基本テキスト
0章

ゲームプログラミングをはじめよう

ver.1.0.0

この章はこれからGodotエンジンをつかってゲーム制作をする前に読んでおいてもらいたい、学習の進め方や基礎的な知識が書かれています。

バージョン4.2.1をベースに作成し、4.3.0にも対応しています。

本書はGodotエンジンの普及、発展を目的に
プログラボ教育事業運営委員会が制作したものです。

本書はインターネット上で無料公開しておりますが、
著作権は放棄しておりません。
無断での転載、複製、配布、改変を固くお断りいたします。
商業目的での利用は、事前に著者の許可を得てください。

Copyright © ProgLab All rights reserved.

0-1 ゲーム制作をはじめよう

Copyright © ProgLab All rights reserved.

この文書を読んでいただきありがとうございます。この基本テキスト0章では、これからゲーム制作をはじめる皆さんに、良い準備をしてもらえるようなことを書いています。

■ゲームをつくろう

ビデオゲームの歴史はさかのぼれば1950年代からスタートしています。日本では1978年に株式会社タイトーの「スペースインベーダー」がアーケードゲームとして大ヒットし、1983年には任天堂から「ファミリーコンピュータ」が販売され家庭用ゲーム機がぐっと身近なものになりました。

今では家庭でもゲーム専用機やPCを使って世界中で開発・販売されているたくさんのゲームを楽しむことができます。

もちろん皆さんも最新のゲームをプレイしたことはあると思いますが、特にAAAタイトルとよばれる大規模開発されているゲームでは、何年もの期間をかけて、数百～数千人規模の人達が、数十億～数百億円の開発資金を使って開発しています。

「最新の◎◎みたいなグラフィックがすごくて、オンライン対戦もできて・・・そんなゲームをつくりたい！」となる気持ちも分かりますが、おそらく1人では何年かかっても完成しませんので、いったんそれは置いておきましょう(笑)

■インディーゲーム

明確な定義は難しいのですが、個人や小規模で開発されたゲームを「インディーゲーム」と呼びます。個人開発でも今ではプロが使うものと同じような開発ツールを使うことができますので、パッと見ただけでは違いが分からない場合もあります。

皆さんがゲーム制作をはじめる目的は様々だと思いますが、もし将来的に自分のオリジナルゲームをつくって販売までしたいと考えているなら、インディーゲームはとても参考になるでしょう。

大規模なゲームショウイベントだけでなく、日本各地で様々なゲームイベントが行われていて、インディゲームに触れる機会もたくさんあります。ぜひ足を運んで、開発者の方々と実際に会ってみるのもいいでしょう。

■総合芸術

ゲーム制作はプログラミングだけでなく、グラフィックやサウンド、ストーリー、その他たくさんの技術や表現が合わさったものづくりであり、総合芸術と言えます。

ゲーム制作を通じて様々なことを学び、興味関心の幅を広げて欲しいと思います。もしかしたらプログラミングよりもCG制作や、音楽、アニメーションなどに興味がわくかもしれませんね。

Copyright © ProgLab All rights reserved.

学習の進め方（１）

これからGodotを使ってゲーム制作をはじめるにあたって、どのように学んでいくか、ゲームプログラミングラボの特徴や教材の使い方などについて記載します。

■ゲーム制作を楽しもう

Godotエンジンでゲーム制作をするにあたって、大きくはGodot自体のツールとしての使い方とプログラミングについて学んでいく必要があります。

Godotはとても強力な開発環境ですので、全てを理解することは難しく（その必要もありません）、まずは小さなゲームをつくりながら、ツールの使い方とプログラミングに慣れていくのが良いでしょう。

各コンテンツの動画を見ながら制作を進めていきますので、初めのうちはどうしてもいわゆる「写経」のような学習スタイルになります。

しかし、まずは実際に手を動かしゲームを完成させ、達成感を得ながらゲーム制作を楽しめることが大事です。

どんなことでもそうですが、何よりも**楽しむことが上達の近道**です。興味を持って取り組めば、自然とスキルも伸びていきます。

■プログラミングについて

Godotで基本的に使用される言語「GDScript」はPythonに似ています。まだタイピングやコードを書くことに不慣れな場合は、Pythonの基本を学習しておくのも良いでしょう。

プログラミング学習を進めていくと、いかに効率よく書けるようにするか、メンテナンスしやすいコードにするかなどについても学んでいきます。

ただしメンテナンス性を重視するとかえってコードが長く複雑になることもあるため、各コンテンツではコード量が長くなることで学習者に負担にならないように、あえて短いコードでシンプルにしている個所もあります。

少し乱暴に言うと、まずはどんなコードの書き方でも良いので、ゲームがきちんと動いて楽しく遊べるものをつくることを目的として、はじめのうちは細かいことは気にせずにゲームを完成させることを大事にしましょう。

プログラミング上達のコツ
- 数をこなす（コピペではなく自分で入力）
- 慣れてきたらコードの意図を意識しながら書く
- コードを改造して動作の違いを知る
- エラーから学ぶ
  →エラーを何度も起こすぐらいいろいろなことを試す

Copyright © ProgLab All rights reserved.

■改造する

各学習コンテンツでは、ゲームの基本部分が完成したら改造してゲームにオリジナル要素を加えることができるようにガイドも用意されています。

説明通りにつくっただけではなかなか自分のものになりません。例えばレゴブロックで遊んだことがある方も多いと思いますが、いったん説明書通りにつくった後に、パーツを付け替えたり手持ちのパーツを加えて改造したり…その繰り返しをするうちに、徐々にオリジナル作品がつくれるようになっていきます。

それと同じで、まずはキャラクターのパラメーターを変えてみる、スピードを速くしてみる、などでも良いので、プログラムをいじってみたことでどのように変化するのかを確認してみましょう。

それができたら、ゲームの要素を追加したり、音やエフェクトで演出を加えたり、オリジナルの画像素材をつくったり、自分なりの表現手段を加えていけるようになりましょう。

ただし、はじめからそういった「後からでも良いこと」を優先してしまうとゲームが完成しなくなります。まずは学習コンテンツ通りにつくり、それから改造する手順を忘れないようにしましょう。

■遊んでもらう

ゲームができたら家族や友人、誰でも良いのでぜひ遊んでもらうようにしましょう。これには人に楽しんでもらうことから得られる達成感や喜びを感じられることはもちろんですが、ゲームをプレイした感想＝フィードバックを得られることも重要なことです。

すぐに操作に慣れるのか、難易度はどうか、楽しんでもらえているか…自分ひとりで制作、プレイしているだけでは分からなかったことがたくさん発見できます。

また誰かとチームを組んだり、学校や企業で活動しなければ、基本的には自宅で一人でゲーム制作学習を進めていくことになりますので、自分自身のモチベーションを保つ意味でも、作品を仕上げて遊んでもらうサイクルをつくるようにしましょう。

Godotは、ゲームをエクスポートしてWEBブラウザで遊べるようにしたり、スマホやタブレットに書き込んでプレイすることもできます。
ぜひたくさんの人に遊んでもらえるようなゲームづくりを目指しましょう。

Copyright © ProgLab All rights reserved.

