イスを支える技術

株式会社インターネット立方体 / 代表取締役
📦 まめお (@PaperSloth)

目次

目次

会社概要：インターネット立方体とは

- 渋谷のゲーム会社
- 右図のものを開発協力しました

(今回は右図の詳細は語りません)

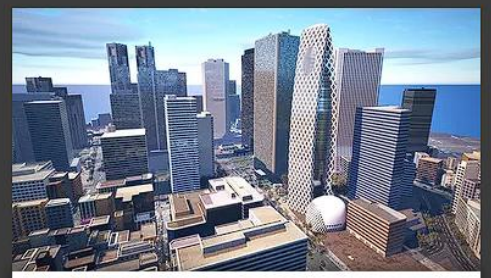
ゲーム会社ではあるものの
VR/ARの開発やC++でのツール制作
UE <-> Unityの移植等も行っています
あとVtuberのオーガナイザーとか

4月からのお仕事募集中

VRoid SDK for Unreal Engine
Tool / Unity -> C++ & UE5
C++, UE5でのプログラミング全般

Palworld
Game / UE5
一部開発協力

非公開Project
R&D / UE5 -> Unity
Unity移植の上でのプログラミング／絵作り全般



非公開Project
R&D AI / UE5
一部開発協力

イスとは

イスとは

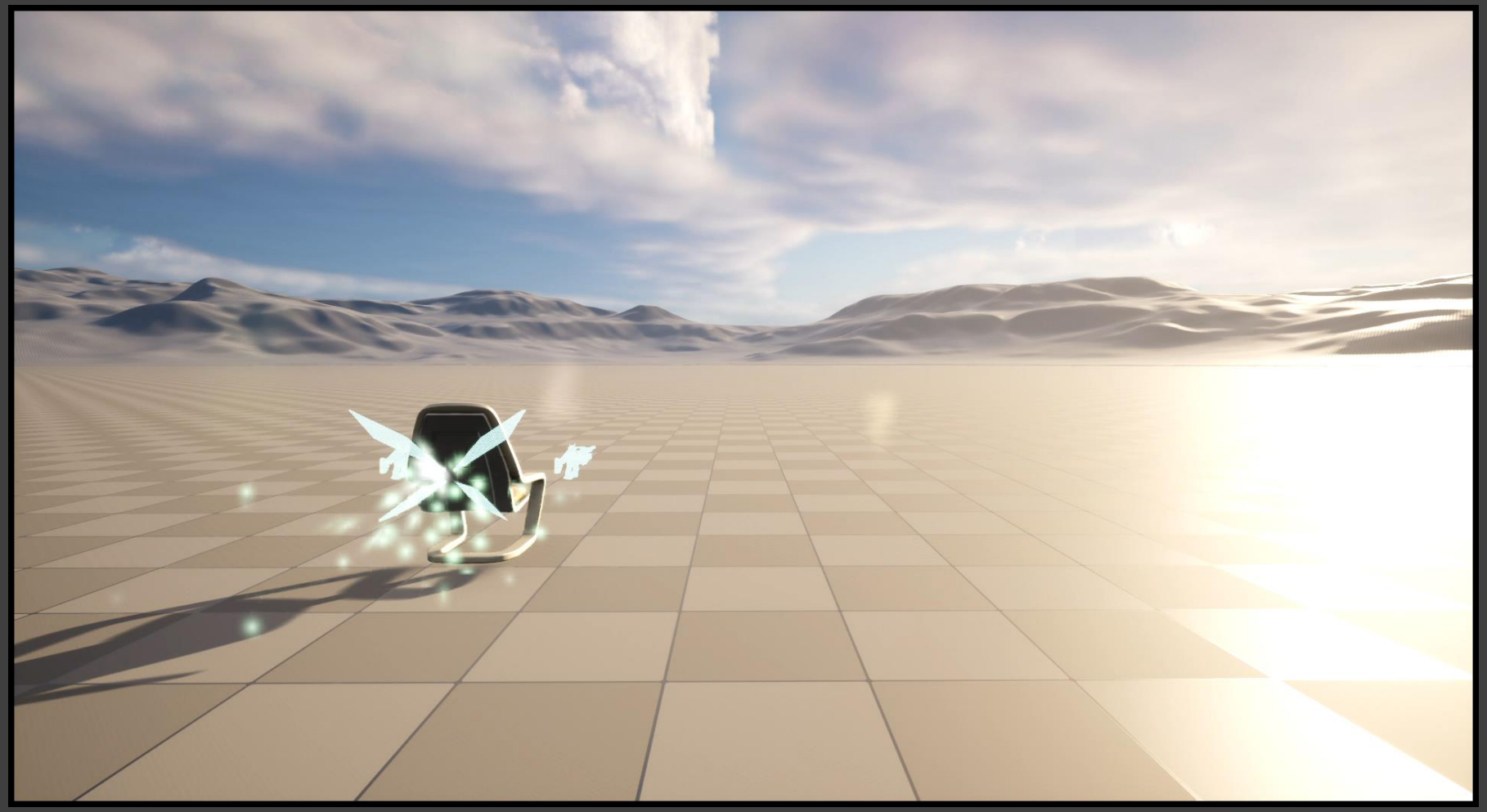
- ジャンル：ハイスピードアクション
- 2017-2018年に制作していた個人制作ゲーム

  C92(100部完売), デジゲー博2017(100部完売)
  C94(150部数頒布), デジゲー博2018(100部頒布)
- 開発は2, 3割がC++で基本的にはBlueprint
- UE4.19 ～ UE4.21で開発

目次

- Introduction
- 各種機能の実装について
- 実装見送りになった検証要素
- プロジェクト運用について
- 最後に

クイックブーストの実装について



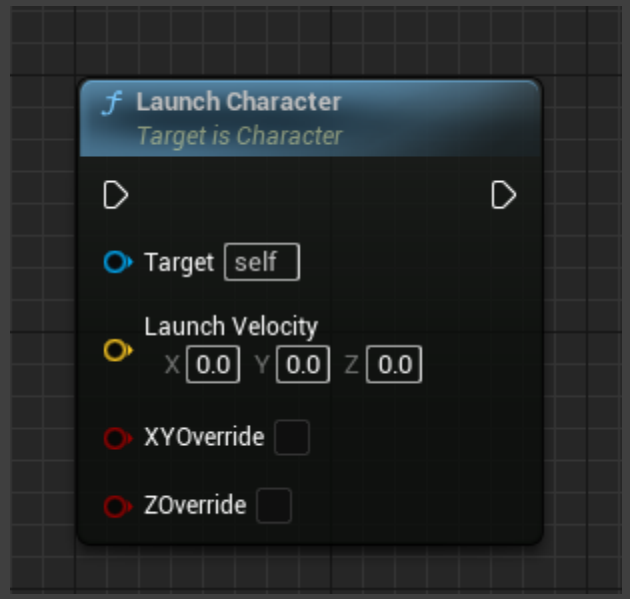
クイックブーストの実装について

実装自体はCharacterクラスのLaunch Characterを使います

Launch: 矢/ロケット等を勢いよく発射する

簡単にCharacterを吹き飛ばせる便利なノードです。

高速回避や吹き飛ばし等色々なケースに使えます。



クイックブーストの実装について

LaunchCharacterは便利なノードですが注意点があります。

Movement Modeが強制的にFalling(落下中)に更新されてしまいます。

以下はUE5.3のCharacterMovementの処理です。

```
bool UCharacterMovementComponent::HandlePendingLaunch()
{
    if (!PendingLaunchVelocity.IsZero() && HasValidData())
    {
        Velocity = PendingLaunchVelocity;
        SetMovementMode(MOVE_Falling);
        PendingLaunchVelocity = FVector::ZeroVector;
        bForceNextFloorCheck = true;
        return true;
    }

    return false;
}
```

