

中高生向け

Lab7

Unity 入門

【各種ジャンルのサンプルゲームを作成するシリーズ】

Unity を使って ゲーム作成にチャレンジ

Vol. 1

第1章.

Unity の基本を知って

簡単なサンプルゲームを作ってみよう

【小学生高学年・中学生・高校生向け】

毎章新たなジャンルのゲームを作り、
それを基に自分オリジナルのゲームを作ろう！
目標はコンテスト出場やゲームリリースだ！！

志知 淳一 著






はじめに

【中高生向け Unity 入門シリーズについて】

この「中高生向け Unity 入門 Unity を使ってゲーム作成にチャレンジ」シリーズはプログラミング初心者である中高生（小学生高学年から高校生）を対象にしたプログラミングのテキストです。



中高生向けと言っても小学高学年から高校生まで、

本格ゲームを作りたいプログラミング初心者が対象だよ。


このシリーズでは皆さんの大好きなゲーム作成を通して今話題のプログラミングを学びます。

具体的には、Unity というプロのゲーム会社でも使われているゲームエンジンを使用します。いろいろなジャンルの本格的なサンプルゲームを作りながら Unity を用いたゲームプログラミングを学びます。

毎回人気のあるジャンルのサンプルゲームを作成して、その後にそこで使われている技術に関して勉強をします。

自分の好きなゲームジャンルであれば、これを改造して自分のオリジナルゲームを作っても構いません。

そして最終的には自分自身のオリジナルゲームを作って、友達や家族だけでなく、多くの人達に遊んでもらうことを目標にしています。



最終的には自分のオリジナルゲームを作り、多くの人達に遊んでもらおう。

これによりプログラミングスキルやそれに伴う論理的思考、問題解決力が得られるだけでなく、他では得られない成功体験が得られるでしょう。

【本テキスト（第1章）について】

本テキストは「中高生向け Unity 入門 Unity を使ってゲーム作成にチャレンジ」シリーズの Vol.1 として、第1章が掲載されています。

この第1章では「Unity の基本を知って簡単なサンプルゲームを作ってみよう」として、簡単なサンプルゲーム作成を通して、そこで使用している Unity の基本部分を学習します。

第1章は簡単なサンプルゲーム作成を通して、そこで使用している Unity の基本部分を学習するよ。



また、最初であるので、プログラミング初心者向けに「プログラミングとは」や「Unity とは」という内容に加え、Unity の学習法と言った学習を進める前に知っておくと良い情報や目標管理法なども紹介しています。

次の章（テキスト）の第2章からは各章で異なったジャンルのゲーム作成法を紹介していきます。前にも記載した通り、最終目標は自分自身のオリジナルゲームを作って、多くの人達に遊んでもらうことです。ぜひ自分のゲーム作りにチャレンジして下さい。

（サンプルゲームの画像を変えて、少し機能や画面を変えるだけで自分のゲームを作っても良いですし、いくつかの章のゲーム機能を統合して新たな自分自身のアイデアのゲームを作っても面白いです。）

【次章以降の内容】

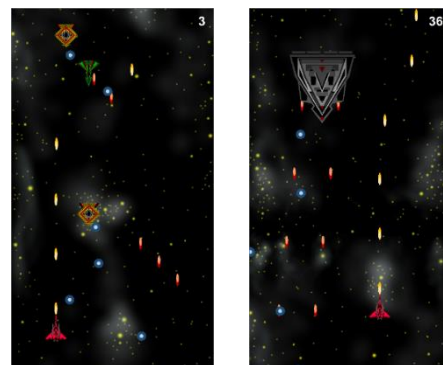
第2章.

Unity で「2D 縦スクロール アクションゲーム」
を作ってみよう



第 3 章.

Unity で「2D 縦スクロール シューティング」を作ってみよう



第 4 章.

Unity で「2D スライドパズルゲーム」を作ってみよう



(これ以降も多くのゲームジャンルが続きます。詳しくは本書最後の「第 2 章の予告」をご覧ください。)

【2D ゲームの作成】

本シリーズでは基本的に自分自身のゲーム作成が比較的容易な 2D ゲームを取り上げます。3D ゲームを作ってみたい人も多いかと思います。しかし、このテキストでは皆さんが最短で自分のオリジナルゲームを作り上げることを目標にし 2D ゲームを説明します。

(3D モデルの用意は難しい場合があります。2D 画像は容易に集められ、自分で描くこともできます。)

実際 Unity は 3D ゲームを開発するためのゲームエンジンなので、3D 作成を得意としています。2D と 3D の作成方法は基本的に同じです。2D ゲームができれば 3D ゲームの作成も可能なので安心して下さい。

【キャラクター紹介】

テキスト内で説明を担当してくれるのがホワイトライオンくん（本名：マー〇〇〇〇）とお友達のマシーンライオンくん達です。

ホワイトライオンくんにはお父さんもいるのでよろしくね。



Contents

はじめに.....	1
【中高生向け Unity 入門シリーズについて】	1
【本テキスト（第 1 章）について】	2
【キャラクター紹介】	4
第 1 章. Unity の基本を知って簡単なサンプルゲームを作ってみよう	10
第 1.1 章. ゲームプログラミングと Unity.....	11
1.1.1. プログラミングとは.....	12
【プログラム、プログラミングとは】	12
【プログラミングって難しいの?】	12
【プログラミング言語】	14
【参考】【C#プログラミング言語とは】	15
1.1.2. Unity とは.....	16
【ゲーム作りに必要なもの】	16
【Unity とは】	17
1.1.3. Unity の学習法.....	18
【Unity の学習法】	18
① とりあえずテキストを見ながら簡単なゲームを作ってみる	18
② テキストを見ながら他のジャンルのゲームをたくさん作ってみる	19
③ 理解した技術を組み合わせて自分のゲームを作成してみる	19
④ 人に遊んでもらい感想を聞きながら自分のゲームをレベルアップ	20
【学習の目標管理】	21
① 自分のゲームを作り上げてみる	21
② プログラムコンテストに参加してみる	21
③ より多くの人に遊んでもらおう	22
【子供向けプログラミングコンテスト】	23

①	Unity インターハイ	23
②	全国小中学生プログラミング大会	23
③	U-22 プログラミングコンテスト	24
④	アプリ甲子園	24
⑤	TECH KIDS GRAND PRIX	25
⑥	日本ゲーム大賞 U18 部門	25
	【多くの人に遊んでもらう】	26
①	Unity Room	26
②	Google Play や App Store	27
③	Steam や Microsoft Store	27
第 1.2 章.	Unity を使ってみよう	28
1.2.1.	Unity のインストール	29
	【インストール作業概要】	29
	【インストール方法】	30
①	Unity Hub インストーラーのダウンロード	30
②	Unity Hub のインストール	31
	【参考】 【Unity Hub とは】	32
③	Unity ID の新規登録とサインイン	33
④	新規ライセンスの認証	35
	【参考】 【Unity のライセンスは無料なの?】	36
⑤	Unity のインストール	37
	【参考】 【他のモジュールの追加】	40
1.2.2.	プロジェクトとシーンの作成	41
	【プロジェクトとは】	41
	【シーンとは】	41
	【プロジェクトとシーンの作成方法】	42

1.2.3. Unity エディタの画面構成と操作法.....	44
【Unity エディタの画面構成】	44
【参考】 【Unity エディタのビュー配置のデフォルト】	45
【Unity エディタの基本操作】	46
【参考】 【Unity エディタでの 2D と 3D の違い】	47
【参考】 【ゲーム実行時の設定変更に注意】	48
【Scene ビューでの視点操作】	49
① 視点のズームイン・ズームアウト	49
② 視点の平行移動.....	49
【Scene ビューでのオブジェクトの変形】	50
① オブジェクトの移動.....	50
② オブジェクトの回転.....	50
③ オブジェクトの拡大・縮小.....	51
【参考】 【ゲーム実行画面のサイズ変更】	51
第 1.3 章. 簡単サンプルゲームを作ってみよう	52
1.3.1. Unity でのゲーム作成の手順.....	53
1.3.2. プロジェクト単位・シーン単位の作業.....	54
【プロジェクト単位の作業】	54
① プロジェクトの新規作成.....	54
② Unity へ素材の取り込み.....	56
③ プラットフォームの設定.....	57
【シーン単位の作業】	58
④ シーンの新規作成.....	58
1.3.3. オブジェクト単位の作業.....	60
【オブジェクト単位の作業】	60
⑤ シーンへオブジェクトを配置.....	60

⑥	オブジェクトのコンポーネントを設定	61
⑦	スクリプトの作成とプログラミング	61
	【参考】【Visual Studio が起動しない場合】	63
	【参考】【プログラムに書き間違いがある場合】	64
	【参考】【プログラムの詳細説明】	65
⑧	スクリプトをオブジェクトに関連付け	66
⑨	ゲーム実行をしてテスト	67
1.3.4.	他のオブジェクトへの作業	68
	【他のオブジェクトへの作業】	68
①	右側のじゃんけんルーレットの設定	68
	【参考】【プログラムの詳細説明】	70
②	停止位置の針の設定	73
	【参考】【Sprite Render の Order in Layer】	74
	【参考】【プログラムの詳細説明】	76
	【練習問題】	77
	【回答】	78
第 1.4 章.	簡単サンプルゲームで使った C#	79
1.4.1.	C#プログラムの基本	80
	【C#プログラムを読むうえでの基本】	80
	【変数とは】	82
	【変数を使ってみよう】	83
	【関数（メソッド）とは】	86
	【実際の関数を見てみよう】	87
①	Unity の標準関数	87
②	自分で作成する関数（引数・戻り値がある場合）	88
③	自分で作成する関数（引数・戻り値がない場合）	90

1.4.2. Start 関数と Update 関数	91
【Start 関数と Update 関数とは】	91
【フレームと FPS とは】	92
1.4.3. 制御文 : If 文	93
【制御文とは】	93
【「if」制御文について】	94
【「if-else」と「else if」制御文について】	97
① if-else 制御文	97
② else if 制御文	98
第 2 章 の 予 告	100
作 者 紹 介	102

第1章. Unity の基本を知って簡単なサンプルゲームを作ってみよう

第 1 章.

Unity の基本を知って 簡単なサンプルゲームを 作ってみよう






第1.1章. ゲームプログラミングと Unity

この章ではプログラミング初心者に向けて、まずそもそもプログラムとは何？というところから説明しています。

そして、皆さんが大好きな本格的なゲームを自分で作成するにはどうしたら良いかという説明で、Unity というゲームエンジンの使用を紹介します。

その後、この Unity をどうやって学習していくべきかを解説し、同時にゲーム作りの目標としてプログラミング大会への参加や、自分のゲームのリリースと言う手段を提案しています。

(お薦めのプログラミング大会をたくさん紹介していますので、ぜひ Unity 学習の1つの目標としてトライしてみてください。)



プログラミングを全くやったことなくても大丈夫。

Unity を使った本格的なゲーム作成をチャレンジしてみよう。

1.1.1. プログラミングとは

【プログラム、プログラミングとは】

皆さんは日々パソコンやスマートフォン、タブレットを用いて、いろいろな便利なソフトウェアや楽しいアプリを使っているかと思います。また、NintendoやPlayStationなんかのゲームコンソール機で遊んでる人達も多いかもしれません。

皆さんの大好きなゲームや Youtube、LINE。これらは全てプログラムで動いています。また、現在は車やエアコン、冷蔵庫にテレビなんかもプログラムが組み込みこまれています。

このプログラムにはコンピュータやスマートフォン、機械や電化製品への指示が書かれており、それに従ってソフトウェアやアプリ、機械等が動くわけです。

そのプログラムを書くことを一般的にプログラミングと言います。

みんなが使っているパソコンやスマートフォン、ゲーム機や電化製品。それらは動きの指示が書かれたプログラムに従って動いているんだ。



【プログラミングって難しいの?】

はい。小学校低学年の子達がプログラムを体験するのに用いられるビジュアルプログラミング言語に比べると難しいです。

ビジュアルプログラミングはとても簡単で直感的に理解できるため、小さな子達にはもってこ

いです。しかし、できることは限られていて、完成したものを見ても本格的なゲームアプリに遊び慣れているの方々には、小さな子のおもちゃに見えてしまうかもしれません。



では、皆さんが今から勉強するプログラムは何でしょう。それは一般的にテキストベースのプログラミング言語と言われているものです。では本テキストで皆さんが最初に作るプログラムを見てみましょう。