資料の探し方

基本テキストや、ゲームごとの動画、テキストはありますが、より詳しい知識を知りたい場合や、新しいことを学びたい場合は、自分の力で情報を取得できるようになることが大事です。

■公式リファレンス

https://docs.godotengine.org/ja/4.x/
インターネット上の公式リファレンスには、言語仕様や様々なドキュメントが記されているので、分からないことがあったらまずは読んでみましょう。

ただし一部は翻訳されていても基本的には英語ですので、翻訳サービスなども使う必要はあります。

■コミュニティ

https://godotengine.org/community/
オフィシャルのものと、ユーザーによって管理されているものがあります。様々なメディアで展開されていますが、英語でのコミュニケーションが必要です。

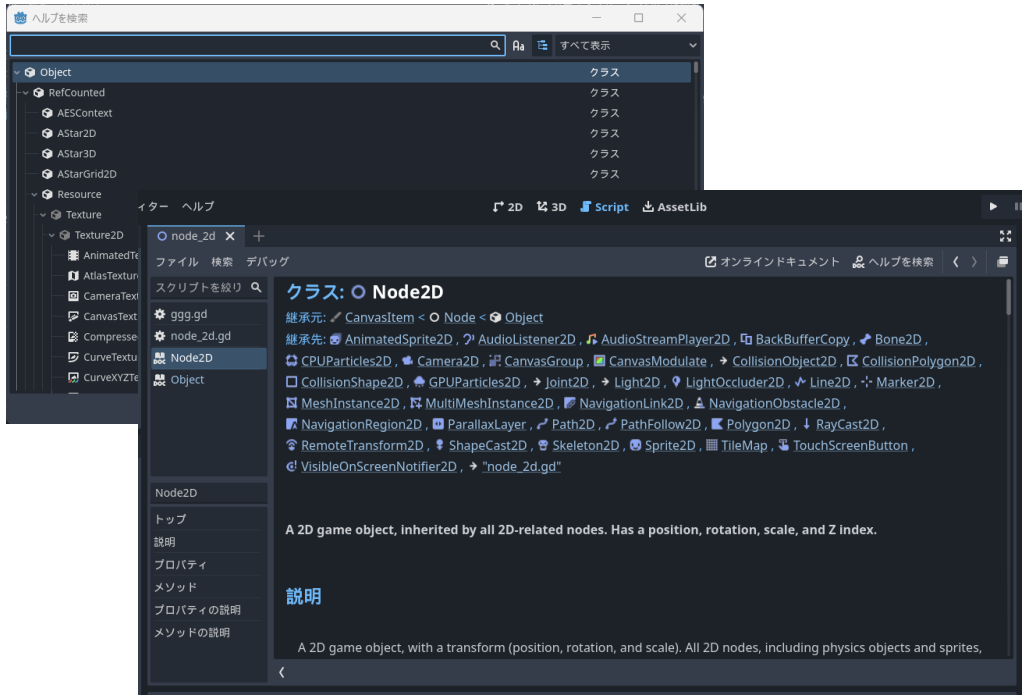
■ネット検索、YouTube

もちろんネット検索やYouTube動画などで得られる情報は役立ちますが、古いバージョンの情報が混ざっていることも多く、情報をきちんと選別できるようにしましょう。

■ヘルプメニュー

[ヘルプ] > [ヘルプを検索]

アプリケーション内のヘルプメニューからも、各ノードのリファレンスを確認できます。



ヘルプも内容はScript画面に表示されます。

■生成AI

ChatGPTに代表される生成AIもGodotの知識を教えてくれます。ただし間違いや古いバージョンのことが出力されることも多々ありますので、参考程度にするのが良さそうです。

Copyright © ProgLab All rights reserved.

素材について

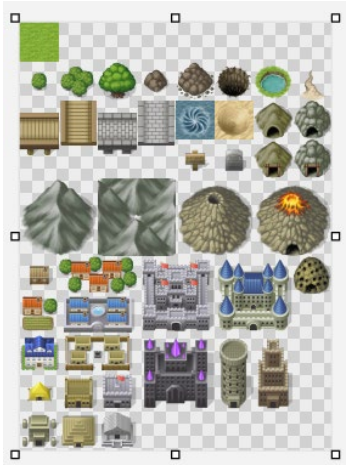
ゲームを作る際に、一から自分でキャラクターや背景の素材を描くことも楽しいですが、絵にこだわってしまうとゲームの完成が遠のいてしまいます。

まずゲームを完成させるために素材を利用するのが良いでしょう。インターネット上には無料・有料の素材がたくさんあります。特に2Dゲームでは**ドット絵の素材**が良く使われます。

■スプライトシート

2Dゲーム用の素材として用いられるものが、スプライトシートです。
図のようにいくつかの絵がタイル状にひとまとめになった画像です。

特にプレイヤーや敵キャラなどを表現するために使われるものは、キャラチップや歩行グラフィックとも呼んだりします。
RPGなどがイメージしやすいですが、四方向へ歩くアニメーションが作れるように素材が用意されています。

