

VRを気軽に体験

モバイルVR コンテンツを作ろう!

酒井 駿介=著

最新版のUnity 2017を使った
Gear VR & ハコスコで動く
VRコンテンツの作り方

インプレス

目次

1. [大扉](#)

2. [本書について](#)

1. [免責事項](#)

3. [第1章 モバイルVR開発をはじめよう](#)

1. [1.1 2016年はVR元年](#)

2. [1.2 気軽に楽しめるモバイルVR](#)

1. [Gear VR](#)

2. [Google Cardboard](#)

3. [Google Daydream](#)

4. [ハコスコ](#)

3. [1.3 モバイルVRの遊び方](#)

4. [1.4 GearVRの遊び方](#)

1. [必要なもの](#)

2. [手順](#)

5. [1.5 Google Cardboard/ハコスコの遊び方](#)

1. [必要なもの](#)

2. [手順](#)

4. [第2章 モバイルVRの開発環境を構築しよう](#)

1. [2.1 モバイルVRコンテンツの開発手法](#)
 2. [2.2 Unityとは](#)
 3. [2.3 HMDごとの環境構築作業を確認する](#)
 4. [2.4 Unityのインストール](#)
 5. [2.5 Android SDKのインストール](#)
 1. [Android Studioの入手とインストール](#)
 2. [SDK Managerでインストール](#)
 6. [2.6 UnityにAndroid SDKを設定する](#)
 7. [2.7 SDKの不具合対策](#)
 8. [2.8 パスを通す](#)
 9. [2.9 osig ファイルの作成\(Gear VRのみ\)](#)
-
5. [第3章 サンプルプロジェクトをビルドしてみよう](#)
 1. [3.1 サンプルプロジェクト入手する](#)
 2. [3.2 VR Samplesのダウンロードとインポート](#)
 3. [3.3 Gear VR向けにビルドする](#)
 6. [第4章 Unityの仕組みを理解しよう](#)
 1. [4.1 Unityエディタ](#)
 2. [4.2 Unityスクリプト](#)
 3. [4.3 Unityエディタを使いこなそう](#)
 1. [6つのメインウィンドウ](#)

2. 3Dの操作に慣れよう

4. 4.4 Unityスクリプトを理解しよう

1. Hello World① クラスを編集しよう

2. Hello World② クラスをコンポーネント化しよう

3. MonoBehaviourクラスについて

5. 4.5 おわりに

7. 第5章 モバイルVRゲームを作ってみよう

1. 5.1 サンプルプロジェクトをビルドしてみよう(ハコスコ/Google Cardboard)

1. 実機にビルドする

2. 5.2 モバイルVRゲームを作ってみよう～設計と素材集め

1. どんなゲームを作るか

2. シューティングゲームの設計

3. シューティングゲームに必要な素材

3. 5.3 おわりに

8. 第6章 VRシューティングゲームを実装しよう

1. 6.1 プレイヤーからの入力を実装する

1. HMDからの入力を受け付ける

2. タッチパッド・コントローラからの入力を受け付ける

3. 弾丸の発射

4. プレイヤー機能の仕上げ

2. 6.2 エネミー機能の実装

1. エネミーの用意とパラメータ定義

2. エネミーの移動処理を実装する

3. エネミーの衝突検知

3. 6.3 おわりに

9. 第7章 VRゲームのグラフィックを強化しよう(前編)

1. 7.1 Unity Asset Storeからモデル入手する

2. 7.2 キャラクターモデルを動かしてみよう

1. アニメーションコントローラの設定

2. Enemyクラスのアイン

3. スクリプトからのステート制御

4. 各アニメーションステートへの遷移処理

5. アニメーションステートが終了したかどうかの処理

3. 7.3 動作確認をしよう

4. 7.4 おわりに

10. 第8章 VRゲームのグラフィックを強化しよう(後編)

1. 8.1 ライティングとは

1. 直接光と間接光

2. リアルタイムライトとベイクライト

3. [staticオブジェクトとnon-staticオブジェクト](#)

4. [ライティングのポイントまとめ](#)

2. [8.2 背景モデルをライティングしてみよう](#)

1. [アセットの最適化](#)

2. [ライトの配置](#)

3. [ライトプローブの作成](#)

4. [ライトマップの作成](#)

5. [シーンの確認と調整](#)

3. [8.3 ビルドして遊んでみよう](#)

11. [著者紹介](#)

1. [スタッフ](#)

12. [奥付](#)

VRを気軽に体験 モバイルVRコンテンツを作ろう！（Think IT Books）

| 酒井 駿介

本書について

本書のサンプルプログラムは以下のURLで公開しています。

<https://github.com/thinkitcojp/mobileVR-samples>

免責事項

- ・本書は、インプレスが運営するWebメディア「Think IT」で、「Gear VR＆ハコスコで動くモバイルVRコンテンツを作ろう！」として連載された技術解説記事を書籍用に再編集したものです。
- ・本書の内容は、執筆時点(2017年8月)までの情報を基に執筆されています。紹介したWebサイトやアプリケーション、サービスは変更される可能性があります。
- ・本書の内容によって生じる、直接または間接被害について、著者ならびに弊社では、一切の責任を負いかねます。
- ・本書中の会社名、製品名、サービス名などは、一般に各社の登録商標、または商標です。なお、本書では©、®、TMは明記していません。

第1章 モバイルVR開発をはじめよう

本書では、モバイルでVRコンテンツを開発するためのテクニックを紹介していく。本格的な開発にとりかかる前に、まずは現在のVRをめぐる状況を整理してみよう。

1.1 2016年はVR元年

多くのメディアで、2016年は「VR元年」と紹介されている。なぜなら、今年は「Oculus Rift」や「HTC Vive」、「PlayStation VR」といった有力なVRデバイスの発売が次々に予定されており、それらを使ったVRコンテンツを本格的に楽しむことができるようになるからだ。

これらのデバイスはヘッドマウントディスプレイ(HMD)になっており、これを頭部に装着することで、ヴァーチャル空間上でゲームやシミュレーションを行うことができるというわけだ。SF映画やアニメで描かれていたことが、ついに現実で体験できるようになっているのだ。

デバイスには、大きく分けて主にデスクトップ・コンソールVRデバイスと、モバイルVRデバイスの2種類が存在する。デスクトップ・コンソール向けは、Oculus RiftやHTC Vive、PlayStation VRといった製品だ。それぞれPCやコンソール機に接続して遊ぶことができる。

筆者はすでにこれらのVRデバイスを遊んでるが、いずれも非常に没入感の高いVR体験を味わうことができる。目の前に広がる映像は本当に美しく、いつまでもヴァーチャル空間にとどまっていたいと思えたほどだ。

一方、VR空間の描画には、それなりにハイスペックなマシン・ハードウェアが必要になる。特にグラフィックカードは高品質な3DCG映像をレンダリングするために、より高性能なものが求められる。さらにはマシンやHMD、センサ類をそれぞれ接続・設置しなくてはならない。こういったセットアップ作業が、とにかく大変なのだ。このようなコンソール型のVRデバイスは、腰をすえてどっぷりヴァーチャル世界に入り浸りたい人以外には、なかなか敷居が高いと思われるかもしれない。

1.2 気軽に楽しめるモバイルVR

一方、「もっと気軽にVRを楽しみたい」、という人も多いはず。その要望に応えるのが「モバイルVR」だ。これは誰もが持っているスマートフォンをHMDにしてVRコンテンツを楽しむというものだ。コンソール型VRデバイスのように、ハイスペックマシンを用意したり面倒な配線に困ることもない。モバイルVRは、よりカジュアルにヴァーチャル空間での体験を楽しむことができるのだ。現在(2017/01)、主に3つのモバイルVRデバイスがすでに発売されている。

Gear VR

Gear VR(図1.1)は、サムスン電子とOculus VR社が展開するモバイルVRデバイスだ。Gear VRにスマートフォン「Galaxyシリーズ」を装着することでHMDとして機能するようになる。モバイルVRといえど、非常に没入感の高いVR体験を味わうことができる。また、「Oculus Store」と呼ばれる独自のアプリ配信プラットフォームが用意されており、各アプリはそこを通じてダウンロード・プレイする仕組みになっている。

なお、2016年にGear VRの改良版が発表され、ハードウェアが改良されたほか、コントローラが追加された。



図1.1: サムスン電子とOculus VR社が展開するモバイルVRデバイス「Gear VR」

Google Cardboard

Google Cardboard(図1.2)は、Googleが設計したVR/ARビューワの総称で、スマホを取り付けることでHMDになる製品だ。このビューワは、物理的なサイズさえ合えば基本的にどんなスマホも取り付けることができる。Googleはこの製品の仕様を公開しており、それを元に各ベンダーが様々な種類のビューワを製造・販売している。ビューワはダンボール製の簡易なものからプラスチックでできた本格的なものまで、多くの選択肢があるのが特徴だ。アプリの入手は通常のスマホのようにApp StoreやGoogle Playから対応したものをダウンロードする。



図1.2: Google が設計した VR/ARビューワ「Google Cardboard」

Google Daydream

Daydreamは、Cardboardの高品質版と位置づけられたVRプラットフォームだ。コントローラ付きのHMDに、Daydream-readyに対応したスマートフォンをセットして使用する。残念ながら、2017年8月現在、日本国内での正式展開はされていないが、本連載の解説では一応ビルドを作ることはできる。

ハコスコ

ハコスコ(図1.3)もまた、手持ちのスマホを装着することでHMDとして利用できるVRビューワの1つだ。日本企業であるハコスコ社から発売されており、ダンボール・紙で作られたものやプラスチックでできた製品も用意されている。Google Cardboardとは違い、一眼タイプのビューワも用意されているため、子供も安全に使用できること(一眼・二眼の違いは後述)。

iOS/Android で公開されている“ハコスコ”アプリを通してVRコンテンツをダウンロードできるほか、App StoreやGoogle Playから対応アプリを個別にインストールできる。



図1.3: 日本のハコスコ社が提供するVRビューワ「ハコスコ」

本連載で想定するデバイスについて

紹介しているHMD本体とスマートフォンは2016年連載時のもので、それぞれ Gear VR(2015年版)とGalaxy S6 edge を、ハコスコ / Google Cardboard 向けでは、Galaxy S6 edgeおよびiPhone 6sでの使用を想定している。

1.3 モバイルVRの遊び方

モバイルVRでコンテンツを楽しむには、事前にいくつかの準備が必要だ。製品によってその手順がことなるため、ここではGear VRとGoogle Cardboard/ハコスコの2つに分けて解説していく。なお、事前に必ずそれぞれの製品に付属している説明書・解説書をよく確認してほしい。また、VR体験は安全な場所でを行い、プレイ途中に気分が悪くなったり、不快感を感じたら、すぐに使用を中止しよう。

1.4 GearVRの遊び方

必要なもの

- Gear VR本体
- Galaxy S6、S6 edge、Note 5、S6 edge+ のいずれか（docomo/auなどのキャリアは問わず）

手順

1. GalaxyをGear VRに取り付けてHMD状態にする（図1.4）。それを頭に装着すると、初回のみOculusアプリのインストールを促す画面が現れる。その後HMDを外し、GalaxyもGear VRから取り外してスマホ状態に戻そう。





図1.4: GalaxyをGear VRに取り付けてHMD状態にする

2. 画面の案内に応じてOculusアプリのインストールを進める(途中でOculusアカウントを作成する必要がある)。その後、スマホ状態でOculusアプリを起動すればコンテンツの購入やダウンロードが行うことができる(図1.5)。

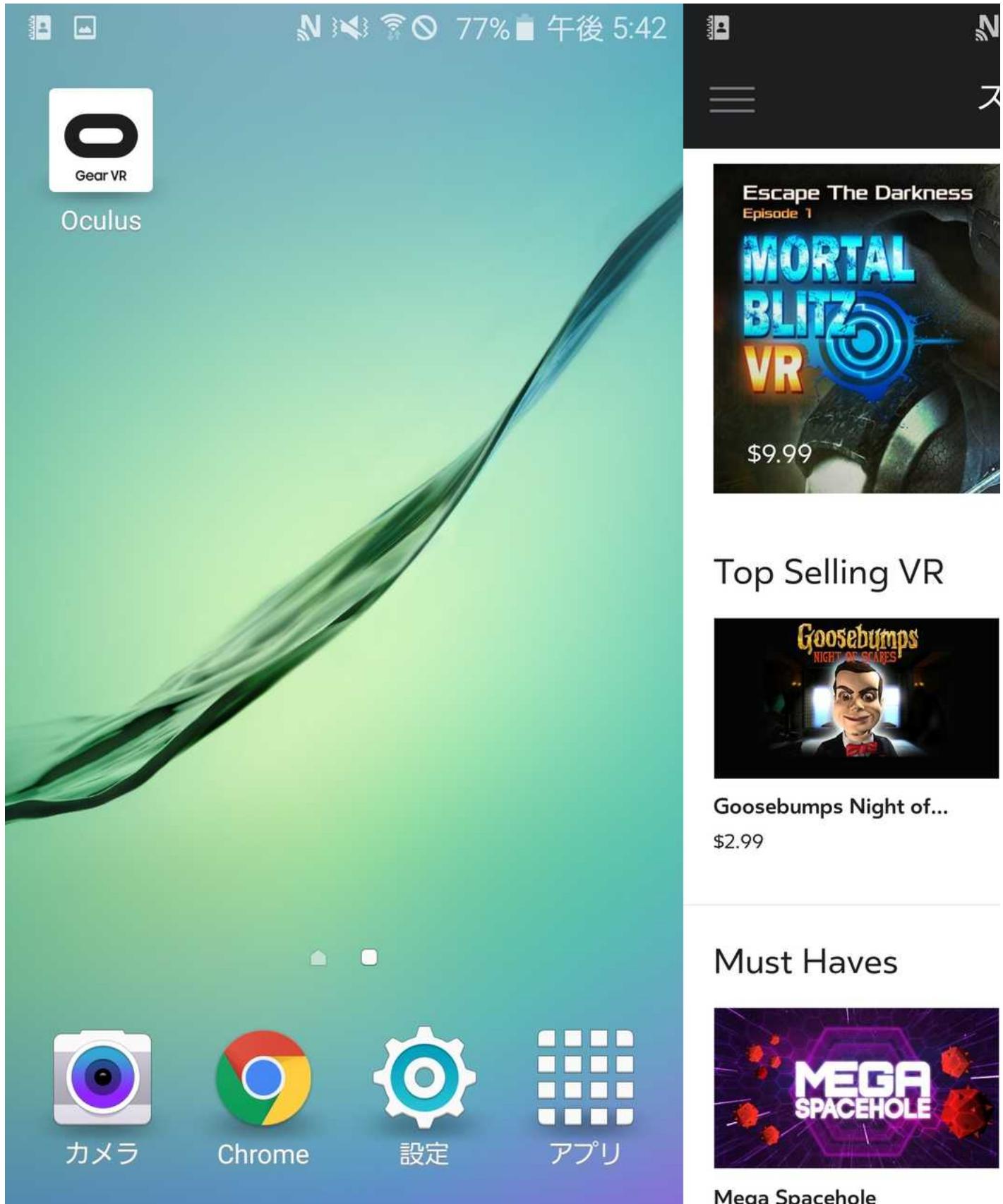


図1.5: Oculusアプリのインストール

3. 再びGalaxyを取り付けてHMD状態にし、頭に装着すると「Oculus Home」という画面が現れる(図1.6)。初回のみここで操作のチュートリアルがあり、その後は自由にアプリやコンテンツをプレイできる状態になる。



図1.6:「Oculus Home」

1.5 Google Cardboard/ハコスコの遊び方

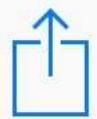
必要なもの

- Google Cardboard/ハコスコ ビューワ
- ビューワに対応したスマホ

手順

1. ビューワを組み立てる。最初から完成している製品もあるが、ダンボールなどで作られたものでは組み立てが必要なことが多い。それぞれの説明書を参考にして作業を行おう。

2. VRに対応しているスマホアプリをApp Store/Google Playからダウンロードして起動する(図1.7)。アプリによっては一眼・二眼のモードが選択できるので、ビューワに合わせて設定しておく。

[Back](#)

シドニーとあやつり王の墓
GREE, Inc. >

[Details](#)[Reviews](#)[Related](#)

図1.7: VR対応のスマホアプリをダウンロード

3. スマホをビューワに取り付けてHMD状態にしたら、頭に装着したり、手で持ちながら顔に添えてコンテンツを視聴する。スマホをビューワにしっかり固定できないものもあるので、VR体験中にはスマホの落下に注意しよう(図1.8)。



図1.8: VR体験中はスマホのズレや落下に注意！

[column]一眼と二眼の違い

前述したように、ビューワによっては一眼と二眼のものが存在する。これは「コンテンツのステレオ表示に対応しているか、していないか」の違いだ。スマホの画面を左右2つに分割するのがステレオ表示だが、この時それぞれの画面に視差をつけることで、二眼のビューワを通して見た時に映像が立体的に視聴できる。より臨場感のある映像を楽しみたい場合は、二眼のビューワを用いてステレオ表示モードがあるコンテンツを選択すると良いだろう(図1.9)。

一方、一眼のビューワーを用いる場合は、コンテンツはモノラルな状態で表示される必要がある。画面は1つのままで当然映像に視差はなく、二眼の場合と比べて迫力は劣ってしまうことになるが、眼に対する負担はこちらのほうが少ない。眼の成長途中有る子どもがVR体験をする場合は一眼ビューワーを用いるべきだ。とはいえ、いずれにしても眼に負担がかかることは間違いないので、保護者の監視のもと長時間のプレイは避けたほうが良いだろう。



図1.9: 二眼ビューワーでは臨場感のある映像を楽しめる

今回は、今が旬のVR技術について、モバイル機器で簡単に体験するための基礎知識を紹介した。モバイルVRを体験するためにどのような機器が必要で、現状どのようにになっているのかを整理できたのではないだろうか。

次章からは、モバイルVR開発に必要な環境構築について解説する。

第2章 モバイルVRの開発環境を構築しよう

今回は、モバイルVRコンテンツ開発に必要な環境を構築する手法を紹介する。実際に自身のPCに開発環境を構築し、コンテンツ制作をはじめられるようにするのが目的だ。

まずは、「そもそもどのようにVRコンテンツを開発していくのか」を確認してみよう。

2.1 モバイルVRコンテンツの開発手法

VRコンテンツは、主に次の2つ機能からなる、とみなすことができる。

1. ヴァーチャル空間を3D映像としてレンダリングする機能
2. HMD(ヘッドマウントディスプレイ)の傾き・位置情報・ボタン等の、入力を行う機能

このうち、1.の3Dレンダリング機能はモバイルであれば「OpenGL ES」アーキテクチャを利用して作ることができるが、もっとも現在はUnityやUnreal Engineなどのゲームエンジンを使用するのが一般的だ。2.の入力機能もまた、ゲームエンジン側が提供している豊富な入力機能を利用して実装することが可能だ。つまり、ゲームエンジンを用いてモバイルアプリをビルドすることでスマホで動くVRコンテンツが制作できるワケだ。

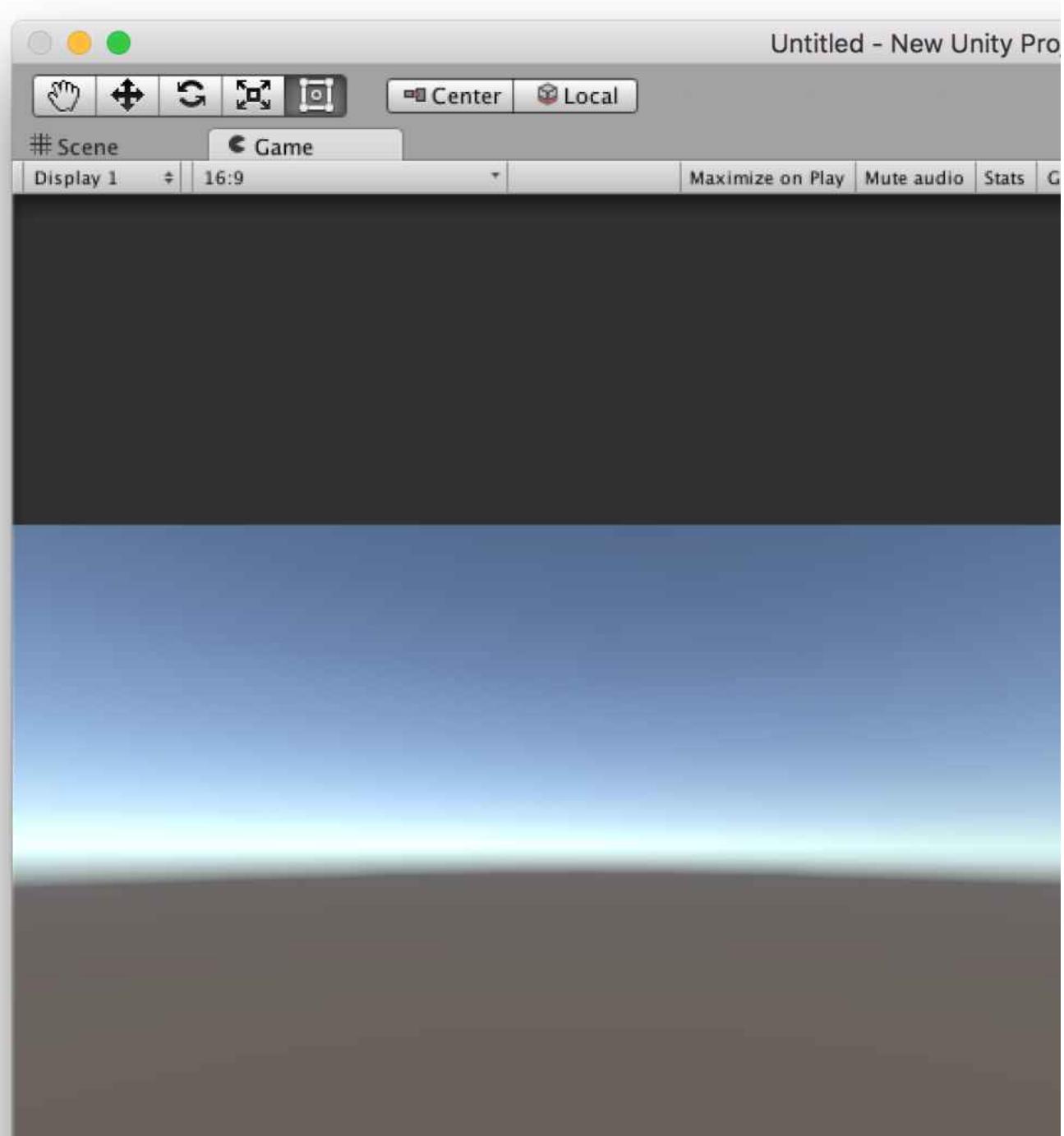
本書では、2016年現在で最も有力なゲームエンジンの1つ、Unityを使ったVRコンテンツ開発を紹介していく。

2.2 Unityとは

Unityを知らない読者はほとんどいないだろうが、ここで改めて紹介しておこう。Unityは様々なプラットフォームへのアプリビルドに対応しているゲームエンジンだ(図2.1)。モバイルゲーム・アプリの分野で数多くのタイトルに採用されていることで知られている。日々アップデートを重ねており、2015年秋にリリースされたバージョン5.1からはOculus VRデバイスのサポートがインテグレートされた。つまり、UnityだけでGear VR用のコンテンツを作ることができるようになったのだ。

また、様々なプラットフォームへのビルドをサポートするUnityなら、1つのソースでGear VRとハコスコ/Google Cardboardで動くアプリを作ることができる^{*1}。

[*1] 動作させるデバイスによって、入力関係の分岐処理やパフォーマンスチューニングが必要



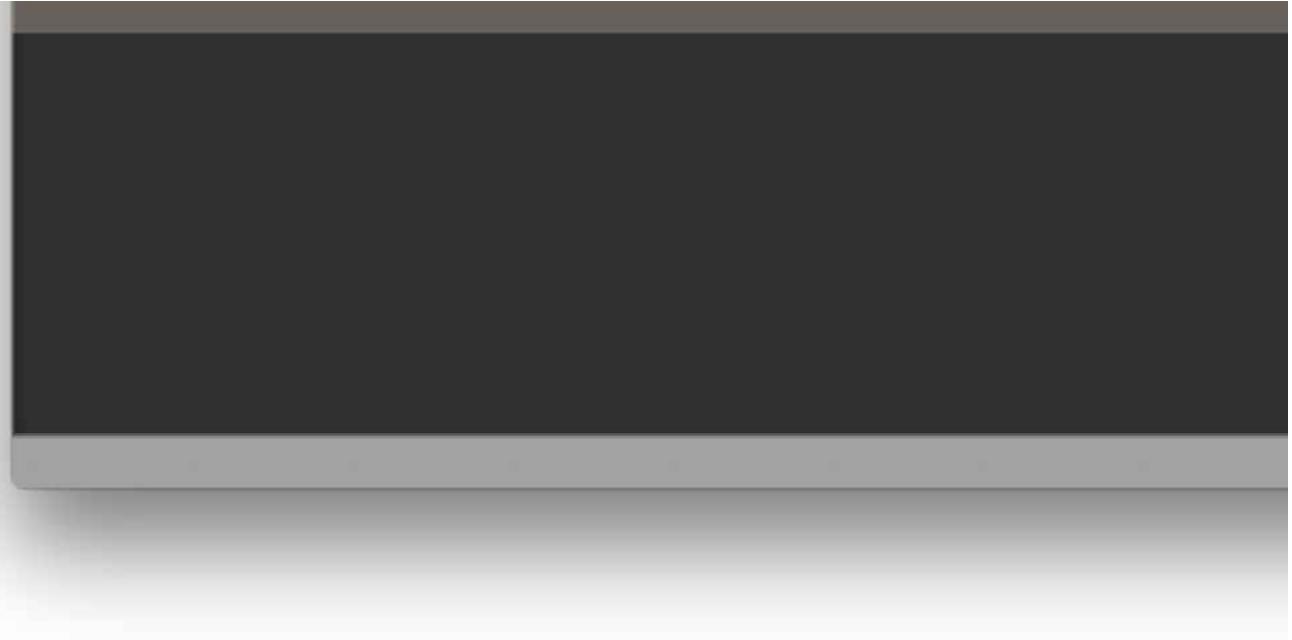


図2.1: Unityの開発画面。画面左のシーンにゲームの要素を配置しながらゲーム作りを進めていく

使用するUnityのバージョンについても気をつける必要がある。

2.3 HMDごとの環境構築作業を確認する

開発環境を構築するにあたり、使用するHMDやモバイルプラットフォームによって必要なものが異なる。Unityは共通して必要になるが、以下の対応表を確認して具体的な作業を行ってほしい。なお、以下で紹介しているものはすべて無料でインストールすることができる。

表2.1: HMD・プラットフォーム別開発対応表

必要なもの	Gear VR	ハコスコ / Google Cardboard (Android)	ハコスコ / Google Cardboard (iOS)
Unity	○	○	○
Android SDK	○	○	×
Osigの作成	○	×	×
Xcode	×	×	○
開発 OS	Windows / mac OS	Windows / mac OS	mac OS

2.4 Unityのインストール

それでは、さっそく開発環境を構築していく。まずはUnityをインストールしよう。と、その前に、開発に使用するPCが次の条件を満たしていることを確認しておこう。

- Windows: Windows 7以上
- mac OS: Yosemite以上

条件の確認が済んだら、Unityをインストールする。2017年8月現在で最新のUnity 2017.1のインストーラをダウンロードしよう。



図2.2: Unityパッチダウンロード

インストールを進めていくと、Unityコンポーネントを選択する画面が表示される(図2.3)。ここでUnityコンポーネントとは、Unityエディタ本体以外のドキュメントやサンプルアセット、他プラットフォームへのビルドに必要なモジュール類のことだ。今回は、最低でも次のコンポーネントにチェックをつけておけば良いだろう。

- Unity 2017.1
- Standard Assets(汎用アセット)
- Android Build Support(Gear VRやAndroid 端末向けにビルドしたい人)
- iOS Build Support(iOS端末にビルドしたい人)

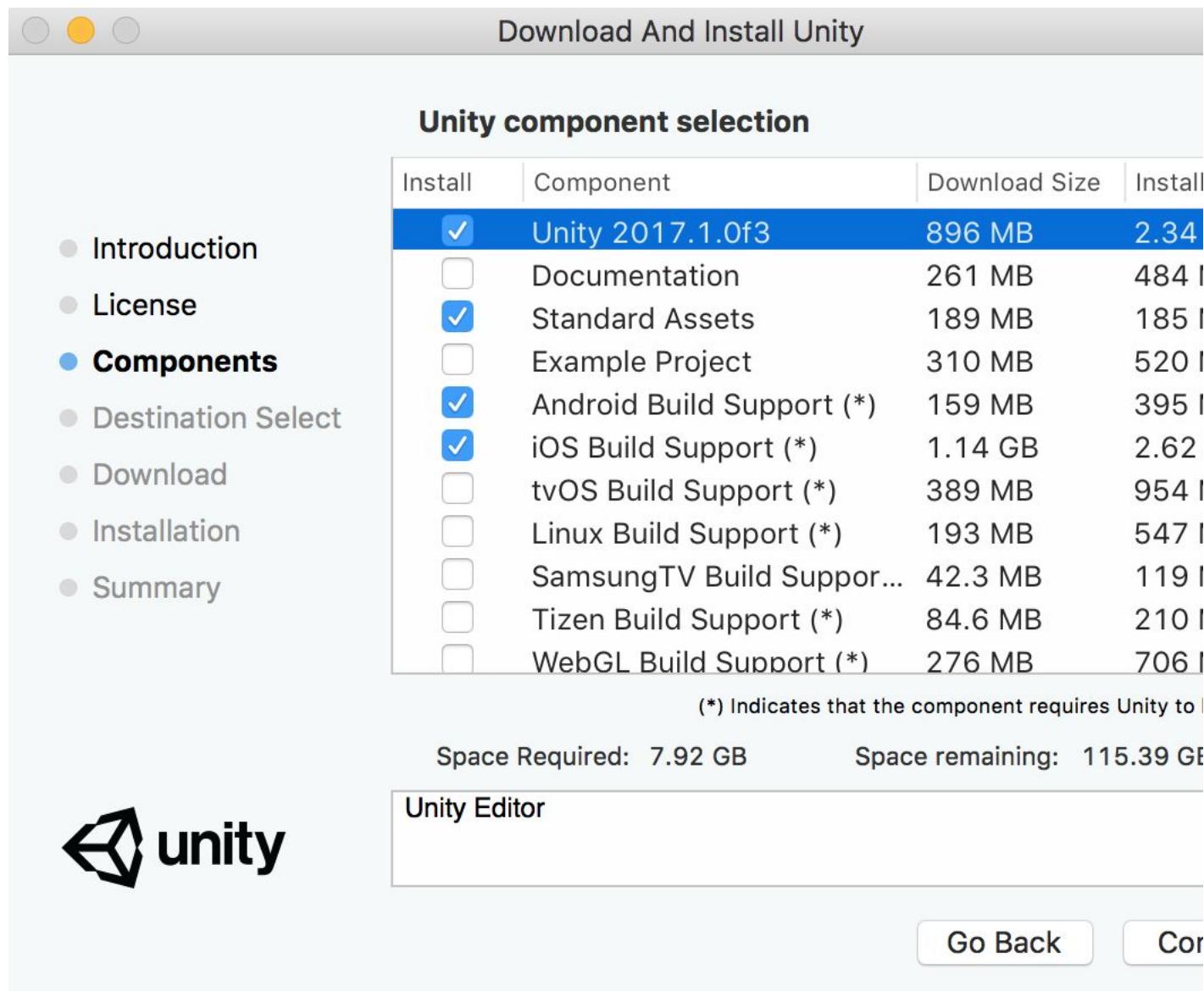


図2.3: インストールするUnityコンポーネントを選択

インストールが終了したらUnityを起動して(図2.4)、試しにUnityプロジェクトを作成したり問題なく使えるか確認しよう。

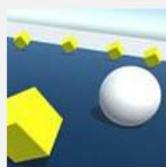


Projects Learn

Tutorials

Resources

Links



Roll-a-ball

Create a simple rolling ball game that teaches you how to work with Unity.



Space shooter

Expand your knowledge and experience of working with Unity by creating an arcade style shooter.



Survival shooter

Learn how to make an isometric 3D survival shooter.



Tanks

Learn how to create a 2 player (one keyboard) tank game.



2D Roguelike

Learn how to make a procedural 2D Roguelike game.