一見意味不明なおまじないにも見えます。。。これを英語の勉強のようにがんばって勉強したり練習する必要があるのでしょうか。

そんなことはありません。実は重要な部分はあまり難しくなく、覚えることもそんなに多くはありません。プログラミングは英語の勉強をするよりも簡単です。

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class RouletteController : MonoBehaviour
6 {
7     public int roulette_No;
8     private float rotaionAngle = 0;
9
10 void Update()
11 {
12     if (roulette_No == 1 && Input.GetKey(KeyCode.Space))
13     {
14         transform.Rotate(0, 0, Random.Range(-180, 180));
15         rotaionAngle = 20;
16     }
17
18     transform.Rotate(0, 0, rotaionAngle);
19
20     rotaionAngle = rotaionAngle * 0.999f;
21
22     if (rotaionAngle < 0.01f) rotaionAngle = 0;
23 }
24 }
```

テキストベースのプログラミング言語

人によっては難しく感じる部分もあるかもしれません。そんな場合は、重要なところから順番に理解していけば良いです。ほんとうにおまじないのような部分もあり、大人が聞いてもすぐには理解できないこともあったりします。（ビジュアルプログラミングでは、おまじないのような箇所は見えないようになっています。）

ビジュアルプログラミングに比べて難しくても全く恐れる必要はありませんよ。すぐできるようになるし、ビジュアルプログラミング以上にいろいろ本格的なゲームが作れるようになります。

ビジュアルプログラミングに比べると難しい
かもしれない。

だけど本当に難しい部分は単なる「おまじない」
だと思って学習を進めてみよう。全部を
理解する必要なんてないんだ。

するとどんどん理解できるところが増えて、
本格的なゲームが作れるようになるよ。



【プログラミング言語】

このプログラムを勉強すると言うのは、プログラミング言語を勉強することになります。そして、普通の言葉に日本語、英語、中国語といろいろあるように、プログラミング言語にも数多くの種類があります。

以下に最近良く使われているプログラミング言語をあげておきます。

(世の中には以下のリストにあがっていないプログラミング言語もたくさんあります。)

【プログラム言語の種類】

プログラミング言語	説明
C# (シー シャープ)	マイクロソフトが C++, Java を参考に作った言語。多くのアプリが開発でき、とても人気がある。Unity で使われているため、ゲーム業界では多く利用されている。
Java (ジャバ)	大型の業務システムからスマホアプリまで汎用的な分野で利用され多くの利用者がいる。この言語から学習する初心者も多い。Java を使った仕事も多い。
Python (パイソン)	データの解析によく使われる言語。難易度は高くなく初心者向け。最近では、AI（人工知能）での機械学習で使われることが多く、とても人気のある言語。
C++ (シー プラス プラス)	古くから使われ、他の言語の元になっている。比較的大規模なシステム、処理速度が重要なシステムで現在も多く使用されている。
Ruby (ルビー)	日本のまつもとゆきひろさんが開発し、世界的に人気のある言語。読みやすく教育にも使われている。日本で開発されたプログラミング言語として初めて国際規格となった。
JavaScript (ジャバ スクリプト)	Webサイトに動きや機能を付ける際に最も使われているブラウザを操作するためのプログラム言語です。習得はそれほど難しくはない。

次の章で解説しますが、Unityで使用されているプログラミング言語は「C#」（シー シャープ）です。

このC#の説明はこれからじっくりしますが、他の言語に関して今は特に意識する必要はありません。ざっくり、こんな言語があるということだけで十分です。

なぜならば、プログラミング言語と言うのは、基本的な作りはどれもとても似ています。1つのプログラミング言語を覚えてしまえば、他の言語はちょっと違いをチェックすればだいたい使えます。あまり僕は何言語を勉強しているんだと意識する必要はありません。



【参考】【C#プログラミング言語とは】

【C#プログラミング言語とは】

プログラミング言語には数多くの種類があると先ほど説明しました。では、このC#とはどのようなものなのでしょうか。

C#はもともとはマイクロソフトが開発したプログラム言語です。既に10年以上の歴史があり何度もバージョンアップされています。言語的には十分に成熟していますが、マイクロソフトが積極的に開発を続けているので、できることがどんどん増えています。

C#はC++言語やJava言語などを参考にして作られています。そのため、これらの文法はかなり似たものとなっています。

【作れるアプリの種類の比較】

C#	Java	C++	Java Script
パソコンアプリ スマホアプリ ウェブアプリ	パソコンアプリ スマホアプリ ウェブアプリ	パソコンアプリ	ウェブアプリ

【勉強の難易度の比較】

C#	Java	C++	Java Script
やや難しい	難しい	かなり難しい	比較的簡単

1.1.2. Unity とは

【ゲーム作りに必要なもの】

では、ゲームアプリを作ってみましょう。。。だけど、どうやって？

皆さんのよく遊んでいるゲームアプリはきれいなグラフィックのキャラが思い通りに動き周り、すごい迫力のエフェクトやサウンドを発しながら、迫りくる多くの敵をなぎ倒しているかもしれません。



そのようなゲームを作る場合、多くのプログラマーやアーティスト、デザイナーが手分けをして作成しています。でもそんな大規模でなくて良いから自分で作ってみたい。。。

それでもゲームを作るにはゲーム企画、ゲーム機能のプログラミング、3Dモデル・エフェクト・サウンド等アセット（部品）の統合、アニメーション、光・重力・衝突等の物理演算…といろいろ勉強することがあります。

しかし、これら全部勉強しないとゲームは作れないのでしょうか？そんなことはありません。ゲーム開発用のソフトウェア「ゲームエンジン」があります。これがあれば、基本的に自分でやる部分は面白いゲームの企画とゲーム機能の設定やプログラミングだけとなります。

Unity というゲームエンジンを使えば自分でも本格的なゲームが作れてしまうよ。



このテキストでは「Unity」と言うゲームエンジンを使用して、プログラミングの部分に焦点を当ててゲーム作りを体験していきます。

【Unity とは】

このテキストでは「Unity」と言うゲームエンジンを使いますが、Unity とはどんなものなのでしょう。



Unity とは、ユニティ・テクノロジーと言う会社が開発したゲームの作成ツールのことです。ゲーム作成に必要なグラフィックス、サウンド、数学・物理演算、エフェクト、アニメーション、3DCG… なんかを簡単な設定でゲームに組み込みます。

Unity を使えば、皆さんは どんなゲームを作ろうかと考えるゲーム企画、あと、ゲームの動きをゲームに指示する設定とプログラミングに集中できます。

しかも、Unity だとこんな便利なゲームエンジンが無料で使用できるのです。

【Unityの良いところ】

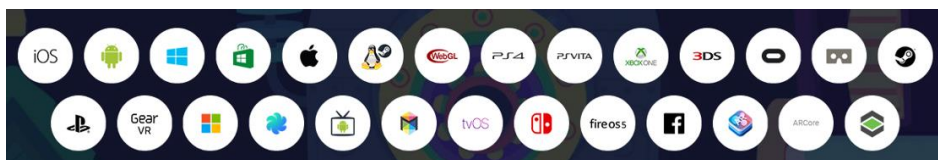
- プロも使う本格ゲームエンジン。
多くの人気ゲームがUnityで作られている。
- 個人で使用するなら無料。
作成したゲームをリリースしても無料。
- パソコン、スマホはもちろん、ゲーム機等
多くのプラットフォームに対応している。

無料だとそんなに大したものを作れないのかなと思いますか？いや、そんなことはありません。本格的なゲームが作成できます。人気ゲームのポケモン GO やマリオカートなんかも作られているプロ用でもあります。



また、対象のプラットフォームもとても多く、パソコン、スマホ、タブレットはもちろん、PlayStation、Xbox、Nintendo Switch なんかのゲーム機、今流行りのVR(Virtual Reality)ゲームも作れてしまいます。

【Unity の対象プラットフォーム】



(Unity の HP より)

1.1.3. Unity の学習法

ここでは、Unity でのゲーム作成技術を効率的に上達するための学習法を解説します。そして学習を進める上で重要となる目標管理も。後半にはこの目標管理のなかで出てくるプログラミングコンテストに関して、いくつかお薦めなものを紹介しています。

【Unity の学習法】

Unity を学習する方法として、市販の本を買う、YouTube の動画を見る、プログラム教室に通う等、多くの学習法の中で何をすると効果的かと言う議論があります。しかし、この文章を用んでいる方々はこのテキストをここまで既に読んでいるでしょうから、ここではこのテキストを用いた効果的な学習法をお話します。(もちろんこのテキストが合わない場合は、このテキストの部分のを他の市販本等に読みかえて下さい。)

① とりあえずテキストを見ながら簡単なゲームを作ってみる

Unity にはとても多くの機能が備わっています。まずゲームを作る前に Unity の機能を全部知らなくてはと考える方もいるかと思いますが、そんな必要はありません。

そして、Unity のプログラミングでは C# プログラミング言語を使います。これも最初に言語の勉強をしなければと考える必要は全くありません。

実際、Unity と C# の勉強にはとても時間がかかります。そして細部まで理解するのは大人でもとてもたいへんです。ですので、全部理解できなくても、とりあえずテキストに沿って簡単なゲームを作ってみましょう。

細かい部分は単なるおまじないだと思って下さい。いったんは全然分からなくても問題ないです。

まずはテキストを見ながら簡単なゲームを作ってみよう。

難しい部分は単なるおまじないと思って飛ばしてしまおう。



そして簡単なゲームを作ってみたら、Unity の機能と C# プログラムに関してそのゲームの重要な部分だけ理解してみましょ。ここでも細かい部分や、まだ難しいと思う部分は飛ばしても構いません。分かったと思える部分が少しでもあれば OK です。

② テキストを見ながら他のジャンルのゲームをたくさん作ってみる

とりあえずいろいろなジャンルのゲームを、テキストを見ながらたくさん作ってみましょ。この際も考え方は前にやった方法と同じです。全部理解できなくても、とりあえずテキストを見ながらゲームを作ってみる。

細かい部分は単なるおまじないだと思って下さい。

そして、ここでも Unity の機能と C# プログラムに関して、そのゲームの重要な部分だけ理解してみましょ。

細かい部分や、まだ難しいと思う部分は飛ばして、分かったと思える部分が少しでも増えれば OK です。

テキストを見ながら他のジャンルのゲームも作ってみよう。

たまに以前作ったプログラムを読み返し、理解できる部分を増やして行こう。



その繰り返しです。そして、たまに前に作ったゲームのテキストやプログラムを再度読んでみましょ。他のジャンルのゲームでも同じような部分があることに気づきます。そして前に理解できなかった部分がどんどん理解できるようになって行くでしょう。

③ 理解した技術を組み合わせて自分のゲームを作成してみる

テキストを見ながら各種ジャンルのゲームを作り終えたら、次は自分のアイデアのゲーム作成にトライしてみましょ。

この時点では自分のオリジナルアイデアのゲームを作ると言うより、できる・理解した技術を組み合わせて自分のゲームを作ってみよう。

最初はテキストに沿って作って見たサンプルゲームで理解した部分をつなげたり、改造して新たな自分のアイデアのゲームを作ってみる。

まだアイデアが無いのなら、テキストのゲームのパラメータを変えたり、イラストを変えるだけでも良いです。

理解した技術を組み合わせ
合わせて自分のゲーム
作成にトライ。



自分でサンプルゲームを改造したり、理解した部分をつなぎ合わせたりすることで、理解できていると思っていただけの部分が本当の理解に変わります。

④ 人に遊んでもらい感想を聞きながら自分のゲームをレベルアップ

前の③で作った簡単な自分のゲームに、理解が増えた技術を追加して自分のゲームをどんどんレベルアップしてみましょう。

ここで、なるべく早いタイミングで家族や友達に遊んでもらい感想を聞いてみよう。人に作成したゲームを見せると、多くの気づきがあるでしょう。

また、多くの感想をもらうことは、もっと面白いゲームを作ろうと言うやる気にもつながります。

友達や家族に遊んでもらい感想を聞き、自分のゲームのレベルアップをしよう。

作り方が分からない場合は、テキストを読み返したり、新たなテキストのサンプルゲームを作ってみよう。



そして、また新たな機能を追加したり、新たなアイデアのゲームを作ってみましょう。作る方法が分からない時は以前のテキストやプログラムを読み返したり、また新たなテキストで新たなジャンルのサンプルゲームを作ってみるのも良いです。

この繰り返しで理解できる部分を少しずつでも良いので増やして行きましょう。

【学習の目標管理】

プログラム学習に限らず、勉強も運動も目標を持って取り組むのは上達にとっても効果的です。ここで言う目標とは大きな最終目標と言うことではありません。（大きな目標も重要ですが。）もっと早期に実現可能な言わば小さな目標です。

① 自分のゲームを作り上げてみる

まず最初の目標として、前に解説した Unity の学習法にもありましたが、テキストのサンプルプログラムをたくさん作ってみる。そして、自分のアイデアの簡単なゲームを作り上げてみるというのはどうでしょう。

ここでは簡単でも良いのでゲームを作り上げることが重要です。作り上げることによって、自分の理解度や実力も飛躍的に向上しますし、次に進むモチベーションにもつながります。

簡単なゲームを1つ作り上げたら人に遊んでもらい感想をもらってみよう。そして、またもう少し難しく自分のアイデアの詰まったゲームを作りあげてみましょう。

② プログラムコンテストに参加してみる

では、それができたら次の目標は何にしたら良いのでしょうか？例えば数多くある子供向けのプログラミングコンテストに応募してみると言うのはどうでしょう。実は子供向け（小学生・中学生・高校生向け）のコンテストはいっぱいあります。このようなコンテストでの入賞なんかを目標にするのもやる気が出てきますよね。

参加することにより自分のゲームの客観的な評価を得ることができ、そして同じ目標の友達にも知り合うことができます。

後でいくつかのお薦めのプログラミングコンテストを紹介しています。ぜひトライをしてみてください。

③ より多くの人に遊んでもらおう

上記のプログラミングコンテストの後でも同時にでも、多くの人に遊んでもらうことを目標にしても良いでしょう。

それにより良い感想だけでなく、悪い感想も受けることもあります。しかし、そんなことを気にする必要はありません。知らない人が自分のゲームを遊んでくれたことだけで嬉しいことです。悪い感想も自分のゲームや技術をレベルアップするのに素晴らしい情報です。ぜひ勇気をもって多くの人に遊んでもらう手段もトライしよう。

多くの人に Unity で作成したゲームを遊んでもらう手段としては「Unity Room」というサイトがあります。WebGL のアプリをインストールなしで遊ぶことができるので、多くの人達が簡単にあなたのゲームを遊んでみるすることができます。詳細は後ほど説明します。

また「Google Play」や「App Store」からリリースすることも多くの人に遊んでもらう手段です。詳細はこちも後ほど解説しますが、Google Play や App Store からリリースすることでより多くの人達に遊んでもらえる可能性があります。そこからまた多くのフィードバックが得られるでしょう。ぜひこのリリースも目標にしてトライしてみてください。

モバイルアプリ用のプラットフォーム以外も PC 用ゲームを配布するプラットフォームもいろいろあります。PC 用アプリの配布プラットフォームで最も有名なのは「Steam」です。

目標を順番に大きくしていこう。

まずはテキストのサンプルをたくさん作り理解を深め、自分のゲームを作り上げてみる。作り上げることは重要だよ。

そして、数あるプログラミングコンテストにチャレンジをしてみよう。コンテスト入賞を狙ってアイデアや技術を磨こう。

最後には、より多くの人達に遊んでもらえるアプリ配布プラットフォームにゲームをリリースしてみよう。



【子供向けプログラミングコンテスト】

いくつかお薦めの子供向けプログラミングコンテストを紹介します。どれも有名で既に何年も行われているコンテストとなります。（これら以外にもたくさんコンテストはあります。）

（各コンテストの攻略のヒントは、次の章（次のテキスト）で詳しく解説されています。）

① Unity インターハイ

まず Unity と言ったら Unity Japan が主催の「Unity インターハイ」と言うゲームコンテストがあります。これは毎年 18 歳以下の子を対象にしたコンテストです。

<https://inter-high.unity3d.jp/>



2020 年の募集要項は以下の通り。（最新の情報は直接ホームページを確認して下さい。）

応募資格	<ul style="list-style-type: none"> 西暦2002年4月2日以降に生まれた方。 チームでの応募も可能です。チームメンバーの人数制限はありません。 チームメンバーの構成員全員が、同じ学校に所属している必要はありません。 正式登録する代表者はリーダー1名、メンバー2名までとします。 Web掲載やプレゼン発表会出場で名前が公開されるのは代表者3名となります。 作品開発のテーマはありません。自由に作品を開発してください。
------	--

② 全国小中学生プログラミング大会

15 歳以下を対象としています。他の高校生も出場するコンテストより中学生には有利かも。本コンテストはゲームに限らず実用的なアプリも対象です。

<http://jjpc.jp/>



2020 年の募集要項は以下の通り。（最新の情報は直接ホームページを確認して下さい。）

> 応募資格	<p>日本国在住の、6歳以上15歳以下(2020年4月1日時点)の小学生・中学生グループで応募する場合は3人以下。</p> <p>応募は1人（または1グループ）何作品でも可能</p>
--------	---

③ U-22 プログラミングコンテスト

とても歴史のあるコンテストです。22歳までが対象で大学生も含まれています。そのため他のコンテストよりレベルが高くなってしまいます。他のコンテストの先にある目標としておいても良いかと思います。



<https://u22procon.com/>

2020年の募集要項は以下の通り。（最新の情報は直接ホームページを確認して下さい。）

参加資格	22歳以下（西暦1998年4月2日以降に生まれた方） ※チームの場合も参加資格は原則上記の通りです。但し、チームの代表者が22歳以下で、メンバーの半数以上が22歳以下で構成されている場合は、同一学校に所属する学生限定で、23歳以上（西暦1998年4月1日以前に生まれた方）もメンバーの一員として参加することを許容します。
------	---

④ アプリ甲子園

モバイルアプリが対象となります。高校生以下が参加できます。こちらにもゲームに限らず多くの実用的なアプリが応募されています。ソースコードを事前に提出してソースコードの実装力と技術チャレンジも審査されます。



<https://www.applikoshien.jp/>

2020年の募集要項は以下の通り。（最新の情報は直接ホームページを確認して下さい。）

応募資格
<ul style="list-style-type: none"> ・中学校、高等学校、高等専門学校(3年生まで)に在籍の方※小学生も応募可能です ・もしくは、生年月日が2002年4月2日以降の方 ・グループでのご応募も可能です

⑤ TECH KIDS GRAND PRIX

このコンテストは小学生が対象です。小学生にとっては他の中高生が出るコンテストよりは向いているかもしれません。しかし、小学生だけと言ってもけっこうレベルは高いです。ゲームの応募が多いですが実用アプリを作成する子もいます。

<https://techkidsschool.jp/grandprix/>



2020年の募集要項は以下の通り。（最新の情報は直接ホームページを確認して下さい。）

対象者	すべての小学生（生年月日が2006年4月2日~2012年4月1日の方）
-----	-------------------------------------

⑥ 日本ゲーム大賞 U18 部門

東京ゲームショーの際に行われる日本ゲーム大賞の18歳以下部門です。作品のテーマには指定はありませんがゲームのみのコンテストとなります。ゲームの作品としての審査と共にプレゼンテーション自体も審査を行う特徴があります。

<https://u18.awards.cesa.or.jp/>