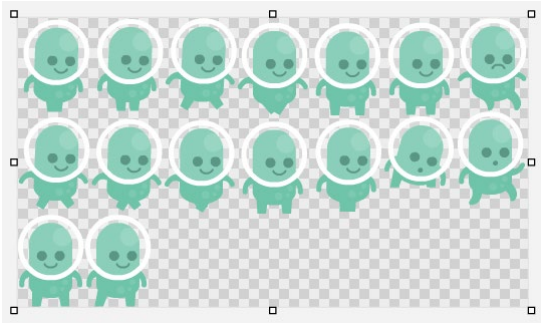


マップの素材なども同じようにマップチップで各タイル画像をまとめて管理します。

ぴぽや倉庫
https://pipoya.net/

■その他の素材

2Dゲームでも横スクロール型アクションゲームに代表されるものは「プラットフォーマー」というジャンル名で呼ばれていて、歩く、走る、ジャンプ、攻撃など多彩なアクションがあるため、素材画像もたくさんの画像が必要になります。



kenney
https://www.kenney.nl/

■素材利用の注意

インターネット上で配布されている無料の素材でも、それぞれに利用規約があります。たいていのものは自由に使えますが、しっかり規約を読んでから利用するようにしましょう。

また、無料のものでも寄付をして開発者を支援することもできます。いつかお金を支払えるようになったら寄付をするのもいいですし、自分が開発者側になって、素材を他の人に使ってもらえるようにするのもいいでしょう。

Copyright © ProgLab All rights reserved.

0-2 Scratch経験者へ向けて

- Scratchとの違い（１）
- Scratchとの違い（２）
- Scratchとの違い（３）
- Scratchとの違い（４）

Copyright © ProgLab All rights reserved.

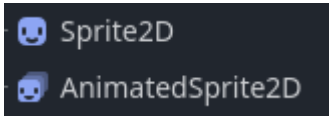
Scratchとの違い（１）

これまでにScratch（スクラッチ）でゲーム制作を経験された方も多いと思います。Scratch自体はゲームエンジンではないですが、ゲームを簡単につくることができる機能が豊富にあります。
ここからはScratchと比較しながら、Godotでのゲーム制作を理解しやすいよう解説します。

■スプライト

Scratchでは画面上に登場するキャラクターや障害物などは「スプライト」をつくり、それぞれにプログラミングをします。

スプライト自体はScratchに限った機能ではなく、コンピューターゲーム全般で使われる、背景画像とは別に個別に画面表示やプログラミングができるしくみです。



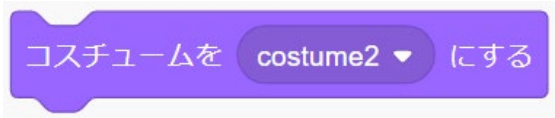
GodotではSprite2DやAnimatedSprite2Dがスプライトと名前がついていますが、これらは画像を表示させるために使われ、ゲーム内で必要とされる機能によって、他のノードと組み合わせて使われます。

Scratchにはあらかじめたくさんのスプライトが用意されているので、絵柄にこだわらなければ、すぐに値用できるのが利点です。

■コスチューム

Scratchのスプライトには「コスチューム」の名前で、スプライト画像を切り替えるしくみがあります。歩くアニメーションや、ダメージを受けた時の表現など、コスチュームを使って簡単に表現することができます。

Godotでも同じように、あらかじめ複数のパターンの画像を用意しておけば、任意のタイミングで切り替えて表示することでアニメーションをさせたり、状況によって画像を変えることができます。



ただし、GodotにはScratchのように簡単に絵を描く機能はありませんので、自分でオリジナルの画像をつくりたい場合は、別途画像処理ソフトで描く必要があります。

GIMP（ギンプ：https://www.gimp.org/）は無料で使えて高機能のアプリケーションとして有名ですので、使えるようになっておくのも良いでしょう。

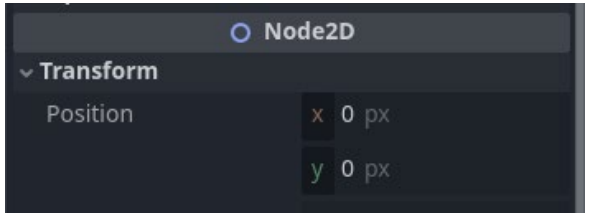
Copyright © ProgLab All rights reserved.

■10歩動かす

Scratchで誰もが一番初めに使うプログラムがこれではないでしょうか。キーボード入力と組み合わせて、ネコのスプライトを動かしたことがあると思います。

単位が「歩」となっていますが、これは座標を使わずに移動を表現するためのもので、実際にはScratch内の座標上での移動処理をしています。
座標で指定するプログラムもありますね。

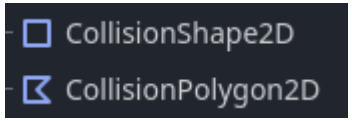
Godotでも同じように画面上のキャラクターの位置を、x方向、y方向に増減させることで移動をさせることができます。
Node2Dのプロパティを持つノードではこのように、
Positionプロパティでx、yの座標値をもっています。



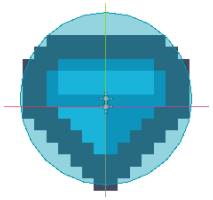
■当たり判定

Scratchでスプライト同士の当たり判定を行うには、色や他のスプライトに触れたことを判定する処理があり、制御処理と組み合わせて使います。

とても簡単に実装できる半面、細かな当たり判定の調整は単純にはできません。GodotにはCollision（コリジョン）を扱うノードがあります。



コリジョンは「衝突・激突」などの意味を持ち、その名の通り当たり判定を行うしくみです。



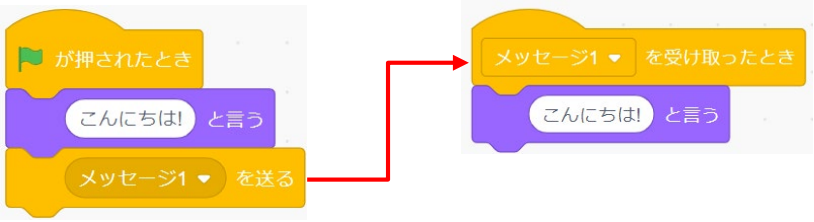
表示される画像とは別に、当たり判定を行うための、実行時には見えない図形を設定します。図形のサイズや形を調整することで、簡単にゲームの難易度の調整を行えます。

Scratchでも同じように、見えなくしたスプライトをキャラクターにかぶせることで、同様のしくみを再現することができます。Godotではこういったゲーム制作で必須のしくみが用意されており、ゲームエンジンとしての利点と言えます。

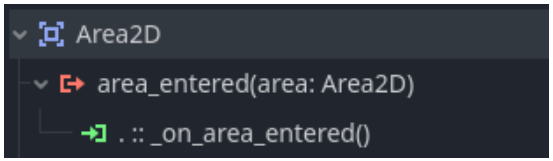
Copyright © ProgLab All rights reserved.