クイックブーストの実装について

そこでイスでは自前のLaunch関数を用意
強制的にVelocityを上書きします
こうすることでFallingに遷移せず
かつ、キャラクターを吹き飛ばせます
LaunchCharacterの処理とは異なりますが
今回実装したい要件は満たせました。

LaunchVelocityには
入力時のInput情報に基づき
前後左右の移動ベクトルを生成
そのベクトルにブースト量を掛けています

```cpp
void AIsuCharacter::CustomLaunch(const FVector& LaunchVelocity, const bool XYOverride, const bool ZOverride)
{
    if (GetCharacterMovement() == false)
    {
        return;
    }

    FVector finalVelocity = LaunchVelocity;
    const FVector CurrentVelocity = GetVelocity();

    if (XYOverride == false)
    {
        finalVelocity.X += CurrentVelocity.X;
        finalVelocity.Y += CurrentVelocity.Y;
    }
    if (ZOverride == false)
    {
        finalVelocity.Z += CurrentVelocity.Z;
    }

    GetCharacterMovement()->Velocity = finalVelocity;
}
```

独自のLog実装について

UE_LOGの改良

Unreal C++でLogを出力する場合にはUE_LOGマクロを使用します。

構文：UE_LOG(CategoryName, Verbosity(LogType), Format, ...);

例　：UE_LOG(LogTemp, Log, TEXT("LogMessage"));

UE_LOG(LogTemp, Log, TEXT("TestValue=%d"), TestValue);

プロジェクト独自の場合

プロジェクト独自のログカテゴリの定義

定義　(header)DECLARE_LOG_CATEGORY_EXTERN(LogIsu, Log, All);

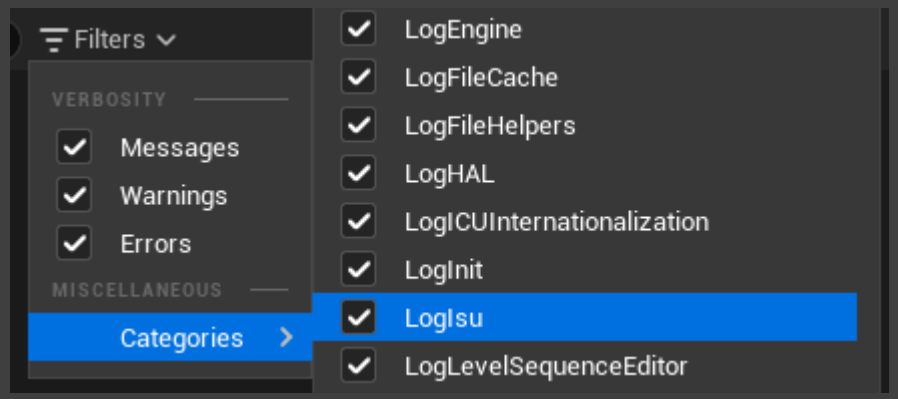
(cpp)　DEFINE_LOG_CATEGORY(LogIsu);

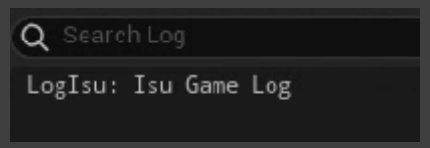
呼び出し

UE_LOG(LogIsu, Log, TEXT("Isu Game Log"));

独自のLog実装について

プロジェクト独自のログカテゴリを使用することで
OutputLog内のFilterができたり、Log検索がしやすくなります





独自のLog実装について

ですが、先程のコードを都度記述するのは面倒なのとUE_LOGで検索も手間なので
別途LOG用のMacroを用意します。
以下のように記述すれば、使用箇所の検索も楽になるため便利です。
適当にLog用のheaderを作成しておくとよいでしょう。

```
DECLARE_LOG_CATEGORY_EXTERN(LogIsu, Log, All);

#define ISU_LOG(fmt, ...) UE_LOG(LogIsu, Log, fmt, ##__VA_ARGS__)
#define ISU_WARNING(fmt, ...) UE_LOG(LogIsu, Warning, fmt, ##__VA_ARGS__)
#define ISU_ERROR(fmt, ...) UE_LOG(LogIsu, Error, fmt, ##__VA_ARGS__)
#define ISU_FATAL(fmt, ...) UE_LOG(LogIsu, Fatal, fmt, ##__VA_ARGS__)
#define ISU_DISPLAY(fmt, ...) UE_LOG(LogIsu, Display, fmt, ##__VA_ARGS__)
#define ISU_VERBOSE(fmt, ...) UE_LOG(LogIsu, Verbose, fmt, ##__VA_ARGS__)
```

独自のLog実装について

また、先程の記述ではPlugin作成時等で
別のModuleで使用したい時に使用できなくて困ります。

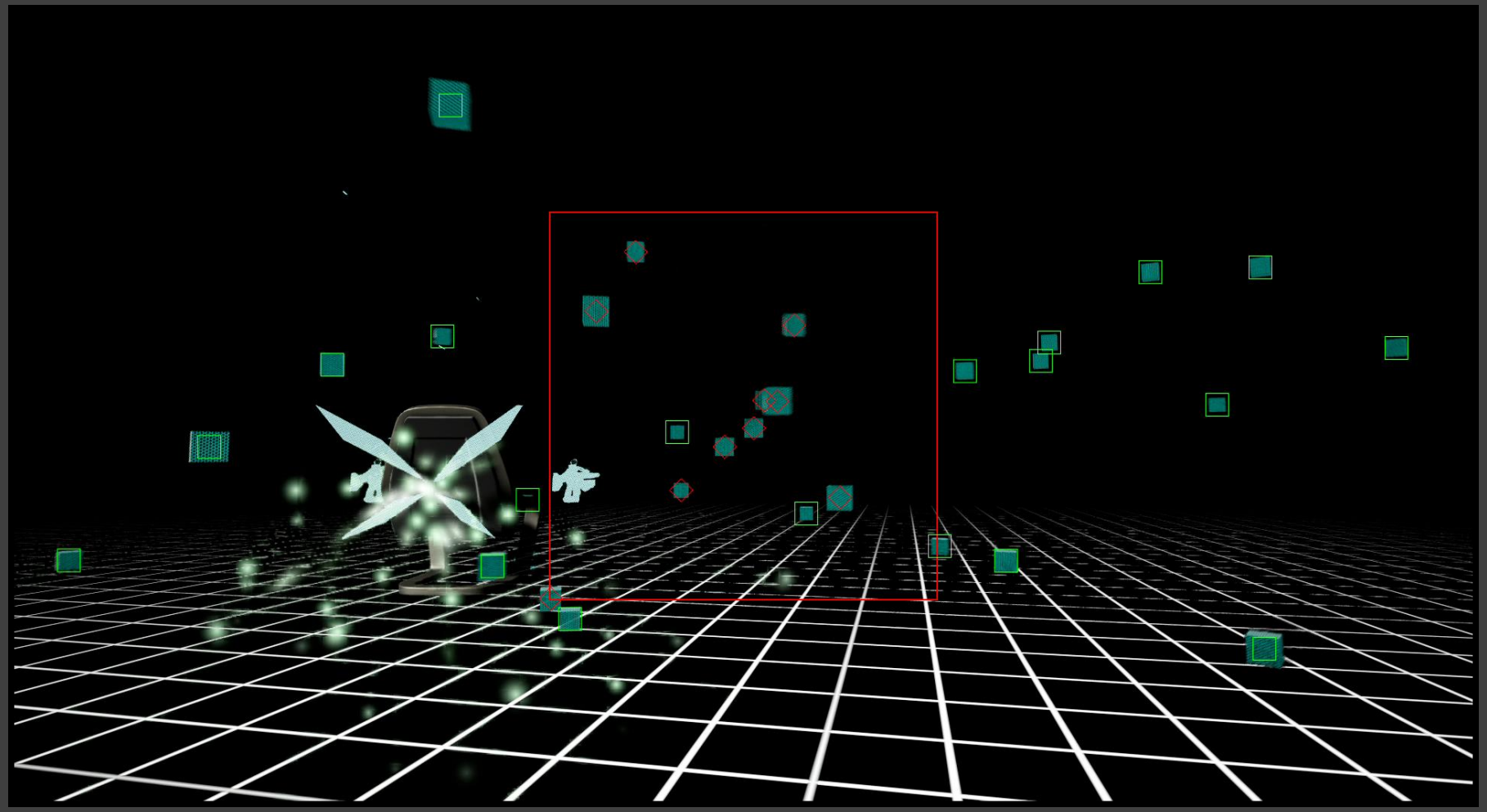
今後Moduleを追加する予定がある場合は以下のように
ModuleName_APIを追加しておけば他のModuleでも使用できるためオススメです。
構文

```
(ModuleName)_API DECLARE_LOG_CATEGORY_EXTERN(Log(ProjectName), Log, All);
```