2020年の募集要項は以下の通り。（最新の情報は直接ホームページを確認して下さい。）

応募資格
<ul style="list-style-type: none"> 西暦2002年4月2日以降に生まれた方。 日本国内在住者に限ります。 応募作品数の制限はありません。 グループでの応募も可能です。グループの人数は最大5名までとします。 予選大会および決勝大会でのプレゼンテーション発表に参加できる人数は、最大3名までとします。 グループの構成員全員が、同じ学校・組織に所属している必要はありません。

プログラミングコンテストのより詳しい情報は次の章
(次のテキスト)を参考にしよう。

各コンテストの攻略のヒントも詳しく載ってるよ!



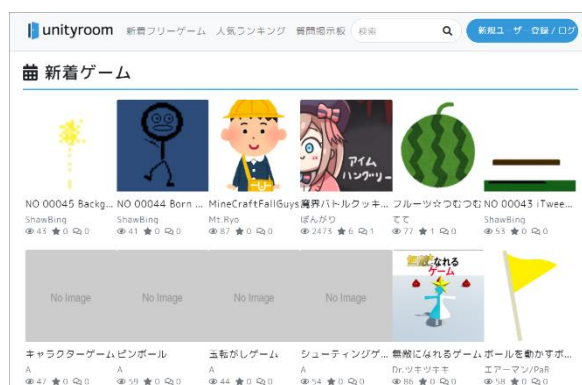
【多くの人に遊んでもらう】

ここではより多くの人に遊んでもらう手段を紹介します。

① Unity Room

<https://unityroom.com/>

Unity で作成したゲームを人に遊んでもらう代表的な方法です。Unity で WebGL アプリとして作成したゲームを Unity Room にアップすることにより、簡単に多くの人に遊んでもらうことができます。



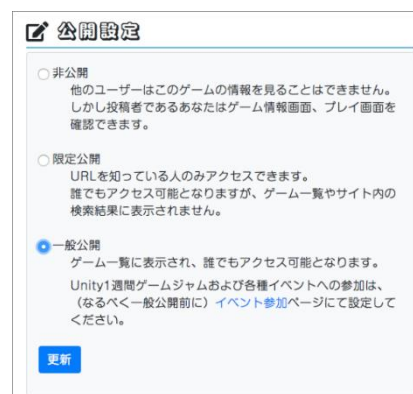
リリースしていないゲームを友達のデバイス（スマホや PC）にインストールすることが

実は簡単ではありません。WebGL アプリであればブラウザで実行できるのでインストールが必要ありません。

しかし、WebGL をブラウザで実行しようとした場合、この WebGL アプリはサーバー上に置かれている必要があります。

それを容易に実現できる手段が Unity Room です。

Unity Room は Twitter のアカウントがあれば無料で利用させてもらえます。公開設定で友達だけに遊んでもらうこともできます。



Unity Room の公開方法に関しては、また先のテキストで詳しく説明します。

② Google Play や App Store

Google Play や App Store からリリースしてみるのも多くの人達に遊んでもらう手段となります。こちらはお金もかかるので、子供には少し難しい部分もあります。

しかし、少し先の目標として置くのにはとても良い手段かと思えます。親御さんのサポートがあれば、あなたが作ったゲームアプリをリリースすることもできます。



やはり自分のゲームアプリが Google Play や App Store からリリースされるのはテンションが上がりますよね。そして友達だけでなく市場からの評価を受けることができます。ぜひトライしてみてください。



App Store

Google Play や App Store からのリリースについても、また先のテキストで詳しく説明します。

③ Steam や Microsoft Store

上記した Google Play や App Store はモバイルアプリの配布プラットフォームです。では本格的な PC ゲームを作った場合はどうすればよいのでしょうか。



一番有名な PC 用アプリの配布プラットフォームは Steam です。ゲーム以外の PC 用アプリも配布していますが、PC ゲームと言ったら Steam が代表です。

<https://store.steampowered.com/>

また Microsoft の Windows アプリを作ったのであれば Microsoft Store という PC 用アプリの配布プラットフォームもあります。




第1.2章. Unityを使ってみよう

この章ではUnityを初めて使う人を対象に、Unityのインストール法からUnityエディタの基本的な使い方までを説明します。

(UnityエディタとはUnityでゲームを作成する際に使用する開発環境です。このUnityエディタでUnity製のゲームを作ることになります。プログラミングはMicrosoftのVisual Studioと言う開発ツールを使用します。)

【注意事項】

このテキスト全体に言えることですが、画面イメージはWindowsのものを使用しています。Macを使用している場合、適宜そのOSに適した操作を行ってください。重要な部分は解説しております。



早速無料の Unity
をインストールして
みよう。

1.2.1. Unityのインストール

ここでは Unity のインストール法を説明します。

Unity を使う際は Visual Studio のインストールも必要となります。Unity と Visual Studio 共にまだインストールされていないことを前提に話を進めます。

【インストール作業概要】

Unity インストールの大きな流れは以下の様になります。この順に沿って説明していきます。

- ① Unity Hubインストーラーのダウンロード
- ② Unity Hubのインストール
- ③ Unity IDの新規登録とサインイン
- ④ 新規ライセンスの認証
- ⑤ Unityのインストール

【インストール方法】

① Unity Hub インストーラーのダウンロード

ブラウザで以下の Unity のサイトにアクセスします。

<https://store.unity.com/ja/download?ref=personal>

(このリンクは Unity Personal ライセンスへのリンクです。他に Unity Plus と Unity Pro があります。無料で使用できるのは Unity Personal となります。Unity Personal は全ての機能を有しています。)

Unity のサイトから無料の Unity Personal ライセンスでダウンロードしよう。



Unity Personal を利用する規約条件に同意する「条件に同意する」フラグにチェックして、「Unity Hub をダウンロード」をクリックします。

(Mac をお使いの場合は、「選択 Mac OS X」をクリックして同じ作業を続けます。)

条件に同意する

利用規約および下記の制限に従って Unity Personal を利用できることをご確認の上、チェックボックスをクリックしてください。

- Unity Personal が商業目的で使用されているのか、社内のプロジェクトまたはプロトタイプ用に使用されているのかかわらず、会社の年間総売上が \$100,000 を超えないようにしてください。
- \$100,000 以上の資金調達を行っていません。
- Unity Plus または Pro を現在利用していません

Unity Personal を使用する資格がない場合は、[ここをクリック](#)して、Unity Plus および Unity Pro の詳細をご覧ください。

Unity Hub をダウンロード

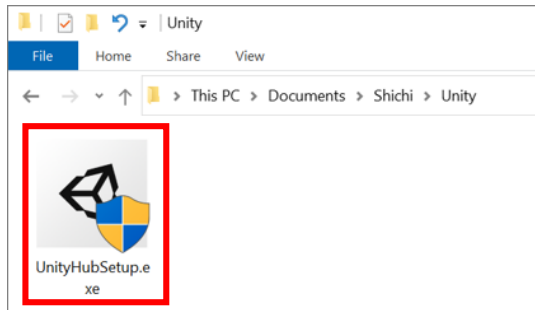
Mac OS X 用の Unity Hub のダウンロードをお探しますか?
[選択 Mac OS X](#)

ブラウザの下側に以下のボタンが出ましたら、「Save」(保存)をクリックして、インストーラを保存します。



② Unity Hub のインストール

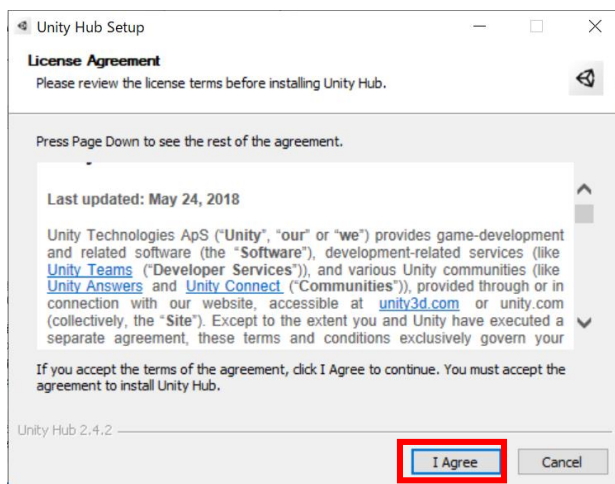
PC にダウンロードしたインストーラーファイルをダブルクリックして起動させます。



Unity Hub のインストーラーが起動されます。

最初にダウンロードしたのはインストーラーだけなので、ここから大量なファイルがダウンロードされます。
(回線スピードが速く、従量課金でない Wifi につなげておくべきです。)

ライセンス契約書の画面が表示されます。内容を読んで、「I Agree」（同意する）をクリックします。

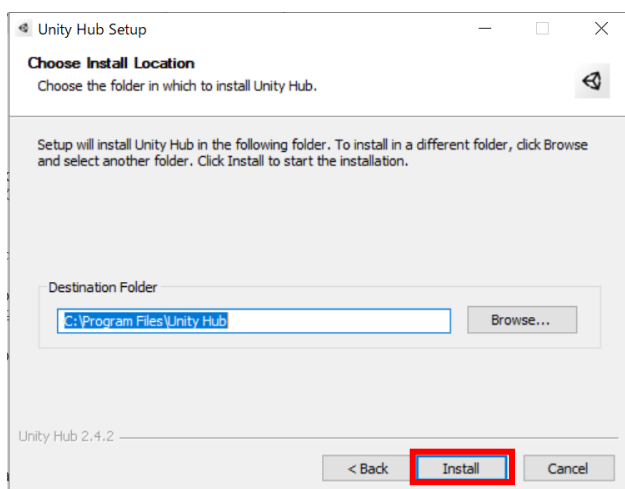


Unity をインストール
するためには Unity
Hub をインストールす
る必要があるんだ。

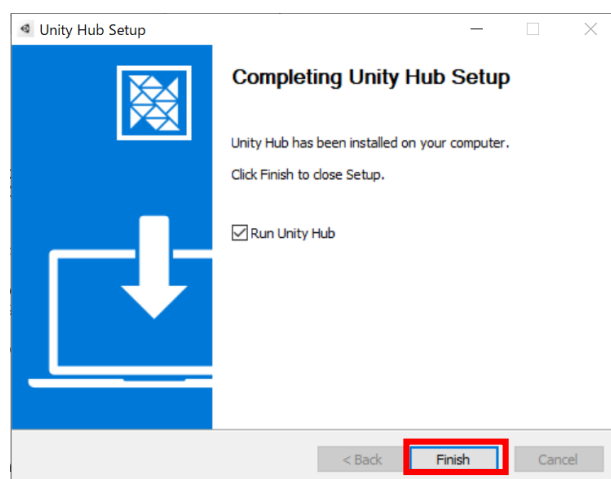


次にインストール先フォルダの設定画面が表示されます。ここは変更の必要はありません。

「Install」 をクリックします。



Unity Hub のインストールには、あまり時間はかかりません。以下の完了メッセージが表示されたら、「Finish」 (完了) をクリックします。

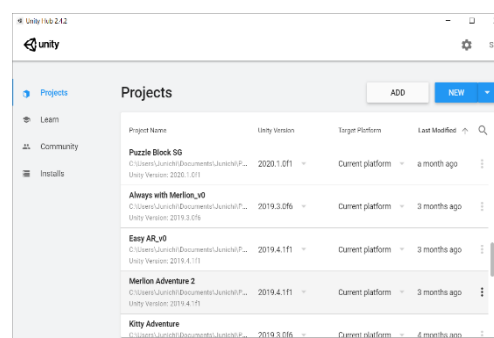


【参考】【Unity Hub とは】

【Unity Hub とは】

Unity Hub とは、Unity で自分が開発しているゲームのプロジェクトと、インストールしてある Unity エディターのバージョンを管理するためのツールです。

新規プロジェクトを作成する際は、Unity Hub から行います。



③ Unity ID の新規登録とサインイン

起動した Unity Hub の右上にある Unity ID アイコンをクリックし、そこで開いたメニューの「サインイン」をクリックします。



Unity ID のサインイン画面が表示されますので、既に Unity ID をお持ちの際は、Email、Password を入力して、「サインイン」をクリックします。

Unity ID をまだ新規登録していない場合は、「ID を作成」をクリックします。



Unity を使うためには Unity ID が
必要なんだ。

持ってない人は新規登録をしよう。
簡単に無料で登録できるよ。



Unity ID の新規登録画面が表示されます。

メールアドレス、パスワード、ユーザーネーム、フルネームを入力します。

(ユーザーネームは他の人に使われていないものがが必要です。もし既に誰かに使われている名前だと、「既に存在するので他の名前にして下さい」とメッセージが出ます。)

2つのチェックボックスの上側「利用規約とプライバシーポリシー」はチェックが必須です。下はどちらでもかまいません。

そして、「Unity ID アカウントを作成」をクリックして Unity ID を新規登録します。



The screenshot shows the 'Unity Hub Sign In' window. The title bar reads 'Unity Hub Sign In'. The Unity logo is in the top left. The main heading is 'Unity ID アカウントを作成'. Below it, a link says 'すでに Unity ID をお持ちの場合は、[こちらからサインインしてください。](#)'. The form has four input fields: 'メールアドレス' (shichi.junichi1@gmail.com), 'パスワード' (masked with dots), 'ユーザーネーム' (Merlion_Lab7), and 'フルネーム' (Shichi Junichi). Below the fields are two checkboxes: 'Unity の利用規約とプライバシーポリシーに同意します' (checked) and 'このボックスにチェックすることで、Unity からのプロモーション広告を受け取ることに同意します。' (checked). At the bottom, there are two buttons: 'すでに Unity ID をお持ちですか?' and 'Unity ID アカウントを作成' (highlighted with a red box). A 'または' link is at the bottom right.

新規登録すると、同時にサインインもされます。既に Unity ID を持っていた方は、サインインしてここから始めます。

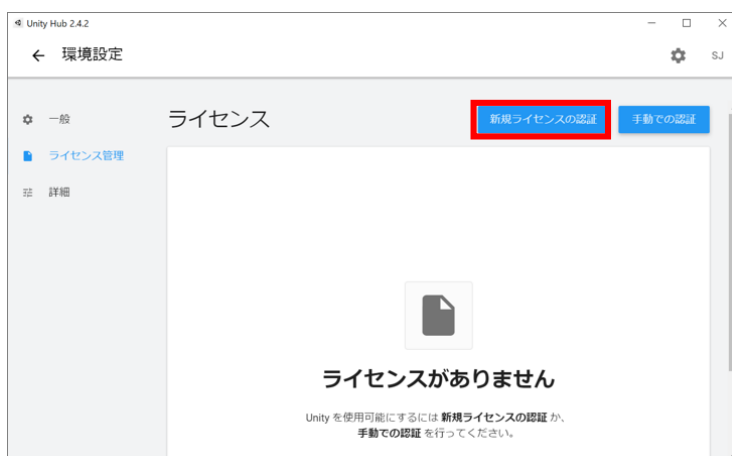
画面の Unity ID アイコンには、自分のイニシャルが表示されています。



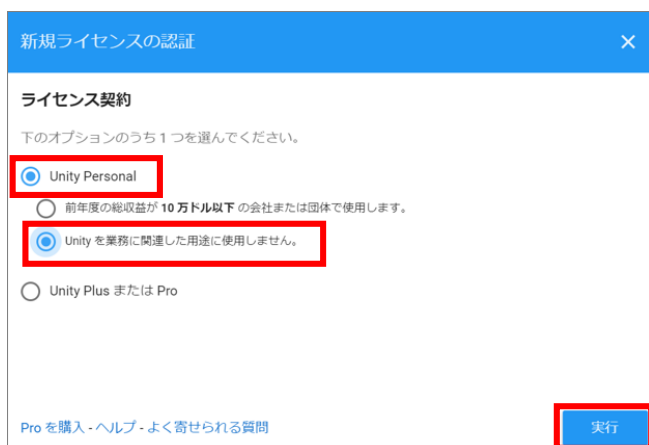
④ 新規ライセンスの認証

最初に無料バージョンの Unity Personal を選んでいます。ここで再度、Unity Personal を使うライセンスの認証をする必要があります。

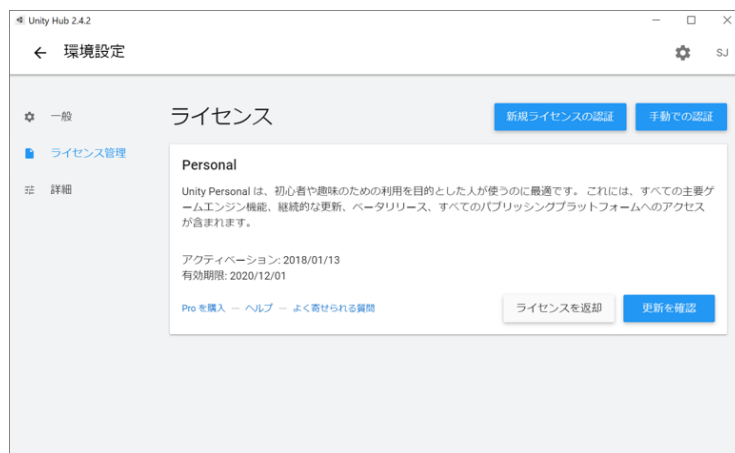
まず、「新規ライセンスの認証」をクリックします。



ライセンス契約画面で「Unity Personal」、「Unity を業務に関連した用途に使用しません。」を選択して、「実行」をクリックします。



Unity Personal のライセンス認証が完了しました。



【参考】【Unity のライセンスは無料なの？】



【Unity のライセンスは無料なの？】

Unity は皆さんのようなプログラムを勉強するために使っている方々には完全に無料です。そして、その作ったゲームをリリースしても、そこから収益を得ても Unity を無料で使えます。

(無料のライセンスである Unity Personal を使うことができます。)

では有料のライセンスは誰が対象なのでしょう。それはその収益額に関連します。収益が約 1000 万円 (10 万ドル) を超えたら Unity Pro のライセンス費用を払う必要があります。

(これらの規約やライセンス費用は変更される可能性がありますので、正確には Unity のサイトで確認して下さい。)

Plus	Pro	Enterprise
より充実した機能とリソースでプロジェクトをパワーアップ	制作、運用、マネタイズのためのプロ向け完全ソリューション	日本の場合は営業窓口から販売しております。オンラインからご購入いただいた場合は英語のみのサポートとなります。
43,995円 (1シートあたりの年額)	198,000円 (1シートあたりの年額)	22,000円 (1シートあたりの月額)
年間プラン、年払い	年間プラン、年払い	1年契約、月払い

⑤ Unity のインストール

では、やっと Unity をインストールする準備ができました。左のメニュー上で「インストール」をクリックしてインストール画面を表示させます。右上の「インストール」ボタンをクリックします。



インストールする Unity のバージョン選択画面が表示されます。推奨リリースか正式版の中でも新しいバージョンを選択します。ここでは推奨リリース「Unity 2019.4.14f1 (LTS)」を選択しました。そして、「次へ」をクリック。

(推奨リリースとは 1 つ前のバージョンでバグなどをかなり解消している安定バージョンです。また、正式版の中で上にあるのが最新バージョンです。これには最新機能も含まれています。)



Unity の推奨リリースを
インストールします。



次にインストールをするモジュールを選択します。

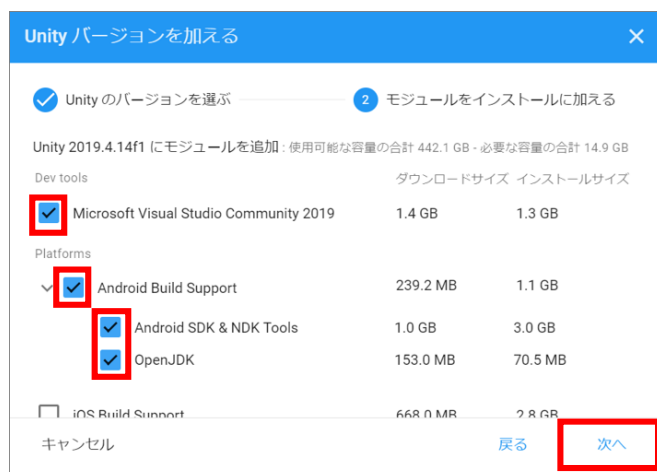
Unity ではプログラムを書く際に Microsoft Visual Studio Community 2019 を使用します。既にインストールしてある場合は表示されませんが、まだインストールしていない場合「Microsoft Visual Studio Community 2019」が表示されチェックがついています。

また、Android 用のアプリを作成する場合、「Android Build Support」にチェックします。そして、そこを展開して「Android SDK & NDK Tools」と「OpenJDK」にもチェックします。

(Mac を使用して Android 用のアプリを作成する場合も上記のようになります。また、Mac を用いて iPhone 用のアプリを作成する場合は、「iOS Build Support」をインストールする必要があります。)

(また、ここで選択しなかったモジュールも後からインストールは可能です。)

最後に、「次へ」をクリックします。

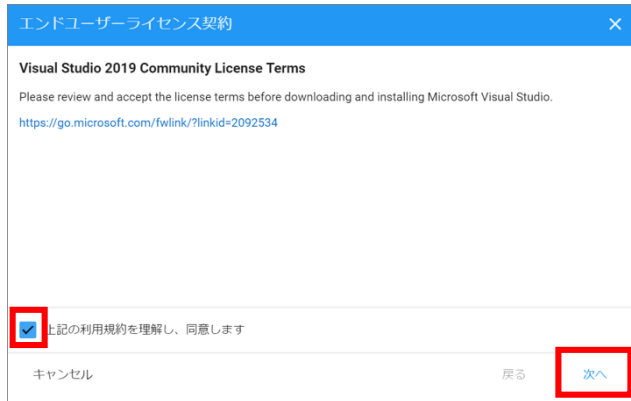


Unity にインストールするモジュールを選択します。

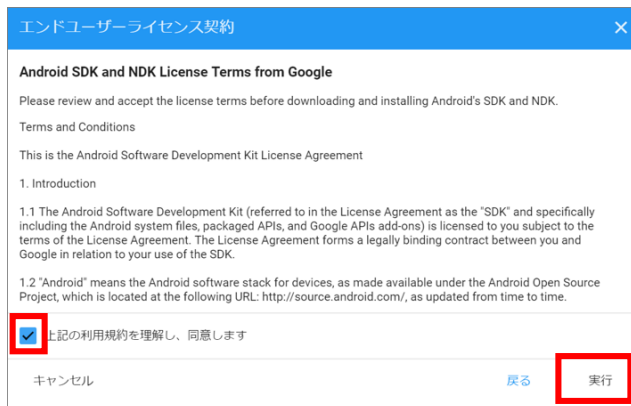
選択するモジュールはターゲットとするプラットフォームによって変わります。



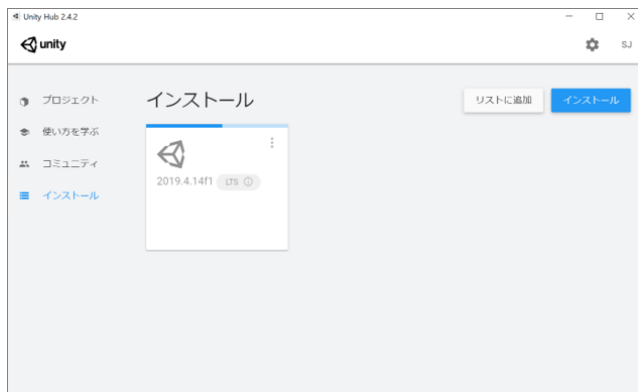
プログラムを作成する際に使用する Visual Studio 2019 Community のライセンス規約の同意画面が表示されます。「同意します」にチェックをして「次へ」をクリックします。



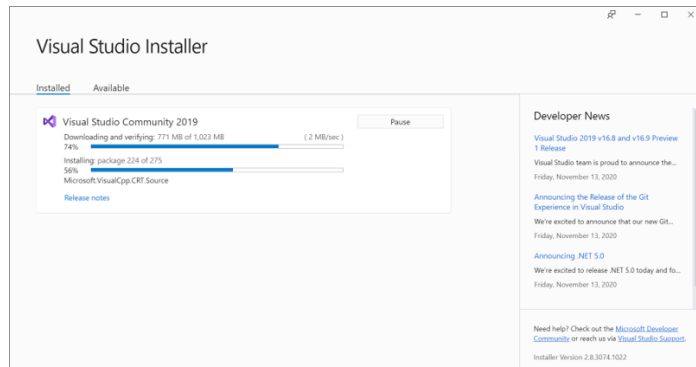
次に Android SDK and NDK License の規約の同意画面が表示されますので、「同意します」にチェックして「実行」をクリックします。



Unity のインストールが始まります。



自動的に Visual Studio のインストールも開始されます。



インストールが無事に終了しました。



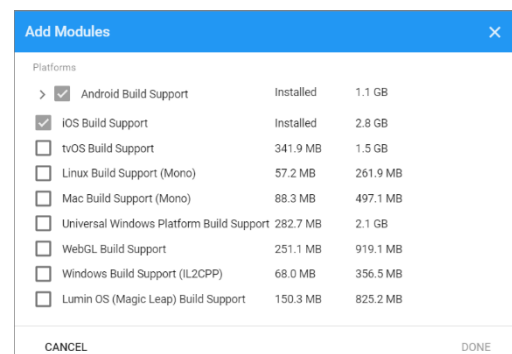
【参考】【他のモジュールの追加】

【他のモジュールの追加】

ここではターゲットのプラットフォームに Android を想定して Android Build Support をインストールしました。

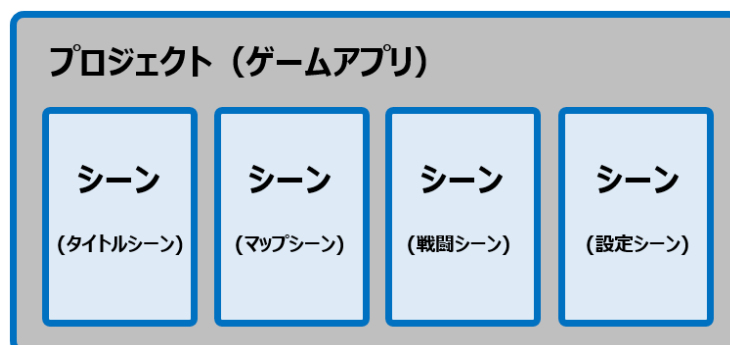
iOS や他の OS、Web 用アプリをターゲットとする場合は該当するモジュールも追加インストールして下さい。

それぞれハードディスクの容量を大量に必要とするので、必要となった時にインストールをすれば良いです。



1.2.2. プロジェクトとシーンの作成

ここでは Unity でどんなゲームを作るときにも必要となる Unity の基礎概念としてプロジェクトとシーンを紹介します。



【プロジェクトとは】

Unity でゲームを作る際、まずプロジェクト (Project) を作成します。プロジェクトとはゲーム全体、ゲームアプリ自体と言えます。作成するゲームアプリに1つ最初にプロジェクトを作成することになります。

次の章で詳細は説明しますが、Unity エディタの Project ビューにはゲーム全体のプログラムや画像等の各種素材がまとめられています。

【シーンとは】

通常ゲームには最初にタイトルシーンがあり、マップ等を選ぶ選択シーンがあります。そしてゲームが始まるとゲームシーンが現れます。これらを Unity ではまさにシーン (Scene) と呼びます。Unity でゲームを作成する際は、このシーン単位の作成を行います。

Unity エディタの Scene ビューと Hierarchy ビューがシーンに該当し、そのシーンで使われるゲームオブジェクトが配置されています。

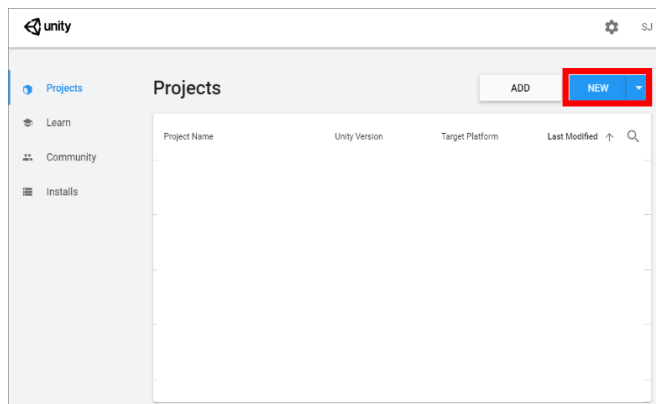
Unity ではプロジェクトとシーンはとても重要だよ。



【プロジェクトとシーンの作成方法】

ここではプロジェクトとシーンの作成方法を説明します。（ゲームを開発する際に毎回最初に行う作業と言えます。）

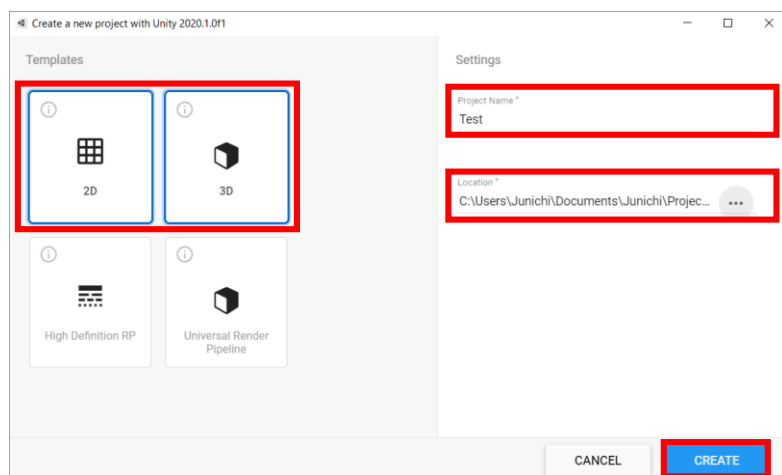
Unity Hub からプロジェクトは作成します。まず、「New」をクリックします。



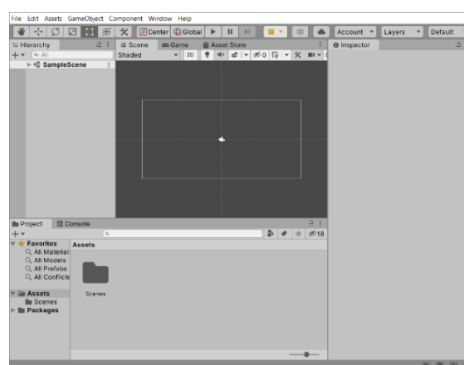
プロジェクト名（Project Name）を入力します。ここでは「Test」にしておきます。

そして、プロジェクトの保存場所（Location）を決めます。プロジェクトの保存場所はどこでも構いません。

テンプレート（Templates）は今から作るゲームが「2D」なのか「3D」なのかを選択します。ここでは「2D」にしておきます。最後に「CREATE」をクリック。

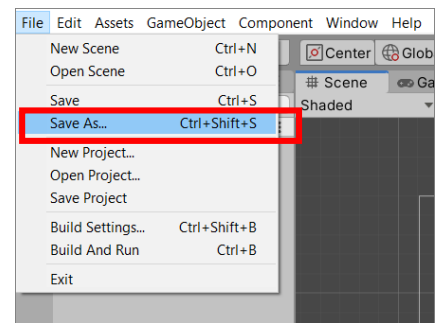


プロジェクトが作成されて、Unity エディタが起動しました。



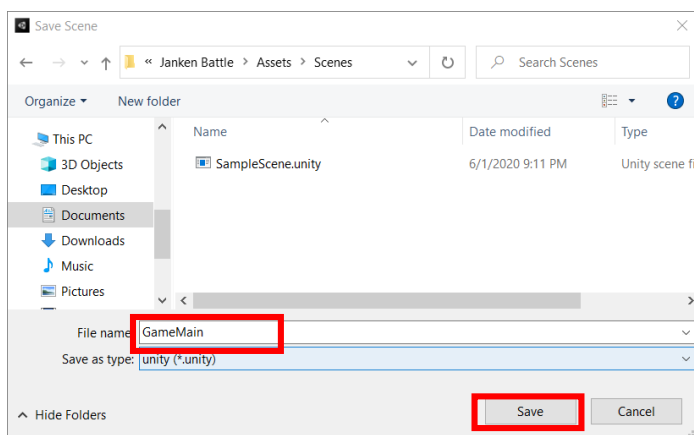
次にシーンを作成してみましょう。

Unity エディタのメニューから「File > Save As...」をクリックします。



「Save Scene」画面が表示されますので、ファイル名を入力し「Save」ボタンをクリック。

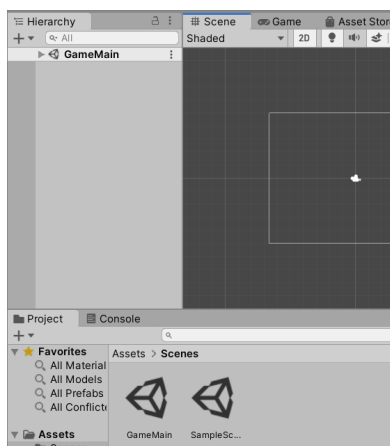
(保存場所は「Assets」か「Scenes」フォルダの中に保存します。)



シーンの保存はどこでも
良いわけではないよ。
Assets か Scenes の中
に保存。

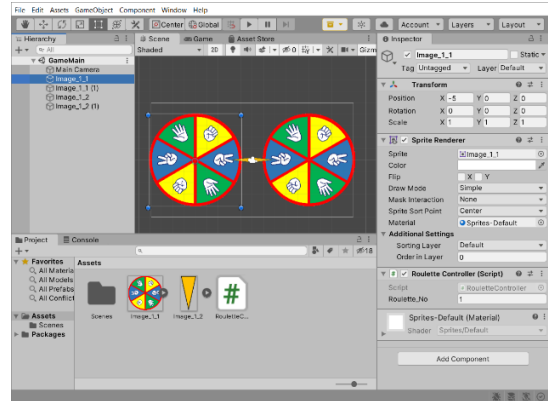


Project ビューに新たなシーンができ、Hierarchy ビューにシーン名が表示されています。



1.2.3. Unityエディタの画面構成と操作法

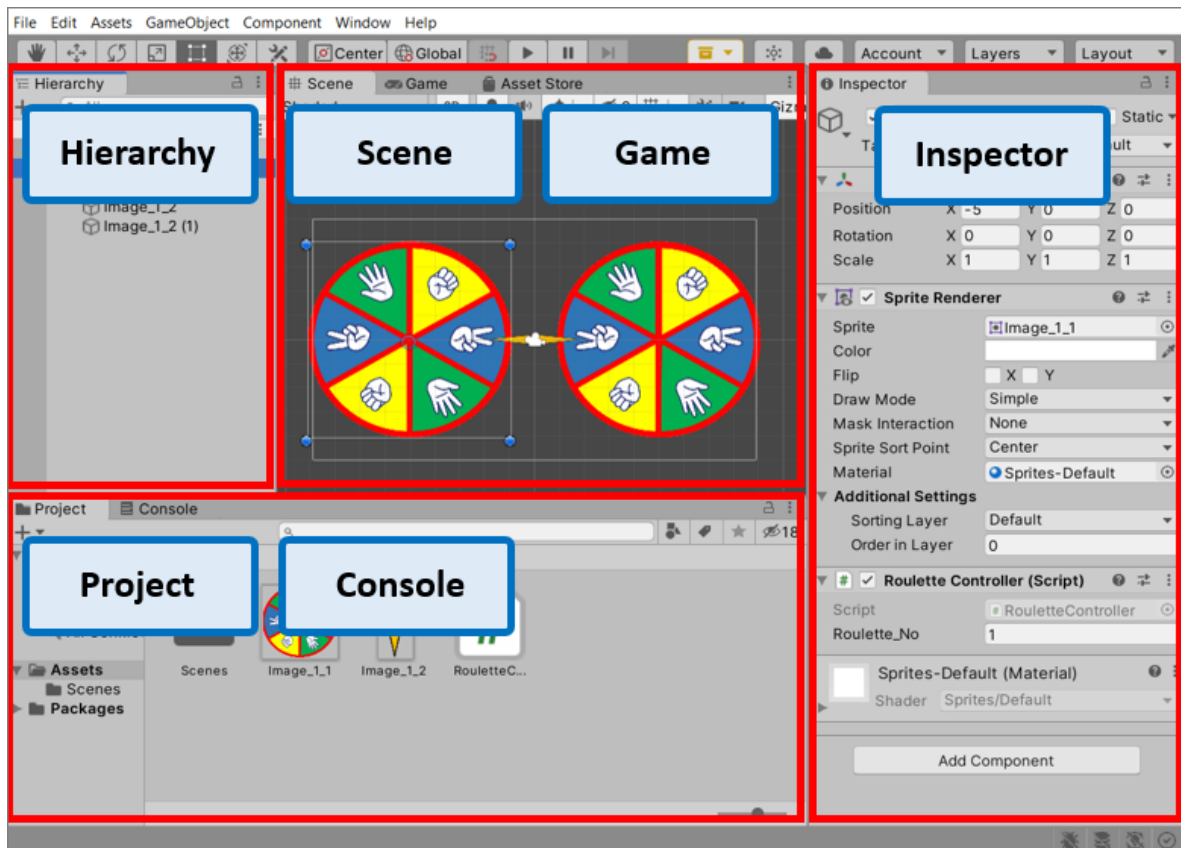
