C++ MIX #13

プラグイン開発で学習する UnrealEngine C++（概要編）

荻野雄季 @YuukiOgino

自己紹介

荻野雄季（おぎのゆうき）

有限会社テクニカルアーツ所属（時々個人事業主としても活動）

https://www.techarts.co.jp/

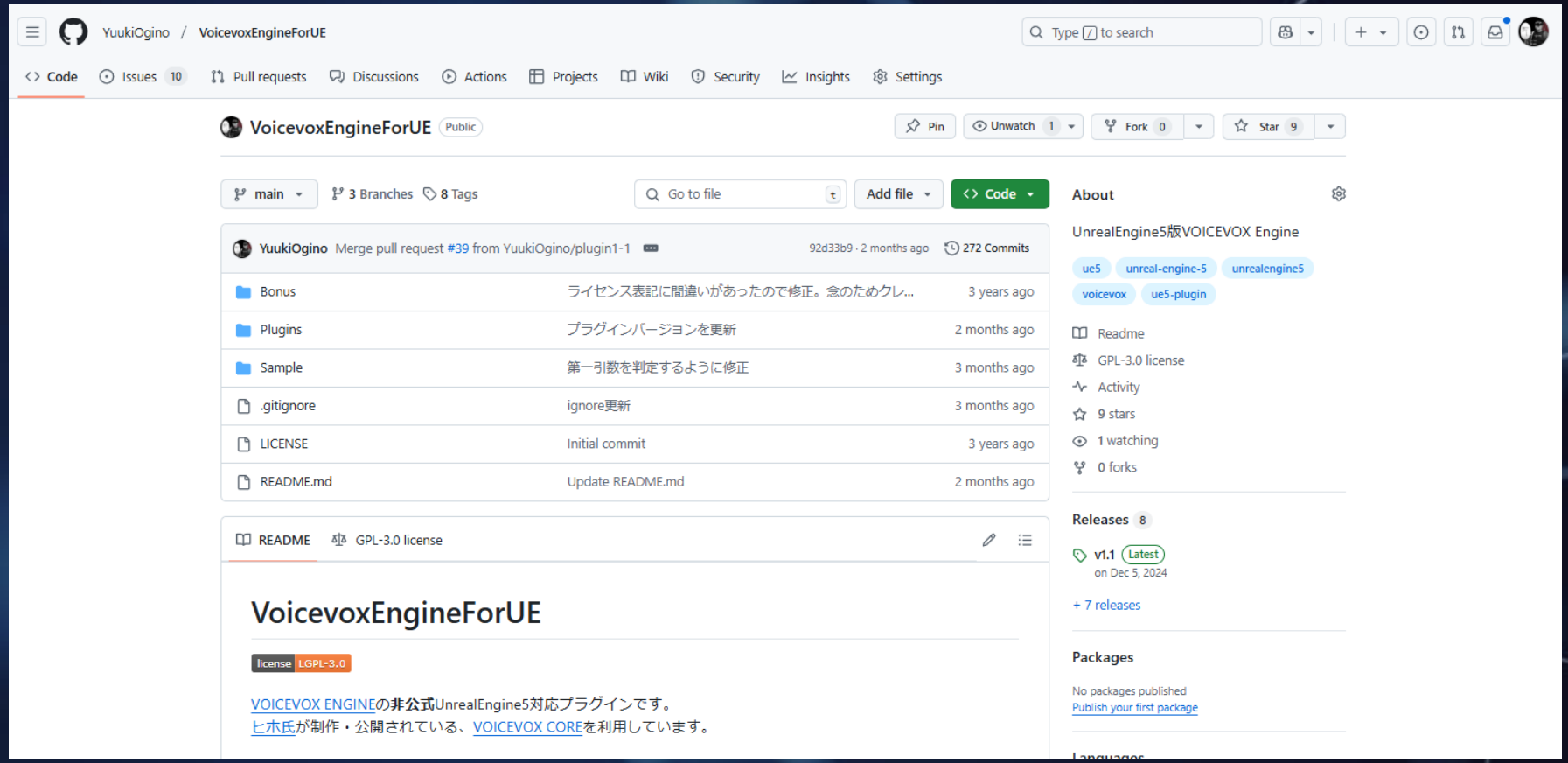
5年前からゲーム開発（C++）に関わっています。元々はWeb系のフロントエンジニア

最近はプロキャンパーがメイン職業、サブとしてプログラマーしてます

X @YuukiOgino

趣味で以下のプラグインを開発した時に
UE C++の学習に良いな、と思ったことを話します

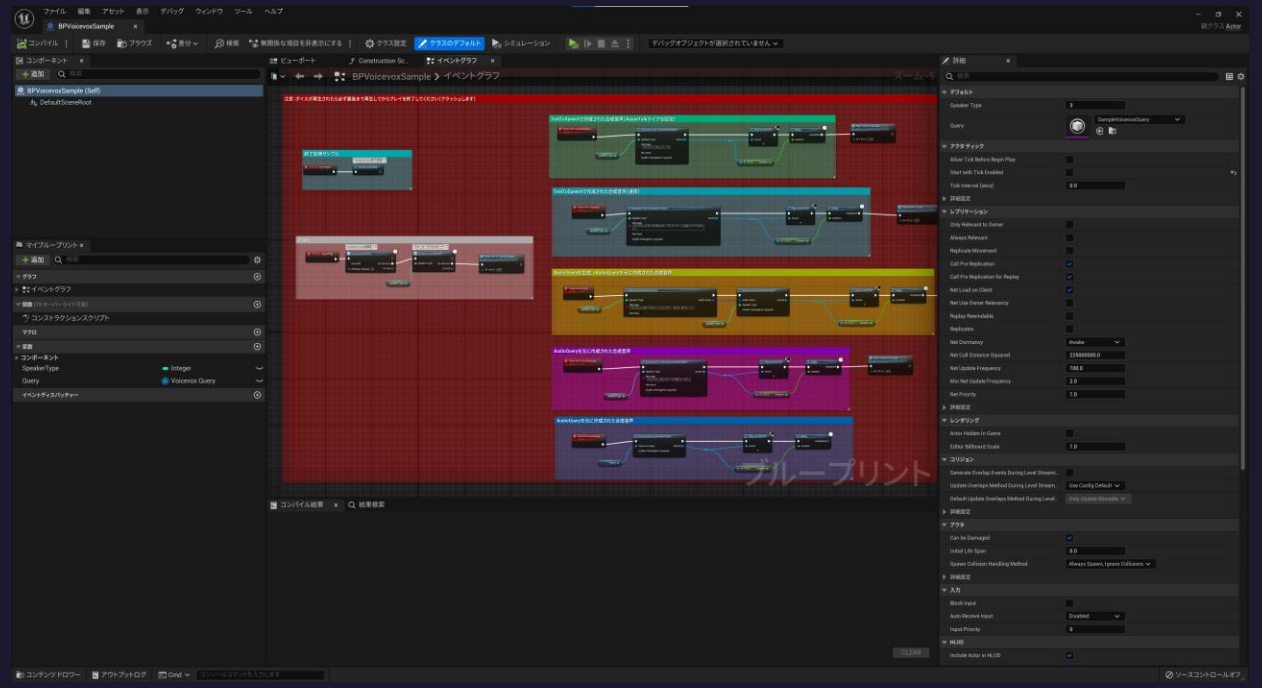
https://github.com/YuukiOgino/VoicevoxEngineForUE



