SECTION 02

Unityをインストールしよう

このセクションでは、Unityでゲームを作成するための環境準備を進めていきます。Unity Hubをインストールしてから、Unityのインストールをしましょう。また、Unityを使うためにライセンスの認証が必要なので、Unity IDを取得してライセンスの認証も行います。

◎ Unity Hubをダウンロードする

Unityをインストールするために、Unityの公式サイトからUnity Hubをダウンロードしてインストールします。Webブラウザで以下のURLにアクセスし、＜Unity Hubをダウンロード＞をクリックして、インストールプログラムをダウンロードします（図1-6）。

- **Unity Hubのダウンロード URL**
  https://unity3d.com/jp/get-unity/download

図1-6 Unity Hubダウンロードページ

SECTION 03

この本で作成するゲームを確認しよう

Unityでゲームを作るのですから、Unityの良さを活かして3Dゲームを作っていきましょう。「キャラクターを操作して、障害物のあるマップを進み、ゴールを目指す」という内容のゲームです。しかし、これだけではゲームを作るための情報が足りません。ゲームを作るときには、具体的なルールや遊び方などを確認してから、ゲームの完成イメージを固めましょう。

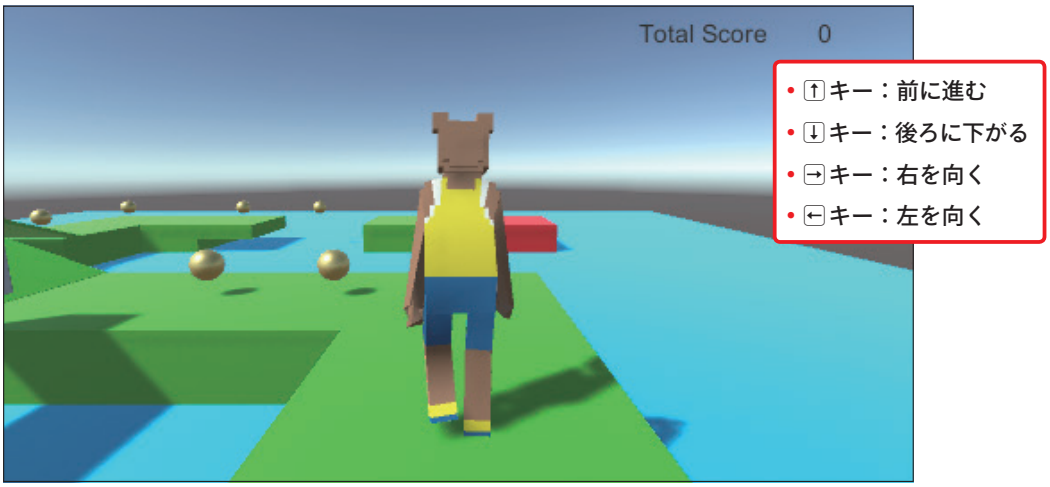
◎ 作るゲームの内容を確認する

ゲームを作り始める前に、ゲームの遊び方やルールなどを確認しましょう。

遊び方

キーボードの←↑→↓キーを押して、キャラクターを操作する（図1-32）。

図1-32 キャラクター操作



SECTION 02

3D空間を把握しよう

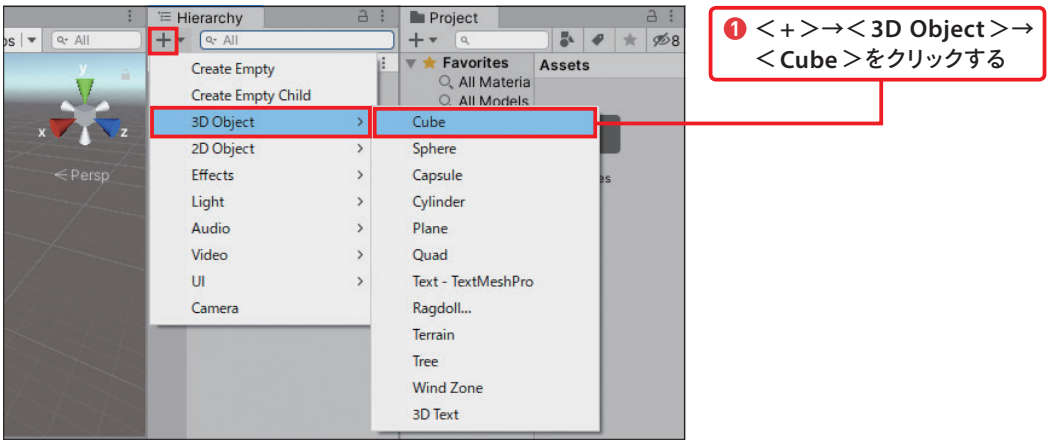
3D空間は、X、Y、Zの3つの軸で表現された空間です。Sceneビューでマウスカーソルを動かしたり、Sceneギズモ（47ページ参照）を操作したりすると、3D空間をさまざまな距離や角度で確認することができます。3D空間の見え方や視点の変更方法を説明します。