例

```
ISU_API DECLARE_LOG_CATEGORY_EXTERN(LogIsu, Log, All);
```

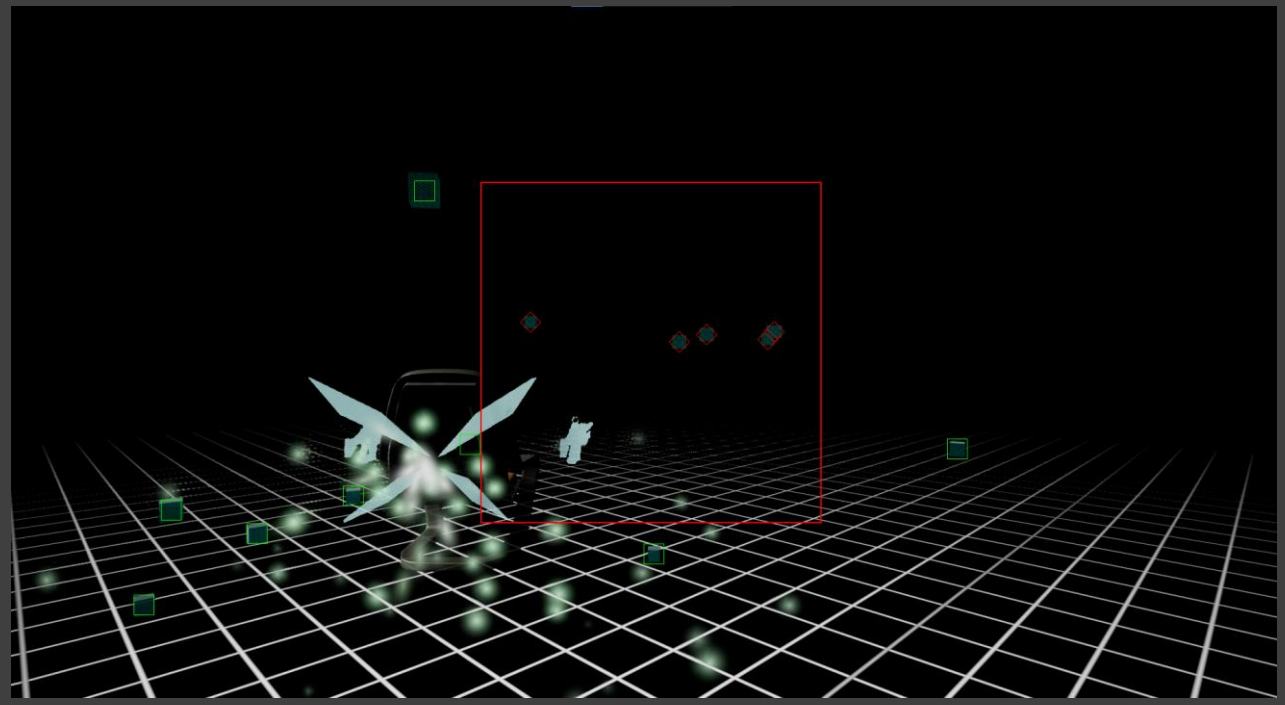
Lock On Systemの実装



Lock On Systemの実装

イスではロックオン可能対象には緑の枠を表示
ロックオンした対象には赤の枠を表示しています

画面中央の大きな赤枠は
Debug用のロックオン可能範囲の表示です



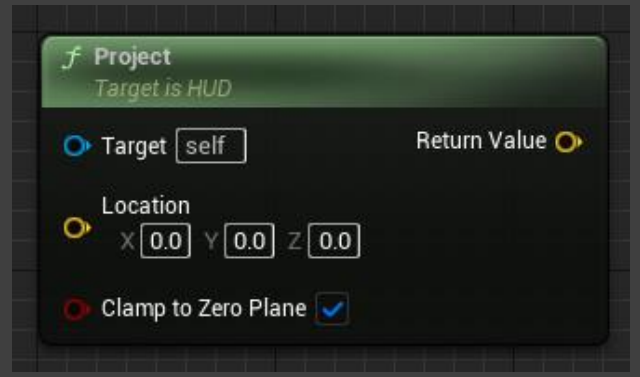
Lock On Systemの実装

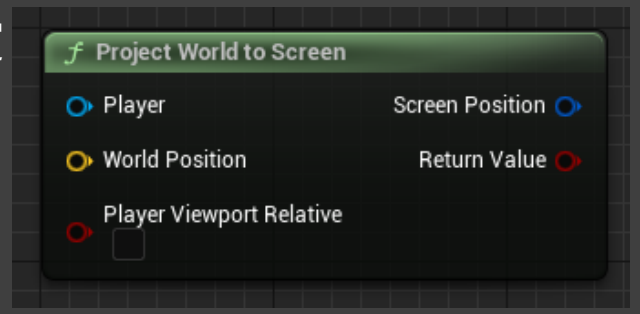
ロックオンの処理としては簡単に説明すると以下のフローで組んでいます。

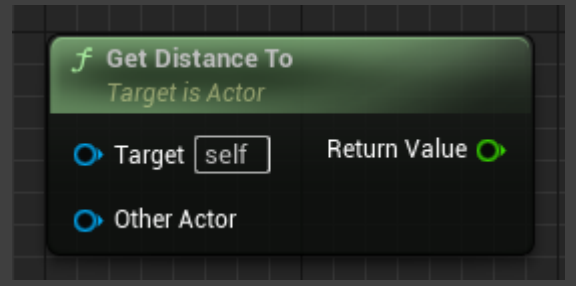
1. ロックオン可能な対象の配列を取得 (後述)

2. HUD Blueprint内のProjectノードで3D座標 -> 2D座標への変換
   後ろにいる敵はロックオンしないため、カメラより奥側にいるかどうかを判定
   ※ProjectノードはReceiveDrawHUD内でしか使用できないため要注意
   代替手段としてはProjectWorldToScreenノード

