応用テキスト
A06

データの書き込み・読み込み

ver.1.0.0

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。

JSONの読み込み

JSON（ジェイソン：JavaScript Object Notation）は、データを保存するための軽量なフォーマットで、辞書や配列の形式でデータを管理できます。

Godotでは標準でJSONファイルを読み書きでき、ゲーム内で使用するデータを、JSON形式の外部ファイルで別途管理することができます。

JSONはこのようなフォーマットで記述します。

```json
{
  "enemies": [
    {
      "id": "enemy_1",
      "name": "スライム",
      "hp": 10,
      "power": 2,
      "speed": 2,
      "sprite_path": "res://images/slime.png"
    },
    {
      "id": "enemy_2",
      "name": "ゴーレム",
      "hp": 20,
      "power": 5,
      "speed": 4,
      "sprite_path": "res://images/golem.png"
    }
  ]
}
```

■JSONの読み込み

JSON形式のファイルを読み込む処理の例です。

```gdscript
var enemy_data = {}
var file_path = "res://enemies.json"

func load_data():
    var json:JSON = JSON.new()
    var file = FileAccess.open(file_path, FileAccess.READ)
    if file:
        var error = json.parse(file.get_as_text())
        if error == OK:
            enemy_data = json.data
        else:
            print("読み込みエラー")
```

var json:JSON = JSON.new()
JSONクラスのインスタンスを作成します。

var file = FileAccess.open(file_path, FileAccess.READ)
FileAccessクラスのopenメソッドを呼び出します。FileAccess.READは読み込み専用モードで開くことを意味します。読み込みに成功したらFileAccessのインスタンスが返ってきます。

var error = json.parse(file.get_as_text())
JSONインスタンスのparse()メソッドを使用して、FileAccessのインスタンスから読み込んだデータをチェックし、エラーコードを返します。問題なければOKが返ります。

enemy_data = json.data
JSONインスタンスから、データを取り出します。取り出したデータは辞書形式になっています。

ゲームには設定メニューが用意されているものも多くあります。言語やコントローラーの設定、難易度、音量など、ゲームの種類によって様々なものがあります。

そういったものは設定データとして外部ファイルに保存しておいて、次にゲームを再開したときにも、設定した内容が引き継がれるようになっています。

設定メニューはたいてい歯車の形をしたアイコンで表されます。

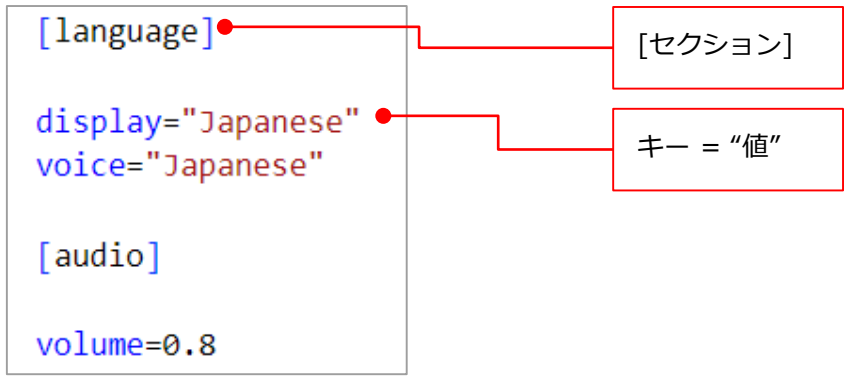
Godotでは設定データ専用の、ConfigFile（コンフィグファイル）クラスがあり、それを利用することで簡単に設定ファイルを扱えるようになっています。

Config（コンフィグ）とは、Configuration（コンフィグレーション）の略語で、特にコンピュータの世界では「設定」という意味で扱われます。

ConfigFileで保存するデータはテキストデータ形式ですが、ファイルの拡張子は「.cfg」を用います。これはGodotに限ったものではなく、一般的なアプリケーションなどの設定ファイルにも使われています。

もちろんゲームの設定だけではなく、ゲームの進行度合いなどのデータを簡易的に保存しても構いません。

設定ファイルの形式はこのようになっています。



セクションはデータを分類するためのもので、実際のデータは、キーと値を指定して保存します。

設定ファイルの保存場所は、デフォルトで決まっており、ファイルパスで指定する「user://」の場所は、OSによって異なります。

Windows:
C:¥Users¥[ユーザー名]¥AppData¥Roaming¥Godot¥app_userdata¥[プロジェクト名]

macOS:
~/Library/Application Support/Godot/app_userdata/[プロジェクト名]

保存場所はプロジェクト設定で変更可能ですが、特に指定する必要がなければデフォルトのままで良いでしょう。

■設定ファイルの書き込み

設定ファイルを書き込む処理の例です。

```
func save_settings():
    var config = ConfigFile.new()
    config.set_value("language", "display", "Japanese")
    config.set_value("audio", "volume", 0.8)

    var err = config.save("user://settings.cfg")
    if err == OK:
        print("設定ファイルを保存しました")
    else:
        print("エラー")
```

var config = ConfigFile.new()
ConfigFileクラスのインスタンスを作成します。

config.set_value("language", "display", "Japanese")
set_value()メソッドでデータを指定します。
引数の1つ目がセクション、2つ目がキー、3つ目が値です。