■メッセージ

Scratchであるスプライトに変化があった場合に、他のスプライトでもその変化を受け取って、何かアクションを起こしたい場合に、メッセージを使って伝達を行います。



GodotではScratchのメッセージと全く同様の機能はありませんが、「シグナル」機能がメッセージに近い機能です。
下の例はArea2Dノードに用意されているシグナルで、何かと衝突した場合にシグナルが発生して、設定した処理が呼び出されるようになっています。

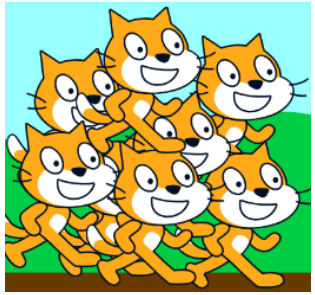


シグナルを使って、ゲーム内のキャラクター同士で通信したり、情報をやりとりしたりすることができます。

■クローン

例えば敵キャラや弾丸など、同じスプライトを画面内に複数登場させるために、スプライトを大量生産するのではなく、コピーする機能がScratchのクローンです。

特にゲームをつくるときには必須の機能とも言えます。



Godotだけでなく、オブジェクト指向プログラミングができる環境では、「クラス」と「インスタンス」がクローンと同じ概念です。

Godotではゲームに登場するキャラクターを「シーン」として作成し、実行時に「インスタンス」にすることでクローンと同じようにコピーすることができます。
シーンがクラスのような働きをします。

具体的にはこのようなコードで、シーンをインスタンスにして、それを画面上に登場させています。

```
var f_instance = f_scene.instantiate()
add_child(f_instance)
```


Copyright © ProgLab All rights reserved.

■ブロック定義

Scratchで複雑なプログラムをつくるようになってくると、同じような処理があちらこちらで発生するようになってきます。また1つの処理が長くなり、見た目にも使い勝手も悪くなることもあります。

そんな場合に「ブロック定義」を使って似たような処理を1つにしたり、小さな処理ごとにまとめるようなことができます。

Godotだけでなく多くのプログラミング言語では関数（かんすう）を使って同じような処理を行います。Godotで関数をつくるには、func（ファンク）キーワードを使います。

```
func game_over():
    $Player.queue_free()
```

Scratchのブロック定義を使ったことがあれば、関数はほとんど同じような感覚で使うことができます。また関数にはブロック定義にはない「戻り値（もどりち）」がありますので、Scratchよりもより便利な使い方もできます。

■変数の表示

Scratchの変数で便利なのが、値をゲーム画面上で確認したり、設定することができる機能です。

Godotで開発中に変数の値を確認するには、print()関数を使って値を出力パネルに表示し、値の変化をチェックする方法が最も簡単です。

Scratchでは変数の値を画面表示することで、そのまま簡易的にユーザーインターフェースとしても使えますが、GodotではLabelノードを使って値を画面表示することが簡単な方法です。

Copyright © ProgLab All rights reserved.

0-3 PCの基礎知識

- 基礎知識（１）
- 基礎知識（２）
- 基礎知識（３）
- 基礎知識（４）
- 基礎知識（５）

Copyright © ProgLab All rights reserved.

ここからは、プログラミングをするときや、PCを扱う上での基本的な知識や操作について記載します。

■半角・全角

キーボードから入力する文字には全角と半角があります。
　全角文字：１２３ＡＢＣ＿ー！あいう
　半角文字：123ABC_-!

プログラミングで使うのは主に**半角文字**です。全角文字は画面に表示する文字列などで使います。

キーボードから入力する時には、この半角と全角を切り替えて使えなければなりません。

Windows
キーボード左上にある、半角/全角切り替えキーを使います。

Mac
キーボード左下にある英数キーで半角に切り替わります。スペースキーを挟んで反対側のかなキーで全角になります。

■修飾キー

WindowsのCtrl（コントロール）キーや、Macの Command（コマンド）キーは修飾キーとよばれ、それを押しただけでは何も起こりませんが、他のキーと組み合わせて特定の機能をします。

特によく使うのはこの２つでしょうか。

ctrl + s	|	Command + s	保存
ctrl + z	|	Command + z	１つ元の操作に戻る

また、ソースコードや画面上のものなどを、コピーするときによく使うのがこの３つです。
※「コピペする」と良く言いますね。

ctrl + c	|	Command + c	選択したもの、範囲をメモリ上にコピーしておく
ctrl + x	|	Command + x	選択したもの、範囲をメモリ上にコピーしつつ、画面からは削除（切り取り）する。
ctrl + v	|	Command + v	メモリ上にコピーされたものを指定した場所に張り付け（ペースト）する。

修飾キーを使った操作は「ショートカット」ともよばれます。

Copyright © ProgLab All rights reserved.

■CAPS LOCK

通常はアルファベットの大文字を入力するときは、Shiftキーを押しながらアルファベットを入力しますが、CAPS LOCK（キャプス・ロック）をオンにすると常に大文字の入力になります。

あまり使うことはないのですが、意図せずONになってしまっていることがあるため、キーボードによっては、CAPS LOCKキーがONになっているときに小さなLEDが点灯するようになっているものもあります。

また、ON/OFFの切り替えも、Shiftを押しながらCAPS LOCKキーを押さないと有効にならないものもあります。

■ドラッグ＆ドロップ

ドラッグは、画面上のものを左クリックで選択し、マウスボタンは押したまま別の場所までマウスを移動させる動作です。ドロップはそのままマウスボタンを離して移動を確定させる動作です。

