1-2 UnityHub とは

この章では、Unity をインストールするのに必要な、Unity のバージョンや Unity で作られたプロジェクトを管理する UnityHub（ユニティハブ）というアプリケーションのインストールと、UnityID の作成までを解説します。

1-2-1 UnityHubの役割

UnityHub とは、複数の Unity のバージョンのインストールの管理や Unity で作成されたプロジェクトの管理を行うためのアプリケーションです（**図 1.2**）。またこれから作成するゲームに合ったテンプレートを利用して新規にプロジェクト作成することもできます。

図1.2　Unity Hubの画面

1-2-2 UnityHubのダウンロード

UnityHub のダウンロードページ（https://unity.com/ja/download）を Web ブラウザで開きます（**図 1.3**）。Web ブラウザで"unity hub"で検索しても見つかると思います。［Windows 用ダウンロード］または［Mac 用ダウンロード］をクリックして、インストーラを保存します。Mac の場合は 1-2-3 へ、Windows の場合は 1-2-4 へ進んでください。

図1.3　Unity Hubのダウンロードページ

1-2-3 UnityHubのインストール(Mac)

「ダウンロード」フォルダにダウンロードされた、UnityHubのインストーラ"UnityHubSetup.dmg"をダブルクリックしてインストールを開始します（**図 1.4**）。

図1.4　Macの UnityHubのインストーラ

利用規約が表示されるので、［Agree］をクリックし、「"UnityHubSetup.dmg"を開いています...」というダイアログが表示されるので終了するまで待ちます。

2-1 開発を始めよう

ここでは、プロジェクトの作成と最初のシーンの保存を行います。そして、Unity の座標系についても学んでいきましょう。

2-1-1 プロジェクトの作成

図 2.1 が今回作成する玉転がしのイメージです。

図2.1　この章で作成する玉転がし

まず最初にプロジェクトを作成していきます。1つのアプリやゲーム単位に1つのプロジェクトを作成します。UnityHub を起動して、ウィンドウの左側の［プロジェクト］を選択します。

1 プロジェクトの設定

右上の［新しいプロジェクト］ボタンをクリックすると「新しいプロジェクト」のウィンドウが開きます。［すべてのテンプレート］、［Universel 3D］を選択します。「プロジェクト名」はゲームのタイトル名のような好きな名前を付けます。ここでは「FallingBall」と入力します（もしこの場所に［テンプレートをダウンロード］ボタンが表示されている場合は、クリックします）。

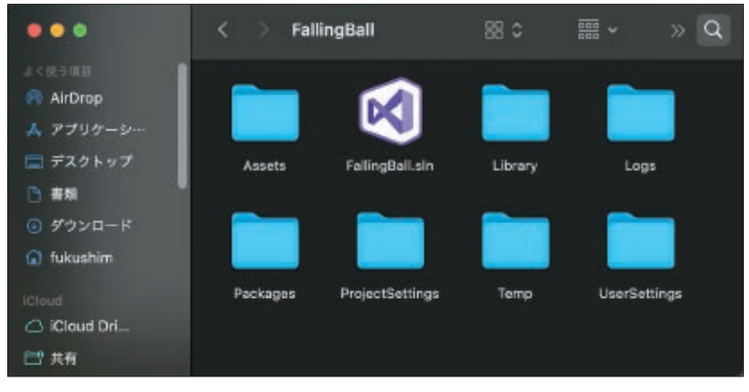
次に、「保存場所」を設定します。例えば、自分のユーザーフォルダに「Unity」というフォルダを作成して指定します。「Unity Cloud に接続」のチェックは外します（**図 2.2**）。

図2.2　「新しいプロジェクト」のウィンドウ

2 プロジェクトの作成

手順 1 の「プロジェクトを作成」ボタンをクリックします。「保存場所」の下に「プロジェクト名」のフォルダができて、その中に今回のプロジェクトで必要となるファイルが作成されます（**図 2.3**、**図 2.4**）。

図2.3　作成されたプロジェクトフォルダ(Mac)

