Program

Creative Note

山内将吾

日本電子専門学校　ゲーム制作研究科

Portfolio 2025

Profile

使用言語・ツール

C++ 1年6ヵ月	C# 2年
Unreal Engine 1年6ヵ月	Unity 2年
Visual Studio / VS Code 2年10ヵ月	Perforce 2年

趣味

- ゲーム
  （第五人格・バウンティラッシュ・APEX）
- 音楽鑑賞
  （Mrs.GREEN APPLE，Vaundy）
- バスケットボール
- カラオケ
- ディズニー
- ゲーム開発・デバッグ

基本情報

氏　　名：山内将吾
生年月日：2004年6月8日
所　　属：日本電子専門学校
　　ゲーム制作研究科
希望職種：ゲームプログラマー

自己PR

私の強みは、「優れた作品や技術から学び続ける姿勢」です。この強みを生かし、
実装だけでなく、映像表現の面でも頼れるプログラマーを目指しています。
そんな頼れるプログラマーには、CGの知見や実際に表現する力が必要不可欠です。
そのため、現在ではシェーダーを活用したビジュアルの向上にも力を入れており、
開発にも積極的に取り入れています。また、開発中により良い表現方法を見出し、
スムーズに実装できるよう、優れた作品を参考にしながら日々学習を続けています。
今後も技術力を磨き、表現と実装の両面で信頼される開発者を目指して努力を続けます。

インセキジャー

プレイ動画

開発期間
2025年6月～現在

ゲーム概要

引力と斥力をうまく活用し、敵に支配された街を救う
3D 磁力ヒーローアクション

メンバー内訳

プランナー：1名、 プログラマー：3名、 デザイナー：5名

担当箇所

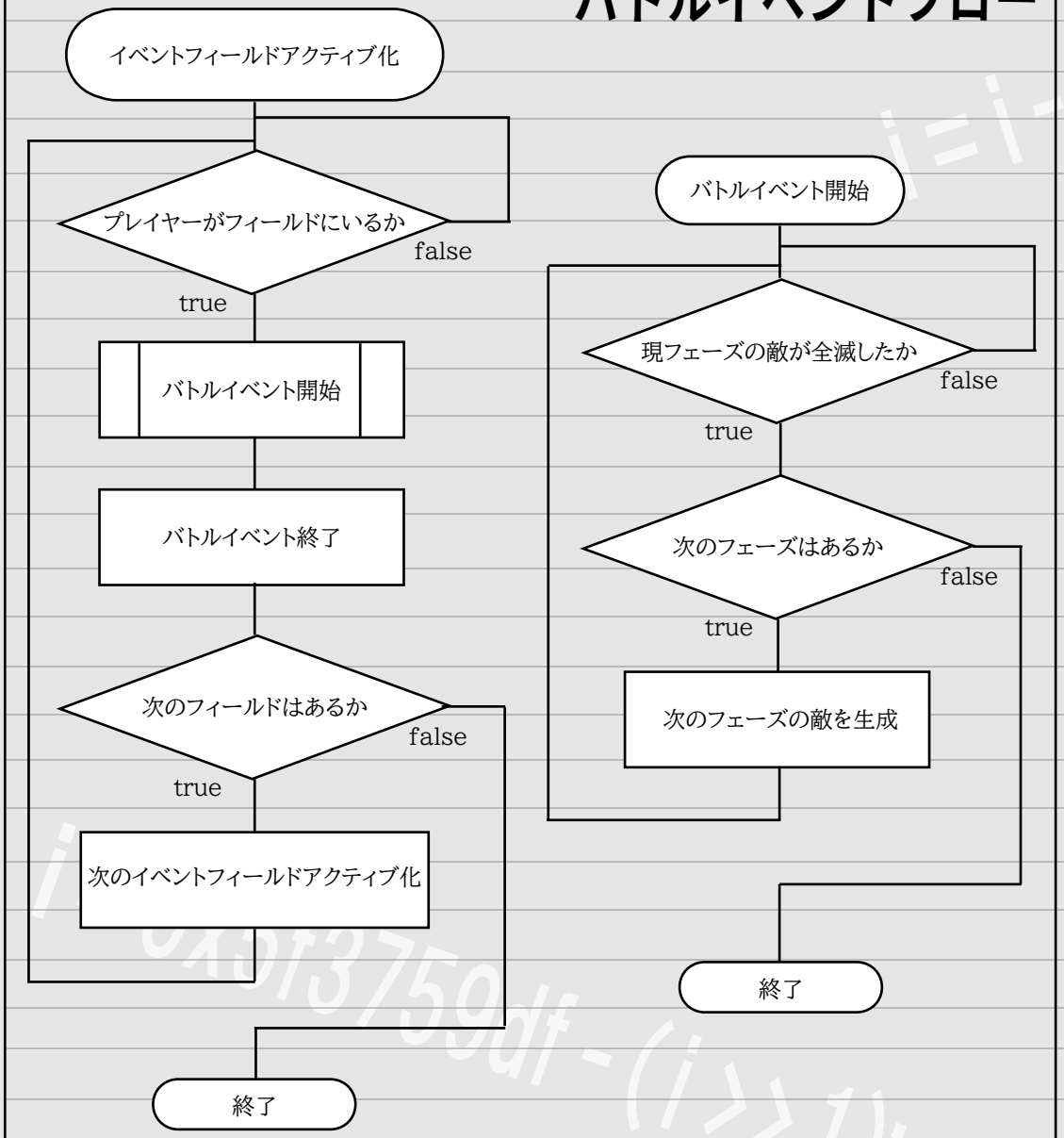
- バトルイベントシステム
- プレイヤーの必殺技挙動制御
- ヒットストップヘルパークラス作成
- VHS 風ポストプロセス
- デモ映像の再生処理