◎ 立方体を作成する

ゲームを作り始める前に、3D空間の見え方を確認しましょう。はじめに、3D ObjectのCube（キューブ、立方体）というゲームオブジェクトを作成します。

＜Hierarchy＞ウィンドウの＜＋＞→＜3D Object＞→＜Cube＞をクリックします（図2-10）。

図2-10 3D ObjectのCubeを作成する



＜Scene＞ビューと＜Hierarchy＞ウィンドウに、ゲームオブジェクト＜Cube＞が追加されます（図2-11）。＜Hierarchy＞ウィンドウからゲームオブジェクトを削除すると、＜Scene＞ビューからも消えてしまうので注意してください。

◎ モデルの設定をする

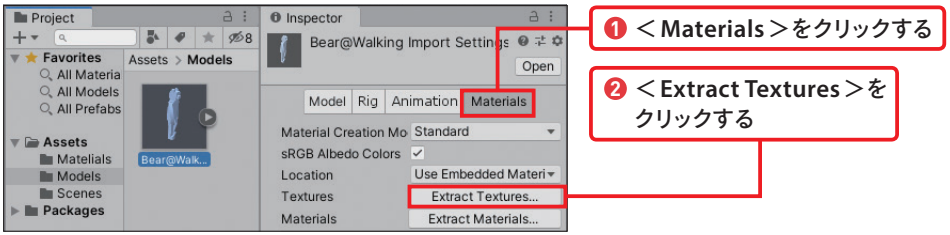
読み込んだBear@Walkingは、全身薄い青色の状態になっています。ゲームオブジェクトとしてSceneビューに配置する前に、いくつか設定を変更していきましょう。

テクスチャの抽出

テクスチャとは、物体の表面を表現する画像のことをいいます。FBX形式のファイルはテクスチャ情報を内包していますが、モデルファイルを読み込んだだけでは、Unityはテクスチャを認識できません。Bear@Walkingからテクスチャ情報を抽出（Extract）して、反映させましょう。

＜Bear@Walking＞の＜Inspector＞ウィンドウを表示してください。＜Inspector＞ウィンドウの＜Materials＞をクリックして、＜Extract Textures＞をクリックします（図3-4）。

図3-4 テクスチャの抽出



抽出するテクスチャを保存するフォルダを選択しますが、初期状態で表示される場所に保存すれば問題ありません。そのまま＜フォルダーの選択＞をクリックしてください。テクスチャが抽出されて、プロジェクトに読み込まれると、＜Bear@Walking＞に色がつきます（図3-5）。

図3-5 テクスチャを保存するフォルダを選択

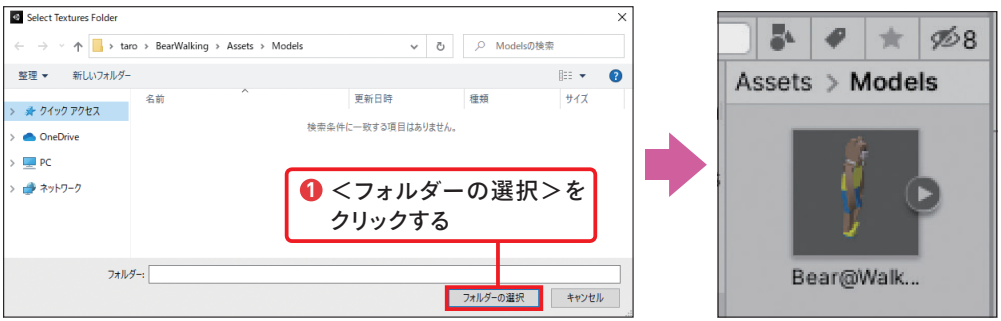
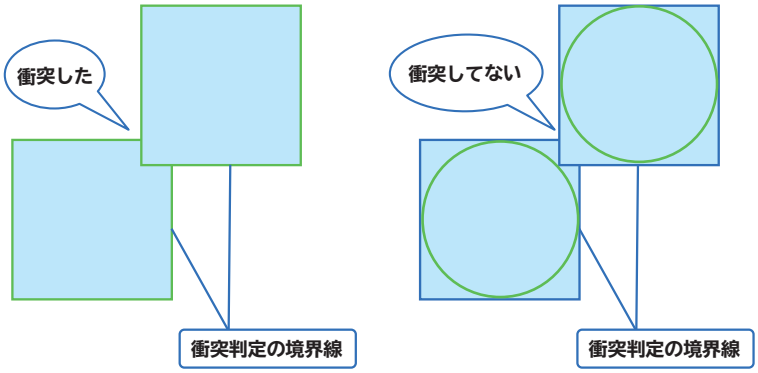