図2.4: インストール後はUnityを起動して正常に動作するか確認しておこう

[column]複数のバージョンのUnityをインストールするには

Unityは頻繁に新機能を取り込みや不具合を修正したバージョンアップを行っているため、異なるバージョンが使われているプロジェクトが複数あると、すべてのバージョンのインストールも必要になる。通常、インストール先のフォルダは“Unity”となるが、インストール後にこのフォルダ名を“Unityバージョン名”というように手動で書き換えれば、複数のバージョンを同時にインストールすることが可能だ(図2.5)。

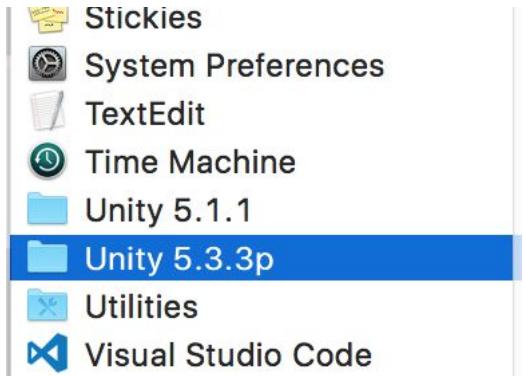


図2.5: 複数バージョンのUnityをインストールした場合のフォルダ構成

なお、Unityプロジェクトはなるべく同一バージョンのUnityで開くようにし、開発途中でバージョンアップを行う場合はプロジェクトのバックアップをとっておくようにしよう。また、複数人で開発を行う場合はメンバーへの周知も行っておこう。異なるバージョンで同一プロジェクトを開くのは様々なトラブルの原因となるため、注意が必要だ。

2.5 Android SDKのインストール

Android OS用のアプリを開発するためには Android SDKをセットアップする必要がある。Gear VRアプリも実のところはAndroidアプリなので、この環境構築は必須だ。Android SDKのセットアップは、Android開発用IDEであるAndroid Studioで行う。

Android Studioの入手とインストール

Windows/mac OSともに、事前にJava SE Development Kit 8 (JDK)*2のインストールが必要になる。忘れず行なっておこう。

次に、Android Studio*3をダウンロードする。(図2.6)。

[*2] <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

[*3] <https://developer.android.com/studio/index.html>

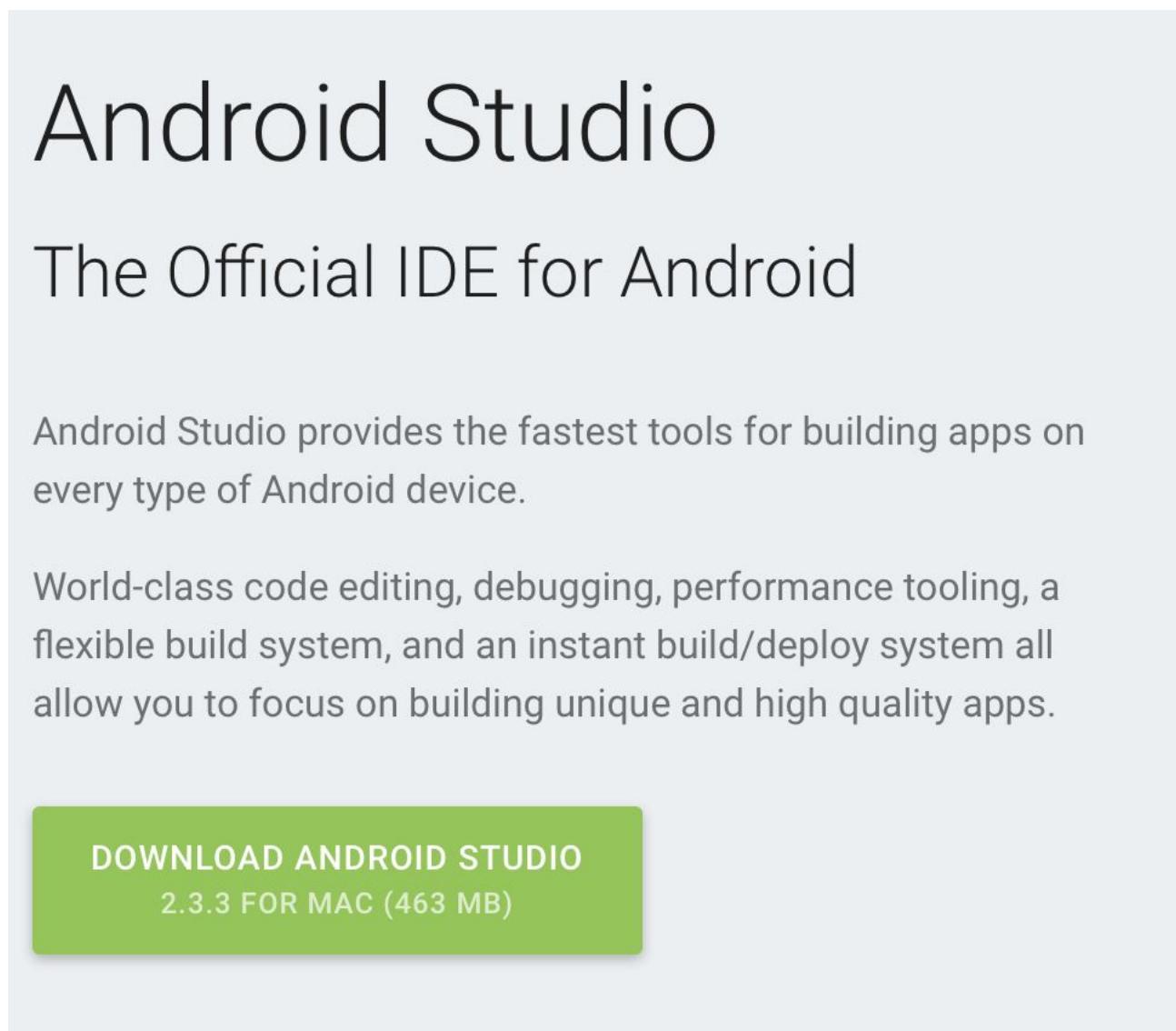
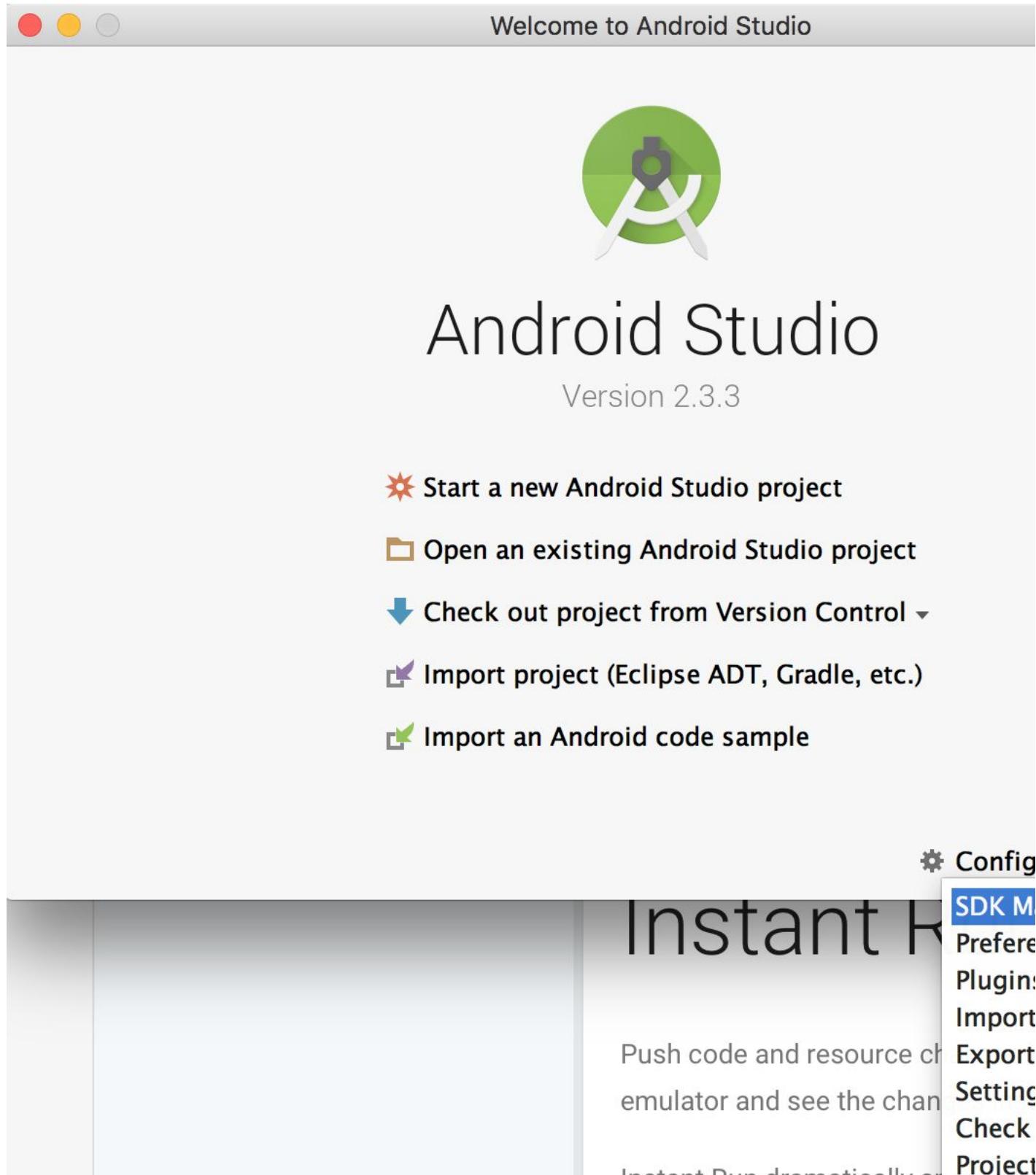


図2.6:「Android Studio」をダウンロード

インストールしたら、Android Studioを起動しよう。プロジェクトのセットアップウィザードが現れるが、ここはキャンセルしておく。

SDK Managerでインストール

Android Studioのウェルカムスクリーンが表示されたら、右下の「configure」からSDK Managerを起動しよう。SDK Managerを通して、Android SDKの各パッケージをダウンロード・インストールしていく。「Android SDK Location」の「Edit」へ進むと、SDKのインストールウィザードが現れる。基本的にNEXTを押下してそのままパッケージをインストールすれば良い。また、ここでのSDKのインストールパスは後で使用するので、控えておこう。



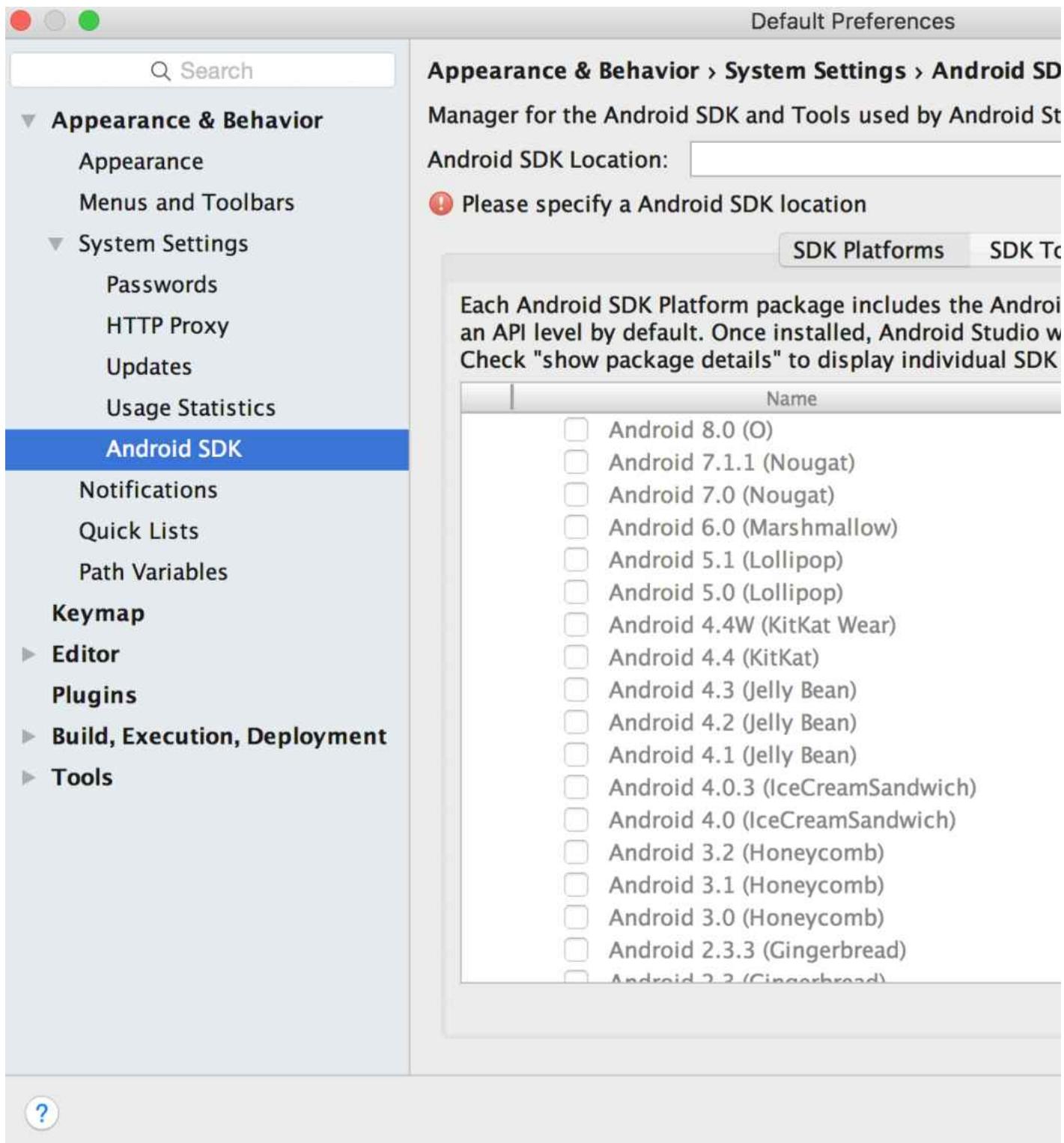


図2.8: SDKをインストール

2.6 UnityにAndroid SDKを設定する

SDKのインストールが完了したら、UnityでAndroid SDKを利用できるようにするための設定を行う。Unityの「Unity Preference」から「External Tools」→「Android」と展開し、「SDK」に先ほど追加したAndroid SDKの場所を指定しておこう(図2.9)。

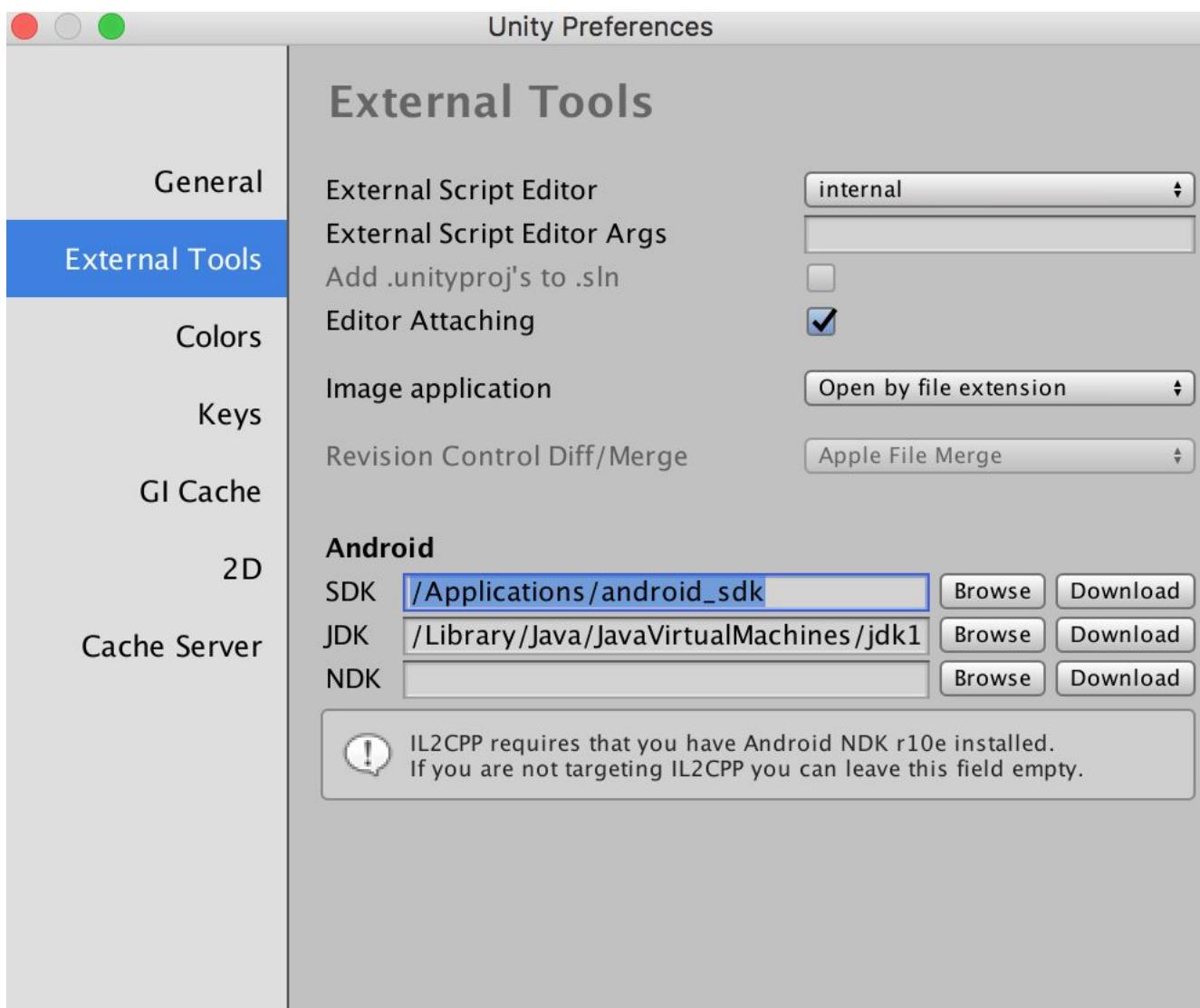


図2.9: UnityにAndroid SDKの場所を指定する

2.7 SDKの不具合対策

Unity 2017.1 と Android Studio経由でインストールするAndroid SDKには、VRサポートでビルドした場合にAndroid Manifest のコンフリクトが起こりビルドができない不具合が確認されている。以下のファイルをダウンロードし、<SDKのフォルダ>/toolsに置き換えることで、この問題を回避できる。

- [macOS: https://dl.google.com/android/repository/tools_r25.2.3-macosx.zip?hl=id](https://dl.google.com/android/repository/tools_r25.2.3-macosx.zip?hl=id)
- [Windows: https://dl.google.com/android/repository/tools_r25.2.3-windows.zip?hl=id](https://dl.google.com/android/repository/tools_r25.2.3-windows.zip?hl=id)

2.8 パスを通す

続けて、コマンドサーチパスにAndroid SDKを登録しよう。これにより、コマンドプロンプト/ターミナル経由で各Android コマンドを実行できるようになる。操作に自信がない場合は、この作業をスキップしても構わない。

Windows の場合は、システム環境変数のPATHに以下のパスを設定する。

```
C:<SDKのフォルダ>\tools\;C:<SDKのフォルダ>\platform-tools\
```

mac OSの場合は、ホームディレクトリの.bash_profileへ以下のようにPATHを記述し、ターミナルを再起動すれば良い。

```
export ANDROID_HOME=<SDKのフォルダ>
export PATH=$PATH:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
```

パスを設定できたら、試しにコマンドプロンプト/ターミナルからadbコマンドを叩いてみよう。図2.10のようなヘルプが表示されれば“OKだ”。

```
[Shunsukes-MacBook:~ shunsukesakai$ adb
Android Debug Bridge version 1.0.32
Revision 09a0d98bebce-android

-a                         - directs adb to listen on all interfaces for a connection
-d                         - directs command to the only connected USB device
-e                         - directs command to the only running emulator.
                           returns an error if more than one emulator is running
-s <specific device>      - directs command to the device or emulator with
                           serial number or qualifier. Overrides ANDROID_SERIAL
                           environment variable.
-p <product name or path> - simple product name like 'sooner', or
                           a relative/absolute path to a product
                           out directory like 'out/target/product/sooner'.
                           If -p is not specified, the ANDROID_PRODUCT_OUT
                           environment variable is used, which must
                           be an absolute path.
-H                         - Name of adb server host (default: localhost)
-P                         - Port of adb server (default: 5037)
devices [-l]                 - list all connected devices
                           ('-l' will also list device qualifiers)
connect <host>[:<port>]    - connect to a device via TCP/IP
                           Port 5555 is used by default if no port number
                           is specified
disconnect [<host>[:<port>]] - disconnect from a TCP/IP device.
                           Port 5555 is used by default if no port number
                           is specified
                           Using this command with no additional arguments
                           will disconnect from all connected TCP/IP devices

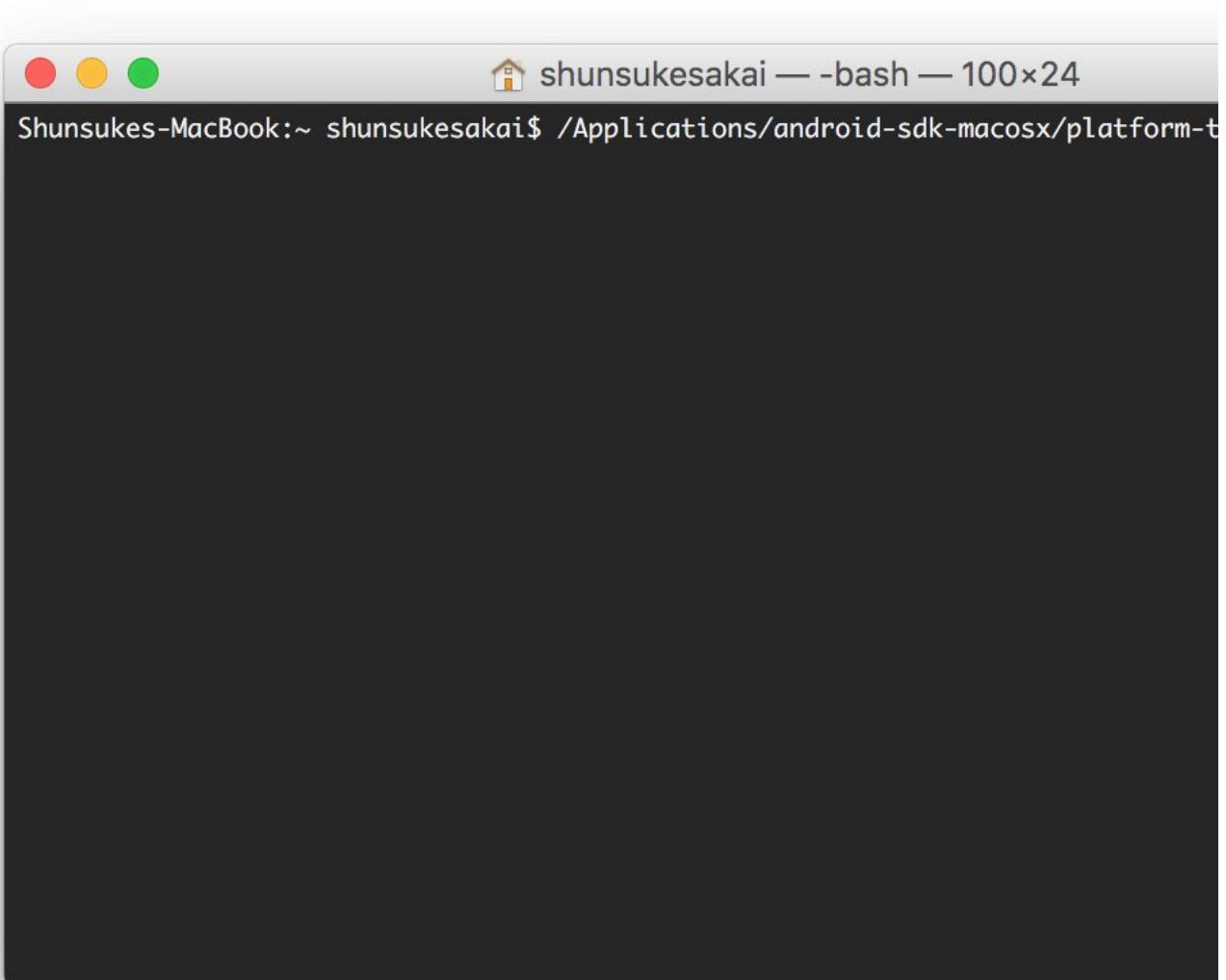
device commands:
adb push [-p] <local> <remote>
          - copy file/dir to device
          ('-p' to display the transfer progress)
adb pull [-p] [-a] <remote> [<local>]
```

図2.10: コマンドを叩いてヘルプが表示されればパスは正しく通っている

2.9 osig ファイルの作成(Gear VRのみ)

Gear VR対応のVRコンテンツを開発する場合はosigファイルを作成する。Gear VRの開発版アプリには「Oculus Signature File(osig)」と呼ばれる証明ファイルを含める必要がある。作成手順は次の通りだ。

- `adb devices` コマンドでGalaxy端末のDevice IDを表示する
- Andriod SDKにパスが通っていない場合は<Android SDKフォルダ>/platform-tools/adbファイルをコマンドプロンプト/ターミナルにドラッグ・アンド・ドロップし、続いて`device`を入力する(図2.11)。



```
shunsukesakai — -bash — 100x24
Shunsukes-MacBook:~ shunsukesakai$ /Applications/android-sdk-macosx/platform-t
```

図2.11: `adb devices`コマンドでGalaxy端末のDevice IDを表示

- device IDを[Oculus Signature File \(osig\) Generator](#)のフォームに入力する(図2.12)

- ファイルをダウンロードする

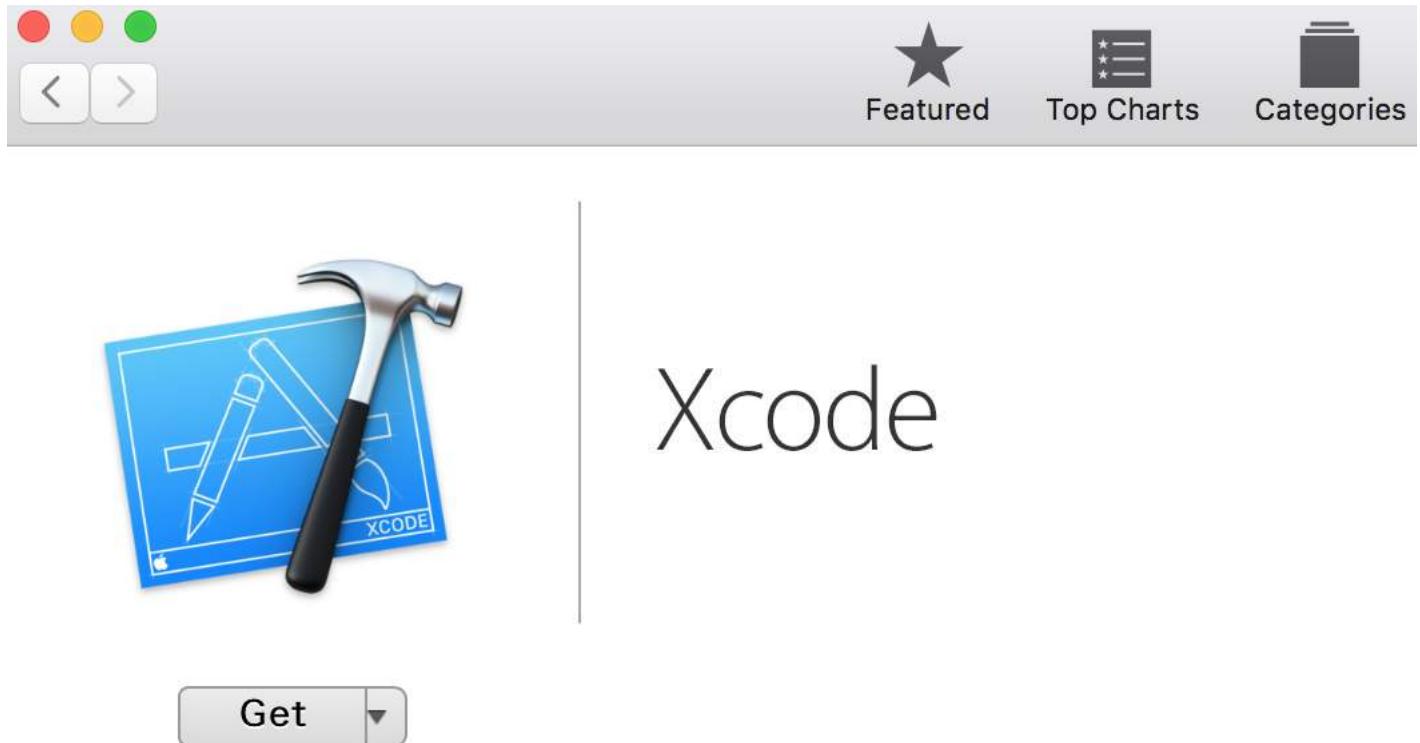


図2.12: device IDをOculus Signature File (osig) Generatorのフォームに入力

このファイルは、後ほどUnityエディタでの開発時に使用するため保管しておこう。また、開発に使用するGalaxy端末毎に作成する必要があることにも注意してほしい。

iOS向けにアプリをビルドする場合は、mac OS専用のIDEである「Xcode」が必要だ。mac OSのApp StoreからXcodeを検索してインストールしよう(図2.13)。

なお、Xcode7からビルドが無料で行えるようになったが、アプリの配布には有償の「Apple Developer Program」に加入した上で、各種証明ファイルなどを作成する必要がある。



Xcode 4+

Essentials

Xcode includes everything developers need to create great applications for Mac, iPhone, iPad, and iPod touch. Xcode provides developers a unified workflow for user interface design, coding, testing, and debugging. Swift and Objective-C combined with the Swift programming language make developing apps easier and more fun. ...

図2.13: Xcodeのダウンロード(App Store)

これで、Unityを使ったモバイルVRコンテンツの開発環境は整った。次章は、サンプルプロジェクトを使ってGearVR・Google Cardboard・ハコスコ向けにアプリをビルドしてみよう。

第3章 サンプルプロジェクトをビルドしてみよう

前章は、モバイルVRの開発に必要な環境をセットアップする方法を解説した。今回は、Unityから実機にサンプルプロジェクトをビルドして、実際にヴァーチャル空間に入ってみよう。

3.1 サンプルプロジェクト入手する

Unityを通して、様々な便利ツールやアートアセットをダウンロードできる仕組みが「Unity Asset Store」だ。もちろん、このAsset StoreからVRの開発に役立つアセットも入手できる。中でも特におすすめしたいのが、Unityがオフィシャルで提供するアセット「VR Samples」だ（図3.1）。このサンプルアセットには、VRの開発に役立つ便利ツールだけでなく、銃や乗り物、UIといったヴァーチャル空間での使用に適したアートアセットも梱包されているのだ。しかも、ありがたいことに無料で提供されている。このVR Samplesを使って、UnityにおけるVR開発のはじめの一歩を踏み出してみよう。

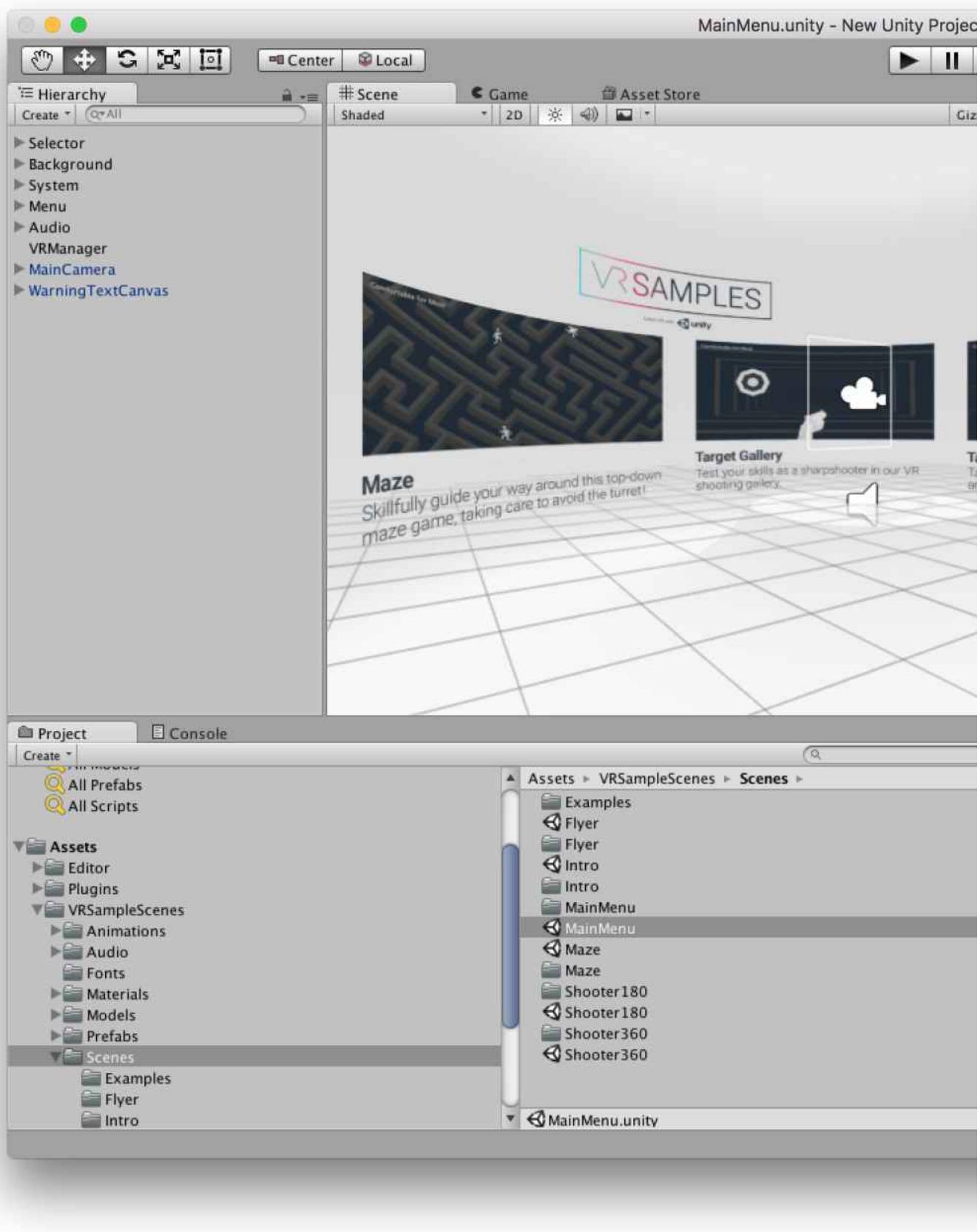


図3.1: Unity VR Samples

3.2 VR Samplesのダウンロードとインポート

それでは、早速Asset StoreからVR Samplesをダウンロードしてインポートしてみよう。次の手順で操作を行なってほしい。

- 操作を始める前に、あらかじめ新規のプロジェクトを作つておこう。その後、Unityのメニューから「Window」→「Asset Store」を選択してアセットストアを開く。このとき、Unityアカウントでログインしておく必要がある。アセットストアが開いたら、画面上のサーチボックスに「VR」と入力して検索しよう(図3.2)。

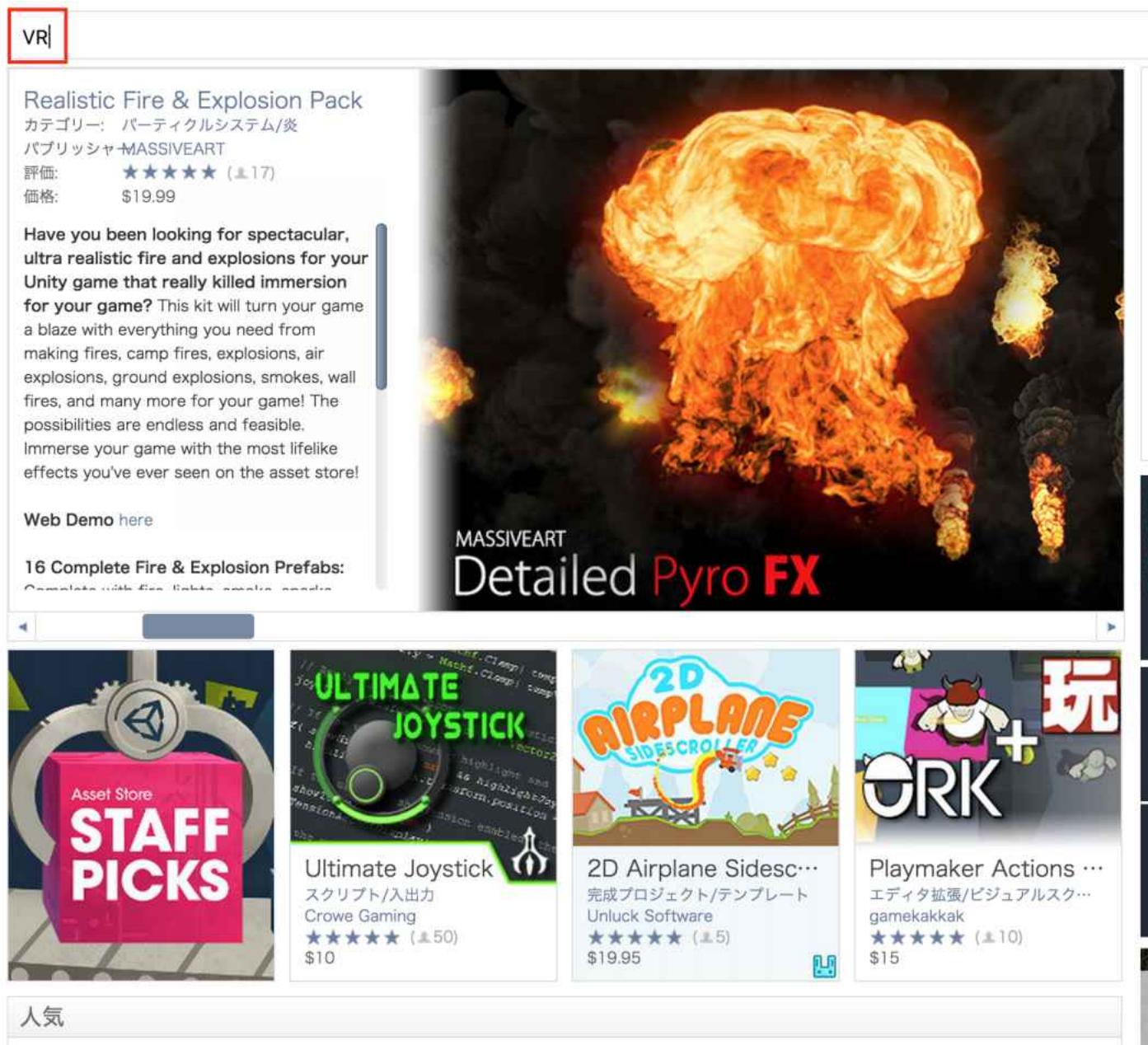


図3.2: アセットストアのサーチボックスに「VR」と入力して検索

- 検索結果が表示されるので、VR Sampleを見つけよう(図3.3)。見つけたらダブルクリックして開き、画面左上の[ダウンロード]ボタンをクリックすれば、自身のUnityアカウントにダウンロードされる(図3.4)。

VR ✕

最高価格

\$

最大サイズ

FREE

5

10

20

50

100

200

∞

1MB

5MB

無料のみ

有償アセットのみ

リリース日

最小評価



対応する Unityのバージョン



< 5.3.3

e.g. 5.2.0

1d

7d

アップデート日

パッケージのみ

リストのみ

1d

7d

以下の項目でソート 関連性 / 人気 / 名前 / 値段 / 評価 / アップデート

1 2 3 4 5 6 1 - 36 of 206



VR Samples

Unity Essentials/Sa…

Unity Technologies

★★★★★ (▲144)

無料

AR&VR

Speech
Recognition
Plugin

VR Speech Plugin

スクリプト/ネットワ…

NAOKI MIYACHI

評価が不足しています

\$35

NAOKI MIYAC



VR Panorama 360 …

スクリプト/ビデオ

OliVR

★★★★★ (▲47)

\$40



LC VR Kit

スクリプト

Laurel Code Inc.