使用言語・ツール

- Unreal Engine 5.6.0
- C++20
- Sourcetree 3.4.17
- GitHub

インセキジャー

バトルイベントフロー

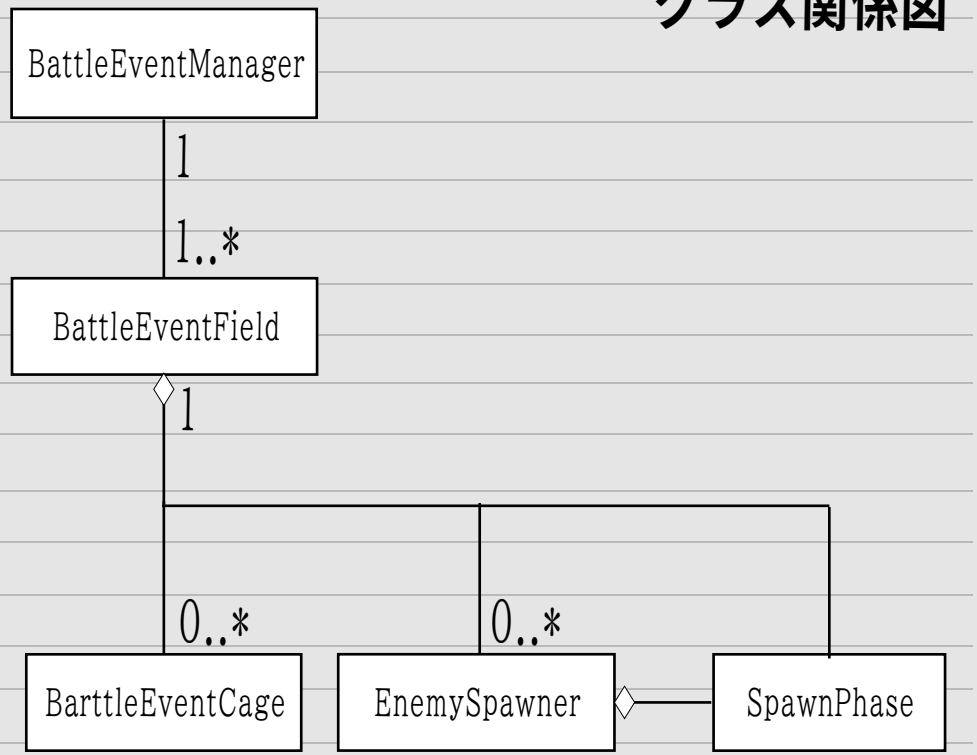


バトルイベントの設計

【概要】

ゲームバランスの調整がしやすいように UE の
レベル上に直接配置してセットするバトルイベントの
システムを実装しました。

クラス関係図



インセキジャー

BattleEventField

```
/*スポナーに敵の生成を促し、敵死亡時のコールバック関数を引数で渡す*/
FScriptDelegate delegate;
delegate.BindUFunction(this, FName("OnEnemyDestroyed"));
spawner->SpawnEnemy(delegate);
```

```
/**
 * @brief 敵が破棄された際に呼ばれる
 */
void ABattleEventField::OnEnemyDestroyed()
{
    /*残りの敵キャラクターの数を減らす*/
    --m_RemainingEnemiesInField;
    --m_RemainingEnemiesInPhase;

    if (m_RemainingEnemiesInPhase <= 0)
    {
        /*次のフェーズがある*/
        if (m_CurrentPhaseIndex < m_FieldPhases.Num())
        {
            /*現在のフェーズ数を更新*/
            ++m_CurrentPhaseIndex;
            /*次のフェーズ開始*/
            StartNextPhase();
        }
        /*フィールド内の敵を全滅させた*/
        else
        {
            /*バトルイベント終了処理*/
            OnEndBattleEvent();
        }
    }
}
```

バトルイベントの設計

【工夫】

イベントフィールドではフェーズごとに敵を全滅させると次のフェーズへ進行します。BattleEventField から EnemySpawner に生成を依頼する際、生成された敵の OnDestroyed にフェーズ進行判定関数をバインドすることで、敵の撃破時のみチェックを行い 処理コストを削減。
さらに、BattleEventField 側が関数を引数として渡す設計により、生成される敵の型を意識せず柔軟に対応 できるようにしました。