err = config.save("user://settings.cfg")
ファイル名を指定して保存します。実行結果にOKが返ってくれば書き込み成功です。

■設定ファイルの読み込み

設定ファイルを読み込む処理の例です。

```
func load_settings():
    var config = ConfigFile.new()
    var err = config.load("user://settings.cfg")

    if err != OK:
        print("エラー")
        return

    var language_display = config.get_value("language", "display")
    var language_voice = config.get_value("language", "voice")
    var audio_volume = config.get_value("audio", "volume")
```

var err = config.load("user://settings.cfg")
load()メソッドで設定ファイルを読み込みます。実行結果にOKが返ってくれば読み込み成功です。

var language_display = config.get_value("language", "display")
get_value()メソッドでデータを取り出します。
引数の1つ目がセクション、2つ目がキーです。
※config.get_value("language", "display", "English")のように、3番目の引数を指定した場合は、セクションやキーが存在しなかった際に、その値がデフォルト値としてセットされます。

ゲームによっては、ゲームの進行度やキャラクターの成長パラメータなどを保存しておいて、継続的にゲームを続けられるようにするしくみが必要になります。

JSON形式を利用してデータ保存してもよいのですが、ゲーム内で使用している配列や辞書形式のまま、特に変換することなく保存できたほうが便利です。

■セーブデータの書き込み

```
func save_data():
    var path = "user://save.txt"
    var file = FileAccess.open(path, FileAccess.WRITE)
    if file == null:
        print("エラー")
        return

    var data = {
        "name": "Syohei",
        "level": 5,
        "score": 1200
    }
    file.store_var(data)
    file.close()
    print("保存しました")
```

FileAccess.open(path, FileAccess.WRITE)
FileAccessクラスの、open()メソッドを使用してファイルを開く処理をします。
引数の1つ目がファイルパス、2つ目が書き込み処理を表します。

file.store_var(data)
file.close()
store_var()は辞書や配列などの形式をそのまま保存できる処理です。
close()でファイルを閉じて、保存処理を終了させます。

■セーブデータの読み込み

```
func load_data():
    var path = "user://save.txt"

    var file = FileAccess.open(path, FileAccess.READ)
    if file == null:
        print("エラー")
        return

    var data = file.get_var()
    file.close()

    print("name:", data["name"])
    print("Level:", data["level"])
    print("Score:", data["score"])
```

FileAccess.open(path, FileAccess.READ)
FileAccessクラスの、open()メソッドを使用してファイルを開く処理をします。
読み込みモードを指定しています。

var data = file.get_var()
file.close()
get_var()は保存された辞書や配列などの形式をそのまま読み込みできる処理です。
close()でファイルを閉じて、読み込み処理を終了させます。

実行結果

```
name:Syohei
Level:5
Score:1200
```

設定ファイルはデータをテキストデータで保存します。これは保存されたファイルを開けば、簡単に内容を知ることができ、また内容を変更してしまうこともできます。
データを暗号化保存することでそれを防ぐことができます。

■暗号化して書き込み

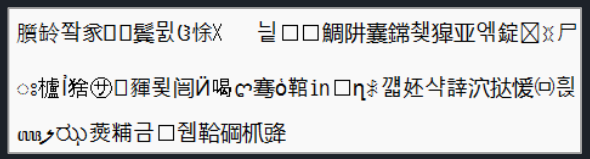
ゲームデータを暗号化して書き込む処理の例です。

```
const KEY = "ABCD1234"

func save_encrypted_data():
    var path = "user://encrypted_save.dat"
    var file = FileAccess.open_encrypted_with_pass(path, FileAccess.WRITE, KEY)
    if file == null:
        print("エラー")
        return

    var data = {
        "name": "Syohei",
        "level": 5,
        "score": 1200
    }
    file.store_var(data)
    file.close()
    print("暗号化保存しました")
```

保存されたデータ



FileAccess.open_encrypted_with_pass(path, FileAccess.WRITE, KEY)
FileAccessクラスの、open_encrypted_with_pass()メソッドを使用して、暗号化処理を使ってファイルを開く処理をします。
引数の1つ目がファイルパス、2つ目が書き込み処理を表し、3つ目が暗号キーです。

var data = file.store_var(data)
file.close()
store_var()は辞書や配列などの形式をそのまま保存できる処理です。
close()でファイルを閉じて、保存処理を終了させます。

const KEY = "ABCD1234"
は、暗号化キーと呼ばれ、任意の文字列を指定し、読み込み時にも同じキーを使用することで、暗号化されたものを復号化することができます。

■復号化して読み込み

暗号化されたデータを復号化して読み込む処理の例です。

```
func load_encrypted_data():
    var path = "user://encrypted_save.dat"

    var file = FileAccess.open_encrypted_with_pass(path, FileAccess.READ, KEY)
    if file == null:
        print("エラー")
        return

    var data = file.get_var()
    file.close()

    print("name:", data["name"])
    print("Level:", data["level"])
    print("Score:", data["score"])
```

FileAccess.open_encrypted_with_pass(path, FileAccess.READ, KEY)
FileAccessクラスの、open_encrypted_with_pass()メソッドを使用して、暗号化処理を使ってファイルを開く処理をします。読み込みモードを指定しています。

var data = file.get_var(data)
file.close()
get_var()は保存された辞書や配列などの形式をそのまま読み込みできる処理です。
close()でファイルを閉じて、読み込み処理を終了させます。

実行結果

```
name:Syohei
Level:5
Score:1200
```