Godotでも素材ファイルを割り当てるために、ファイルをドラッグ＆ドロップさせる操作や、ソースコードを範囲選択してまとめて移動させたりもします。

■右クリック

マウスの右側のボタンをクリックします。基本的にマウスは左側のボタンを使って操作しますが、右側のボタンは画面上で押す場所によって特殊な動作をします。

たいていは特別なメニューが表示されます。
右図の例は、ソースコード上で右クリックしたときに表示されるメニューです。

修飾キーを使ったショートカットも書かれていますので、これを見ながら覚えるのもいいでしょう。

■ダブルクリック

マウスの左ボタンを２回連続で押します。いろいろな場面で使えますが、意外とソースコードを入力する際にも便利に使えます。

コード中の語句の上でダブルクリックすると、その語句全体を選択した状態になります。

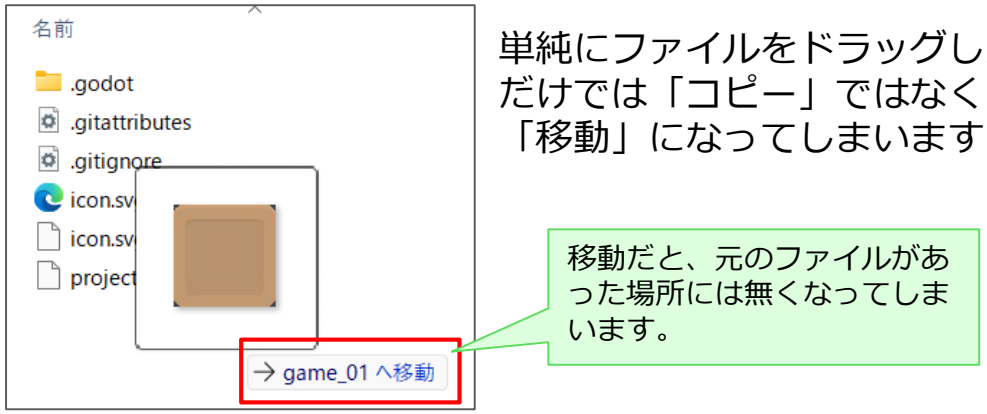
選択ミスがなくなりますし、そのままドラッグ＆ドロップすることもできます。

Copyright © ProgLab All rights reserved.

■ファイル、フォルダの操作

PCでは様々なアプリケーションを使って作業を行います。
アプリケーションで使用するデータや作成したものは、ファイルとして保存します。
そのファイルをまとめて整理するのがフォルダです。

Godotで制作を行う際によく行う操作は、ダウンロードしたファイルを、プロジェクトフォルダの中にコピーする作業です。



単純にファイルをドラッグしただけでは「コピー」ではなく「移動」になってしまいます。

コピーをしたい場合は、修飾キーを使います。
・Windows：Ctrlキーを押しながらドラッグ＆ドロップ
・Mac：Optionキーを押しながらドラッグ＆ドロップ

ただしUSBメモリなど、PC本体ではない場所からドラッグした場合は、デフォルトの操作がコピーになります。

また、ファイルを選択してから、ショートカットを使ったコピー＆ペーストでも、ファイルをコピーできます。

他には、間違ってファイルを移動してしまった場合などは、同じくショートカットで１つ元の操作に戻る、「Ctrl+z（Command+z）」を使えば、操作を１つずつ取り消すことができます。

ファイル、フォルダ名の変更

Godotのプロジェクトで使用している、ファイルやフォルダの名前は、Godotのアプリケーション内以外で変更してしまうと、プロジェクト内の参照関係が崩れてしまって動作しなくなります。

例えばプロジェクトフォルダの名前を変えてしまうと、プロジェクトマネージャーから見つからなくなってしまいます。

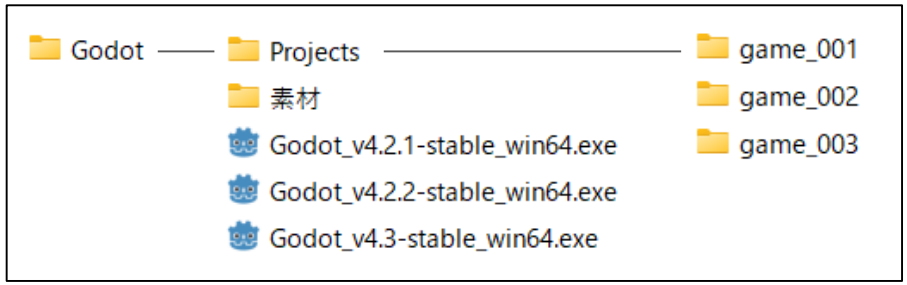
プロジェクト名を変更したい場合は、プロジェクトマネージャーのメニューから行うようにします。

Copyright © ProgLab All rights reserved.

■ファイル、フォルダの管理

ファイルやフォルダをどこに保存するのか迷う場合があると思います。これが正解というものはありませんが、基本的には、Windowsの場合は「ドキュメント」、Macの場合は「書類」フォルダ内がデフォルトの保存場所になります。

その中に例えば「Godot」フォルダをつくって、そこをGodot用のファイル置き場とします。



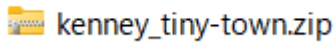
この例では、Projectsフォルダ内にGodotで作成するゲームのプロジェクトを保存しています。

素材フォルダはダウンロードした画像などの素材ファイルをまとめて保存しておくところです。（ゲーム内で使う場合は必要なもののみ、ここから各プロジェクトにコピーします）

Godotの実行ファイルもここに置いておくと分かりやすいですね。

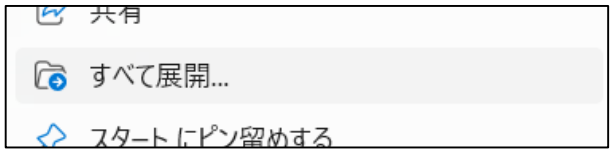
■圧縮と解凍（展開）

特にインターネットで素材やファイルをダウンロードすると、圧縮ファイルの状態で保存されます。圧縮ファイルにするとファイルサイズを小さくすることができるため、インターネット上での転送に都合が良くなります。



.zip（ジップ）形式のものが一般的ですが、他にも種類があります。圧縮されたファイルは「解凍（展開）」しなければ使うことはできません。

Windows
圧縮ファイル上で右クリックして表示されるメニューから「すべて展開...」を選んで、任意の場所に展開します。基本的には圧縮ファイルと同じ名前のフォルダが生成されます。



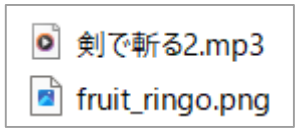
Mac
圧縮ファイルをダブルクリックします。

要注意
Windowsでもダブルクリックでzipファイルの中を確認できますが、解凍されていないためそのままファイルを使うことができません。圧縮ファイルは必ず解凍してから使います。

Copyright © ProgLab All rights reserved.