EnemySpawner

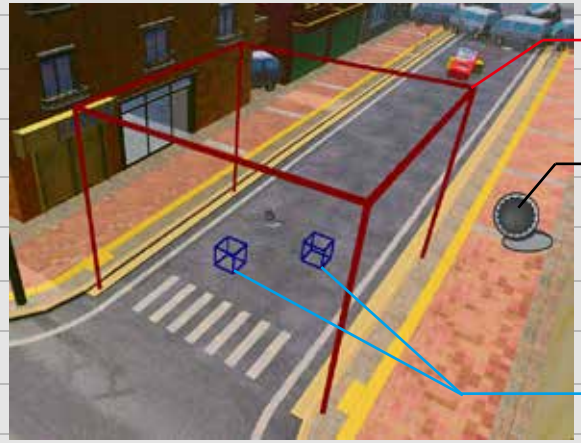
```
/**
 * @brief 敵キャラクターのスポーンを行う関数
 *
 * @param スポーンした敵が破棄されたときに呼ぶデリゲート関数
 */
void AEnemySpawner::SpawnEnemy(const FScriptDelegate& onDestroyedCallback)
{
    if (m_SpawnEnemy == nullptr)
    {
        UE_LOG(LogTemp, Warning, TEXT("No Enemy are set to be spawned by the EnemySpawner."));
        return;
    }

    /*敵キャラクターをスポーンさせる*/
    AActor* enemyActor = GetWorld()->SpawnActor<AActor>(
        m_SpawnEnemy,
        GetActorLocation(),
        GetActorRotation()
    );

    if (enemyActor == nullptr)
    {
        return;
    }

    /*デリゲートに登録*/
    enemyActor->OnDestroyed.Add(onDestroyedCallback);
}
```

インセキジャー



起動すると・・・

実際の動画はこちら

バトルイベントの設計

【工夫】

今回は調整しやすいように UE のレベル上に直接配置してセットするように設計しました。そのためその設置の際の人的ミス（配置し忘れ、アタッチし忘れ）を極力減らすために調整の幅を狭めない程度に出来る限り、プログラム側で設定を行うよう心がけました。

① フィールドは複数の当たり判定があり、BeginPlay() で当たっているスポナーを取得し保持するように設計。
→アタッチなどでスポナーを取得するのは手間であり、アタッチミスなどが起きるかも知れないため。

② フィールドは複数のスポナーを TSet で保持しているため、もし複数の当たり判定にスポナーが当たっていても、実際に配置したスポナーの数分の敵が生成される。
→予期せぬ敵の出現や、フィールドの当たり判定を細かく調整する作業をなくすため。

③ アタッチ操作が多いためもし人的なミスが起きてもすぐ気づくように UE_LOG を使って何が原因かなどを伝えられるようした。
→他の人（主にプランナー）でもスムーズに調整しやすいようにするため。