簡単な説明

UNREAL ENGINE

- Epic Gamesが開発したゲームエンジン
  - ゲーム開発以外にも映像作品、シミュレーション等のエンタープライズにも使われている
  - 原則無料だが、年間総収益が100ドルを超えた場合は5%のロイヤリティを払う必要がある
- Blueprintと呼ばれるビジュアルスクリプトがある
  - C++が分からなくてもゲーム開発は可



https://www.unrealengine.com/ja

簡単な説明

VOICEVOX CORE

- 無料で使える中品質なテキスト読み上げソフトウェアの音声合成コア
- VOICEVOX CORE のソースコード及びビルド成果物のライセンスは MIT LICENSE
  - Release版はCOREに同梱されている利用規約を必ず一読し、守ってください
  - プラグインはRelease版を使用することを想定しています

https://github.com/VOICEVOX/voicevox_core

VoicevoxEngineForUEプラグイン

- VOICEVOX ENGINEの非公式UnrealEngine5対応プラグイン
- VOICEVOX COREを使用
  - VOICEVOXのマルチエンジン機能を搭載
- C++、Blueprintで使用可
- プラグインのライセンスはVOICEVOX ENGINEのライセンスを継承し、LGPL v3と、ソースコードの公開が不要な別ライセンスのデュアルライセンス
  - 上記ライセンスとは別に、VOICEVOX COREの禁止事項に抵触しないこと、使用した音声モデルの利用規約を守れば使用OK

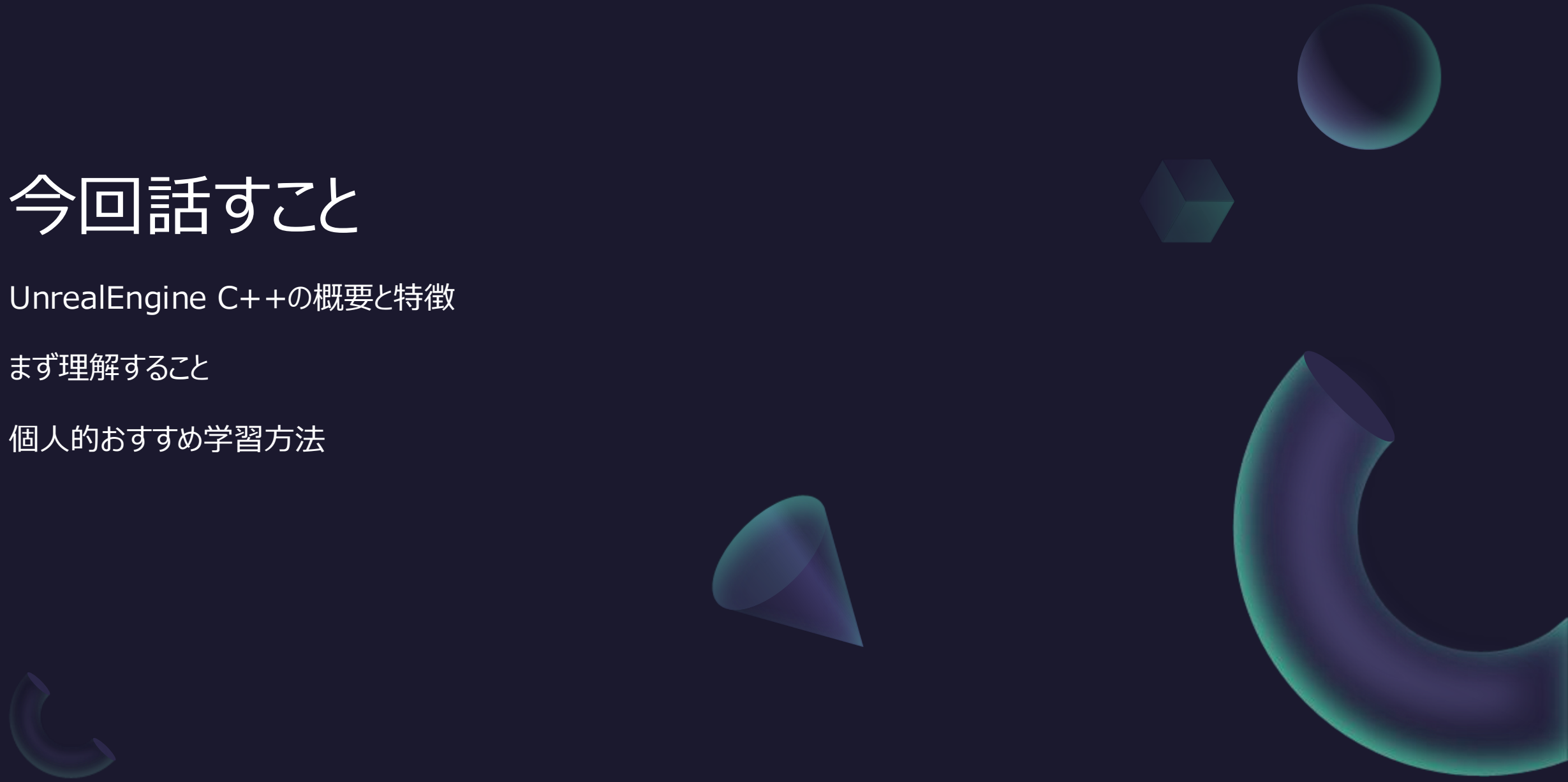
https://github.com/YuukiOgino/VoicevoxEngineForUE