■ファイル拡張子

PCで扱うファイルには、そのファイルが何の種類のファイルなのかを表す「拡張子」というものがあります。



この例の場合は「.mp3」「.png」が拡張子にあたります。

PCによっては拡張子が表示されない設定になっている場合もありますので、表示されるように設定しておいたほうが、何のファイルなのか分かりやすくなります。

代表的なものを記載します。

文書ファイル

.txt	書式なしのテキスト
.doc　.docx	Microsoft Wordファイル
.rtf	リッチテキスト（簡易的なフォーマット付きテキスト）
.pdf	PDF（Portable Document Format）

表計算ファイル

.xls　.xlsx	Microsoft Excelファイル
.csv	カンマ区切りテキストファイル

プレゼンテーションファイル

.ppt　.pptx	Microsoft PowerPointファイル

画像ファイル

.jpg　.jpeg	一般的な写真などに使われる
.png	背景が透過できるので、ゲーム用に使いやすい pngはピングとよびます
.gif	アニメーションも可能だが、最近はあまり使われなくなっている
.svg	ベクター形式で拡大しても劣化しない
.webp	Web向けの高圧縮・高品質なフォーマット

音声ファイル

.mp3	一般的な音楽ファイル
.wav	圧縮されておらず高品質
.ogg	MP3よりもサイズが小さく高音質なためゲーム制作で使いやすい
.m4a	MP3よりもサイズが小さく高音質でApple製品と相性が良い

動画ファイル

.mp4	一般的な動画ファイル
.wmv	Windows環境で利用されることが多い
.mov	Apple製品の標準フォーマット

Copyright © ProgLab All rights reserved.

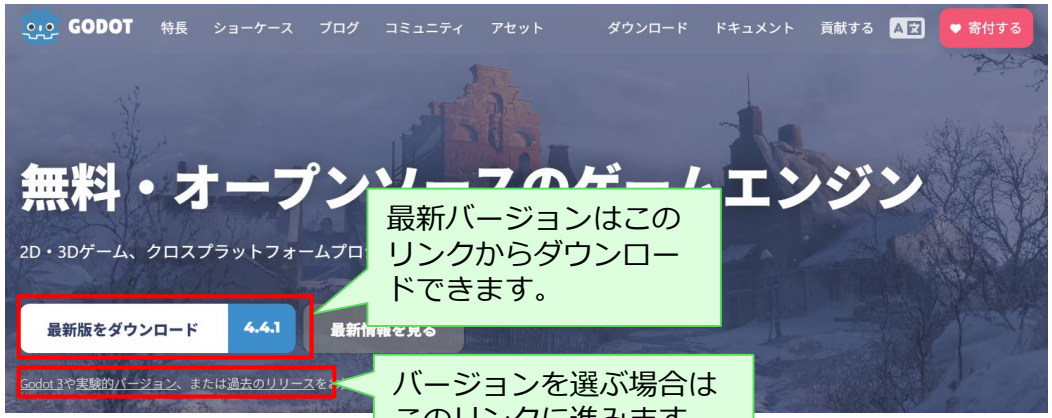
0-4 Godotをはじめよう

- ダウンロード
- プログラミング準備
- プログラミング基礎（１）
- プログラミング基礎（２）
- プログラミング基礎（３）
- プログラミング基礎（４）

Copyright © ProgLab All rights reserved.

ダウンロード

Godotは公式サイトからダウンロードします。
https://godotengine.org/



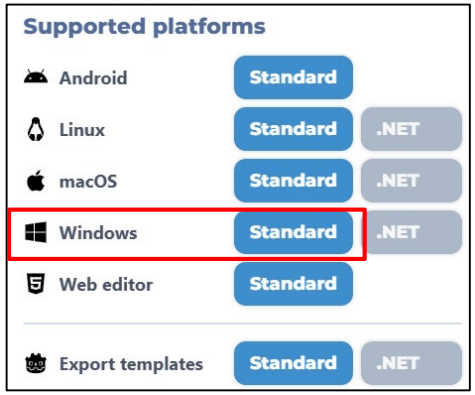
dev：Development：開発版
まだテスト中で、安定性が保証されていないバージョン

rc：Release Candidate：リリース候補
正式リリース前の最終確認用バージョン

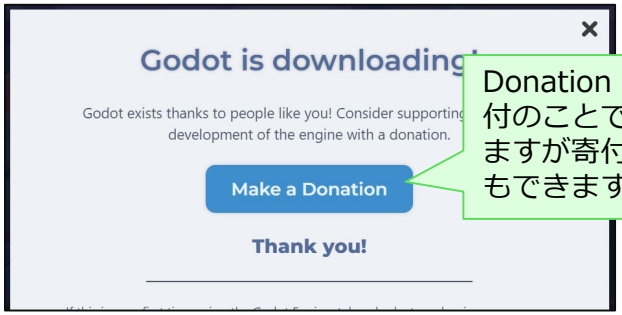
beta：ベータ版
機能は一通り実装されているが、まだ正式リリース前

stable：安定版
十分にテストされ、正式リリースされたバージョン

バージョンを選ぶ場合は、基本的にはstableと書かれた安定したバージョンをダウンロードします。



自分が使っているPC環境のものを選択します。
（Windows版を例にします）



Godotの起動やプロジェクトの始め方については、基本テキストを参照してください。

基本テキスト 1-1 プロジェクトと画面構成

Copyright © ProgLab All rights reserved.