次の章で作成するゲームの作成画面を見てみましょう。Unity は以下のような画面でゲームを作ります。このゲームを作る開発環境を「Unity エディタ」と言います。



【Unity エディタの画面構成】

この Unity エディタの画面はいくつもの小さな画面（ビュー）で構成されます。まずはこの各ビューの呼び名と用途を覚えましょう。

（各ビューは画面内の場所を移動させることができるので、全体の見た目が異なる場合もあります。）



【各ビューの名前と用途】

ビューの名前	用途
Project ビュー (プロジェクトビュー)	現在作成中のプロジェクト（ゲームアプリ）が持つ全ゲームオブジェクト（プログラム、画像ファイル、サウンドファイル、3D モデル等）を表示します。
Scene ビュー (シーンビュー)	現在作成中のシーンの開発画面です。ゲームオブジェクトの位置を直接確認・設定ができます。 Scene ビュー上にはゲームに出てくるゲームオブジェクト以外にもカメラやライトなどゲーム画面からは見ることができないオブジェクトも表示されています。
Hierarchy ビュー (ヒエラルキービュー)	現在表示中のシーン上にあるゲームオブジェクトを全て階層構造で表示します。
Inspector ビュー (インスペクタビュー)	Hierarchy ビュー又は Project ビュー上でクリックしたゲームオブジェクトの詳細設定画面です。
Game ビュー (ゲームビュー)	Scene ビュー上のカメラから撮影されたゲーム自体の映像が表示されます。要は実際のゲーム画面となります。
Console ビュー (コンソールビュー)	開発中やゲーム実行中のエラーやメッセージが表示されます。

これ以降のゲームを作成する説明でも、これらのビューの名前が各所に出てきますので、覚えてしまいましょう。

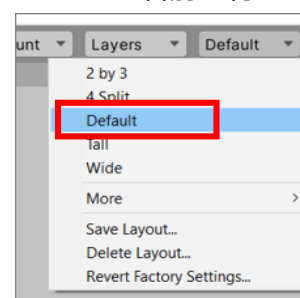
【参考】【Unity エディタのビュー配置のデフォルト】

【Unity エディタのビュー配置のデフォルト】

Unity エディタのビューの位置は個別に自由に移動できます。ビューの名前の付いているタブをドラッグすることで行えます。

本テキストの画面イメージはデフォルトレイアウトです。

Unity エディタの右上にあるプルダウンより「Default」を選ぶことで同じ配置にすることができます。

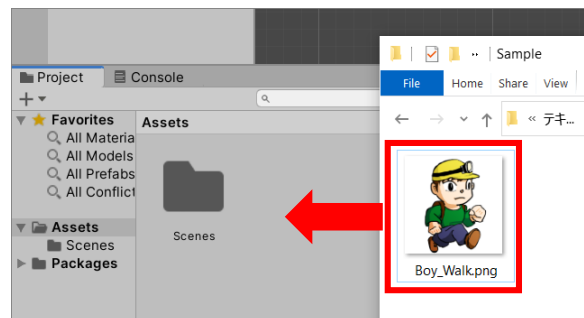


【Unity エディタの基本操作】

先ほど作成した 2D の「Test」プロジェクトを使って Unity エディタの操作法を学んでみましょう。

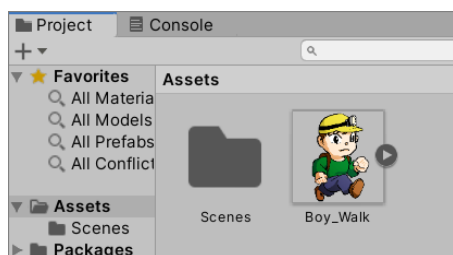
2D の画面で使用する画像データを Unity エディタに取り込みます。

ここは操作法を試すだけなのでどんな画像でもかまいません。



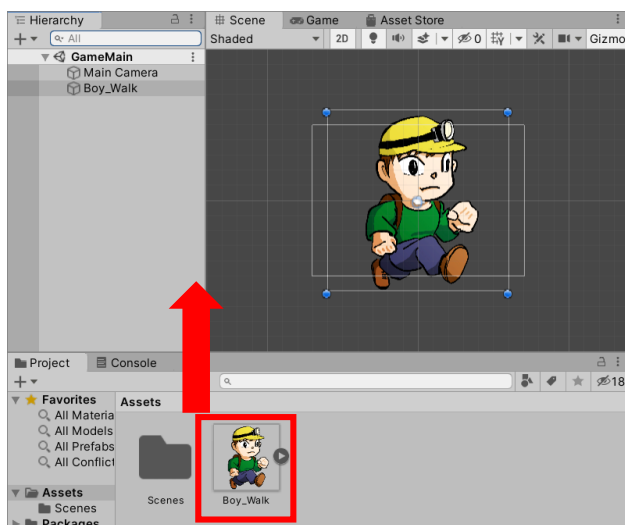
ファイルエクスプローラーから画像ファイルを Unity エディタ上の Project ビューにドラッグ&ドロップします。

画像ファイルが Unity エディタの Project ビューに表示されました。



そして、その画像を Project ビューから Hierarchy ビューにドラッグ&ドロップします。
すると Scene ビューに画像が現れました。

(Project ビューから Scene ビューへ直接ドラッグ&ドロップしても構いません。)



シーンに配置した画像オブジェクトを Hierarchy ビュー上でクリックすると Inspector ビューにそのオブジェクトの設定が表示されます。

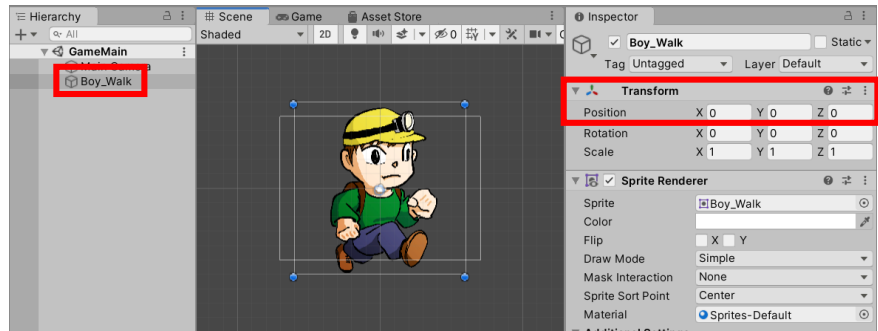
Transform コンポーネントの Position でオブジェクトの位置が調整できます。

Transform の Position に「X: 0, Y: 0, Z: 0」とあります。この画像オブジェクトのゲーム

内での位置が X・Y・

Z という 3 つの座標で

表されています。



画面の横方向が X 軸、

画面の縦方向が Y 軸で

す。

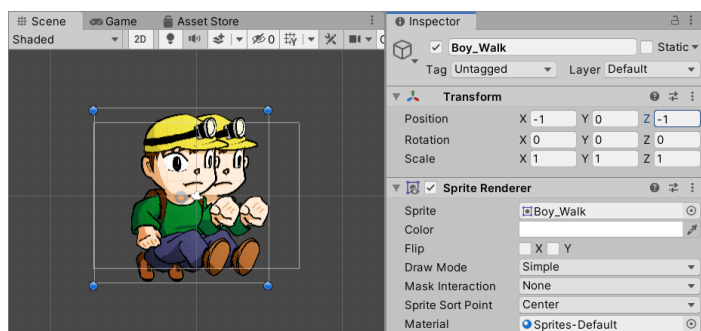
要は X に数字を入れると左右に画像が移動して、Y では上下に画像が移動するわけです。

【参考】【Unity エディタでの 2D と 3D の違い】

【Unity エディタでの 2D と 3D の違い】

基本的に Unity は 3D のゲームエンジンですので、Unity 内では情報を 3D で保持します。よって、2D のゲームでも X・Y・Z の座標で位置を表現します。しかし、ゲームの画面上では 3D は X・Y・Z 座標で表せられる奥行きのある 3D 空間で表示されますが、当然 2D は平たい平面の画面となります。

2D の場合、横軸が X 軸、縦軸が Y 軸、画面の奥へ向かって Z 軸が伸びている形となります。ですので、画像オブジェクトが 2D 画面で表示される際は平面上にあるように見えますが、画像が重なり合っている場合 Z 軸の数字が小さい画像が画面の手前側に表示されます。



ゲームを実行してみましょう。ゲームの実行は Unity エディタ上側の実行ツールで行います。

実行ツールは左側から「実行」「一時停止」「コマ送り」となります。



この「実行」ボタンをクリックすると、ゲームが起動して Game ビューにゲームが表示されます。

(ここでは画像を置いただけなので、ゲームは画像が表示されるだけです。)



【参考】【ゲーム実行時の設定変更に注意】

【ゲーム実行時の設定変更に注意】

ゲーム実行時にも Inspector ビューからオブジェクトの位置等設定を変えることができます。しかし、その場合の設定変更はゲーム実行中のみ有効で、ゲームを停止すると設定変更はクリアされてしまいます。

よくある失敗として、ゲーム実行テストで変更点に気づきゲームを停止するのを忘れたまま設定変更を行ってしまった場合。途中でゲーム実行中であることに気づいても変更した設定は停止と共にクリアされてしまいます。設定変更をする場合はゲームが起動中かどうかを必ず確認するようにしましょう。

(ゲーム実行中)



【Scene ビューでの視点操作】

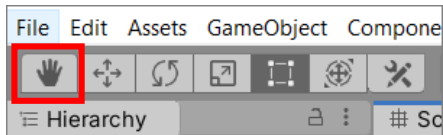
① 視点のズームイン・ズームアウト

Scene ビューでの視点のズームインとズームアウトは、マウスの真ん中でくるくる回るマウスホイールで行います。マウスホイールを手前にまわすとシーン全体をズームインすることができ、マウスホイールを奥側に回すとズームアウトすることができます。



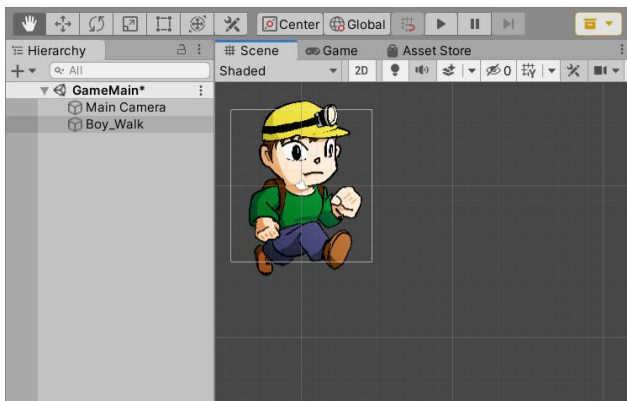
② 視点の平行移動

Scene ビューで視点の平行移動するには、画面左上にある操作ツールで画面移動アイコンをクリックします。



するとマウスのアイコンが手の形に変わり、Scene ビュー上でドラッグした方向に画面が平行移動します。

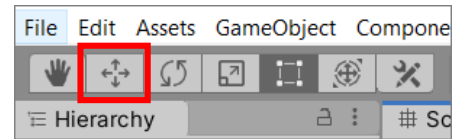
(マウスホイールをクリックしたままドラッグしても同じ操作ができます。)



【Scene ビューでのオブジェクトの変形】

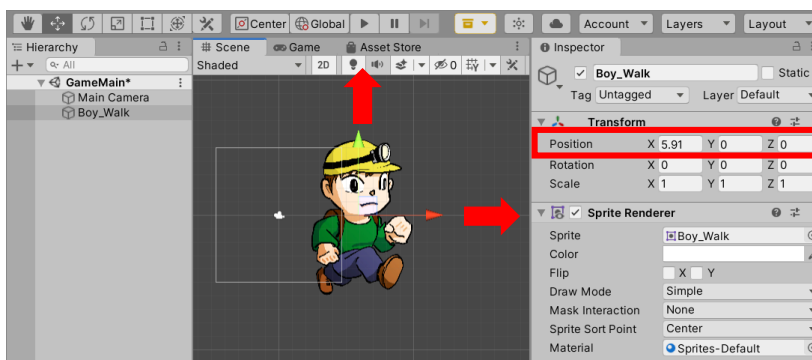
① オブジェクトの移動

オブジェクトを移動するには、画面左上にある操作ツールで移動アイコンをクリックします。



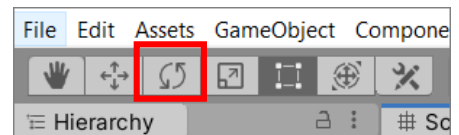
すると Scene ビュー上の画像オブジェクトに X 軸（赤色） Y 軸（緑色）の矢印が現れます。その矢印をドラッグすると軸に沿ってオブジェクトを移動することができます。

Inspector ビュー上の Position（位置）の数字も自動的に変わります。

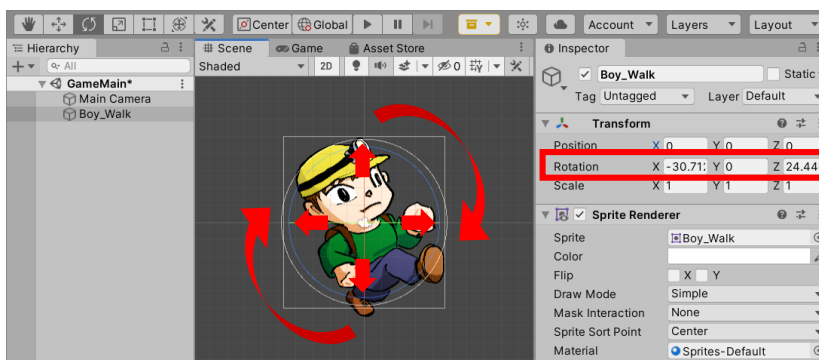


② オブジェクトの回転

オブジェクトを回転させるには、操作ツールの回転アイコンをクリックします。

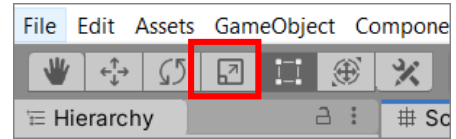


すると Scene ビュー上の画像オブジェクトに X 軸（赤色） Y 軸（緑色） Z 軸（青色）を中心とした円が現れます。その円をドラッグするとそれぞれの軸を中心にオブジェクトを回転させられます。Inspector ビュー上の Rotation（回転）の数字も自動的に変わります。

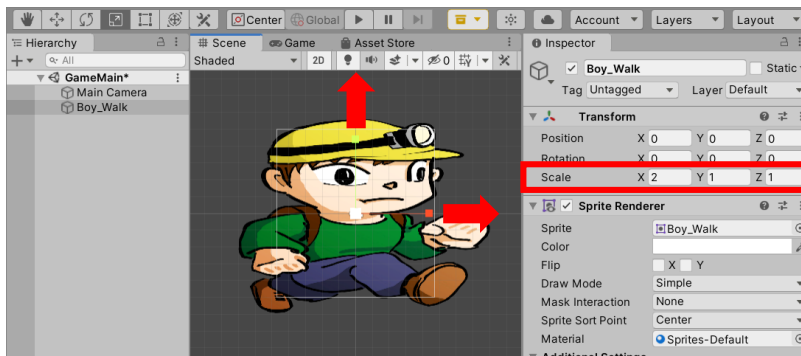


③ オブジェクトの拡大・縮小

オブジェクトを拡大・縮小するには、画面左上にある操作ツールで拡大・縮小アイコンをクリックします。



すると Scene ビュー上の画像オブジェクトに X 軸（赤色） Y 軸（緑色）の四角矢印が現れます。その四角矢印をドラッグすると軸に沿ってオブジェクトを拡大・縮小させることができます。Inspector ビュー上の Scale（大きさ）の数字も自動的に変わります。

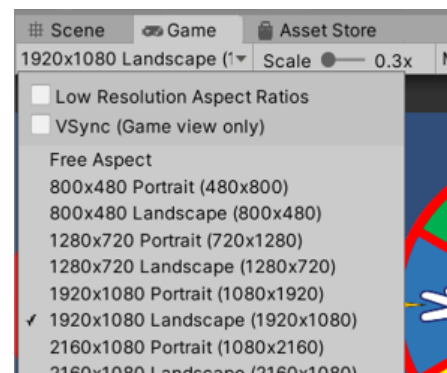


【参考】【ゲーム実行画面のサイズ変更】

【ゲーム実行画面のサイズ変更】

Game ビューの左上のプルダウンによりゲーム実行時の画面サイズを変更できます。

画面のアスペクト比（画面の横と縦の比率）を設定できます。選択した画面サイズに合わせてカメラに映る範囲が変わります。



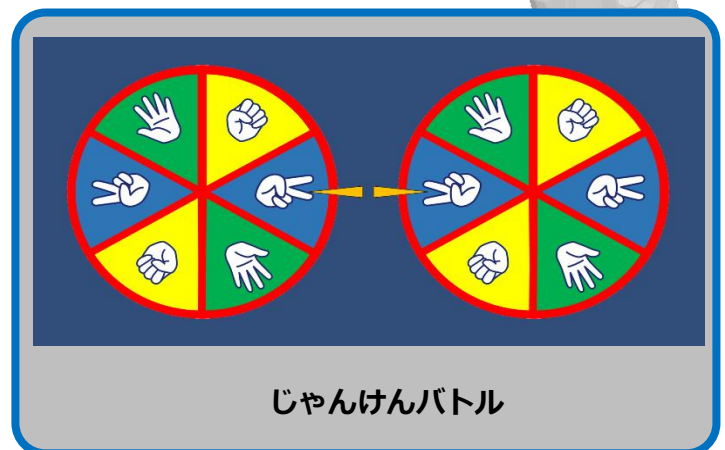
第1.3章. 簡単サンプルゲームを作ってみよう

この章ではとても簡単なサンプルゲームを作ります。サンプルゲームの作成を通して Unity でゲームを作る際の大きな流れを把握しましょう。

ここで作成するサンプルゲームは右のような画面の2人対戦「じゃんけんバトル」です。

1人がキーボード上のスペースキーを使って左側のじゃんけんルーレットを操作し、もう1人が右矢印キーを使って右側のじゃんけんルーレットを操作します。

Unity でゲームを作る際の流れを知るため、簡単な「じゃんけんバトル」を作ってみよう。



【素材（画像ファイル）のダウンロード】

サンプルゲームで使用する素材（画像ファイル）は以下のサイトからダウンロードできます。（ここからテキストにあるC#スクリプトもダウンロードできます。）

<https://www.lab7sg.com/unity-text-download/>

この「じゃんけんバトル」に必要な画像ファイルは、このサイトからダウンロードできるよ。

1.3.1. Unity でのゲーム作成の手順

では、Unity でゲームを作成する手順の大きな流れを解説します。

ここでのゲーム作成の手順は、サンプルゲームの作成のような作るものが決まっておき、画像等の素材もそろっている際の手順となります。自分でより本格的なゲームを作成する際はゲームデザインや素材（画像等）の作成等、他にも作業が発生します。

【Unity でのゲーム作成手順】



次はこの作成の流れに沿って「じゃんけんバトル」ゲームを作成して行きましょう。

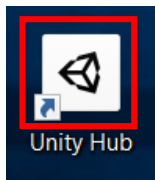
1.3.2. プロジェクト単位・シーン単位の作業

では「じゃんけんバトル」の作成を始めてみましょう。

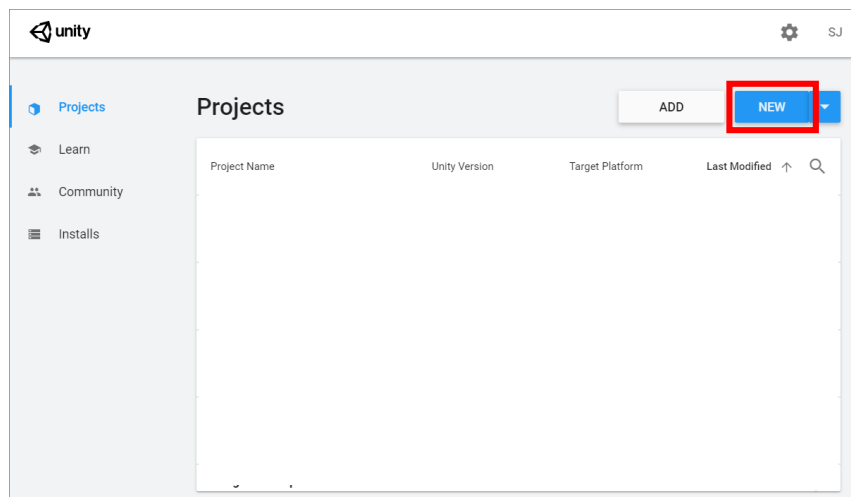
【プロジェクト単位の作業】

① プロジェクトの新規作成

まずは Unity を起動させます。デスクトップから Unity Hub のアイコンをダブルクリックします。



Unity Hub のプロジェクトの作成画面が出ますので、「New」をクリックします。

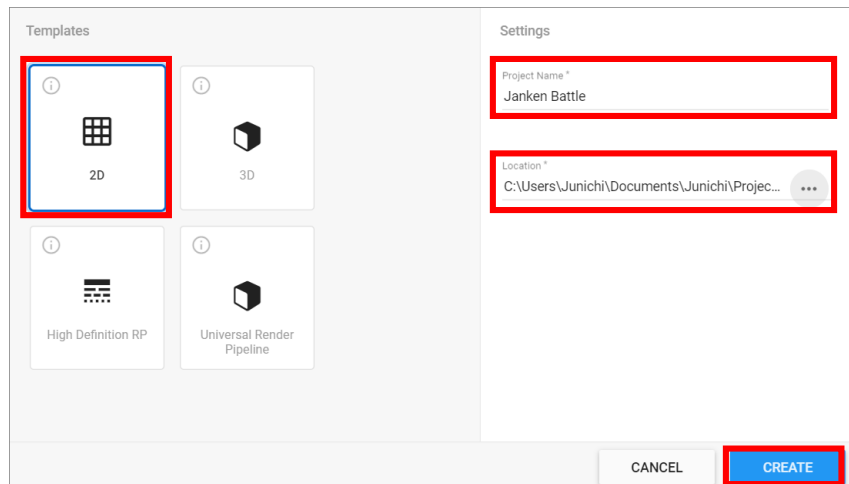


まず Unity Hub から「NEW」をクリックでプロジェクトを作成しよう。

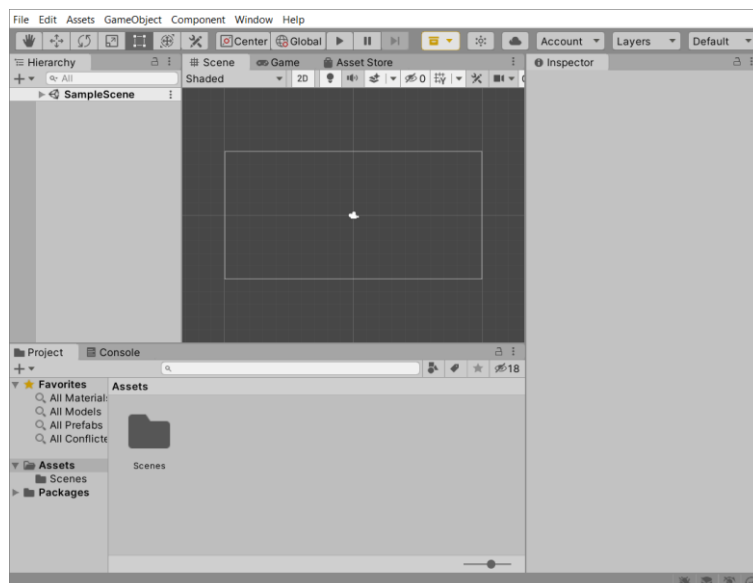


プロジェクト名 (Project Name) に「Janken Battle」を入力して、プロジェクトの保存場所 (Location) を決めます。プロジェクトの保存場所はどこでも構いません。

テンプレート (Templates) は「2D」を選択してください。そして、「CREATE」をクリックします。(プロジェクト名に日本語文字は使わないように。)



Unity エディタが起動しました。

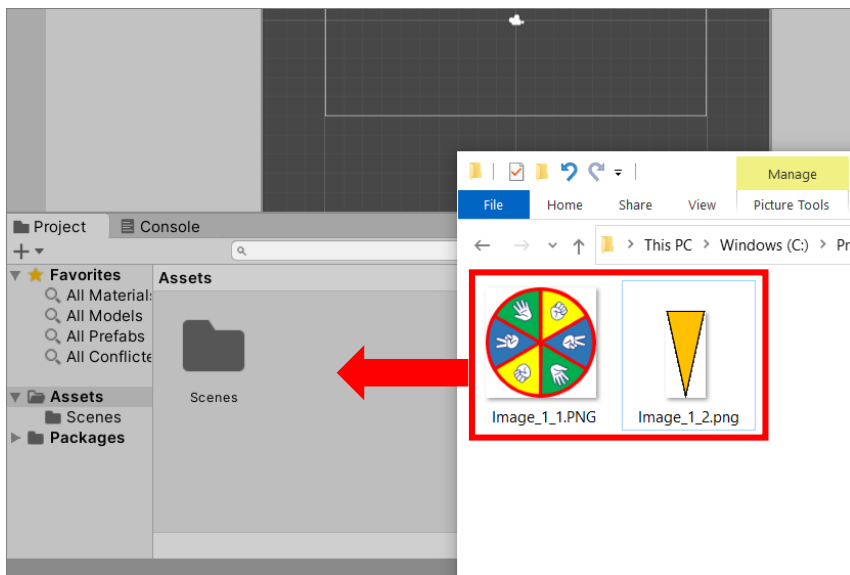


プロジェクト名、保存場所、テンプレート「2D」を選択して、「CREATE」をクリックでプロジェクトが作成できるよ。



② Unity へ素材の取り込み

ファイルエクスプローラーから「Image_1_1」と「Image_1_2」の画像ファイルを Unity エディタ上の Project ビューにドラッグ&ドロップします。（画像ファイルは事前にダウンロードしたもの。）



画像ファイルが Unity エディタの Project ビューに表示されました。



画像ファイルを Unity エディタの Project ビューにドラッグ&ドロップして取り込もう。



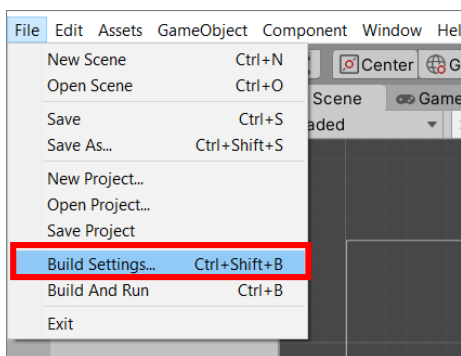
③ プラットフォームの設定

プラットフォームの設定を行います。

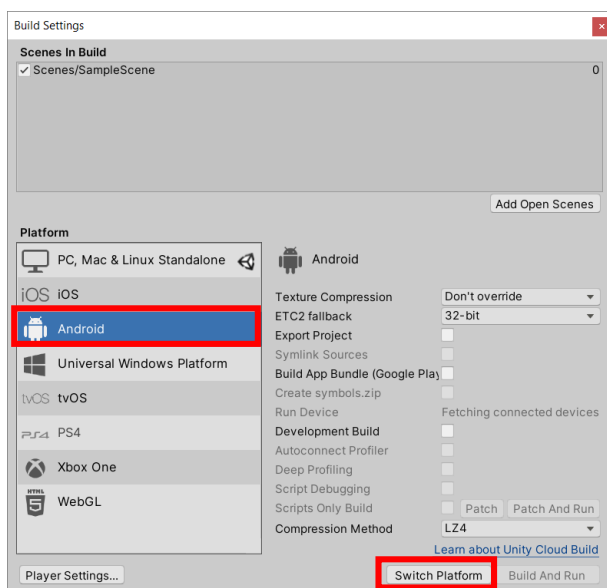
プラットフォームとは最終ターゲットとなるゲーム環境です。例えば Android や iPhone スマホ等があります。通常はその最終ターゲットに向けた画面設定をすべきなのでここでプラットフォーム設定を行います。

今回の簡単サンプルゲームではスマホへのインストールまでは説明はしませんが、仮置きで Android スマホをターゲットとしておきます。

「File > Build Settings...」をクリックします。



Build Settings スクリーンが表示されますので、Platform ビューから「Android」を選択して「Switch Platform」をクリックします。（Platform の変更は時間がかかる場合があります。）



Build Settings で
Platform を設定しよう。

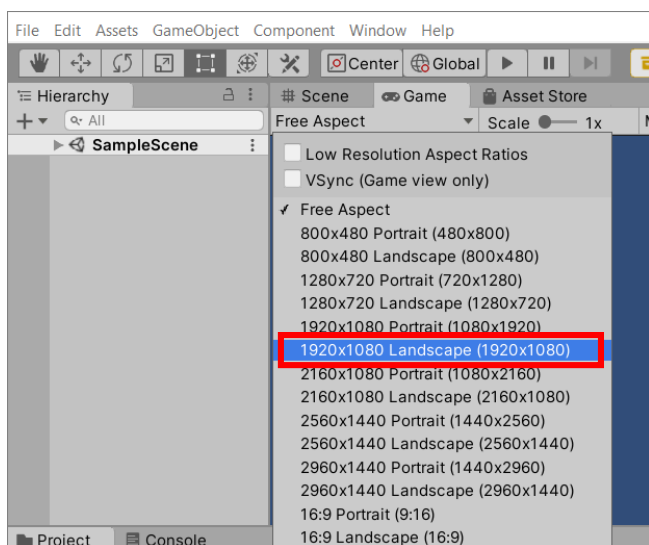
ここでは Android を選択
して「Switch Platform」
をクリック。



Game ビューの「Free Aspect」とある画面解像度設定プルダウンを開くと Android スマホにある解像度が表示されます。

通常は自分のスマホの解像度に合わせておけば良いですが、ここでは「1920×1080」にしておきます。

今回は横向き画面のゲームとなりますので Landscape（横方向）と書かれた「1920×1080 Landscape」をクリックします。



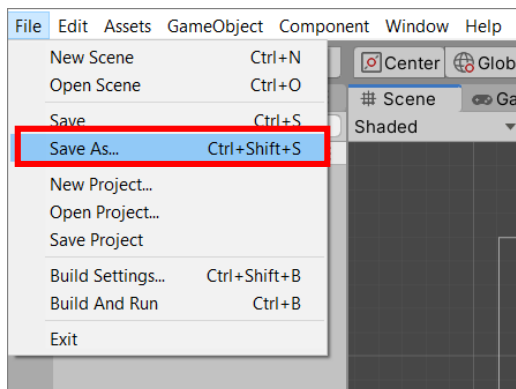
Game ビューで
画面解像度も設定
しておこう。



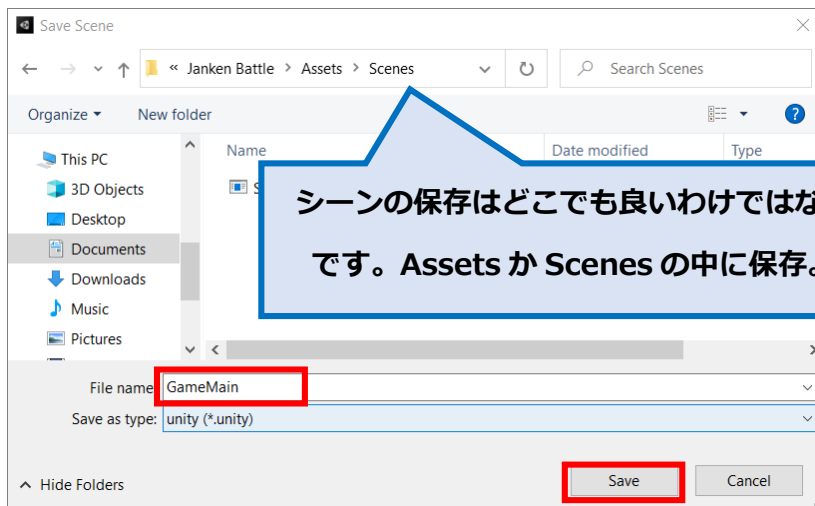
【シーン単位の作業】

④ シーンの新規作成

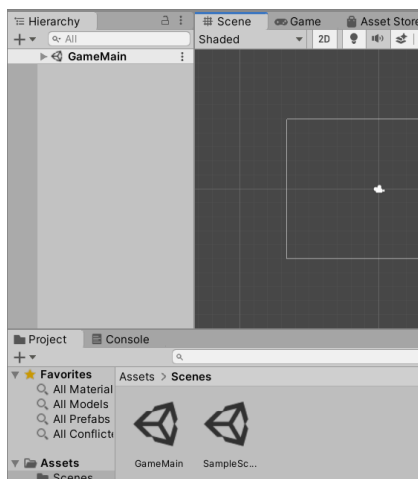
シーンを保存します。「File > Save As...」をクリックします。



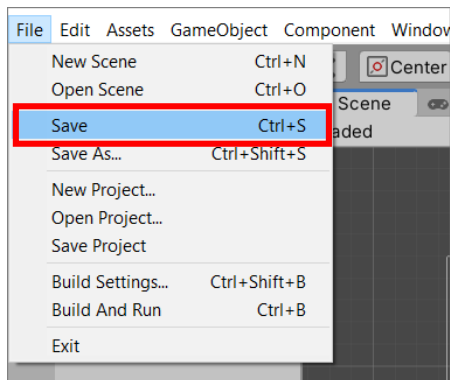
「Save Scene」画面が表示されますので、ファイル名を「GameMain」と入力します。そして、「Save」ボタンをクリック。（保存場所は「Assets」か「Scenes」フォルダの中に保存します。）



「GameMain」というシーンが保存されました。



こまめに「File > Save」をクリックして、シーンを保存しておきましょう。



1.3.3. オブジェクト単位の作業

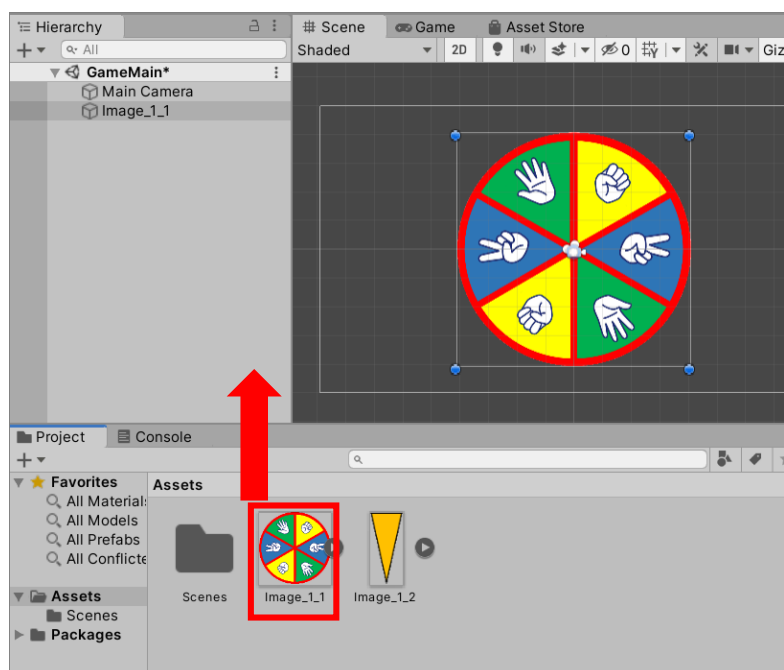
次にオブジェクト単位の作業を行きましょう。これはこの次の章の他のオブジェクトへの作業で繰り返し行うベースとなる作業になります。

【オブジェクト単位の作業】

⑤ シーンへオブジェクトを配置

じゃんけんルーレットの画像「Image_1_1」を Project ビューから Hierarchy ビューにドラッグ&ドロップします。すると Scene ビューに画像が現れました。

(Project ビューから Scene ビューへ直接ドラッグ&ドロップでも構いません。)



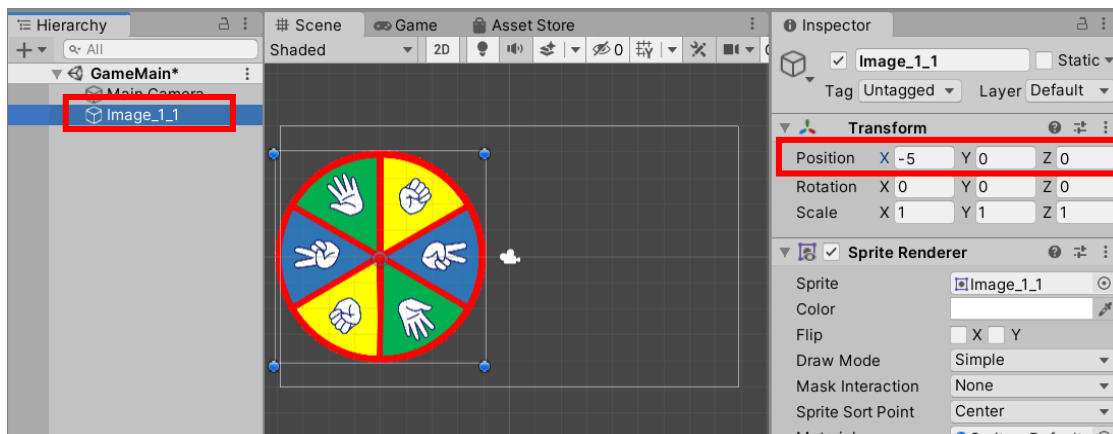
じゃんけんルーレットの画像を Hierarchy ビューにドラッグ&ドロップしてみよう。



⑥ オブジェクトのコンポーネントを設定

シーンに配置したルーレットの画像オブジェクトを Hierarchy ビュー上でクリックすると Inspector ビューにそのオブジェクトの設定が表示されます。

Transform コンポーネントの Position でオブジェクトの位置を調整します。Transform の Position を「X: -5, Y: 0, Z: 0」にします。

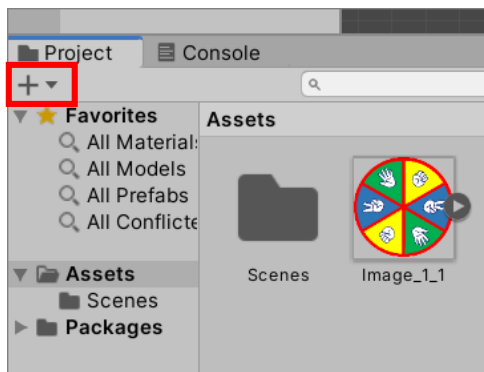


Hierarchy ビュー上でルーレットの画像を選択して、位置 (Position) を調整するよ。

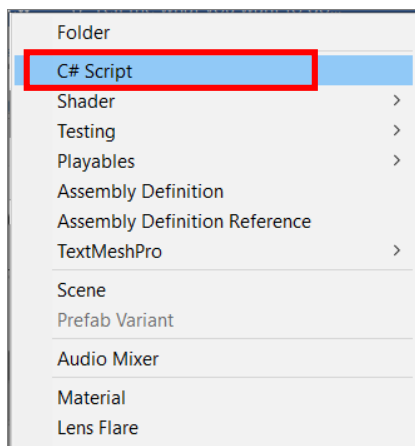


⑦ スクリプトの作成とプログラミング

Project ビューの「+」ボタンをクリックすると各種オブジェクトの作成メニューが表示されます。

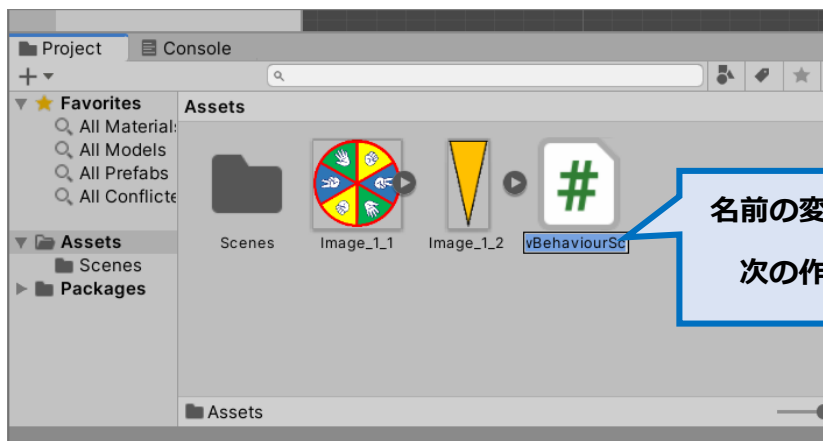


メニューにある「C# Script」をクリックします。



Project ビュー内に「NewBehaviourScript」と言う C#スクリプト (プログラム) ができました。

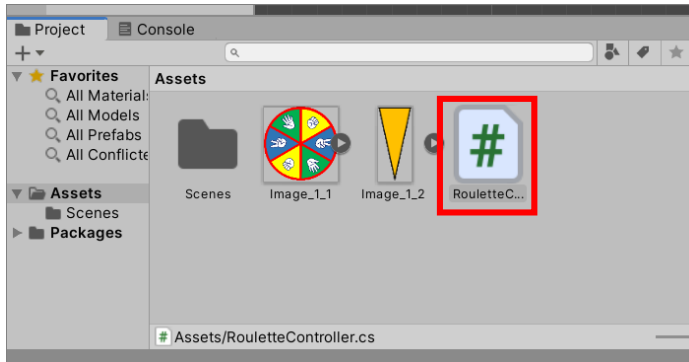
(注意) 名前の変更モードの状態です。



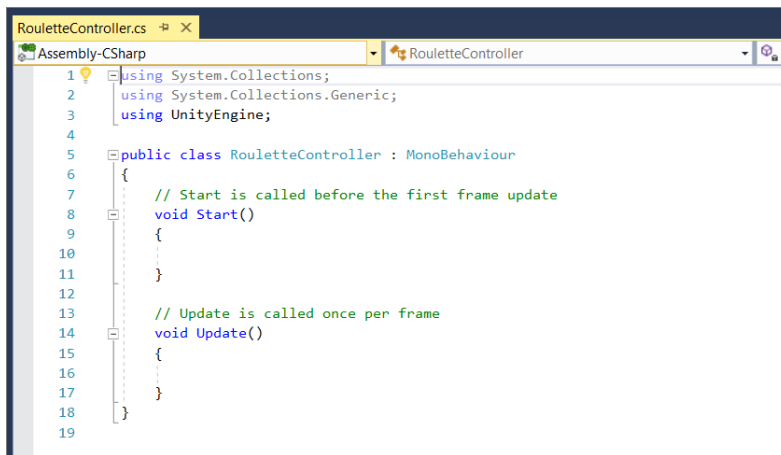
今作成されたスクリプトの名前を「RouletteController」に変更します。



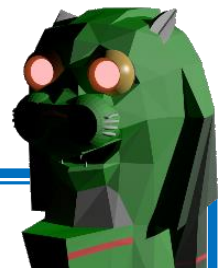
この C# スクリプトをダブルクリックして Visual Studio を立ち上げます。



起動した Visual Studio に「RouletteController」が表示されました。

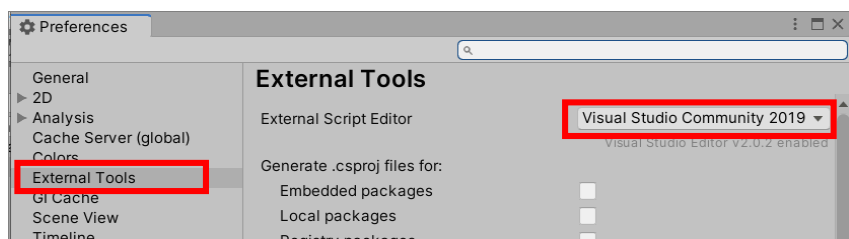


【参考】【Visual Studio が起動しない場合】



【Visual Studio が起動しない場合】

そもそも Visual Studio がインストールされていないか、Unity エディタに登録されていない可能性が考えられます。インストールされていない場合はインストールをして下さい。また、Unity エディタへの登録は、Unity エディタのメニュー Edit > Preferences で Preferences スクリーンを開き、External Tools の「External Script Editor」にてインストール済みの Visual Studio を選択して下さい。



以下の様にスクリプトの中身を修正します。(赤枠の部分を修正します。)

(注意) クラス名が「RouletteController」でなく「NewBehaviourScript」になっている場合は、そこも修正して下さい。

The screenshot shows a C# script named `RouletteController` with the following code:

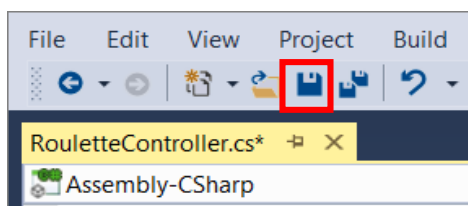
```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class RouletteController : MonoBehaviour
6  {
7      public int roulette_No;
8      private float rotaionAngle = 0;
9
10     void Update()
11     {
12         if (roulette_No == 1 && Input.GetKey(KeyCode.Space))
13         {
14             transform.Rotate(0, 0, Random.Range(-180, 180));
15             rotaionAngle = 20;
16         }
17
18         transform.Rotate(0, 0, rotaionAngle);
19
20         rotaionAngle = rotaionAngle * 0.999f;
21
22         if (rotaionAngle < 0.01f) rotaionAngle = 0;
23     }
24 }
    
