4-1 ゲームの設計を考えよう

3章はサンプル要素が強かったので、4章からはもう少しゲームっぽくするために、UIを表示したり、効果音を鳴らしたりと、さまざまな要素を取り入れていきます。ただし、一度にすべてを学ぶのは大変です。そこで、この章では3章よりも少し複雑なサンプルを作りながら、UIの表示や監督オブジェクトの作成方法などを一歩ずつ学びましょう。

4-1-1 ゲームの企画を作る

4章では、車を旗のぎりぎり手前で止める「寸止めゲーム」を作ります。完成図はFig.4-1のようになります。ゲーム開始直後、画面左下に車が表示されており、画面をスワイプすると車が走り始め、徐々に減速して止まります。スワイプの長さを変えることで、車の走行距離を変えることができます。画面右下には旗を表示して、画面中央には車と旗の距離を表示します。

Fig.4-1 これから作るゲームの画面

4-1-2 ゲームの部品を考える

Fig.4-1のゲームイメージをもとに、ゲームの設計を考えてみましょう。今回も3章と同様に、次の5ステップで考えていきます。

Step❶ 画面上のオブジェクトをすべて書き出す

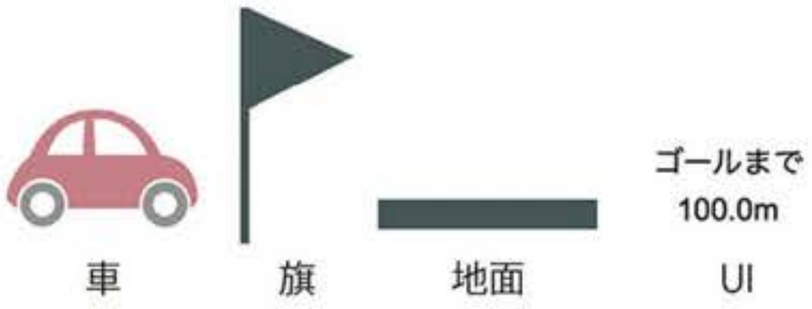
Step❷ オブジェクトを動かすためのコントローラスクリプトを決める

Step❸　オブジェクトを自動生成するためのジェネレータスクリプトを決める
Step❹　UIを更新するための監督スクリプトを用意する
Step❺　スクリプトを作る流れを考える

ステップ① 画面上のオブジェクトをすべて書き出す

まずは画面上にあるオブジェクトを書き出します。Fig.4-1を見て、どのようなオブジェクトがあるかを考えてみましょう。ここでは「車」と「旗」、見落としがちですが「地面」、「距離を表示するUI」が挙げられます。

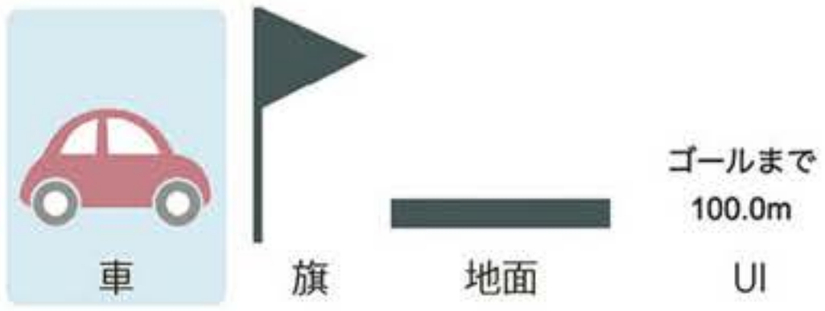
Fig.4-2 画面上のオブジェクトを書き出す



ステップ② オブジェクトを動かすためのコントローラスクリプトを決める

次に、ステップ①で書き出したオブジェクトのなかから、動くオブジェクトを探しましょう。車は走るので動くオブジェクトに分類できそうです。旗と地面は動きません。車と旗の距離を表示するUIの表示内容は変化しますが、**表示場所が移動するわけではないので、動くオブジェクトには含めません。**

Fig.4-3 動くオブジェクトを探す



3章でも書いた通り、動くオブジェクトには、オブジェクトの動きを制御する台本（スクリプト）が必要です。ここでは車の動きを制御するため、「車コントローラ」が必要になります。

必要なコントローラスクリプト
・車コントローラ

ステップ③ オブジェクトを自動生成するためのジェネレータスクリプトを決める

このステップではゲームプレイ時に次々と生成されるオブジェクトを探します。今回もゲームを実行している時に生成されるオブジェクトはなさそうです。

ステップ④ UIを更新するための監督スクリプトを用意する

演劇の場合、監督が劇の進行状況を把握して俳優達に指示を出します。これと同様に、ゲームを滞りなく進めるためには監督スクリプトが必要です。監督スクリプトは**ゲーム全体に目を通してUIを書き換えたり、ゲームオーバを判定したりします。**今回のゲームでは、車と旗の距離をUIに表示する必要があるため、監督スクリプトを作成します。

Fig.4-4 監督スクリプトの役割

必要な監督スクリプト

・UIを書き換えるための監督

ステップ⑤ スクリプトを作る流れを考える

3章と同様、各スクリプトをもとにして、どのようにゲームを作っていくかを考えます。基本的には**「コントローラスクリプト」→「ジェネレータスクリプト」→「監督スクリプト」**の順番で作っていくのでしたね。

Fig.4-5 スクリプトを作る流れ

4章で作るスクリプトは「車コントローラ」と「ゲームシーン監督」の2つです。**この2つのスクリプトさえ作れば、本章のゲームが動くようになります。**

車コントローラ

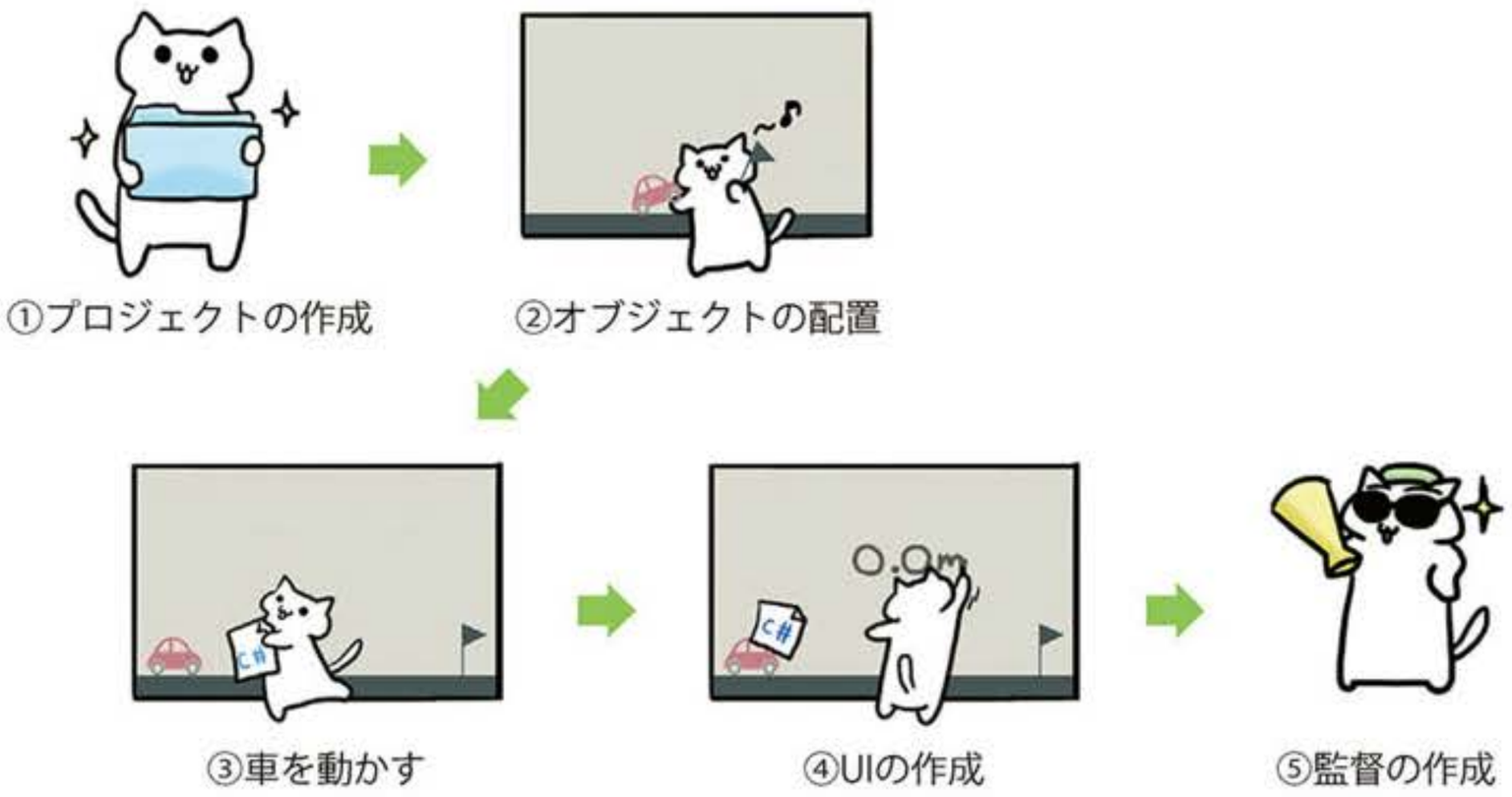
スワイプで走り始めるようにし、徐々に減速して停止させます。また、スワイプの長さによって走る距離が変化するようにします。

ゲームシーン監督

「車」と「旗」の座標を調べて、その差分を距離としてUIに表示します。

4章で作成する「車」は、見た目は違いますが3章で作成したルーレットとほぼ同じ手順で作ることができます。**動くオブジェクトの作り方は、どのようなオブジェクトでも変わらない**ので、繰り返し練習して流れを覚えましょう！ 今回作るゲームは動くオブジェクトに加えて、UIも使っています。UIの作成手順も動くオブジェクトと同様、ゲームによって変わることはないので、しっかりと覚えましょう。全体の流れをまとめるとFig.4-6のようになります。

Fig.4-6 ゲームを作る流れ



4-2 プロジェクトとシーンを作成しよう

4-2-1 プロジェクトの作成

プロジェクトの作成から始めましょう。Unity Hubを起動した時に表示される画面から新規作成をクリックするか、画面上部のツールバーからFile→New Projectを選択してプロジェクトの設定画面を開きます。

プロジェクト名は「SwipeCar」にします。テンプレートの項目は「2D」を選択します。画面右下の青色の作成ボタンをクリックすると、指定したフォルダにプロジェクトが作成されUnityエディタが起動します。

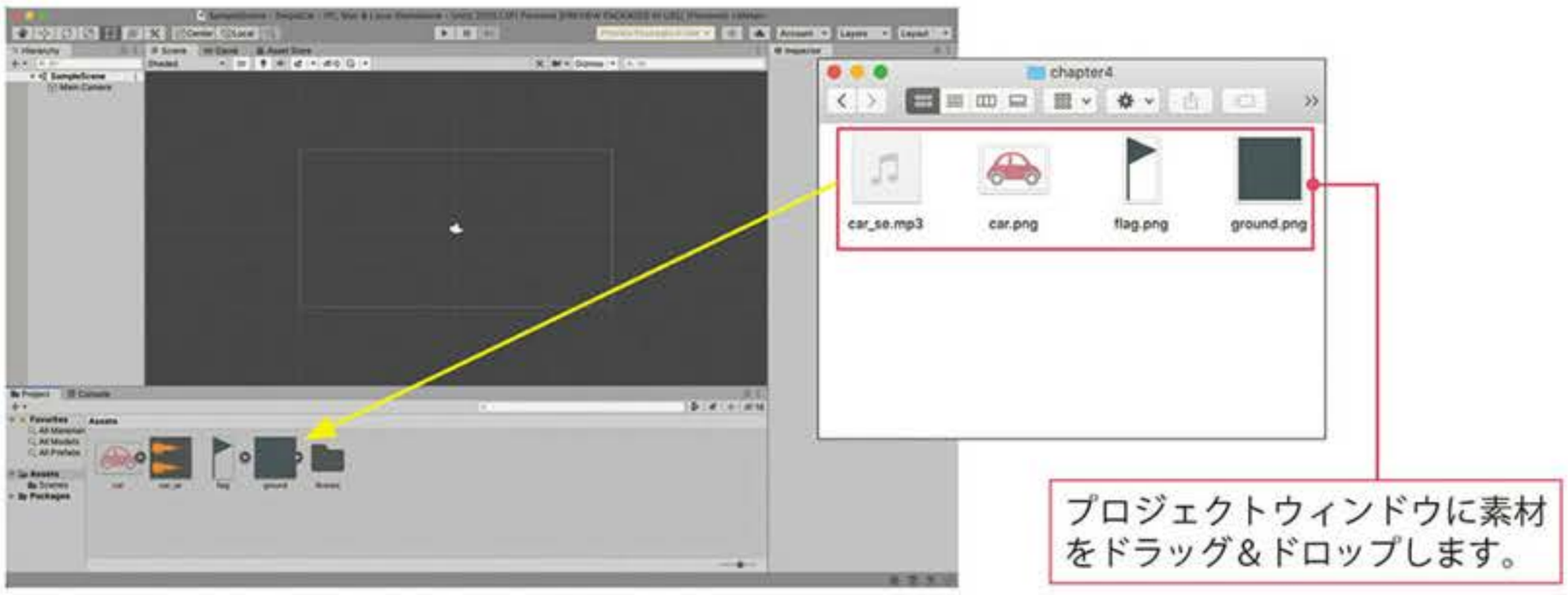
プロジェクトの作成→SwipeCar
テンプレートの選択→2D

プロジェクトに素材を追加する

Unityエディタが起動したら、今回のゲームで使用する素材をプロジェクトに追加しましょう。ダウンロードした素材ファイルの「Chapter4」フォルダを開いて、中身の素材をすべてプロジェクトウィンドウにドラッグ＆ドロップしてください。