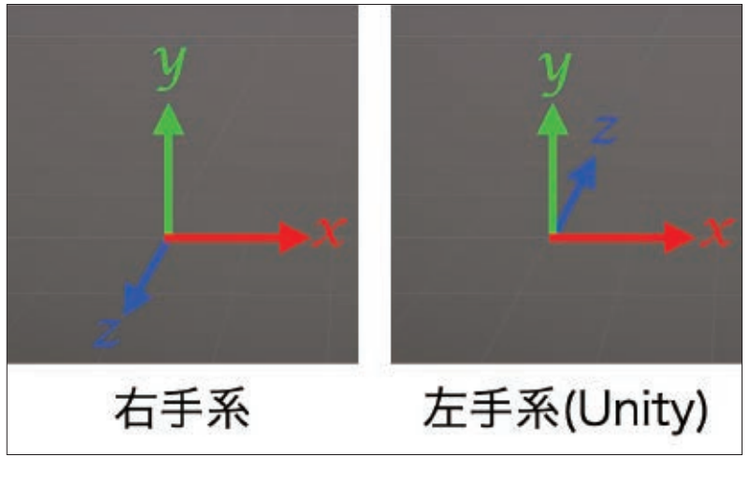


Unity が起動すると**図 2.5** のようなウィンドウが表示されます。これで開発を始める準備ができました。

2-1-3 Unityの座標系

Unityの座標系を理解しておきましょう。Unityエディタで、キャラクターや背景物などの配置されるすべてのものを「ゲームオブジェクト」と呼びます。このゲームオブジェクトを配置する空間の座標系を「ワールド座標系」と呼びます。ワールド座標系内で、ゲームオブジェクトの位置を示すのにX,Y,Z座標を用います。このとき各軸のプラス方向がどの向きなのかによって右手座標系と左手座標系に分かれます（**図2.9**）。

図2.9　右手系と左手系の座標系



Unityの場合は、左手座標系を用います。つまり、X座標は右がプラス方向で、Y座標は上がプラス方向、Z座標は奥がプラス方向になります。ゲーム中のキャラクター等を思ったように動かすには各軸のプラス方向がどちらなのかを理解しておく必要があります。

2-1-4 ワールド座標系とローカル座標系

ワールド座標系の空間内にゲームオブジェクトを置いたとき、そのゲームオブジェクトの座標を基準とした相対的な座標系を「ローカル座標系」と呼びます。例えると、宇宙空間が絶対的な座標系のワールド座標空間で、太陽の位置をワールド座標の中心絶座標（0,0,0）と仮定します。地球は太陽を中心に公転していて、さらに自転もしています。このときの地球上の座標系をローカル座標と呼びます。地球上に人が立ち止まっている場合、地球のローカル座標系に相当する経度緯度で見ると動いていませんが、ワールド座標系では動いていることになります（**図2.10**）。

モニターアームを例に、ゲームオブジェクトが親子関係にある場合のローカル座標の変化を見てみましょう。アームAの子がアームB、アームBの子がアームCとします（**図2.11**）。根本の関節Aを回転させると、子のアームBとCは連動して動きます。同様に関節Bを回転させるとアームCも連動して回転します。このように親の回転は子へ継承されていきます。このように座標、回転、スケールも親から子へ継承されていきます。

図2.10　ワールド座標系とローカル座標系

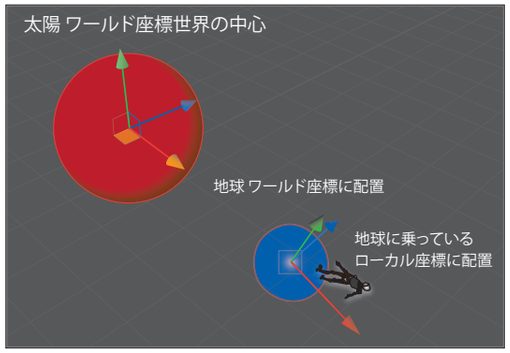
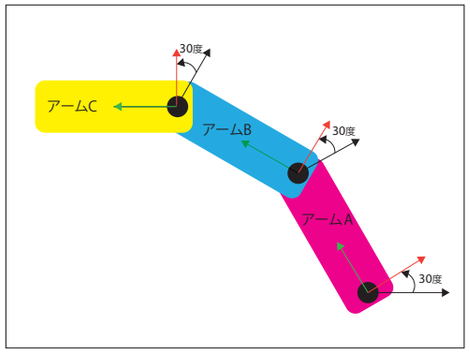


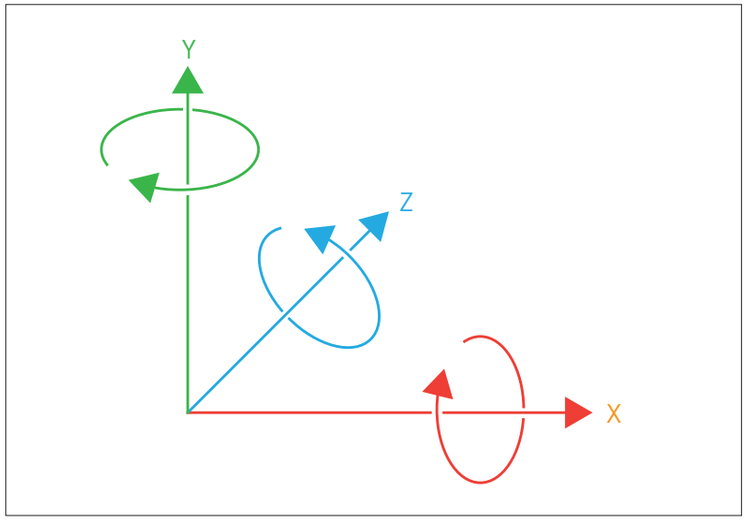
図2.11　モニターアームで見る親子関係



2-1-5 回転方向

回転方向について見ていきましょう。Unityの回転方向は左ねじの法則と呼ばれる反時計回りがプラスの方向になります（**図2.12**）。X,Y,Z軸で回転角を指定する場合は、どの順番で軸を回転させるかによって物体の向きが変わります。つまり各軸で同じ角度を指定したとしても、x軸→y軸→z軸の順番で回転させた場合と、z軸→y軸→x軸で回転させた場合では最終的な向きが異なります。このように軸を順番に回転させる手法をオイラー角と呼びます。Unityエディタの「Inspector」ウィンドウで指定した場合は、Z→X→Y軸の順で回転します。