3. 次に敵とPlayerとの距離をGetDistanceToノードで取得
   一定範囲内であればロックオン







Lock On Systemの実装

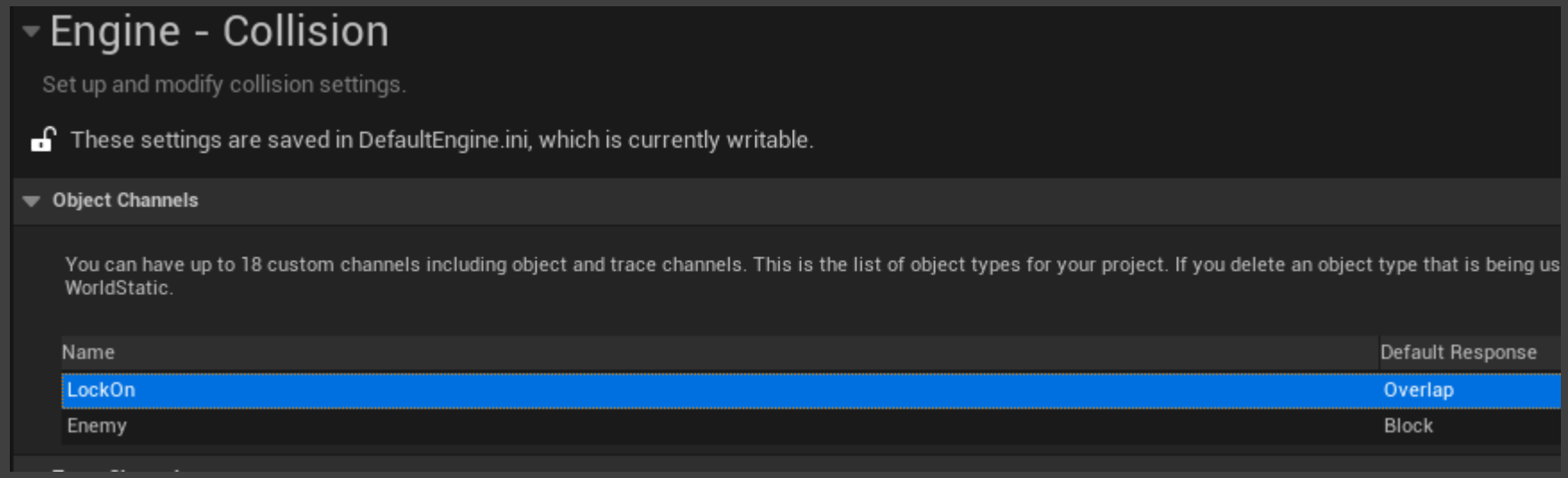
ロックオン可能な対象の配列の登録 / 削除の手法について

　一言で書くと大きいSphereを用意してBegin Overlap / End Overlapを使用

まずはProjectSettings内のCollision内の

Object Channels 内にLockOn用のChannelを追加

ついでに敵のChannel も追加



Lock On Systemの実装

ロックオン可能な対象の配列の登録 / 削除の手法について

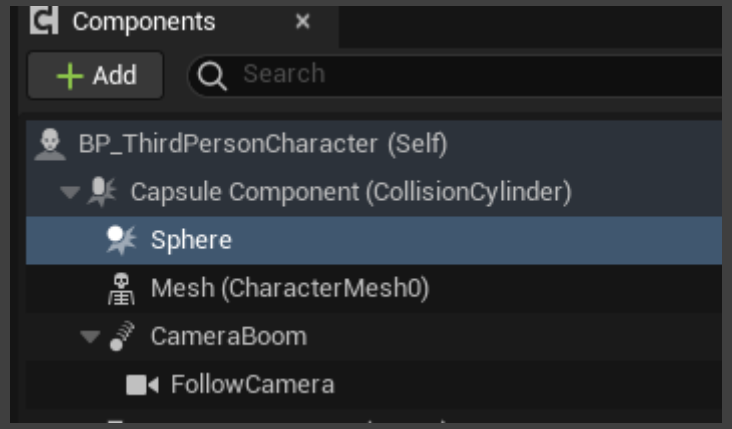
続いて、Sphere CollisionをプレイヤーBPに追加

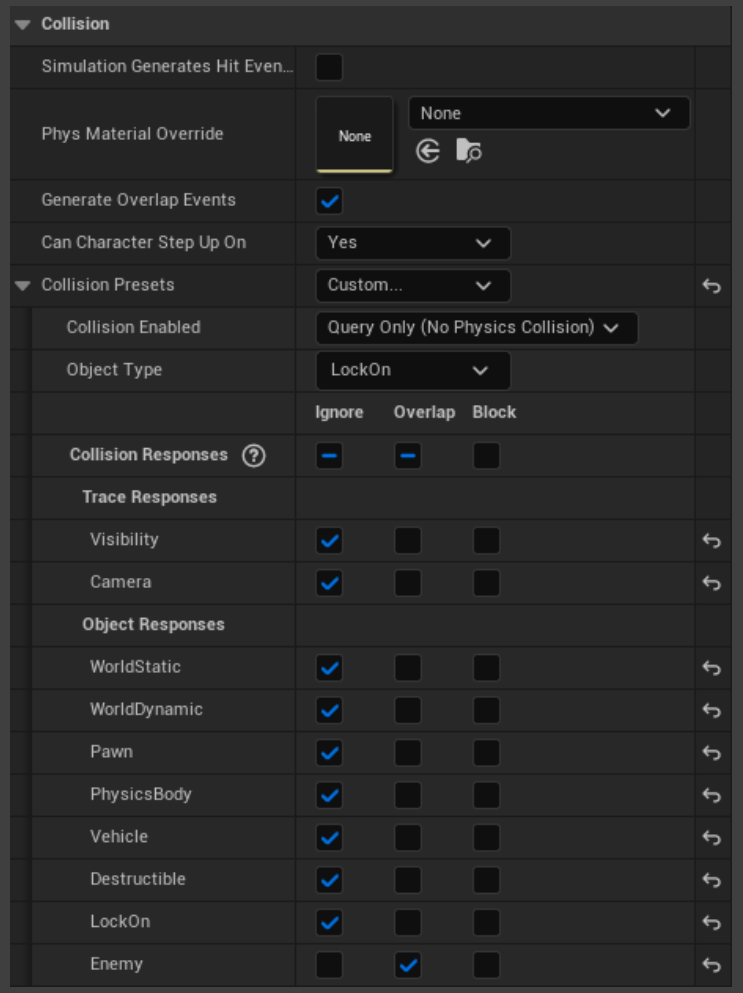
(説明のためThirdPersonCharacterを使用)