```

Callouts and corrections:

- Top callout:** クラス名が「RouletteController」でなく「NewBehaviourScript」になっている場合はそこも修正。
- Line 7-8 callout:** アルファベットの大文字と小文字をしっかりと確認しよう。(Note: `rotaionAngle` is misspelled as `rotaion`).
- Line 20 callout:** 少数には最後に「f」を付けます。(Note: `0.999` is corrected to `0.999f`).

そして最後に「保存ボタン」をクリックします。



こまめに保存しておこう。
プログラムの中身の説明は後で詳しくするよ。

【参考】【プログラムに書き間違いがある場合】

【プログラムに書き間違いがある場合】

Visual Studio で右のような赤い波線が出た場合はそこにエラーがあります。よくテキストと見比べてみよう。

```

15     rotationAngle = 20;
16 }
17
18     ttransform.Rotate(0, 0, rotationAngle);
19
20     rotationAngle = rotationAngle * 0.999f;
21 }
    
```

【参考】【プログラムの詳細説明】



【プログラムの詳細説明】

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

////////////////////////////////////
// 【ルーレットのコントロールに使用するクラス】
// ルーレットをキーボード処理に合わせて回転させ始め、停止させる処理
////////////////////////////////////
public class RouletteController : MonoBehaviour
{
    ///////////////////////////////////
    // 【変数の設定】
    ///////////////////////////////////

    // 【Unity Editorから設定するPublic変数】
    public int roulette_No; // どちらのルーレットかを示す番号（左が1、右が2）

    // 【このプログラム内で使用するPrivate変数】
    private float rotationAngle = 0; // ルーレットが回転する角度（回転スピード）

    ///////////////////////////////////
    // 【関数の設定】
    ///////////////////////////////////

    // 【毎フレームに1度動く関数】
    // (Unityの標準関数)
    void Update()
    {
        // ルーレット番号が1でこのプログラムと関連付けられている画像について、
        // スペースキーが押された場合
        if (roulette_No == 1 && Input.GetKey(KeyCode.Space))
        {
            // -180度から180度の間の角度でランダムな角度に一度回転する
            transform.Rotate(0, 0, Random.Range(-180, 180));
            // 回転スピードrotationAngleの初期値を20度にする
            rotationAngle = 20;
        }

        // このプログラムが関連付けられているルーレットを
        // 回転スピードrotationAngleの角度だけ回転する
        transform.Rotate(0, 0, rotationAngle);

        // 回転スピードrotationAngleの角度に1未満の小数をかけて回転スピードを減速させる
        rotationAngle = rotationAngle * 0.999f;

        // 回転スピードrotationAngleが一定値未満になった場合
        // 止まったと考えて回転スピードrotationAngleをゼロにする
        // （これがないと見えないスピードだが永遠に回転が終わらない）
        if (rotationAngle < 0.01f) rotationAngle = 0;
    }
}
```