図2.12　回転方向



2-3 スロープを作ろう

ここでは、ボールが転がる滑り台のようなスロープを作成していきましょう。スロープは複数の "Cube" を組み合わせて、親子構造を持つ１つのゲームオブジェクトにします。また、ゲームオブジェクトの複製方法も学びます。

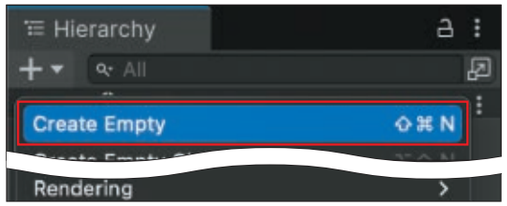
2-3-1 スロープの作成

ボールを転がす凹状のスロープを作っていきます。最初に親となる空のゲームオブジェクトを作成し、子として３つの "Cube" を追加して組み合わせてスロープを作ります。

1 親のゲームオブジェクトの作成

「Hierarchy」ウィンドウの ＋ をクリックし、[Create Empty] をクリックします（**図2.23**）。"GameObject" という名前のゲームオブジェクトが追加されます。これを一番上の親とします。

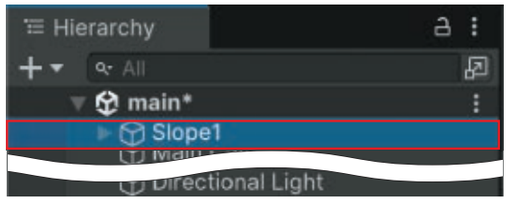
図2.23　空のゲームオブジェクトの作成



2 名前の変更

追加直後は「Hierarchy」ウィンドウで名前を変更できる状態になっているので、"Slope1" に変更します。もし "GameObject" と名前が確定してしまった場合は、右クリックのメニューから [Rename] を選択し、名前入力状態にして "Slope1" に名前を変更します（**図2.24**）。

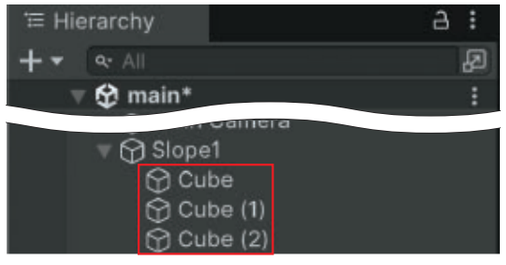
図2.24　追加された "Slope1" のゲームオブジェクト



3 子に３つの "Cube" を作成

"Slope1" を右クリックし、表示されたメニューから [3D Object] → [Cube] の選択を３回繰り返して "Slope1" の子に "Cube" を３つ追加します（**図2.25**）。

図2.25　"Slope1" の子に追加された３つの "Cube"



4 "Slope1" の位置の調整

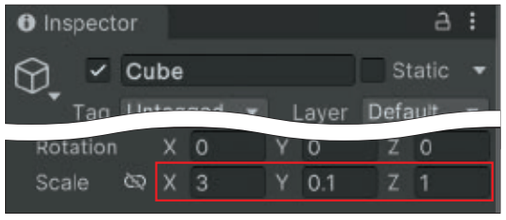
"Slope1" を見やすいように、シーンビューの移動ツールでＹ座標（緑の矢印）を上に移動します（**図2.26**）。また何もない場所でマウスの右ボタンを押したままドラッグして角度を回転したり、中ボタンを押したままドラッグで上下左右に位置を移動したり、マウスホイールで位置を前後に移動して調整してください。

図2.26　"Slope1" を見やすい位置に移動

5 "Cube" の調整

"Cube" の Scale を（3, 0.1, 1）にします。これはスロープの底面になります（**図2.27**）。

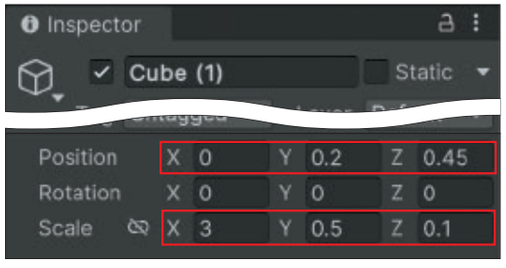
図2.27　底面となる "Cube" の Scale の設定



6 "Cube(1)" の調整

"Cube(1)" の Position を（0, 0.2, 0.45）、Scale を（3, 0.5, 0.1）にします（**図2.28**）。これはスロープの片方の側面になります。