今回話すこと

UnrealEngine C++の概要と特徴

まず理解すること

個人的おすすめ学習方法



UnrealEngine C++

- Epic Gamesが拡張した独自のC++
  - 補助付きC++と考えるといいそうです（UE4.27ドキュメントより）
  - https://dev.epicgames.com/documentation/ja-jp/unreal-engine/introduction-to-cplusplus-programming-in-ue4?application_version=4.27
- 標準のC++開発から見ると、お作法がかなり特殊
  - UnrealEngineリフレクションシステム等の様々な独自機能
  - UnrealEngine独自のビルドツール

UnrealEngine C++の特徴

- ガベージコレクションがある
- C#のようなデリゲート（他言語ではイベントディスパッチャー）がある
- ゲーム開発において使いやすい文字列、コンテナが用意されている
- ホット リロード機能がある
- マクロでメタデータ指定子、プロパティ指定子を設定できる
- C++ 標準ライブラリ リファレンス (STL)も使える
- Unreal Engine C++の場合、大体のクラスは「UObject」を継承
- エンジンのソースコードが公開されている

UnrealEngine C++の特徴

- コーディング規約
  - https://dev.epicgames.com/documentation/ja-jp/unreal-engine/epic-cplusplus-coding-standard-for-unreal-engine
  - 標準ライブラリの使用についても触れられています

コーディング規約

確立された標準とベストプラクティスを順守して、保守可能なコードを記述します。

Epic ではシンプルなコーディング規約をいくつか使用しています。このページでは、現在の Epic Games のコーディング規約を説明します。コーディング規約には必ず従わなければなりません。

コーディング規約がプログラマーにとって重要な理由はいくつかあります。

- ソフトウェアのライフタイム コストの 80 %はメンテナンス関連です。
- ソフトウェアの最初の制作者がライフタイムを通してメンテナンスを継続することはほとんどありません。
- コーディング規約によってソフトウェアの可読性が向上し、エンジニアは新しいコードを迅速にしっかりと理解できるようになります。
- MOD コミュニティのデベロッパーに公開するソース コードを理解しやすく用意することが重要です。
- クロスコンパイラの互換性を維持するために、こうした規則の多くが必要です。

以下のコーディング規約は C++ が中心となっていますが、どの言語を使用した場合でもこの規約に従うことが求められます。特定の言語に対して必要な場合には、同等のルールや例外が示されています。

コード例

```cpp
#pragma once

#include "Subsystems/VoicevoxNativeCoreSubsystem.h"

class UVoicevoxNativeObject;

/**
 * @class FVoicevoxSubsystemCollection
 * @brief VOICEVOX COREのネイティブAPIを参照するSubsystem管理クラス
 */
class FVoicevoxSubsystemCollection final : public FSubsystemCollectionBase
{

	//------------------------------------------------------------------
	// friend class
	//------------------------------------------------------------------

	friend class UVoicevoxNativeObject;

	//------------------------------------------------------------------
	// Function
	//------------------------------------------------------------------

	/**
	 * @brief 指定したクラスのSubsystemを取得
	 * @param SubsystemClass
	 * @return 指定したVOICEVOX CORE Subsystemクラス
	 */
	USubsystem* GetSubsystem(UClass* SubsystemClass) const
	{
		return GetSubsystemInternal(SubsystemClass);
	}

public:

	/**
	 * @brief コンストラクタ
	 */
	FVoicevoxSubsystemCollection() : FSubsystemCollectionBase(UVoicevoxNativeCoreSubsystem::StaticClass()) {}
};
```

コード例

```cpp
#pragma once

#include "Subsystems/VoicevoxNativeCoreSubsystem.h"
#include "CoreSubsystem.generated.h"

//------------------------------------------------------------------------
// class
//------------------------------------------------------------------------

/**
 * @class UCoreSubsystem
 * @brief VOICEVOX COREのAPIをまとめたSubsystem
 */
UCLASS(MinimalAPI)
class UCoreSubsystem final : public UVoicevoxNativeCoreSubsystem
{
	GENERATED_BODY()
protected:

	//------------------------------------------------------------------
	// Function
	//------------------------------------------------------------------

	/**
	 * @brief OpenJtakeのディレクトリ名を取得
	 * @return OpneJtakeのディレクトリ名
	 */
	virtual FString GetOpenJtakeDirectoryName() override;

public:

	//------------------------------------------------------------------
	// Function
	//------------------------------------------------------------------

	//--------------------------------
	// コンストラクタ
	//--------------------------------

	/**
	 * @brief コンストラクタ
	 */
	UCoreSubsystem() = default;

	//--------------------------------
	// override
	//--------------------------------

	/**
	 * @brief Initialize
	 */
	virtual void Initialize(FSubsystemCollectionBase& Collection) override;

	/**
	 * @brief Deinitialize
	 */
	virtual void Deinitialize() override;

	//--------------------------------
	// VOICEVOX CORE Property関連
	//--------------------------------

	/**
	 * @fn
	 *  VOICEVOX CORE名取得
	 * @brief VOICEVOX COREの名前取得
	 * @return VOICEVOX COREの名前取得
	 */
	virtual FString GetVoicevoxCoreName() override;
};
```

独学で勉強してみた

@YuukiOgino (雄季 荻野)