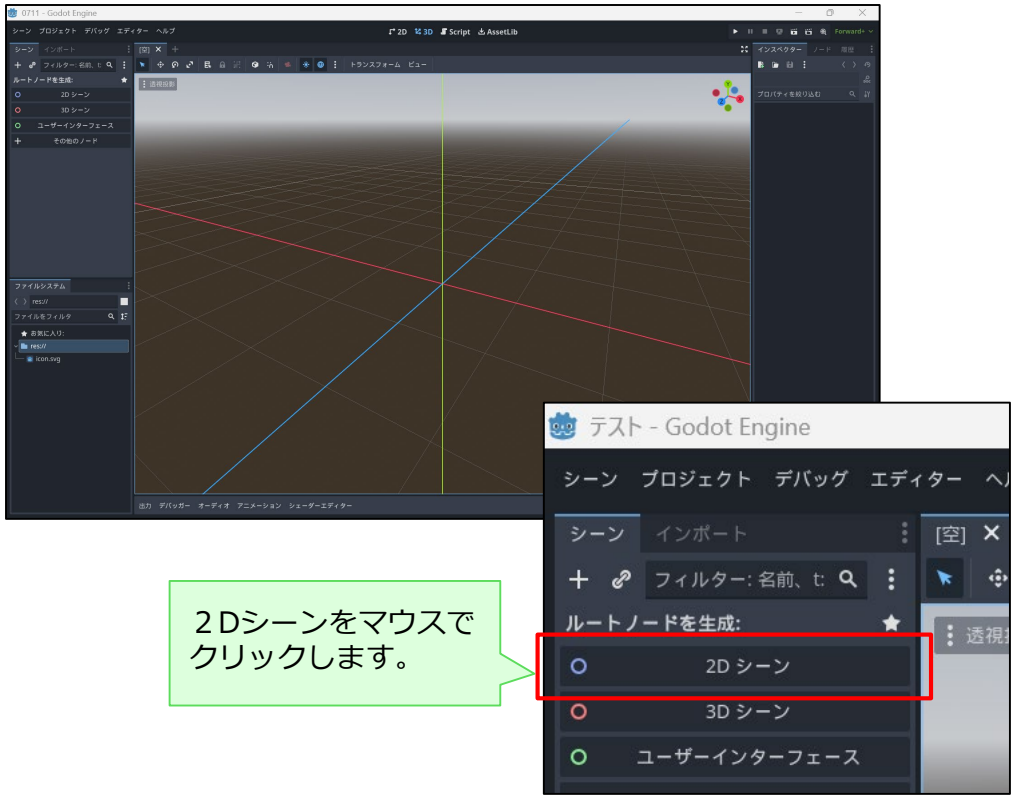
プログラミング準備

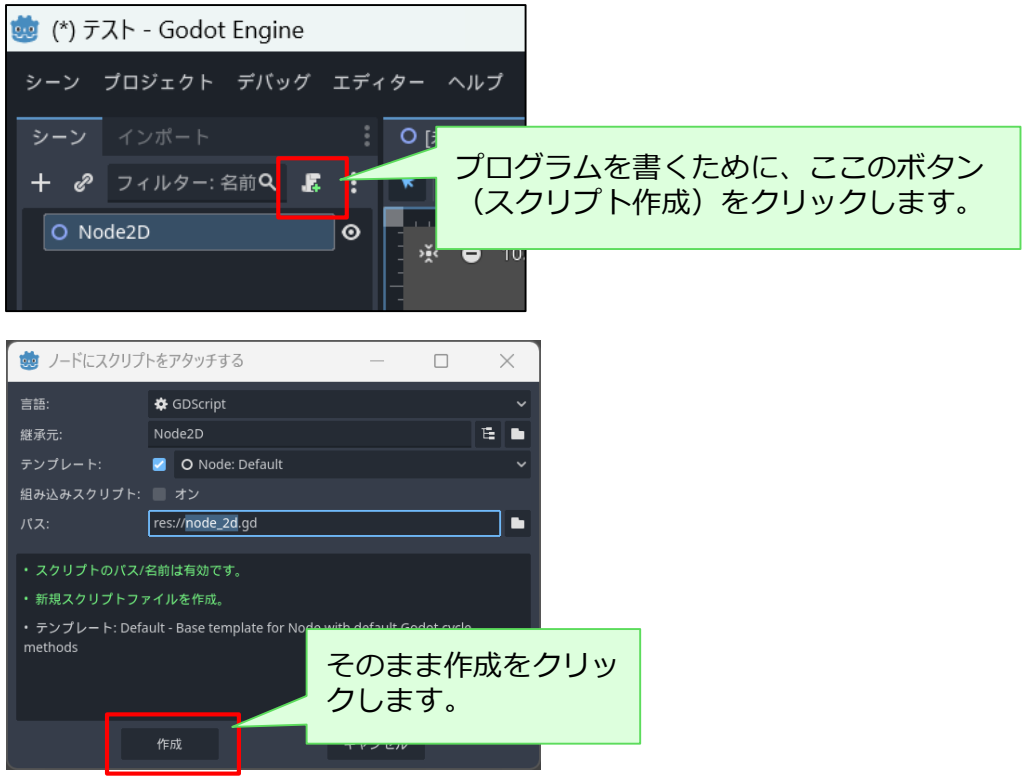
Godotでゲームをつくるには、画像や音の素材も必要ですし、ゲームの構成を考えたり、面白くするアイデアも重要です。

プログラミングについてはゲーム制作をしながら習得していくことになりますが、まずは基礎を学んでおく必要があります。

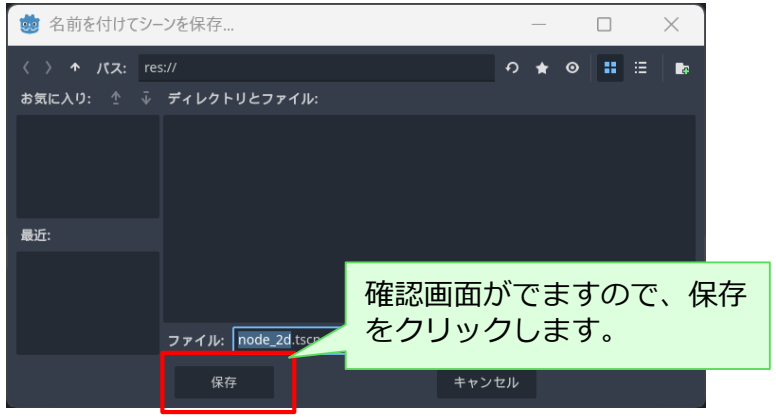
ここからはプログラミングの演習を行うため、テスト用のプロジェクトを使ってGodotのスクリプトに慣れるようにします。

新規のプロジェクトを開いたところからはじめます。





一度プロジェクトを保存します。このあとも定期的に保存するので便利な方法として、キーボードの[ctrl]と[S]を同時に押します。



Copyright © ProgLab All rights reserved.