URL 本書のサポートページ
https://isbn2.sbcr.jp/06657/

Fig.4-7 素材を追加する



使用する各ファイルの役割は以下の通りです。

Table 4-1 使用する素材の形式と役割

ファイル名	形式	役割
car.png	pngファイル	車の画像
ground.png	pngファイル	地面の画像
flag.png	pngファイル	旗の画像
car_se.mp3	mp3ファイル	車の効果音

Fig.4-8 使用する素材

実行時の描画設定

実行した時のフレームの描画速度をモニターの更新速度に合わせるため、Gameタブをクリックして画面サイズ設定（Aspect）のドロップダウンリストを開き、VSync（Game view only）にチェックを入れてください。詳しい手順は3章を参照してください（122ページ）。

4-2-2 スマートフォン用に設定する

スマートフォン用にビルドするための設定をします。ツールバーからFile→Build Settingsを選択します。Build Settingsウィンドウが開くので左下のPlatform欄から「iOS（Android用にビルドする場合はAndroid）」を選択して、Switch Platformボタンをクリックします。詳しい手順は3章を参照してください（122ページ）。

画面サイズの設定

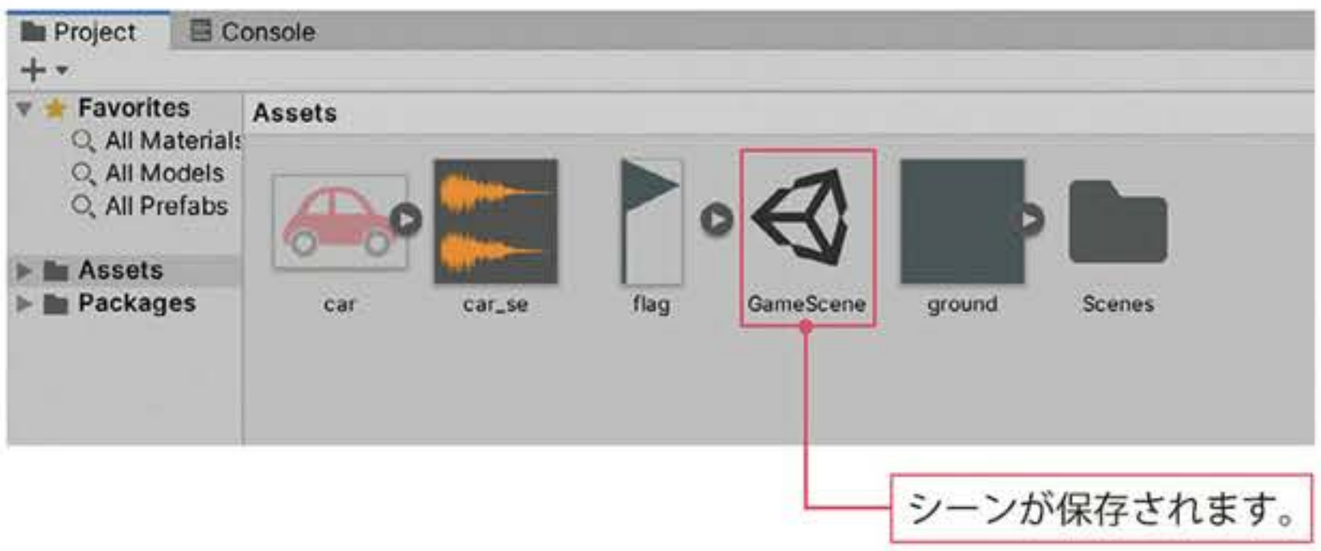
続けて、ゲームの画面サイズを設定しましょう。シーンビューのGameタブをクリックし、ゲームビュー左上にある画面サイズ設定のドロップダウンリストを開き、対象となるスマートフォンの画面サイズに合ったものを選択してください（本書では「iPhone X/XS 2436×1125 Landscape」を選択しています）。詳しい手順は3章を参照してください（123ページ）。

4-2-3 シーンを保存する

続いてシーンを作成します。ツールバーからFile→Save Asを選択し、シーン名を「Game Scene」として保存してください。保存できるとUnityエディタのプロジェクトウィンドウに、シーンのアイコンが出現します。詳しい手順は3章を参照してください（124ページ）。

シーンの作成→GameScene

Fig.4-9 シーンの作成までを行った状態



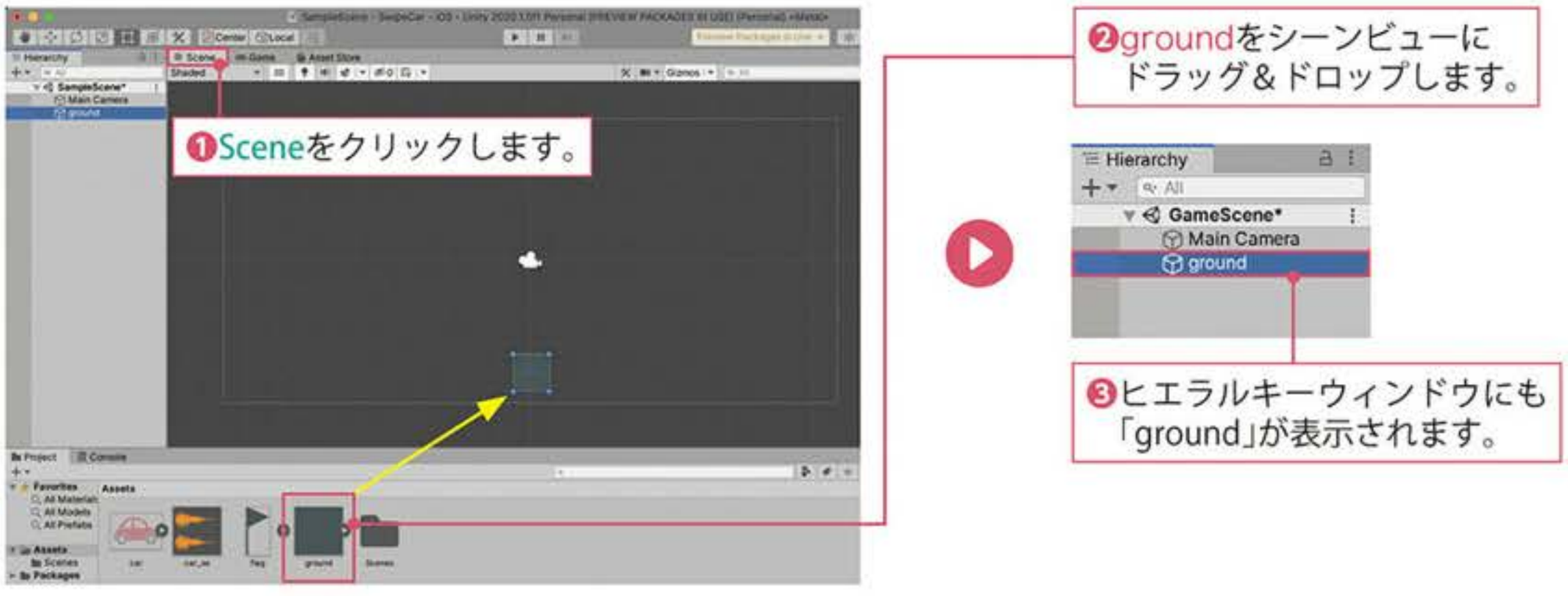
4-3 シーンにオブジェクトを配置しよう

4-3-1 地面の配置

4-3節では、ゲームに必要なオブジェクトをシーンに配置していきます。ここで配置するのは、「地面」「車」「旗」の3つです。スプライトの配置の流れは3章と同じですので、サクッと進めましょう（2Dゲームで使う画像のことを「スプライト」と呼びます）。

まずはシーンに地面を配置します。Sceneタブをクリックして、プロジェクトウィンドウから地面の画像「ground」をシーンビューにドラッグ＆ドロップしてください。シーンビューにスプライトが表示され、ヒエラルキーウィンドウにも「ground」と表示されます。

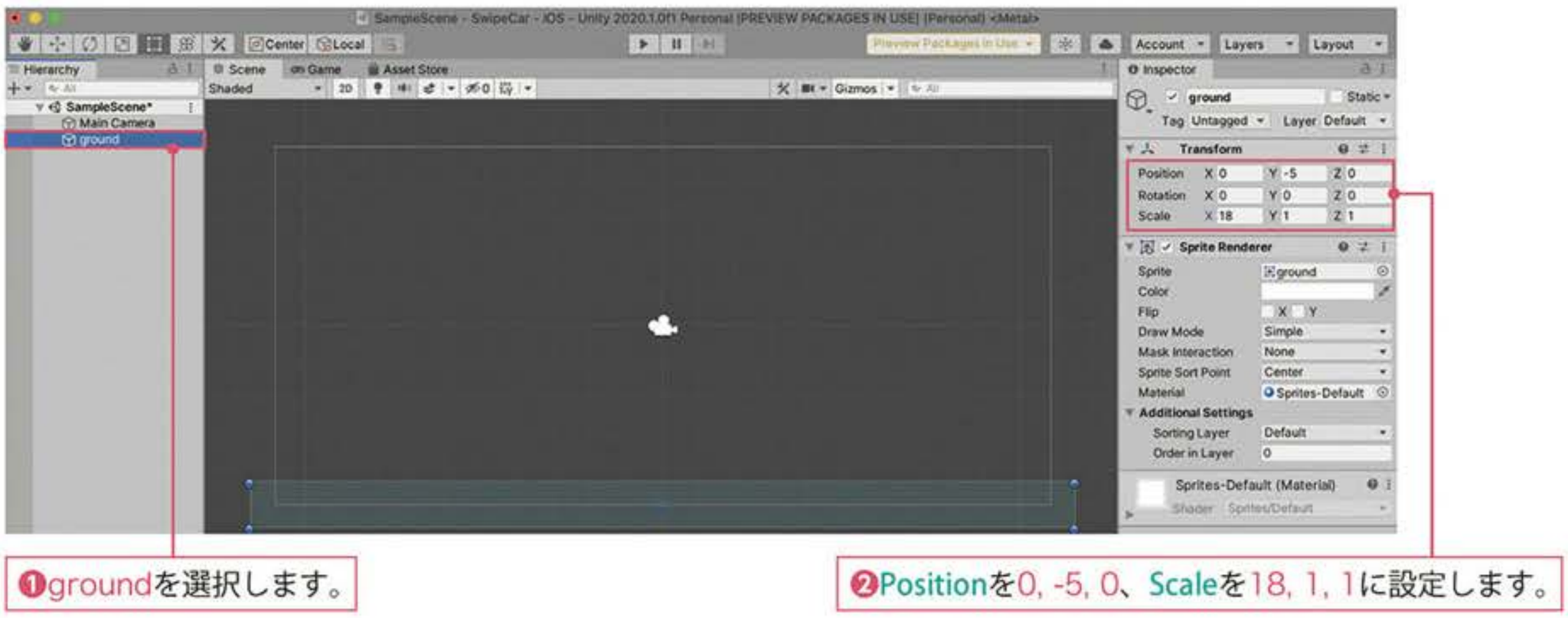
Fig.4-10 地面をシーンに追加する



続いて**地面の位置と大きさをインスペクターで調整します。**ヒエラルキーウィンドウでgroundを選択すると詳細情報がインスペクターに表示されるので、配置する座標と大きさを指定します。Transform項目のPositionに「0, -5, 0」と設定し、Scaleに「18, 1, 1」と設定します（Fig.4-11）。