【UE4メモ】C++初心者がUE4エンジンソースを解析してみよう　SetActorLocation編

C++ UE4 UnrealEngine

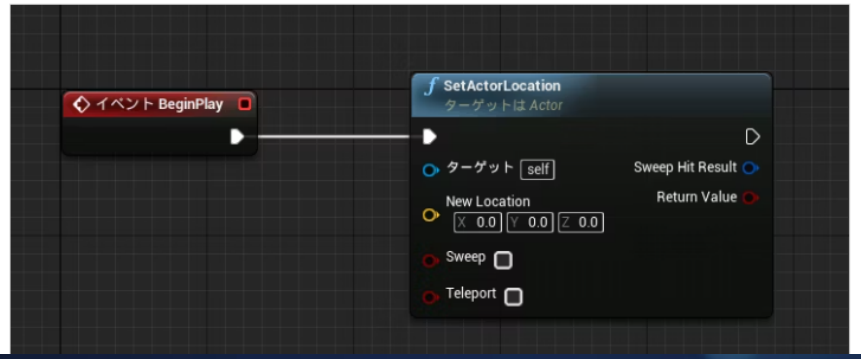
投稿日 2017年01月27日 7072 views

C++初心者がUE4の挙動を覚えるために無謀にもエンジンソースの解析を試みた記事になります。
C++初心者ゆえ、間違いがあると思うのでやんわりと指摘していただけると助かります。

今回はSetActorLocationを解析してみます。
検証は4.15プレビュー版で行っています。
将来処理が変わる可能性大なので、参考程度にしてください。

#SetActorLocationを解析してみる

BP上でのSetActorLocationを見てみましょう。



EngineTypes.h

```
FORCEINLINE ETeleportType TeleportFlagToEnum(bool bTeleport) { return bTeleport ? ETeleportTy
```

戻り値がETeleportTypeという列挙型みたいですね。

K2_SetActorLocationの解析はさっと終わったので、SetActorLocationメソッドの中身を読んでみます。

Actor.cpp

```
bool AActor::SetActorLocation(const FVector& NewLocation, bool bSweep, FHitResult* OutSweepH
{
    if (RootComponent)
    {
        const FVector Delta = NewLocation - GetActorLocation();
        return RootComponent->MoveComponent(Delta, GetActorQuat(), bSweep, OutSweepHitResult
    }
    else if (OutSweepHitResult)
    {
        *OutSweepHitResult = FHitResult();
    }

    return false;
}
```

必要ないと思いますが、BPでのSetActorLocationとC++のSetActorLocationの引数の違いです。

https://qiita.com/YuukiOgino/items/000cf8218a4fa478beb6

記録が残っている限り初めて触ったのが2017年あたり、まだC++の標準ライブラリすら知らなかった超初心者の時です

コード例

```cpp
#pragma once

#include "CoreMinimal.h"
#include "Blueprint/UserWidget.h"
#include "Components/Button.h"
#include "VoicevoxEditorCommonHeaderBar.generated.h"

//------------------------------------------------------------------------
// class
//------------------------------------------------------------------------

/**
 * @class UVoicevoxEditorCommonHeaderBar
 * @brief VOICEVOX編集エディターのヘッダーバークラス
 */
UCLASS()
class VOICEVOXUECOREEDITOR_API UVoicevoxEditorCommonHeaderBar : public UUserWidget
{
	GENERATED_BODY()
 /**
 * @brief AudioQueryアセット保存ボタンクリック（イベントハンドラー）
 */
 UFUNCTION()
 void OnAudioQuerySaveButtonClick();

 /**
  * @brief AudioQueryアセット読み込みボタンクリック（イベントハンドラー）
  */
 UFUNCTION()
 void OnAudioQueryLoadButtonClick();

 /**
  * @brief SoundWaveアセット保存ボタンクリック（イベントハンドラー）
  */
 UFUNCTION()
 void OnSoundWaveSaveButtonClick();

 /**
  * @brief WavFile保存ボタンクリック（イベントハンドラー）
  */
 UFUNCTION()
 void OnWaveFileSaveButtonClick();
```

```cpp
public:

 DECLARE_DYNAMIC_MULTICAST_DELEGATE(FAudioQuerySaveDelegate);
 DECLARE_DYNAMIC_MULTICAST_DELEGATE(FAudioQueryLoadDelegate);
 DECLARE_DYNAMIC_MULTICAST_DELEGATE(FSoundWaveSaveDelegate);
 DECLARE_DYNAMIC_MULTICAST_DELEGATE(FWavSaveDelegate);

 //! AudioQuery保存実行イベントディスパッチャー
 UPROPERTY(BlueprintAssignable, Category="VOICEVOX Editor")
 FAudioQuerySaveDelegate OnAudioQuerySave;

 //! AudioQuery読み込み実行イベントディスパッチャー
 UPROPERTY(BlueprintAssignable, Category="VOICEVOX Editor")
 FAudioQueryLoadDelegate OnAudioQueryLoad;

 //! SoundWave保存実行イベントディスパッチャー
 UPROPERTY(BlueprintAssignable, Category="VOICEVOX Editor")
 FSoundWaveSaveDelegate OnSoundWaveSave;

 //! WavFile保存実行イベントディスパッチャー
 UPROPERTY(BlueprintAssignable, Category="VOICEVOX Editor")
 FWavSaveDelegate OnWavFileSave;

protected:

 //! AudioQuery保存ボタン
 UPROPERTY(BlueprintReadWrite, meta=(BindWidget))
 UButton* AudioQuerySaveButton;

 //! AudioQuery読込ボタン
 UPROPERTY(BlueprintReadWrite, meta=(BindWidget))
 UButton* AudioQueryLoadButton;

 //! SoundWave保存ボタン
 UPROPERTY(BlueprintReadWrite, meta=(BindWidget))
 UButton* SoundWaveSaveButton;

 //! WavFile保存ボタン
 UPROPERTY(BlueprintReadWrite, meta=(BindWidget))
 UButton* WavSaveButton;
```

初心者が躓きやすい（混乱）ポイント

2017年ごろのC++初心者だった自分を思い出して書いてみました

- インテリセンスが働かない
- インテリセンスのエラーが大量にでる
- エラーが出力されているのにビルドが通ってエディタが立ち上がる
- 使用したいAPIがあるのにエラーがでる
- C++でプラグイン（モジュール）を有効にする方法がわからない
- UE独特のリンクエラーの原因がわからない
- Windows APIをインクルードしたのにビルドエラーが起きる
- （人によっては）UnrealEngineのコーディング規約
- 外部のライブラリをソリューションに登録するといつの間にか消える…等々