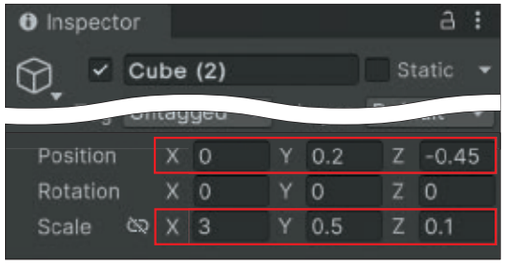
図2.28　側面となる "Cube (1)" の Position と Scale の設定



7 "Cube(2)" の調整

"Cube（2）" の Position を（0, 0.2, -0.45）、Scale を（3, 0.5, 0.1）にします（**図2.29**）。これはスロープのもう片方の側面になります。

図2.29　もう片方の側面となる "Cube (2)" の Position と Scale の設定



3-1 プロジェクトを作成しよう

今回作成するゲームは、一定時間毎にランダムで生成されていく床に沿って、キーボードでボールに力を加えて床から落ちないように転がしていき、ゴールまで導くという簡単なゲームを作成していきます。

3-1-1 今回作成するゲームの概要

ランダムで生成された床は一定時間後に消えるため、ボールを落ちないように上手く転がして新しく生成された床へ進めなければなりません。また段差のある床も生成されるので、ジャンプで乗り越えて行きましょう。青い床のゴールに到達することがこのゲームの目的です。

図3.1　作成するゲーム

3-1-2 プロジェクトとシーンの作成

それでは、プロジェクトを作成していきます。

1 プロジェクトの作成

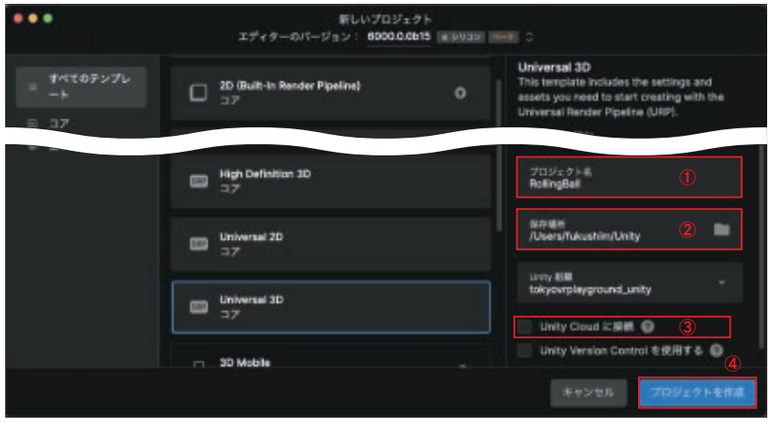
UnityHubを起動して、ウィンドウの左側の［プロジェクト］を選択します。右上の［新しいプロジェクト］ボタンをクリックすると「新しいプロジェクト」のウィンドウが開きます（**図3.2**）。［すべてのテンプレート］を選択し（①）、［Universal 3D］を選択します（②）。［テンプレートをダウンロード］ボタンが表示されている場合は（③）、クリックします。表示されていない場合は次に進みます。

図3.2　プロジェクトの作成ウィンドウ

2 プロジェクト名の設定

「プロジェクト名」は"RollingBall"と入力し（**図3.3**①）、「保存場所」をユーザーフォルダ等に設定後（②）、「Unity Cloudに接続」のチェックは外します（③）。
右下の［プロジェクトを作成］ボタンをクリックします（④）。保存場所のフォルダにプロジェクト名のフォルダができて、その中に今回のプロジェクトで必要となるファイルが作成されます。

図3.3　プロジェクトを作成する



3-3 ボールを操作しよう

ここでは、ゲームオブジェクトとC# スクリプトを連係して、C# スクリプトでゲームオブジェクトを制御していきます。キーボードの入力を取得してボールに力を加えて転がしたり、ジャンプさせてみましょう。