インスペクターの値に合わせて、シーンビュー上の地面の位置と大きさが変わります。

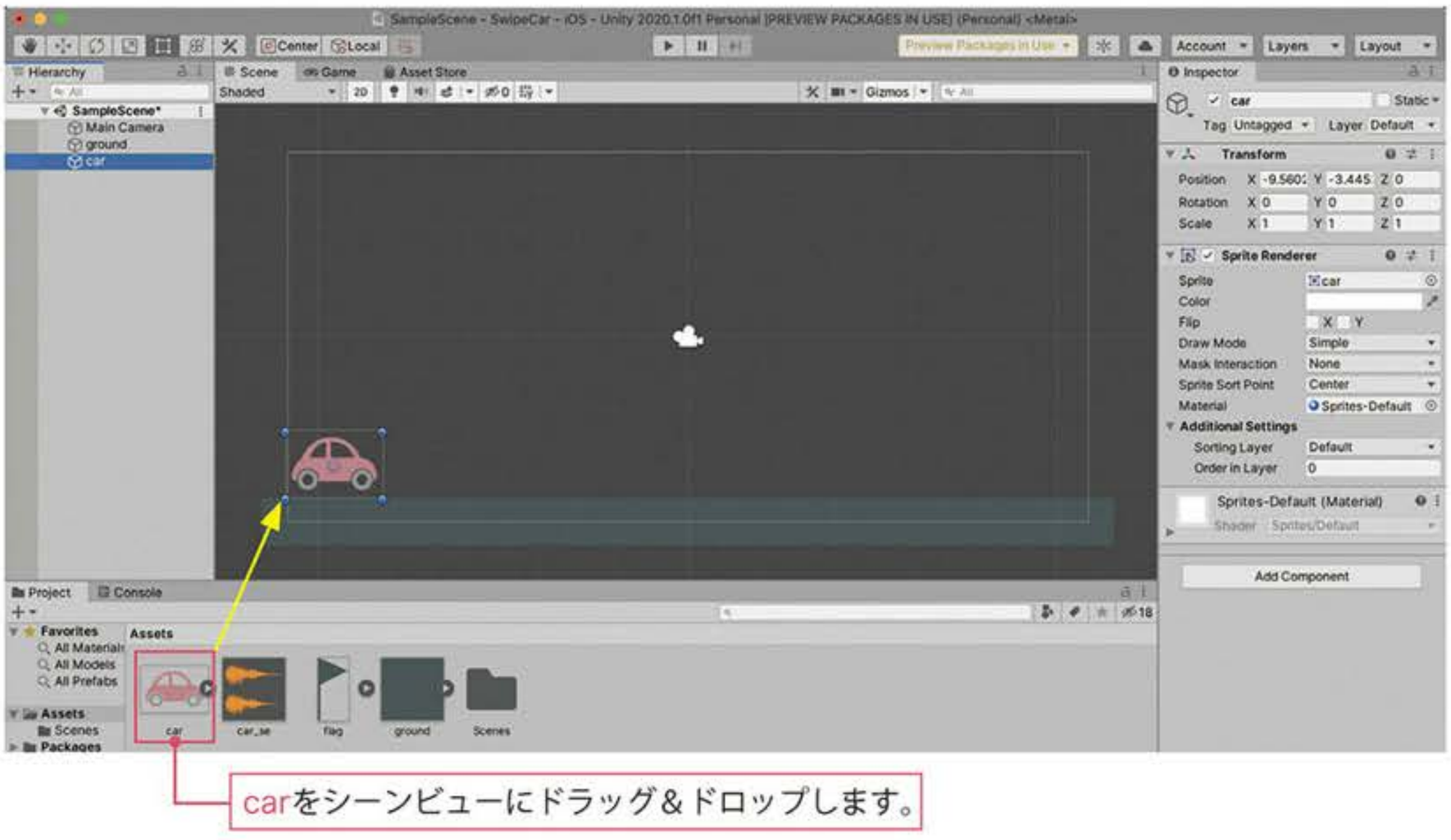
Fig.4-11 地面の座標と大きさを変更する



4-3-2 車の配置

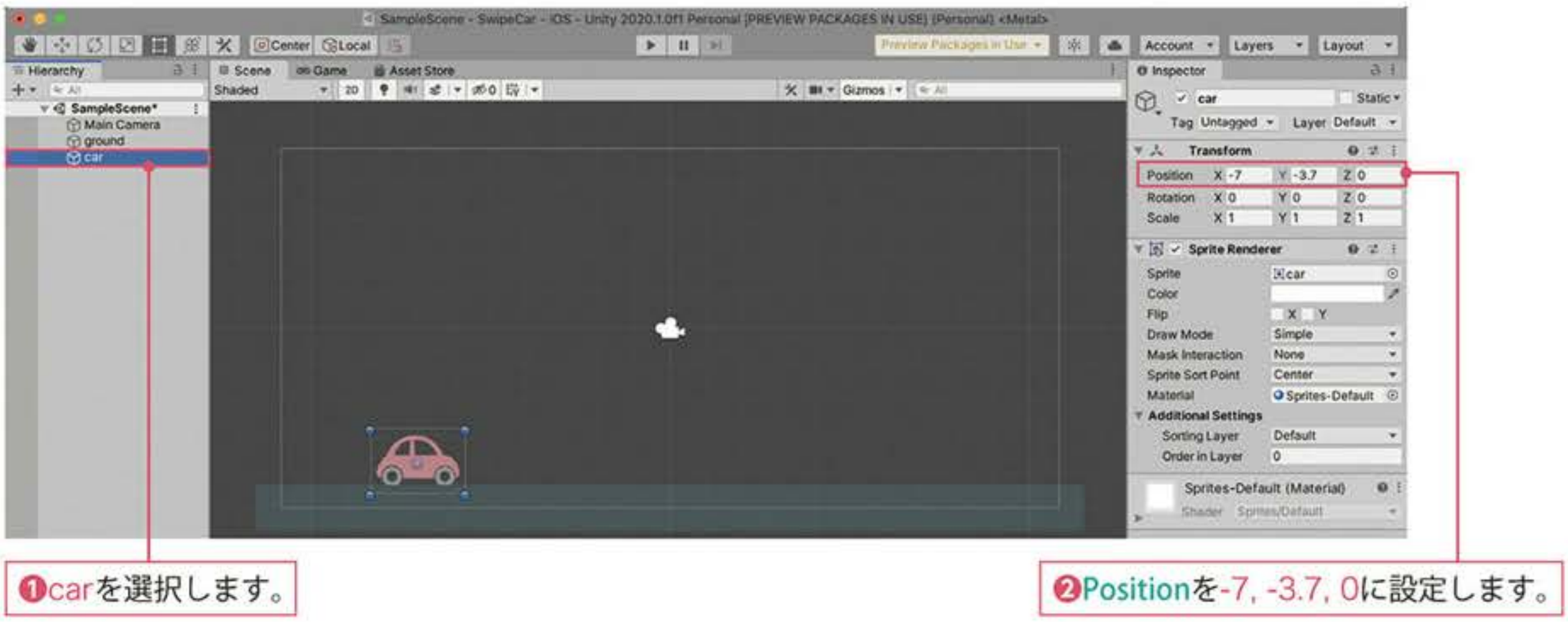
次に「車」を配置します。車の画像「car」をシーンビューにドラッグ＆ドロップしてください。シーン中のスプライトに対応する「car」が、ヒエラルキーウィンドウにも表示されます。

Fig.4-12 車をシーンに追加する



続いて車の位置をインスペクターで調整します。先ほどと同様、ヒエラルキーウィンドウでcarを選択し、インスペクターから座標を指定します。Transform項目のPositionを「-7, -3.7, 0」に設定します。

Fig.4-13 車の座標を変更する

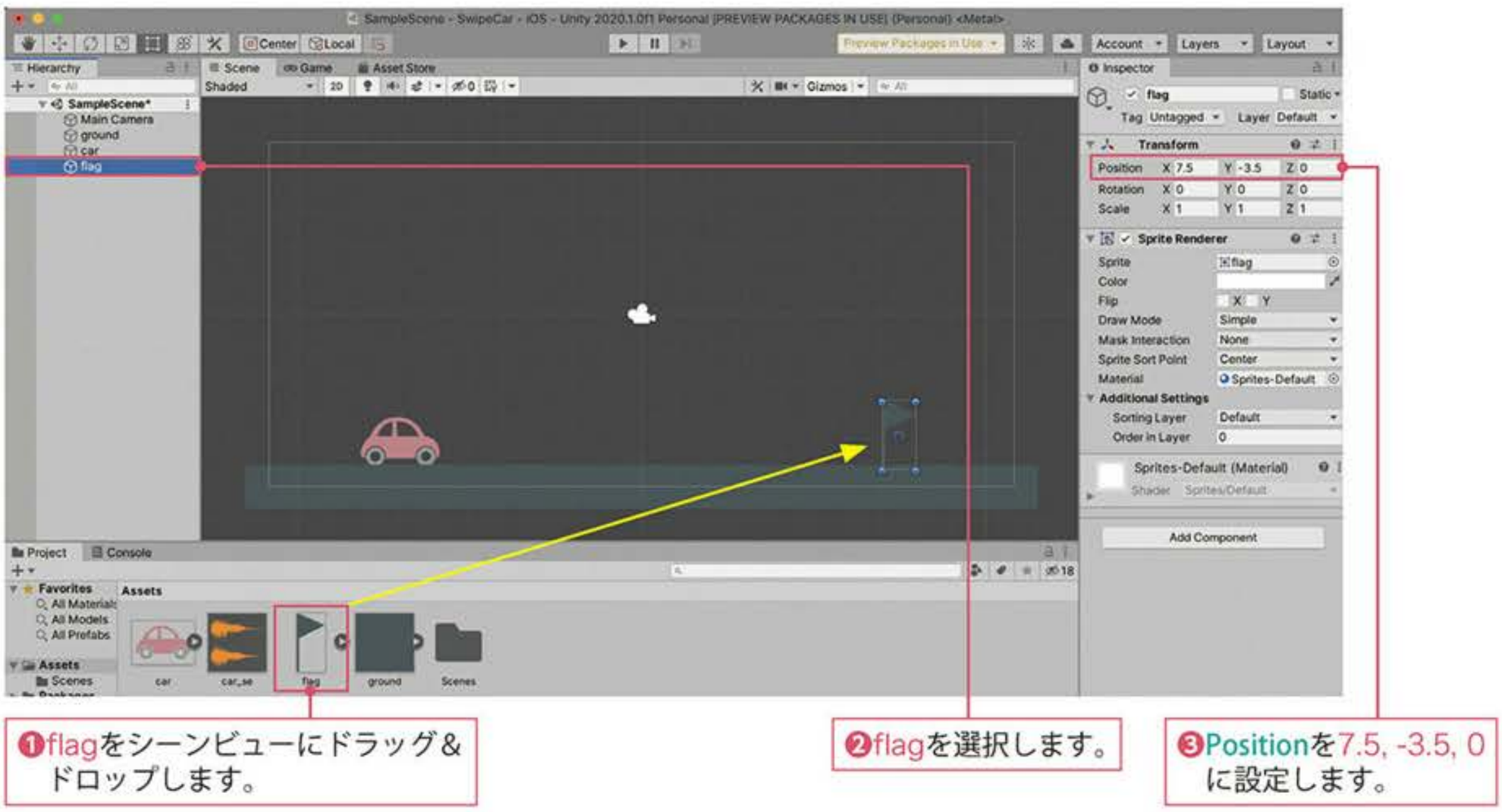


4-3-3 旗の配置

最後に「旗」を配置します。旗の画像「flag」をシーンビューにドラッグ＆ドロップしてください。旗のスプライトに対応する「flag」がヒエラルキーウィンドウにも表示されます。

続いて旗の位置をインスペクターで調整します。ヒエラルキーウィンドウでflagを選択して、インスペクターで座標を指定します。Transform項目のPositionを「7.5, -3.5, 0」に設定します。

Fig.4-14 旗をシーンに追加して座標を調整する



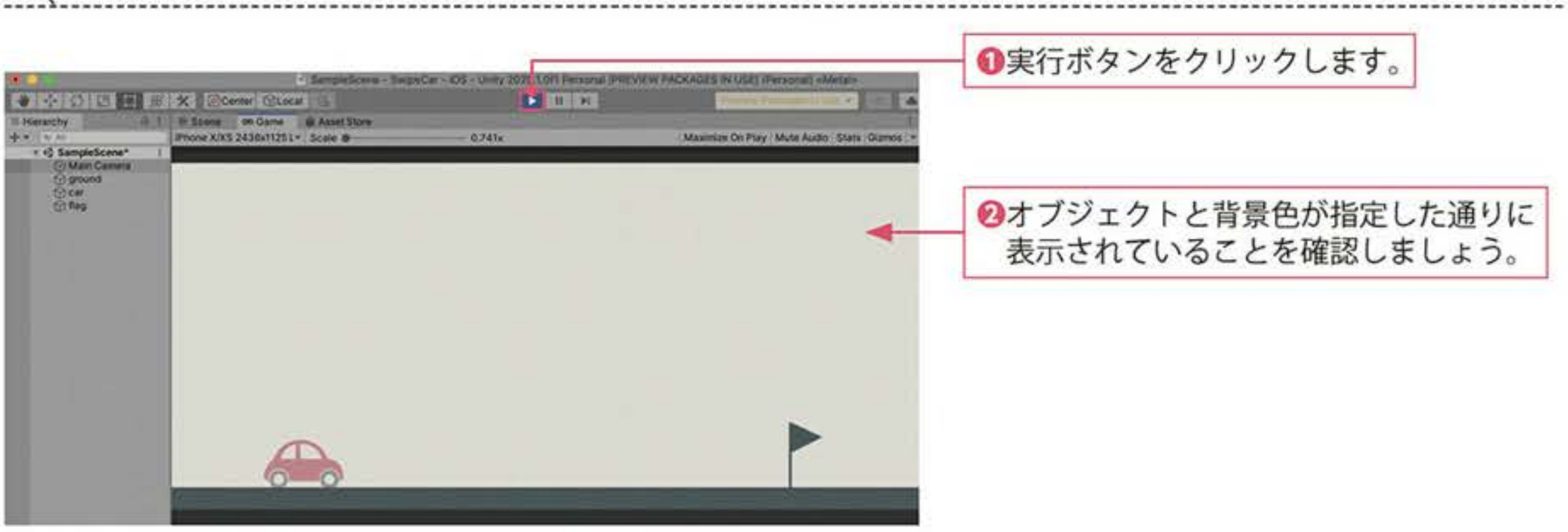
4-3-4 背景色の変更

背景色が青色だと美しくないので、もう少し淡い色にしましょう。**背景色はカメラオブジェクトのインスペクターから変更する**ことができました（128ページ）。ヒエラルキーウィンドウでMain Cameraを選択し、インスペクターからCamera項目のBackgroundのカラーバーをクリックして、Colorウィンドウを表示します。今回は背景色に「DEDBD2」を指定しましょう。

Fig.4-15 背景色を変更する

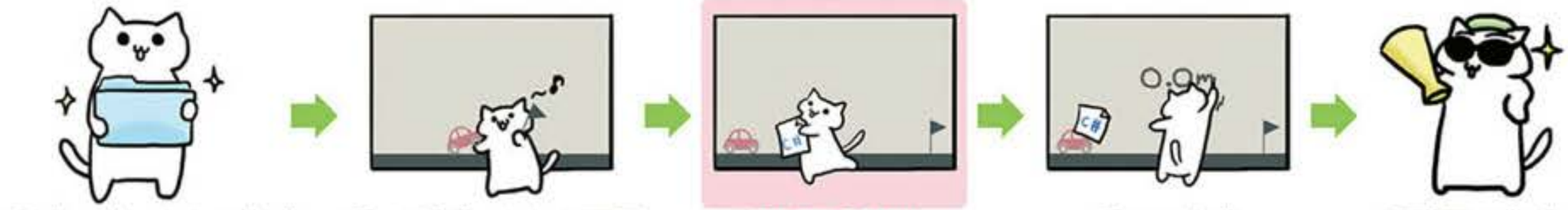
ここまでできたら、さっそくゲームを実行して結果を確認してみましょう。Unityエディタ上部の実行ボタンを押してください。配置したオブジェクトと設定した背景色がちゃんと表示できていますね！

Fig.4-16 ゲームを実行してみる



4-3節ではゲームに必要な画像を配置して、背景色も変更しました。さすがに3回も「画像の配置」→「インスペクターで座標設定」を繰り返すと、Unityの操作にも慣れてきたのではないでしょうか。この調子で次は車を動かしてみましょう！

4-4 スワイプで車を動かす方法を考えよう



4-4-1 車のスクリプトを作る

4-4節では車を動かします。車を動かすには、車の動かし方を書いた台本（コントローラスクリプト）が必要です。そこで「動くオブジェクトの作り方」にしたがって、車の台本を作成して車オブジェクトにアタッチしましょう。

動くオブジェクトの作り方 重要!