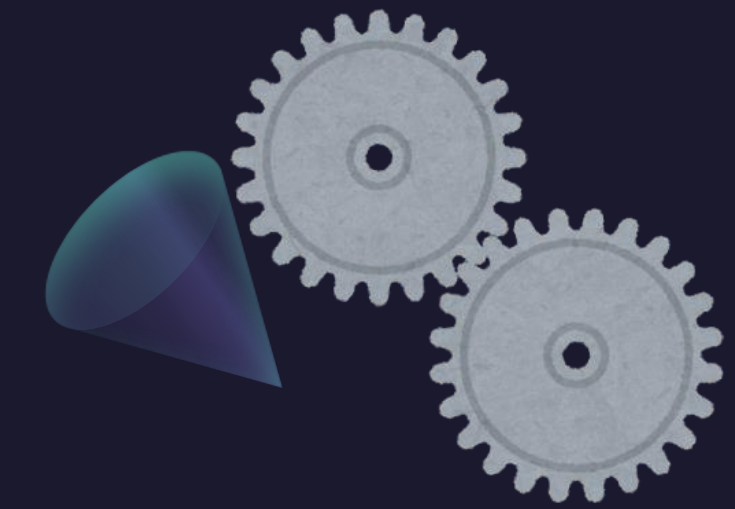
まず理解すること

UnrealEngine C++はモジュール開発

https://dev.epicgames.com/documentation/ja-jp/unreal-engine/unreal-engine-modules

- エディタ、ランタイム機能、ライブラリ等の機能を作成する時にモジュールが作られる
  - そのため、UEのビルドシステムはIDEのソリューションファイルではなく、プロジェクトは「Target.cs」、モジュールは「Build.cs」に基づいてビルドされる
    - ソリューション、xcworkspaceの生成は別の方法で行う
  - ビルド時にモジュールから動的ライブラリ（DLL、dylib）を作成

- 他にも色々ありますが、初めての人は「UEのC++は動的ライブラリを作る」という点だけ理解してください。
  - プラグイン開発もモジュールを作成します。



```
public VoicevoxCoreNemo(ReadOnlyTargetRules Target) : base(Target)
{
        Type = ModuleType.External;
        CppStandard = CppStandardVersion.Latest;

        if (Target.Platform == UnrealTargetPlatform.Win64)
        {
                const string platformName = "x64";
                const string binPlatformName = "Win64";
                const string thirdPartyName = "VoicevoxCoreNemo";

                // Add the import library
                PublicAdditionalLibraries.Add(Path.Combine(ModuleDirectory, platformName, "voicevox_core.lib"));

                // Delay-load the DLL, so we can load it from the right place first
                PublicDelayLoadDLLs.Add("voicevox_core.dll");

                // Ensure that the DLL is staged along with the executable
                RuntimeDependencies.Add($"$(PluginDir)/Binaries/ThirdParty/{thirdPartyName}/{binPlatformName}/voicevox_core.dll", Path.Combine(ModuleDirectory, platformName, "voicevox_core.dll"));

                // modelフォルダもコピーする
                AddRuntimeDependenciesDirectory("model", platformName, binPlatformName, true);
}
        else if (Target.Platform == UnrealTargetPlatform.Mac)
        {
                const string platformName = "osx";
                const string binPlatformName = "Mac";
                const string thirdPartyName = "VoicevoxCoreNemo";

                PublicSystemIncludePaths.Add(Path.GetFullPath(Path.Combine(ModuleDirectory, platformName)));

                RuntimeDependencies.Add($"$(PluginDir)/Binaries/ThirdParty/{thirdPartyName}/{binPlatformName}/libvoicevox_core_nemo.dylib", Path.Combine(ModuleDirectory, platformName, "libvoicevox_core.dylib"));

                // modelフォルダもコピーする
                AddRuntimeDependenciesDirectory("model", platformName, binPlatformName, true);
        }

        PublicDefinitions.Add($"OPEN_JTALK_DIC_NAME=\"{OpenJtalkDicName}\"");
}
```