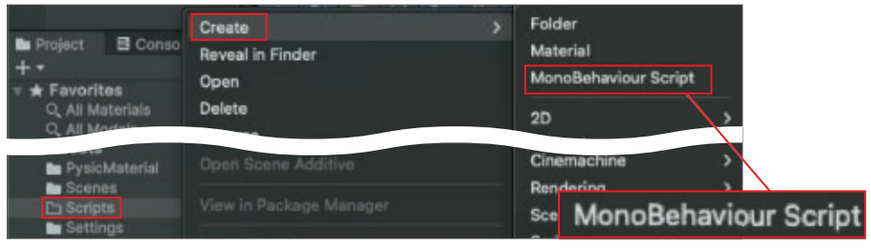
3-3-1 スクリプトの設定（ボール）

まず最初にC#スクリプトを作成して、ボールのゲームオブジェクトにセットします。

1 ボールのスクリプトの作成

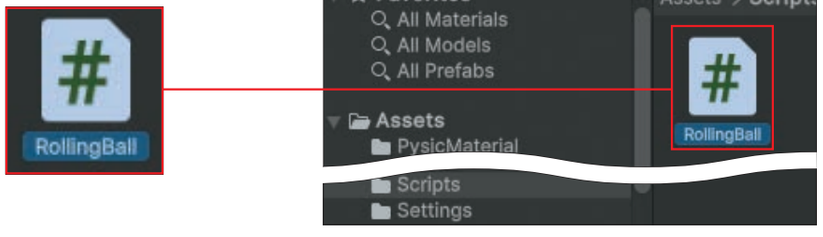
「Project」ウィンドウで"Assets"フォルダを選択して、左上のプラスボタンをクリックし"Folder"を選択して、"Scripts"という名前のフォルダを作成します。"Scripts"を選択して、右クリックメニューから「Create→MonoBehaviour Script」を選択します（**図3.12**）。

図3.12 "MonoBehaviour Script"の作成



作成されたファイルに"RollingBall"と名前を付けます（**図3.13**）。これがC#スクリプトファイルです。

図3.13 "RollingBall.cs"スクリプト

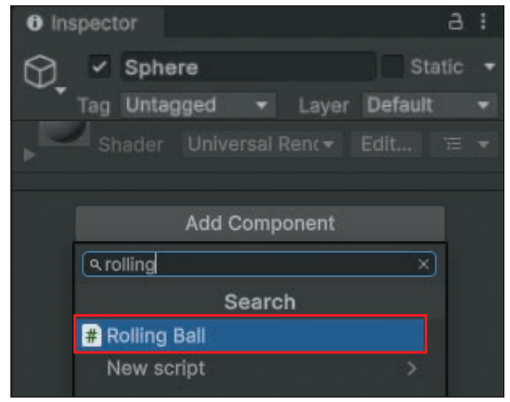


2 スクリプトをボールに設定

「Hierarchy」ウィンドウで"Sphere"を選択し、「Inspector」ウィンドウで[Add Component]ボタンをクリックします。検索ボックスに"Rolling"と入力し今作成した"RollingBall"スクリプトを選択します（**図3.14**）。

これでゲームオブジェクトの"Sphere"にC#スクリプトがセットされ、連携することができました。

図3.14 "Sphere"に"RollingBall.cs"スクリプトを追加



コラム　C#スクリプトファイルの注意点

手順1でC#スクリプトファイルに名前を付ける際の注意点です。もし名前を間違って確定してしまった場合うまくいかなくなります。その場合、作成されたファイルを右クリックしてメニューから「Delete」を選択し一旦削除してから、再度「MonoBehaviour Script」を作成し直します。ファイル名を後から変更しても正しく動きません。また大文字小文字は間違えないでください。

3-3-2 ボールの物理挙動

次に、C#スクリプトからボールに力を加えて、転がしてみます。

1 Visual Studioの起動

「Project」ウィンドウで"RollingBall"スクリプトをダブルクリックするとVisual Studioが起動して、C#スクリプト"RollingBall.cs"が表示されます（**リスト3.1**）。もしVisual Studioをインストールしていない場合は、Unity Hubの追加コンポーネントのインストールからインストールしてください。