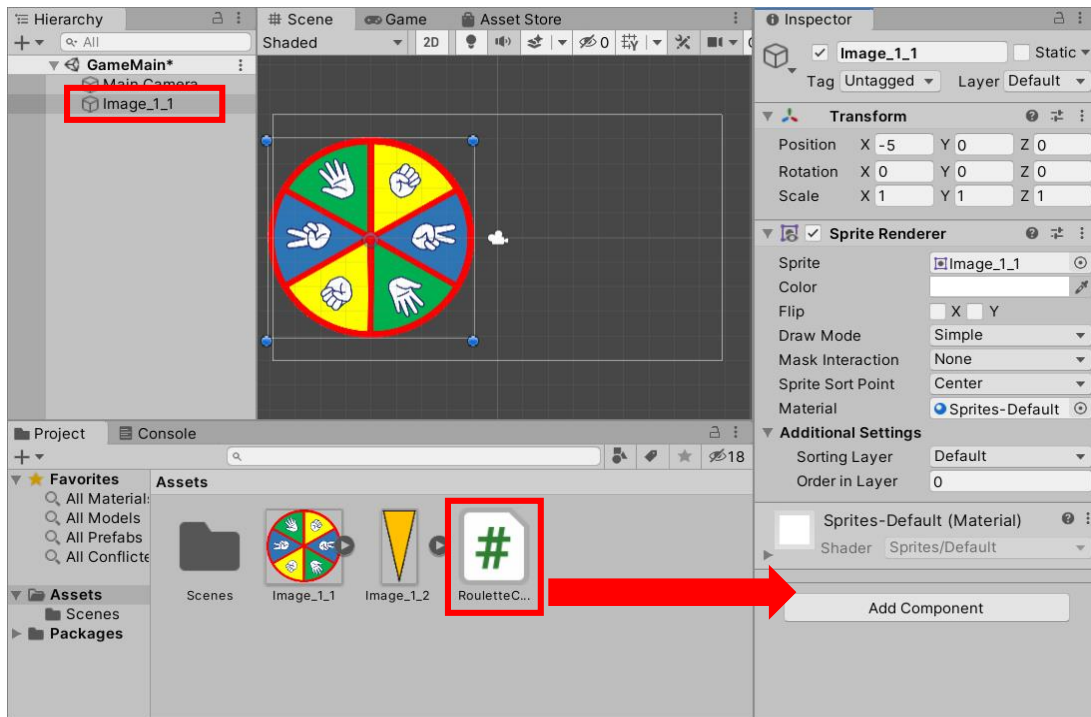
(プログラムがまだ理解できてなくてもあせらず進めてください。後からまた戻ってこれば良いですよ。)

(ここで使った C#プログラムの基礎部分は次の章で詳しく解説します。)

⑧ スクリプトをオブジェクトに関連付け

Hierarchy ビュー上でルーレットの画像オブジェクト「Image_1_1」を選択して、Inspector ビューにそのオブジェクトの情報を表示させます。

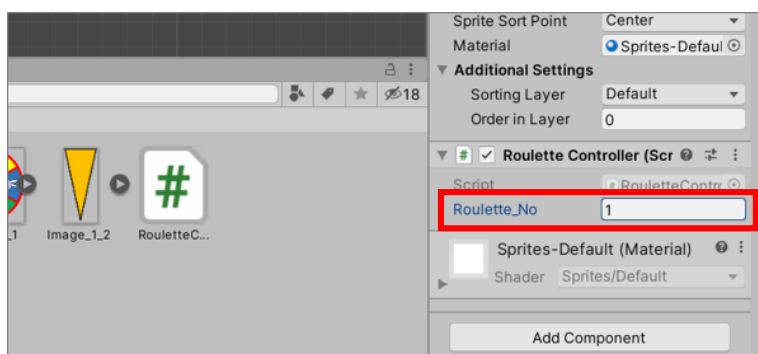
Project ビューから今作成した「RouletteController」スクリプトを Inspector ビュー上にドラッグ&ドロップします。



スクリプト内で Public 変数を使用してオブジェクトの「Roulette_No」を Unity エディタ上から設定できるようにしてあります。Inspector ビューから「Roulette_No」に「1」と入力します。

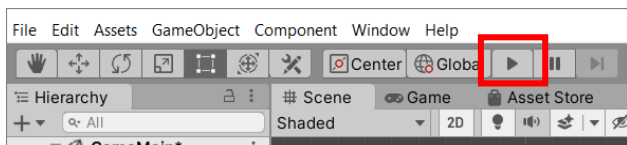
(スクリプト内で Roulette_No は、ルーレットが左側 (1) と右側 (2) を区別するために使用しています。)

オブジェクトにスクリプトを関連付けることによって、スクリプトが実行できるようになるんだ。

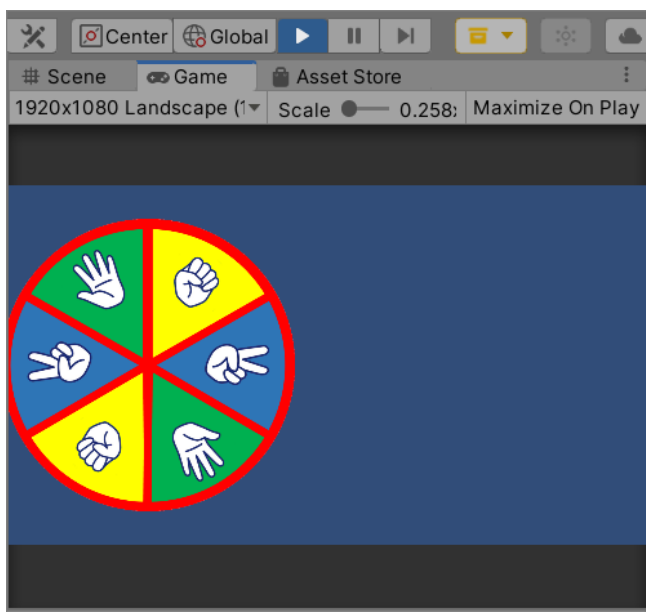


⑨ ゲーム実行をしてテスト

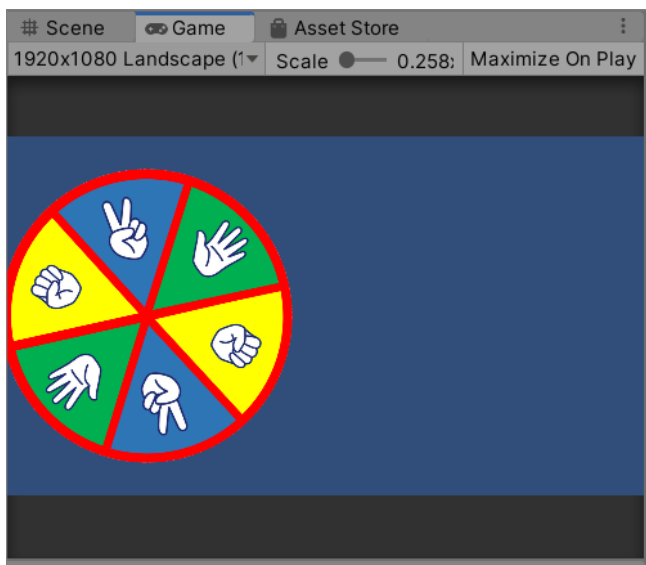
ゲーム実行ボタンをクリックしてテストをしてみましょう。



Game ビューに実行されているゲームが表示されました。



キーボード上のスペースキーを押すと回転が始まり、少しずつ回転スピードが遅くなり最後には止まりました。プログラム通りの動きです。



1.3.4. 他のオブジェクトへの作業

他のオブジェクトにも前の章で行ったオブジェクト単位の作業を行っていきましょう。

【他のオブジェクトへの作業】

① 右側のじゃんけんルーレットの設定

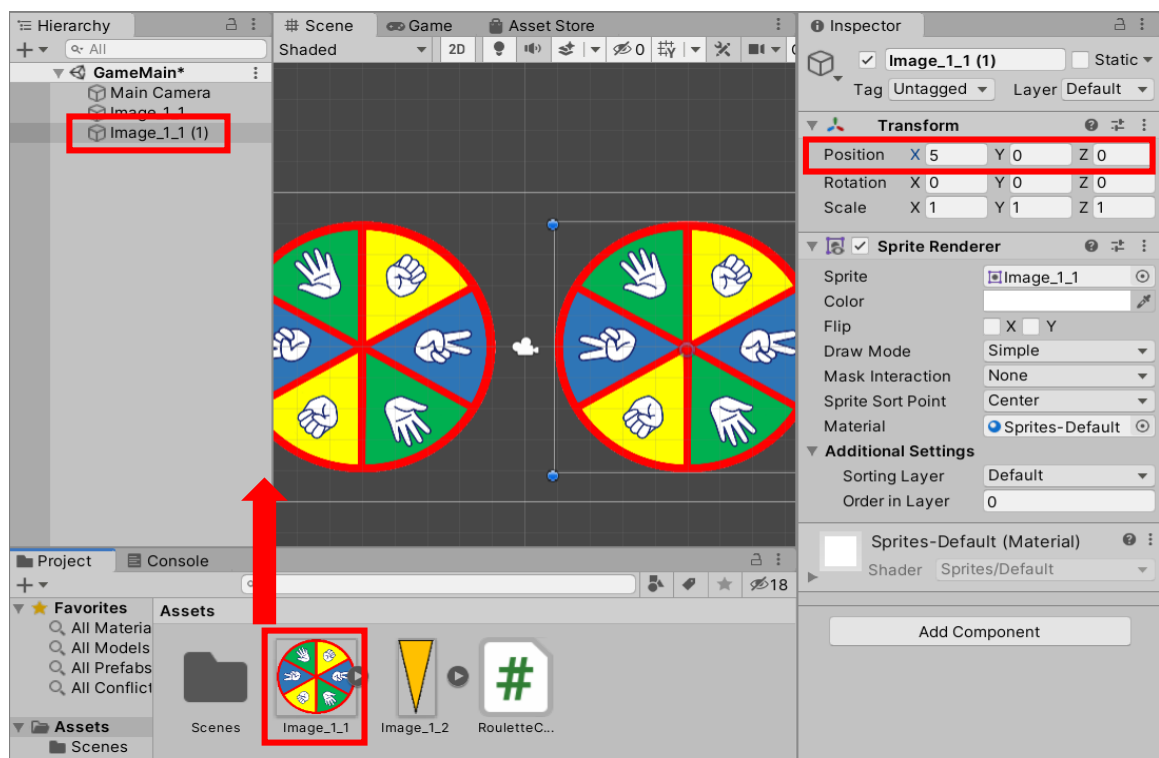
まず画面右側に置くもう1つのじゃんけんルーレットの画像「Image_1_1」を、先ほどと同じように Project ビューから Hierarchy ビューにドラッグ&ドロップします。

(Hierarchy ビュー上では同じオブジェクトがあるので、自動的に「Image_1_1 (1)」と言う名前になります。)

画面上の各画像オブジェクトに対して、前の章で行った作業をやっていくよ。



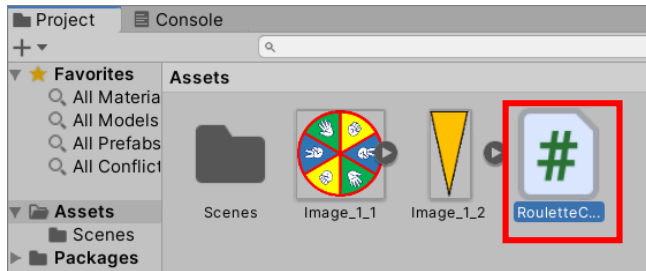
そして、その画像の位置を調整するため、Inspector ビュー上の Transform コンポーネントの Position を「X: 5, Y: 0, Z: 0」にします。



次にこの画像オブジェクト用のスクリプト（プログラム）を作成します。

このじゃんけんルーレットの動きは左側も右側もほぼ同じ動きをします。ですので、ここでは先ほど作成した「RouletteController」を少し修正して、左側にも右側にも使えるようにします。

「RouletteController」スクリプトをダブルクリックして Visual Studio を立ち上げます。



先ほど作成したスクリプトに一部追加するだけだよ。



そして今回は以下の様にスクリプトの中身を修正します。（赤枠の部分を追加します。）

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class RouletteController : MonoBehaviour
6  {
7      public int roulette_No;
8      private float rotationAngle = 0;
9
10     void Update()
11     {
12         if (roulette_No == 1 && Input.GetKey(KeyCode.Space))
13         {
14             transform.Rotate(0, 0, Random.Range(-180, 180));
15             rotationAngle = 20;
16         }
17
18         if (roulette_No == 2 && Input.GetKey(KeyCode.RightArrow))
19         {
20             transform.Rotate(0, 0, Random.Range(-180, 180));
21             rotationAngle = 20;
22         }
23
24         transform.Rotate(0, 0, rotationAngle);
25
26         rotationAngle = rotationAngle * 0.999f;
27
28         if (rotationAngle < 0.01f) rotationAngle = 0;
29     }
30 }
```

【参考】 【プログラムの詳細説明】

【プログラムの詳細説明】

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

public class RouletteController : MonoBehaviour
{
    public int roulette_No;
    private float rotationAngle = 0;

    void Update()
    {
        if (roulette_No == 1 && Input.GetKey(KeyCode.Space))
        {
            transform.Rotate(0, 0, Random.Range(-180, 180));
            rotationAngle = 20;
        }

        // ルーレット番号が2でこのプログラムと関連付けられている画像について、
        // 右矢印キーが押された場合
        if (roulette_No == 2 && Input.GetKey(KeyCode.RightArrow))
        {
            // -180度から180度の間の角度でランダムな角度に一度回転する
            transform.Rotate(0, 0, Random.Range(-180, 180));
            // 回転スピードrotationAngleの初期値を20度にする
            rotationAngle = 20;
        }

        transform.Rotate(0, 0, rotationAngle);

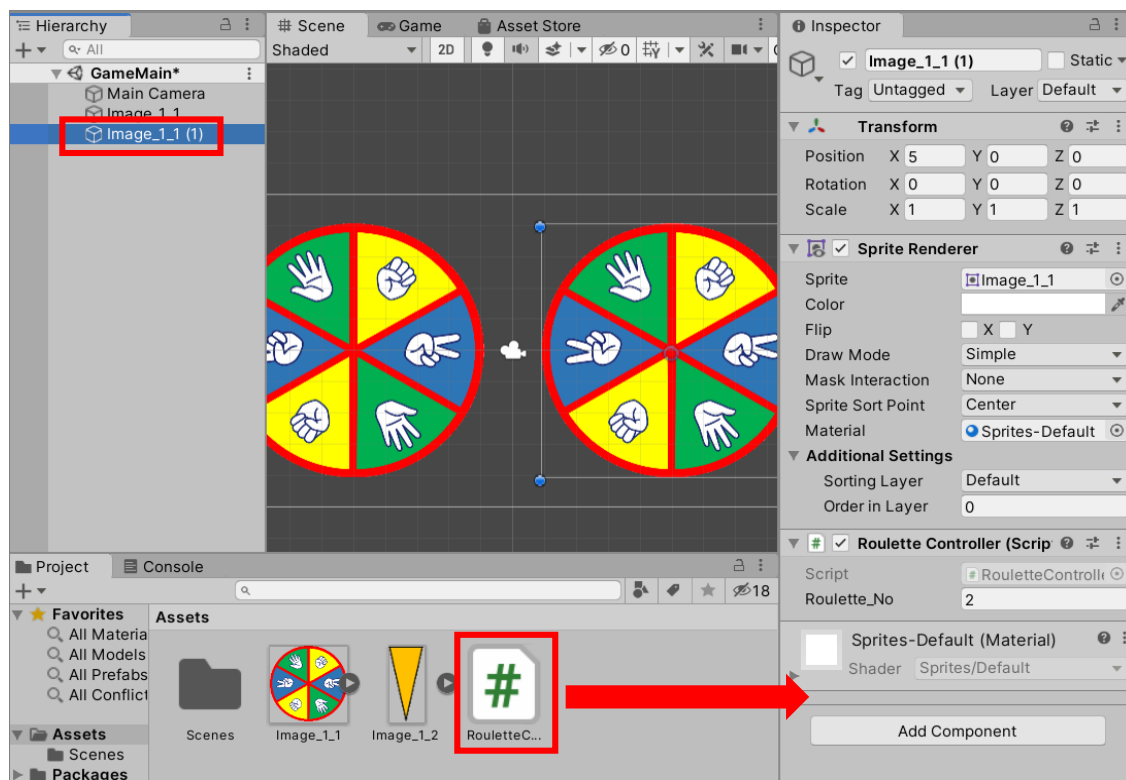
        rotationAngle = rotationAngle * 0.999f;

        if (rotationAngle < 0.01f) rotationAngle = 0;
    }
}
```



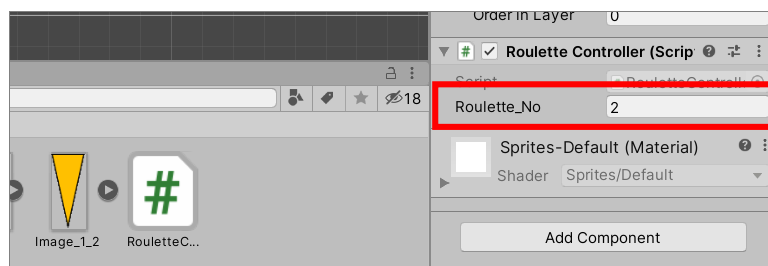
そして、修正した「RouletteController」スクリプトを右側のじゃんけんルーレットに関連付けます。

Hierarchy ビュー上で「Image_1_1 (1)」を選択します。そして、Project ビューから「RouletteController」スクリプトを Inspector ビュー上にドラッグ&ドロップします。



Inspector ビューから「Roulette_No」に「2」と入力します。

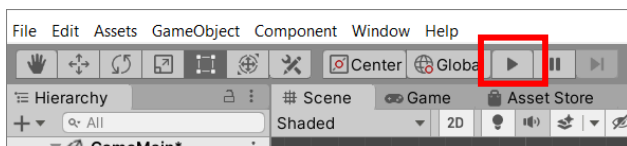
(スクリプト内で Roulette_No はルーレットが左側 (1) と右側 (2) を区別するために使用しています。)



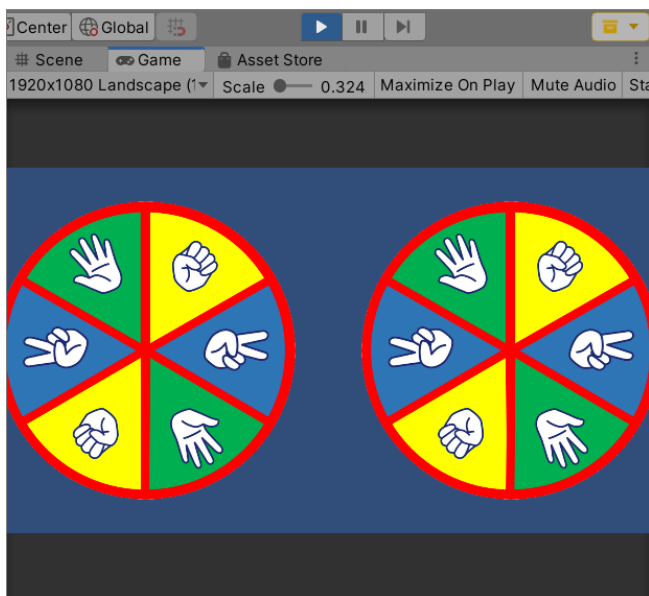
この作業は前の章で作成した左側のじゃんけんルーレットの作業と同じだったでしょ。



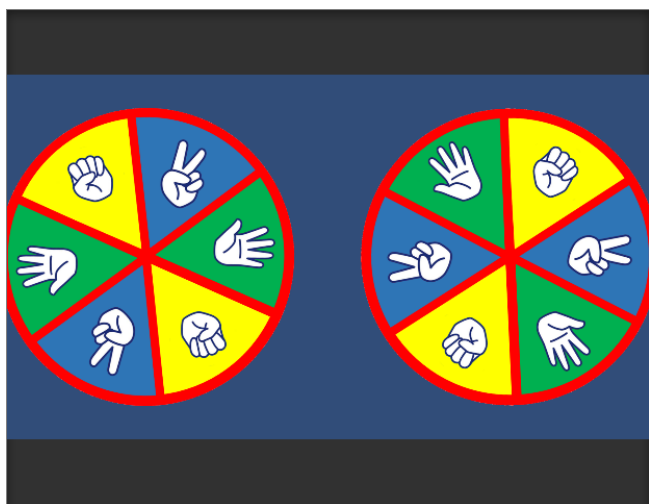
ゲーム実行ボタンをクリックしてテストをしてみましょう。



Game ビューに実行されているゲームが表示されました。



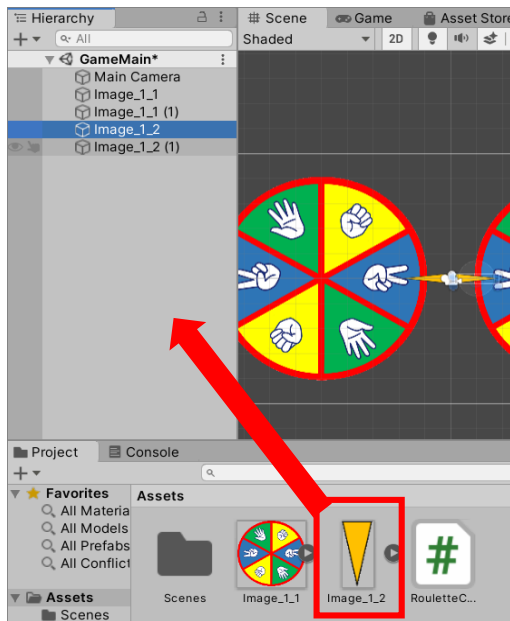
キーボード上のスペースキーで左側のじゃんけんルーレットが、右矢印キーで右側のじゃんけんルーレットが回転し始めました。どちらも少しずつ回転スピードが遅くなり最後には止まりました。今回もプログラム通りの動きでした。



左右のじゃんけんルーレットは同じスクリプトを使っています。
しかし、なぜ最後に止まる角度が違うか分かるかな？



② 停止位置の針の設定



次に停止位置を示す 2 本の針を置きましょう。画像「Image_1_2」を Project ビューから Hierarchy ビューに 2 回ドラッグ&ドロップします。

(Hierarchy ビュー上には「Image_1_2」と「Image_1_2 (1)」が現れます。)

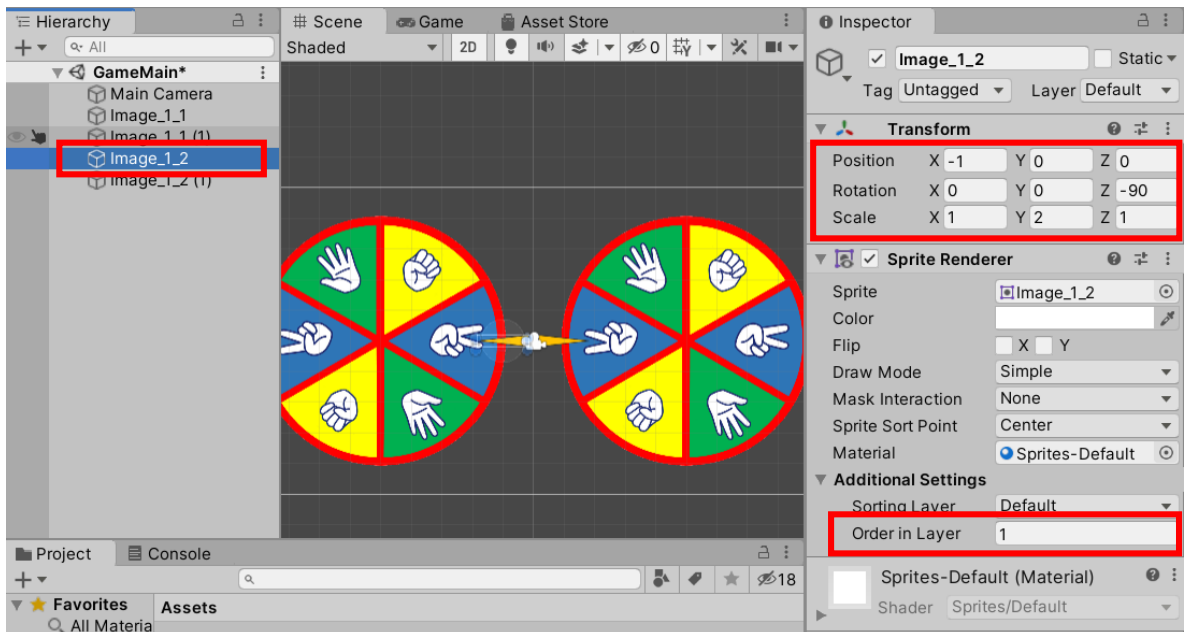
針は 2 本だから
2 回繰り返すよ。



そして、それらの画像の位置をそれぞれ調整します。まず Hierarchy ビュー上で「Image_1_2」を選択します。

Inspector ビュー上の Transform コンポーネントの Position を「X: -1, Y: 0, Z: 0」に、Rotation を「X: 0, Y: 0, Z: -90」に、Scale を「X: 1, Y: 2, Z: 1」にします。