評価が不足しています

\$20



LC VR Kit Free

図3.3: VR Sampleを見つけたらダブルクリックして展開



VR Cardboard

VR Samples

カテゴリ: Unity Essentials/Sample Projects
パブリッシャー: Unity Technologies
評価: ★★★★☆ (144)
価格: 無料

[ダウンロード](#)

VR開発をすぐに始められるUnity製のVRサンプルパックです。

このVRサンプルはOculus Rift DK2とGear VR向けのコンテンツ開発に役立つ、メニューと4つのミニゲームが入った売り物のゲーム集のような感じで構成されています。

このサンプルプロジェクトはLearnセクションに新設された新しいチュートリアルとセットになっています。

これらのチュートリアル記事は私たちのVRサンプルゲームがどのように構築されているのかを解説すると同時に、VRを始めたい初心者向けのアドバイスなどがまとめられています。

チュートリアル記事はこちらからご覧いただけます。



SAMPLES
VR Samples

VR ESSENTIALS



バージョン: 1.1a (Mar 14, 2016) サイズ: 138.8 MB ライセンスを表示

サポート ウェブサイト パブリ:

初版リリース日: 8月 2015

サ

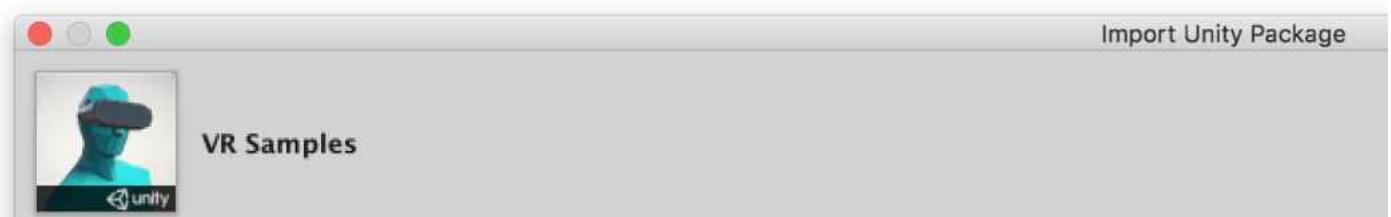
このパッケージはUnity 5.3.0, and 5.4.0で作されました。

パッケージ内容

- [Editor](#)
- [CrossPlatformInput](#)

図3.4: [ダウンロード]ボタンをクリックして自身のUnityアカウントへダウンロード

3. ダウンロードが完了したら、[Import]ボタンをクリックしてプロジェクトにインポートしよう。



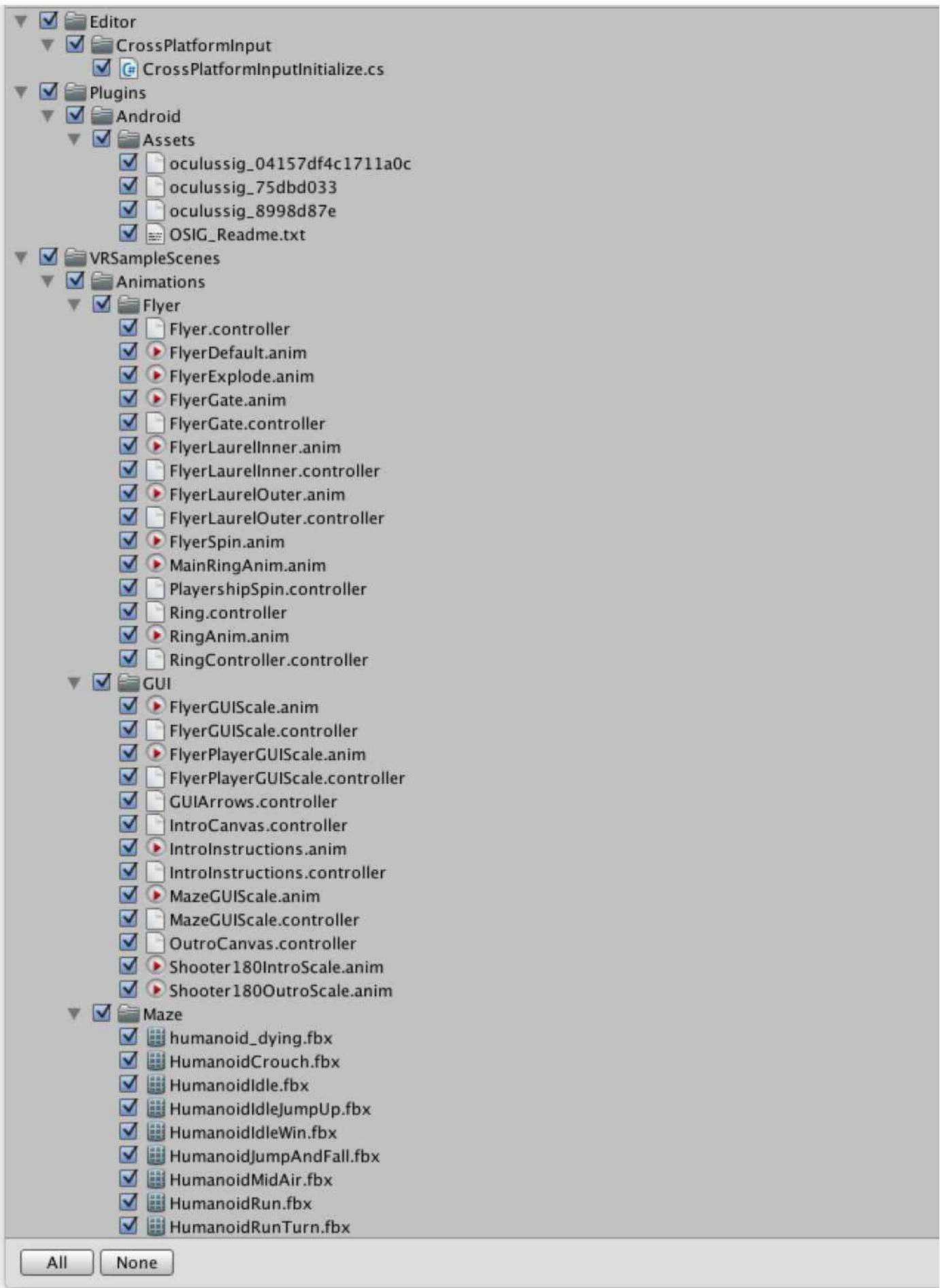


図3.5: ダウンロード完了後、Importボタンでプロジェクトにインポート

3.3 Gear VR向けにビルドする

サンプルプロジェクトのインポートが済んだら、いよいよGear VR向けにビルドを行ってみよう。なお、この手順で何かしらトラブルが起こった場合、第2章で紹介した環境構築も含めて、もう一度見直してみてほしい。

また、ハロスコ/Google Cardboard向けにビルドするには、これにさらに手を加える必要がある。そのためにはUnityに関する解説をもう少しだけ行う必要があるため、次章の解説とさせていただきたい。

1. Unityを起動し、まずは第2章で作成した.osigファイルをPlugins/Android/Assetsの中に格納しておこう(図3.6)

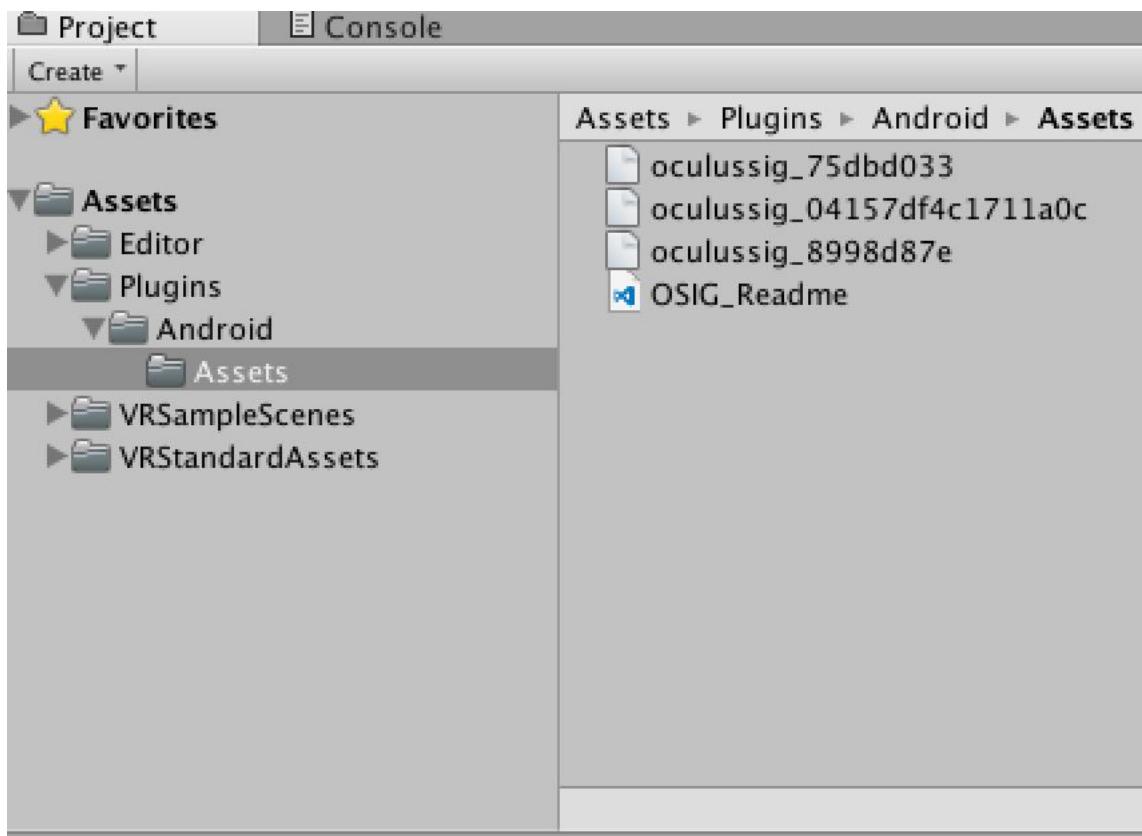
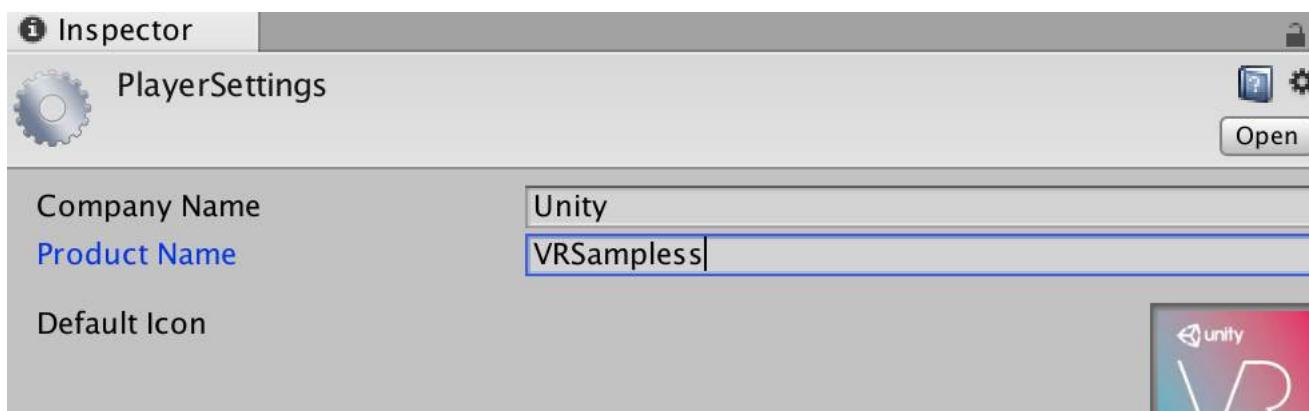


図3.6: .osig ファイルをPlugins/Android/Assetsに格納

2. 「File」→「Build Settings」と選択してビルド設定を開く
3. 「Build Settings」画面から、「Player Settings」を開く。Virtual Reality Supported にチェックが入っていることを確認し、Virtual Reality SDKsに「Oculus」を入れる



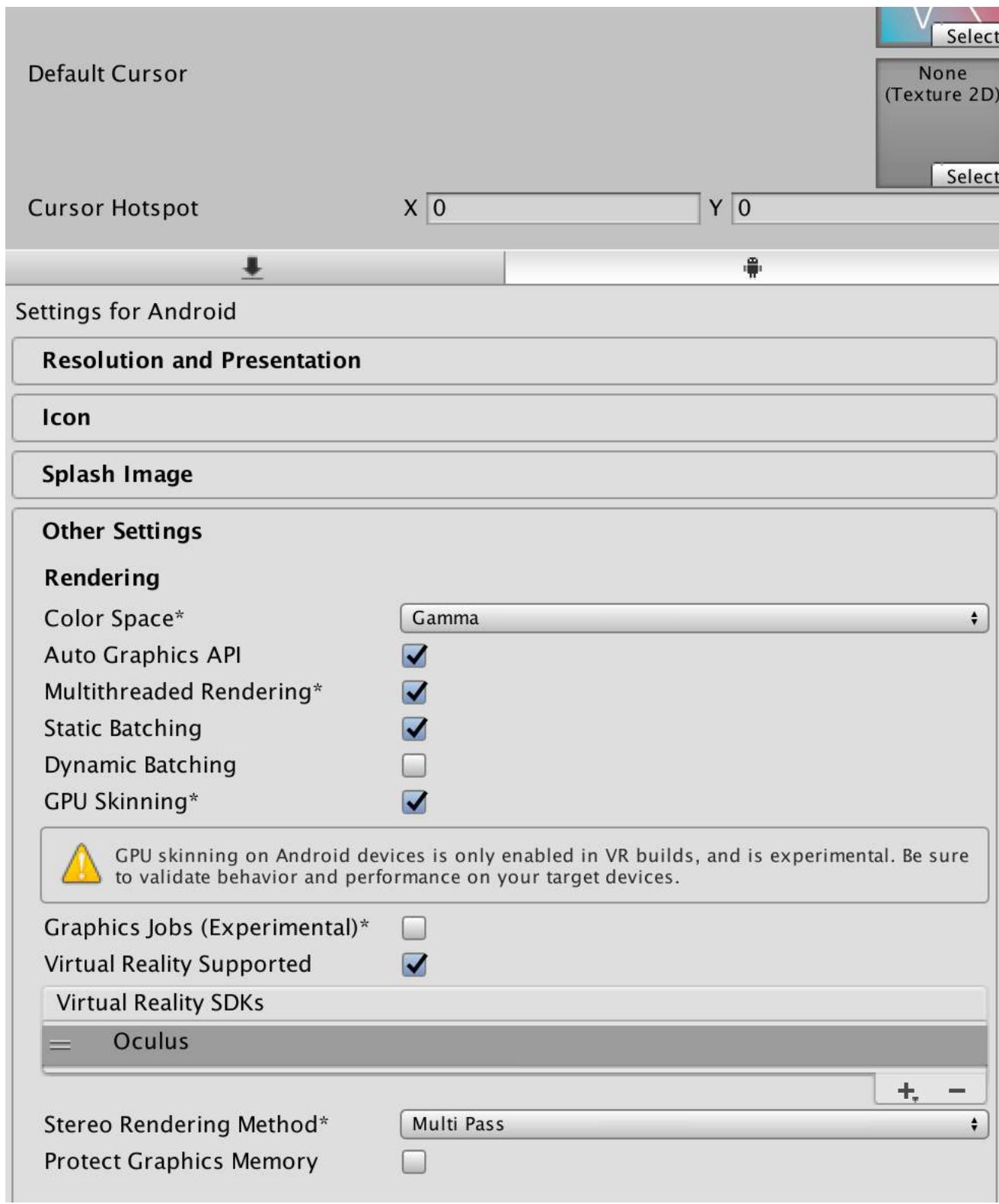


図3.7: Player Settingsを確認

4. Platformから「Android」を選択し、画面左下の[Switch Platform]をクリックする。しばらくして、ビルドプラットフォームが「Android」に変わったことを確認する(図3.8)

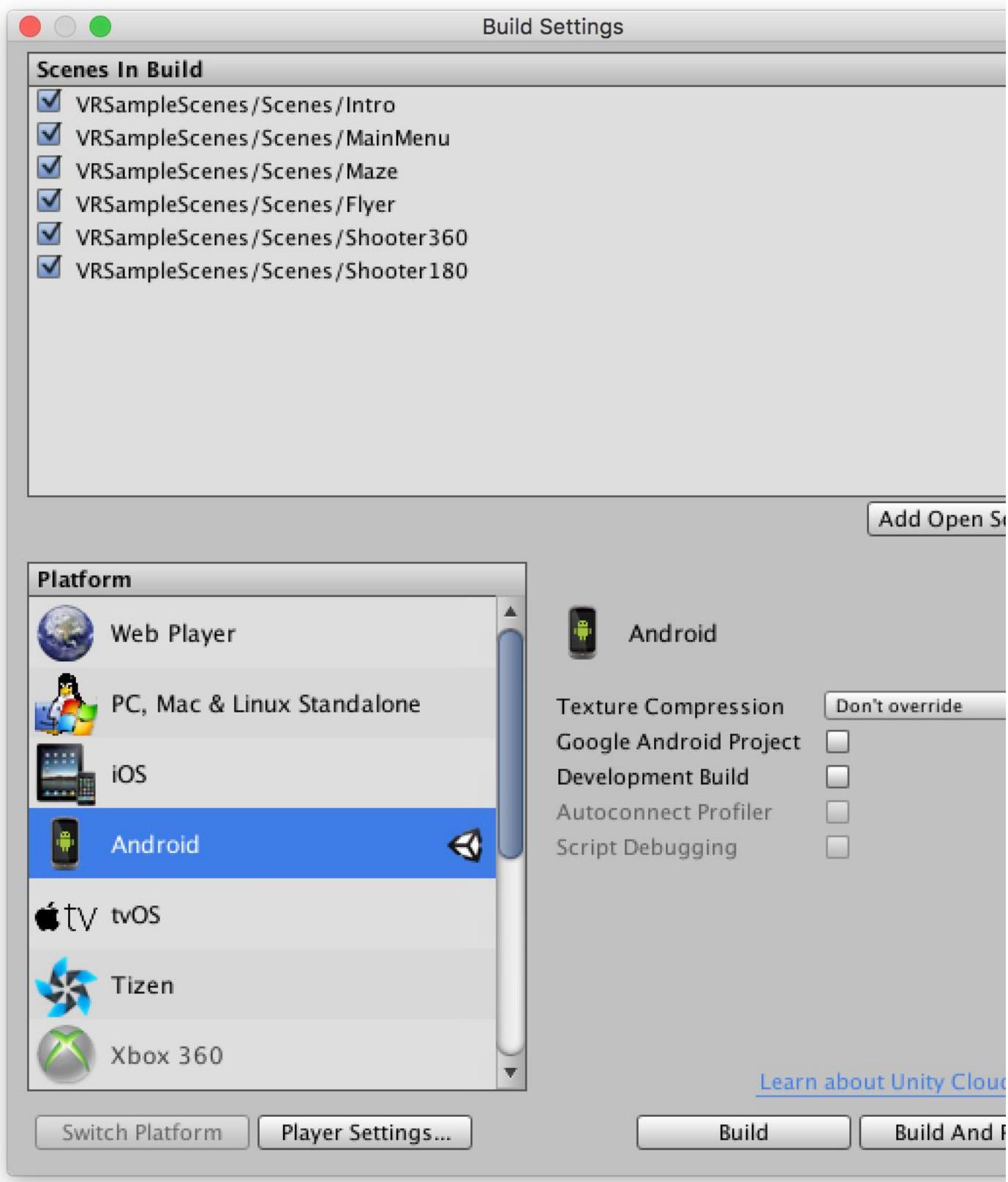


図3.8: ビルドプラットフォームを「Android」に変更する

5. 画面7の右下にある[Build]もしくは[Build And Run]ボタンをクリックし、これからビルドする.apkファイルの保存先を指定する(図3.9)

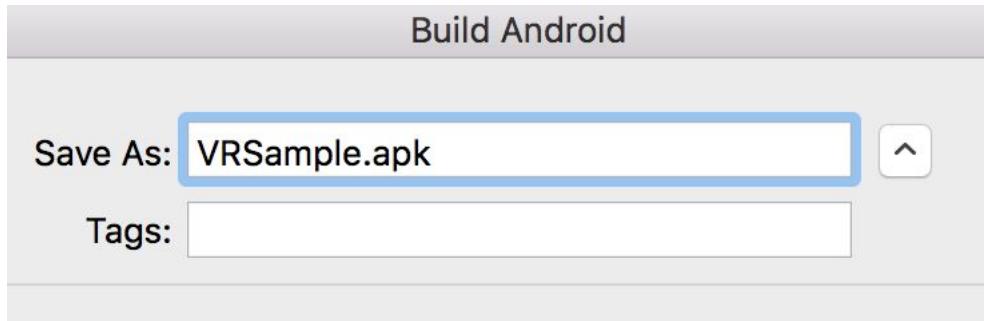


図3.9: ビルドする.apkファイルの保存先を指定

6. ビルドに問題がなければ、3で指定した名前の.apkファイルが生成される。[Build And Run]ボタンをクリックし、かつ端末が適切に接続されていれば.apkファイルが自動で端末に転送されるはずだ(図3.10)。



図3.10: .apkファイルが端末に転送され「VR Samples」が起動する

7. なお、.apkのみがビルドされた場合は、次のようにadbコマンドを使って実機に転送しよう。

```
adb install ***.apk
```

8. GearVRに装着し、さっそく遊んでみよう(図3.11)。基本的にはGear VRの横のタッチボタンで先にすすめる他、タッチパッドを接続している場合はそちらも使用できる。

端末を挿入

このアプリケーションを起動するには、端末をGear VRに挿入します。

キャンセ

図3.11: .apkファイルが端末に転送され「VR Samples」が起動する

[column]Unityエディタ上でVRの確認を行うには

ここまで、モバイルデバイス上でVRの確認方法を紹介してきたが、「いちいちビルドするのは面倒だ」と思う人もいるはずだ。そこで、開発PCに「Oculus DK2」や「Oculus Rift」といったOculusデバイスを取り付けると、直接Unityエディタ上にVRを表示できる。開発のイテレーションを早めるためには、これらのデスクトップ用デバイスを利用するのも手だ。実際には、次の手順で操作をしていこう。

1. PCにOculusデバイスを接続し、必要なランタイム等をインストールする
2. Unityのメニューから[Edit]→[Project Settings]→[Player Settings]を選択する
3. 「PC,Mac&Linux Standalone」であることを確認し、「Other Settings」の「Virtual Reality Supported」にチェックを入れる

4. Virtual Reality SDKsに「Oculus」を入れる

これで、サンプルプロジェクトをGear VR向けにビルドできるようになった。「やっとVR開発のはじめの一歩」を踏み出せたと行っても良いかも知れない。しかし、オリジナルのVRコンテンツを生み出すためには、もう少し開発環境であるUnityの仕組みを知らなくてはならない。また、今回は触れられなかったハコスコ/Google Cardboardの解説も、次章で併せて行っていく。

第4章 Unityの仕組みを理解しよう

これからUnityを使ってVRコンテンツを作っていくわけだが、そのためにはまずUnityの基本的な仕組みを理解しておく必要がある。Unityに関する情報はすでに多くのWebメディア・書籍などで紹介されているため詳細な解説は割愛するが、今回はUnityをつとり早くマスターするために、次の2つに分けて解説していく。

4.1 Unityエディタ

「Unityエディタ」とは、文字通りアプリケーションとしてのUnityそのものである。Unityエディタは様々なウィンドウやビューで構成されており、それぞれの役目に応じた機能を提供している(図4.1)。例えば、3Dオブジェクトを配置してVR空間を構成したり、パラメータを調整してゲームの難易度を調整するなど、グラフィック・レベルデザイン領域の作業は、このUnityエディタを介して行うことが多い。なお今回は取り扱わないが、このUnityエディタは独自に実装することで、容易に拡張することもできる。

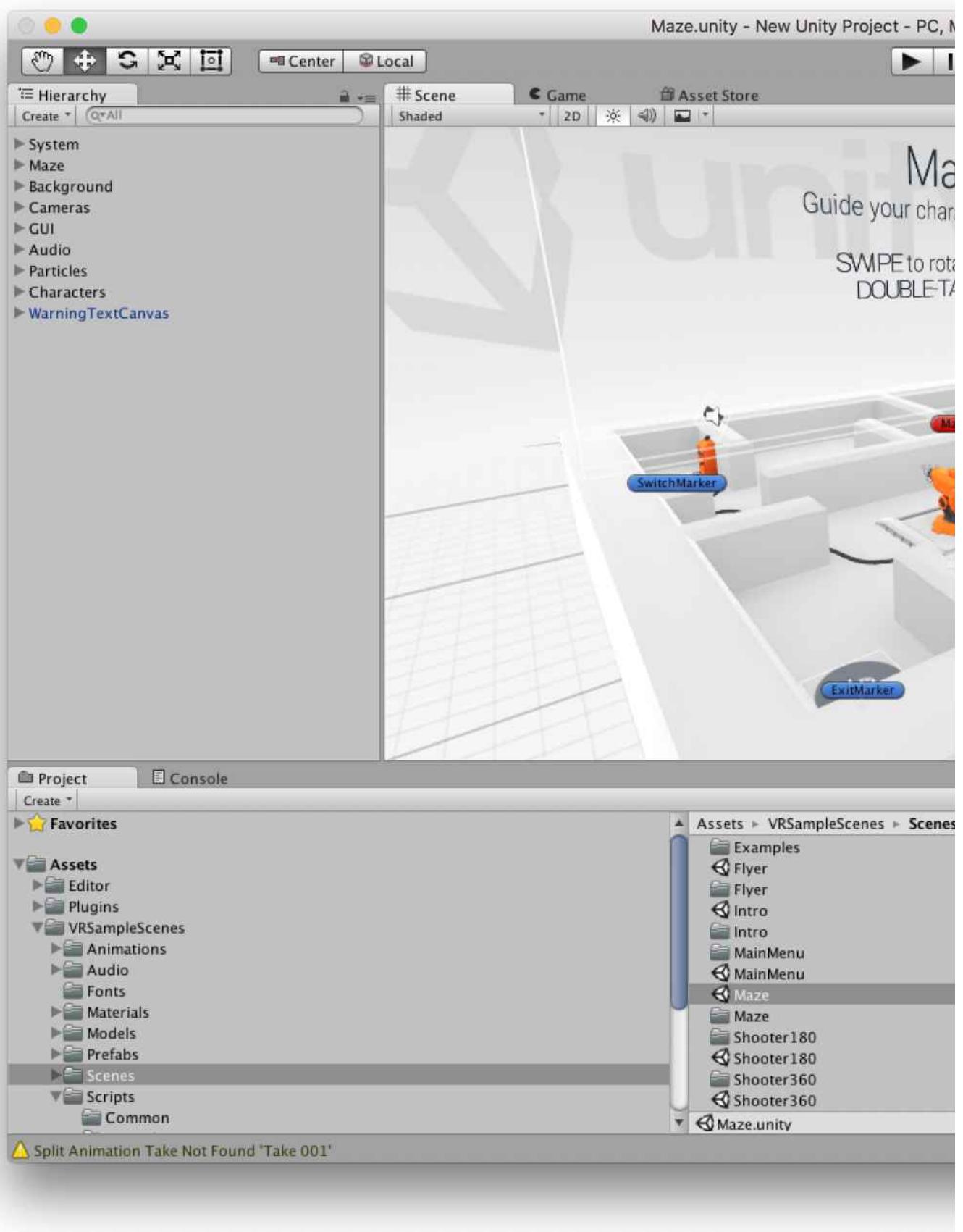


図4.1: お馴染みのUnityエディタ。Unityバージョン3の時からほとんど変わらない

4.2 Unityスクリプト

「Unityスクリプト」とは、Unityでゲームを作るためのプログラムのことだ(図4.2)。例えば、シューティングゲームを作る場合を想定してみると、次のようなシーンが考えられるだろう。

- プレイヤーがボタンを押して弾丸を発射する
- 発射した弾丸が敵にヒットする
- 敵が爆発する

このようなゲーム内で発生するさまざま現象は、すべてUnityスクリプトで制御することになる。これらをゲームロジックと呼び、ゲームロジックを組み合わせることで1つのゲームが完成するのだ。

Unityスクリプトの記述にはC#とJavaScriptのどちらかを使用できる。1つのUnityプロジェクトでこれらを混在させることもできるが、その場合には多くの制約があり現実的ではない。そのため本書では、C#をUnityスクリプト言語として扱っていく。C#のほうが処理速度・ドキュメントの充実度などが優れている上、実際のゲーム開発現場でJavaScriptが採用されるケースは少ないからだ。

```
1 using UnityEngine;
2 using UnityEngine.VR;
3
4 namespace VRStandardAssets.Examples
5 {
6     // This script shows how gameobjects can react to
7     // the rotation of the user's head. It tilts the
8     // gameobject's transform so it's front edge is
9     // perpendicular to the user's line of sight.
10    public class ExampleRotation : MonoBehaviour
11    {
12        [SerializeField] private float m_Damping = 0.2f;
13        [SerializeField] private float m_MaxYRotation = 20f;
14        [SerializeField] private float m_MinYRotation = -20f;
15
16        private const float k_ExpDampCoef = -20f;
17
18        private void Update()
19        {
20            // Store the Euler rotation of the gameobject.
21            var eulerRotation = transform.rotation.eulerAngles;
22
23            // Set the rotation to be the same as the user's
24            eulerRotation.x = 0;
25            eulerRotation.z = 0;
26            eulerRotation.y = InputTracking.GetLocalRotation();
27
28            // Add 360 to the rotation so that it can effectively
29            if (eulerRotation.y < 270)
30                eulerRotation.y += 360;
31
32            // Clamp the rotation between the minimum and maximum
33            eulerRotation.y = Mathf.Clamp(eulerRotation.y, 36);
34
35            // Smoothly damp the rotation towards the newly calculated value
36            transform.rotation = Quaternion.Lerp(transform.rotation,
37                Quaternion.Euler(eulerRotation.x, eulerRotation.y, eulerRotation.z),
38                m_Damping * (1 - Mathf.Exp(k_ExpDampCoef * Time.deltaTime)));
39        }
40    }
41}
42}
```

図4.2: こちらはMonodevelop。Windowsでは「Microsoft Visual Studio」が使用できる(後述)

なお、プログラム言語としてのC#については解説しないが、本書では既存のスクリプトアセットを利用するなどして、複雑なプログラミングを扱わないので安心してほしい。目安としてクラス・変数・関数(メソッド)などの概念が理解できれば十分だ。

4.3 Unityエディタを使いこなそう

早速、Unityエディタから見ていこう。ここでは、それぞれのウィンドウやビューの役割と、マウスやトラックパッド等をつかって3D画面を動かす方法を解説する。

6つのメインウィンドウ

Unityエディタの画面は、主に以下の6つの画面で構成されている。

1. プロジェクトウィンドウ(図3)

プロジェクトに必要なアセットを管理するウィンドウ。アセットとはゲーム中で使用する3Dモデル、アニメーション、テクスチャ、サウンドなどのバイナリファイルに加えて、Unityスクリプトのファイル(クラス)を指す。また、この画面は開発マシン上のスペースを指している。例えば、プロジェクトウィンドウ上で右クリック→Reveal in Folderを実行するとエクスプローラー/Finderが開き、Unityプロジェクトを保存したスペースのAssetsフォルダ以下であることが確認できる。ちなみに、Assetsフォルダ以下はエクスプローラー/Finderと完全に同期しており、フォルダ上で直接ファイルの追加・削除等を行うとUntiyのプロジェクトウィンドウでも実行される。

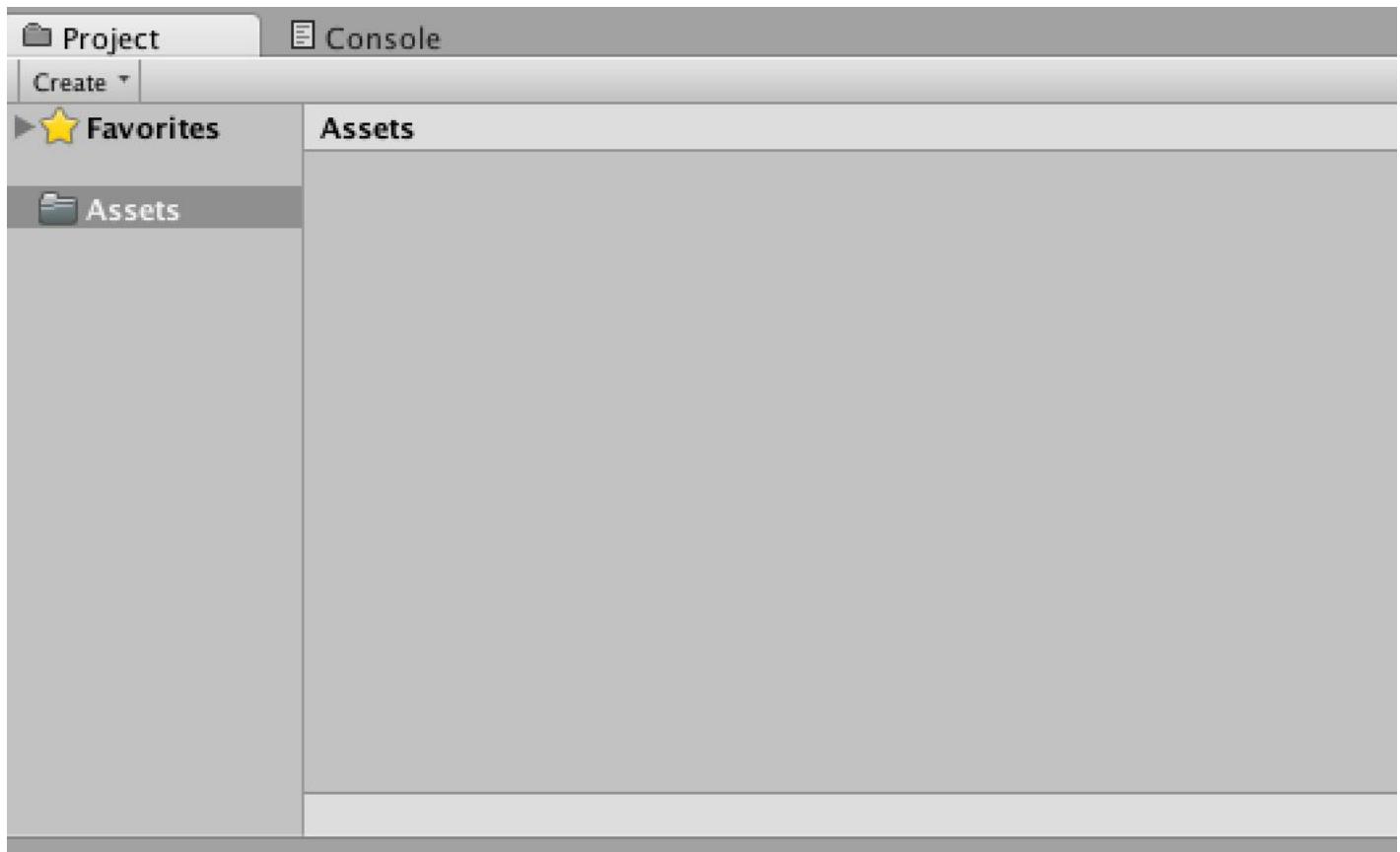


図4.3: プロジェクトウィンドウとAssetsフォルダの中身が同一であることを確認しておこう

2. シーンビュー(図4)

Unityのシーンはゲーム上の3D空間であり、この空間を表示しているのが「シーンビュー」だ。3Dオブジェクトを配置して空間の背景を作り、そこに敵やギミックなどの小道具を配置する、といった作業はこのシーンビューで行う。また、Unityではシーンビューに配置される項目のことをゲームオブジェクトと呼ぶ。このシーンビューは操作が若干特殊なため、詳細は後述の「3Dの操作に慣れよう」で確認してほしい。

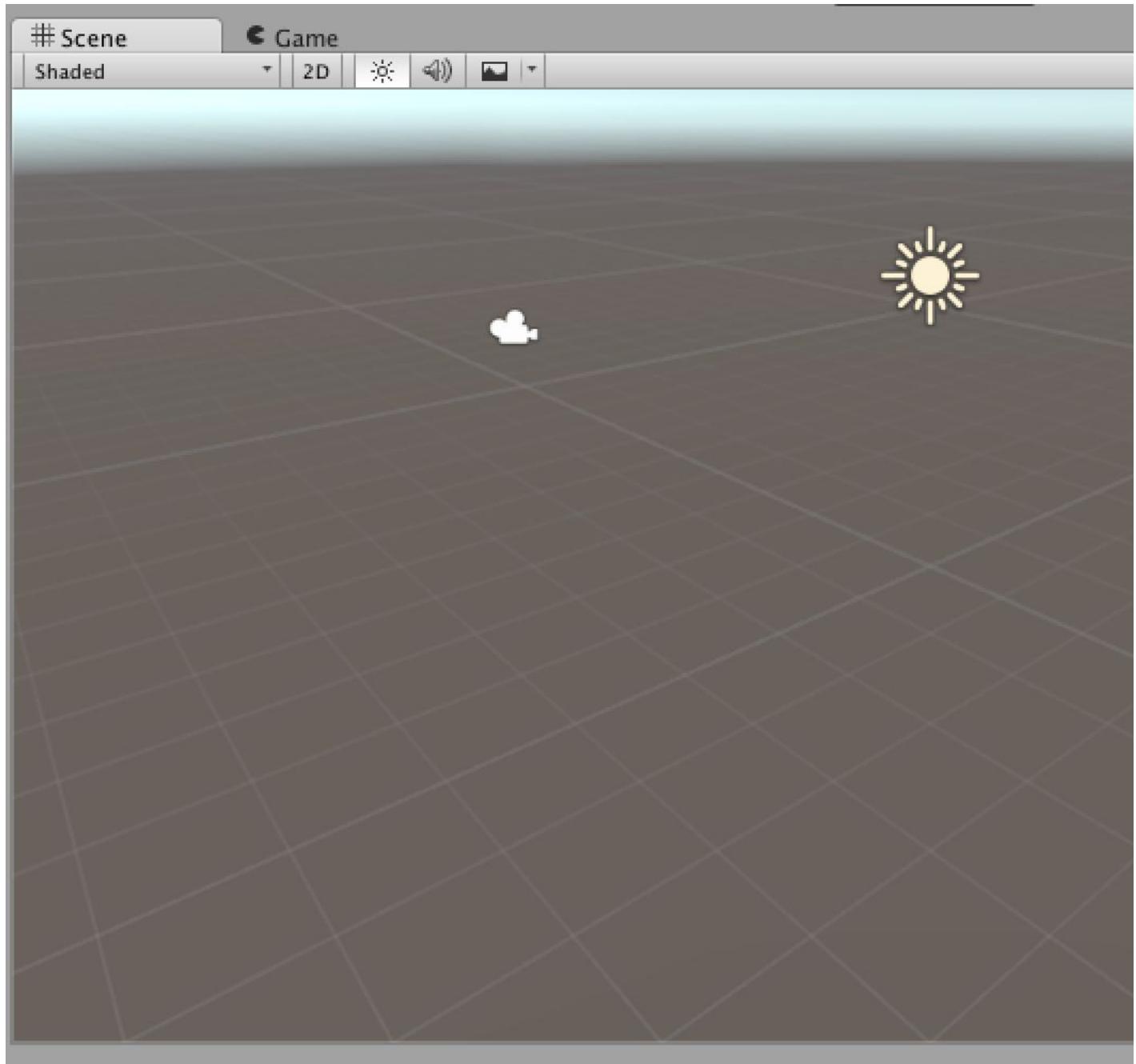


図4.4: 3D操作をマスターしておこう

3. ヒエラルキーウィンドウ(図5)

「ヒエラルキーウィンドウ」には、シーンビューと同一のゲームオブジェクトが表示されている。つまり、ゲームオブジェクトをグラフィックで表示したものがシーンビュー、文字で表示したものがヒエラルキーウィンドウと考えてもよい。ヒエラルキーの言葉が表すように、階層関係(親子関係とも呼ぶ)が文字列で表示されている。ゲームオブジェクトをまとめて操作する場合などは、こちらのウィンドウで行ったほうがやりやすい。また、上部に表示されている「サーチボックス」を使えば、ゲームオブジェクト名で目当てのゲームオブジェクトを絞り込むことも可能だ。目的に応じてヒエラルキーウィンドウとシーンビューを使い分けることになる。