リスト3.1 "RollingBall.cs"

```
using UnityEngine;

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

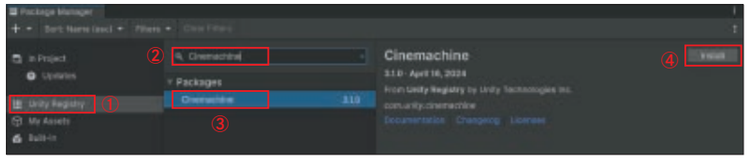
3-4-5 カメラの追従（ボール）

現状、カメラがボールの動きについて来ていません。「Cinemachine」パッケージを追加して「Follow Camera」を使い、カメラがボールについてくるようにしてみます。次の手順に沿い設定してみましょう。

1 「Cinemachine」のインポート

Unityのメニュー「Window→Package Manager」を選びます。パッケージマネージャーが開いたら、左の [Unity Registry] を選択します。中央上部の [Search Unity Registory] と表示されている箇所に"Cinemachine"と入力します。[Cinemachine] を選択して、右側上部の [Install] ボタンをクリックします（**図3.25**）。これでインストールされます。

図3.25　Package Managerから「Cinemachine」のインストール



2 "Follow Camera"の配置

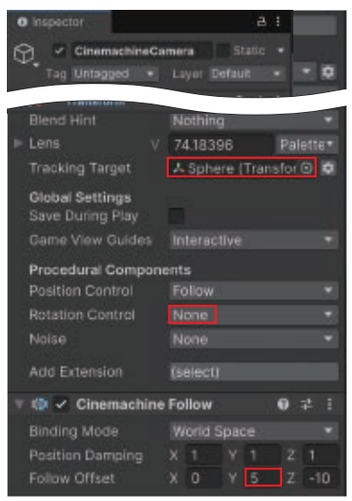
インストールが完了したら、メニューから [GameObject] →[Cinemachine] →[Targeted Camera] →[Follow Camera] を選択します。シーンに"Cinemachine Camera"ゲームオブジェクトが配置されます（**図3.26**）。

3 カメラがボールを追うように設定

まず追従するターゲットをボールにします。「Hierarchy」ウィンドウで"CinemachineCamera"を選択し、「Inspector」ウィンドウの [Tracking Target] に「Hierarchy」ウィンドウの"Sphere"をドロップします。今回回転の追従は行わないので [Rotation Control] は"None"にします。「Cinemachine Follow」コンポーネントの「Follow Offset」のY=5にして、やや上からカメラがボールを見るようにします（**図3.27**）。また「Transform」の「Rotation」のXを"26.565"にして見下ろすようにします。

図3.26　"CinemachineCamera"の配置

図3.27　"CinemachineCamera"の設定



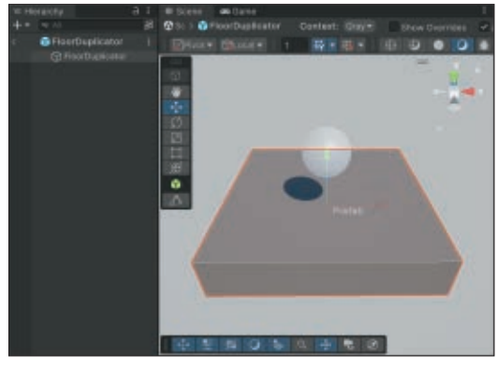
3-4-6 接地判定

現在ジャンプはしますが、空中でもジャンプできてしまいます。これを接地しているときだけジャンプできるようにしてみます。そのため、衝突判定で床に衝突したかを判別する必要があります。Unityには各ゲームオブジェクトに「タグ（Tag）」というものがあり、ゲームオブジェクトの種類を識別するために使うことができるものが用意されています。今回はこのタグを使い、ボールが何に衝突したかを判別してみましょう。タグには任意の文字列を指定できます。

1 プレハブ編集モードに入る

タグを設定するために今回はプレハブを直接編集してみます。「Project」ウィンドウでプレハブ"Prefabs/Floor Duplicator.prefab"のアイコンをダブルクリックするとプレハブ編集モードに入ります。それまで表示されていたシーンが消えて、「Hierarchy」ウィンドウと「Scene」ビューがプレハブ編集モードになります（**図3.28**）。

図3.28　プレハブ編集モード



2 "Floor"タグの追加

新しく床用に"Floor"というタグを追加してみましょう。まず、「Inspector」ウィンドウの [Tag] のドロップダウンをクリックします。開いたメニューの一番下の [Add Tag...] ボタンをクリックします（**図3.29**）。
「Tags & Layers」ウィンドウで、プラスボタンをクリックし、「New Tag Name」に"Floor"と入力して、「Save」ボタンをクリックします（**図3.30**）。

図3.29　タグの設定ウィンドウの表示

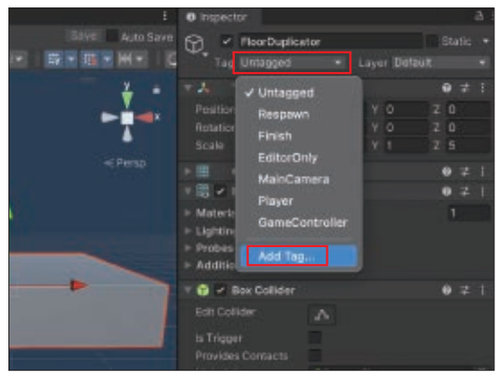
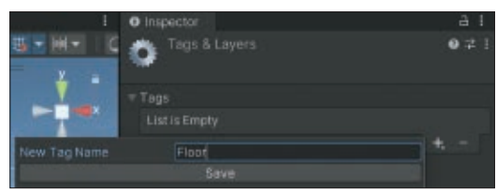


図3.30　"Floor"タグの追加



3 「Nav Mesh Agent」の設定ウィンドウ

[Agent Type] のドロップダウンを開き [Open Agent Settings...] を選択すると設定ウィンドウが表示されます（**図4.19**）。このウィンドウではシリンダーの形が画像で表示されていて設定が感覚的につかみやすくなっています。[Step Height] は、この高さ（設定では75cm）までの段差は乗り越えて行きます。また、[Max Slope] はこの角度（設定では45度）以下の坂は登って行きます。ひとまずこのままにしておきます。

図4.19 「Agent Settings」

コラム　Nav Mesh Agentコンポーネント