個人的おすすめ学習方法

ドキュメントを見る、ユーザー記事検索や技術本購入はあえて省きます。

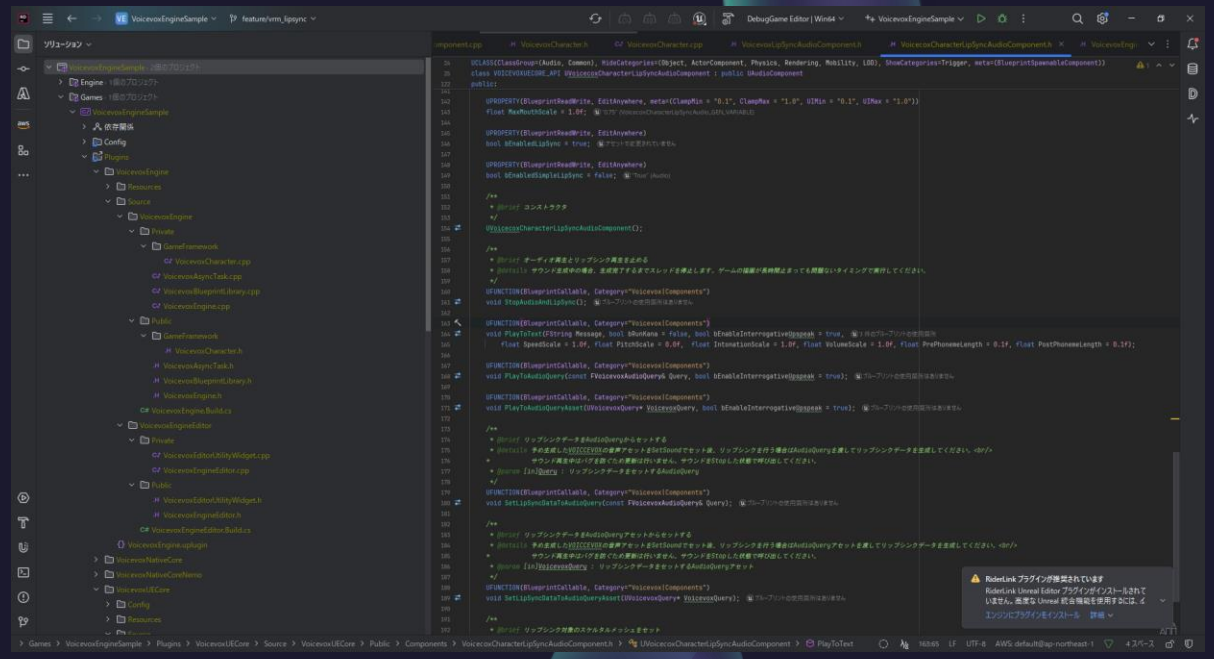
Unreal Engine C++の個人的おススメ学習方法

- JetBrains Riderを使おう
- プラグインを開発してみよう

JetBrains Rider

https://www.jetbrains.com/ja-jp/lp/rider-unreal/

- 非商用利用の場合は**無料**
- クロスプラットフォーム C++ サポート
  - ReShaper C++が組み込まれているので、VisualStudioにない細かい部分をサポートしてくれる
  - Unreal用のVS拡張がデフォルトで入っているような、初心者にかなり優しいIDE
  - 特にMacでC++開発するときは、Riderは開発効率がかなりあがる
- Unreal Engine 向けに最適化
- リフレクションメカニズムと RPC を支援
- コード解析と命名スタイルのインスペクション
  - 最適なコードの提案、コーディング規約もチェックしてくれるので高品質なコードが生産できる
  - 赤ペン先生みたいな機能が素晴らしい
- 最近のバージョンで導入されたAI Assistantがそこそこ使える



プラグイン開発してみよう

- プラグインの作成は以下のドキュメントを見てください
  - **新しいプラグインを作成する**
  - https://dev.epicgames.com/documentation/ja-jp/unreal-engine/plugins-in-unreal-engine#%E6%96%B0%E3%81%97%E3%81%84%E3%83%97%E3%83%A9%E3%82%B0%E3%82%A4%E3%83%B3%E3%82%92%E4%BD%9C%E6%88%90%E3%81%99%E3%82%8B
- uplugin に書き込む内容が追加されるだけで、ソースコード自体はモジュール開発と変わらないので省略します
- エディタ拡張もプラグインで実装すると便利です
  - 描画等、エンジンソースコードの改造が必須になる機能は除きます。

UnrealEngine C++をプラグイン開発で学習するメリット

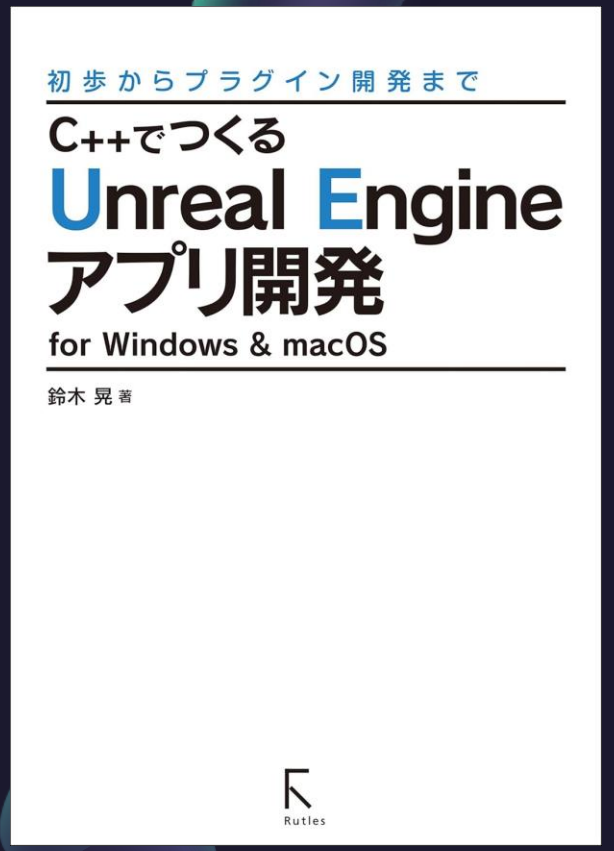
- 小規模な成功体験を得やすい
  - ゲームプロジェクトより小規模な機能をプラグイン化することで、成功体験を積みやすい
    - 個人的になんでもいいのでリリースをすることが一番の学習効果と思っています
  - VOICCEVOX CORE等のオープンソースライブラリをUEに使えるようにする、ということも立派な成功体験
- 汎用的に使用する処理を複数のプロジェクトに使用することが可能
  - 一度組み込んだモジュールを外部のプロジェクトに移植すると変更点が多いので面倒
    - 特にモジュール名を変えた場合、外部公開する際に前に付く、非常に特殊なマクロの変換作業
      - →**[モジュール名]_API**
    - 外部に切り離して初めて発生する動的ライブラリ特有のバグも発生する・・・かも？
- 不要なプラグインはエディタでOFFに出来る
  - モジュールでOFFにするときはBuild.csを修正して再ビルドするので少し面倒

実際にVoicevoxEngineForUEプラグインを開発して得た知識

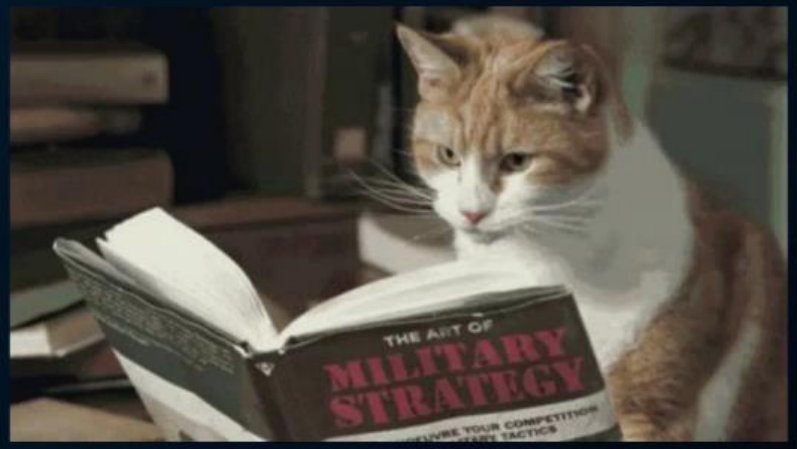
- サードパーティ製を用いたプラグイン開発の知識
- USoundWaveの知見
  - ストリーム再生、及びSoundWaveアセットの生成をC++で書けるようになった
    - **USoundWaveにツール等で生成した音データを入れて再生する(UE5.0〜5.5まとめ)**
    - https://zenn.dev/yuukiogino/articles/0a12c789f8c2e1
- Subsystemの知見を得た
  - 大雑把に説明するとUE版シングルトンパターン
  - Subsystemを利用することでAPIは一緒でも中身が違うDLLを、プラグインの切り替えだけで対応できたのが一番知見としてデカい
    - **オレオレSubsytem管理システムを作る**
    - https://zenn.dev/yuukiogino/articles/1d6138139d85b1