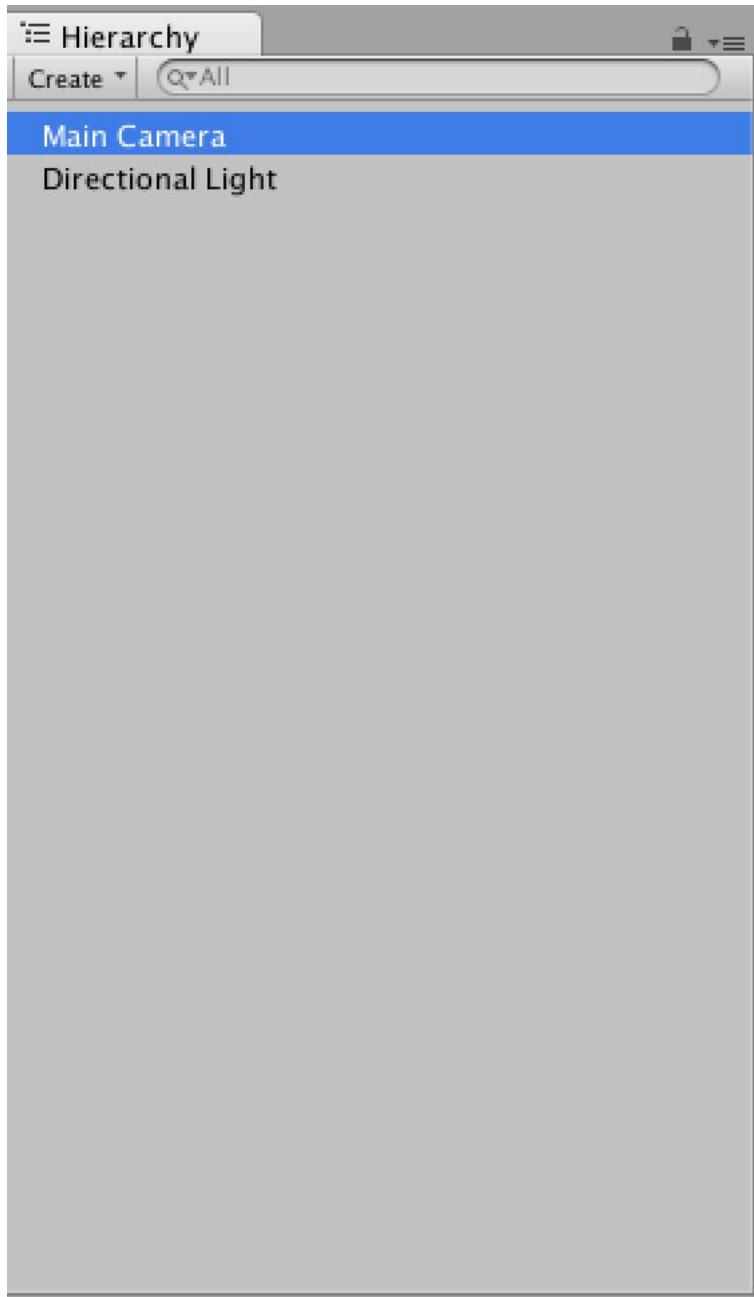


図4.5: ヒエラルキーウィンドウとシーンビューの実体は同一だ

4. インスペクタ(図6)

ゲームオブジェクトを選択した時に、そのゲームオブジェクトの属性を表示するのが「インスペクタ」だ。ゲームオブジェクトの属性はコンポーネントと呼ばれており、位置情報を示す `Transform` や、照明の情報である `Light`、カメラ情報の `Camera` などがあり、後述するクラスもコンポーネントとしてゲームオブジェクトの属性となるものだ。これらコンポーネントはそれぞれ様々な値(プロパティ)を持っており、インスペクタから値を変更することでゲームのバランス調整などを行うことができる。



図4.6: 各々のゲームオブジェクトのプロパティを調整できる

5. ツールバー(図7)

「ツールバー」は、複数のボタンで構成されている。ゲームを実行するためのPlayボタンはおそらく最もよく使うボタンだろう。ほかにも、メニューの操作ボタンやローカル・グローバル座標の切替ボタン、レイヤ制御ボタンなどがある。



図4.7: ゲームの実行以外にもポーズやフレーム送りなどもできる

6. ゲームビュー(図8)

「ゲームビュー」は、カメラコンポーネントが最終的に描画した3D空間を表示する画面だ。Unityでシーンを作成すると必ずカメラコンポーネントも作成されるが、そのカメラが映し出す空間こそ、このゲームビューである。通常のゲームではビルドしたアプリはこのゲームビューの通りに表示されるが、VRコンテンツでは自動的にアプリ側で左右に分割され、ステレオレンダリングの形式で表示される。

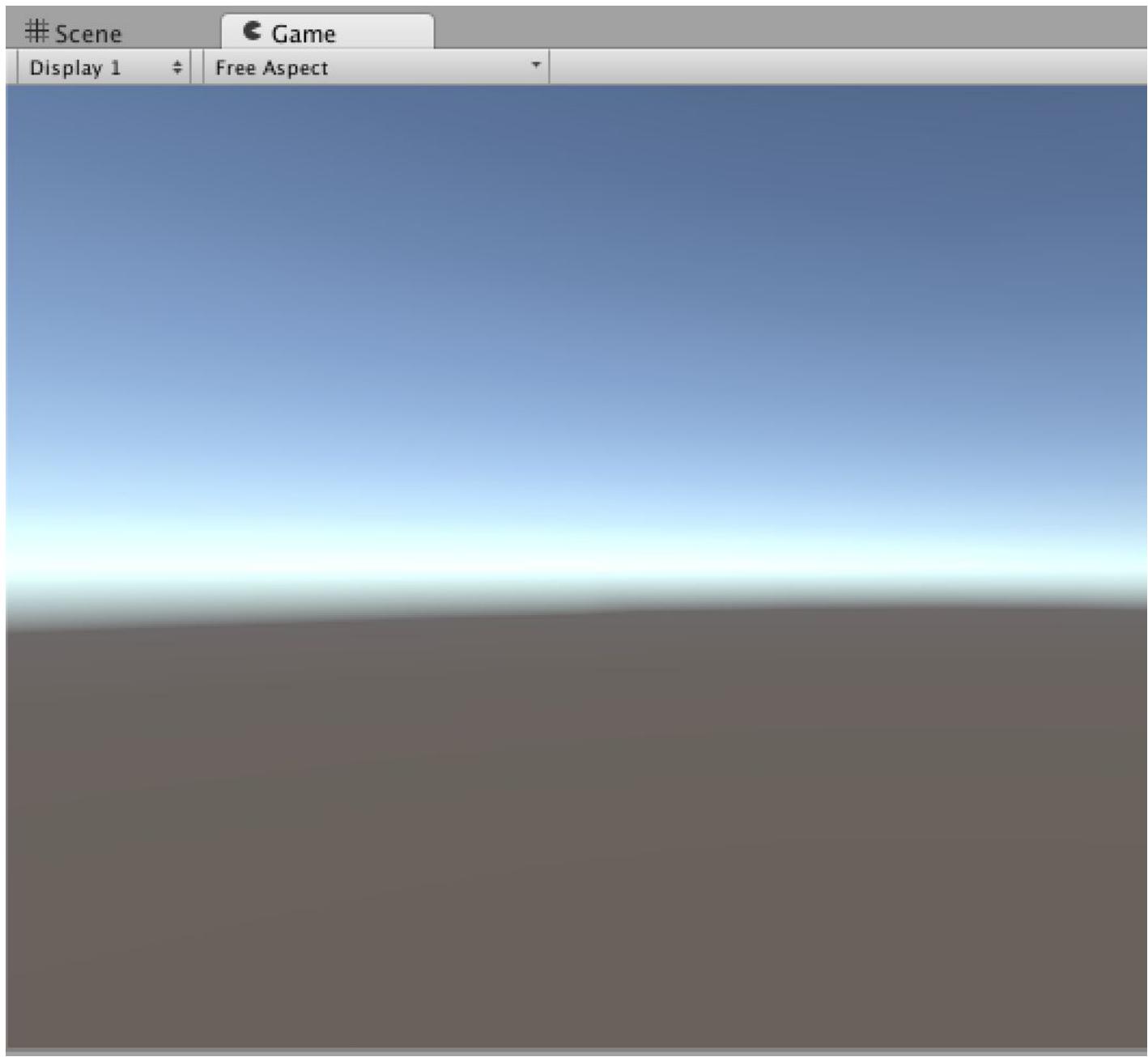


図4.8: この画面が実際のゲーム画面としてレンダリングされる

3Dの操作に慣れよう

6つのメインウィンドウのうち、シーンビューは3D空間を表示する画面であるため(図4.9)、操作に若干の慣れが必要だ。なお、一連の操作のホットキーはMayaなどのDCCツールと同一なので、それらの経験者は問題ないだろう。

表4.1:

操作	マウス	トラックパッド(Mac)
----	-----	--------------

移動	[Alt]キー+中クリックでドラッグ	[Alt]キー+Command+クリックでドラッグ
回転	[Alt]キー+クリックでドラッグ	[Alt]キー+クリックでドラッグ
ズーム	[Alt]キー+右クリックでドラッグ	[Alt]キー+Command+右クリックでドラッグ

なお、ツールバーのボタンからも同様の操作が可能だが、なるべくホットキーを覚えたほうが効率がいいだろう。

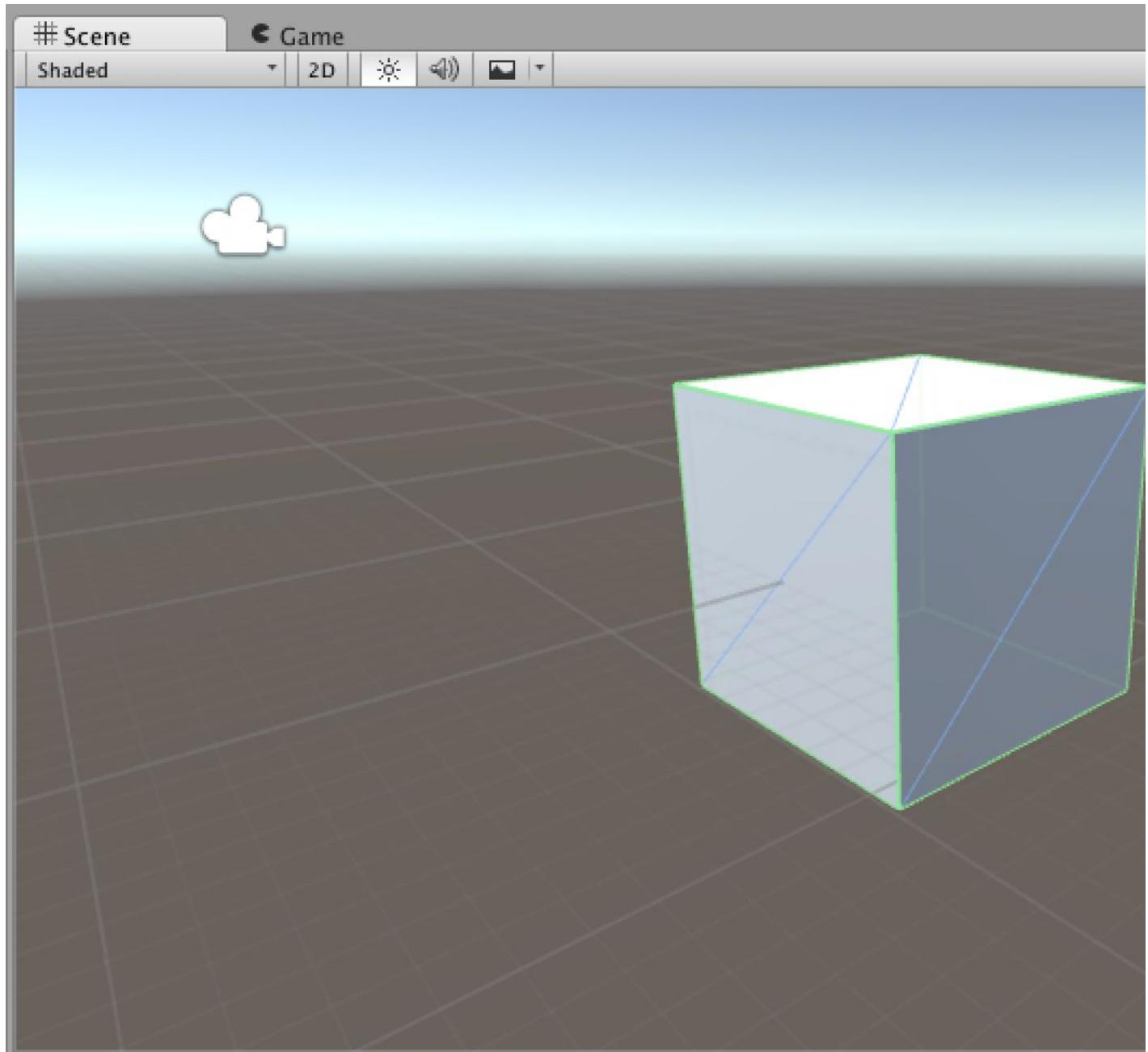


図4.9: 慣れない人はヒエラルキーインドウから[Create]→[3D Object]→[Cube]で3Dオブジェクトを配置して自由なアングルで見られるように練習しよう

4.4 Unityスクリプトを理解しよう

基本を押されたところで、VRコンテンツ制作に必要なUnityスクリプトのプログラミングを覚えていこう。試しに、プロジェクトウィンドウで右クリック→[Create]→[C# Script]を実行してみると、C#のクラスが作成されるはずだ。この時、クラス名を”UnitySample”とでもしておこう(図4.10)。なお、C#の規約上、名前の先頭に半角数字や途中に半角スペースなどを入れるとコンパイルエラーを引き起こすので注意してほしい。全角文字を入れるのはもってのほかだ。

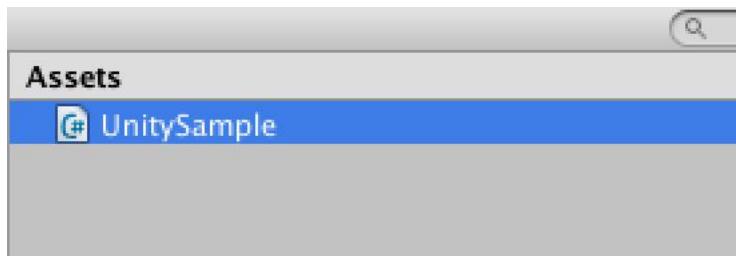


図4.10: C#のクラスを作成したところ

Hello World① クラスを編集しよう

クラスをダブルクリックすると、スクリプトを編集するためのスクリプトエディタが開く。Windowsであれば、おそらくVisual StudioもしくはMonoDevelopが、Macの場合はMonoDevelopがデフォルトのスクリプトエディタに設定されている。この時、稀にクラスのファイル名と実際のクラス名が異なっていることがあるが、この場合もコンパイルエラーになってしまうため、手動でどちらかに統一しておこう。

そして、`Start()`関数の中に`Debug.Log ("Hello, Wolrd");`と記述してみよう(図4.11)。プログラミング入門でおなじみの、いわゆる「Hello World」だ。`Debug.Log ()`クラスを使うと、Unityの下部に表示されるコンソール画面に文字列を表示できる。今後もプログラムのデバッグでよく使うことになるだろう。

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class UnitySample : MonoBehaviour {
5
6     // Use this for initialization
7     void Start () {
8         Debug.Log ("Hello, Wolrd");
9     }
10}
```

図4.11: 試しにクラスを記述してみる

しかし、このままではHello Worldは実行できない。クラスを作るだけでは、その処理がどこからも呼び出されないからだ。

Hello World② クラスをコンポーネント化しよう

さて、いったんUnityエディタへ戻り、プロジェクトウィンドウ上のクラスを選択して、そのままシーンビュー上の適当なゲームオブジェクトにドラッグ & ドロップしてみよう。その後、ゲームオブジェクトを選択してインスペクタを見ると、クラスがコンポーネントとして登録されていることを確認できる(図4.12)。このようにクラスをコンポーネント化することをクラスのアサインと呼ぶ。

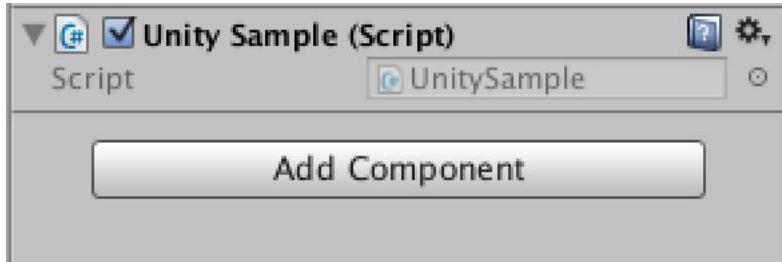


図4.12: クラスがコンポーネントとしてアサインされた状態

この状態でツールバーのPlayボタンを押下して、ゲームを実行してみよう。コンソール画面に「Hello World」が出力されたはずだ(図4.13)。

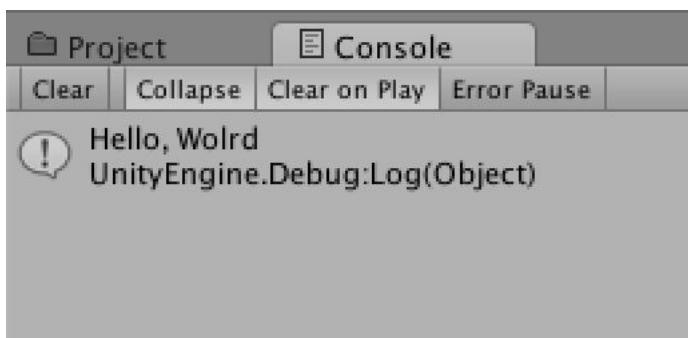


図4.13: コンソール画面に「HELLO, WORLD」を表示できた

このように、クラスはコンポーネント化することで処理として呼び出されるようになるのだ。

MonoBehaviourクラスについて

先ほど、`Debug.Log ("Hello, Wolrd");`を`Start()`関数に記述したが、この`Start()`とは何か気になった人もいるだろう。他にも、クラスを作成した当初から`Update()`関数が定義されていることに気づく。

結論を解説する前に、今一度スクリプトエディタのほうへ目を向けてみよう。このクラスは`MonoBehaviour`という基本クラスを継承している。実は、この`MonoBehaviour`はUnityのすべてのコンポーネントに継承されている。つまり、すべてのゲームオブジェクトは`MonoBehaviour`の派生クラスなのだ。

ゲームオブジェクトとしてシーンに存在する`MonoBehaviour`クラスには、ゲームの実行(ランタイム)時に様々なタイミングで特定の関数がコールされる。その関数が、例えば`Start()`や`Update()`だったりするのだ。この関数を使い分けてゲームロジックを作っていくことがUnityスクリプトの基本となる。以下、代表的な関数を紹介しよう。

`Start()`

ゲームの開始時に1回だけ呼ばれる。初期化などの処理に最適。

`Update()`

`Start()`の後、毎フレーム呼ばれる。つまり、画面が60FPSで描画されていれば、1秒に60回呼ばれることになる。ゲームオブジェクトの移動など画面描画と連動した処理に使用する。

`Awake()`

ゲームの開始時に1回だけ呼ばれる。`Start()`よりも前に呼ばれることが保証されており、例えば初期化を二段階で行いたい時に使用する。

この他にも、さまざまな関数が用意されている。詳しくは、公式の[APIリファレンス](#)を参照してほしい。

4.5 おわりに

今まで、環境構築、サンプルのビルド、そしてUnityの基礎を解説した。これで必要な準備はすべて整ったので、次章からは本格的にVRコンテンツ開発を始めていこう！

第5章 モバイルVRゲームを作ってみよう

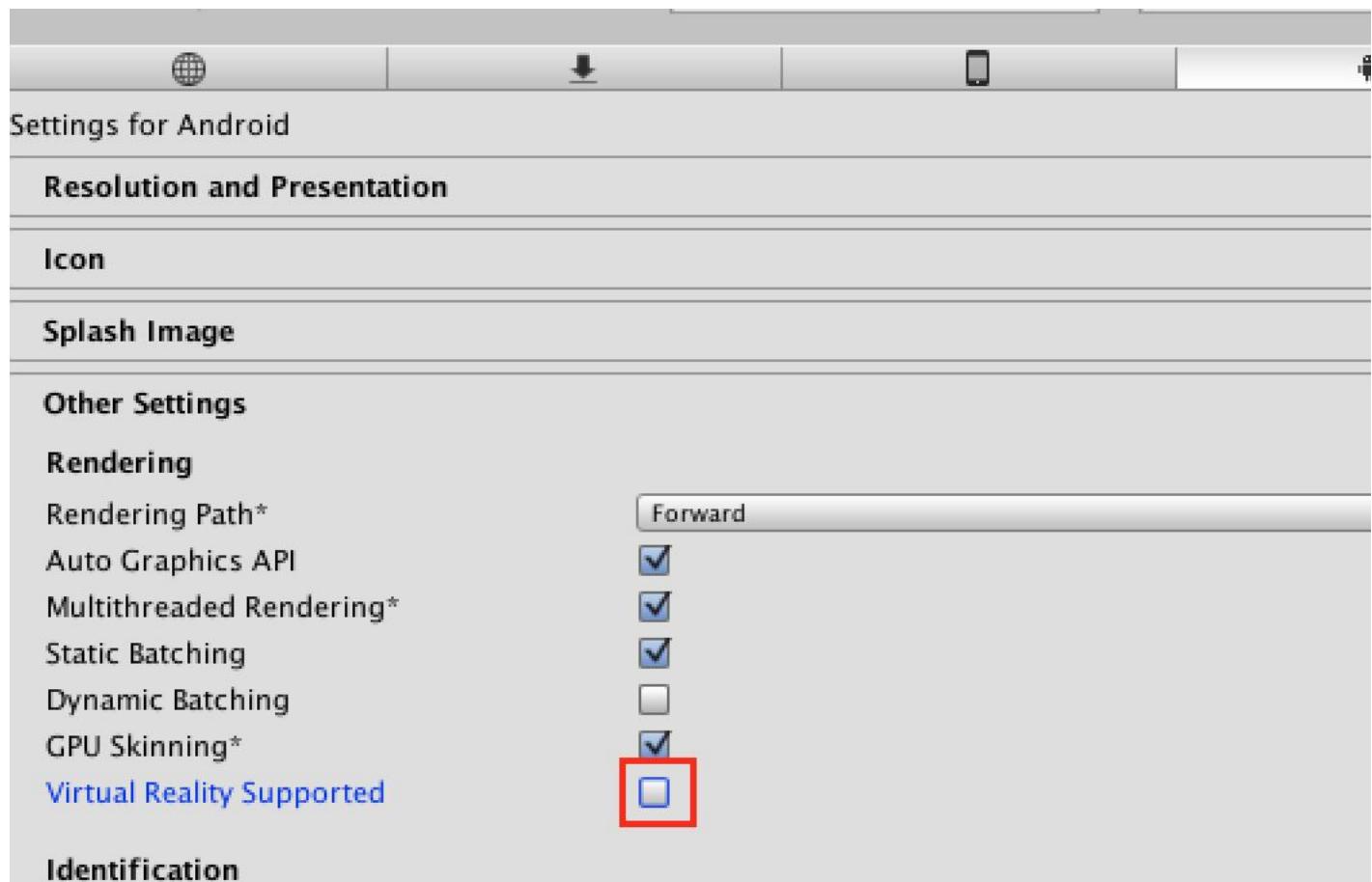
前章までを通して、モバイルVRゲーム制作に必要な環境の準備と、ゲームエンジン「Unity」の基礎をマスターできた。本章以降は、これらの知識を駆使して自力でVRゲームの完成を目指していく。

5.1 サンプルプロジェクトをビルドしてみよう(ハコスコ/Google Cardboard)

モバイルVRゲームの作成に入る前に、本書の第3章「サンプルプロジェクトをビルドしてみよう」では紹介しなかったハコスコとGoogle Cardboard向けにサンプルプロジェクトをビルドする方法を解説する。あらかじめ「Unity VR Samples」をUnityプロジェクトにインポートしておいてほしい(詳細な手順は第3章を参照)。

なお、各デバイスの詳細は第1章「モバイルVR開発をはじめよう」、開発環境の詳細は第2章「モバイルVRの開発環境を準備しよう」をそれぞれ参照してほしい。

1. Unityのメニューから[Edit]→[Project Settings]→[Player Settings]を選択する
2. Other Settingsの「Virtual Reality Supported」にチェックを入れる
3. Virtual Reality SDKsに「Cardboard」を入れる



実機にビルドする

プレハブの配置が終わったら、AndroidもしくはiOSにプラットフォームを変更し、Unityからビルドを行う。Google Cardboard向けにビルドした場合は、画面下部の歯車ボタンを押下し(図5.1)、デバイスに印刷されているQRコードをスキャンすることで、そのデバイスに最適な表示に切り替えてくれる(図5.2)。

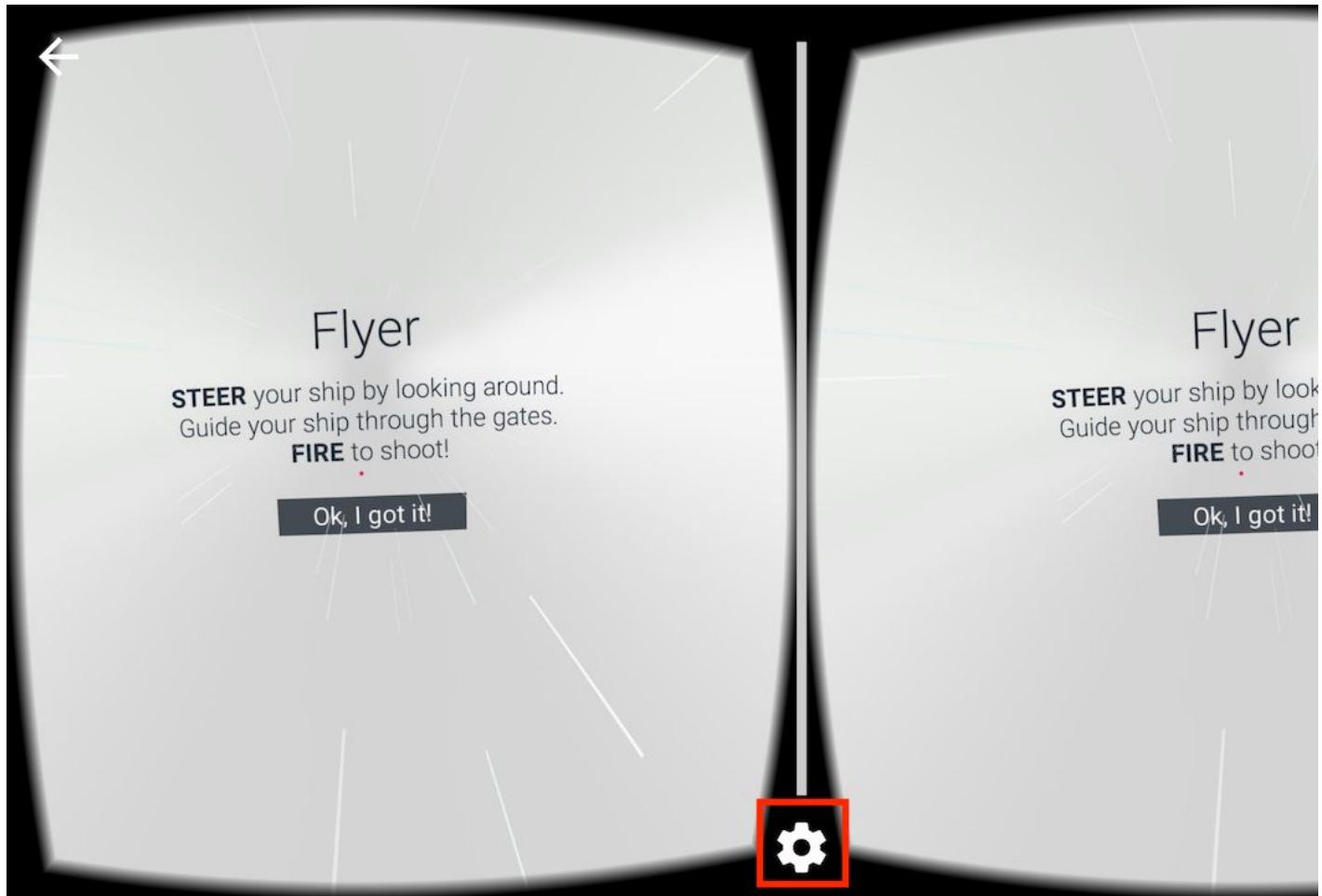


図5.1: 齒車ボタンを押下するとQRコードをスキャンする画面に遷移する



スマートフォンに次のビューアを設定しました:

cardboard v2.0

製造元: Cardboard

ビューアを切り替え

図5.2: 適切なビューワを設定したところ

以上で、ハコスコ/Google Cardboard向けのサンプルプロジェクトのビルトは完了だ。

[column]「Google Daydream」について

本書では解説しないが、Virtual Reality SDKsに「Daydream」を入れることで[Google Daydream](#)にも対応する。

5.2 モバイルVRゲームを作ってみよう～設計と素材集め

さて、少し前置きが長くなったが、以降ではモバイルVRゲームの作成に入っていく。まずは、改めて作成するゲームについて考えてみよう。

どんなゲームを作るか

筆者は今回、「VRシューティングゲーム」を作成しようと考えている。なぜシューティングかというと、ゲームにHMDからの入力や操作をシンプルに反映できる上、短いサイクルでゲームを成立させることができるために、ゼロからゲームを作るにはうってつけのジャンルだからだ（図5.3）。ここで学んだことを活かせば、謎解き脱出ゲームやパズルゲームなど、他のジャンルのゲーム制作にも手が届くようになるはずだ。

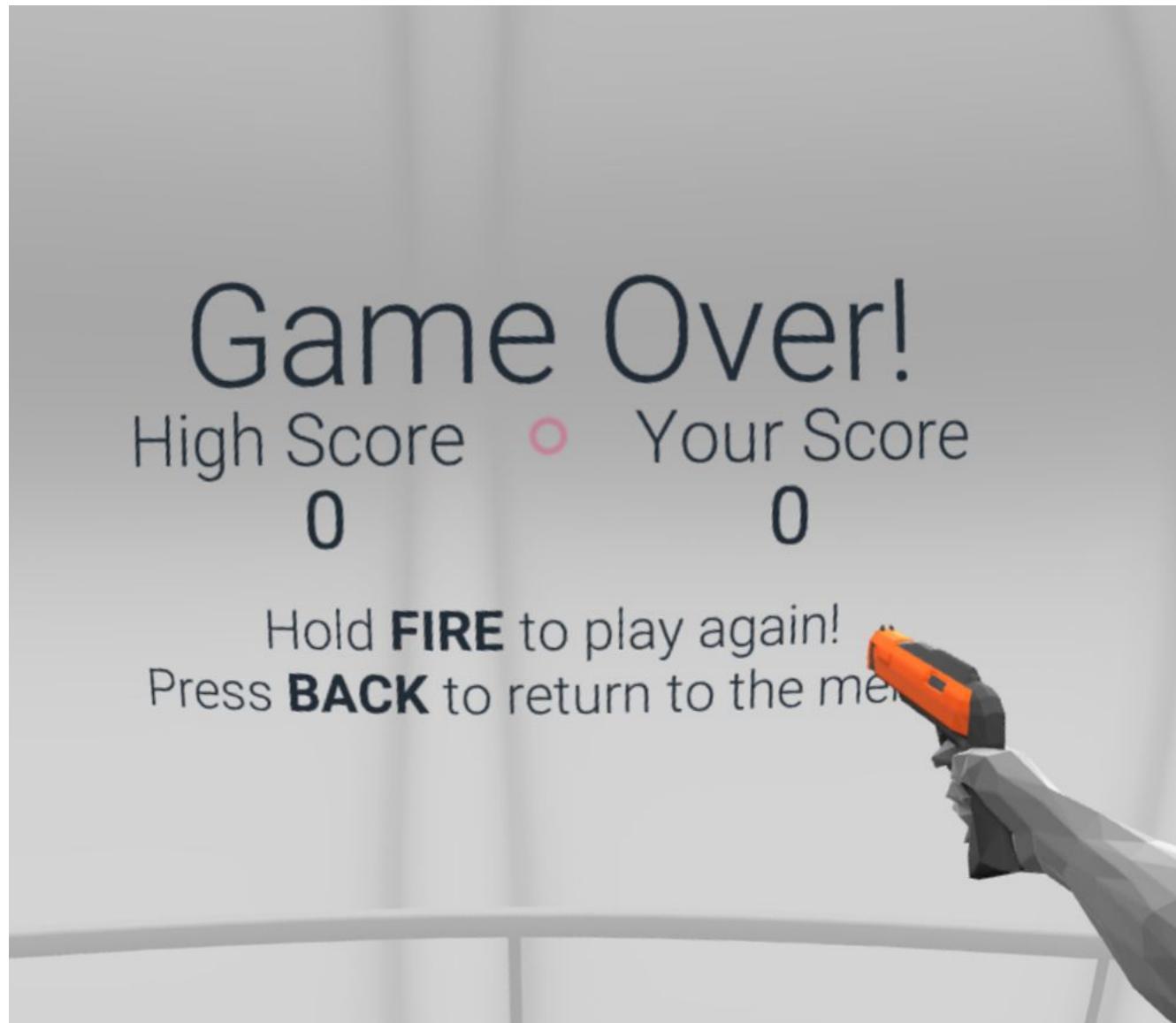


図5.3: シューティングゲームはゲーム作り入門に最適なジャンルだ

シューティングゲームの設計

通常、ゲームプログラミングの前段階において必ずコンテンツの「設計」を行う。「どのように作っていくか」を考える作業のことだ。ゲームプログラミングに入る前に、ある程度この設計ができていなければ、最後まで作品を完成させることも覚束ない。

設計には様々な方法があるが、今回はシューティングゲームの構成要素を分解し、どのような機能の実装が必要かを洗い出すことで、ゲームの設計作業としたい。

それでは、ざっくりとシューティングゲームに必要な機能を考えてみよう。ゲームとして成立させるには、大まかに次の3つの機能を実装すれば良いだろう。

プレイヤー

まず最初に必要な機能は「プレイヤー」だ。プレイヤーには2つの役割がある。1つは「現実世界のプレイヤーからの入力をゲームに反映させる」ことだ。HMDを被ったプレイヤーの頭の傾きや回転情報がVR空間に入力されることで、VRカメラもその通りに動く。またシューティングゲームにおいては、ボタンやコントローラからの入力によって弾が発射される。このようなHMDやコントローラからの入力を受け付けるのがプレイヤーの機能の1つと考えても良いだろう。

2つ目の役割は「プレイヤーのパラメータを保持する」ことだ。プレイヤーは体力(HP)や発射する弾の攻撃力といったパラメータを持つことが想定される。他にも、例えば一定期間無敵になるアイテムがあるなら、通常状態と無敵状態を分ける「モード」パラメータが必要だろう。これらのパラメータを保持し、必要に応じて増減、もしくは変更するのもプレイヤーの重要な役割だ。

エネミー

「エネミー」はプレイヤーの敵となる機能だ。プレイヤーの攻撃を回避しながら移動しつつ、攻撃してくる。また、逆にプレイヤーがエネミーを攻撃して爆発エフェクト(もしくは血しぶきでも良いかもしれない)を出しつつ消滅する、という機能がエネミーの主な役割だ。この機能は、敵の「行動」と言い換えてもいい。またプレイヤーと共に通して、エネミー自身にもHPや攻撃力といったパラメータを保持する役割がある。

プレイヤー機能と大きく異なるのは、ゲーム中にはエネミーが複数存在することだ。敵が一体しか出現しないのではつまらないで、当然複数のエネミーが必要になってくる。しかも、敵のグラフィックや強さが違ったり、攻撃パターンが異なったりするなど、バリエーションに富んでいたほうがゲームも面白くなる。つまり、敵のバリエーションによって複数のエネミー機能を実装する必要があることを覚えておこう。

ゲームサイクル

最後に「ゲームサイクル」だ。これは、いわばそのゲームシステムの「進行役」である。例えば、プレイヤーがエネミーからの攻撃を受け続けたHPが0になった場合はゲームオーバーとなるだろう。このとき、HPが0になったことを条件にゲームをゲームオーバー状態へ移行させる役割がゲームサイクルだ。もちろん、「一定のエネミーを倒したらゲームクリア」という状態にする機能などもゲームサイクルの一部に含まれる。

シューティングゲームに必要な素材

大まかなゲームの設計ができたら、それを基に必要なアセット(素材)をピックアップしてみよう。ゲームはプログラムだけで成立しているのではなく、グラフィックとして表示させるための素材やサウンドなども必要だ。ここで、ゲーム制作に使用される代表的なアセットを簡単に解説しよう。

3Dグラフィック

VR空間は3DのCGで描画されるため、3Dデータ素材が必要だ。主に使用されるものとして「モデルデータ」と「アニメーションデータ」の2種類があり、さらにそれらに付随する「テクスチャ」「マテリアル」「シェーダ」などのデータも必要になってくる。なお、これらのファイルはたいていひとまとめになっていることが多い。

モデルとアニメーション

「モデル」は頂点のかたまりから構成されるメッシュのデータで、VR空間の描画にはすべてモデルデータが使用される。一方、「アニメーション」は時間軸で変化する回転や移動、拡縮といった情報を持つ。

例えば、背景のモデルは動く必要がないためモデルデータのみでいいが、動作するキャラクター等はモデルデータとアニメーションデータを組み合わせる必要がある(図5.4)。



図5.4: VR Samplesにあるモデルデータを表示しているところ

これらのデータは「Maya」や「Cinema4D」、「Blender(無料)」といったDCCツールで制作する。Unityでは、これらのツールから出力できるファイルのうち.fbxや.dae形式のデータを使用するのが一般的だ。

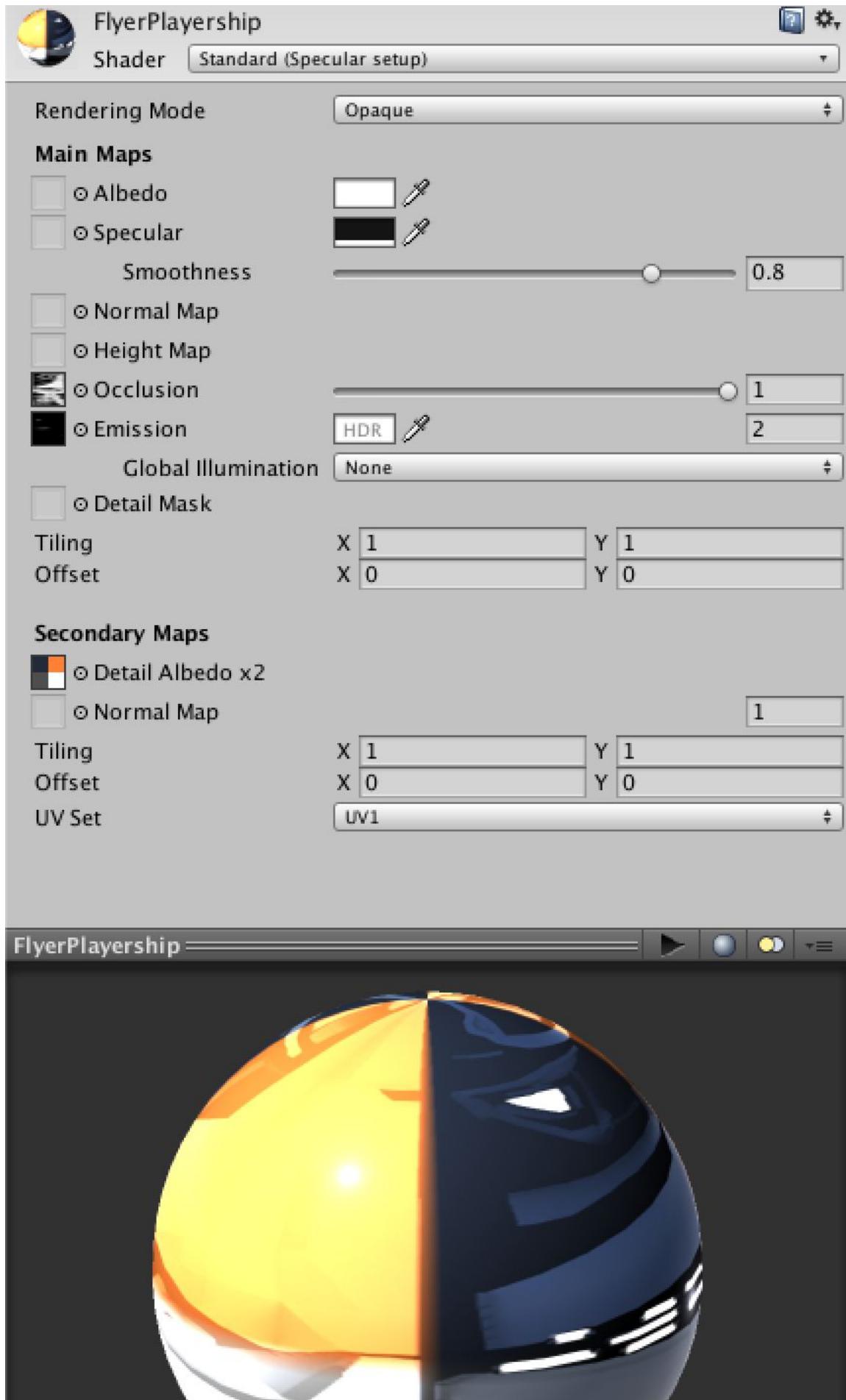
テクスチャ・マテリアル・シェーダ

2D画像の「テクスチャ」をモデルに貼り付けると、その画像がモデルの色味になる。テクスチャデータは単一ではなく、色味を決定するAlbedoテクスチャや細かい凹凸を表現するNormalテクスチャ、光沢を制御するSpecularテクスチャなど様々な種類がある。Unityで使う場合は.tga形式や.png形式で用意するのが一般的だ。

「マテリアル」は複数のテクスチャをまとめてモデルデータに適応するためのデータだ。Unityの場合、モデルを読み込むと.mat形式の独自ファイルを自動で生成してくれる。

「シェーダ」は、ここでは「マテリアルの質感を定義するプログラムデータ」と定義しておこう。Unityでは.shaderという独自形式で取り扱うことになる。一步踏み込んだ3D表現を行うにはシェーダにも手を入れる必要がある(図5.5)。





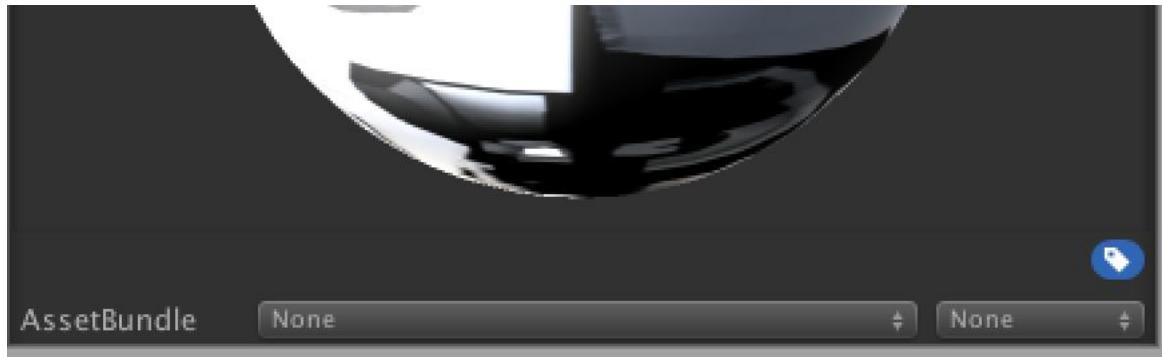


図5.5: マテリアルは1つのシェーダを基に複数のテクスチャから構成される

UI

「UI」とは、User Interfaceとして表示する2D画像やフォントのことだ。UIはプレイヤーにアイコンで遊び方を説明したり、ダメージを数値で表現したりするなど、ゲーム上で重要な要素の1つだ。もちろんVR空間にも3Dモデルとは別にUIを表示できるが、その場合は「VR空間にペラペラの画像が浮かんでいる」というのをイメージするといいだろう。

Unityでは「uGUI」という機能を使ってUIを描画する(図5.6)。**.tga**形式や**.png**形式の画像を読み込んで表示できるほか、**.ttf**形式のTrueTypeフォントやOpenTypeフォント.**.otf**を使えば自由に文字列を表示することもできる。UnityにはデフォルトでテクスチャやArialフォント等が含まれている。

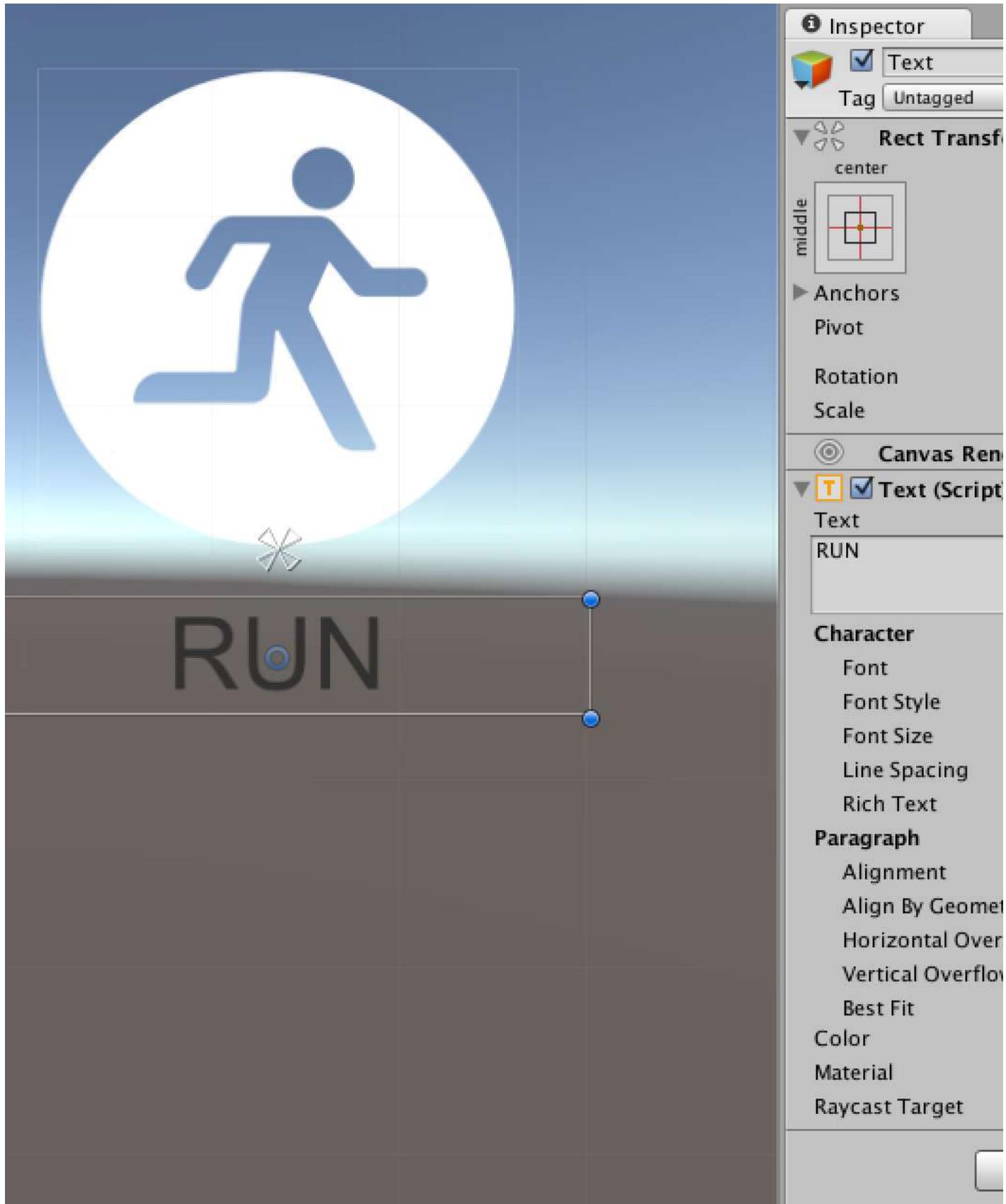


図5.6: uGUIを使って画像(スプライト)とフォントを表示しているところ

エフェクト

「エフェクト」は炎や液体、爆発などを表現するための3Dデータの1つだ(図5.7)。これらはパーティクルという技術を用いて描画するが、UnityにもShurikenと呼ばれるパーティクルを描画する仕組みが備えられている。

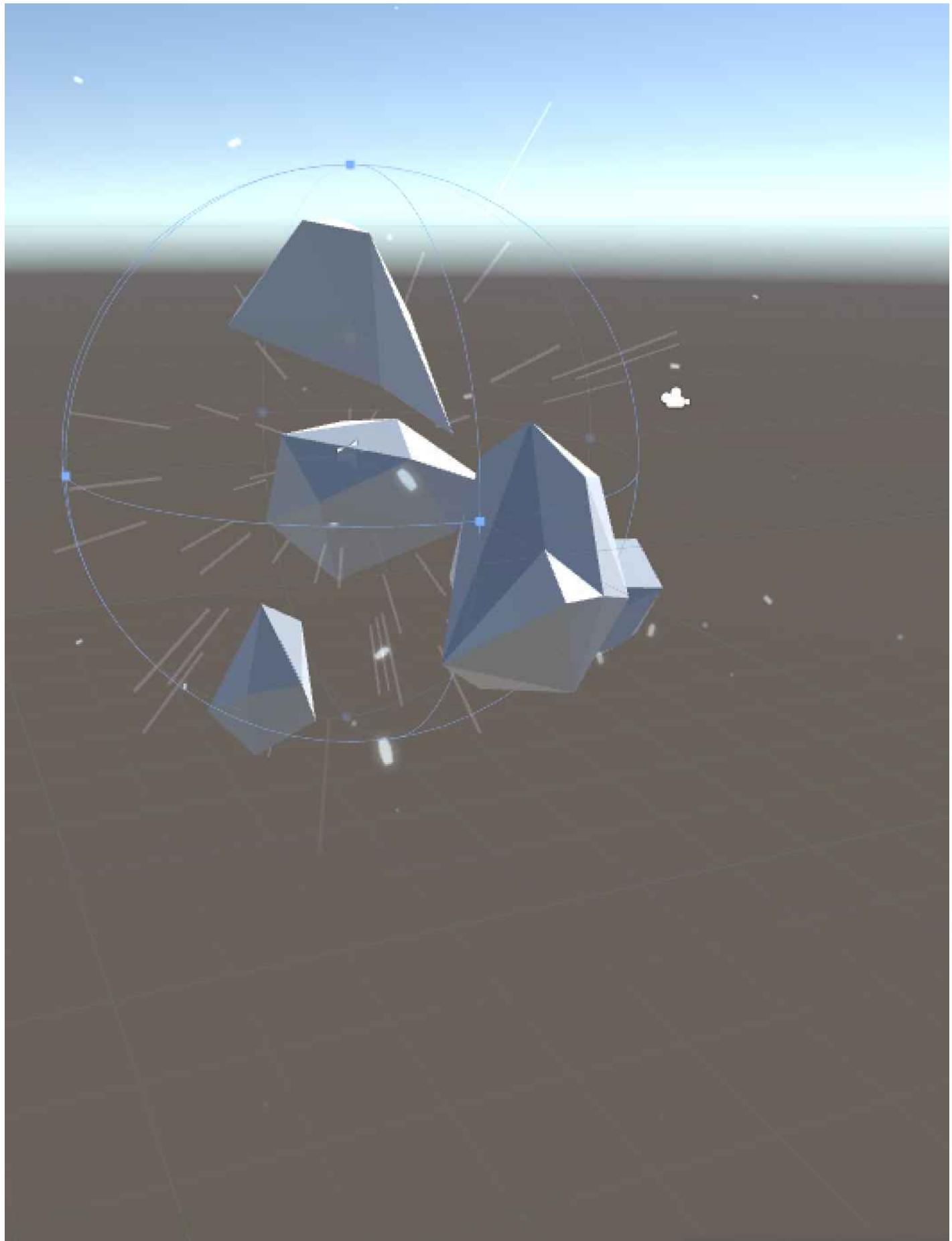


図5.7: VR Samples のエフェクトを再生しているところ

エフェクトは、主にShuriken(コンポーネント名「ParticleSystem」)のデータと、Particleにアサインされるテクスチャ・マテリアルデータから構成されている。

サウンド

最後は「サウンド」だ。ゲームの雰囲気を盛り上げるうえで効果音(SE)やBGMは欠かせない。Unityでは.mp3やwav、ogg形式の音声ファイルを読み込むことができる。また、インスペクター上で音源の設定を変えれば、3Dサウンドで音声ファイルを再生することも可能だ(図5.8)。

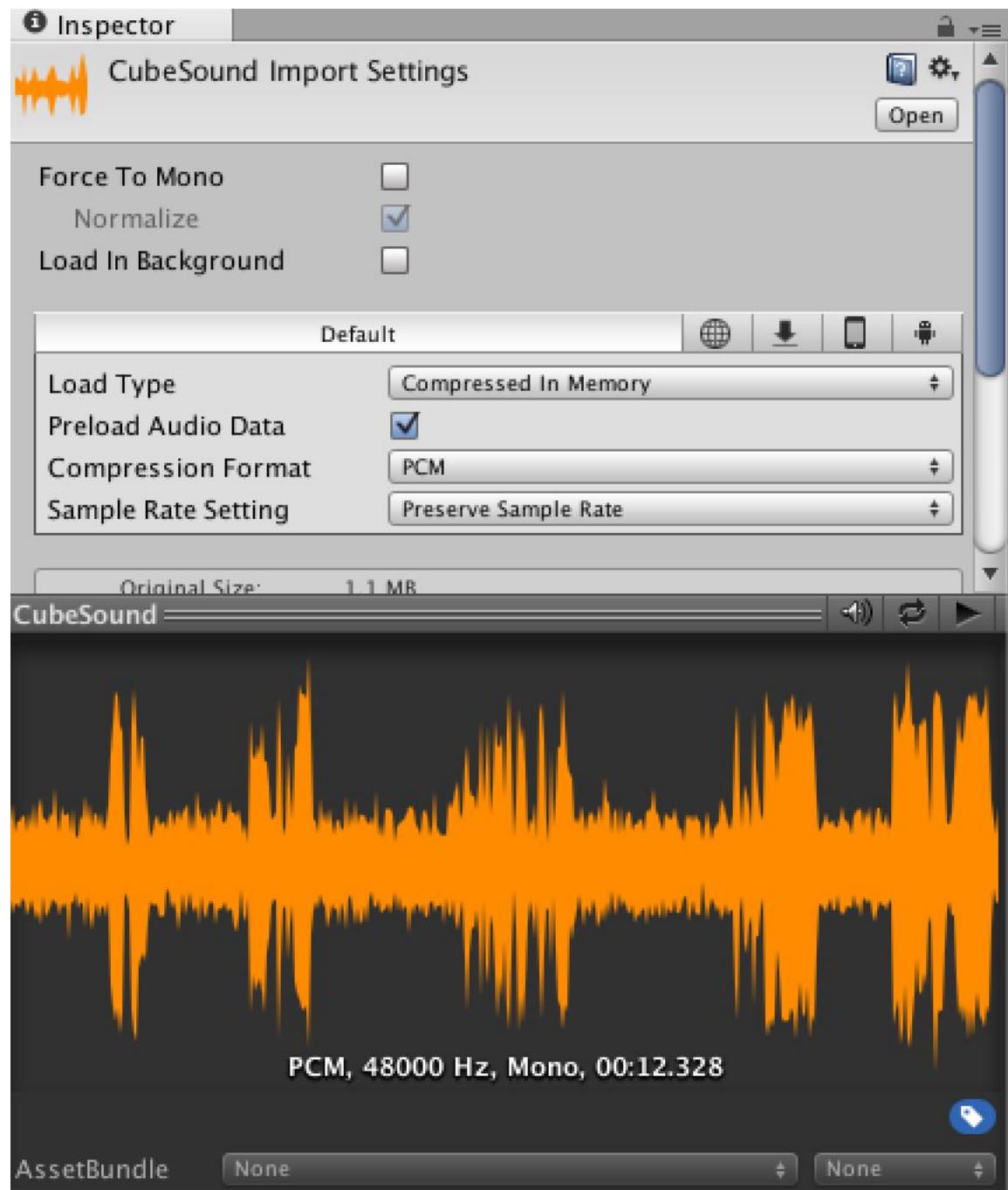


図5.8: サウンドファイルもインスペクタから再生できる

5.3 おわりに

今回は、本格的なモバイルVRゲームを開発するに当たって、シューティングゲームの設計と必要な素材について解説した。次章からは、今回行ったシューティングゲームに必要な基本的な設計を元に機能を実装していく方法を解説する。

第6章 VRシューティングゲームを実装しよう

前章では、ゲームの「設計」を行うことで、これから実装するシューティングゲームの大まかな機能を定義した。今回はこれらをもとに、Unity上で具体的な実装を行っていこう。第2章の「モバイルVRの開発環境を構築しよう」、第3章「サンプルプロジェクトをビルドしてみよう」(Gear VR・ハコスコ/Google Cardboard)も参照してほしい。

6.1 プレイヤーからの入力を実装する

プレイヤー機能の定義の1つに「HMDの動きやタッチパッド・コントローラからの入力を受け付けてVR空間に反映させる」というものがあった。これを実現させるための手段を見ていこう。

HMDからの入力を受け付ける

「HMDからの入力」は、つまりところプレイヤーの頭部の動きのことだ。これをVR空間のカメラに回転情報として流し込めば、HMDの動きがVR空間でも同期されることになる。この時点できつい読者もいるかもしれないが、過去にも触れたとおり本機能はUnity本体、もしくはSDK自体がサポートしているものなので、特にプログラムを組む作業は不要だ。念のため、プラットフォームごとに簡単に振り返ってみよう。

Gear VRの場合はMain Cameraを置くだけ(ビルド時にはPlayer Settings > Other Settings > Virtual Reality Supportedを忘れずに)、ハコスコ/Google Cardboardの場合はSDKからGvrViewerMain.prefabを配置するだけで完了だ。

いずれのプラットフォームでもSceneにはMain Cameraが配置されている状態だと思うが、ここでは「プレイヤー」としての機能をMain Cameraゲームオブジェクトに集約していくことにする。

タッチパッド・コントローラからの入力を受け付ける

Gear VRではHMDの右側にタッチパッドが、Google Cardboardにも簡易的なタッチパッドが搭載されている。また、タッチパッドが搭載されていないハコスコでも、Bluetoothコントローラを接続すればボタン入力をゲームに反映できる。これらの入力によって何らかの処理を行うプログラムを書いてみよう。

Project Viewで右クリック > Create > C# Script を選択して新規クラスを作成後、名前を「Player」に変更する。ダブルクリックして編集できる状態にしたら、スクリプトのクラス名がファイル名と同一であることを確認しておこう。そうしないと、エラーとなりスクリプトが動作しない。

確認できたら、まず入力を受け付ける処理を書くためにMonoBehaviour関数を定義するが、ズバリ、ここは FixedUpdate関数を使う。FixedUpdateはUpdate関数と同じく毎フレーム呼び出される関数だが、処理速度によってフレームレートが変化しても影響を受けない。つまり、処理速度が低下して動作が遅延した場合でも必ず一定のタイミングで呼び出されるため、このような入力を受け付ける処理は必ず FixedUpdate関数に記述するのが鉄則だ。