インセキジャー

VHS なし

VHS あり

実際の動画は[こちら]

VHS 風ポストプロセス

【概要】

VHS 風のポストプロセスマテリアルを
Custom ノードで制作。映像ノイズや歪みを
表現することで、映像全体にアナログ特有の質感を表現。

【工夫】

ポストプロセスマテリアルでは、水平ワブル、
スキャンライン、ノイズライン、縦線ノイズ、
色味調整を組み合わせて表現。
これにより、特撮ヒーロー作品のような質感を持つ
映像効果を実現し、荒廃した街並みの印象を強調しました。

インセキジャー

必殺技の吸い込み挙動

【概要】

プレイヤーキャラクターの必殺技では、ブラックホールのような力によって敵を中心へ引き寄せる処理を実装。
引き寄せ時には、敵がブラックホールの周囲を回転しながら吸い込まれるように移動する計算を行っている。

```
/*目的地に引き寄せられる*/
const FVector currentLocation = actor->GetActorLocation();
const FVector movedirection = (GetActorLocation() - currentLocation).GetSafeNormal();
FVector newLocation = currentLocation + movedirection * m_AttractSpeed * DeltaTime;

/*目的地を中心に周りをまわる*/
const FVector rotationAxis = GetActorUpVector().Cross(-movedirection);
float Theta = 1000.f * DeltaTime;
FVector v2 = -rotationAxis;
FVector verticalV = -movedirection - v2;
FVector w = rotationAxis * -movedirection;
FVector dashVerticalV  = FMath::Cos(Theta) * verticalV + FMath::Sin(Theta) * w;
FVector dashV = dashVerticalV + v2;
newLocation = newLocation + dashV * m_RotationSpeed;

actor->SetActorLocation(newLocation, true);
```

【工夫】

既存関数を使わずベクトル演算で吸引と回転を制御することで、数学的な理解を深めるとともに、処理を軽量化し挙動の制御精度を高めました。
また、ブラックホールのように敵が渦を巻く自然な動きを再現することに成功しました。

実際の動画はこちら

JewelChase

プレイ動画

開発期間
2024年11月～2025年2月
3ヵ月間

ゲーム概要

怪盗が夜の街をかけるランアクション × ガンシューティング

メンバー内訳

プランナー：1名、 プログラマー：4 名、 デザイナー：1 名

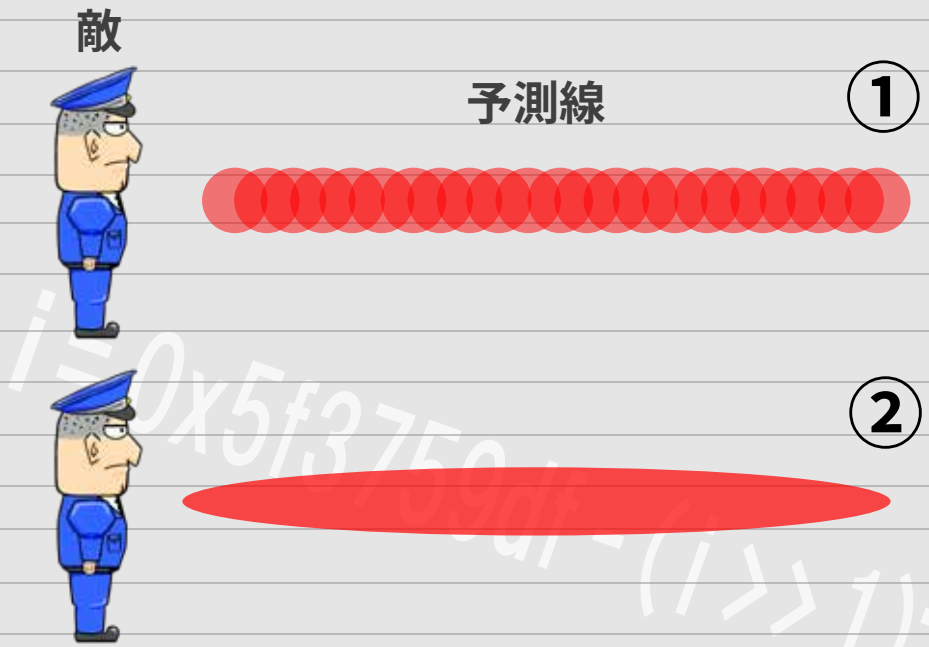
担当箇所

- 予測線の挙動実装
- プレイヤーのノックバック処理
- プレイヤーの点滅処理
- 敵キャラクターの挙動制御

使用言語・ツール

- Unreal Engine 5.3.2
- C++20
- Sourcetree 3.4.17
- GitHub

JewelChase



予測線の挙動制御

【概要】

SplineComponent を使い、赤い予測線専用のアクターを Spline の長さ分引き延ばして予測線を表現。
またアタッチされたアクターに依存しない、
LineControllerComponent を実装し制御している。