図3-16 衝突判定の境界線



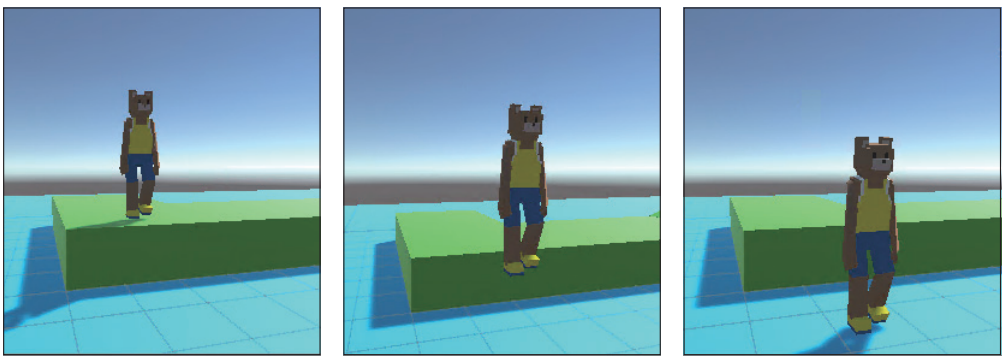
Colliderコンポーネントがアタッチされていないと衝突判定が行われないため、床や壁などすり抜けてはいけないゲームオブジェクトをすり抜ける現象が発生してしまいます。そのため、PlaneやCubeなどの3D Objectには、あらかじめ形に適したColliderがアタッチされた状態で作られます。

Bear@Walkingのような読み込んだファイルを元に、Sceneビューに配置したゲームオブジェクトには、Colliderコンポーネントがアタッチされていない場合があります。ゲームオブジェクトの衝突判定をしたいときには、形にあったColliderコンポーネントをアタッチする必要があります。

Rigidbodyコンポーネント

Rigidbodyはゲームオブジェクトに、物理法則に従った動きをさせるコンポーネントです。物理法則に従った動きとは、重さや摩擦（滑りにくさ）、重量などの影響を受けた動きです。たとえば、「道を踏み外したら床に落ちる」のは、重力の影響を受けて落下する動きです（図3-17）。

図3-17 重力の影響を受けて落下する



SECTION 03

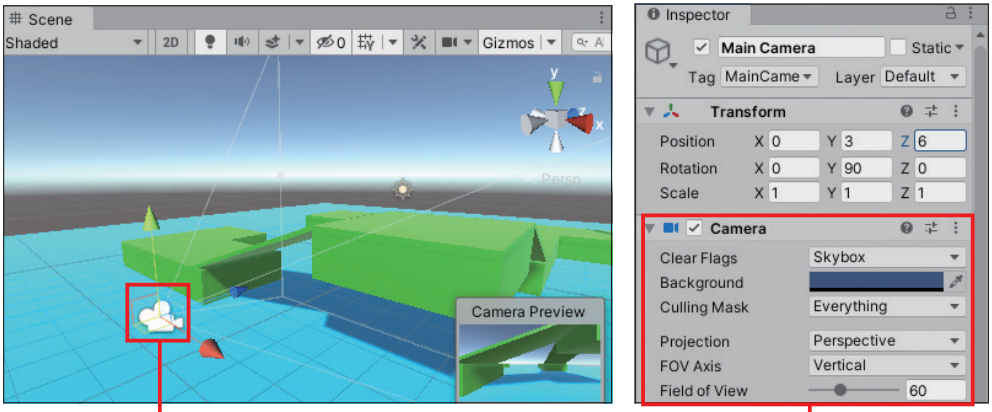
カメラでキャラクターを確認してみよう

ゲームを実行したときに表示されるのは、Gameビューに表示されている範囲です。Gameビューに表示される範囲は、カメラの位置によって決まります。カメラが固定の位置だとキャラクターを操作しづらいので、キャラクターを動かしたときに一緒に移動するように、キャラクターとカメラを親子関係の設定をします。