❶シーンビューにオブジェクトを配置します。
❷オブジェクトの動かし方を書いたスクリプトを作成します。
❸作成したスクリプトをオブジェクトにアタッチします。

ここまでの手順でオブジェクトの配置はできているので、スクリプトの作成から行います。

Fig.4-17 車の台本を作る

最終的にはスワイプした長さに応じて車の移動距離を決めたいのですが、最初から難しいことをしようとすると、何から始めてよいかわからなくなってしまいます。そこで、**まずは画面をクリックすると車が動き出し、徐々に減速して止まる**ところまで作成します。車が走り出して止まるという動作は、3章のルーレットが回転して止まる動作と同様の仕組みで作れそうですね！

では、「マウスをクリックすると車が走り出して止まる」スクリプトを作成しましょう。

プロジェクトウィンドウ内で右クリックし、Create→C# Scriptを選択してスクリプトファイルを作成します。作成したスクリプトのファイル名は「CarController」に変更します。

スクリプトの作成→CarController

ファイル名を変更したらプロジェクトウィンドウの「CarController」をダブルクリックして開き、開いたファイルにList 4-1のスクリプトを入力・保存してください。

List 4-1 「マウスをクリックすると車が走り出して止まる」スクリプト

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class CarController : MonoBehaviour
6  {
7      float speed = 0;
8
9      void Start()
10     {
11
12     }
13
14     void Update()
15     {
16         if (Input.GetMouseButtonDown(0))          // マウスがクリックされたら
17         {
18             this.speed = 0.2f;                    // 初速度を設定
19         }
20
21         transform.Translate(this.speed, 0, 0);    // 移動
22         this.speed *= 0.98f;                      // 減速
23     }
24 }
```

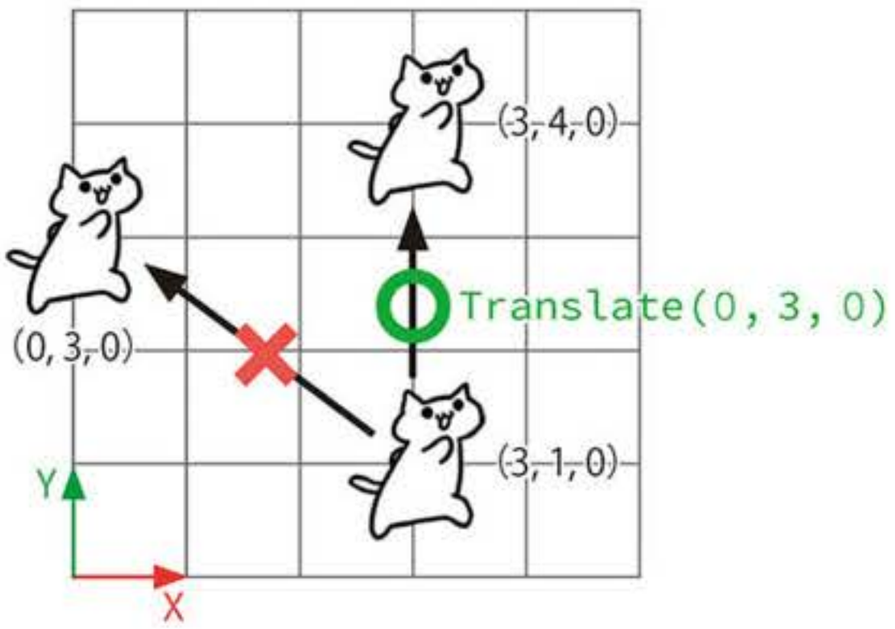
List 4-1のスクリプトは3章で作ったルーレットの回転の仕組みとほぼ同じです。車の速度はspeed変数で管理します。ゲーム開始時は「speed」が「0」なので車は動きませんが、マウスをクリックすると「speed」に値が設定され車が移動を始めます（Fig.4-18）。車の移動にはTranslateメソッドを使い、「speed」に設定された速さで車が移動しています。

Translateは、ゲームオブジェクトを現在の座標から引数に与えた量だけ移動させるメソッドです。**引数は座標を直接示しているわけではありません。現在の座標からの相対的な移動量を示しています。**つまり、`Translate(0, 3, 0)`と書いた場合、「0, 3, 0」の座標に移動するのではなく、現在の地点からY方向に「3」進みます（Fig.4-19）。

Fig.4-18 車の移動アルゴリズム

speedに「0.2」を設定　speed = 0.2で走り出す　だんだん遅くなって・・・　停止する

Fig.4-19 Translateの働き



車を徐々に減速させるために、22行目で減速の処理を行っています。速度の変数に「0.98」を掛けて、フレームごとに徐々に速度を落としています。この処理も3章のルーレットの減速方法と同じですね。

やってみよう!

初速度の「0.2」(18行目) や減衰係数の「0.98」(22行目) の値を変えることで、車の挙動が大きく変わります。通常のゲーム作りでは最後に、**ゲームを手触りのよいものにするため、これらのパラメータをすべて調整します。**この段階で自分の好みにパラメータを調整してみるのもよいでしょう。

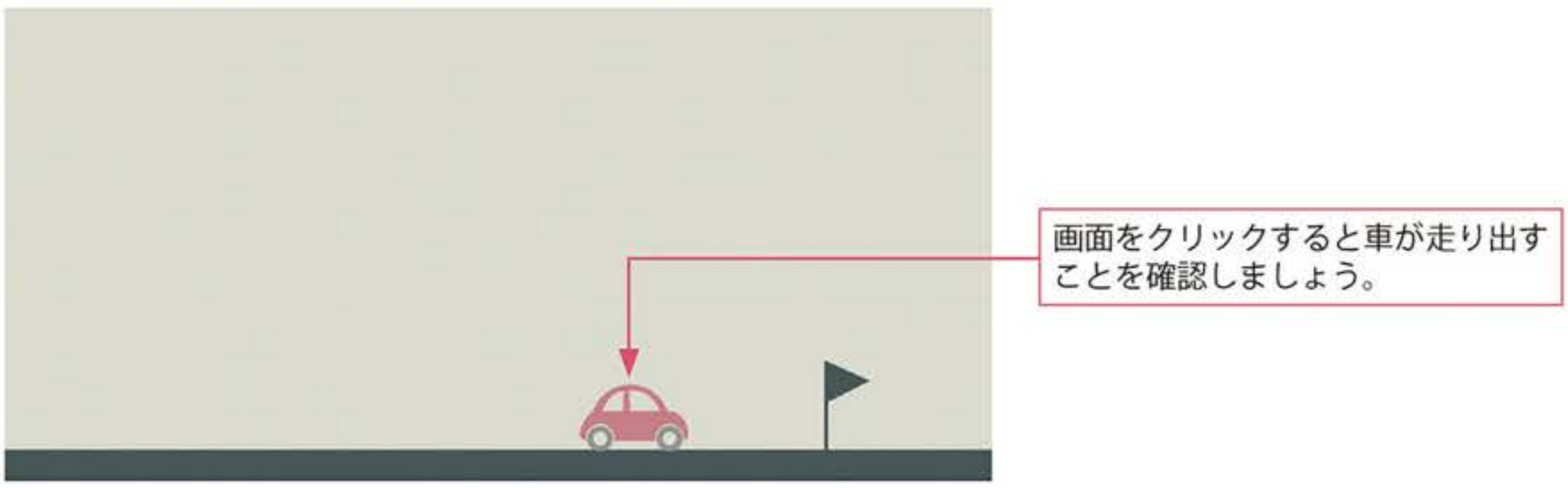
4-4-2 スクリプトを車オブジェクトにアタッチする

先ほど作成した車コントローラを、車のスプライトにアタッチします (アタッチとは役者さんに台本を渡すようなことでしたね)。プロジェクトウィンドウの「CarController」をヒエラルキーウィンドウの「car」にドラッグ＆ドロップします (Fig.4-20)。

Fig.4-20 「car」にスクリプトをアタッチする

車にスクリプトがアタッチできた（役者に台本が渡せた）ので、クリックで車が動くことを確認しましょう。ゲームを実行して、画面上でクリックすると車が走り出します！

Fig.4-21 ゲームを実行してみる



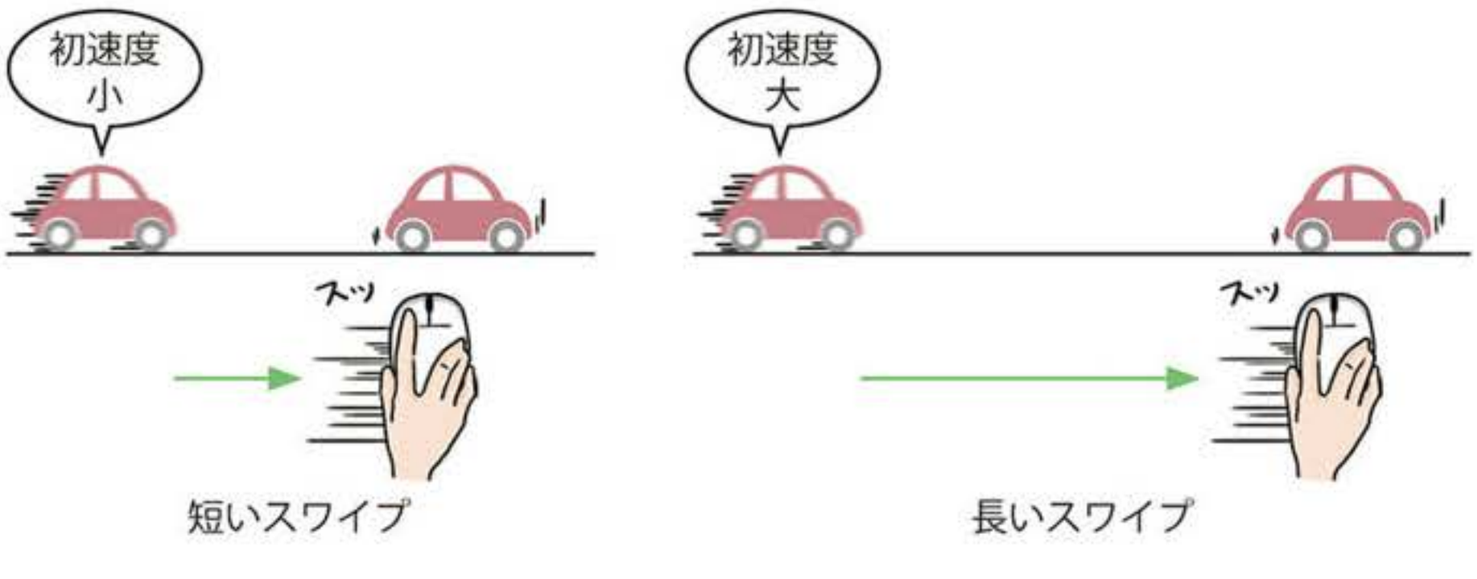
4-4-3 スワイプの長さに応じて車の移動距離を変える

車は動くようになりましたが、移動距離が毎回同じなので、これではゲームになりません。そこで、スワイプの長さに応じて車の移動距離が変わるようにしましょう。なお、ここでは、スマートフォンでのスワイプと整合性を取るために、**マウスのドラッグ動作をスワイプとします。**

スワイプの長さ（＝ドラッグの長さ）に応じて車の移動距離を変えるためには、**スワイプの長さを車の初速度（List 4-1の18行目）に設定**すればよさそうです。

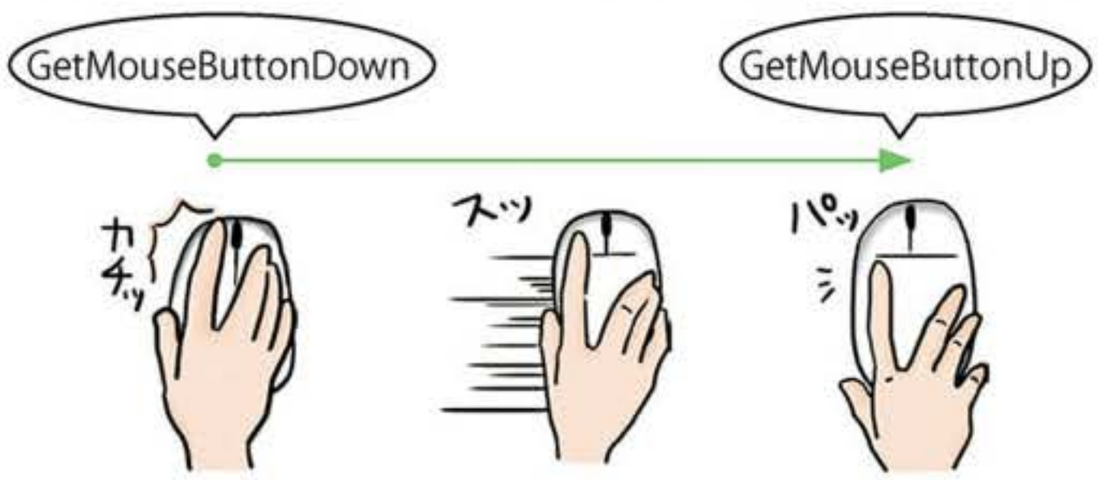
スワイプが短いと初速度が小さくなって少しの距離だけ走り、スワイプが長いと初速度が大きくなって長距離走ります。

Fig.4-22 スワイプの長さと走行距離



では、スワイプの長さをどうやって計算するか考えてみましょう。**クリックを開始した座標と、クリックが終わった座標がわかれば、その差がスワイプの長さとして使えそうです。**クリック開始と終了時のタイミングは、GetMouseButtonDownとGetMouseButtonUpメソッド(133ページ)で取得できます。それぞれのタイミングでマウスの座標(Input.MousePosition)を取得して、これらの差分を計算することで、スワイプの長さを求めます。

Fig.4-23 スワイプの長さの求め方



マウスクリックを取得するGetMouseButtonDownやGetMouseButtonUpメソッドは、スマートフォンのタッチスクリーンの操作に対応しています。したがって、スクリプトを書き換えることなく、実機でもスワイプ操作ができます。

スワイプの長さを求める処理をスクリプトに実装しましょう。プロジェクトウィンドウの「Car Controller」をダブルクリックして開き、スクリプトをList 4-2のように修正してください。

List 4-2 「スワイプの長さで移動距離を決める」スクリプト

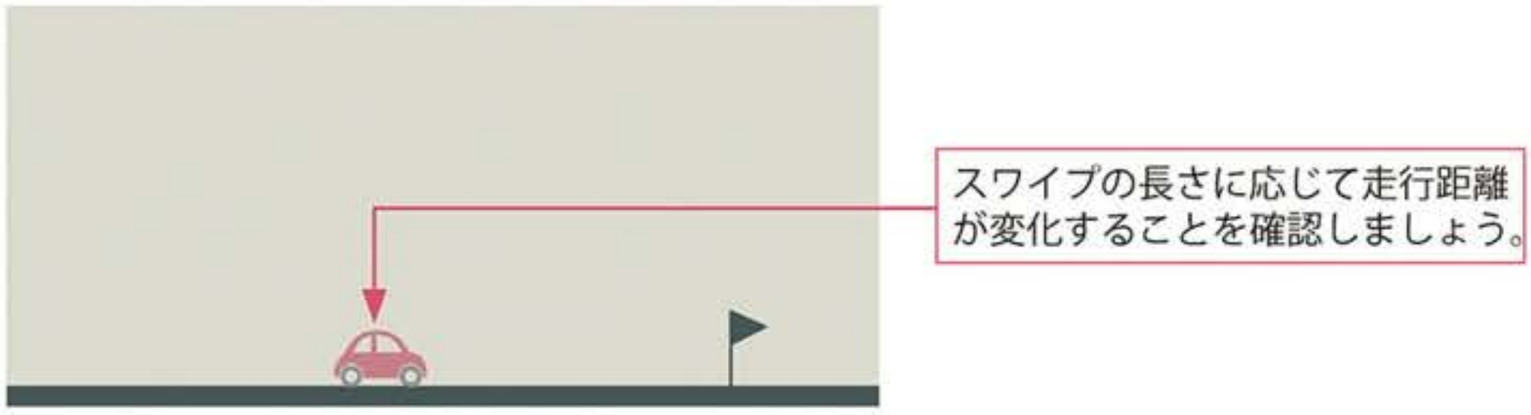
```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class CarController : MonoBehaviour
6  {
7      float speed = 0;
8      Vector2 startPos;
9
10     void Start()
11     {
12
13     }
14
15     void Update()
16     {
17         // スワイプの長さを求める
18         if (Input.GetMouseButtonDown(0))
19         {
20             // マウスをクリックした座標
21             this.startPos = Input.mousePosition;
22         }
23         else if (Input.GetMouseButtonUp(0))
24         {
25             // マウスを離した座標
26             Vector2 endPos = Input.mousePosition;
27             float swipeLength = endPos.x - this.startPos.x;
28
29             // スワイプの長さを初速度に変換する
30             this.speed = swipeLength / 500.0f;
31         }
32
33         transform.Translate(this.speed, 0, 0);
34         this.speed *= 0.98f;
35     }
36 }
```

Updateメソッドのなかで、クリックの開始地点（GetMouseButtonDown）と終了地点（GetMouseButtonUp）の座標を検出し、開始点の座標を「startPos」、終了点の座標を「endPos」に代入します。Input.mousePositionには、GetMouseButtonDownやGetMouseButtonUpメソッドがtrueになった時点のマウスの座標が入っています。