「Nav Mesh Agent」コンポーネントはナビメッシュ情報に従って、目的地へ自動的に移動する機能を提供します。また「Nav Mesh Agent」同士がぶつからないように互いに回避する機能も持ちます。

「Nav Mesh Agent」を見ると色々なパラメーターがありますが、ここでまず見ておくのは、キャラクターの半径の"Radius"と"Height"の高さです。ちょうど円筒（シリンダー）の形になります。ナビメッシュをベイク（Bake）することで、この円筒のサイズが通れる場所を探してくれます。[Height] はキャラクターの中心座標（"RobotEnemy"は足元）から頭までの高さで、頭がぶつかるかどうかも調べます（もし、キャラクターの中心座標が足元でない場合は、"Base Offset"で調整できます）。

4-2-8 歩ける場所の作成

次に"Nav Mesh Surface"を使って歩ける場所を作成します。下記手順に従って作成します。

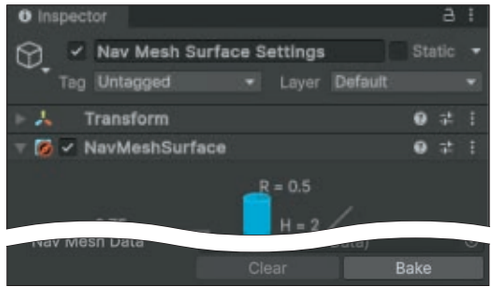
1 歩ける場所管理用ゲームオブジェクトの作成

「Hierarchy」ウィンドウでプラスボタンから「Create Empty」でゲームオブジェクトを作成し、名前を"Nav Mesh Surface Settings"にします（**図4.20**）。

2 「Nav Mesh Surface」コンポーネントの追加

"Nav Mesh Surface Settings"に「Nav Mesh Surface」コンポーネントを追加します（**図4.20**）。

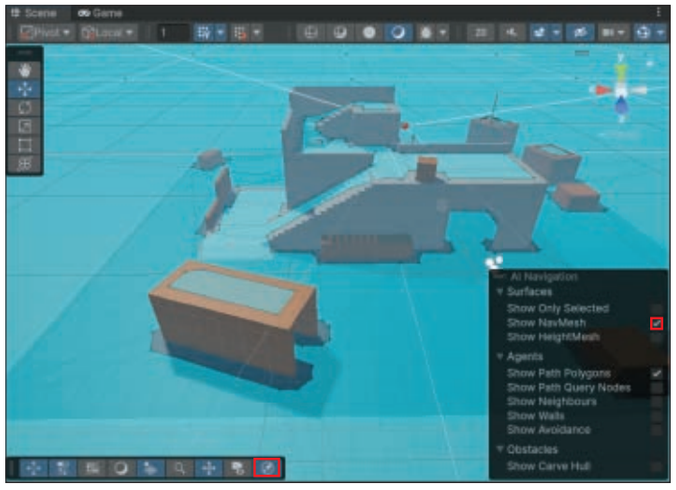
図4.20 「Nav Mesh Surface」コンポーネント



3 ナビメッシュのベイク

右下にある [Bake] ボタンを押します。歩ける場所を自動的に調べて、水色に表示されます（**図4.21**）。なお、右下の「AI Navigation」ウィンドウは、下部の「Overlay Menu」のコンパスのアイコンをクリックすることで表示/非表示を切り替えることができます。また、水色のナビメッシュ領域の表示は、「AI Navigation」ウィンドウの [Show NavMesh] で切り替えることができます。両方とも一旦非表示にしておきます。

図4.21 ナビメッシュが敷かれた状態



4-2-9 敵のスクリプトの作成

敵がプレイヤーを追いかけるようにするためには、「Nav Mesh Agent」に移動先となるプレイヤーの座標を指定する必要があります。そのためのスクリプトを作成していきましょう。

1 敵のスクリプトの作成

「Project」ウィンドウで"Assets/Scripts"フォルダを作成して、そこに"Enemy.cs"スクリプトを作成してください（**リスト4.1**）。navMeshAgent.destinationにプレイヤーの座標をセットすることで、最適な経路で自動的に移動するようになります。

4-4 ゲームクリアを作成しよう

ステージ上のアイテムを、全て集めたらゴール地点が現れて、プレイヤーがそこに到達するとゲームクリアにしたいと思います。

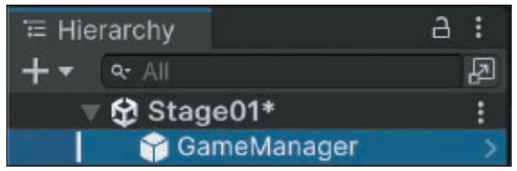
4-4-1 ゲームマネージャーの作成

ゲーム全体の進行を管理するゲームマネージャーというゲームオブジェクトを作成して、まずステージ上のアイテムを全部集めたかを判定しようと思います。

1 ゲームマネージャーゲームオブジェクトの作成

「Hierarchy」ウィンドウの [+] メニューから [Create Empty] を選んで空のゲームオブジェクトを作成してください。名前を"GameManager"にします(**図4.38**)。

図4.38 "GameManager"ゲームオブジェクトの作成



2 ゲームマネージャースクリプトの作成

"Assets/Scripts/GameManager.cs"スクリプトを作成して、"GameManager"ゲームオブジェクトに追加し、**リスト4.4**のスクリプトを入力します。**リスト4.4**はアイテム生成時にAddItem()を呼び、破棄時にRemoveItem()を呼ぶことで全てのアイテムの参照を保持するコードになります。

リスト4.4 GameManager.cs

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

public class GameManager : MonoBehaviour
{
    public int TotalItemCount => itemGoldList.Count;
    private List<ItemGold> itemGoldList = new List<ItemGold>();

    public void AddItem(ItemGold itemGold)
    {
        itemGoldList.Add(itemGold);
    }
    public void RemoveItem(ItemGold itemGold)
    {
        itemGoldList.Remove(itemGold);
    }
}
```