◎ カメラとは

GameビューにSceneビューの3D空間を映し出すゲームオブジェクトを**カメラ**と呼びます。ゲームを実行したときに表示する範囲を決めるのは、Camera（カメラ）コンポーネントがアタッチされたゲームオブジェクトです。プロジェクトを作成した段階で、Main Cameraという名前のゲームオブジェクトが、Sceneビューに配置されています（図3-28）。ゲームオブジェクトMain CameraにCameraコンポーネントが付いているため、Gameビューに3D空間が映し出されているのです。

図3-28 カメラについて



SECTION 01

アニメーションを設定しよう

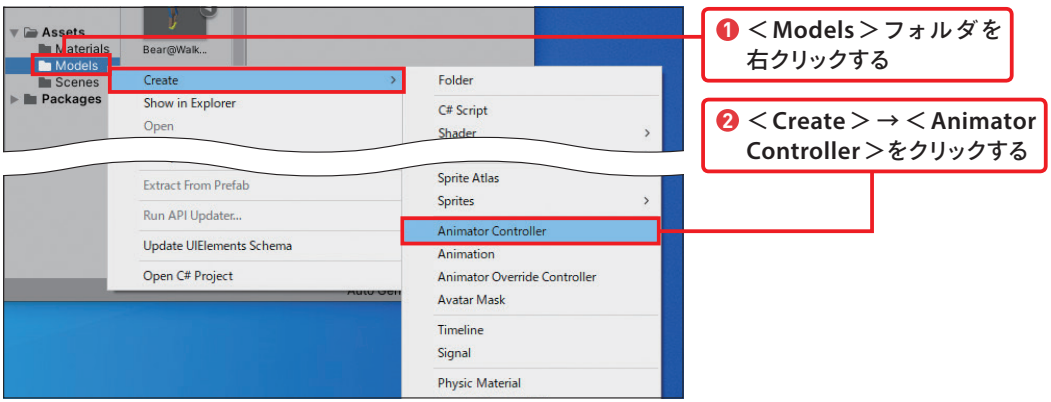
Chapter3では、キャラクターの3Dモデルをプロジェクトに読み込み、コンポーネントやカメラの設定を行いました。このChapterでは、キャラクターのアニメーションを再生する設定と、カーソルキーで操作するためのスクリプトを書いていきます。

◎ キャラクターのアニメーションを設定する

Unity上で、キャラクターなどのアニメーションを表示するためには、Animationコンポーネントにアニメーターコントローラー（Animator Controller）というアニメーションを制御するためのアセットを設定する必要があります。ゲームオブジェクトBear@Warkingには、あらかじめAnimationコンポーネントがアタッチされています（72ページ参照）。「BearAnimationController」という名前のアニメーターコントローラーを作って、Bear@WarkingのAnimationコンポーネントに設定しましょう。

まずアニメーターコントローラーを作成します。＜Project＞ウィンドウの＜Models＞フォルダを右クリックして、＜Create＞→＜Animator Controller＞をクリックします（図4-1）。

図4-1 Animator Controllerを作成



◎ カーソルキー操作のスクリプトを書く

キャラクターを操作するためのスクリプトをBearController.csに書いていきましょう。カーソルキーの入力に応じて、ゲームオブジェクトBear@Warkingを動かす処理を作っていきます。

◉ 変数宣言とStartメソッドの処理

BearControllerクラスで使う変数を宣言して、Startメソッドの処理を書きます（リスト4-2）。

リスト4-2 変数宣言とStartメソッド（BearController.cs）

```
001:  using System.Collections;
002:  using System.Collections.Generic;
003:  using UnityEngine;
004:
005:  public class BearController : MonoBehaviour
006:  {
007:      private float speed = 3f;
008:      private float verticalInput = 0f;
009:      private Rigidbody bearRigidbody;
010:
011:      void Start()
012:      {
013:          bearRigidbody = GetComponent<Rigidbody>();
014:      }
```

変数を宣言する

Rigidbodyコンポーネントを取得する

数値などのデータを「値」と呼びます。**変数**と呼ばれるものに名前を付けて、値を記憶しておくことができます。事前に繰り返し使うことがわかっている値は、変数を用意しておくと便利です。また、変数を作ることを**宣言する**といいます。変数は、数値や文字列など、値の型（タイプ）にあわせて宣言します（リスト4-3）。

リスト4-3 変数の宣言例

```
float val = 0.0f;
```

float型の変数valを宣言して、0.0fを代入する

float（フロート）型は、浮動小数点と呼ばれる値を入れるための型です。ここではfloat型のvarという変数を宣言しています。

変数に値を入れることを**代入**といい、変数名と値を「=（イコール）」で繋いで書きます。「=」は、C#では代入するという意味になります。float型の変数に代入する値は最後に「f」を付ける必要があります。