```
public class Player : MonoBehaviour {
    void FixedUpdate () {
    }
}
```

FixedUpdate内では、if文で囲ったInput.GetButtonDown ("<ボタン名>") を記述する。Input.GetButtonDown()は指定したボタンが押された場合にTrueを返す、つまり入力があったときif文内の処理が実行される。ボタン名は"Fire1"と指定しておく。

```
if (Input.GetButtonDown ("Fire1")) {
    // 処理
}
```

"Fire1"はUnity内部で定義されている値で、画面タップやコントローラの1ボタン(どの位置のボタンかはハードウェアによって異なる)、マウスの左クリックからの入力を識別できる共通の値である。つまり、このように記述するだけで画面タップとコントローラだけでなく、マウスにも対応した処理を一度に書けてしまうのだ。

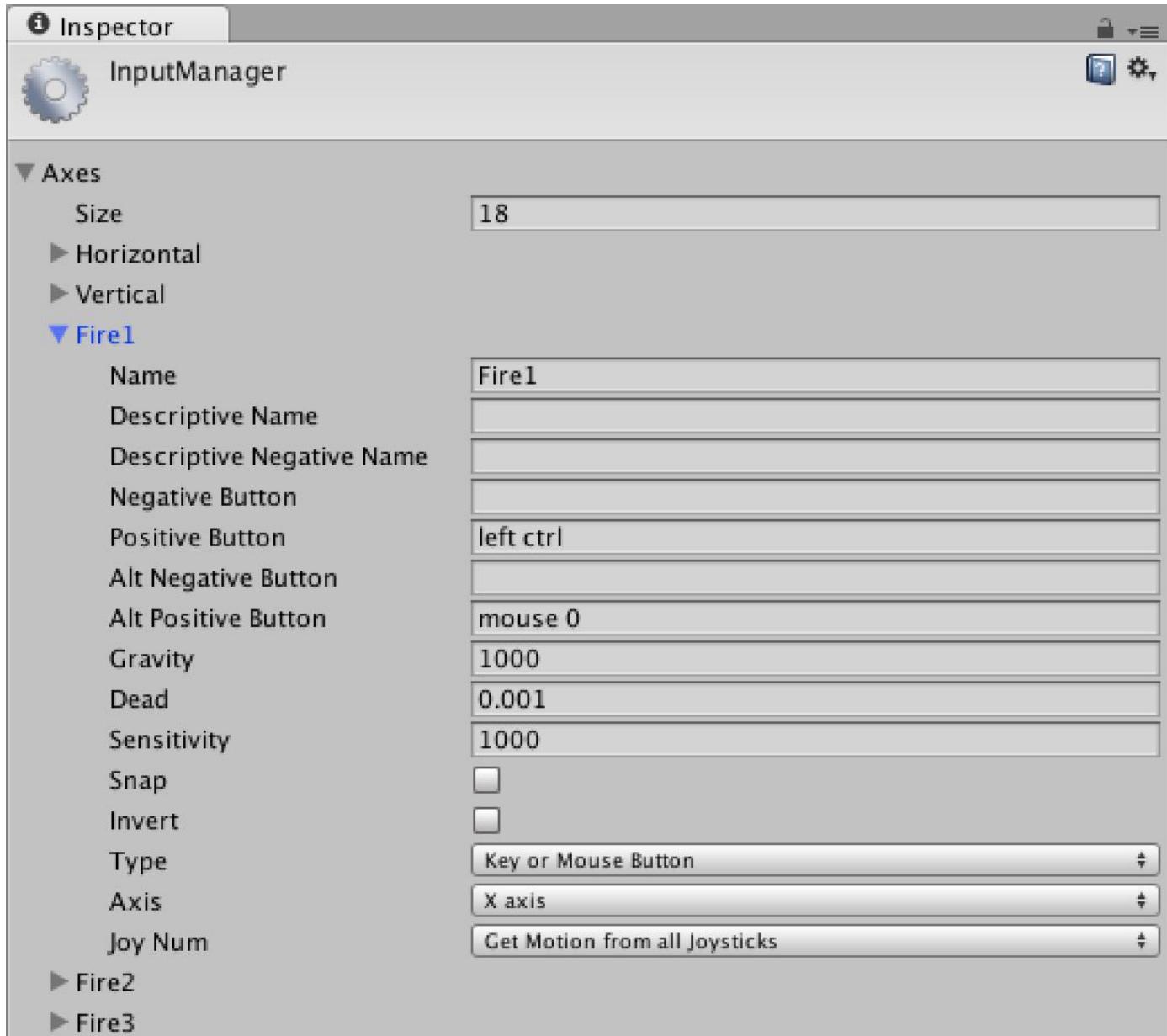


図6.1: なおEdit > Project Settings > Inputで他の入力の定義の値を確認できる

そして、Player.csクラスをプレイヤーゲームオブジェクトであるMain Cameraにアサインしておこう。

弾丸の発射

今度は、画面タップもしくはコントローラのボタンを押すとプレイヤーから弾丸が発射される仕組みを実装していく。まずは、弾丸となるゲームオブジェクトを作成しよう。Hierarchy ViewでCreate > 3D Object > Sphereを作成し、名前をBulletとする。BulletにはInspectorからAddComponentボタンをクリックしてRigidbodyコンポーネントをアサインしておく。その後Project Viewにドラッグ・アンド・ドロップしてプレハブ化したら、Hierarchy View上からは削除しておこう。

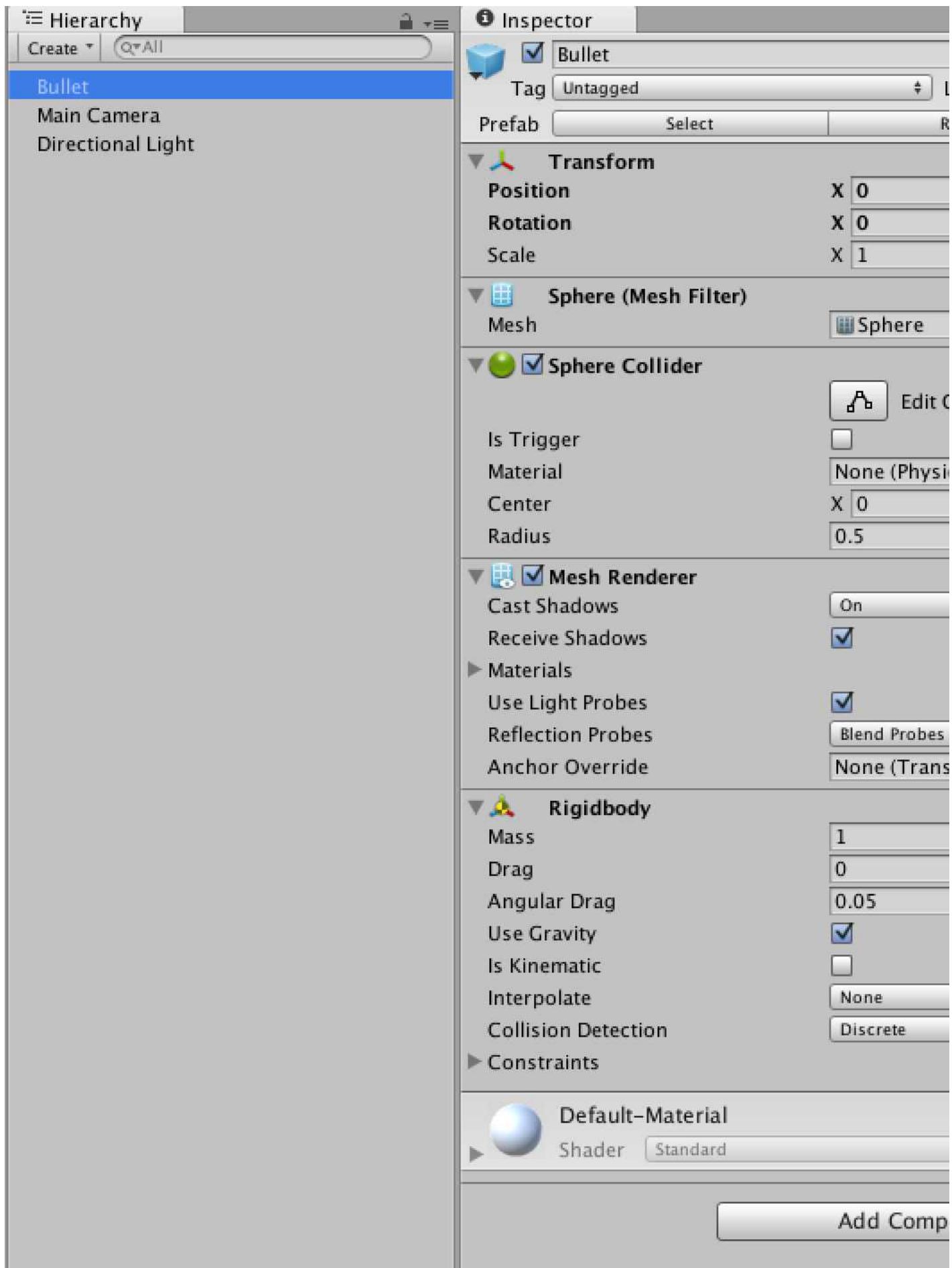


図6.2: Bulletをプレハブ化したらScene上には不要だ

次に、先程作ったBulletを発射するための「場所」をMain Cameraに定義しよう。Hierarchy ViewでCreate > Create Emptyし、名前をShootPositionとする。ShootPositionをMain Cameraにドラッグ・アンド・ドロップしてMain Cameraの子ゲームオブジェクトにしたら、ShootPositionのPositionをMain Cameraの少し前方に置かれるように調整する。



図6.3: ShootPositionからBulletが発射されるようにする

ShootPositionからBulletが発射されるようにする。

終わったらPlayerクラスの編集に戻り、メンバ変数に次の2つを定義しよう。

```
public GameObject bullet;  
public GameObject shootPosition;
```

Hierarchy Viewに戻り、プレイヤーゲームオブジェクト(Main Camera)のInspectorに注目すると、上記で定義した2つのメンバ変数を格納するフィールドが現れるので、bulletにはProject ViewからプレハブBulletを、shootPositionへはHierarchy ViewからShootPositionをアサインする。これで、プレイヤークラスからそれぞれのゲームオブジェクトにアクセスできるようになった。

Inspector

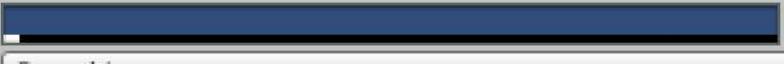
Main Camera Static

Tag MainCamera Layer Default

Transform

Position X 0 Y 1 Z -10
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Camera

Skybox: Skybox
Background: 
Culling Mask: Everything
Projection: Perspective
Field of View: 60
Clipping Planes
Near: 0.3 Far: 1000
Viewport Rect X 0 Y 0 W 1 H 1
Depth: -1
Rendering Path: Use Player Settings
Target Texture: None (Render Texture)
Occlusion Culling:
HDR:

GUI Layer
Flare Layer
Audio Listener

Player (Script)

Script: Player
Bullet: Bullet
Shoot Position: ShootingPosition
Shoot Speed: 1000
Player HP: 3

Sphere Collider

Is Trigger:
Edit Collider
Material: None (Physic Material)
Center X 0 Y 0 Z 0
Radius: 0.5

Add Component

図6.4: このような見た目になればOKだ

最後に、弾丸が発射される仕組みを書いていこう。`Input.GetButtonDown()` の結果を判定するif文の中に、次のように記述する。

```
if (Input.GetButtonDown ("Fire1")) {  
    // Bullet のゲームオブジェクトを生成する  
    GameObject bulletInstance = Instantiate<GameObject>(bullet);  
    // 生成した Bullet の位置を shootPosition に合わせる  
    bulletInstance.transform.position = shootPosition.transform.position;  
    bulletInstance.GetComponent<Rigidbody> ().AddForce >(shootPosition.transform.forward * 1000f);  
}
```

基本的にはスクリプト上のコメントの通りだが、ポイント毎に解説すると次のようになる。

- `Instantiate<型>()` で指定したオブジェクトをScene上に生成し、さらに指定した型で変数に格納できる
- `GetComponent<型>()` で指定した型のコンポーネントを取得できる(ここでは行っていないが、そのまま変数にも代入できる)
- `Rigidbody`コンポーネントの`AddForce(方向)`を使うと、物体を指定した方向に加速できる。`transform.forward`は自身の`transform`の前方を意味する。乗算している`1000f`は物体の速度パラメータ

最終的なコードは、次になる。変更点は`AddForce()`を行っている箇所で物体の速度をメンバ変数で定義し、後から値を変更できるようにした。また、SDKを利用するハコスコ/Google Cardboardの場合は`Start()`関数内の処理を加える必要がある。これは`ShootPosition`の回転を正しく反映させるためだ。

```
public class Player : MonoBehaviour {  
  
    public GameObject bullet;  
    public GameObject shootPosition;  
    public float shootSpeed = 1000f;  
  
    /// <summary>  
    /// ハコスコ/Google Cardboard のみ必要な処理(Gear VR の場合は不要)  
    /// </summary>  
    void Start() {  
        shootPosition.transform.parent = transform.FindChild ("Main Camera Left");  
    }  
  
    void FixedUpdate () {  
        if (Input.GetButtonDown ("Fire1")) {  
            // Bullet のゲームオブジェクトを生成する  
            GameObject bulletInstance = Instantiate<GameObject>(bullet);  
            // 生成した Bullet の位置を shootPosition に合わせる  
            bulletInstance.transform.position = shootPosition.transform.position;  
            bulletInstance.GetComponent<Rigidbody> ().AddForce >(shootPosition.transform.forward *  
shootSpeed);  
        }  
    }  
}
```

プレイヤー機能の仕上げ

プレイヤー機能の仕上げとして、最後にゲームの進行上必要なパラメータをPlayerクラスに設定しよう。ここは、HPパラメータをメンバ変数の形で設定しておく。