いよいよプログラミングを書く準備ができました。一つだけ重要なルールについて説明しておきます。

■インデント

```
1   extends Node2D
2
3
4   # Called when the node enters the scene
5   func _ready() -> void:
6       pass # Replace with function body.
7
```

ここの空白のことを
「インデント」とよびます。

Godotのプログラムでは、処理のまとまりを区切るのに「インデント」を使用します。インデントとは**字下げ**のことです。

```
if 条件式：
    処理１
else:
    処理２
```

インデント → 処理１
インデント → 処理２

インデントは **[Tab]キー**で入力します。

■プログラミングのきそ

どんなプログラミング言語でも、基本的なところは変わりません。まずは実際に入力しながら慣れていこう。

基本の制御構造
- 順次処理
- 繰り返し
- 条件分岐

基本の要素
- 変数
- 関数
- 演算子、式

次のページからプログラミングを書いていきます。
6行目から入力しますので、いったん今あるものを消しておきます。

```
4   # Called when the node ente
5   func _ready() -> void:
6
7
8
```

「インデント」は残しておきます。

Copyright © ProgLab All rights reserved.

■順次処理（じゅんじしょり）

順次処理とは、上から順に命令を実行することです。プログラムコードは上から順番に一行ずつ実行されます。

コード【1】

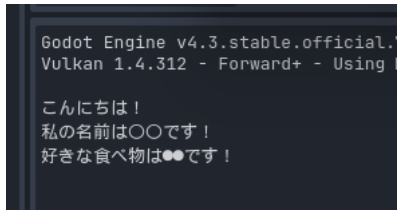
```
5  func _ready() -> void:
6      print("こんにちは！")
7      print("私の名前は〇〇です！")
8      print("好きな食べ物は●●です！")
```

日本語の部分のみ全角入力します。

print(“～ ～ ～”)　は文字を出力する命令です。画面下の出力パネルに表示されます。

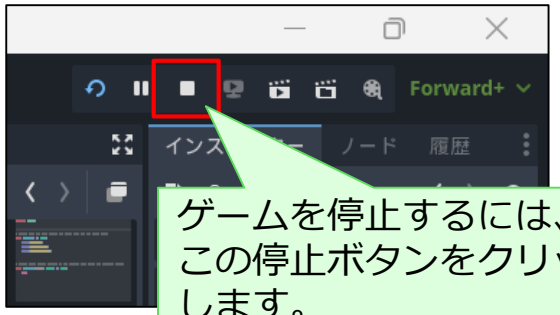
入力ができたら、画面右上の再生ボタンを押して、ゲームを実行します。

ゲームはまだ何もつくっていませんので、真っ暗の画面だけが表示されますが、画面の下に実行結果が表示されています。



こんにちは！
私の名前は〇〇です！
好きな食べ物は●●です！

プログラムの6行目、7行目、8行目の順番に処理が実行されているのがわかります。



func _ready()
func（ファンク）というのは「関数（かんすう）」というもので、まとめて実行したいことを、一つのかたまりにすることができます。

英語の、function（ファンクション）を縮めています。

数学で学ぶ「関数」は、特定の計算結果が出てくるまとまりみたいなイメージですので、ちょっと似ているところがあるかも。

Copyright © ProgLab All rights reserved.

■繰り返し（くりかえし：ループ）

人間が同じことを何度も正確に繰り返すのは大変ですが、決まったことを何度も繰り返して実行するのはコンピュータが得意な仕事です。

コード【2】

```
5  func _ready() -> void:
6      for i in range(5):
7          print("こんにちは！")
8          print("私の名前は〇〇です！")
9          print("好きな食べ物は●●です！")
```

赤い表示はエラーが出ている状態

```
5  func _ready() -> void:
       for i in range(5):
print("こんにちは！")
print("私の名前は〇〇です！")
print("好きな食べ物は●●です！")
```

forの行を入力した後に、[tab]キーでインデントを入力します。

繰り返し処理をするには、for～を使います。ここでは、range(5)を指定することで5回繰り返します。

range（レンジ）は
英語で「範囲」という意味です。

ここでは繰り返したい回数を指定します。

```
こんにちは！
私の名前は〇〇です！
好きな食べ物は●●です！
こんにちは！
私の名前は〇〇です！
好きな食べ物は●●です！
こんにちは！
私の名前は〇〇です！
好きな食べ物は●●です！
こんにちは！
私の名前は〇〇です！
好きな食べ物は●●です！
こんにちは！
私の名前は〇〇です！
好きな食べ物は●●です！
```

range(5) のところの数字を変更して、繰り返し回数が変わるのを確かめよう。

Copyright © ProgLab All rights reserved.

■条件分岐（じょうけんぶんき：もし～なら）

条件分岐は「もし〇〇なら、××する」といった、条件によってやることを変えるしくみです。

人間の生活に例えると
・もし雨が降っていたら → 傘を持っていく
・もしおなかがすいていたら → ごはんを食べる

これをコンピュータに教えるのが「**if 文（イフぶん）**」です。

コード【3】

```
5  func _ready() -> void:
6      var score = 85
7      if score >= 90:
8          print("すごい！")
9      elif score >= 70:
10         print("よくがんばった！")
11     else:
12         print("もう少しがんばろう！")
```

何と表示されるか予想してみよう

if **条件１**	イフ	**条件１**が当てはまったら実行
elif **条件２**	エルイフ	**条件１**がダメなら次の**条件２**をチェック
else	エルス	どれにも当てはまらなければ実行

var score = 85
var（ヴァー）というのは「変数（へんすう）」というもので、数字や文字などのデータをしまっておける入れ物のことです。

英語の、variable（ヴァリアヴル）が語源です。

この例では、score（スコア：点数）という名前の変数をつくって、値として85を入れています。scoreからは、好きなタイミングで中に入っている値を取り出すことができます。

■比較演算子（ひかくえんざんし）

左辺と右辺の数値の大小関係を比較します。

==	A == B（AとBは等しい）
!=	A != B（AとBは等しくない）
>	A > B（AはBより大きい）
>=	A >= B（AはBと同じか大きい）
<	A < B（AはBより小さい）
<=	A <= B（AはBと同じか小さい）

Copyright © ProgLab All rights reserved.