次にCollisionの設定を敵や弾等、追加した対象だけをOverlapにチェック

LockOn可能距離はSphere Collisionの半径で調整ができるため

かなり調整と管理が楽になります。





Lock On Systemの実装

ロックオン可能な対象の配列の登録 / 削除の手法について

最後に、Begin Overlapで登録処理 / End Overlapで削除処理

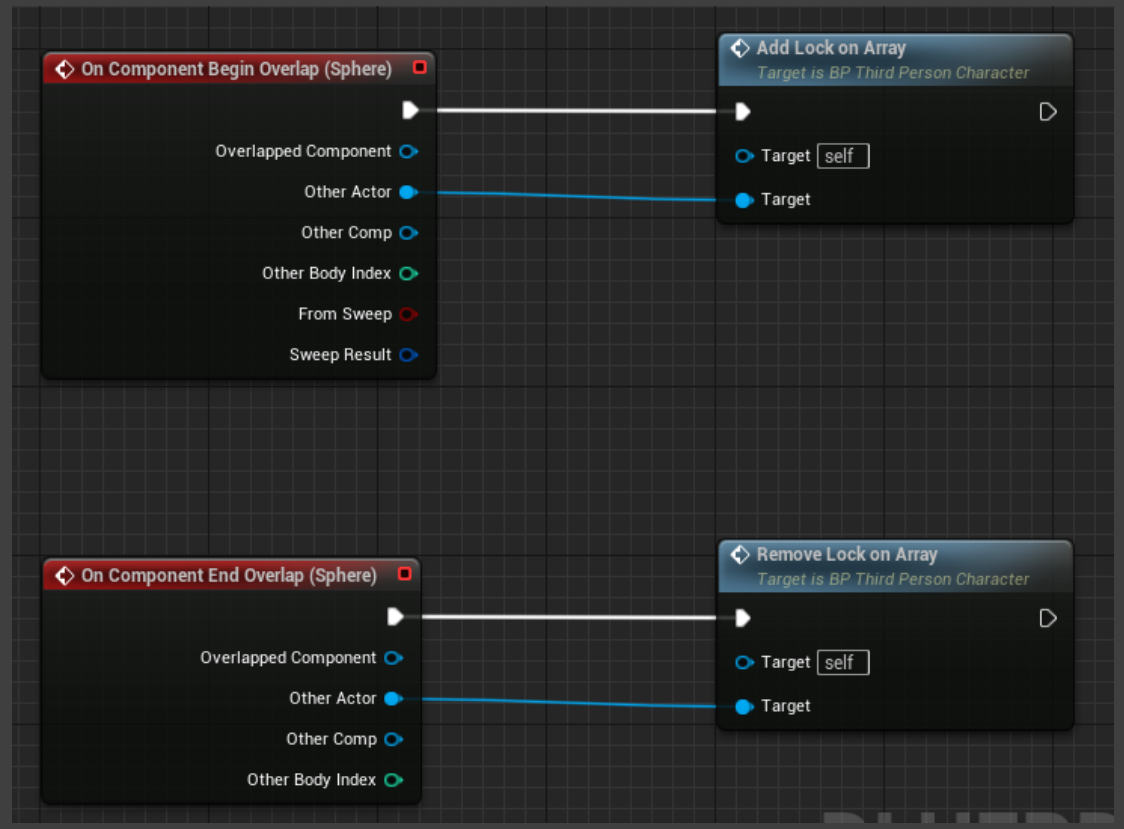
あと、敵が破壊された時にも

配列に存在すれば削除処理もお忘れなく

登録にあたってはActorHasTag等を活用して

敵か味方か弾か等の判定を行います。

これで360°の一定範囲内の敵を配列に登録できます。



使用アセット紹介

- Multi Lock-on System

  非売品になっていました。

  Multi Lock-onのサンプル、美しいホーミングレーザーの実装が入っていて

  大変お得なアセットだったのですが・・・

- Projectiles Pack

  SF系のかっこいい攻撃要素が一通り入っています(最新版でさらに増えてそう)

  マルチミサイル、ブラックホール発生装置、レーザービーム、レールガン、ブレード群...etc

  https://www.unrealengine.com/marketplace/ja/product/projectiles-pack

最適化について(少しだけ)

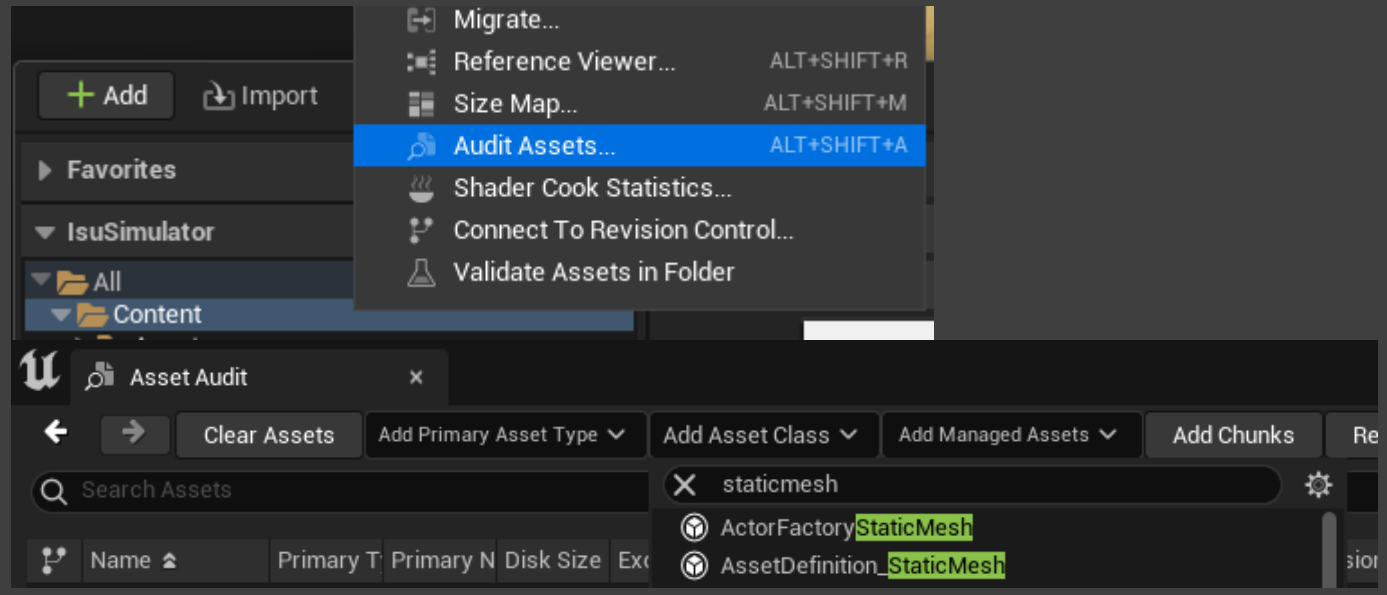
- Audit Assets...(アセットの監査)

  Asset Audit Windowが開きます。

  Add Asset Class でStaticMeshやTextureを追加して、該当アセットの一覧が見れるようになります。

  重いアセットを発見したらReference Viewerで参照されているかを必ず確認します。

  未使用のアセットをいくら軽くしてもしょうがないですからね。



最適化について(少しだけ)

- Textureの最適化(設定見直し)

  Texture一覧を表示したあとに、FormatやDimensionsでSortします。

  圧縮Formatの選択と実際の圧縮が正しく行われているか

  そもそも圧縮自体が行われていないというケースもあります(購入したAsset等)

  テクスチャ解像度が2の累乗や4の倍数になっているか

  LOD Biasを設定している場合、LODとTexture解像度でゲームへの影響の確認。...etc

Compression Settings	Dimensions	Format ▾	Has Alpha Channel	LOD Bias
BC7	128 × 128	BC7	True	0
BC7	256 × 256 × 256	BC7		0
HDR Compressed	1,024 × 512	BC6H		0
Normalmap	16 × 16	BC5	False	0
Normalmap	16 × 16	BC5	False	0
Normalmap	1,024 × 1,024	BC5	False	0
Normalmap	16 × 16	BC5	False	0
Normalmap	16 × 16	BC5	False	0
Normalmap	16 × 16	BC5	False	0
Normalmap	512 × 512	BC5	False	0
Normalmap	512 × 512	BC5	False	0
Normalmap	512 × 512	BC5	False	3
Normalmap	32 × 32	BC5	False	0

最適化について(少しだけ)

- StaticMeshの最適化(設定見直し)

  Static Meshの場合はCompressed SizeやLODs等を見直しています。
  1アセットで数百MBも容量をとるようなアセットが含まれてしまう場合もあります。
  また、大きなアセットでもLODを適切に設定してやったりReductionしたりして
  ある程度容量の削減が行なえます。