```
public int playerHP = 3;
```

また、Main CameraオブジェクトにAddComponentボタンからSphere Colliderコンポーネントをアサインしておこう。このColliderは衝突判定を行う際に必要で、後述するエネミー機能の実装で使用する。

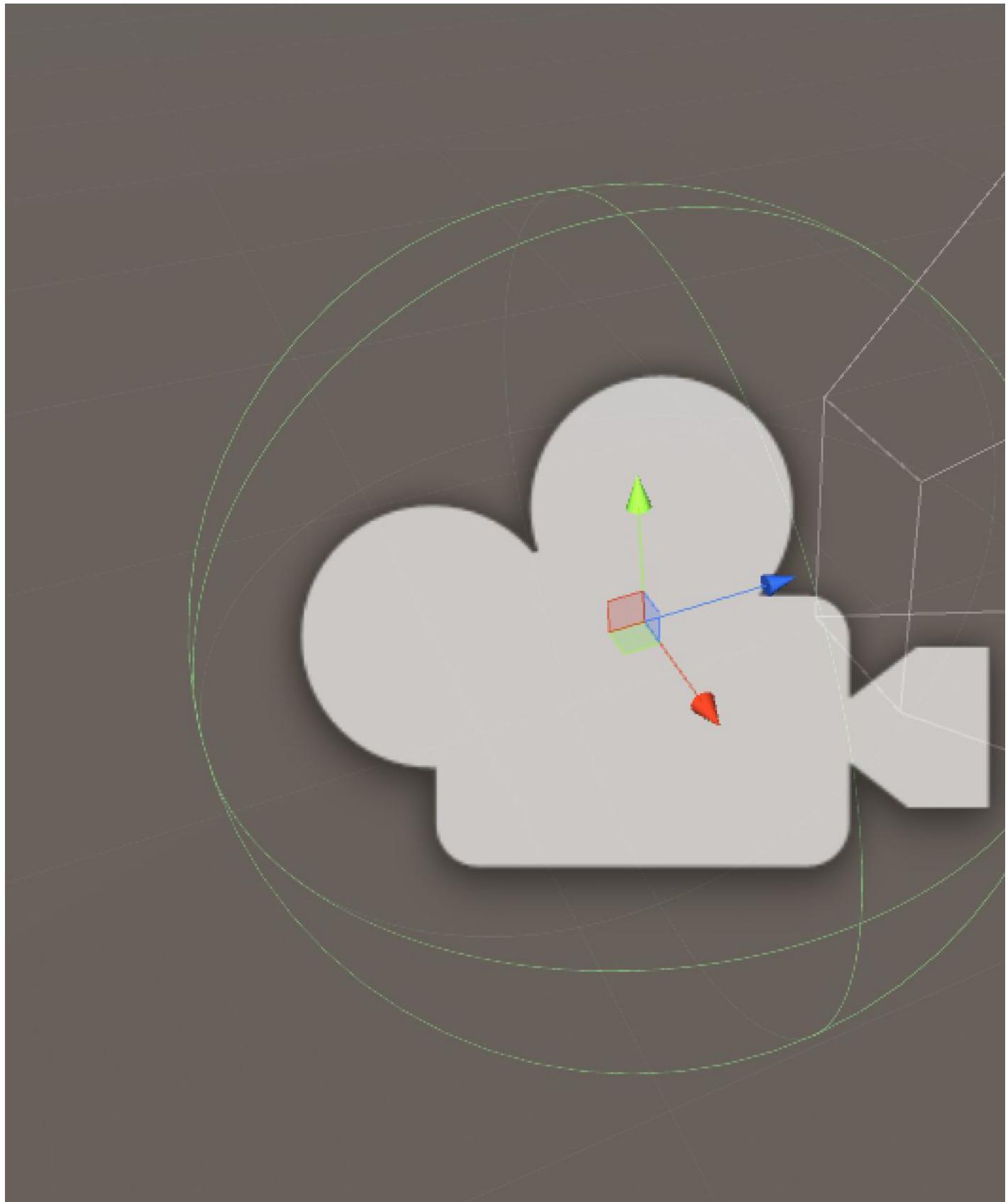


図6.5: 緑の球体がSphere Colliderだ

6.2 エネミー機能の実装

今度はエネミー機能を実装していく。ここでは最低限の機能として、次の3つの要素を定義した。

- HP・攻撃力・移動スピードのパラメータを持つ
- ゲームが開始するとプレイヤーに向かってきて、プレイヤーと衝突したらプレイヤーのHPを攻撃力分減少させる
- Bulletと衝突したら消滅する

エネミーの用意とパラメータ定義

まずは、Hierarchy上でCreate > 3D Object > Cubeを作成し、エネミーゲームオブジェクトにしよう。次に、AddComponentをクリックしてRigidbodyコンポーネントをアサインし、Use Gravityのチェックを外しておく。最後は、例によってEnemy.csを作成し、エネミーゲームオブジェクトにアサインする。

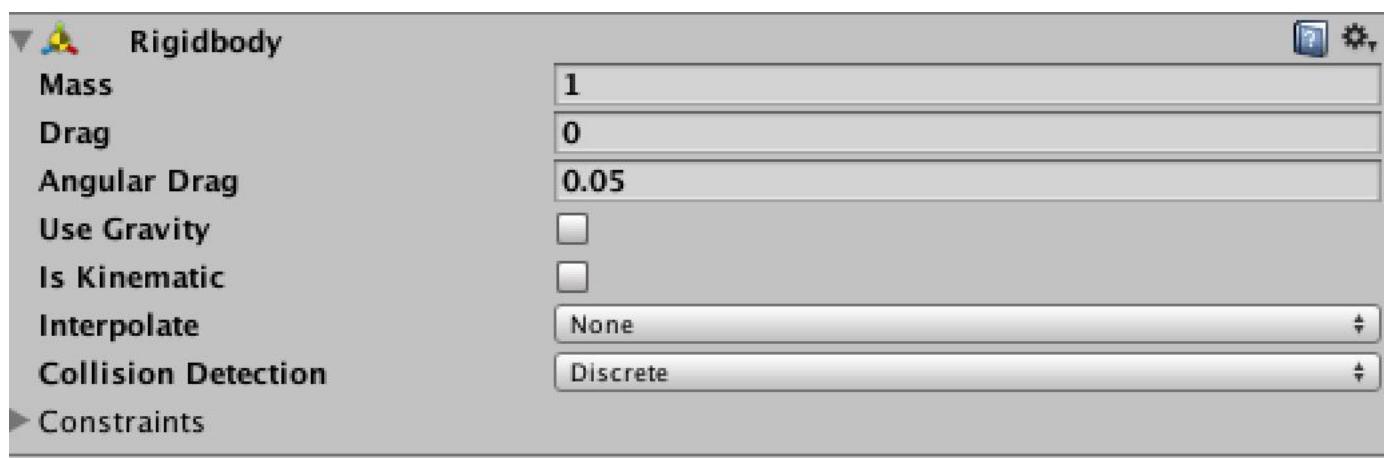


図6.6: このオプションを外すとゲーム開始時に重力によって無限に-Y方向へ移動してしまうのを防ぐ

続けて、HP・攻撃力・移動スピードのパラメータをメンバ変数として記述する。

```
public int enemyHP = 1;
public int enemyAttack = 1;
public float enemySpeed = 1;
```

エネミーの移動処理を実装する

エネミーの移動機能は、まずStart()関数でプレイヤーを探しGameObject.Find(名前)、毎フレーム分の処理を実行するUpdate()関数内で必ずプレイヤーをエネミーの正面に捉えるようにし(transform.LookAt(方向))、transform.Translate(方向)でエネミーの前方へ移動させるようとする。これで、エネミーはどのような位置にいても必ずプレイヤーに向かって移動するようになるわけだ。試しに、ゲーム実行中にプレイヤー(Main Camera)を手動で上に移動してみると、エネミーは追いかけるようについてくるはずだ。

```
private Player player;

void Start () {
    // プレイヤーゲームオブジェクトを探し、Playerコンポーネント(クラス)をメンバ変数に格納する
    player = GameObject.Find ("Main Camera").GetComponent<Player> ();
}
```

```

void Update () {
    // プレイヤーの方を向く
    transform.LookAt (player.transform);
    // 自分の前方(forward)へ移動する
    transform.Translate (transform.forward * enemySpeed, Space.World);
}

```

エネミーの衝突検知

ゲームオブジェクト同士がぶつかりあったとき、次の2つの条件であれば、その衝突を検知できる。

- ゲームオブジェクト同士がColliderコンポーネントを持つ
- ゲームオブジェクトのうち片方・もしくは両方にRigidbodyコンポーネントがある

今回はこの衝突検知を使って、次のケースを実装してみよう。

- プレイヤーから発射された弾がエネミーに当たったとき
- プレイヤーに接触したとき

EnemyクラスでOnCollisionEnter(Collision衝突)を使って実装していく。引数のCollisionで衝突した相手のゲームオブジェクト名を取得できることで、プレイヤーに衝突した場合と弾丸で撃墜された場合とで処理を分けることができる。Destroy()関数はMonoBehaviour関数の1つで、引数に指定したゲームオブジェクトをScene上から削除できる。ここでは、プレイヤーに衝突・もしくは弾丸と衝突した場合に自身を削除するようにしている。

```

void OnCollisionEnter(Collision collision) {

    GameObject collisionTarget = collision.gameObject;

    if (collisionTarget.name.Contains ("Main Camera")) {
        // プレイヤーの HP を攻撃力分減らす
        collisionTarget.GetComponent<Player> ().playerHP -= enemyAttack;
        // 自身(エネミー)を Scene 上から削除
        Destroy (gameObject);
    }
    else if(collisionTarget.name.Contains ("Bullet"))
    {
        // 自身(エネミー)を Scene 上から削除
        Destroy (gameObject);
    }
}

```

ひと通りの実装ができたら、実機にビルドするなどしてテストプレイをしてみよう。エネミーゲームオブジェクトを[Ctrl] + [D]キー(macなら[cmd] + [D]キー)で複製し、それぞれのエネミーのパラメータや位置を調整するなどして難易度を変えてみるのも良いだろう。

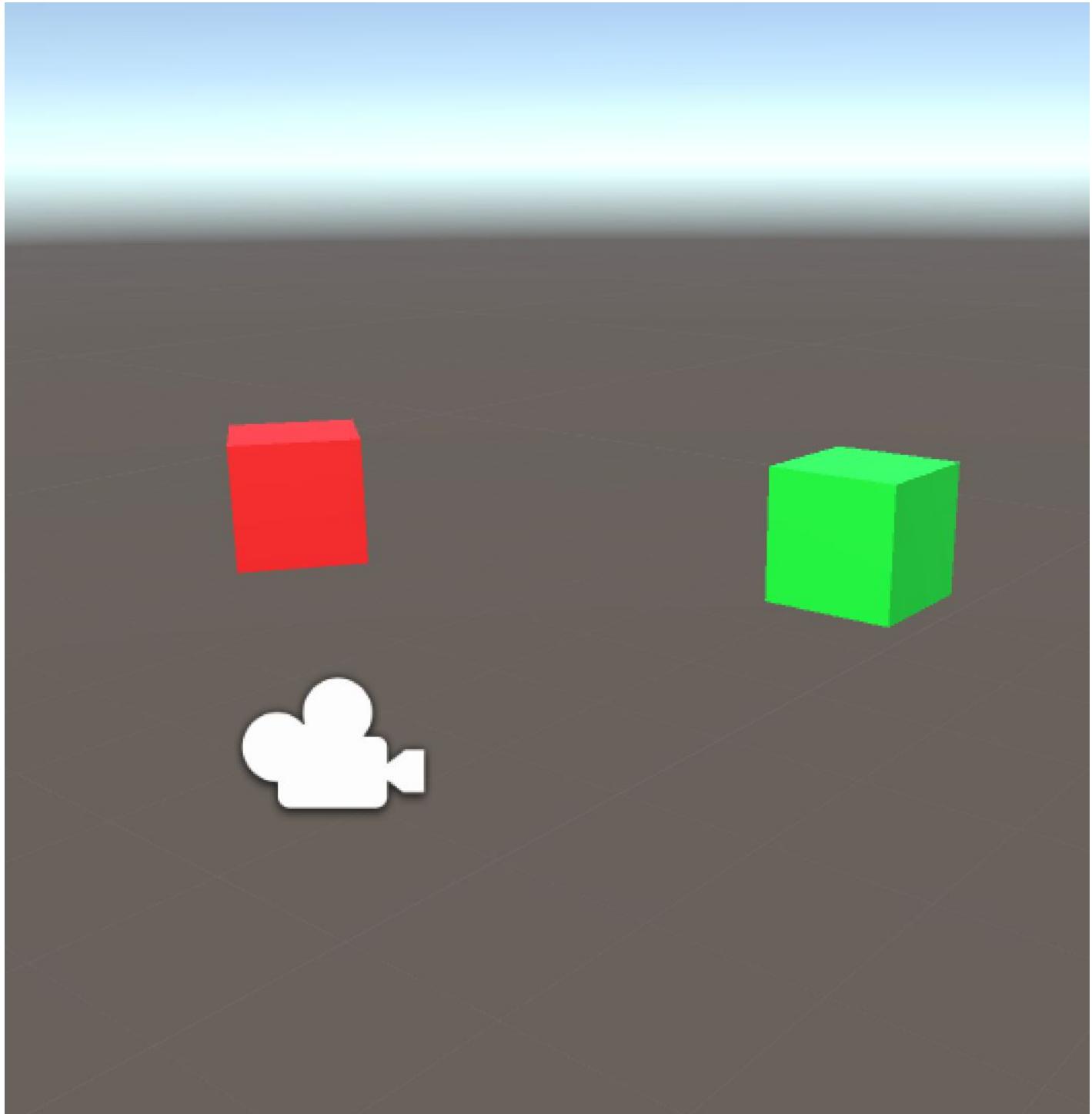


図6.7: 複数のエネミーを配置してマテリアルの色も替えることで調整がしやすくなる

6.3 おわりに

次章は、残る機能「ゲームサイクル」の実装と、グラフィックを強化しゲームの「にぎやかし」要素を増やすことで、ゲームの完成度を高めていく。

第7章 VRゲームのグラフィックを強化しよう(前編)

前章はC#スクリプトを用いて、HPや弾を撃つなどのプレイヤー機能と敵の機能などを実装した。仮のグラフィックとしてキューブなどのオブジェクトを配置していたが、これでは全く味気がない。そこで今回は、敵のグラフィックをテクスチャやアニメーションの付いた3Dモデルに置き換える、またシーン全体のライティングを調整してゲーム全体のグラフィックを強化してみたい。

7.1 Unity Asset Storeからモデルを入手する

通常、3Dモデルなどのアセットは「Maya」などのDCCツールを用いて自分で制作するか、オンライン上に公開・販売されているものをダウンロードして入手する。後者の場合、第3章でサンプルプロジェクトをダウンロードした「Unity Asset Store」からさまざまなアセットを使用することが可能だ。今回は例として、以下のアニメーション付きキャラクターモデルと背景モデルの2種類のアセットを入手し、ゲームに取り込んでみる。なお、モデルデータの扱いは基本的にはどれも同じなので、自分の好きな世界観に合うアセットを入手してもOKだ(2017年02月現在、無料ダウンロードが可能)。

- Zombie: <http://u3d.as/bFW>(図7.1)

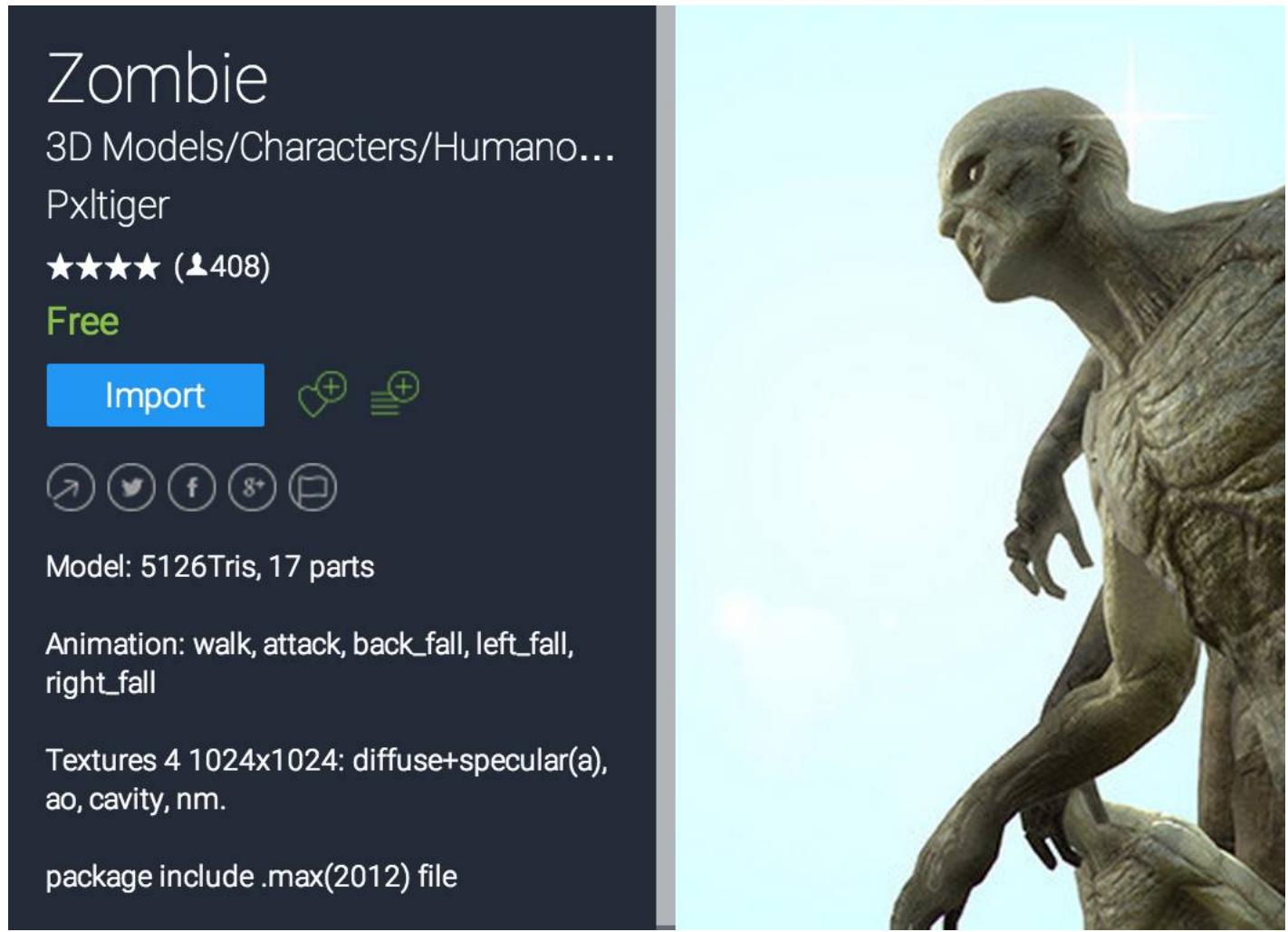


図7.1: Zombie

- Stylized Simple Cartoon City: <http://u3d.as/mb9>(図7.2)

Stylized Simple Cartoon City

3D Models/Environments/Urban

Area730

★★★★ (142)

Free

Import



A pack of stylized buildings, trees, bushes, cars, military vehicles, planes (including military), characters (with animations) and props.

Includes:

- 9 houses
- 5 planes (with animations)

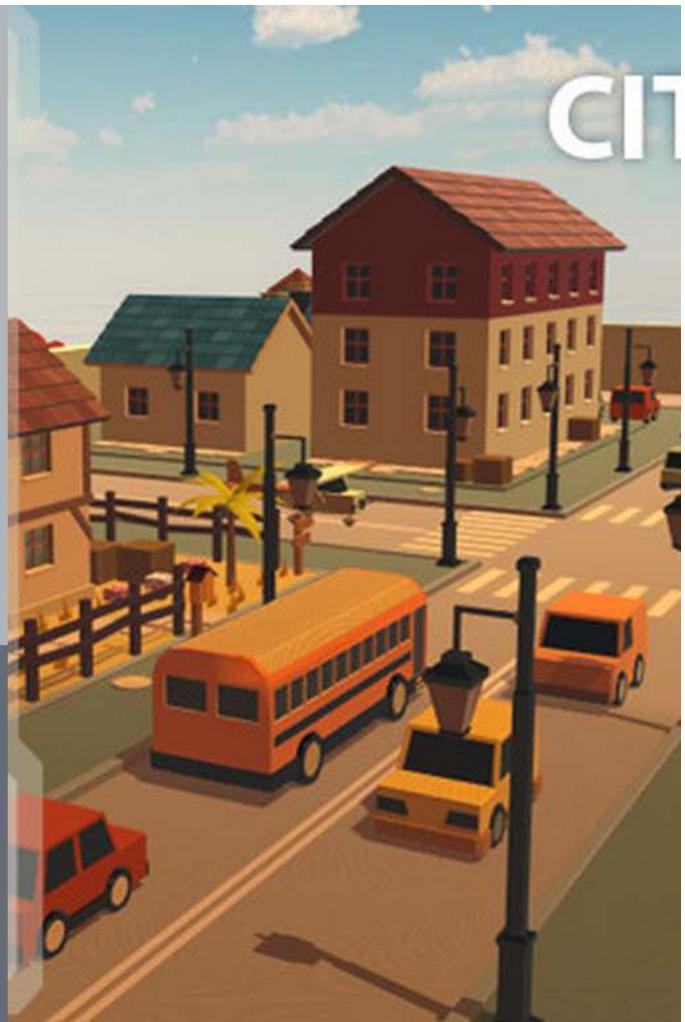


图 7.2: Stylized Simple Cartoon City

7.2 キャラクター モデルを動かしてみよう

手始めに、Asset Storeから入手したこのゾンビのキャラクターの3Dモデルを動かしてみよう。新しいシーンを作成して Assets/Zombie/Modelz@walk を配置すると、キャラクターがシーンに表示されるはずだ(図7.3)。このゲームオブジェクトは、以前解説した基本的な3DデータであるFBXデータそのものである。

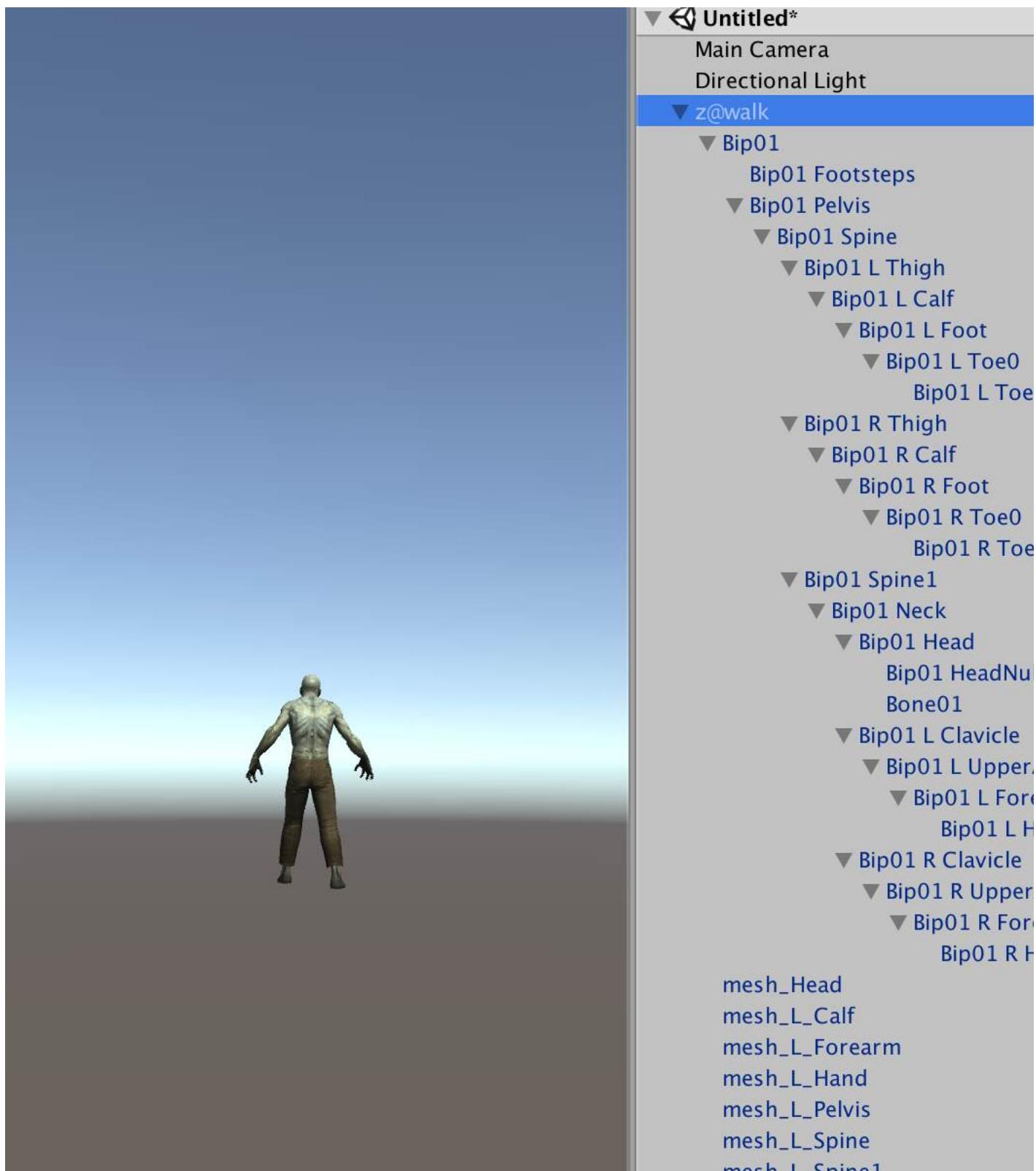




図7.3: Sceneに配置してキャラクターを構成する階層をすべて表示したところ

また、Assets/Zombie以下には、マテリアルやそれにアサインされているテクスチャ、アニメーションファイルなども含まれている（図7.4）。アニメーションファイルもファイル形式上は.fbxとなっている。

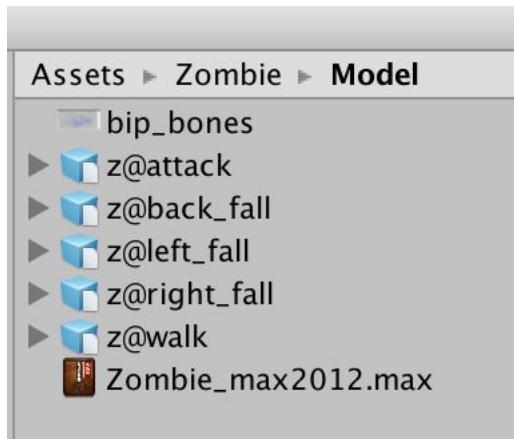


図7.4: 本アセットには3DSMaxで開けるオリジナルデータやジョイントの構成が一覧できる画像データも入っている

[column]自分で作成したモデルをUnityにインポートするには

自分で用意したモデルをUnityに入れる場合、次の点に気をつけてDCCツールからデータを出力してみよう。

- 複数人とデータを共有する場合はfbx形式で出力する
- デフォーマはジョイントのアニメーションにベイクする（ブレンドシェイプはインポート可能）
- DCCツール上のカメラ・ライト・マテリアルはUnityと互換性がない
- テクスチャをfbxにembedしない場合は手動でUnityにインポートする

アニメーションコントローラの設定

このモデルには5種類のアニメーションが含まれており、それぞれアニメーションさせることができた。しかし、それにはアニメーションコントローラを作成して必要な設定を行わなければならない。アニメーションコントローラはプロジェクトビュー上で右クリック > Animation Controllerで作成する。名前をZombieとしたらファイルをダブルクリックするとAnimatorウインドウが開く（図7.5）。



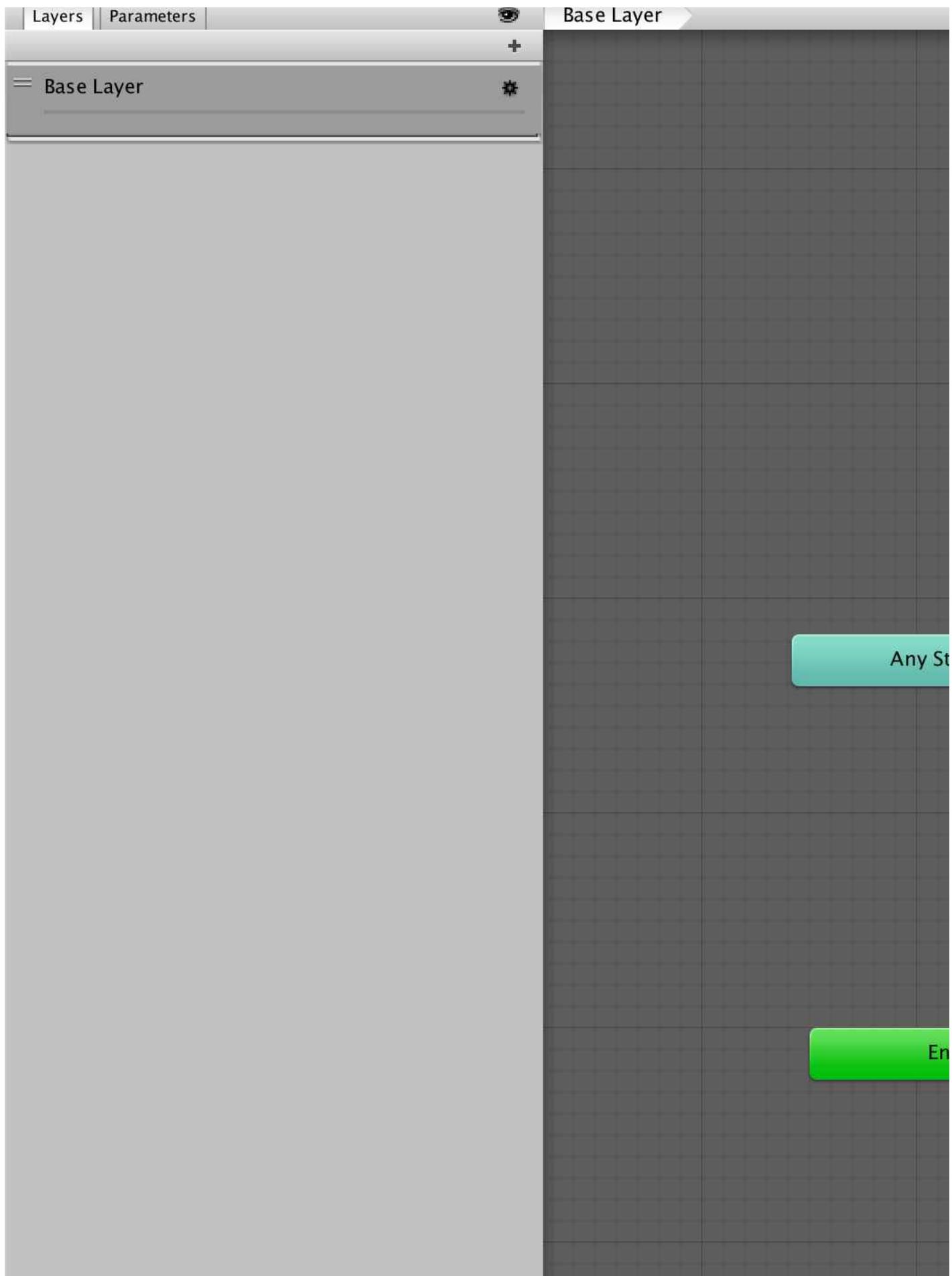
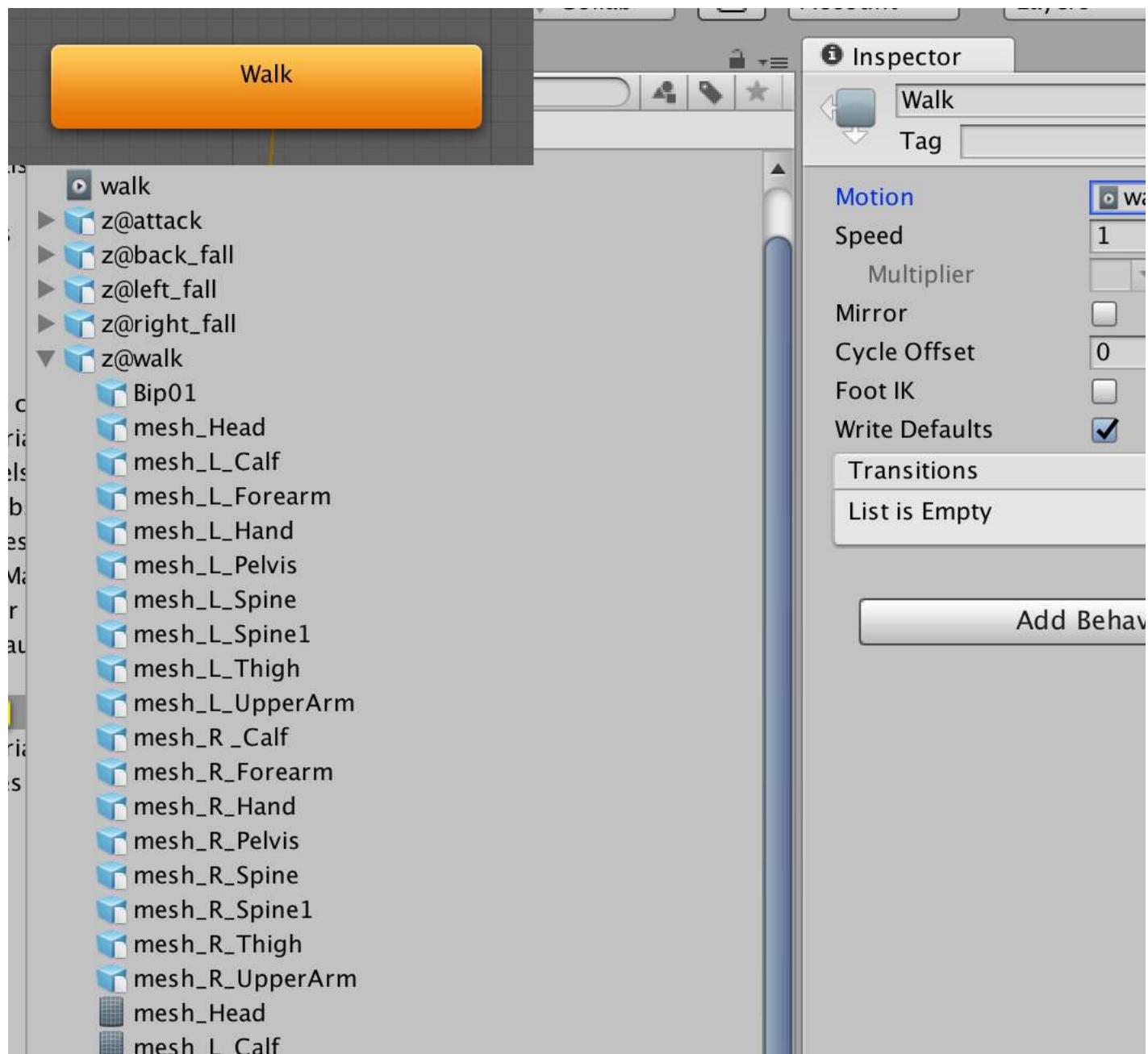




図7.5: Animatorウィンドウを開いたところ

このウィンドウでは各アニメーションの遷移を定義する。例えば、キャラクターが移動するときは「walk」アニメーションを再生するが、攻撃するときは「attack」を再生する、といった具合だ。このようなそれぞれのアニメーションの状態をステートと呼び、各ステートを定義するのがこのAnimatorウィンドウなのだ。アニメーションコントローラを作成するとデフォルトで「Entry」と「End」ステートがすでに存在している。それぞれ「開始」と「終了」を意味するステートだ。

手始めにAnimatorウィンドウ上で右クリック > Create State > Emptyと操作して、新規ステートを作つてみよう。New Stateを選択するとインスペクタにステートが表示されるので、名前をWalkにしよう。続いて、Motionフィールドにプロジェクトビューのz@walkの下階層にあるwalkというオブジェクトをアサインしてみよう(図7.6)。これでWalkステートにwalkアニメーションが設定される。



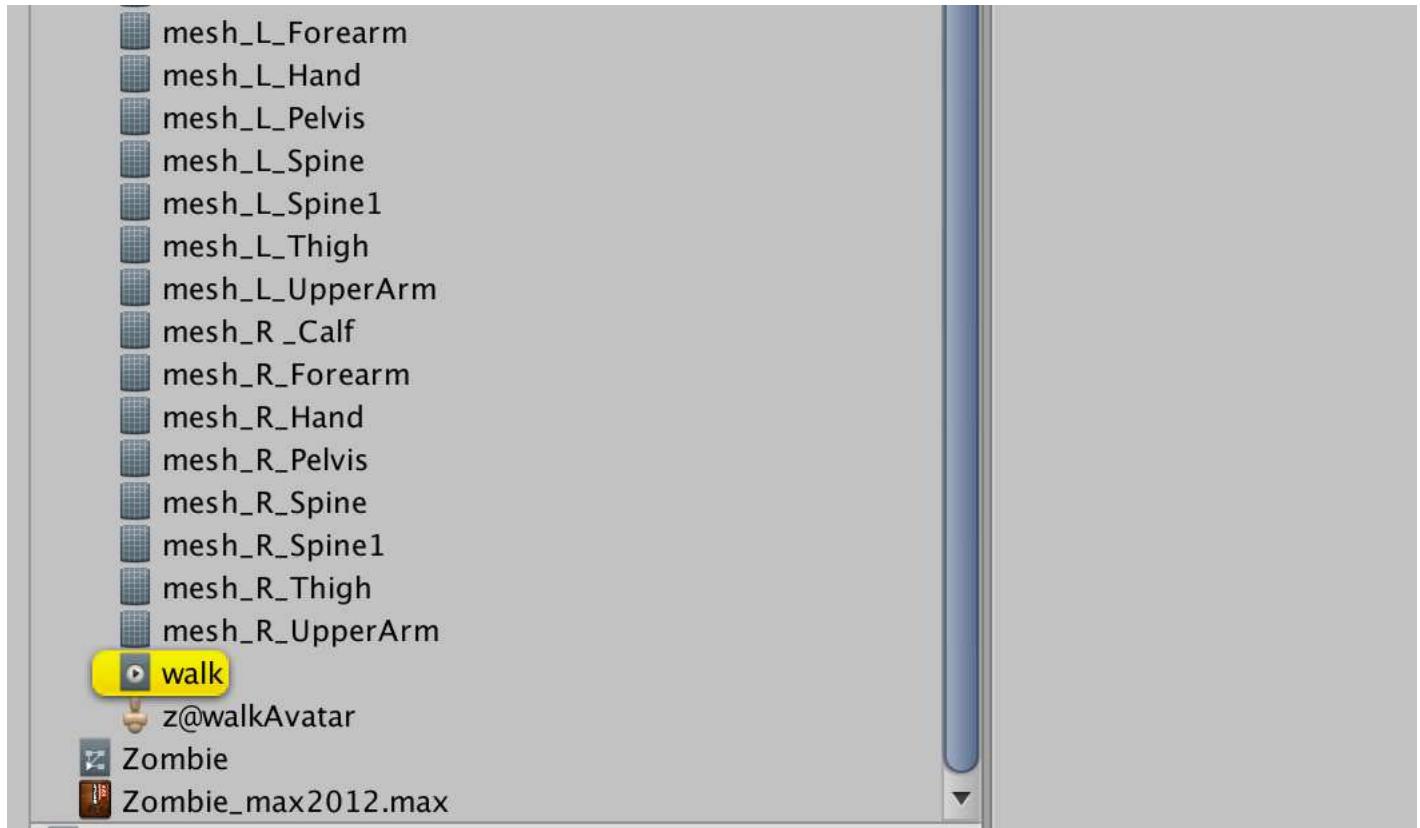


図7.6: ステートを作成してMotionフィールドにアニメーションファイルをアサインする

次に、作成したアニメーションコントローラをシーン上のz@walkのAnimatorコンポーネントのControllerにアサインしよう(図7.7)。この状態でゲームをプレイしてみると、キャラクターがアニメーションするはずだ。

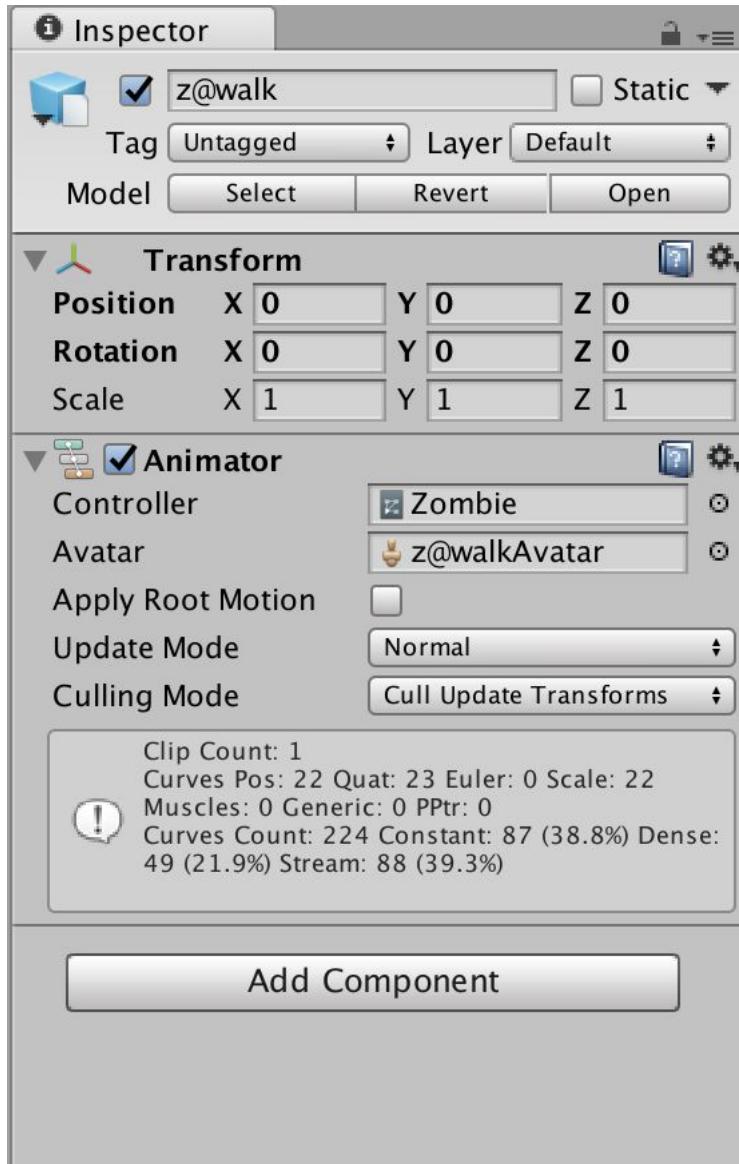


図7.7: 作成したコントローラを、ゲームオブジェクトにアサインする

これは、ゲームを開始したことでのアニメーションの初期ステートである「Entry」から歩きステートである「Walk」に遷移した、ということなのだ。また、Animatorウィンドウ上のオレンジの矢印(図7.8)は各ステートの遷移の方向を意味していることも理解しておこう。この矢印をトランジションと呼び、各ステートを右クリック > Make Transition > 遷移したいステートを押下することで設定できる。また、今回のWalkステートのように最初に作成したステートをデフォルトステートと呼び、ステートがオレンジ色になっている。これは、あらかじめEntryステートからトランジションが設定されているということを覚えておこう。

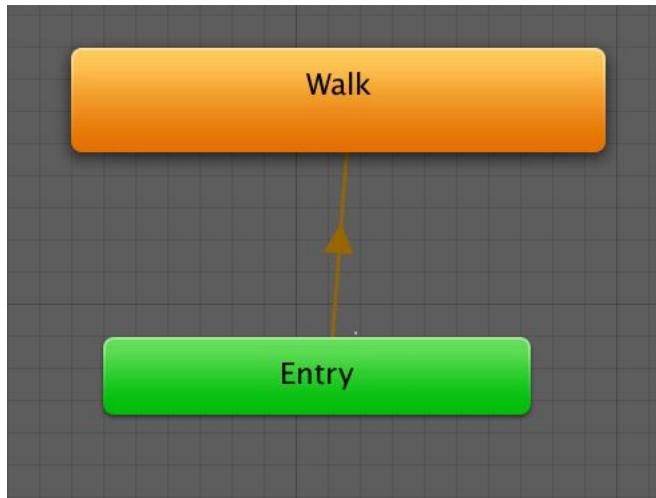


図7.8: EntryステートからWalkステートに遷移することでキャラクターがアニメーションした

しかし、このままではWalkアニメーションが1ループしただけで終了してしまうので、プロジェクトビューのz@walkを選択し、インスペクタからAnimationタブ > Loop Timeにチェックを入れてApplyしておこう(図7.9)。これでWalkアニメーションがループ再生される。

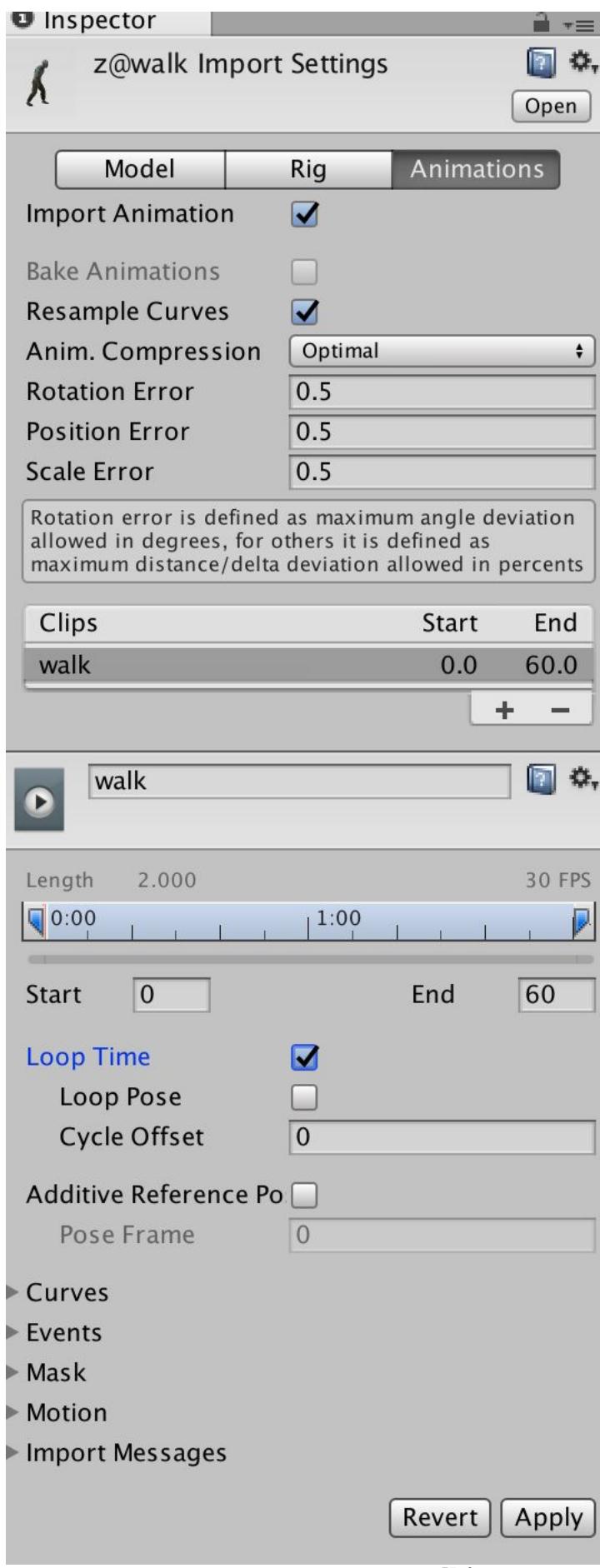


図7.9: この設定でアニメーションがループ再生されるようになる

最終的に、全5種類のアニメーションを図7.10のような配置でステート設定した。「Dead」という名前のステートがあるが、これはLeft FallおよびRight Fallアニメーションの終了時にエネミーが消滅するのを自然に見せるためのいわばダミーステートで、アニメーションはアサインされていない。

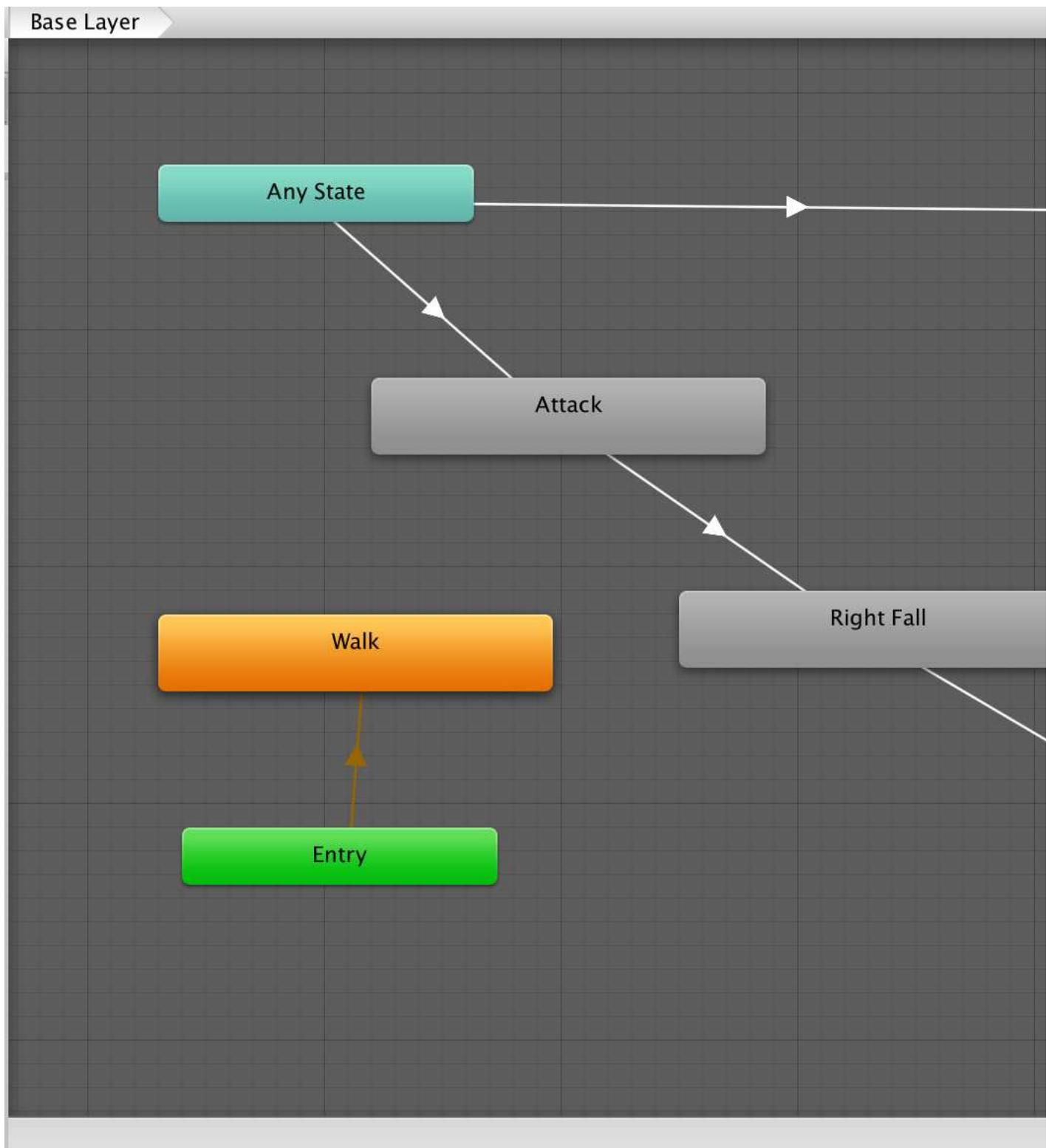
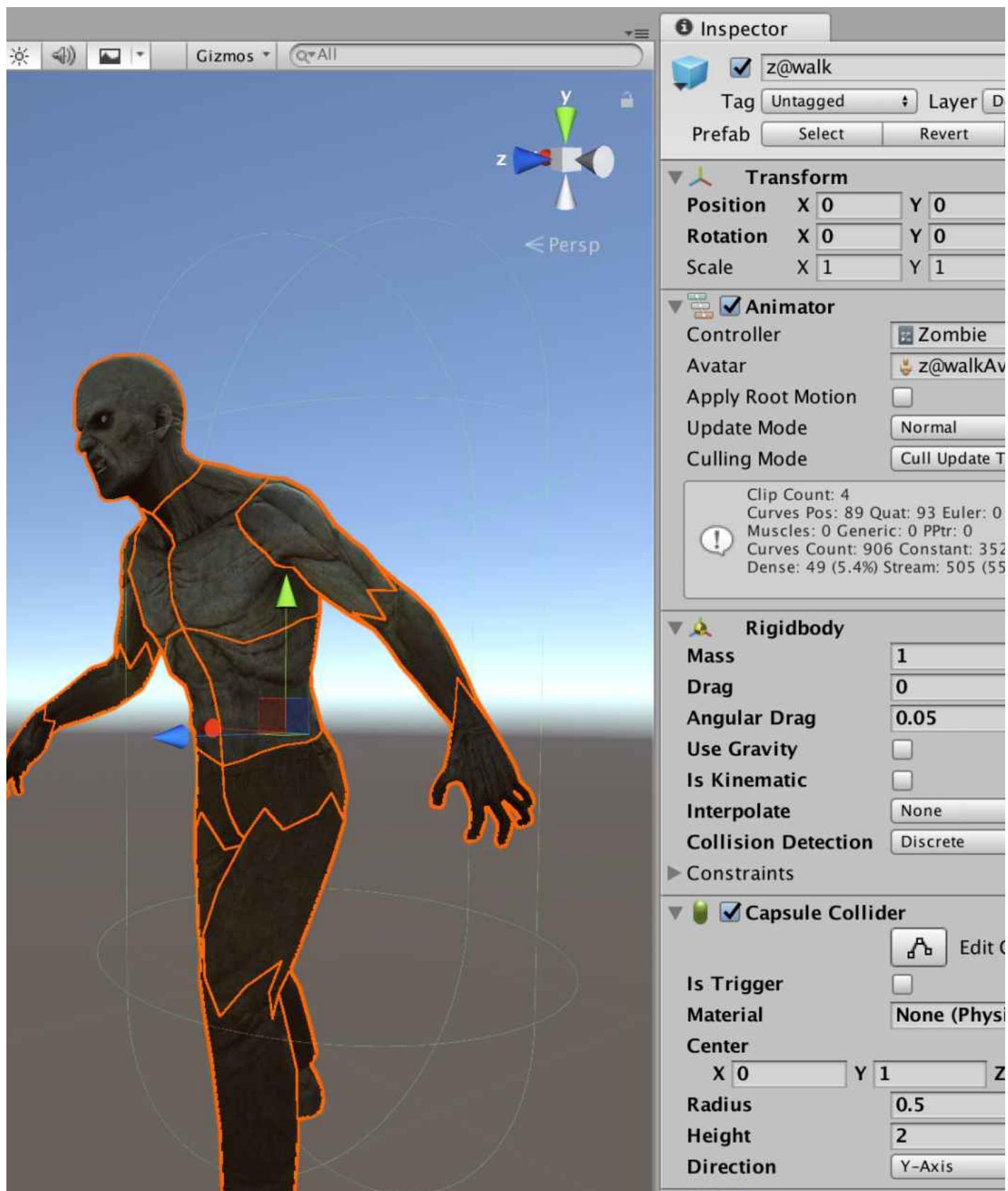


図7.10: 全5種類のアニメーションの最終的なステート配置

Enemyクラスのアサイン

前章「VRシーティングゲームを実装しよう」でEnemyクラスを作成したときは仮モデルとしてキューブオブジェクトを使用したが、今回はz@walkにEnemyクラスをアサインする。Enemy Speedは0.01前後がちょうどいいだろう。このEnemyクラスでは衝突検知のためにコライダーやジッドボディコンポーネントを使用するため、それらもアサインしておこう。今回のような縦に長い形状のキャラクターにはCapsule Colliderを設定するのが一般的だ。コライダのサイズ調整も忘れずに。RigidbodyはUse Gravityのチェックを外しておこう(図7.11)。これで試しにゲームを実行してみると、キャラクターが歩きながらカメラ(プレイヤー)に近づいてくるのが確認できるはずだ。



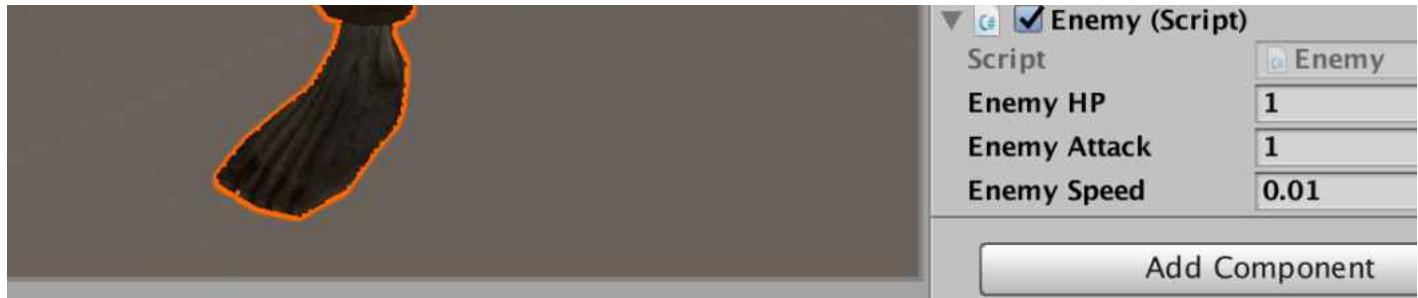


図7.11: キャラクターにEnemyクラス、Capsule Collider、Rigidbodyをアサイン

なお、前回作成したEnemyクラスはMain CameraにPlayer.csがアサインされていないと動作しない。注意しよう。

スクリプトからのステート制御

続いて、エネミーの行動によって再生するアニメーションを細かく制御していく。前回作成したEnemy.csを次のように修正しよう。