27行目では、この2点間のX軸方向の距離をスワイプの長さとしています。30行目で、このスワイプの長さを車の速度に変換しています。

画面上部の実行ボタンを押してゲームを実行し、スワイプの長さに応じて車の走行距離が変わることを確認しましょう。

Fig.4-24 スワイプの長さによって走行距離が変わる



コントローラスクリプトを使って車が動くようになりました！ スクリプトを作る時のコツは、「簡単な動きから始めて、徐々に機能追加」です！ 複雑な処理が必要なスクリプトでも、最初は単純な実験スクリプトから始めて機能追加していけば、比較的簡単に作成できますよ！ 次の4-5節では、車が旗の手前で停止したのか、通り過ぎたのかがわかるように、旗と車の距離を表示するUIを作成します。

やってみよう!

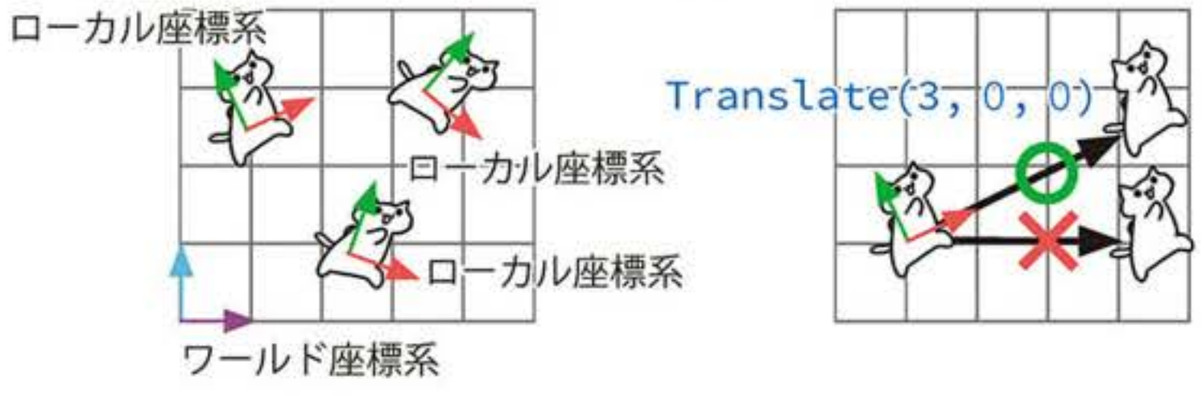
30行目で、スワイプの長さを初速度に設定する時に「500.0」で割っていますが、この値を変えると車の初速度が変化します。車の速度や移動距離を変えたい時は、この数値を変更してみてください。

Tips ワールド座標系とローカル座標系

ワールド座標系とは、ゲーム世界のどこにオブジェクトがあるのかを示すための座標系です。これまでインスペクターで設定していた座標はワールド座標系です。一方、ローカル座標系とは、ゲーム中のオブジェクトが個別に持つ座標系のことです。

Translateメソッドを使った場合の移動方向はワールド座標系ではなく、ローカル座標系で計算されます。したがって、オブジェクトが回転している場合には、回転した座標系で方向が計算されます。オブジェクトの回転と移動を同時に行う場合には注意してくださいね。

Fig.4-25 ワールド座標系とローカル座標系



4-5 UIを表示しよう

4-5-1 UIの設計方針

UI（ユーザインターフェイス）は**ゲームの状態や進行状況を表示する**もので、ユーザが気持ちよくゲームを進めるために重要な役割を果たします。UnityではUI部品のライブラリが提供されており、簡単にUIの設計が行えます。本節ではUI部品を使って、**車から旗までの距離を表示**します。UIの作成手順は次の通りです。

UIの作り方 重要!

❶UIの部品をシーンビューに配置します。
❷UIを書き換える監督スクリプトを作成します。
❸監督オブジェクトを作り、作成したスクリプトをアタッチします。

まずは、使用するUI部品を画面上に配置します。今回は**車と旗の距離をテキストで表示**するので、UIオブジェクト（Unityが用意しているUI用の部品）のTextを使います。Textが配置できたら、Textの表示を更新する監督スクリプトを作成し、監督オブジェクトにアタッチします。

4-5節では❶のUI部品を配置する手順を説明し、4-6節で❷と❸の手順を説明します。

4-5-2 Textを使って距離を表示する

車と旗の距離を表示するテキストのUIを作成します。ヒエラルキーウィンドウの＋→UI→Textを選択します。するとヒエラルキーウィンドウにCanvasという名前のオブジェクトが追加され、そのCanvasの下にTextが作成されます。

Fig.4-26 テキストを作成する

UIの設計画面は、通常のゲーム設計画面のサイズよりもはるかに大きいので、追加したTextがシーンビュー上で見当たらない場合があります。その場合は、ヒエラルキーウィンドウで「Text」をダブルクリックしてみてください。するとTextの座標付近まで視点が移動して「New Text」の文字が画面中央に表示されます。

UIの設計画面は、シーンビュー上では非常に大きく表示されますが、**実際の大きさはゲーム画面と対応しています。**実行してみるとUIだけ画面からはみ出してしまうということはありません。

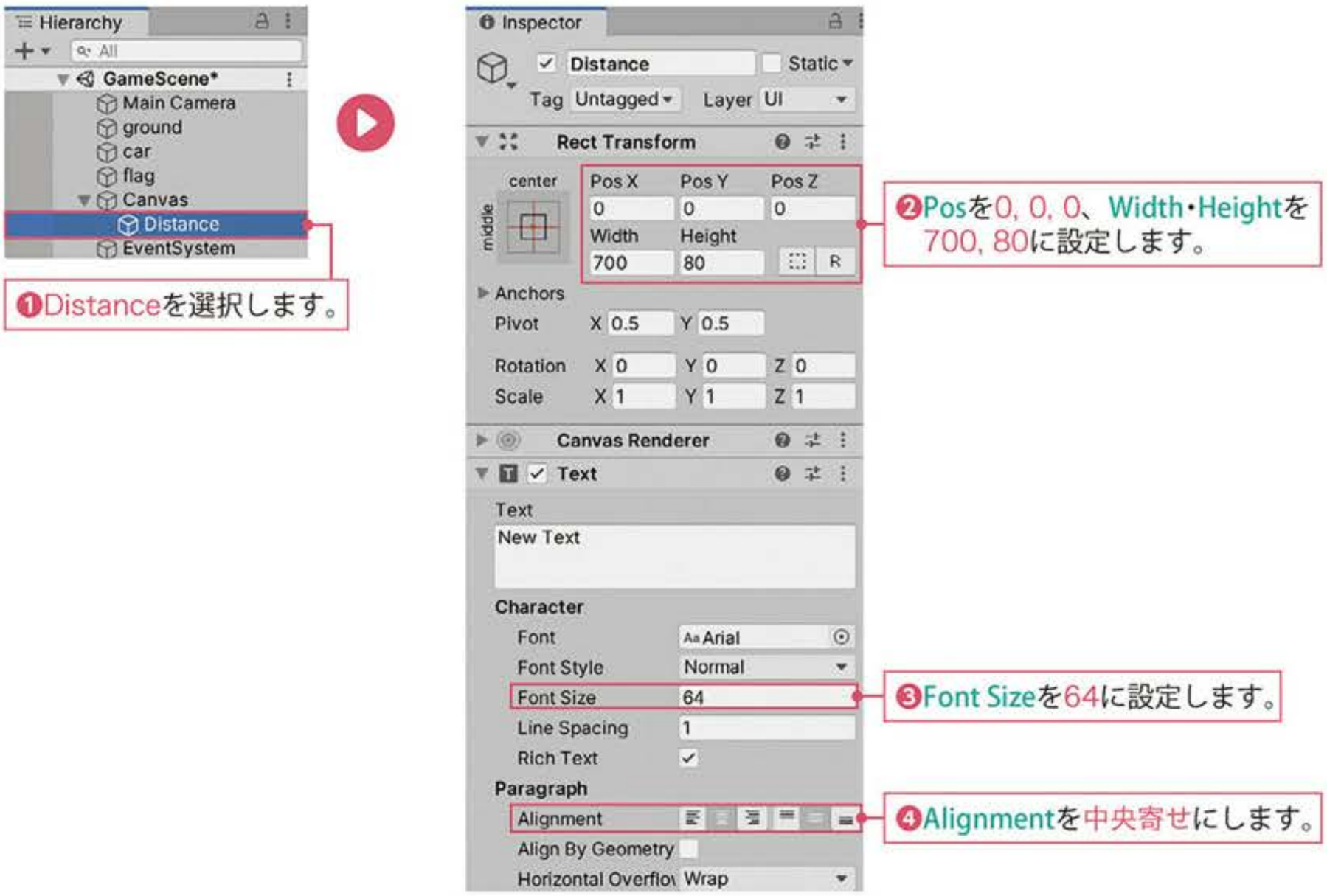
Fig.4-27 ゲームの設計画面とUIの設計画面の大きさの違い

ヒエラルキーウィンドウでTextの名前を変更して「Distance」にしておきましょう。ヒエラルキーウィンドウでTextを選択し、再度クリックすると名前の編集状態になります。その状態で「Distance」に変更します。編集した後に、エンターキーを押すと名前を確定できます。

Fig.4-28 「Text」の名前を変更する

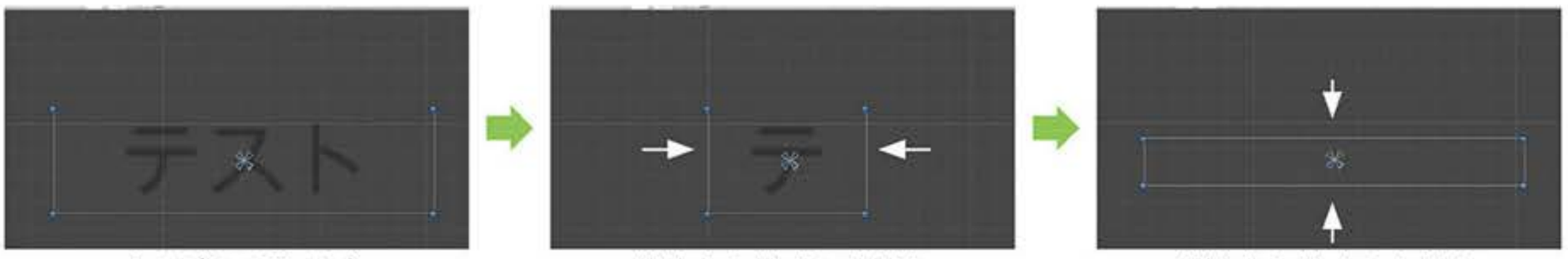
続いて、Distanceの位置とサイズを調整します。ヒエラルキーウィンドウでDistanceを選択し、インスペクターからRect Transform項目のPosを「0, 0, 0」、Width・Heightを「700, 80」、Font Sizeを「64」、Alignmentを縦方向、横方向とも中央寄せで設定します。

Fig.4-29 距離表示用のUIの設定



Textを配置する際、Rect Transformの「Width」と「Height」の値が、表示する文章に必要なサイズよりも小さいと、画面にテキストが正しく表示されないので注意してください。

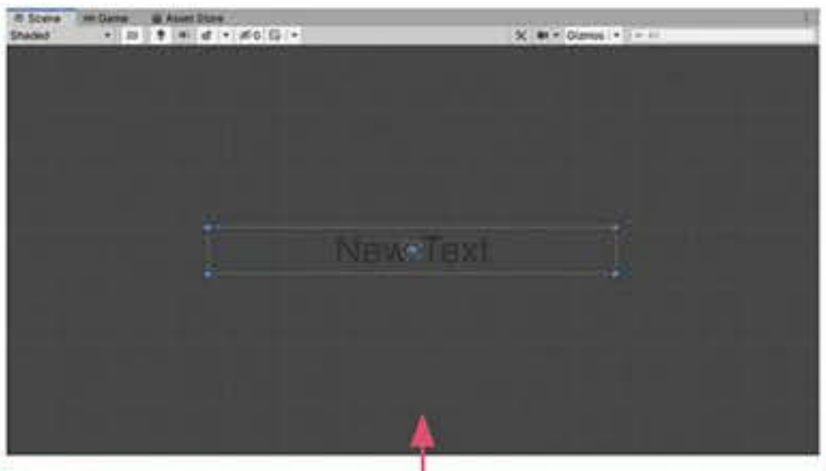
Fig.4-30 Textの大きさと表示のされ方



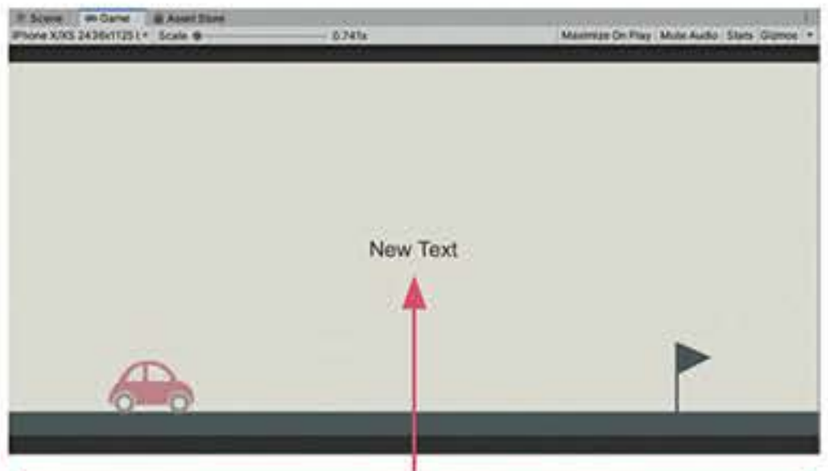
オリジナルサイズ　横方向に縮小した場合　縦方向に縮小した場合

UIの部品をシーンに配置できたので、ゲームを実行してみて画面中央に「New Text」と表示されていることを確認しましょう（Fig.4-31）。次の4-6節では「New Text」の部分に車と旗の距離を表示します！

Fig.4-31 UIの配置結果



❶シーンビューはこのように表示されています。



❷ゲームを実行して、「New Text」が画面中央に表示されることを確認してください。

Tips EventSystemとは？

UIテキストを追加すると、ヒエラルキーウィンドウに「Canvas」と一緒に「EventSystem」が追加されます。このEventSystemはユーザの入力をUI部品へと中継するオブジェクトで、UI部品を使う時は必ず1つ必要になります。EventSystemを使うことで、入力の割り当てや無効化など、キーやマウスの設定の変更ができます。

Tips Rect Transformとは？

UIでは部品の座標を表すために、「Transform」ではなく「Rect Transform」が使われています。これらの違いは、Transformが「位置」「回転」「サイズ」を変更できるのに対して、Rect Transformは「位置」「回転」「サイズ」に加えて、「ピボット」と「アンカー」を変更できます。ピボットとは回転や拡大・縮小時に使う中心座標です。また、アンカーとはUI部品を置く時に基準となる位置を指定するものです。詳しくは5章で説明します。

4-6 UIを書き換える監督を作ろう

4-6-1 UIを書き換えるスクリプトを作る

UIテキストは配置できましたが、常に「New Text」と表示していても意味がありません。そこで**「New Text」の部分に「車」と「旗」の距離を表示するための監督スクリプト**を作成しましょう。監督スクリプトはシーンビュー上の「車」と「旗」の座標を調べて、その距離を先ほど作成したUIテキストに表示します。

Fig.4-32 監督スクリプトの役割

まずは監督スクリプトを作成しましょう。プロジェクトウィンドウを右クリックしてCreate→C# Scriptを選択し、作成されたスクリプトファイルの名前を「GameDirector」に変更します。

スクリプトの作成→GameDirector

続いて、作成した「GameDirector」をダブルクリックして開き、List 4-3のスクリプトを入力・保存してください。

List 4-3 「距離情報を表示する」スクリプト

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;  // UI部品を使うために必要！
5
6  public class GameDirector : MonoBehaviour
7  {
8      GameObject car;
9      GameObject flag;
10     GameObject distance;
11
12     void Start()
13     {
14         this.car = GameObject.Find("car");
15         this.flag = GameObject.Find("flag");
16         this.distance = GameObject.Find("Distance");
17     }
18
19     void Update()
20     {
21         float length = this.flag.transform.position.x -
               this.car.transform.position.x;
22         this.distance.GetComponent<Text>().text =
               "ゴールまで" + length.ToString("F2") + "m";
23     }
24 }
```