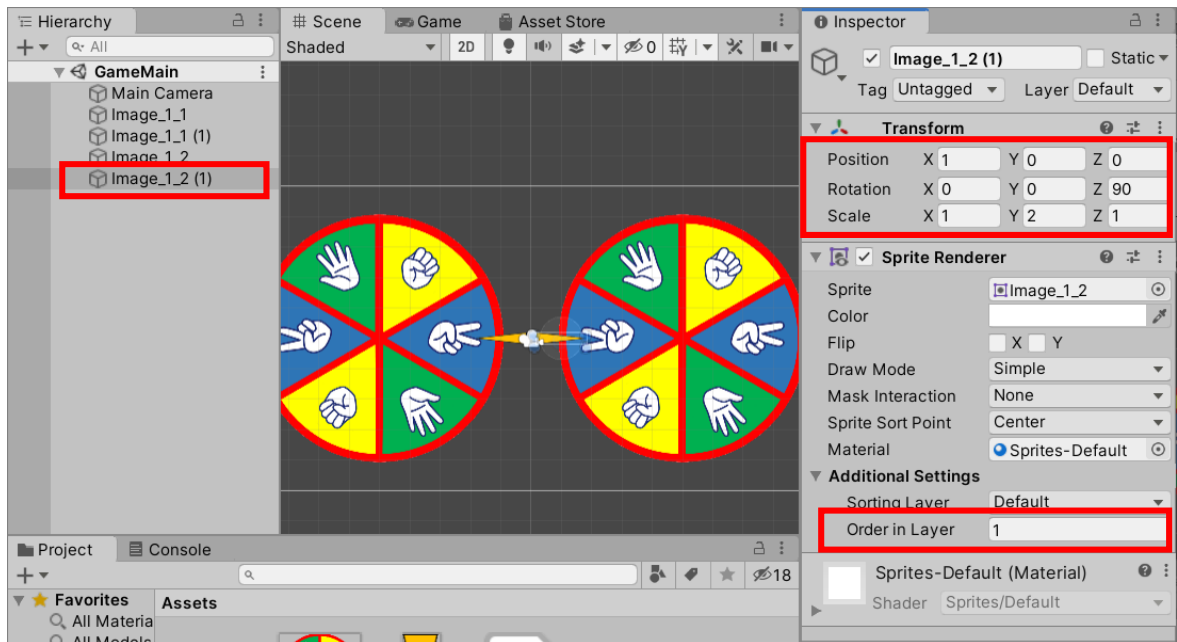
また、Sprite Renderer コンポーネントの Order in Layer を「1」にします。



次に右側の画像を調整します。Hierarchy ビュー上で「Image_1_2 (1)」を選択します。

Inspector ビュー上の Transform コンポーネントの Position を「X: 1, Y: 0, Z: 0」に、Rotation を「X: 0, Y: 0, Z: 90」に、Scale を「X: 1, Y: 2, Z: 1」にします。

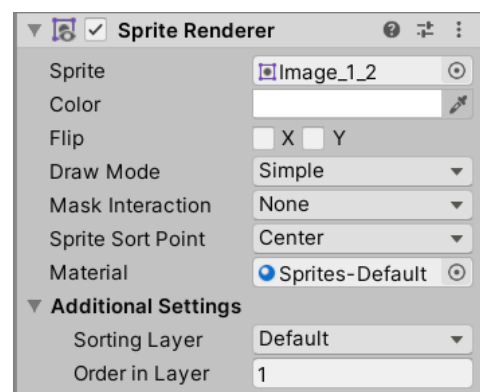
また、Sprite Renderer コンポーネントの Order in Layer を「1」にします。



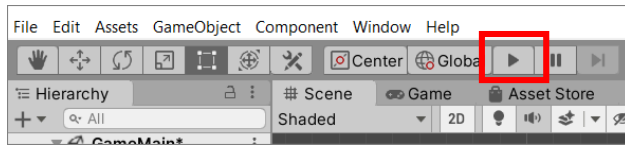
【参考】【Sprite Render の Order in Layer】

【Sprite Render の Order in Layer】

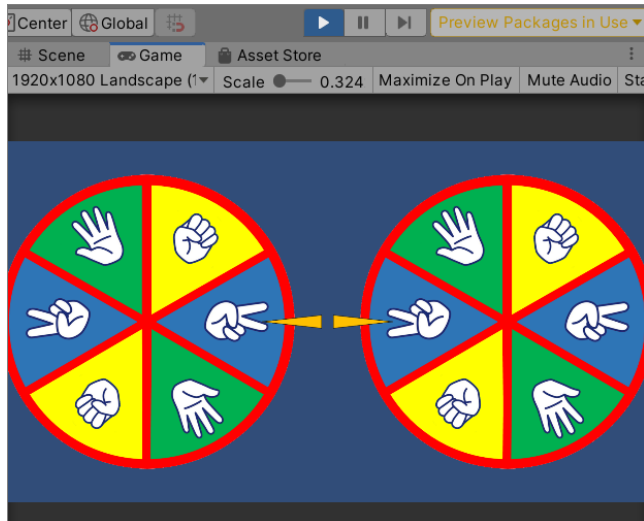
先ほど設定したじゃんけんルーレットでは設定しませんでした。これは画像を画面に表示する順番を示す数字です。数字が大きくな方が前面に表示されます。ルーレットには初期値「0」が入っていたので、前面に表示させる針に「1」を入力したわけです。



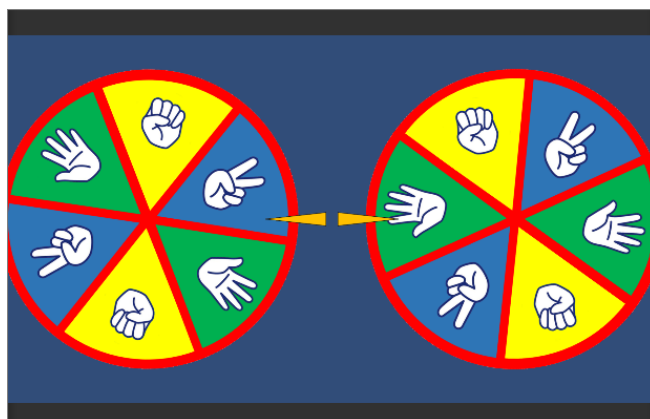
では、ゲーム実行ボタンをクリックしてテストをしてみましょう。



Game ビューに実行されているゲームが表示されました。



キーボード上のスペースキーで左側のじゃんけんルーレットが、右矢印キーで右側のじゃんけんルーレットが回転し始め最後には止まりました。これでじゃんけんバトルの完成です。



じゃんけんバトルの
ゲームの完成だよ。

友達と実際に遊んで
みよう。



【参考】【プログラムの詳細説明】



【プログラムの詳細説明】（RouletteController）

```

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

////////////////////////////////////
// 【ルーレットのコントロールに使用するクラス】
// ルーレットをキーボード処理に合わせて回転させ始め、停止させる処理
////////////////////////////////////
public class RouletteController : MonoBehaviour
{
    //////////////////////////////////////
    // 【変数の設定】
    //////////////////////////////////////

    // 【Unity Editorから設定するPublic変数】
    public int roulette_No; // どちらのルーレットかを示す番号（左が1、右が2）

    // 【このプログラム内で使用するPrivate変数】
    private float rotationAngle = 0; // ルーレットが回転する角度（回転スピード）

    //////////////////////////////////////
    // 【関数の設定】
    //////////////////////////////////////

    // 【毎フレームに1度動く関数】
    // （Unityの標準関数）
    void Update()
    {
        // ルーレット番号が1でこのプログラムと関連付けられている画像について、
        // スペースキーが押された場合
        if (roulette_No == 1 && Input.GetKey(KeyCode.Space))
        {
            // -180度から180度の間の角度でランダムな角度に一度回転する
            transform.Rotate(0, 0, Random.Range(-180, 180));
            // 回転スピードrotationAngleの初期値を20度にする
            rotationAngle = 20;
        }

        // ルーレット番号が2でこのプログラムと関連付けられている画像について、
        // 右矢印キーが押された場合
        if (roulette_No == 2 && Input.GetKey(KeyCode.RightArrow))
        {
            // -180度から180度の間の角度でランダムな角度に一度回転する
            transform.Rotate(0, 0, Random.Range(-180, 180));
            // 回転スピードrotationAngleの初期値を20度にする
            rotationAngle = 20;
        }

        // このプログラムが関連付けられているルーレットを
        // 回転スピードrotationAngleの角度だけ回転する
        transform.Rotate(0, 0, rotationAngle);

        // 回転スピードrotationAngleの角度に1未満の小数をかけて回転スピードを減速させる
        rotationAngle = rotationAngle * 0.999f;

        // 回転スピードrotationAngleが一定値未満になった場合
        // 止まったと考えて回転スピードrotationAngleをゼロにする
        // （これがないと見えないスピードだが永遠に回転が終わらない）
        if (rotationAngle < 0.01f) rotationAngle = 0;
    }
}

```

【練習問題】

- ① じゃんけんルーレットが止まるまでの時間をもっと長くしてみよう。

【ヒント】

じゃんけんルーレットの回り続ける時間は回転スピードに影響を受けます。そして、回転スピードに影響を与えるのは回転スピードの初期値と減速の割合です。

- ② 回転スピードの初期値は 20 度になっていますが、初期値をどんどん上げていくとどうなるか考えてみよう。

【ヒント】

この回転スピードは 1 フレームで画像が回転する角度のことです。回転スピードをあげていくと見た目はどんどん速くなっていきます。しかし、回転スピードが 180 度を超えると見た目はどうなるでしょうか。そして、360 度を超えるとどうなるでしょうか。

- ③ 同じスクリプトを使っている左右のじゃんけんルーレットが、なぜ最後に違う角度で止まるか考えてみよう。

【ヒント】

キーが押されて初期値がセットされるあたりを中心に調べてみよう。なぜ左右のじゃんけんルーレットの角度が変わっているのか。

スクリプトの詳細
説明も見ながら考
えてみよう。



【回答】

① じゃんけんルーレットが止まるまでの時間をもっと長くしてみよう。

ヒントにもあった回転スピードの初期値と減速の割合を変えてみましょう。

回転スピード rotationAngle の初期値は「20」でしたので、値をそれより大きくします。

```
rotationAngle = 20;
```

また、減速の割合は「0.999」なので、値をそれより小さくします。

```
rotationAngle = rotationAngle * 0.999f;
```

② 回転スピードの初期値は 20 度になっていますが、初期値をどんどん上げていくとどうなるか考えてみよう。

回転スピードは 1 フレームで画像が回転する角度のことです。回転スピードをあげていくと見た目はどんどん速くなります。

回転スピードの初期値 20 度だと 1 フレームで 20 度時計回りに回転します。数字が大きくなるとその 1 フレームで回る角度が大きくなり回転が速くなるように見えます。しかし、例えば 1 フレームで 270 度（180 度以上）時計回りに回転すると、見た目は逆方向に 90 度回転したように見えるでしょう。

また、1 フレームの回転速度が 360 度を超えた 450 度回転すると、 $450-360=90$ 度だけ回転したように見えるでしょう。

③ 同じスクリプトを使っている左右のじゃんけんルーレットが、なぜ最後に違う角度で止まるか考えてみよう。

Update 関数内の roulette_No が 1 と 2 に対する if 文の最初に以下のような関数があります。

```
transform.Rotate(0, 0, Random.Range(-180, 180));
```

Rotate という関数は画像を回転させます。その回転させる角度を Random.Range で指定しています。これはランダムな数字を指定した範囲（ここでは-180 から 180 の間）の数字をランダムに生み出します。よって、そのランダムな角度だけ最初にそれぞれの画像が回転するので左右のじゃんけんルーレットの止まる角度が変わります。



第1.4章. 簡単サンプルゲームで使った C#


前章で簡単なサンプルゲームと言ってもいきなりプログラムを書いてびっくりしたかもしれません。一応プログラムを書いてみてゲームは動いたけど、理解が出来てないという人も多いかと思います。

大丈夫です。このテキストではいったんゲームを作ってみて、そのゲームの作りを紐解きながら C#プログラムの基本を順に解説していきます。

【注意事項】

ここではほんとうにプログラムなんて見るのも初めてだったと言うような初心者の方にも分かるように解説します。

これくらいは分かっていると言う方は、どんどん先に進んで下さい。



まずは簡単サンプルゲームで書いた C#プログラムを理解していこう。

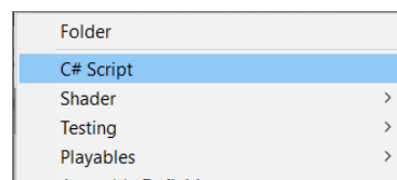
1.4.1. C#プログラムの基本

【C#プログラムを読むうえでの基本】

最初に説明しましたが、プログラムと言うのは細部まで理解することは大人でも難しい場合があります。難しいと思った場合は基本的に先に進んでも良いです。しかし、ここの話は最低限まず最初に理解しておきましょう。

【C#プログラムの基本中の基本】

- Unity エディタの中ではC#プログラムをスクリプト (Script) と記していました。スクリプトはプログラムのことと考えて構いません。



- プログラムは基本的に上から下に読んでいきます。
(後から説明する関数が呼ばれると、呼ばれ先の関数に処理が移動するので、基本は上から下に読みますが、関数を呼び出したらその関数の中の処理を読むこととなります。)
- プログラムは英語のアルファベットで書かれ、C#プログラムは大文字と小文字を別に扱います。
(初心者のよくやる間違いで、アルファベットの大文字と小文字を間違えて書いてしまうことがあります。テキストを良く見て確認しましょう。)

- プログラム内でのコメントアウト
(プログラム内に説明を書くため、処理を実行しないで無視する部分。

```
// 【毎フレームに1度動く関数】  
// (Unityの標準関数)  
void Update()  
{  
    // ルーレット番号が1でこのプログラムと関連付けられている画像について、  
    // スペースキーが押された場合  
    if (roulette_No == 1 && Input.GetKey(KeyCode.Space))
```

緑色の部分) は、行の初めに「//」と書く。

(今回のサンプルでは使ってないですが、「/*」と「*/」で囲んでもコメントアウトができます。)

- データとしてテキストを使用する場合、「"」で囲む。

```
name = "Ken";  
name2 = name + " san";
```

- 日本語はプログラム内では使えません。テキストデータとコメントアウトしている部分は使用可能です。

- C#プログラムは括弧（カッコ）がとても重要です。特に処理や機能のかたまりをまとめる波括弧「{ }」がサンプルプログラムにもたくさん出てきました。また丸括弧「()」もいくつか出てきました。これは後ほど解説する関数に引き渡す変数を指定するのに使います。

（初心者のよくやる間違いで、括弧の数が合わないと言うことがあります。閉じ括弧が足りない等です。括弧の数が合うようにテキストをよく見て確認しましょう。）

```
public class Test : MonoBehaviour  
{  
    void Start()  
    {  
        int a = 2;  
        int b = 5;  
  
        int c = AddInteger(a, b);  
  
        Debug.Log(c);  
    }  
  
    int AddInteger(int x, int y)  
    {  
        int z = x + y;  
  
        return z;  
    }  
}
```

- 1つの処理の最後は「;」で終わります。処理を改行して2行にして、2行目に「;」が出てきても構いません。

（初心者のよくやる間違いで、「;」の書き忘れがあります。絶対必要な記号です。テキストをよく見て確認しましょう。）

- プログラムは基本的に変数と関数と制御文でできています。まず変数と関数は基本的な使い方を理解しましょう。制御文はいくつもあるので順番に覚えていきましょう。

（すでに変数と関数は何度も使っています。いくつかの制御文も既に使いました。これらは後ほど解説します。）

- スペースや改行は基本的にプログラム処理としては無視されます。プログラムが読みやすいように自由に使うことができます。

まず、プログラムを読む
むうえでの超基本は抑
えておこう。



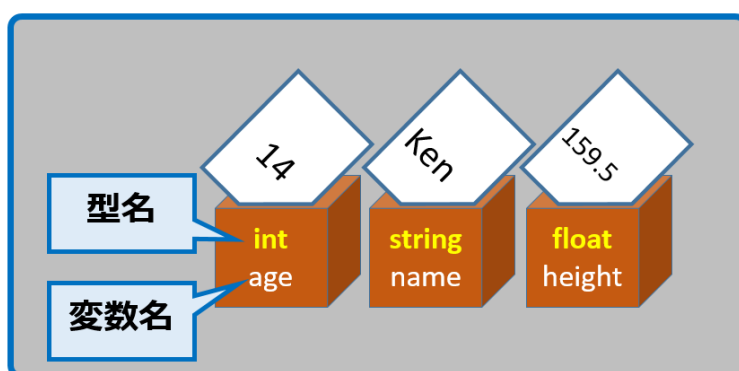
【変数とは】

プログラムの中で数字や文字を使ってコンピューターやスマートフォンに指示をすることがあります。その場合、「変数」というものがよく使われます。

中学生の数学で X や Y を使った数式を解くと思いますが、それと同じようなものです。

変数とは数字や文字をしまっておく箱のようなものとも言えます。プログラム内の後で使う数字や文字を入れておく箱です。

変数を使う場合、どのようなものを入れる箱なのかを最初に宣言する必要があります（型名）。整数をいれるのか、少数をいれるのか、文字を入れるのか。また、その変数の名前も宣言します（変数名）。



プログラム内での変数の宣言と、変数への代入は以下のように行います。

【変数の宣言方法】

型名 変数名;

【変数への代入方法】

変数名 = 代入する値;

また、C#では、この型名としていろいろな型を指定できます。以下にいくつかのものを挙げておきます。変数の型はこれ以外にもたくさんあります。

覚えるのが大変ですね。しかし、大丈夫。よく使うのは int, float, bool, string くらいです。これでだいたいのプログラムは書けてしまいます。

【変数の型の種類】

型名	型の説明	数値の取りうる範囲
int	整数型	-2,147,483,648 ~ 2,147,483,647
float	浮動小数点型	-3.402823E+38 ~ 3.402823E+38 (有効桁数 7 桁)
double	倍制度浮動小数点型	-1.79769313486232E+308 ~ 1.79769313486232E+308 (有効桁数 15~16 桁)
bool	ブール型	true (真) 又は false (偽)
char	文字型	1 文字のテキスト
string	文字列型	可変長のテキスト

【変数を使ってみよう】


では、実際に変数を使ったプログラムを見てみましょう。自分で試せる人は Unity でこの C#プログラムを作成してみましょう。

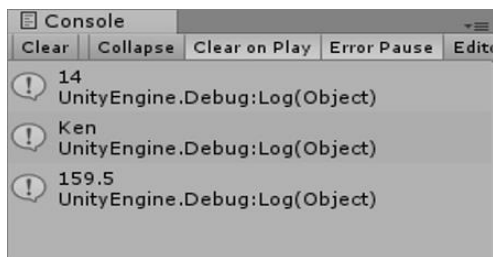
```
5 public class Test_01 : MonoBehaviour
6 {
7     void Start ()
8     {
9         int age;
10        string name;
11        float height;
12
13        age = 14;
14        name = "Ken";
15        height = 159.5f;
16
17        Debug.Log(age);
18        Debug.Log(name);
19        Debug.Log(height);
20
21    }
22 }
23 }
```

変数の宣言

変数へ値の代入

変数の値を Consoleビューに出力

では、ゲーム実行ボタン  を押し、ゲームを実行してみましょう。
 Console ビューに以下の様に変数内の数字や文字が表示されました。



次に、もう少し C# プログラム内の変数をいじってみましょう。

```


5 public class Test_01 : MonoBehaviour
6 {
7     void Start ()
8     {
9         int age, age2;
10        string name, name2;
11        float height, height2;
12        double d_height, d_height2;
13
14        age = 14;
15        age2 = age / 3;
16
17        name = "Ken";
18        name2 = name + " san";
19
20        height = 159.5f;
21        height2 = height / 3;
22
23        d_height = 159.5;
24        d_height2 = d_height / 3;
25
26        Debug.Log(age2);
27        Debug.Log(name2);
28        Debug.Log(height2);
29        Debug.Log(d_height2);
30    }
31 }
32
33
    
```

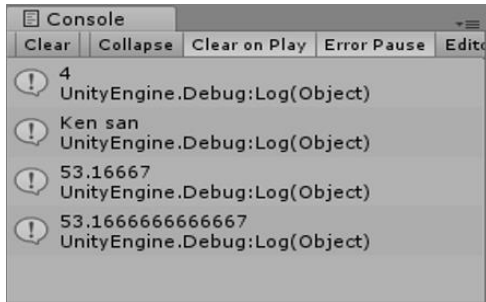
変数の宣言

変数へ値の代入

変数の値を Consoleビューに出力

これを実行する前に一度出力結果を自分で予想してみましょう。

では、ボタン  を押し、ゲームを実行してみましょう。予想通りでしたか？



変数はプログラム内で使う
データを入れておくための
箱ということだね。



【関数（メソッド）とは】

プログラム内にはアプリに対して実行する処理を書いていくわけですが、本格的なプログラムだと処理の記述が続く長い長いプログラムになることが予想できます。

そんな場合、少し小さな目的を実現する処理の流れをブロックとして切り分けます。

例えば皆さんが通学するという行動の流れを細かく書くと、「靴を履く」「玄関を開ける」「玄関を閉める」「鍵を閉める」・・・とすごく長い行動の流れが書けるでしょう。しかし全体が分かりづらいです。

では、通学するを「家を出る」「駅に向かう」「電車で学校のある駅に移動」「学校に向かう」と言ったように小さな目的の行動ブロックを作ると全体が分かりやすいです。細かくはそれぞれのブロック内に書きます。

この小さな目的を実現する処理の流れのブロックをプログラムでは関数（メソッド）と言います。要は長くなった処理の流れを意味のある処理のブロックごとに名前を付けて分解する仕組みのことです。

こうすることにより、プログラムを書いた本人も後から読むのがとても楽になります。

また、その処理のブロックの再利用も可能です。要は同じ処理を何度も行う必要がある場合、その処理を関数（メソッド）にまとめてしまえば、その処理を行う際、毎回処理内容を書く必要なく関数（メソッド）を呼ぶだけで済みます。