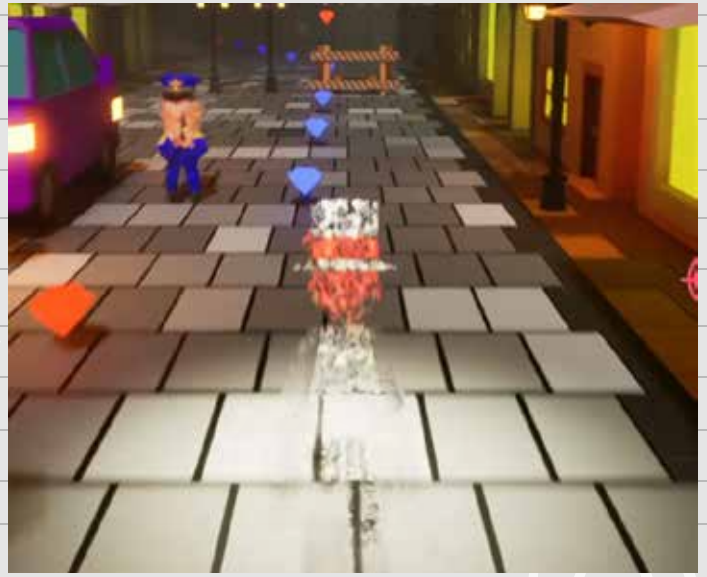
【工夫①】

実装当初は SplineComponent 上に予測線用のメッシュを並べて疑似的に表現しようとしていました。（ 左図①参照 ）
しかし、今回の予測線は動的に始点や終点が変わる仕様にしたため処理負荷が高く、一斉に予測線を複数出すことは難しい状態でした。そこで SplineComponent の長さ分予測線用のメッシュを引き延ばし表現することで、処理コストを削減させる事に成功しました。（ 左図②参照 ）

【工夫②】

攻撃予測線の表示については、「視認性と緊張感のバランス」を重視してプレイヤーに理不尽な攻撃だと思われぬよう、
試行錯誤を重ね調整しました。最終的に、
「攻撃直前の短時間表示」を採用することで、プレイヤーに瞬間的な判断のスリルと、回避成功時の爽快感を
与える演出を実現しました。

JewelChase



実際の動画は[こちら]