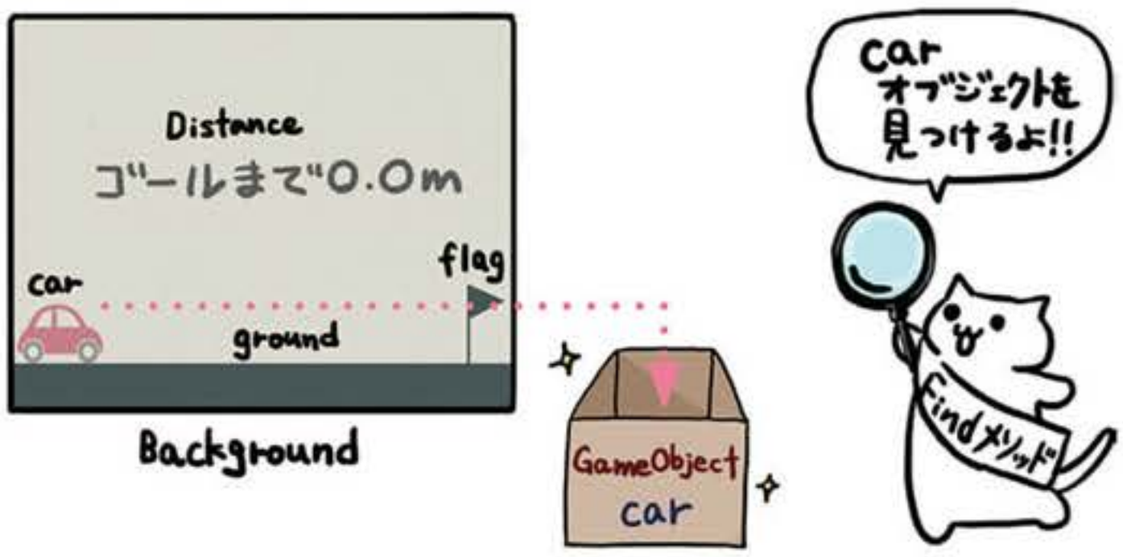
UIオブジェクトをスクリプトで使用するため、4行目でUnityEngine.UIを追加しています。これはUI部品を扱う時には必要なので、忘れないように注意してください！

監督スクリプトは、「車」と「旗」の位置を調べて距離を計算し、その結果を「UI」に表示します。そのためにはスクリプト内で「車」と「旗」と「UI」のオブジェクトをスクリプトから扱える必要があります。そこで、8〜10行目で「車」「旗」「UI」に対応するGameObject型の変数を用意しています。ただし、この時点ではGameObject型の箱を作っただけなので中身は空っぽです。

Fig.4-33 オブジェクトを格納する変数を宣言する

これらの箱に対応するオブジェクトをシーン中から見つけて入れなくてはいけません。シーン中からオブジェクトを探すために、UnityにはFindメソッドが用意されています。**Findメソッドはオブジェクト名を引数に取り、引数名と同じものがゲームシーン中にあれば、そのオブジェクトを返してくれます。**

Fig.4-34 Findメソッドの働き



シーン中から見つけた「車」と「旗」オブジェクトのそれぞれのX座標を、`car.transform.position.x`と`flag.transform.position.x`として取得しています(21行目)。この書き方の意味は本節の最後のTipsで紹介するので、ここでは**ゲームオブジェクトの座標は、「ゲームオブジェクト名.transform.position」で取得できる**と覚えておきましょう。

22行目では、distanceオブジェクトの持つTextコンポーネントに、21行目で求めた距離を代入しています。21～22行目のスクリプトは、今の段階では暗号のように感じると思います。この2行の意味を理解するには、Unityのコンポーネントについての理解が必須です。コンポーネントの説明は本節の最後のTipsで行います。 ここでは「車」と「旗」の距離を小数点第二位まで表示するため、ToStringメソッドを使って書式設定をしている部分だけ理解しておきましょう。

ToStringは数値を文字列に変換するメソッドです。引数には数値を表示する時の書式を指定することができます。引数に与える書式として代表的なものを、次のTable 4-2にまとめておきます。

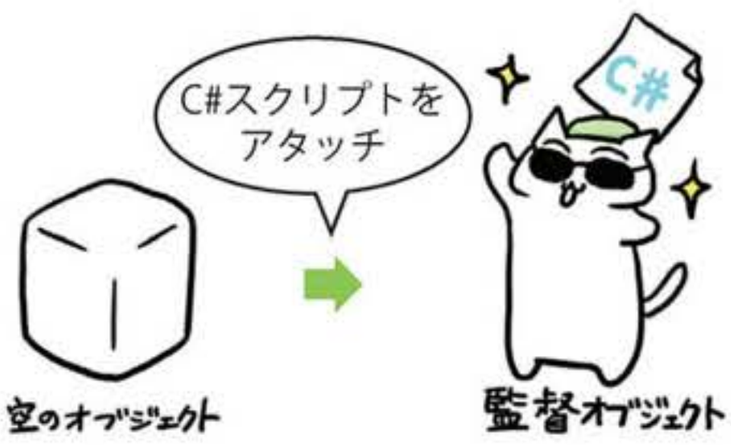
Table 4-2 ToStringで使う書式指定子

書式指定子	説明	例
整数型　D[桁数]	整数を表示する場合に使用する。指定した桁数に満たない場合は、左側にゼロが挿入される	(456).ToString("D5") → 00456
固定小数点型　F[桁数]	小数を表示する場合の小数点以下の桁数を措定する	(12.3456).ToString("F3") → 12.345

4-6-2 スクリプトを監督オブジェクトにアタッチする

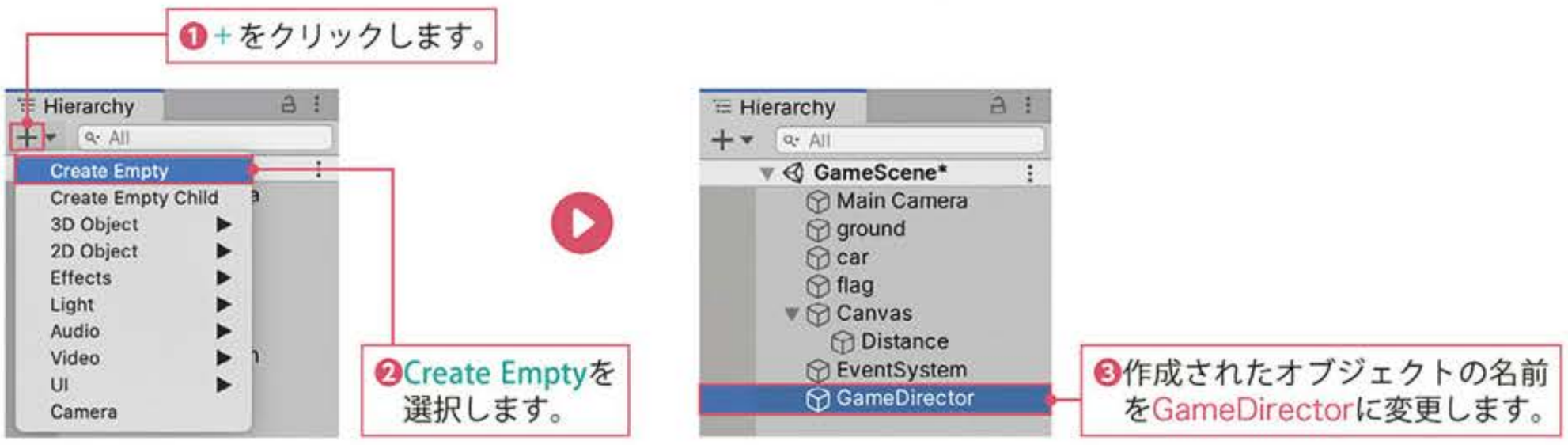
台本は、役者に渡して初めて役に立つように、スクリプトはオブジェクトに渡してはじめて役に立ちます。でも、監督スクリプトを渡すオブジェクトがいません。そこで、まっさらな「空のオブジェクト」を作り、そのオブジェクトにスクリプトを渡しましょう。**まっさらな空のオブジェクトは監督スクリプトを渡すことで監督オブジェクトに変身し、監督の役割を果たしてくれるのです。**