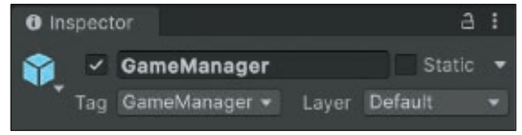
4-4-2 アイテムの登録と解除

アイテムはStart()で"GameManager"のAddItem()を呼び出して登録し、破棄されるときにRemoveItem()を呼び出して登録解除するようにします。そのためにアイテムからこのゲームマネージャーを探すことができるようにタグを追加します。

1 ゲームマネージャーのタグ追加と設定

「Inspector」ウィンドウの「Tag」のプルダウンを開き「Add Tag」を選びます。「Tags & Layers」ウィンドウの「Tags」のプラスボタンをクリックして「New Tag Name」に"GameManager"と入力し、「Save」ボタンを押して保存します。作成したタグ"GameManager"をゲームオブジェクト"GameManager"に設定します(**図4.39**)。

図4.39 Tag="GameManager"の設定



2 ゲームマネージャーにアイテムの登録と解除

リスト4.5の①と②の箇所をItemGold.csに追加します。Start()でアイテム自身をGameManagerに登録し(①)、Destroy()で自身を破棄する前にGameManagerから登録を解除しています(②)。登録を解除したときにリスト"itemGoldList"のサイズが0になったら、アイテムを全て集めたと判定しようと思います。

リスト4.5 "ItemGold.cs"

```
using UnityEngine;

public class ItemGold : MonoBehaviour
{
    //①ここから
    private GameManager gameManager;
```

5-7 プレイヤーの画像を変更しよう

現在プレイヤー１もプレイヤー２も同じ宇宙船なので、見分けがつくようにプレイヤー２の宇宙船の画像を変更してみることにします。プレイヤー２の画像は、インポートした画像にある赤い宇宙船の画像に差し替えてみます。Sprite クラスを変更することによって画像の変更を行います。

5-7-1 スプライト画像の変更

プレイヤー2の宇宙船の画像を変更してみることにします。**リスト5.10**の⑭⑮を追加します。以下コードの解説です。

リスト5.10⑭で"spaceShipSprites"配列に順に、プレイヤー１のスプライト、プレイヤー２のスプライトを「Inspector」ウィンドウでセットします。

"playerIndex"によって使用するスプライトを切り替えています（⑮）。

リスト5.10 プレイヤーの画像の変更

```
(前略)
public class PlayerController : MonoBehaviour
{
(中略)
    private float fireIntervalTimer = 0f;
    //⑪ここまで

    //⑭ここから
    // Spaceshipの画像
    [SerializeField]
    private Sprite[] spaceShipSprites;
    //⑭ここまで

    // 最初に一度呼び出されます
    private void Awake()
    {
        rigidbody2d = GetComponent<Rigidbody2D>();
        //④
        playerIndex = GetComponent<PlayerInput>().playerIndex;// ②

        //⑮ここから
        SpriteRenderer renderer = GetComponent<SpriteRenderer>();
        renderer.sprite = spaceShipSprites[playerIndex];
        //⑮ここまで
(中略)
    }
(後略)
```

コードを入力したら、"Spaceship"プレハブをダブルクリックして編集モードに入ってください。「Inspector」ウィンドウで"Space Ship Sprites"配列のプラスボタンを２回クリックして"Element"を２つ用意します。プレイヤー１の画像"SpaceshipBlue.png"とプレイヤー２の画像"SpaceshipRed.png"を順にセットしてください（**図5.43**）。

実行して、プレイヤー１とプレイヤー２の画像が変わっていることを確認しましょう（**図5.44**）。

図5.43 プレイヤーの画像の設定

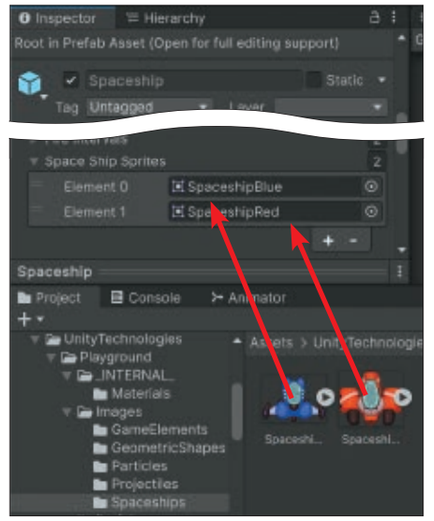


図5.44 プレイヤー1とプレイヤー2の画像