Approx Size	Est Total Compressed Size	LODs	Vertices
3,747,716 × 3,747,715 × 3,756	469.094 KiB	1	8,194
23,064 × 23,064 × 14,562	11.3 KiB	1	312
8,192 × 8,192 × 8,192	90.882 KiB	1	2,208
8,192 × 8,192 × 8,192	87.924 KiB	1	2,208
8,192 × 8,192 × 8,192	88.955 KiB	1	2,208
25,500 × 25,500 × 600	1.445 MiB	1	260,100

最適化について(少しだけ)

・そんなことよりMemReport –full

Audit Assetで使われているか分からない各Assetを調べて、使われていたら調整

そんな地道なことをするよりもMemReportを叩いて使用されているアセット一覧を見るのが早いです

/Saved/Profiling/MemReports 以下に保存されます。 (PIEではなくDevelopment BuildやTest Build推奨)

数万行のテキストファイルが作られた！こわい！でも色々なことがわかる便利なコマンド！

・使用メモリ

・オブジェクト一覧

・動的にSpawnしたActor一覧

・テクスチャ一覧

・スタティックメッシュ一覧

... etc

最適化について(少しだけ)

- 最初に見るのはほんの一部だけで良い

  memreportを行うと色々な情報が見れます。

  大量に使われてるComponentや特に容量を食っているClass等。

  - Mem FromReportを見てみよう

    Physical Memory: xx MB used, xx MB free, xx MB total

    物理メモリにどれくらい余裕があるのかが見れる

  - listtextures uncompressedを見てみよう

    非圧縮のTexture一覧が出力されています。

    設定ミスや解像度がおかしいせいで圧縮できていないTextureを駆逐しましょう。

最適化について(少しだけ)

- 最初に見るのはほんの一部だけで良い
  - obj list class=StaticMesh –resourcesizesortを見てみよう
    異常なまでにメモリ食べてるやつがいる！とここで気付くことも。
    LODの設定が多すぎないか、適切かを見極めよう。

  - ListSpawnedActorsを見てみよう
    Level上で生成されたActor一覧が見れる。
    多すぎる場合はObject Poolを作成して再利用する等

他にも色々なメモリの使用状況がわかる。
限界までチューニングするのは仕事でない限りは避けて良いものの
家庭用ゲーム機で動かしたい場合はハードごとのチューニングもまた必須。

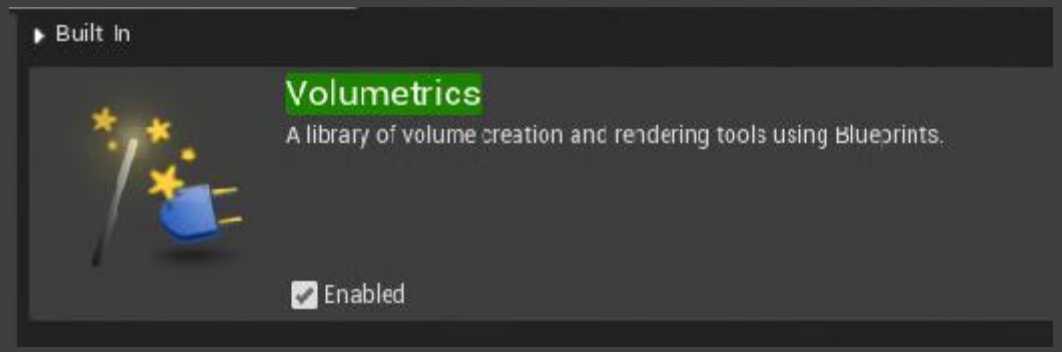
目次

雲海での戦闘マップ

雲海での戦闘マップ

通常、雲海を描画する際にはRay marchingを使用して描画することが多いです。
ですが、そういった要素を実現するにはグラフィックス専任のエンジニアが
かなりチューニングをしてやって動くというものです。

でも、Unreal Engineなら簡単に雲海が作れる！
そう！Volumetrics Pluginを使えばね！
　※UE5ではまともに動かなかったと記憶しています。



雲海での戦闘マップ

なぜ見送りになったのか

答えは単純でめちゃくちゃに重くてゲームでは使い物にならなかったからです。
当時、RTX2070を積んだPCで30-40fpsくらいまでしか出ませんでした。
雲の描画だけで、です。

なんとかゲームに載せられないかと調整を続けてみて
50-60fpsまで出るようになりました。
ですがRTX2070を積んだPCで、です。
また、当時のSpecですので、今でいうとRTX 4070くらい積んでようやくみたいな状況です。
ユーザーのみんながみんなRTX4090を積んでいれば・・・というわけにもいかないので見送りになりました。