【関数（メソッド）のメリット】

- ✓ 処理の流れが意味のある処理ブロックとしてまとまっているので読むのが楽。
- ✓ 同じ処理を何度も行う場合、毎回同じ処理を書く必要がない。

そのため、関数（メソッド）を使用することはプログラムを読むにも、書くにもとても有効な手段と言えます。

（この後は「関数（メソッド）」を「関数」とだけ記述します。）

関数を使えばプログラムを読むのも書くのも楽になるよ。



【実際の関数を見てみよう】

① Unity の標準関数

実際の関数の例として、まず皆さんが Unity でプログラムを新規に作る際に自動で作られる C# プログラムの中身を見てみましょう。

```
5 public class Test : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
```

ここに以下のようなブロックがあります。

- `void Start () { }`
- `void Update () { }`

これらが関数です。既に簡単サンプルゲームでも出て来ました。簡単サンプルゲーム内ではこのブロック（括弧 { } 内）の中にいろいろな処理を書いていきました。

（これらの Start と Update 関数自体は次の章で詳しく説明します。）

これらは Unity で事前に定義されている関数です。Unity 側で決まったタイミングで呼び出されて、そのタイミングにこのブロックの中に書かれた処理が実行されます。

Unity の標準関数として初心者がまず使って、今後も使い続けるのがこの Start と Update 関数です。これら以外にもとても多くの標準関数が Unity には用意されています。

それらはサンプルゲームに出てきたタイミングで順番に覚えて行けば良いです。プロのゲーム開発者でも全てを覚えている人はいないと思います。（要は全て覚える必要はありません。）

② 自分で作成する関数（引数・戻り値がある場合）

関数は自分で自由に作ることもできます。自分で関数を作成する場合、その宣言の仕方と、その宣言した関数の呼び出し方法は以下のようになります。

【関数の宣言方法】

```
戻り値の型 関数名 (引数の型 引数名)
{
    処理;
    return 戻り値;
}
```

【関数の呼び出し方法】

```
関数名 (引数);
```

やはり実際に見てみないと分かりづらいので、事例を見てみましょう。簡単サンプルゲーム内では自分で関数は作りませんでしたので、簡単な例を作ってみましょう。

```
5 public class Test : MonoBehaviour
6 {
7     void Start()
8     {
9         int a = 2;
10        int b = 5;
11
12        int c = AddInteger(a, b);
13
14        Debug.Log(c);
15    }
16
17    int AddInteger(int x, int y)
18    {
19        int z = x + y;
20
21        return z;
22    }
23 }
```

関数の呼び出し

関数の宣言

この中の関数の宣言と関数の呼び出し部分を説明します。

まず関数の宣言は以下の部分です。「AddInteger」という関数名で宣言されています。そして int 型（整数型）の引数名「x」と「y」という引数を受け取る形になっています。

(注) 後で説明する呼び出し側で「2」と「5」を渡してくるので、ここでは「2」と「5」を引数として受け取ることになります。

処理としてはその引数「x」と「y」を足して「z」に代入。その int 型（整数型）の「z」を戻り値として返します。

(注) ここでは計算結果「z」の値は「7」なので、「7」を戻り値として返します。

```
17     int AddInteger(int x, int y)
18     {
19         int z = x + y;
20
21         return z;
22     }
```

ではこの関数の呼び出し側を見てみましょう。ここで「AddInteger」という関数が呼ばれています。int 型（整数型）の「a」と「b」を引数として関数に渡しています。

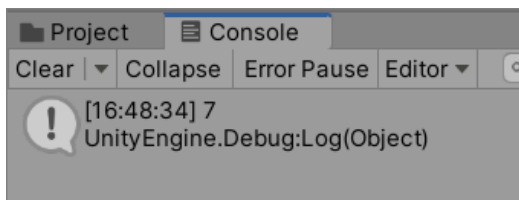
(注) 「a」と「b」を渡しているとは、整数である「a」と「b」の中身「2」と「5」を渡しています。

```
11
12     int c = AddInteger(a, b);
13
```

そして、「AddInteger」関数の計算結果を戻り値として受け取り、それを int 型（整数型）の「c」に代入しているわけです。

(注) ここでは関数が戻り値として整数「7」を返してくるので、7を変数「c」に代入します。

このプログラムの実行結果を見てみましょう。予想通り計算結果の「7」が表示されます。



関数の宣言の仕方と
呼び出し方法は覚えて
おこう。



③ 自分で作成する関数（引数・戻り値がない場合）

今見てみた自分で作成する関数には引数と戻り値がありました。しかし、実は引数や戻り値は絶対必要なわけではありません。処理によっては引数や戻り値のどちらか、又は両方とも必要としない場合もあります。

その場合の記述の仕方は以下のようになります。

【戻り値のない関数の宣言方法】

```
void 関数名 (引数の型 引数名)
{
    処理;
}
```

【引数のない関数の宣言方法】

```
戻り値の型 関数名 ()
{
    処理;
    return 戻り値;
}
```

戻り値のない場合は戻り値の型を書く部分に「void」、英語で空っぽという意味の言葉を付けます。

そして、引数のない場合は、関数名の後の括弧「()」の中に何も書かなければ良いだけです。

この良い例として、サンプルプログラム内でも使った Update 関数があります。Update 関数には引数がありませんでした。そして、戻り値の型を書く部分には「void」と書かれていましたね。

上記した通り、関数には引数があるのとない場合、戻り値があるのとない場合があります。よって、全部で 4 種類の関数があることになります。

1.4.2. Start 関数と Update 関数

【Start 関数と Update 関数とは】

既に Unity の標準関数の例として紹介しましたが、Unity でプログラムを新規に作る際に自動で作られる C# プログラムの中にある関数。

```
5 public class Test : MonoBehaviour
6 {
7     // Start is called before the first frame update
8     void Start()
9     {
10
11     }
12
13     // Update is called once per frame
14     void Update()
15     {
16
17     }
18 }
```

常に以下の Start 関数と Update 関数が作られます。

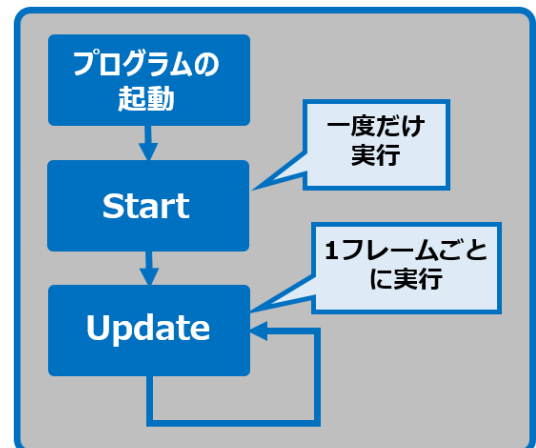
- `void Start () { }`
- `void Update () { }`

「Start 関数」には、プログラムが実行された際、最初に一度だけ実行される処理が書かれます。

(プログラムが関連付けられているゲームオブジェクトがシーン上で出現したタイミング。)

「Update 関数」には、プログラムが実行された後、毎フレーム実行される処理が書かれます。

(プログラムが関連付けられているゲームオブジェクトがシーン上で出現している間の毎フレーム。)



(自動で作られた Start 関数と Update 関数が必要のない場合は、削除しても問題ありません。)

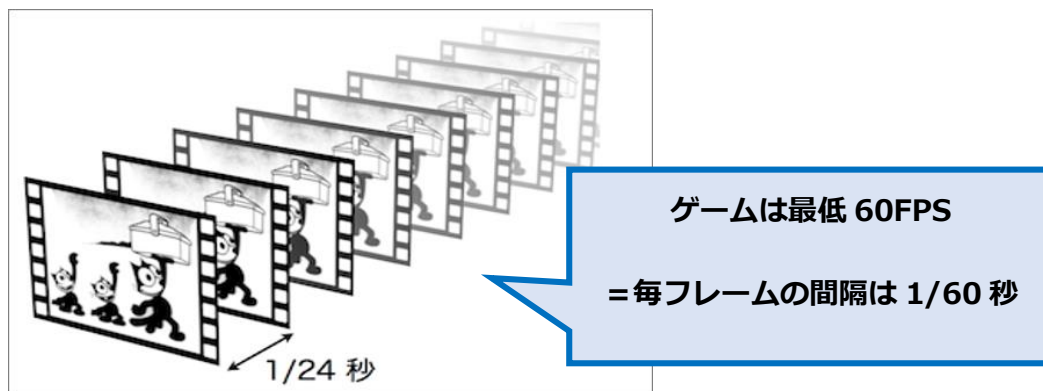
【フレームと FPS とは】

今 Update 関数の説明にあった「フレーム」。これを聞いたことがありますか？

このフレームとはゲーム上に表示される 1 コマの絵です。皆さんが遊んでいるゲームアプリとは、実際は映画やアニメと同じように 1 コマ 1 コマの絵のパラパラ漫画方式で表示されています。この 1 コマの絵の表示をフレームと呼びます。

1 秒間に表示されるフレーム数は「FPS」 (Frame Per Second) で表します。

通常映画では 24FPS (1 秒間に 24 フレーム表示) とされ、ゲームではだいたい最低 60FPS (1 秒間に 60 フレーム表示) あればキャラクターがかくかく動かないで、スムーズに動いているように見えます。



実際のゲームでは、スマートフォンやネットワークなどの性能や、ゲームの負荷により速くなったり遅くなったりします。

Start 関数と Update 関数は一番よく使う関数なので覚えてしまおう。



1.4.3. 制御文 : If 文

【制御文とは】

ここでは「制御文」を説明します。少し前にプログラムは変数と関数と制御文でできていると言いました。そうです、制御文はどんなプログラムを書くうえでもほぼ必須なものとなります。

(それはC#プログラミング言語に限らず、他の多くのプログラム言語にもあてはまります。)

このように重要な制御文なのですが、そんなに難しいものではありません。もう簡単サンプルプログラムでも制御文を何回か書いています。

では、この制御文とは何でしょう。基本的にプログラムとは上の行から下に処理が進んでいきます。しかし、それではあまりにも単純なプログラムしか書けません。全くゲームになりません。

例えばゲームであれば、あなたが右ボタンを押したらキャラクターは右に、左ボタンを押したら左に行く必要があります。要は状況によって、プログラムの流れや指示を変える必要があります。

それを行うのが制御文となります。制御文を使うことで、ある条件のときだけ ある処理を実行したり、繰り返し ある処理を実行したりすることができます。

今回は数ある制御文の中から、既に簡単サンプルプログラムで使用した制御文を取りあげます。「if」制御文です。

```
void Update()
{
    if (roulette_No == 1 && Input.GetKey(KeyCode.Space))
    {
        transform.Rotate(0, 0, Random.Range(-180, 180));
        rotationAngle = 20;
    }
}
```

【「if」制御文について】

「もし (if) あなたが左ボタンを押したら、キャラクターを左に 1 歩移動させる。」そんな指示に必要なのが if 制御文となります。英語の if (もし) と同じです。

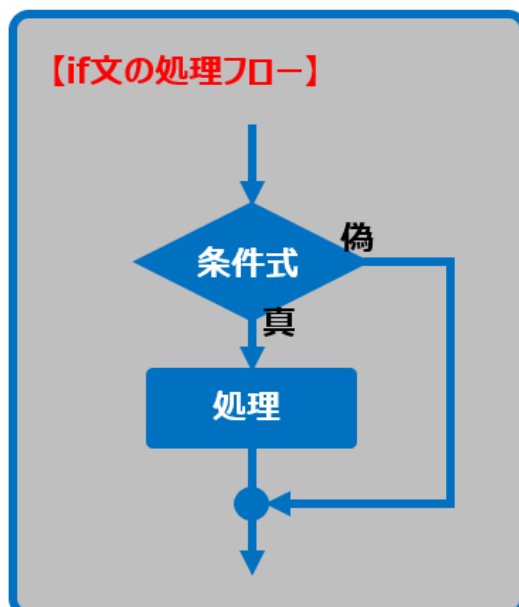
if 文の書き方にはいくつかの種類があります。まず最も簡単なパターンが以下です。

【if文の書き方】

```
if (条件式)
{
    処理;
}
```

これは、条件式に示した条件を満たした場合（条件が真の場合）には、{ } 内に書かれた処理を実行します。

条件式を満たさなかった場合（条件が偽の場合）には、{ } 内の処理は実行せずに次に進みます。



ここで、if 文内で使っている条件式に関して。この条件式の式は比較演算子を使って書くことができます。比較演算子には以下の様なものがあります。

【比較演算子の種類】

比較演算子	比較結果
==	左辺と右辺が等しい場合に真
!=	左辺と右辺が等しくない場合に真
>	左辺が右辺より大きい場合に真
<	左辺が右辺より小さい場合に真
>=	左辺が右辺以上の場合に真
<=	左辺が右辺以下の場合に真


では、実際に if 文を使ったプログラムを見てみましょう。

```
5 public class Test2 : MonoBehaviour
6 {
7     void Start()
8     {
9         int Num = 15;
10
11         if (Num > 8)
12         {
13             Debug.Log("if文内の処理実行");
14         }
15
16         Debug.Log("プログラム終了");
17     }
18 }
```

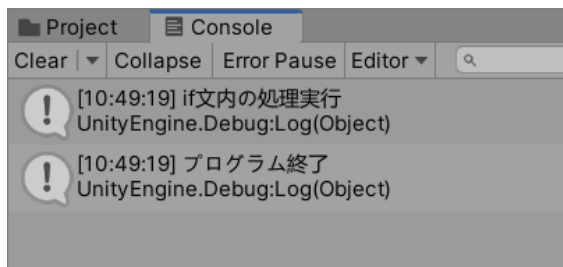
変数 Num の値が 8 より大きい
場合、{ }内の処理を実行。

実行結果を見る前にまず結果を予測してみましょう。

(「Debug.Log();」とは ()内のテキストを Console ビューに表示させるための命令です。)

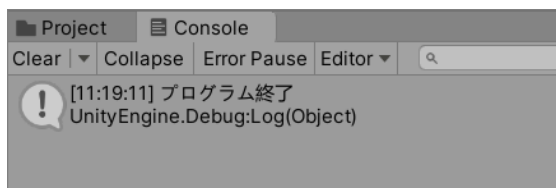
結果を予測できましたか？ では、 ボタンを押し、ゲームを実行してみましょう。

Console ビューに結果が表示されました。予測した通りになりましたか？



プログラム内の変数 Num の初期値を 7 にするとどうなるかも考えてみましょう。

ゲーム実行すると、Console ビューに以下の結果が表示されました。



【「if-else」と「else if」制御文について】

① if-else 制御文

次に、if文の応用として、if-else 制御文を使ってみましょう。

例えば「もし (if) あなたが左ボタンを押したら、キャラクターを左に 1 歩移動させる。そうでなければ (else) 真っ直ぐ前進する。」そんな指示も必要となります。

では、その書き方を見てみましょう。

【if-else文の書き方】

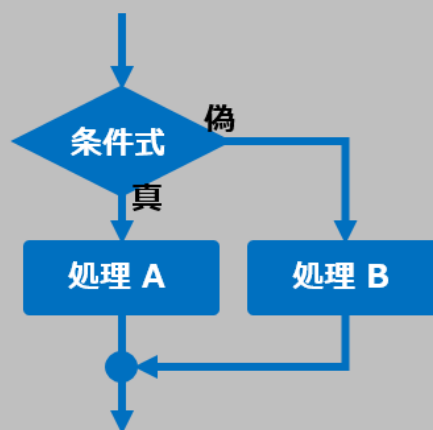
```
if (条件式)
{
    処理 A;
} else
{
    処理 B;
}
```

これは、条件式に示した条件を満たした場合（条件が真の場合）には、処理 A を実行します。

条件式を満たさなかった場合（条件が偽の場合）には、処理 B を実行します。

（よって、必ずどちらかの処理が実行されることになります。）

【if-else文の処理フロー】



② else if 制御文

また、if 文と else 文を組み合わせた else if 制御文というパターンもあります。

【else if 文の書き方】

```
if (条件式 A)
{
    処理 A;
} else if (条件式 B)
{
    処理 B;
} else
{
    処理 C;
}
```

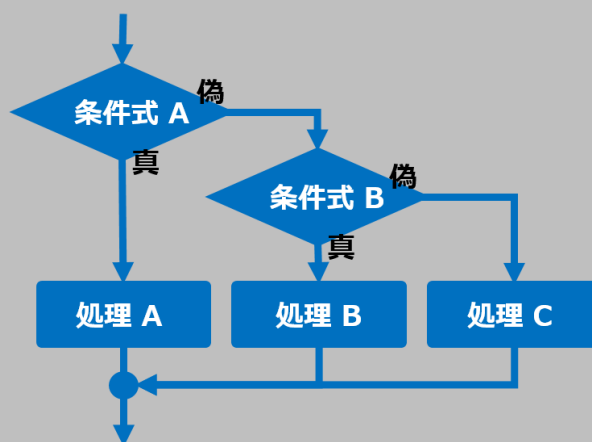
これは、条件式 A を満たした場合
には処理 A を実行します。

条件式 A を満たさなかった場合で
条件式 B を満たした場合には処理
B を実行します。

条件式 A、条件式 B も満たさな
かった場合には処理 C を実行しま
す。

(これも必ずどれかの処理が実行される
ことになります。)

【else if 文の処理フロー】



この else if 文が理解でき
れば if 文はもう完璧だ。



では、また実際に else if 文を使ったプログラムを見てみましょう。

```

5 public class Test2 : MonoBehaviour
6 {
7     void Start()
8     {
9         int Num = 1;
10
11         if (Num < 1)
12         {
13             Debug.Log("処理A");
14         } else if (Num > 1)
15         {
16             Debug.Log("処理B");
17         } else
18         {
19             Debug.Log("処理C");
20         }
21
22         Debug.Log("プログラム終了");
23     }
24 }

```

変数 Num の値が 1 より小さい場合、すぐ下の { } 内の処理を実行。

Num の値が 1 より大きい場合、すぐ下の { } 内の処理を実行。

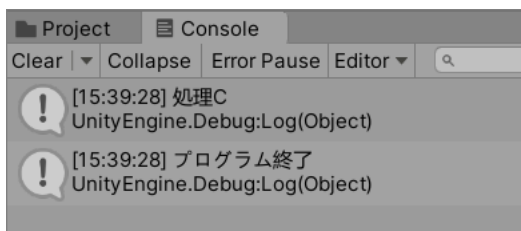
上記以外の場合、すぐ下の { } 内の処理を実行。

実行結果を見る前にまず結果を予測してみましょう。

(「Debug.Log();」とは () 内のテキストを Console ビューに表示させるための命令です。)

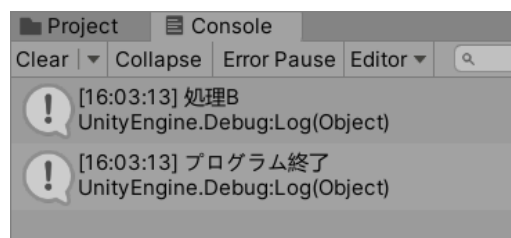
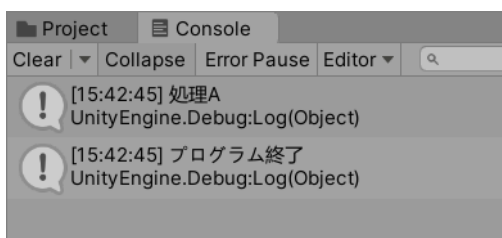
結果を予測できましたか？ では、▶ ボタンを押し、ゲームを実行してみましょう。

Console ビューに結果が表示されました。予測した通りになりましたか？



プログラム内の変数 Num の初期値を 0 又は 2 にするとどうなるかも考えてみよう。

ゲーム実行すると、Console ビューにそれぞれ以下の結果が表示されました。



第2章の予告

ここまで、どうでしたか？難しかったですか？

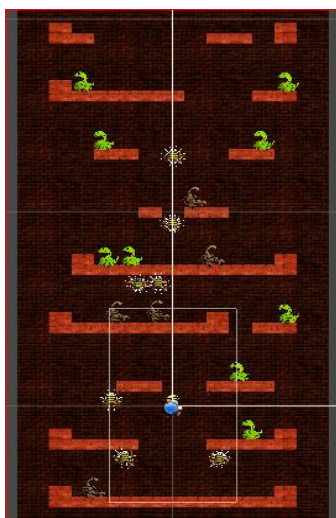
第1章はプログラムとは と言うところから始まり Unity の基礎に関して学びました。そして、Unity を実際に使って簡単サンプルプログラムを作成し、その後そのサンプルプログラムで使用した C#プログラムの基礎を学習しました。

ここまで理解できていればかなり Unity に詳しくなっていると思います。しかし、簡単サンプルプログラムはゲームとしては少しシンプル過ぎと感じたかもしれません。

次からは本章の Unity の学習法でも説明した通り、どんどん本格的なサンプルプログラムを作っていきます。少し手を入れたり、レベルを増やせば自分の本格的なゲームにもなりますし、同じジャンルのゲームを作る際の参考にもできるでしょう。ぜひどんどん多くのジャンルのゲーム作成にトライしてみてください。

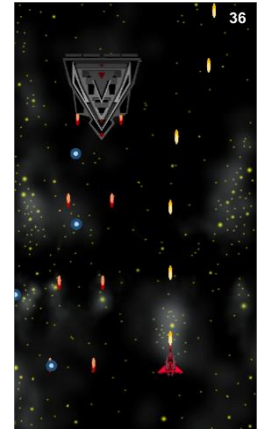
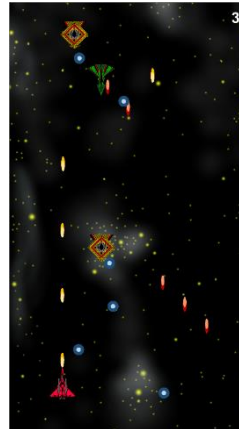
第2章.

Unity で「2D 縦スクロール アクションゲーム」を作ってみよう



第3章.

Unity で「2D 縦スクロール シューティング」を作ってみよう



第4章.

Unity で「2D スライドパズルゲーム」を作ってみよう



第5章.

Unity で「2D トップビュー アクション RPG」を作ってみよう

第6章.

Unity で「2D サイドビュー タワーディフェンス」を作ってみよう

第7章.

Unity で「2D マルチプレイ テーブルゲーム」を作ってみよう

第8章.

Unity で「2D トップビュー 戦略シミュレーション」を作ってみよう

この後も各種ジャンルのゲームを企画中。あなたの希望ジャンルも教えてね。

ぜひあなたが作ってみたいゲームジャンルを
教えてね。



作者紹介

志知 淳一 (プログラマー、ゲームデザイナー)



Lab7 としてゲーム開発の企画、デザイン、開発を行っている。もちろん Unity を使用。

そして、プログラミング教育にも取り組んでおり、子供が熱中できるゲームとプログラミングの組み合わせに注目。子供向けの Unity でのプログラミング教室を実施中。

また、最新技術として AR と AI に関心があり、プロダクトやサービスの企画・開発も行っている。

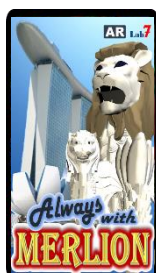
シンガポールに在住 10 年以上。

【ブログ】

<http://blog.lab7.biz/>

【ホームページ】

<https://www.lab7sg.com/>



@Junichi_SG_Life

(軽い海外生活ネタ)

- ・海外での仕事・生活
- ・勉強(英語/中国語)・自己啓発
- ・初心者向けプログラミング

@ShichiJunichi

(専門技術的なネタ)

- ・AR/VR/MR・AIの技術/製品情報
- ・UnityでのAR/ゲーム開発
- ・Unityでのプログラミング教育

@Junichi_SG_Game

(ゲーム情報のネタ)

- ・ゲームやゲームイベント情報
- ・海外のインディゲーム情報
- ・ゲーム実況者・ユーザ・開発者