プラグイン開発してみよう

- 迷ったら、この本に記載されている『ファイル入出力プラグイン』をUE5.5で作ってみるのも面白いかもしれません
- 公開されているサードパーティの通信APIとやり取りをするプラグインを作るのもありです
- 小規模なプラグインをガンガン開発し、トライ＆エラーを繰り返して特殊なC++を使いこなせるようになりましょう



大事なのは
インプット と アウトプット
を ガンガン回すこと！



UNREAL ENGINE

猫でも分かるUnreal Engineの学び方 - 超初心者向け編 - 2023 v1.0 より

Unreal Engine C++の勉強に使えそうな資料をまとめました
スライドは公開済みなので、後で見てください

早めに終わりそうなら軽く紹介します。

Unreal Engine 5で極めるゲーム開発　サンプルデータと動画で学ぶゲーム制作プロジェクト

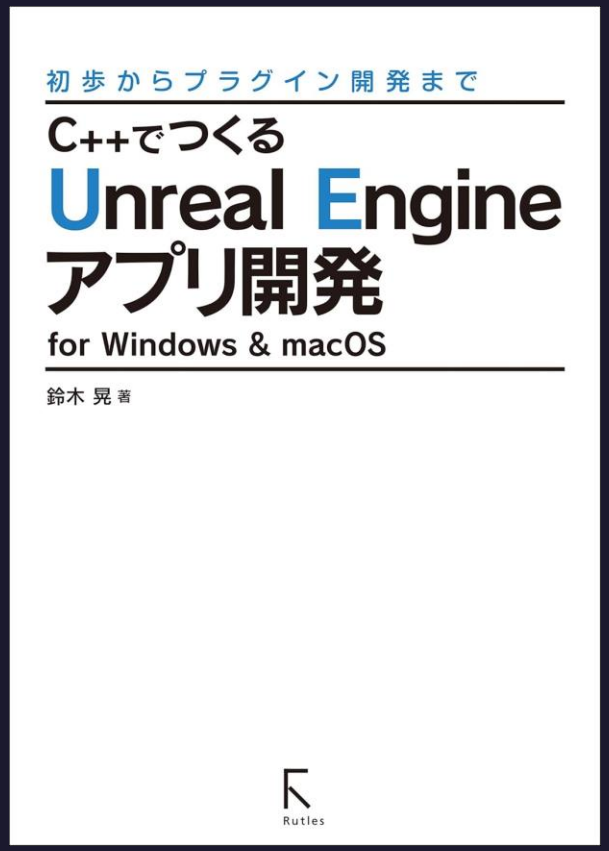
- UnrealEngineを初めて触る人、ゲーム開発の初心者へおすすめ
- C++以外のUnrealEngineの機能を一通り学べる

猫でも分かるUnreal Engineの学び方 - 超初心者向け編 - 2023 v1.0

- EGJ公式のスライド資料
- UnrealEngineを初めて触る人、ゲーム開発の初心者へおすすめ
- 勉強にオススメの資料が一通り載っています
- https://www.docswell.com/s/EpicGamesJapan/KW1WNR-HowToLearnUE5_2023

C++でつくるUnreal Engineアプリ開発 for Windows & macOS

- UnrealEngine C++を取り扱っている貴重な技術本
- 手に入れるなら電子書籍がオススメ
- UE4.18の頃なので古い
- 一先ずC++を知りたい方に



Unreal Engine 5から始める C++ & Blueprint

- UnrealEngine5で C++を触るなら今はこちらがオススメ
- UnrealEngine　C++で主要な部分は一通り学べます
- 有料だけど無料で見れる
- https://zenn.dev/posita33/books/ue5_starter_cpp_and_bp_001/viewer/chap_00_about
- 課金したい人は以下のURL
- https://zenn.dev/posita33/books/ue5_starter_cpp_and_bp_001

アンリアる！ C++入門編 ～対話形式で学ぶUnreal Engine～

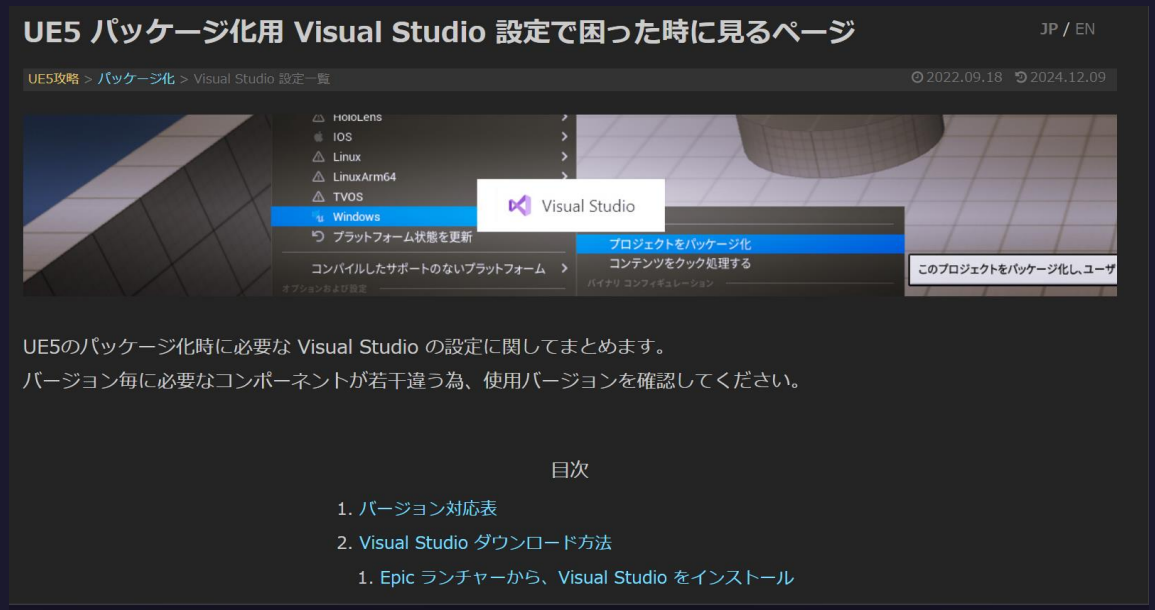
- UnrealEngine5で C++をはじめて触るならこちらもオススメ
- https://booth.pm/ja/items/4734728