海(水面)での戦闘マップ

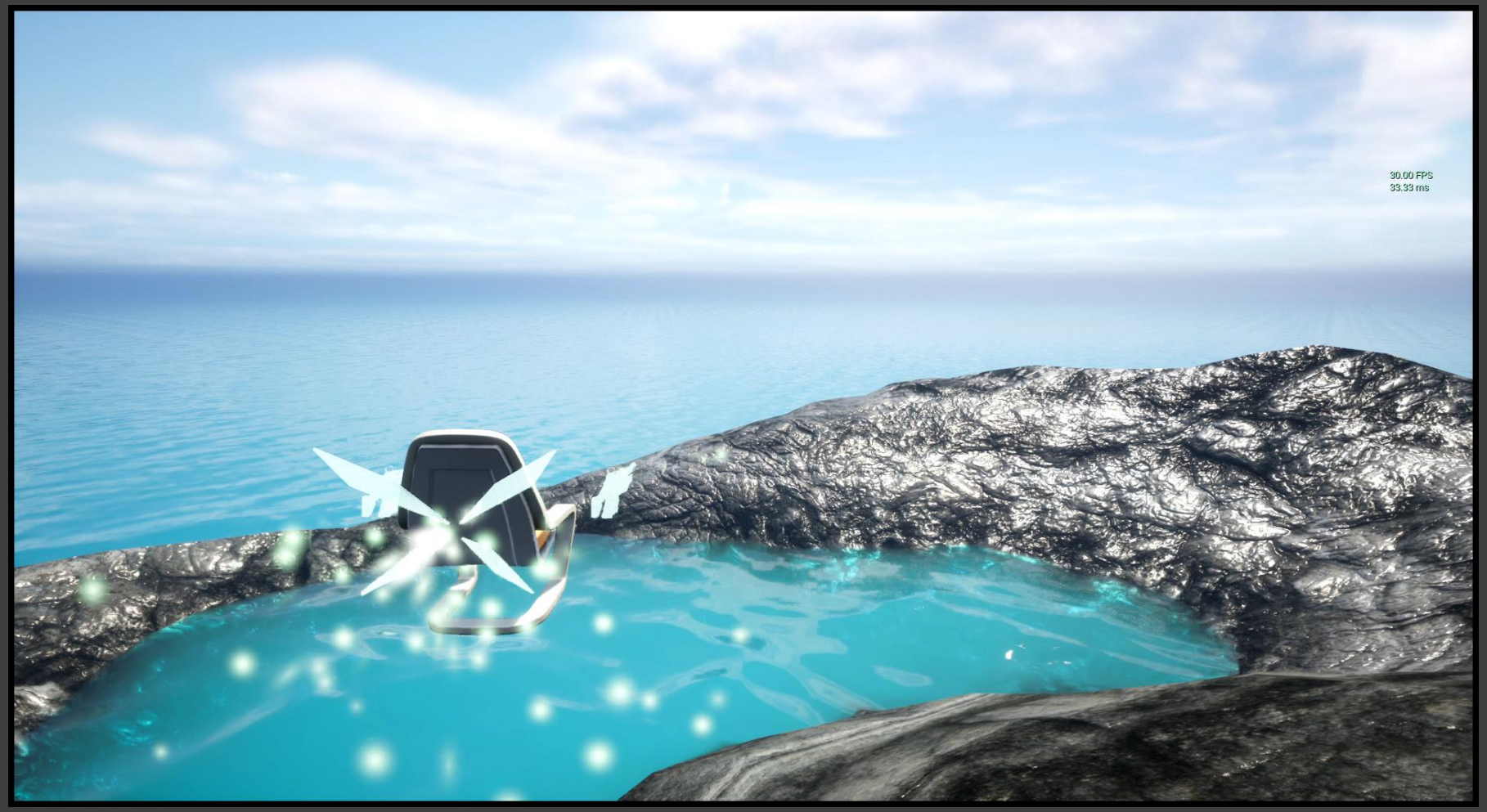
海(水面)での戦闘マップ

今度は雲海と違ってパフォーマンスは問題がありませんでした。
RTX 3050 Ti Laptopで60fpsが維持できています。
Spec的には GTX 780 < 3050 Ti Laptop < GTX 970くらいの性能です。

これならリリースできそう！
ですが、今度の問題は技術的な課題で一旦見送りに。
RenderTarget3枚(512x512)をふんだんに使用して水面の揺れを表現しています。
今回の課題としては、あまり大きな水面にすると破綻してしまうという問題がありました。
また、調整コストが大きくこれを作り込むよりは別のマップを作成した方が早いという判断です。

UE5だとWater Pluginでいい感じにできそうな気もしますね。

自動生成Map



自動生成Map

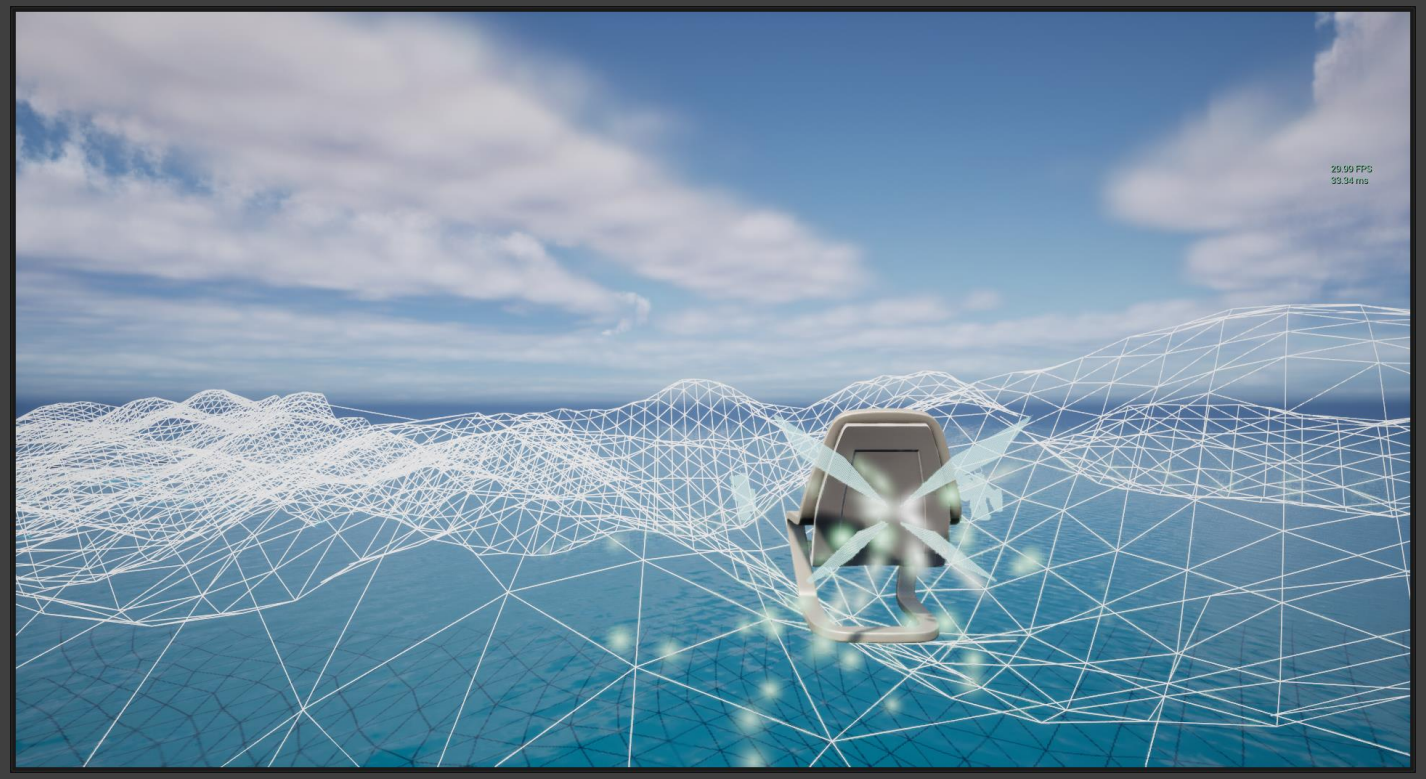
Procedural Mesh Componentを使用した自動生成のLandscape

~~人類は皆一度はProceduralでオープンワールドを作ろうとする愚かな生き物~~

以前より、自動生成で楽出来ないかなーと思い調査を進めていたもの

Procedural Mesh Componentの使用は
Pluginから有効化する必要があります

また、C++で作成しており
DependencyModuleに
“ProceduralMeshComponent”
の追記が必要



自動生成Map

結果的としては、ある程度スペックがあれば問題なく動作。
レベルデザイン面に関してもイスの場合であれば
無限ダンジョン的な感じで生存時間や撃破数を競うミニゲームとして使える

また、Procedural Meshを使用している都合でSectionという単位でエリアを区切っており
Section毎に敵の生成やアイテムの生成といったロジックもある程度コントロールできる

~~実装に飽きてやめた~~
それよりもイスにおいてはステージ数や敵の種類、武器の種類等の
コンテンツを増やす方向性に注力した方が良いと感じたため一度実装を中断。

結局日の目を見ることはなかった。

目次

Unreal C++のプロジェクト構成

- Unreal C++のソリューション構成について

  通常C++コードを追加した場合は以下の構成になります。

  (UE4の頃)

  ProjectRoot

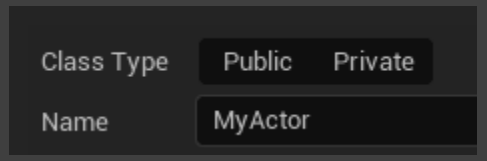
  +- Source

  +- ProjectName　実際にはModule名

  +- Private　cppファイル

  +- Public　headerファイル

  UE5だと新規C++クラス追加時にPrivate / Publicの箇所がデフォルトではオフになっています。



Unreal C++のプロジェクト構成

ここで本題です。デフォルトの構成だと全てのクラスが1つのフォルダに入っていて
Solution Explorerでかなり見にくいですし、クラスの管理もしにくいです。
そこで、本プロジェクトでは以下の構成にしています。
UE5内のSource/Runtime/... 以下に近しい構成で
Robo Recall Mod Kitのファイル構成が特に参考になります。

```
ProjectRoot
    +- Source
        +- ModuleName     プロジェクトの開発コード等
            +- Character          キャラクター関連の実装(cpp/header)
            +- Weapons            武器関連の実装(cpp/header)
            ...etc
```