```

~~~

public Animator anim;

~~~

void OnCollisionEnter(Collision collision) {

    GameObject collisionTarget = collision.gameObject;

    if (collisionTarget.name.Contains ("Main Camera")) {
        // 行動を停止
        Stop ();
        // 攻撃ステート開始
        anim.SetTrigger("Attack");
    }
    else if(collisionTarget.name.Contains ("Bullet"))
    {
        // 自身のコライダを無効
        gameObject.GetComponent<Collider>().enabled = false;
        // 行動を停止
        Stop ();
        // 撃破ステート開始
        anim.SetTrigger ("Left Fall");
    }
}

void Stop(){
    // 移動を停止 & Rigidbodyを無効
    enemySpeed = 0;
    gameObject.GetComponent<Rigidbody> ().isKinematic = true;
}

public void OnFinishedAttack() {
    // 自身のコライダを無効
    gameObject.GetComponent<Collider>().enabled = false;
    // プレイヤーの HP を攻撃力分減らす
    player.playerHP -= enemyAttack;
}

```

```

public void OnFinishedFall() {
    // 自身(エネミー)を Scene 上から削除
    Destroy (gameObject);
}

```

新たに`anim`というメンバ変数を`public`で設けたので、ゲームオブジェクト上のフィールドにEnemy自身のAnimatorコンポーネントをアサインすることを忘れずに(図7.12)。

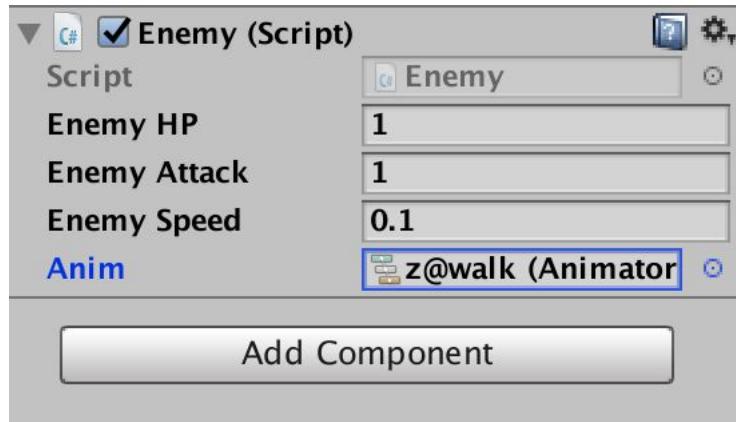


図7.12: Animatorフィールドに自身のAnimatorコンポーネントをアサインする

今回の修正は、次の2つの目的がある。

1. 各アニメーションステートへの遷移処理
2. アニメーションステートが終了したかどうかの処理

各アニメーションステートへの遷移処理

衝突検知時の分岐で`SetTrigger()`というメソッドがあるが、これはアニメーショントリガを呼ぶための処理である。アニメーショントリガはアニメーションコントローラで設定したトランジションを実行するためのパラメータで、スクリプトからステート遷移を行うために欠かせないものだ。つまり、スクリプトから`SetTrigger("トリガ名")`を実行することでトリガによってトランジションが開始され、アニメーションステートが遷移する。例えば、Walkステート中にスクリプトからAttackトリガが呼ばれるとステートがAttackに遷移し、Attackアニメーションが再生されるわけだ。

このアニメーショントリガはまだ未作成なので、Zombieアニメーションコントローラをもう一度開いて設定しよう。図7.13に示すボタンでアニメーショントリガを作成できる。名前は「Attack」としておこう。

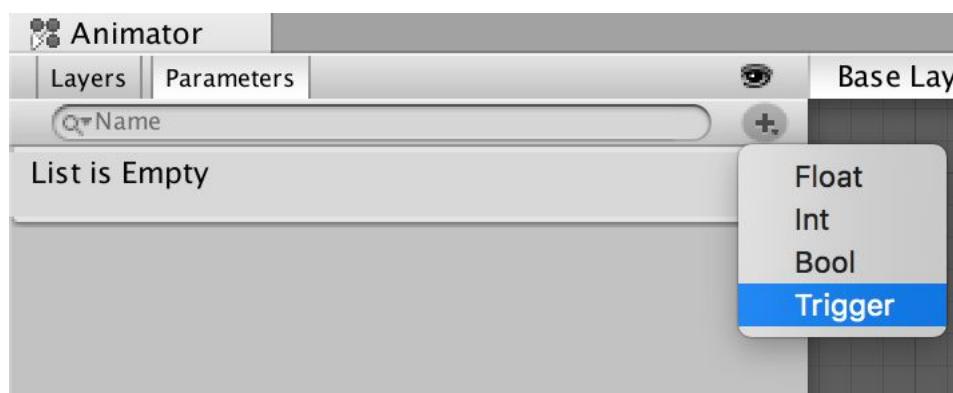


図7.13: アニメーショントリガの作成

Any StateからAttackステートへのトランジション(矢印)を選択すると、インスペクタにトランジションの内容が表示される。ここで Conditionsに先ほど作成したアニメーショントリガ「Attack」を設定すればOKだ(図7.13)。同様にAny StateからLeft FallにもLeft Fallトリガを作成してトランジションを設定しておこう。

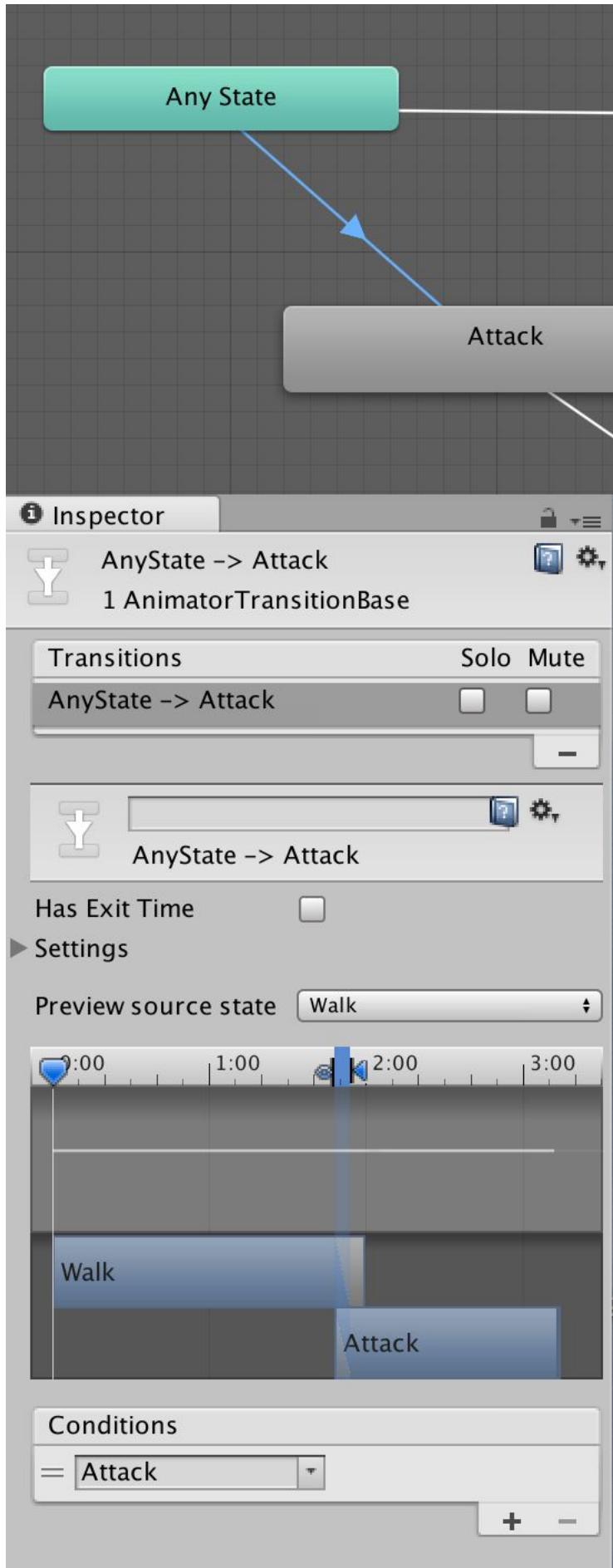


図7.14: トランジション(矢印)を選択してAttackステートのConditionにトリガを設定する

これで、スクリプトからプレイヤーへの攻撃時にはAttackトリガが、エネミー撃破時にはLeft Fallトリガが呼ばれ、それぞれのステートにトランジションするようになった。

アニメーションステートが終了したかどうかの処理

「あるステートが終了したタイミングで何らかの処理をしたい」というケースがある。例えば、Left Fallアニメーションが終了したら OnFinishedFall() メソッドを呼びたい、という場合だ。このような場合は StateMachineBehaviour を継承したクラスを用意することで対応できる。

まず、名前をEnemyFall.csとする新規クラスを作成して、次のようにしてみよう。

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

public class EnemyFall : StateMachineBehaviour {

    override public void OnStateExit (Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.GetComponent<Enemy> () .OnFinishedFall ();
    }
}
```

スクリプト内のOnStateExit()はアニメーションステートが終了するときに呼び出されるオーバーライドメソッドだ。ここでは、アニメーション再生が終わりステートが終了する間際にエネミーを消去するメソッドであるOnFinishedFall()が呼ばれるようになっている。なお、StateMachineBehaviour継承クラスはゲームオブジェクトではなくアニメーションステートにアサインしなければならないため、アニメーションコントローラを開いてインスペクタからLeft FallおよびRight Fallステートにアサインしよう(図7.15)。



図7.15: StateMachineBehaviour継承クラスをアニメーションステートにアサインする

同様に、Attackステート終了時にもプレイヤーのHPを減らす処理 `OnFinishedAttack()` を実行したいため、以下の内容でスクリプトを作成しAttackステートにアサインしておこう。