Fig.4-35 監督スクリプトをアタッチ



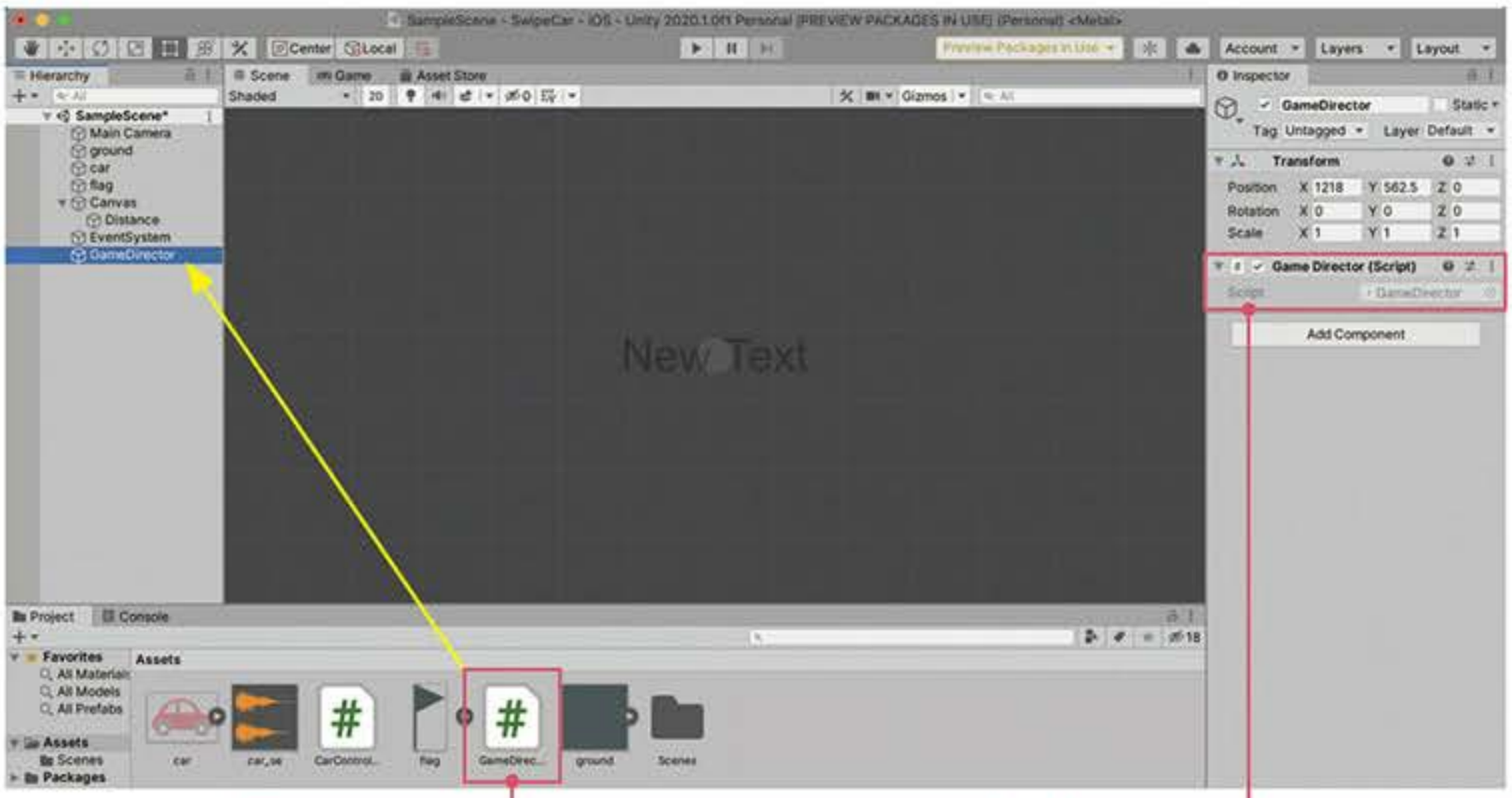
ヒエラルキーウィンドウの＋→Create Emptyを選択して、空のオブジェクトを作成します。作成するとヒエラルキーウィンドウに「GameObject」が作成されるので、名前を「GameDirector」に変更します。

Fig.4-36 空のオブジェクトを作る



作成したGameDirectorオブジェクトに、GameDirectorスクリプトをアタッチしましょう。Fig.4-37のように、プロジェクトウィンドウの「GameDirector」スクリプトを、ヒエラルキーウィンドウにある「GameDirector」オブジェクトにドラッグ＆ドロップしてください。なお、ここではアタッチするスクリプトとオブジェクトが同じ名前になっていますが、必ずしも一致している必要はありません。

Fig.4-37 空のオブジェクトにスクリプトをアタッチする

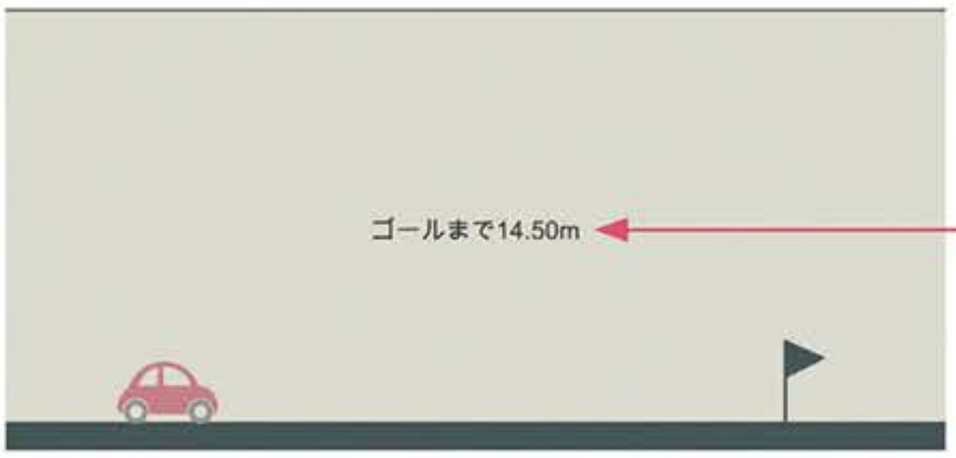


❶GameDirectorスクリプトをGameDirectorオブジェクトにドラッグ＆ドロップします。

❷「GameDirector」がアタッチされます。

ゲームを実行してみましょう。監督の指示によって、UIに車と旗の距離がリアルタイムで反映されるようになりました！

Fig.4-38 UIに車と旗の距離が表示される



車から旗までの距離がリアルタイムに表示されます。

監督の作成手順をまとめておきます。多くのゲームにおいて、UIは監督が制御することになります。「監督の作り方」も「動くオブジェクトの作り方」と同じく定石の手順になります。しっかりと理解して、使いこなせるようになってください！

監督の作り方 重要!

❶監督スクリプトを作成します。
❷空のオブジェクトを作ります。
❸空のオブジェクトに監督スクリプトをアタッチします。

やってみよう!

ゲームにおいて、チャレンジが成功したのか？ 失敗したのか？ をちゃんと提示してあげることはとても大切です。今回作ったゲームの場合、車が旗を通り越してしまったら、画面上に「ゲームオーバ！」と表示したいですね。「GameDirector」のUpdateメソッドを少し書き換えてみましょう（ここでは修正するUpdateメソッドのみを示します）。

```
void Update()
{
    float length = this.flag.transform.position.x -
        this.car.transform.position.x;
    if (length >= 0)
    {
        this.distance.GetComponent<Text>().text =
            "ゴールまで" + length.ToString("F2") + "m";
    }
    else
    {
        this.distance.GetComponent<Text>().text = "ゲームオーバ";
    }
}
```

ここで修正したところは、「車」と「旗」の距離（length）を見て、距離が「0」以上の場合はゴールまでの距離を表示し、距離が「0」未満の場合にはゲームオーバと表示する点です。このように、監督はUIの更新を行うだけでなく、ゲーム状況の把握やゲームオーバの判定なども行います。

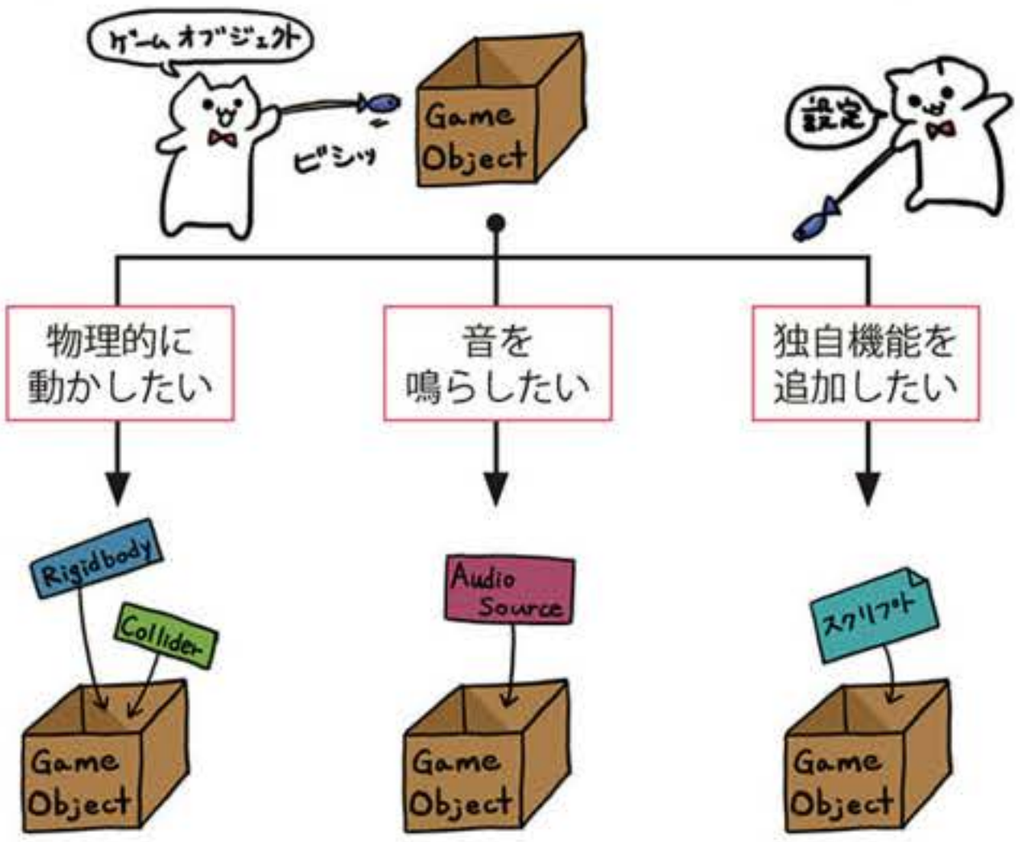
Tips コンポーネントってどういうもの？

ここでは、本節で説明しきれなかった「コンポーネント」のお話をしたいと思います。4章では車オブジェクトの座標にアクセスする際にcar.transform.position.xと書きました。車の座標なので「car.position」と書いてもよさそうなものですが、実際には「trasform」という変数が間に入っています。この「transform」は何者で、どこからやってきたのでしょうか？ それを、このTipsで詳しく説明したいと思います。

UnityのオブジェクトはGameObjectという空の箱に、設定資料（コンポーネント）を追加（アタッチ）することで機能追加することができます。例えば、オブジェクトに物理挙動をさせたい場合は、Rigidbodyコンポーネントをアタッチし、音を鳴らしたい時には4-7節に出てくるAudio Sourceコンポーネントをアタッチします。独自機能を追加したい時には、スクリプトコンポーネントをアタッチします（コントローラスクリプトや監督スクリプトもコンポーネントの一種です）。

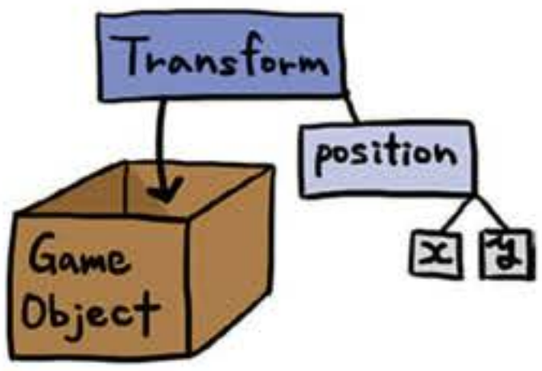
また、オブジェクトの座標や回転を管理するコンポーネントとしてTransformコンポーネントがあります。AudioSourceコンポーネントがCDプレイヤのようなものであるのに対して、Transformコンポーネントはハンドルのようなもので、座標や回転、移動といったオブジェクトの動きに関する機能を提供します。

Fig.4-39 コンポーネントの考え方



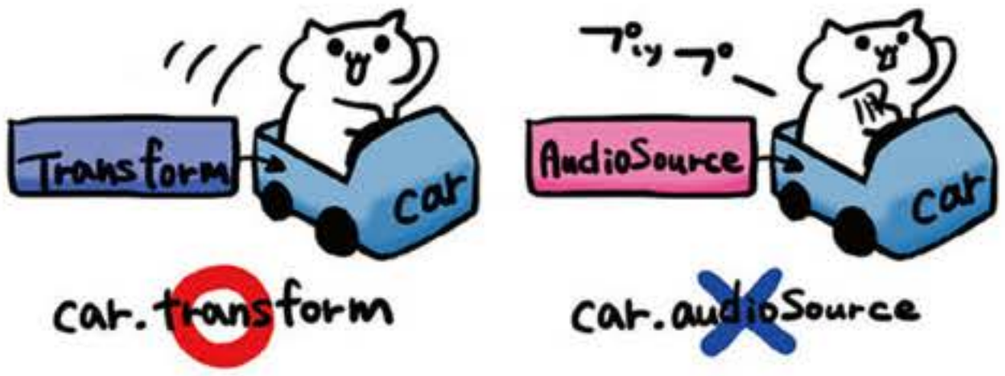
これをふまえて冒頭のcar.transform.position.xを見てみましょう。これは**車オブジェクト（car）にアタッチされたTransformコンポーネントが持つ座標（position）の情報**にアクセスしているのです。これで、transform変数が必要な謎が解けました！