ノーディング規約

- そもそもコーディング規約とは

  ノーディング規約なんて言葉はおそらく世の中にまだ存在しないと思います。
  通常、プログラミングを行う際にはコーディング規約というものを決めます。
  チーム開発を円滑に進めるために、コードを記述する際のルール決めを行うわけです。
  ありがたいことに、UEの場合は公式docがありますので、そちらを参照するとよいです。
  https://docs.unrealengine.com/5.3/ja/epic-cplusplus-coding-standard-for-unreal-engine/

ノーディング規約

- ノーディング規約とは

  コーディング規約同様に、ルール決めを行い

  これをBlueprint、Material、NiagaraのModule Scriptにも当てはめて運用します。

  UE4/5 を扱っている各社でもほとんどのケースで存在しないと思います。

  各社の登壇内容を見聞きしていると大半が以下の運用になります。

  ・アーティスト、プランナーが組んだ処理に対してエンジニアがたまにレビューをする

  ・重い処理を見つけたらピンポイント(ここ大事)で最適化する

  ・そもそもBlueprintを使わない(さすがにAnimation Blueprintは使ってるよね・・・)

  ...etc　これらが悪いという話ではないです。

  むしろピンポイントでの最適化やたまにレビューをするというのはコスト的にも理に適っています。

ノーディング規約

- 実際に使用しているノーディング規約
  1. ノードは直線、もしくはZの形状で行う (左から右、上から下に追える形状に配置)
  2. ワイヤーの交差は極力回避する
  3. ピンの参照先が複数に渡る際はReroute Nodeで重なりを回避する
  4. 極端に肥大化した際はEventGraphを分割する
  5. 関数、変数、マクロに対してカテゴリを適切に設定する
  6. コメントに対して色分けを行い、軽い処理と重い処理を視覚化する
  7. Macroは極力避ける(コンパイル時に展開される都合で実行速度の低下を招く恐れがある)
  8. Function / Macroにはアセット参照を持たせない
     特にFunction Library, Macro Libraryに持たせると無駄なアセット参照が増える

ノーディング規約

話すと長いので色分けについてだけ
以下のルールに沿って色分けを実施

- 緑

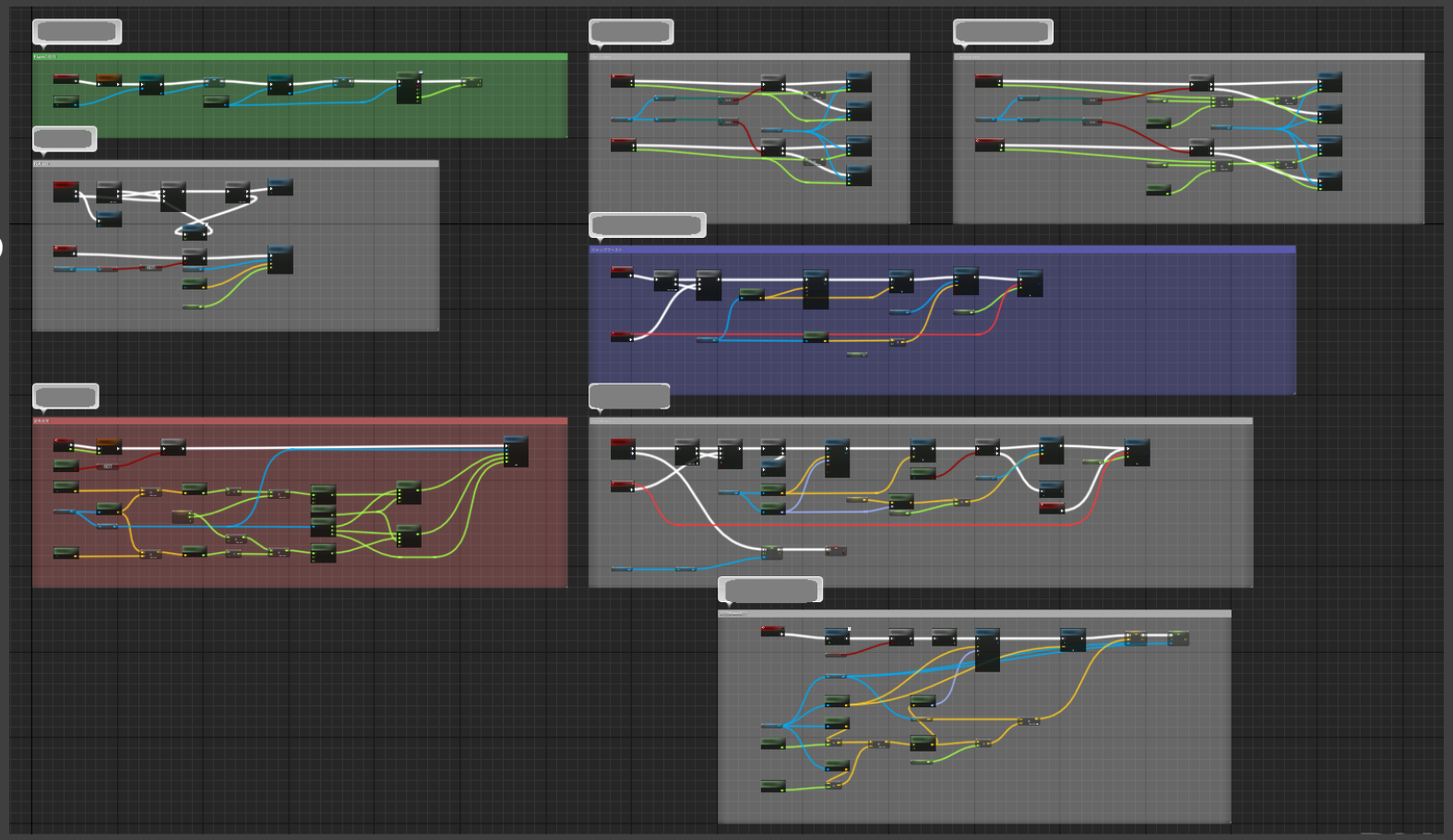
  BeginPlay等 初期化に使用するもの
  1回の呼び出しだけで負荷が軽いもの

- 赤

  Tick等 毎Frame呼び出されるもの
  負荷的に重いもの

- 青

  実験的な機能、検証用ノード



目次

VRoid SDK for Unreal Engineについて

VRoid SDK for Unreal Engineについて

弊社が開発させていただいているUE5向けのPluginです。
オリジナルアバターで遊べるアプリを簡単に！
というコンセプトのプロジェクトになっています。

初期のバージョンではRetargetに多くの課題がありましたが
現在はBlender, Unityで変更を加えたUniVRMを含めて数多くのVRMを動かせるようになりました！

是非とも、現在開催中の第21回UE5ぷちコン等で活用いただければと思います！
たくさんの開発者様からのフィードバックをお待ちしております！！！

参考資料

・[UE4] UE_LOGについてあれこれ / ヒストリア開発ブログ

https://historia.co.jp/archives/5532/

・[UE4] Objコマンドによるオブジェクト解析 / Qiita(どんぶつさん)

https://qiita.com/donbutsu17/items/dd9e00bee27d6868ed3d

・『グランブルーファンタジー Project Re:LINK』におけるリアルタイム雲レンダリングのノンフォトリアル表現と高速化事例 / CEDEC2017

https://cedec.cesa.or.jp/2017/session/ENG/s5922ab3980408.html

・ブループリントを書くにあたって大切なこと /出張ヒストリア

https://www.docswell.com/s/historia_Inc/ZRG73K-ss-64773800#p1