SECTION 01

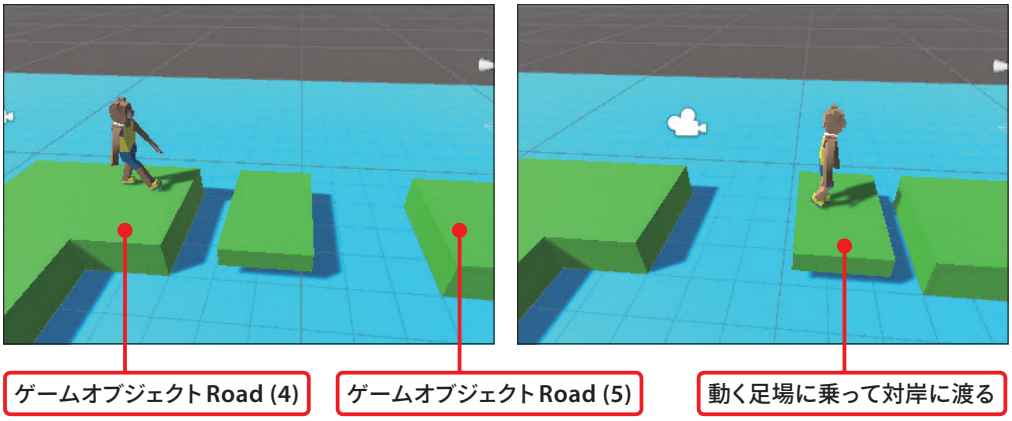
動く足場を作ろう

キャラクターが何もない道を進むだけではゲームとして面白くないので、進行を妨害する障害物を作りましょう。このセクションでは、一定間隔で移動を繰り返す足場を作ります。「一定間隔で移動を繰り返す」という処理のスクリプトを書いて、キャラクターを動く足場で対岸に渡すような仕組みにします。

◎ 動く足場を作る

63ページで道を作りましたが、ゲームオブジェクトRoad (4)とRoad (5)の距離は離れている状態です。ゲームオブジェクトRoad (4)とRoad (5)の間に、動く足場を配置しましょう（図5-1）。Bear@Walkingが足場に乗ると、足場の動きにあわせてBear@Walkingも動く（対岸に渡る）ようにします。「足場が一定間隔で移動する」という動き方は、スクリプトで制御します。

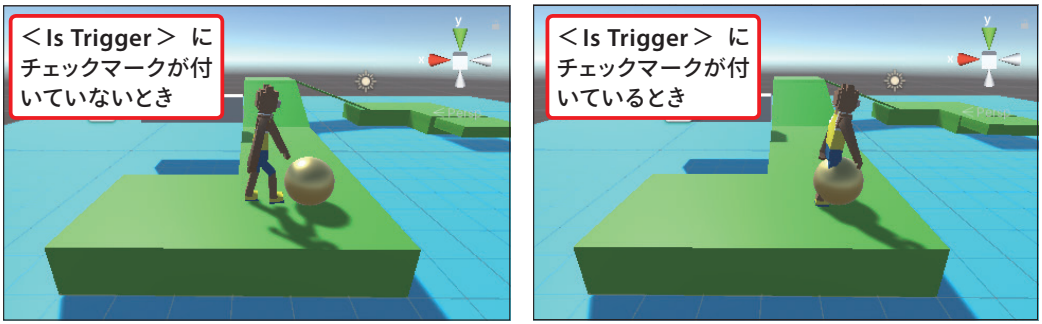
図5-1 動く足場



Is Triggerを有効にする

Colliderコンポーネント（75ページ参照）には、**Is Trigger**（イズトリガー）という設定項目があります。Colliderコンポーネントの＜Is Trigger＞にチェックマークを付ける（有効にする）と、他のColliderコンポーネントがアタッチされたゲームオブジェクトと衝突してもお互いに影響を受けません。つまり、すり抜けられるということです（図6-8）。

図6-8 Is Triggerを有効にする



拾えるアイテムなどは、衝突したときに前に進めなくなってしまうと不自然なので、すり抜けられるようにColliderコンポーネントの＜Is Trigger＞を有効にします。＜Is Trigger＞が有効な状態で、ゲームオブジェクトと衝突すると、「OnTriggerEnter」というメソッドが呼ばれるようになります。衝突したときに何か処理をしたい場合は、アタッチするスクリプトにOnTriggerEnterメソッドを作成して、実行したい処理を書きましょう（図6-9）。

図6-9 OnTriggerEnterメソッドの呼び出し