法がわかる場合がよくある
エラーメッセージ	エラーの詳しい内容が表示される。エラーの種類だけでは対処が難しい場合、これを元に対処方法を探すことになる（たいていは英語で表示される。英語が苦手な人はDeepLやGoogle翻訳を活用するとよい）
エラーの発生個所	エラーの発生個所と、その処理を呼び出しているスクリプトが順に表示される。青文字の部分をクリックすると、スクリプトの該当個所にジャンプする

図13.1 Consoleに表示されたエラー



自分の書いたスクリプトでエラーが発生している場合は、該当個所を開いて修正を試みましょう。スクリプト以外でエラーが発生していて、かつ対処方法が不明な場合は、13-1-3に進んでください。

13-1-2 よくあるスクリプトエラー

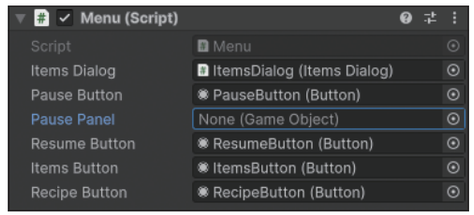
Unityで発生するエラーは多種多様ですが、中でも特によく発生するエラーと対処方法をいくつか紹介します。開発しているとほぼ確実に遭遇するものばかりですので、心の片隅に留めておきましょう。

UnassignedReferenceException・NullReferenceException

このエラーは、スクリプトで値がセットされていないフィールドにアクセスしようとすると発生します。

Inspectorウインドウで「None」や「Missing」になっているフィールドは値がセットされていませんので、チェックしてみましょう。

図13.2 値をセットし忘れたときの表示



MissingReferenceException

このエラーは、Destroy () を使ってゲームオブジェクトやコンポーネントを破棄したあと、その破棄したものにアクセスを試みる際に発生します。

破棄するかもしれないゲームオブジェクトやコンポーネントにアクセスする際は、値がnullかどうかをチェックすると良いでしょう。

```
以下のobjはDestroy()で破棄する場合があるとする
[SerializeField] private GameObject obj;
略
objは破棄されてnullになっているかもしれないので、チェックしてから処理を行う
if (obj != null) {
    obj.SomeAction();
}
```

MissingComponentException

このエラーは、GetComponent () を使って取得しようとしたコンポーネントが、対象のゲームオブジェクトにアタッチされていない場合に発生します。

取得対象のゲームオブジェクトが間違っていないかをチェックし、もし間違っていなければ、コンポーネントがアタッチされているかどうかをチェックしましょう。