プレイヤーの点滅処理

実際の動画はこちら

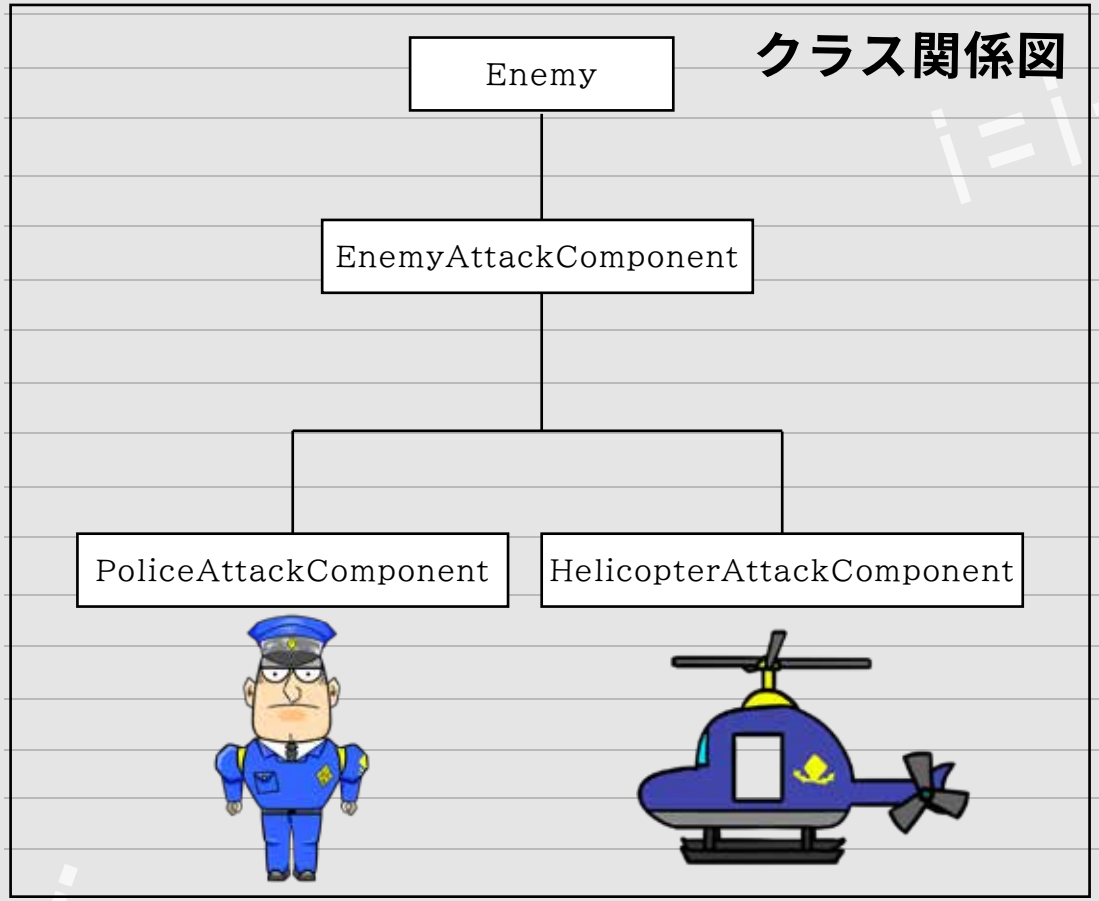
【概要】

Unreal Engine の Dither TemporalAA を使い、
プレイヤーがダメージを受けた際の無敵時間を表現しました。

【工夫】

Sin カーブを用いることで透明度が滑らかに変化するよう
調整しました。また、無敵時間の残りに応じて点滅間隔を
徐々に短縮することで、無敵状態の終了が分かりやすくなるよう
工夫を施しました。また、透明化の表現には
ディザー（ディザリング）を採用しました。このディザーによって
生じるポリゴンのエッジのギザギザとした表現が、このゲームの
カートゥーン調のデザインに調和し、視覚的な魅力を高めました。
これにより、プレイヤーに対する視認性を向上させ、
ゲームの操作性とユーザー体験を向上させることができました。

JewelChase



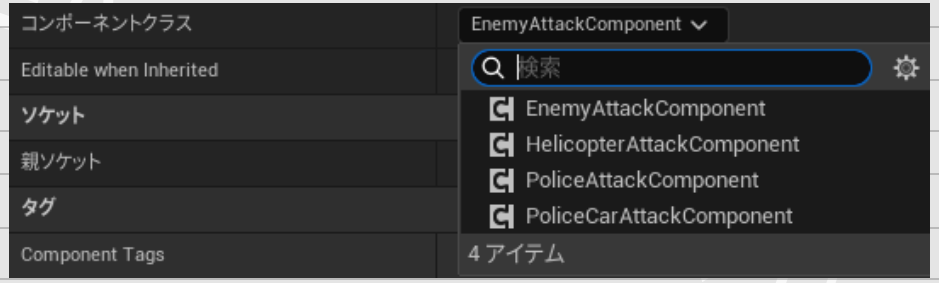
敵キャラクターの設計

【概要】

Enemy クラスは敵キャラクターの基本機能を提供し、
全敵に共通する挙動（攻撃、死亡、ダメージなど）を
各コンポーネントとして分離しています。

【工夫】

新たな敵キャラクターを追加する際に、
基本となる Enemy クラスを継承した新しい C++ クラスを
作成する必要がなく、Blueprint 上で見た目や
挙動コンポーネントを切り替えるだけで済むようにしています。



各敵キャラクターの BP で
使うアタックコンポーネントを指定

最後までご覧いただきありがとうございます。

作品 URL

もしよろしければこちらもご覧ください。

https://bit.ly/4m7jpgU