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

public class EnemyAttack : StateMachineBehaviour {

    override public void OnStateExit (Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.GetComponent<Enemy> ().OnFinishedAttack ();
    }
}
```

なお、StateMachineBehaviour継承クラスでは、他にもさまざまなステートのタイミングで処理を呼び出すことができる。ぜひ[APIリファレンス*1](#)で確認してみよう。

[*1] <https://docs.unity3d.com/ScriptReference/StateMachineBehaviour.html>

7.3 動作確認をしよう

ここまで の作業が完了したら、改めて動作確認をしてみよう。次の流れで処理が実行されていれば完璧だ(図7.16)。

- エネミーがプレイヤーによって撃破される場合
 - Walk → 弾丸の衝突検知 → Left Fall → OnFinishedFall()によって消滅
- エネミーがプレイヤーに攻撃する場合
 - Walk → プレイヤーの衝突検知 → Attack → OnFinishedAttack()によってプレイヤーHP減少 & Right Fall → OnFinishedFall()によって消滅

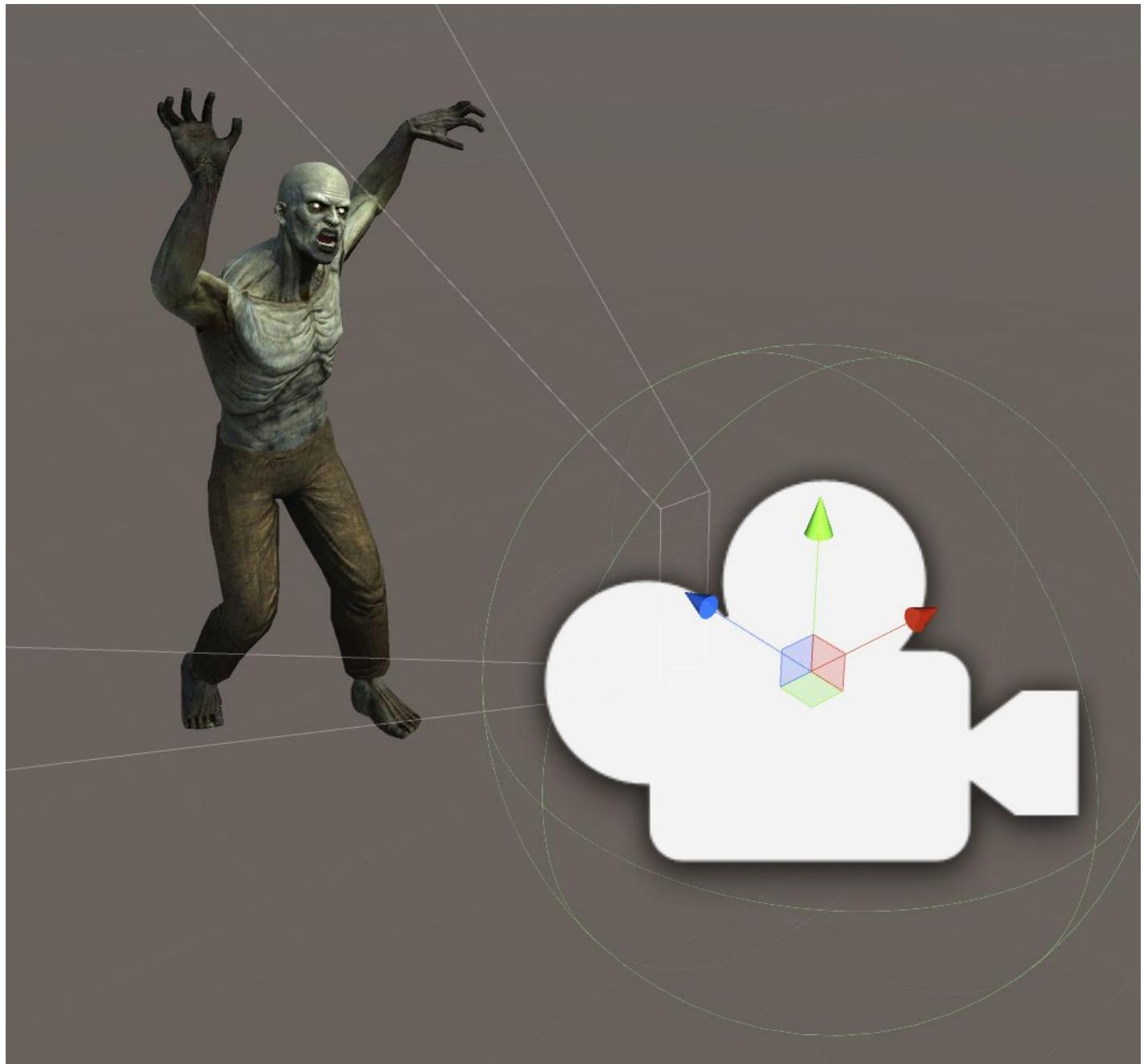


図7.16: カメラ(プレイヤー)にキャラクターが移動 & 攻撃しているところ

最後に、シーン上のz@walkはゲームオブジェクトとしての状態を保存するためにプレハブ化しておこう。

7.4 おわりに

これで、グラフックを強化してゲームの完成度を高めることができた。次章は背景モデルにライティングを施して、ゲームをよりリッチな見た目にしていく。

第8章 VRゲームのグラフィックを強化しよう(後編)

前章は、アニメーションの付いた3Dキャラクターモデルの設定を行い、それをエネミークラスと連携させる方法を解説した。次は、背景モデルに対してライティングを行うことで、ゲームの見た目をよりリッチにしてみよう。なお今回の解説では、Unityアセットストアで無料で公開されている[Stylized city](#)を使用する。こちらを利用したい人は、事前にアセットストアから入手し、プロジェクトにインポートしておこう。

8.1 ライティングとは

CGのライティング(Lighting)とは、その名の通り「照明」の効果をシーンに適応することだ。例えば現実世界では、屋内において天井の照明や暖炉の火の明かりの有無などで、部屋の見え方が大きく変わってくるし、屋外では太陽という「照明」が時間によって変化することで朝・昼・夜といった異なる状況が生まれる。ライティングは、このような光による変化をCG空間上で再現するための手段なのだ。しかし、リアルタイムで描画されるCGの照明は現実と異なったはたらきをしているため、それを理解するためにはいくつかポイントを抑えておく必要がある。なお、ライトそのものに関する基礎的な内容はここでは解説しないので、Unityの[公式ページ*1](https://docs.unity3d.com/ja/current/Manual/LightSources.html)などで確認しておこう。

[*1] <https://docs.unity3d.com/ja/current/Manual/LightSources.html>

直接光と間接光

ゲームエンジンにおける光には、「直接光」と「間接光」の2種類がある。ここでは、ライトから直接照射される光が直接光で、その直接光が物体に当たって跳ね返った光を間接光としておこう。間接光は光が跳ね返る分だけ、何回も計算を行う必要があるため、リアルタイムCGで表現するには難しい。そのため、間接光は事前に計算しデータに保存しておくことで、ゲーム中に表現することができる。このような直接光と間接光の表現を、一般的なCGではグローバリルミネーション(GI)表現と呼ぶので覚えておこう。このGI表現はUnityだと、「リアルタイムGI」・「ライトマップ」・「ライトプローブ」の3種類の方法で表現できるが、リアルタイムGIはモバイルVRには処理負荷的に適さないためは除外し、ライトマップ・ライトプローブの2つの方法を解説していく。

リアルタイムライトとベイクライト

ライトにも、「リアルタイムライト」と「ベイクライト」、その2つの両方の性質をもつ「ミックスライト」の3つの種類がある。これらを見分けるために、ライトのインスペクタの「Mode」に表示されている項目を見てみよう。リアルタイムライトは、直接光として常にオブジェクトに対して光を照射し、リアルタイムGIの間接光にも影響を与える。(ここでは詳しくは解説しない)

ベイクライトは、ライトマップやライトプローブを作成するときに使用されるライトだ。ライトマップというのは、直接光及び間接光を事前に計算し、その結果をテクスチャとして保存したものだ。一方のライトプローブは、直接光と間接光の情報を「空間」に保存する。このように事前に計算し保存しておいた光の情報を使用することで、直接光+間接光の表現をゲーム中にリアルタイムで行えるわけだ。ミックスライトは、前述の通りリアルタイムライト・ベイクライトの2つの性質をもつ。

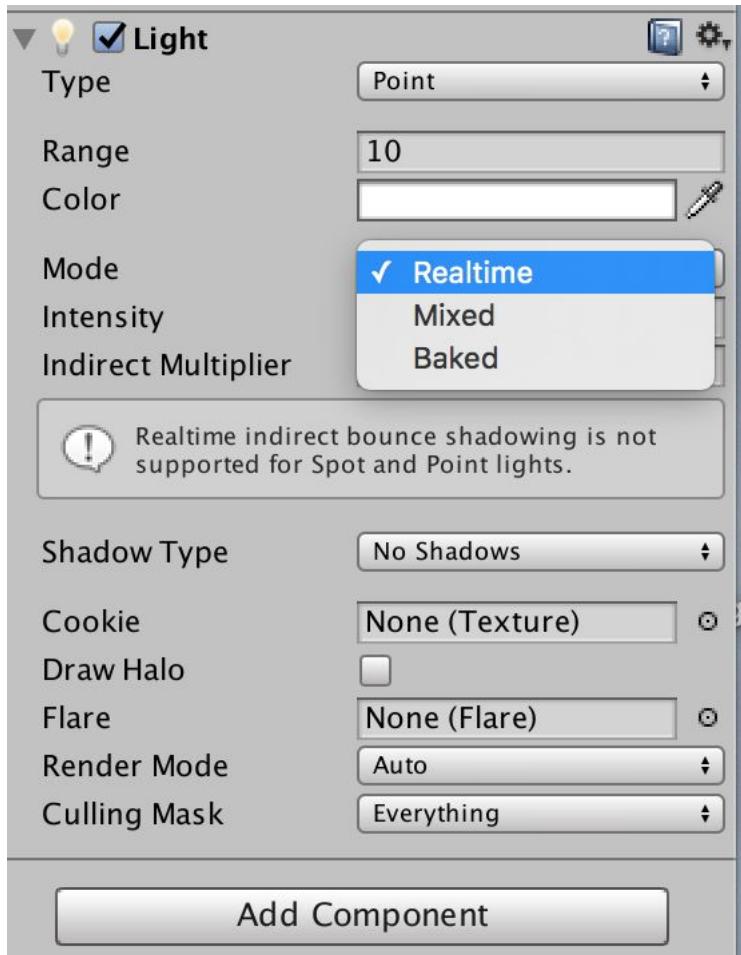


図8.1: ライトのモードを抑えておこう

staticオブジェクトとnon-staticオブジェクト

ライティングに関する最後の解説として、static / non-static オブジェクトについて説明しておこう。その場から動かないオブジェクトを static オブジェクト、そうでないものを non-static と呼ぶ。具体的には、背景モデルなどは、ゲーム中に絶対にその場から動かないモデルが static オブジェクトで、キャラクター モデルや、背景モデルでもドアや宝箱といった、アニメーションするオブジェクトが non-static オブジェクトだ。このようにモデルを2つに分けるのは、主に描画負荷を減らすための仕組みである「ドローコールバッ칭」が動作するようにするためだ。 static になっているオブジェクトは、たとえ異なるオブジェクトでも、諸条件さえ揃えば自動で描画をひとまとめにするドローコールバッ칭が行われる。それによる処理負荷軽減の恩恵をうけるため、背景などの動かないオブジェクトは、必ず static オブジェクトにする必要がある。

ライティングの話に戻すと、static オブジェクトはライトマップ、そして non-static オブジェクトはライトプローブによって、ライティング表現する。

すべてのゲームオブジェクトは、何もしなければ常に non-static オブジェクトの状態だ。インスペクタの右上に表示されている「static」オプションを押下すれば、static オブジェクトにすることができる。

ライティングのポイントまとめ

これまでに解説したポイントをまとめると、以下のようなマトリクスになる。

表8.1:

オブジェクトの種類	表現方法	具体例
static	ライトマップ	背景モデルなど
non-static	ライトプローブ	キャラクター モデル・動く背景モデルなど

※リアルタイム GI を使用するケースは含まず

8.2 背景モデルをライティングしてみよう

ライティングに関する大まかな概要をつかめたところで、実際にシーンをライティングしてみよう。今回使用する背景アセット「Stylized Simple Cartoon City」は、Assets > Area730 > Stylized city > Prefabsにモデルのプレハブが保存されている。新規のシーンを作り、そこへプレハブを組み合わせて配置し、背景シーンを組み立ててみよう。

ヒエラルキー パネル上のCreate Emptyで空のトランスフォームを作成し、そこに背景のオブジェクトを入れておくと、ヒエラルキーがすっきりするので、こちらもやっておこう。

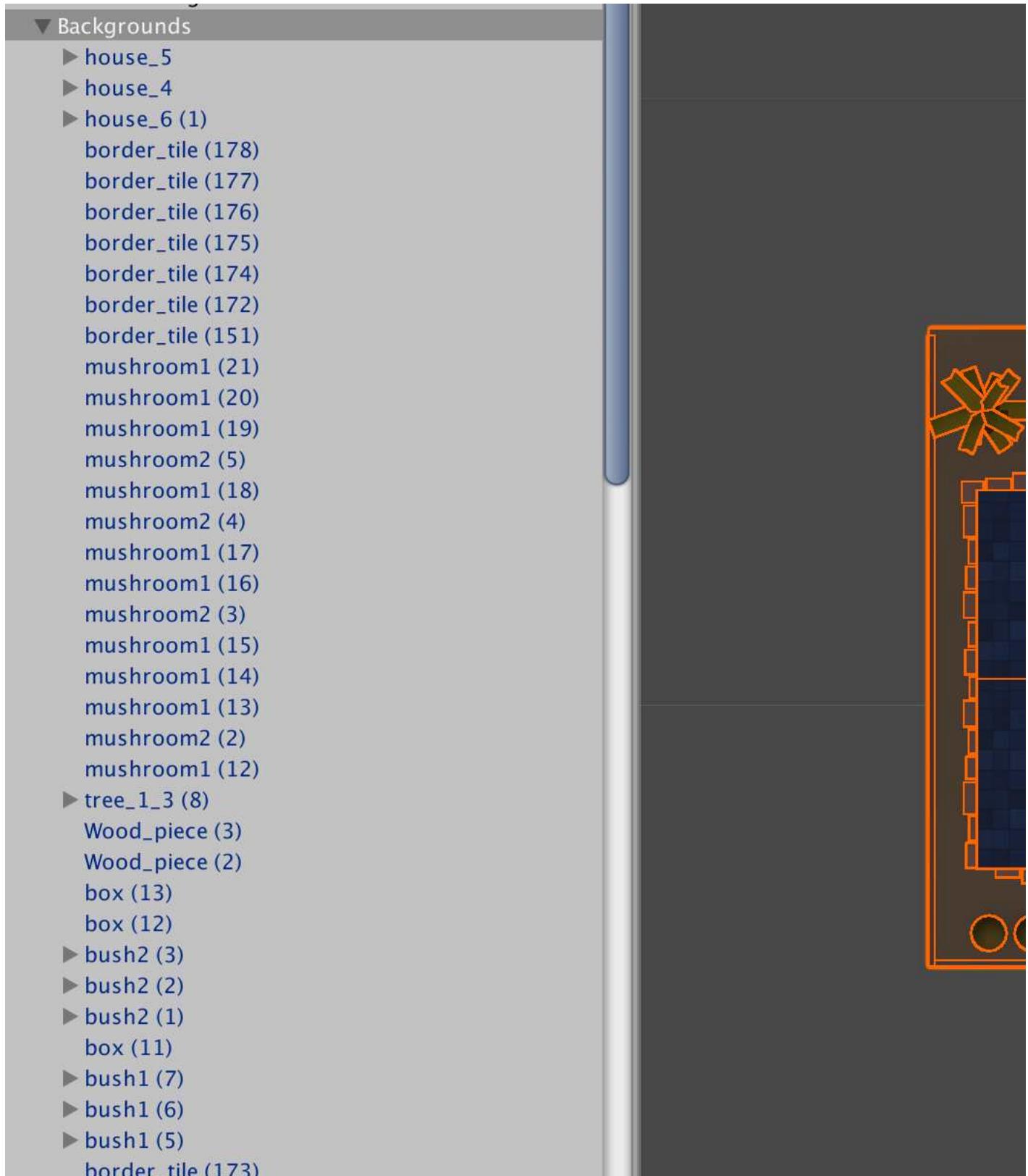


図8.2: 空のTransformはオブジェクトのグルーピングにも活用できる

また、背景オブジェクトはstaticオブジェクト扱いとなるので、ルートとなるトランスマーフォームを選択してから、インスペクタ上のstaticのチェックボックスを有効にしておこう。この時、ダイアログで「Yes, change children」とし、子オブジェクトすべてに対してstaticが有効になるようにしよう。

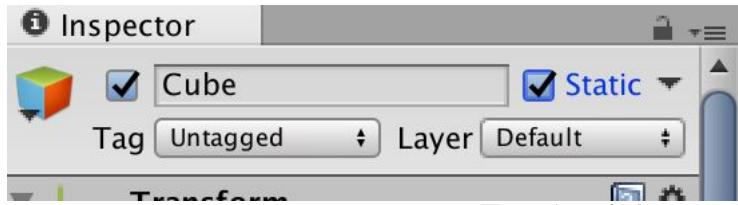


図8.3: インスペクタのstaticオプションを有効にする



図8.4: ダイアログにて「Yes change children」を選択することで、オブジェクトの子供までオプションが有効になる

最後に、シーンの保存も忘れずに。

アセットの最適化

次は、アセットの最適化だ。本アセットの殆どのマテリアルは Standardマテリアルと呼ばれるPBR表現(物理的に正確な光表現)が行えるマテリアルで構成されており、それだと重すぎてモバイル用途に向かないため、軽量なマテリアルに変更する必要がある。

Assets > Area730 > Stylized city > Materialsの中にあるマテリアルアセットをすべて選択し(この時フォルダは選択しないよう)に、マテリアルのシェーダを Legacy Shaders > Diffuse に変えておこう。

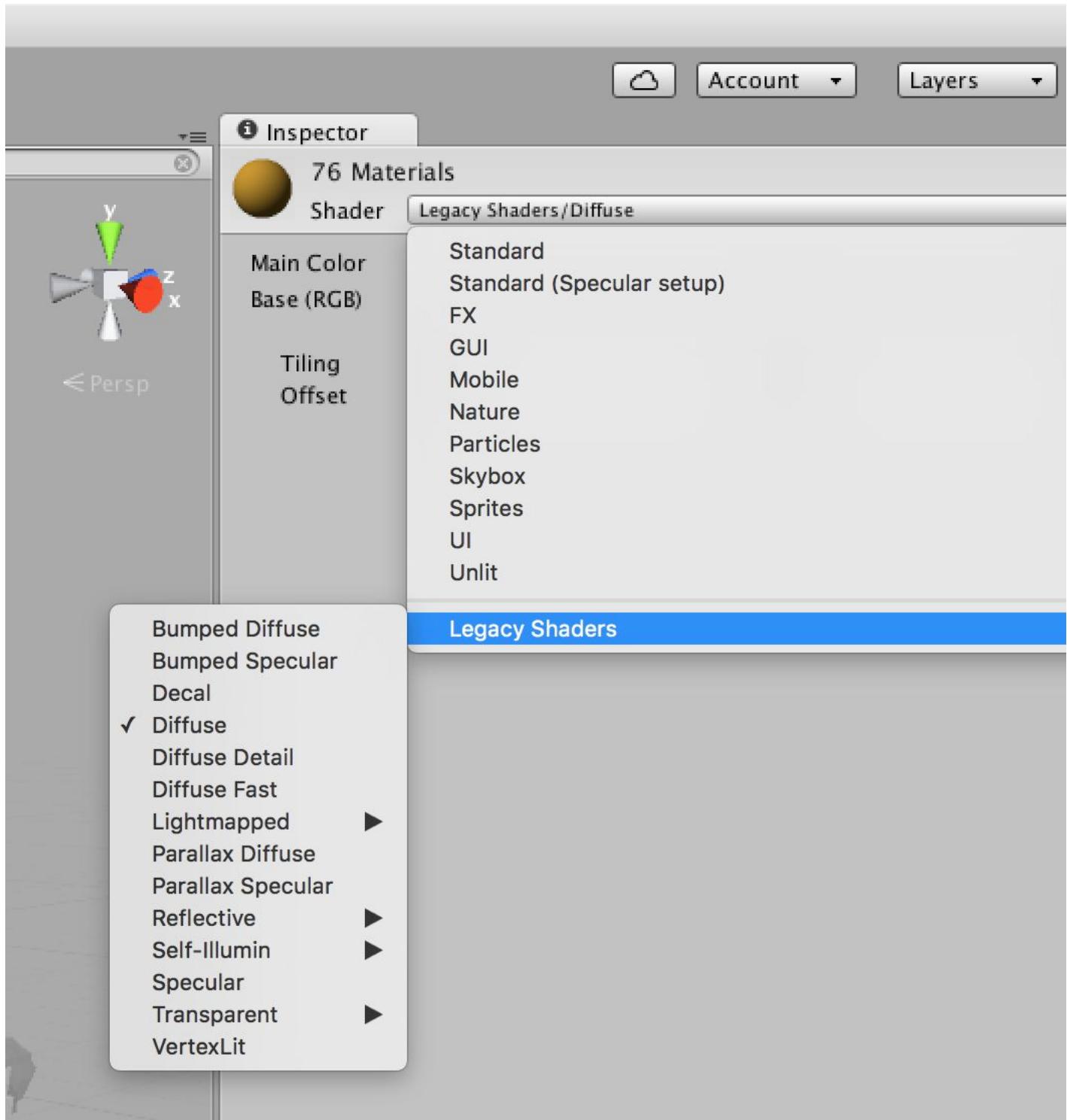
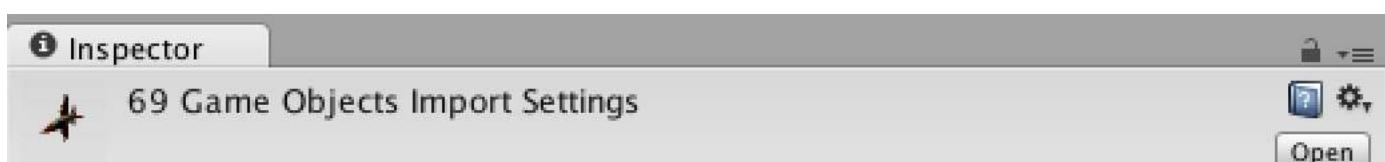


図8.5: マテリアルのシェーダを軽量なものに変更する

最後に、Assets > Area730 > Stylized city > Modelsから、マテリアルの時よ同様にすべてのモデルアセットを選択し、インスペクタからGenerate Lightmap UVsにチェックを入れ、Applyしておこう。これは、後で解説するライトマップが正しくモデルに反映されるようにするためにだ。



Model **Rig** **Animations**

Meshes

Scale Factor	1
File Scale	1
Mesh Compression	Off
Read/Write Enabled	<input checked="" type="checkbox"/>
Optimize Mesh	<input checked="" type="checkbox"/>
Import BlendShapes	<input checked="" type="checkbox"/>
Generate Colliders	<input type="checkbox"/>
Keep Quads	<input type="checkbox"/>
Swap UVs	<input type="checkbox"/>
Generate Lightmap UVs	<input checked="" type="checkbox"/>

► Advanced

Normals & Tangents

Normals	Import
Smoothing Angle	<input type="range" value="60"/>
Tangents	Calculate Legacy – Split Tangents

Materials

Import Materials	<input checked="" type="checkbox"/>
Material Naming	By Base Texture Name
Material Search	Recursive-Up

For each imported material, Unity first looks for an existing material named [BaseTextureName].
 Unity will do a recursive-up search for it in all Materials folders up to the Assets folder.
 If it doesn't exist, a new one is created in the local Materials folder.

Imported Object

<input checked="" type="checkbox"/>	—	<input type="checkbox"/> Static	
Tag	Untagged	<input type="button" value=""/>	
Transform	<input type="button" value="?"/> <input type="button" value=""/>		
Position	X <input type="text" value="—"/>	Y <input type="text" value="—"/>	Z <input type="text" value="—"/>
Rotation	X <input type="text" value="—"/>	Y <input type="text" value="—"/>	Z <input type="text" value="—"/>
Scale	X <input type="text" value="—"/>	Y <input type="text" value="—"/>	Z <input type="text" value="—"/>

Components that are only on some of the selected objects cannot be multi-edited.

図8.6: ここで自動生成されるセカンダリUVをもとにライトマップがマッピングされる

なお、自前の背景モデルを使用する場合も、上記の処理をマテリアルとモデルに対して行えばOKだ。

ライトの配置

いよいよライトを配置していこう。まずは、Create > Light > Directional Lightで、直接光となるライトを配置してみよう。Directional Lightは太陽光などの平行光源を表現するのに使われるライトで、今回のような屋外のシーンをライティングするのにピッタリだ。

この光源の特徴として、ライトを回転させることで、光源の向きを調整することができる。また、Intensityで光源の強さを、Colorで色を変えることができる。今回は、夜のシーンを表現したいので、青めのライト設定にした。最後に、ライトのインスペクタからModeをBakedに、Shadow TypeをSoft Shadowにしておこう。こうすることで、後でライトマップを生成したときにライトにあたったオブジェクトから影が落ちる設定になった。これ以外にも自由に任意のライトを配置してもよいが、次の点に注意しよう。かならずベイクライトにしておこう。



図8.7: 色や方向を変え、シーンをライティングしてみよう

そして、ライトもまた専用の空トランスフォームの下階層でまとめれば、管理が楽になる。

ライトプローブの作成

次は、non-staticなゲームオブジェクトの環境光を設定するために、ライトプローブを設置してみよう。ヒエラルキーパネル上で、Create > Light > Light Probe Groupと進み、ライトプローブを作成する。ライトプローブを選択したら、インスペクタ上からEdit Light Probeを実行しよう。

シーンビュー黄色いスフィアで表示されているのがライトプローブで、このライトプローブに囲まれたエリアの環境光をサンプリングし、それをゲームオブジェクトに照射する。なので、このライトプローブはできるだけnon-staticなオブジェクトが存在する可能性のある空間をカバーするように配置するのが望ましい。スフィアを選択し、位置を調整してみよう。マップが複雑な場合は、ライトプローブの数を増やしてライトプローブグループの形状を変えることもできるが、ライトプローブの数が増えれば処理負荷も上がるため、注意が必要だ。

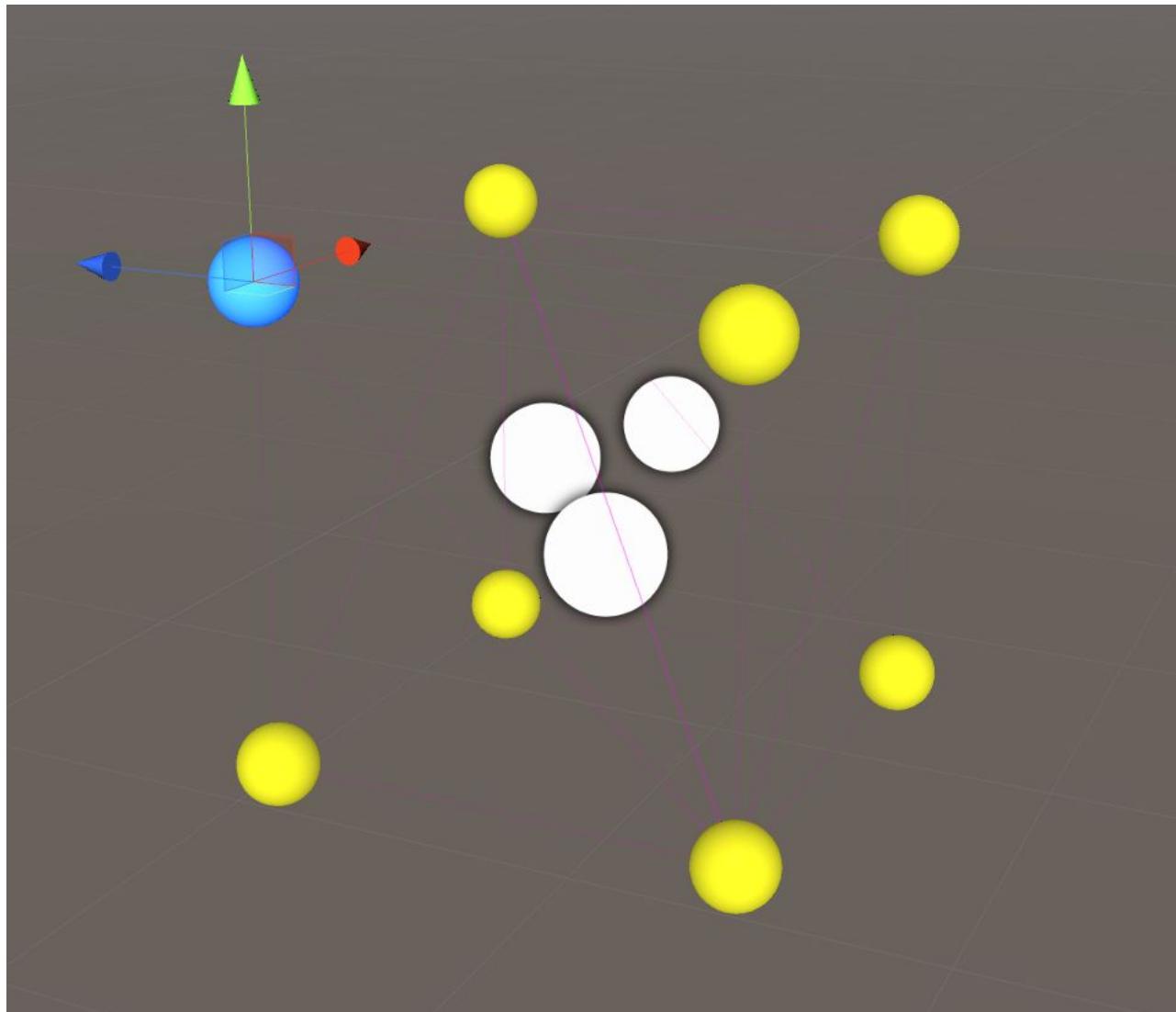


図8.8: 黄色のスフィアを移動することで、ライトプローブの空間作る

今回はモバイルでビルドすることに加えて、ライトプローブグループが1つでもシーンに存在すれば自然な環境光を作り出してくれるため、今回は最小の8個のライトプローブでライトプローブグループを作成した。

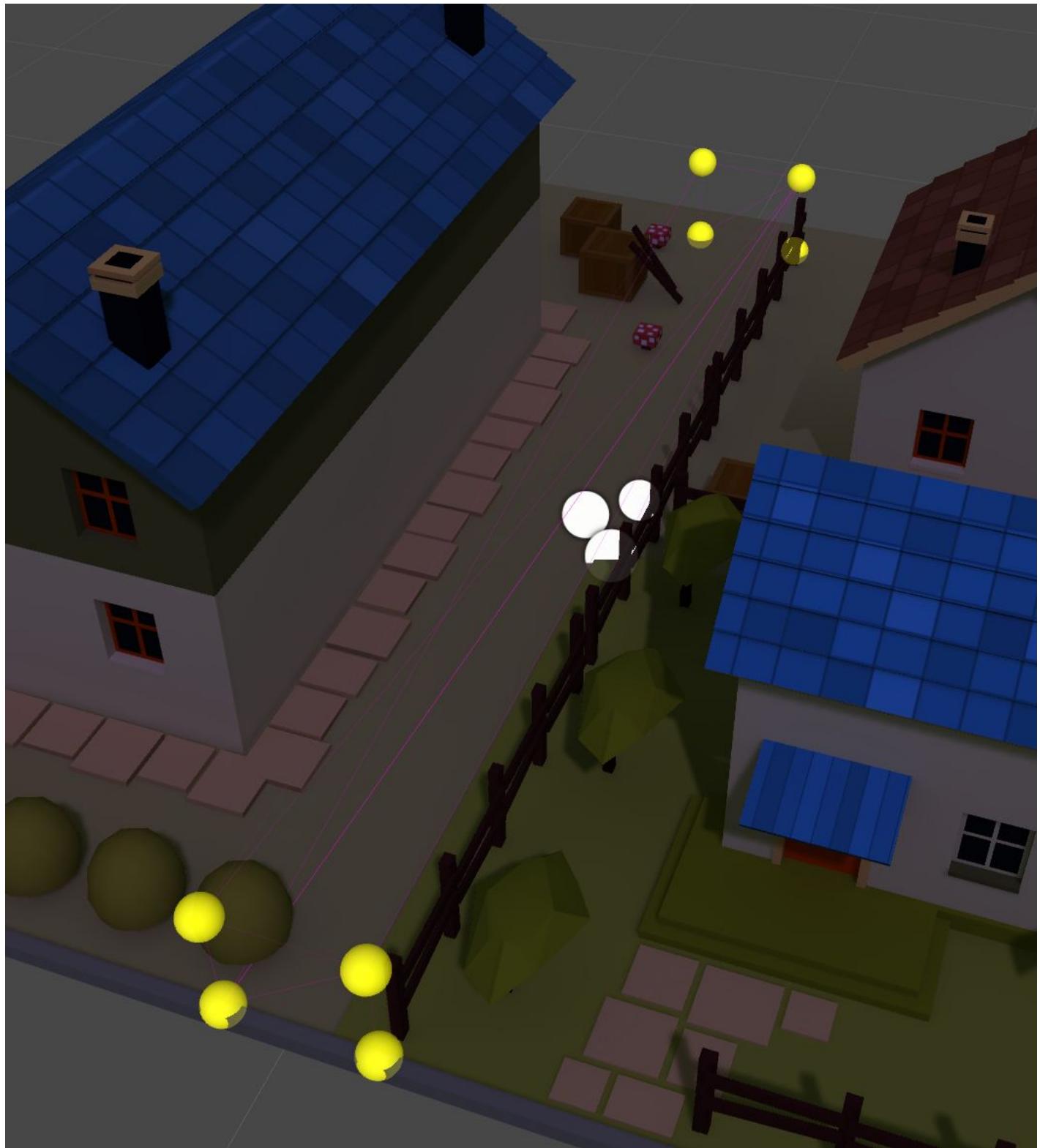


図8.9: 家と家の間の路地にライトプローブ空間ができるように設置した

ライトプローブからの環境光を実際に確かめるためには、この後のライトマップの作成が終わるまで待つ必要がある。

ライトマップの作成

本解説ではUnity 2017以降を対象としている。それ以前のバージョンでは、操作方法が異なるので注意。

Window > Lighting > Settingsへ進み、ライティングの設定画面が開く。次の点に注意して設定してみよう。

- Skybox Material
 - Skybox はCG空間で最も奥で描画され、文字通り空などを表現するときに使用する。ただし、Default-Skyboxは描画コストがかかるため、モバイル向けの場合は None にしておくほうが無難だ。
- Environment Lighting
 - ここでは環境光の設定を行う。ここで設定はライトマップ・ライトプローブに影響を与える
 - Source は Color にし、Ambient Color で環境光の色を指定する
 - Ambient Mode は Baked にしよう。
- Realtime Lighting
 - Realtime Global Illumination のチェックをはずす。(リアルタイムGIは使用しないため)
- Mixed Lighting
 - Lighting Mode は Baked Indirect にしておく。
- Lightmapping Settings
 - Directional Mode は Non-Directional にしておく。
- Lightmap Parameters から、ライトマップの品質に関するプリセットが選べる。ライトマップの作成に時間がかかるようなら、Default-VeryLowResolutionを選ぶ。

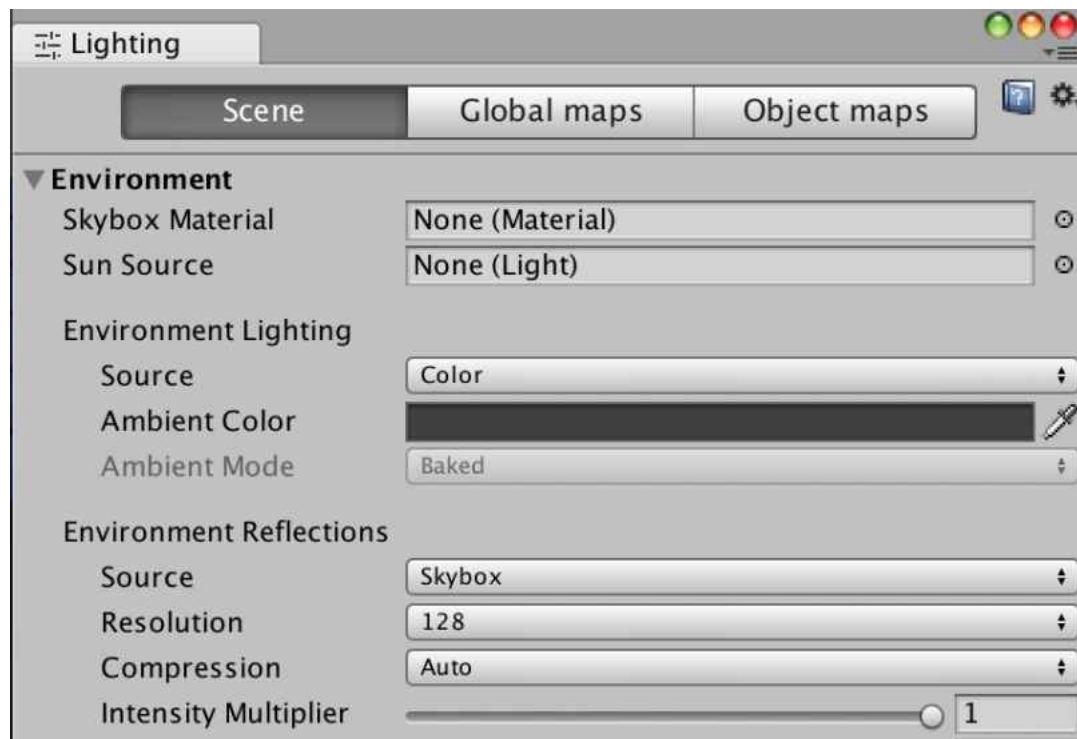




図8.10: ライトマップ生成の設定画面

最後に、ウインドウの最下部の Auto Generate のチェックを外し、隣の Generate Lighting ボタンを押すと、ライトマップの生成が始ま。Unity Editor の右下のプログレスバーがなくなれば、完了だ。

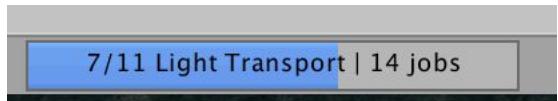


図8.11: ライトマップ生成には時間がかかる

シーンの確認と調整

ライトマップが作成が終わったら、シーン全体を見渡してみて、staticなオブジェクトにライトマップが適応されているかどうか（影などがテクスチャとして焼きこまれているのがわかるはずだ）、non-staticなオブジェクトに直接光と間接光が当たっているか、確認してみよう。



図8.12: 影がテクスチャ(ライトマップ)として焼きこまれている

最後に、Lighting Settings パネルの下部にある Fog の設定をしてみよう。Fog は霧がかかったように奥に行くほど霞んでみえる表現を作ることができる。フォグの色や深度などを設定し、より霧囲気を高めてみよう。



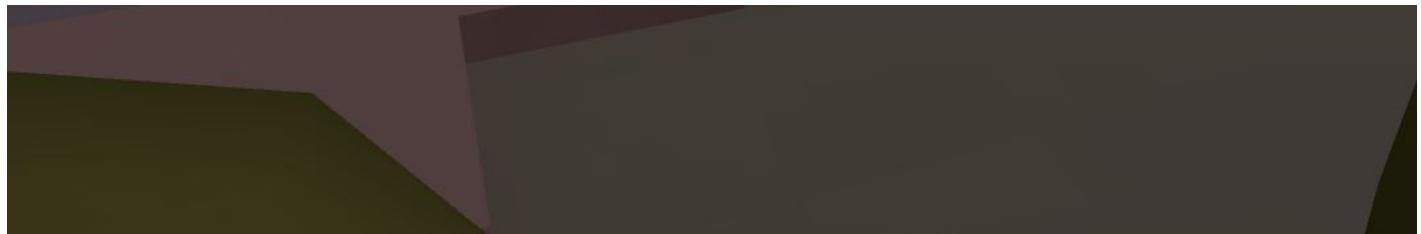


図8.13: フォグの効果で雰囲気がだいぶ変わる

8.3 ビルドして遊んでみよう

ライティングの設定が一段落したら、今まで作ったクラスやキャラクターをシーンに設定しなおし、Unity Editor 上で動作を確認したら、端末にもビルドして遊んでみよう。

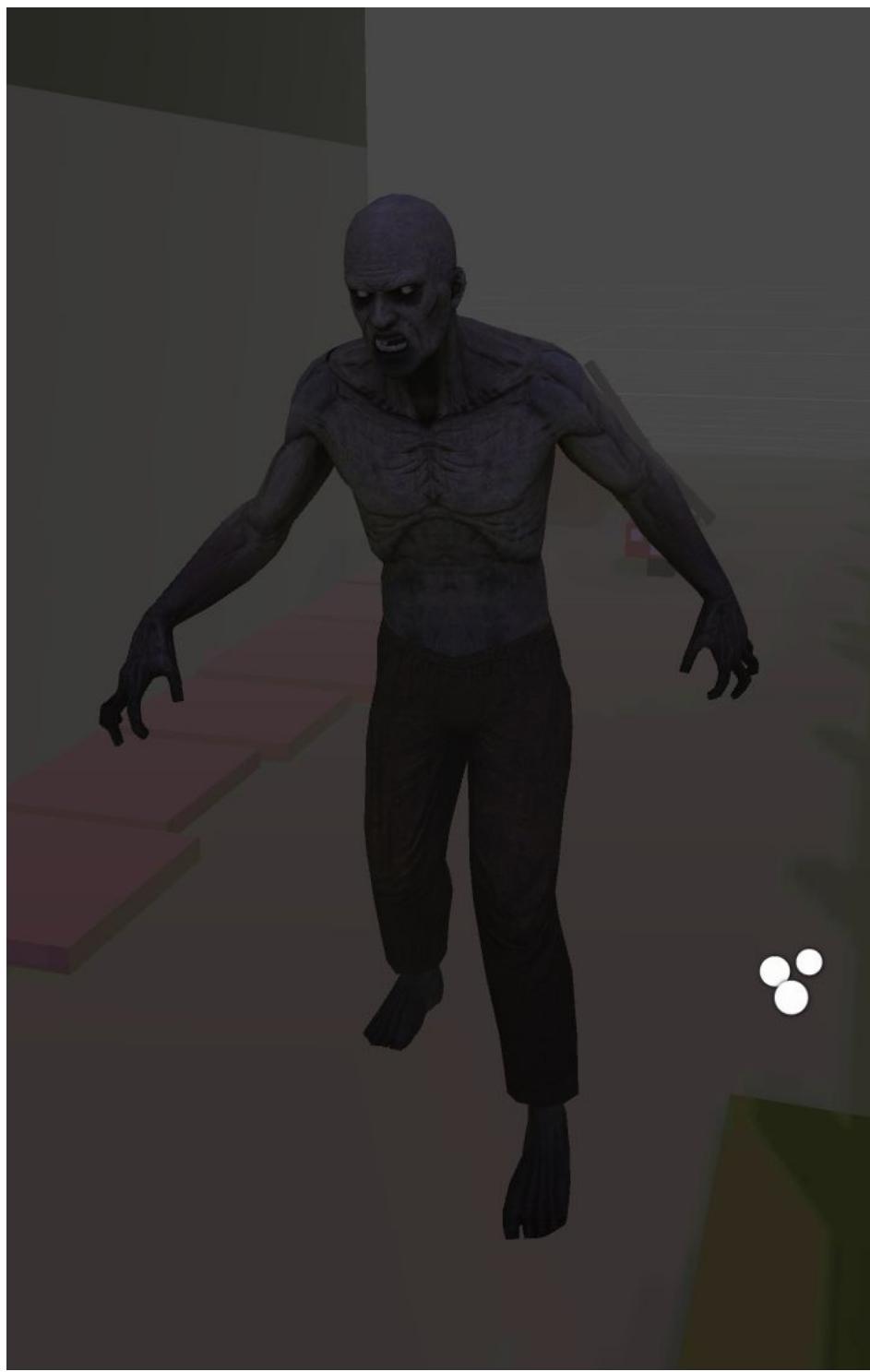


図8.14: 前回制作したエネミーのキャラクターを設置したところ。ライトプローブによって、直接光の紫色と周囲の間接光がかかっているのがわかる

今回の解説をもって、モバイルVR開発入門はすべて完了となる。開発環境のセットアップから、Unityそのものの仕組み、C#を使ったゲームロジックの作成、グラフィックス設定の方法を紹介してきたが、実はこれでほとんどのVR開発に必要な基礎的な知識をカバーしたことになる。

なる。もちろん、この他にもさまざまな機能があるが、まずは最低限ゲームとして成り立つプログラムを作った後(ミニマルデザイン)、さらに高度な表現を実現するための機能を自分で調べていくとよいだろう。

著者紹介

酒井 駿介

グリー株式会社

アプリ制作会社などでモバイルアプリの開発業務を経て、2015年よりグリー株式会社所属。Technical Artist チームにて、3D アートアセットパイプラインの構築や、シェーダ開発、処理負荷の最適化などにあたっている。Unity Certified Developer(2016)

スタッフ

- 田中 佑佳(表紙デザイン)
- 伊藤 隆司(Web連載編集)

奥付

本書のご感想をぜひお寄せください

<https://book.impress.co.jp/books/1117101050>

「アンケートに答える」をクリックしてアンケートにぜひご協力ください。はじめての方は「CLUB Impress(クラブインプレス)」にご登録いただく必要があります(無料)。アンケート回答者の中から、抽選で商品券(1万円分)や図書カード(1,000円分)などを毎月プレゼント。当選は賞品の発送をもって代えさせていただきます。

●本書の内容に関するご質問は、書名・ISBN・お名前・電話番号と、該当するページや具体的な質問内容、お使いの動作環境などを明記のうえ、インプレスカスタマーセンターまでメールまたは封書にてお問い合わせください。電話やFAX等でのご質問には対応しておりません。なお、本書の範囲を超える質問に関してはお答えできませんのでご了承ください。

●落丁・乱丁本はお手数ですがインプレスカスタマーセンターまでお送りください。送料弊社負担にてお取り替えさせていただきます。但し、古書店で購入されたものについてはお取り替えできません。

■読者の窓口
インプレスカスタマーセンター
〒101-0051 東京都千代田区神田神保町一丁目105番地
TEL 03-6837-5016 / FAX 03-6837-5023
info@impress.co.jp

■書店／販売店のご注文窓口
株式会社インプレス 受注センター
TEL 048-449-8040
FAX 048-449-8041

VRを気軽に体験 モバイルVRコンテンツを作ろう！
(Think IT Books)

2017年9月1日 初版発行

著 者 酒井 駿介
発行人 土田 米一
編集人 高橋 隆志
発行所 株式会社インプレス
〒101-0051 東京都千代田区神田神保町一丁目105番地
TEL 03-6837-4635(出版営業統括部)
ホームページ <https://book.impress.co.jp/>

本書は著作権法上の保護を受けています。本書の一部あるいは全部について(ソフトウェア及びプログラムを含む)、株式会社インプレスから文書による許諾を得ずに、いかなる方

法においても無断で複写、複製することは禁じられています。

Copyright © 2017 Shunsuke Sakai. All rights reserved.

印刷所 京葉流通倉庫株式会社

ISBN978-4-295-00234-5 C3055

Printed in Japan