Fig.4-40 Transformコンポーネントの役割



では、もう少し詳しいお話をしていきます。スクリプト中で「Transformコンポーネント」は「transform」と書きました。ではAudioSourceコンポーネントなら「audioSource」、Rigidbodyコンポーネントなら「rigidbody」と書けるのでしょうか？　残念ながらTransformを除いて、このように書けるコンポーネントはほとんどありません。

Fig.4-41 コンポーネントへのアクセス方法



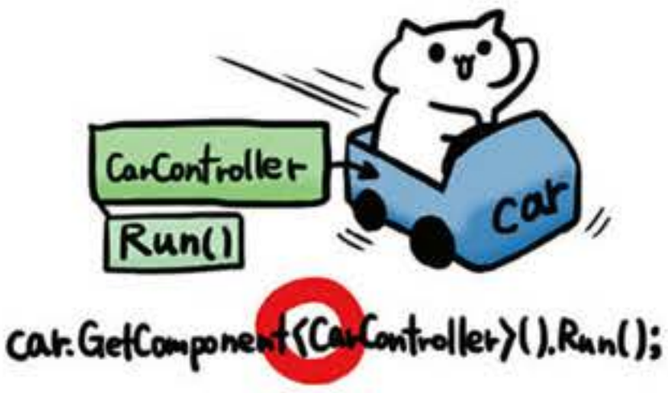
では、どのようにしてコンポーネントにアクセスすればよいのでしょうか。この問題を解決してくれるのが4-6節で登場したGetComponentメソッドです。GetComponentはゲームオブジェクトに対して「○○コンポーネントをください！」とお願いすると、該当するコンポーネントを返してくれるメソッドです。AudioSourceコンポーネントが欲しければ、`GetComponent<AudioSource>()`、Textコンポーネントが欲しければ、`GetComponent<Text>()`と書きます。

Fig.4-42 GetComponentでコンポーネントを取得する

ただ、座標を取得するためだけに毎回「GetComponent<Transform>()」と書くのは面倒くさいですよね。そこで、よく使うTransformコンポーネントには、「GetComponent<Transform>」の簡単な書き方としてtransformが用意されているのです。つまり`transform`イコール`GetComponent<Transform>()`と考えられます。

また、自分で作ったスクリプトもコンポーネントの一種ですので、GetComponentメソッドを使って取得できます。CarControllerスクリプトにRunメソッドを実装した場合、`car.GetComponent<CarController>().Run()`という記述で、車オブジェクトにアタッチされたCarControllerスクリプトのRunメソッドを呼び出すことができるのです。

Fig.4-43 GetComponentでメソッドにアクセスする



GetComponentを使って自作スクリプトのメソッドを呼び出す例は、これからもたくさん出てきます。オブジェクトとコンポーネントの関係について、しっかりと理解しておきましょう！

自分以外のオブジェクトの持つコンポーネントにアクセスする方法 重要!

❶Findメソッドでオブジェクトを探し出します。
❷GetComponentメソッドでオブジェクトの持つコンポーネントを取得します。
❸コンポーネントの持つデータにアクセスします。

4-7 効果音の鳴らし方を学ぼう

最後の仕上げとして車に効果音を付けます。効果音はゲームの手触りをよくする重要な要素です。ゲーム作りにおいてサウンドはどうしても後回しになりがちですが、手を抜かずに納得できる効果音を探すことが、質の高いゲームを作ることにつながります。

4-7-1 AudioSourceコンポーネントの使い方

Unityで効果音を鳴らすにはAudioSourceコンポーネントを使います。このコンポーネントを使って、スワイプ時に「ブーン」という効果音を付けてみましょう。効果音を鳴らす流れは次の3ステップです。

効果音の鳴し方 重要!

❶音を鳴らしたいオブジェクトにAudioSourceコンポーネントをアタッチします。
❷AudioSourceコンポーネントに効果音をセットします。
❸効果音を鳴らしたいタイミングに合わせて、スクリプトからPlayメソッドを呼びます。

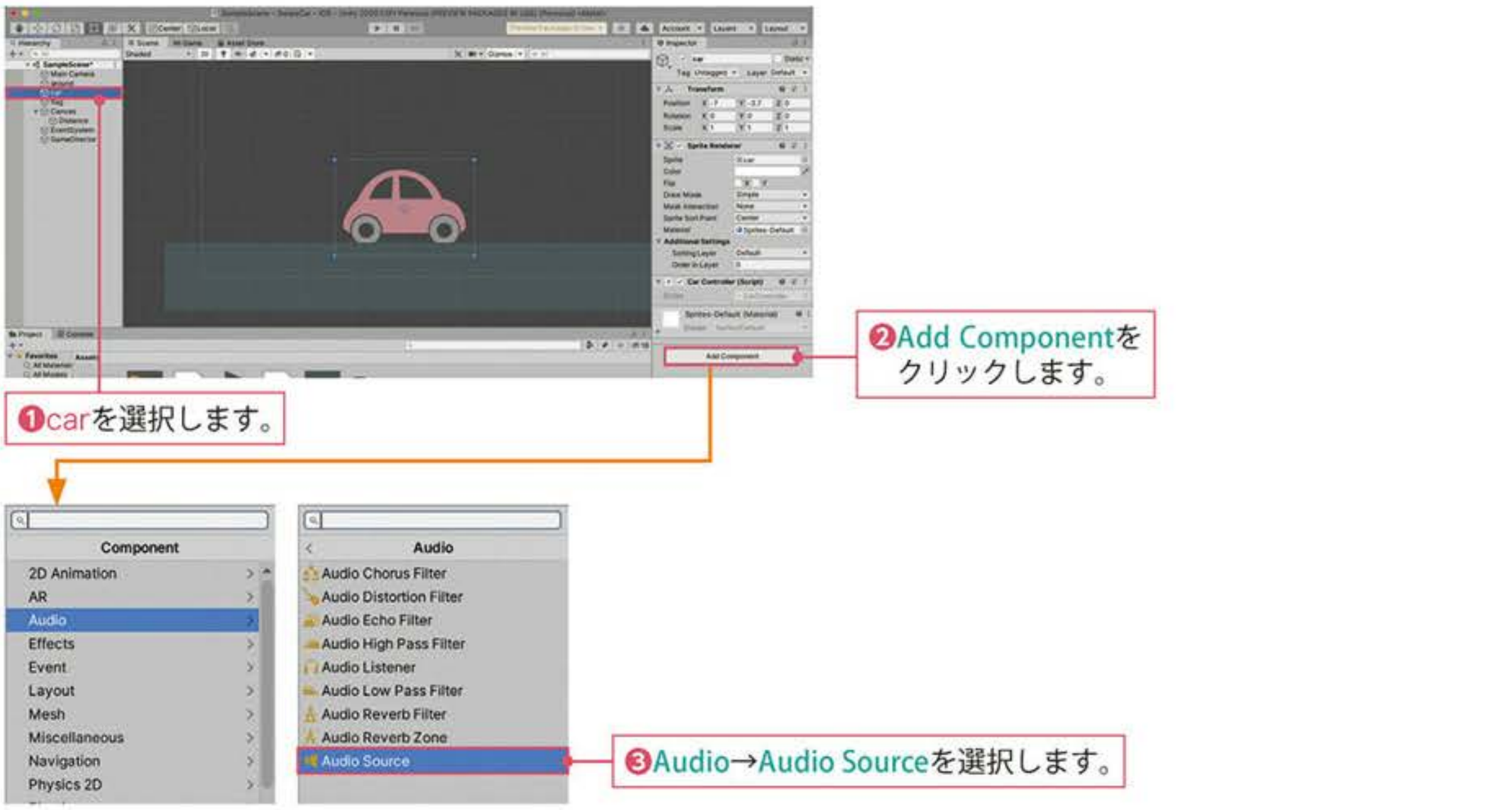
4-7-2 AudioSourceコンポーネントのアタッチ

AudioSourceコンポーネントとはCDプレイヤのようなものです。そこに音源となるディスク（音楽ファイル）をセットすることで、好きな音を鳴らすことができます。今回は車から効果音を鳴らしたいので、車オブジェクトにAudioSource（CDプレイヤ）をセットしましょう。

Fig.4-44 AudioSourceコンポーネントとは

車オブジェクトにAudioSourceコンポーネントをアタッチするため、ヒエラルキーウィンドウからcarを選択し、インスペクターからAdd Componentボタンをクリックし、Audio→Audio Sourceを選択してください。

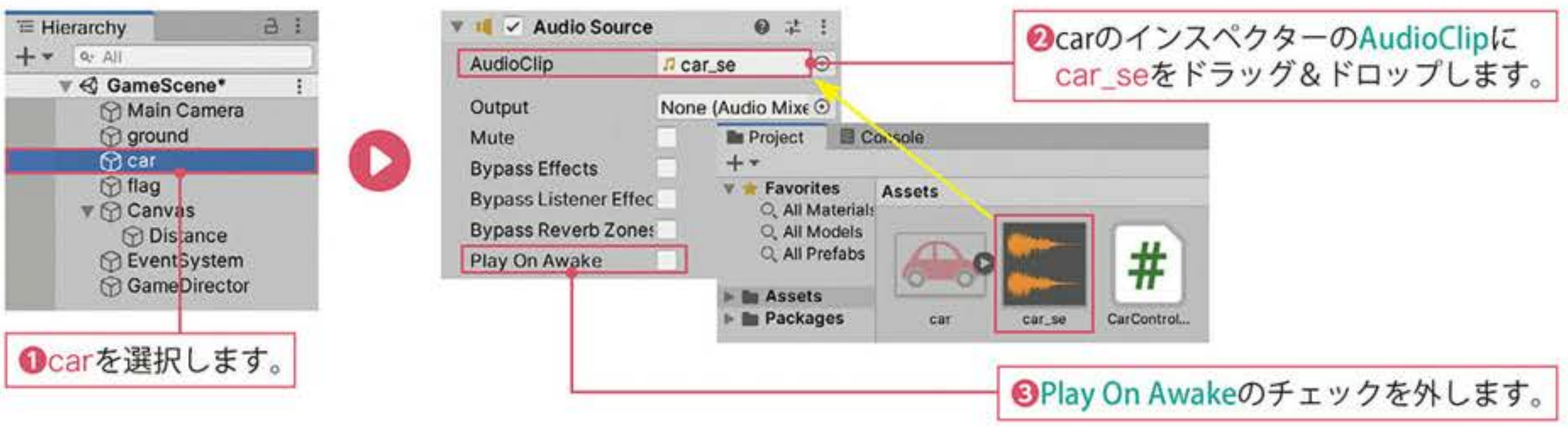
Fig.4-45 AudioSourceコンポーネントをアタッチする



4-7-3 効果音をセットする

車にCDプレイヤ（AudioSourceコンポーネント）をアタッチできたので、次はCDプレイヤにディスク（音楽ファイル）をセットします。「car」のインスペクターのAudio Source項目のAudioClipの欄に、プロジェクトウィンドウから「car_se」をドラッグ＆ドロップしてください。ここで、Play On Awakeの欄のチェックボックスは外しておきます。この欄にチェックが入っていると、ゲームを開始した時点で自動的に音が再生されてしまいます。

Fig.4-46 Audio Sourceにサウンドファイルをセットする



4-7-4 スクリプトから音を再生する

効果音を再生するには、スクリプトからAudioSourceコンポーネントのPlayメソッドを呼び出します。AudioSourceコンポーネントはcarオブジェクトにアタッチしたので、同じくcarオブジェクトにアタッチされているスクリプト（CarController）からPlayメソッドを呼ぶことにしましょう。

Fig.4-47 スクリプトから再生する

プロジェクトウィンドウにある「CarController」スクリプトをダブルクリックして開き、List 4-4のように効果音を鳴らす処理を追加してください。

List 4-4 効果音を鳴らす処理を追加する

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class CarController : MonoBehaviour
6  {
7      float speed = 0;
8      Vector2 startPos;
9
10     void Start()
11     {
12
13     }
14
15     void Update()
16     {
17         // スワイプの長さを求める
18         if (Input.GetMouseButtonDown(0))
19         {
20             // マウスをクリックした座標
21             this.startPos = Input.mousePosition;
```

```
22          }
23          else if (Input.GetMouseButtonUp(0))
24          {
25              // マウスを離した座標
26              Vector2 endPos = Input.mousePosition;
27              float swipeLength = endPos.x - this.startPos.x;
28
29              // スワイプの長さを初速度に変換する
30              this.speed = swipeLength/500.0f;
31
32              // 効果音再生
33              GetComponent<AudioSource>().Play();
34          }
35
36          transform.Translate(this.speed, 0, 0);
37          this.speed *= 0.98f;
38      }
39  }
```

車をスワイプして指が離れた瞬間に効果音が鳴るよう、33行目に効果音を再生するスクリプトを追加しています。`GetComponent<AudioSource>()`でAudioSourceコンポーネントを取得して、AudioSourceコンポーネントの持つPlayメソッドを呼んでいます。

ゲームを実行して車をスワイプしてみてください。スワイプした瞬間に効果音が鳴りました！Unityを使えば効果音を付けるのも簡単なことがわかりました！

4-7節では効果音の付け方を説明しました。効果音はゲームの手触りのよさに直結します。市販のゲームでは、どこに、どんな効果音を付けているのか、じっくり観察してみると勉強になるので、ぜひ確認してみてください。

≻Tips≺ 使える音源ファイルの種類と拡張子

Unityでは幅広い音源ファイルをサポートしています。代表的なものはTable 4-3の通りです。その他の音源に関してはUnityのWebサイトで確認してください。

Table 4-3 Unityで使える音源

形式	拡張子
MPEG Layer3	.mp3
Ogg Vorbis	.ogg
Microsoft Wave	.wav
Audio Interchange File Format	.aiff/.aif