C++ プログラミングのチュートリアル

- 公式のチュートリアル
- https://dev.epicgames.com/documentation/ja-jp/unreal-engine/unreal-engine-cpp-programming-tutorials

UE5 パッケージ化用 Visual Studio 設定で困った時に見るページ

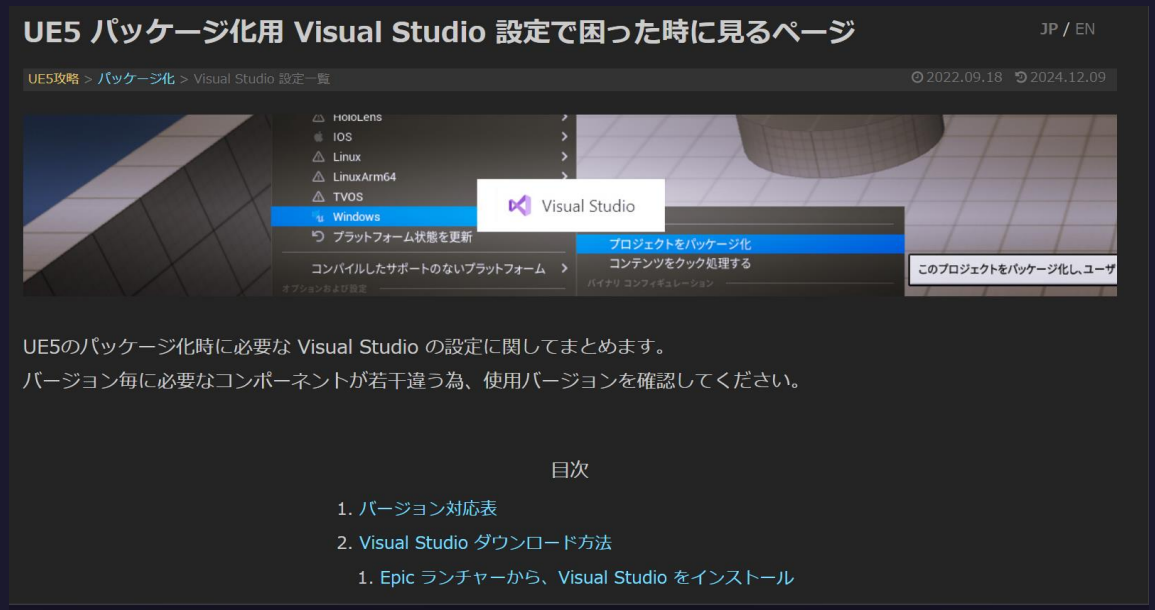
- C++ビルド時、VisualStudioのバージョンをうっかりアップデートしたら動かなくなった人にオススメ
- https://ue5study.com/how/unrealengine-packaging-visualstudio-settings/



UE5攻略リンク

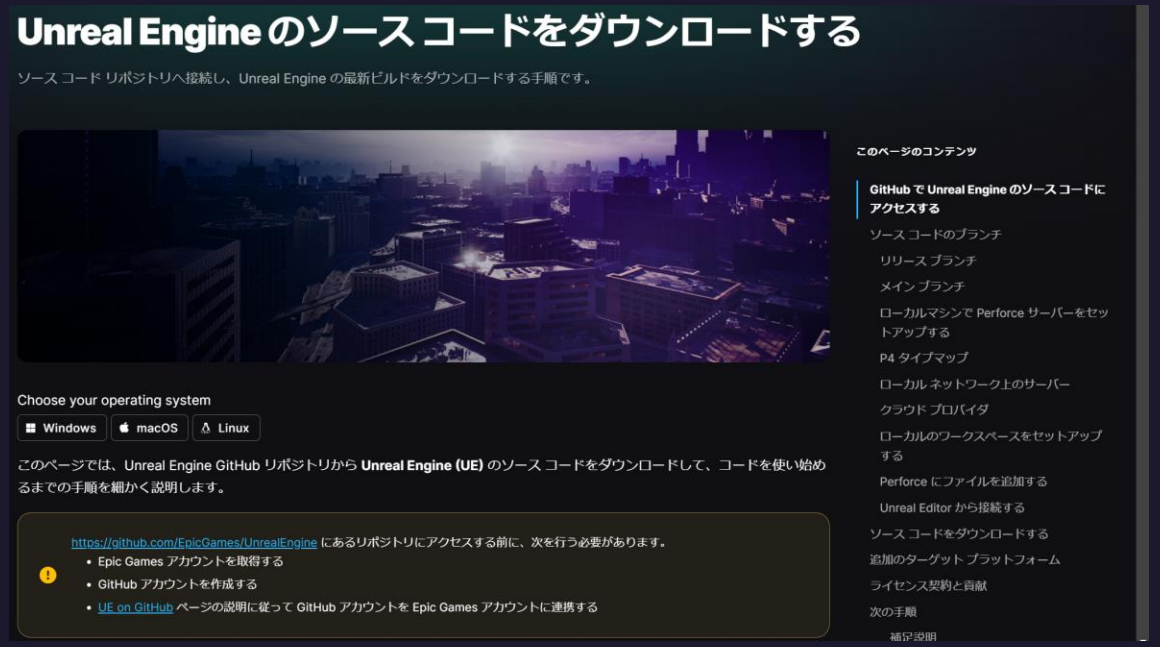
- 公式ドキュメント、ユーザー記事を多くまとめているサイト
- C++も含めて紹介しているリンクが多すぎるので、困ったことがあれば一度見てみるのがオススメ

- 以下はC++のリンク
- https://ue5study.com/unrealengine-cpp/



エンジンのソースコード

- なんでもいいから挙動を知りたい方に
- 最低限、ビルドが独学で出来るスキルは必要です
- 無料で見れますが、EpicGamesにGithub登録してもらう手続きが必要（要EpicGamesアカウント）
- https://dev.epicgames.com/documentation/ja-jp/unreal-engine/downloading-unreal-engine-source-code



ありがとうございました