プログラム解説

１：制作期間

制作期間は 2021/9～2021/10 と 2022/2～2022/5 の約 6 ヵ月です。空いた期間は GFF Award というゲーム制作コンテストに提出するゲームを UnrealEngine4 で作っていました。

9 月は車や子供、ボスなどの敵の処理をモックとして作成し、ゲームとしての面白さを確認し、企画の一部変更や開発スケジュールの作成などを行いました。

10 月はマップの作成と AI のテンプレート構築を行い、AI を敵アクターごとに適用しました。また、アイテムを作成しました。

2 月はマップの拡大を決定し、マップを作り直しました。また、マップの拡大に伴い車の移動処理、アルゴリズムを変更しました。

3 月は当たり判定関係の再構築、UI の作成、追跡型ボスの追加、ライブラリの作成を行いました。

4 月はアイテム関係の再構築、追跡型ボスの追跡処理の変更、UI 関係の再構築、子供アクターの再構築を行いました。

5 月はサウンドの作成、カットインの作成、ボスのアルゴリズムの再構築、レベルデザインの見直し、変更を行いました。また、アイテムの追加作成、UI の最終調整を行いました。

２：プログラムのアピールポイント

・ポリモーフィズム

後からオブジェクトを追加しやすいように、オブジェクトタイプごとに基底クラスを作っています。

アイテムなどは、アイテムの使用処理や、位置情報更新、投擲されたときの処理などは共通の処理が多いため基底クラスに作成し、それを継承するだけで同じ処理が使用できるようにしました。また、アイテムごとにスピードやダメージ、特性は異なるため、そこにも注意して臨機応変に対応できるように作成しました。

・ファクトリーメソッド

ポリモーフィズムで行ったように似た処理を行うオブジェクトを簡易に追加できるようにしています。しかし、これは似たような、ではなく完全な同一処理を行うもの（UI 関連や落ちているアイテム等）だけ行っています。

std::vector を用いて生成リクエストを各クラスから直接行うことで、シングルトンパターンの乱用にも注意し、余分な配列宣言によるメモリの消費の対策も行いました。しかし、この方法は、スマートな反面処理コストが高く、ゲームが重くなる原因であるため頼り過ぎないように注意しました。特に、新規リクエストはゲーム中に行うのは不向きだと考え、極力生成はタイトルからプレイシーンに遷移するタイミングにまとめ、描画をフラグで管理することでゲームのスピード感を守りました。

・UnrealEngine4 の当たり判定システムの再現

ゲームエンジンと同じ当たり判定構築に挑戦し、ほぼ標準ライブラリのみで開発に成功しました。

当たり判定が欲しい静的オブジェクトクラスは右の図のような情報を入力し生成リクエストを行うだけで、生成されている全ての当たり判定オブジェクトとの判定を、コリジョンマネージャーが行い結果を出してくれます。動的オブジェクトは、これに加え、位置情報と処理するかどうかの更新が必要ですが、これにより、当たり判定処理はただ生成、更新するだけで出来るようになりました。

３：プログラムを組む上で苦労したこと・工夫したこと

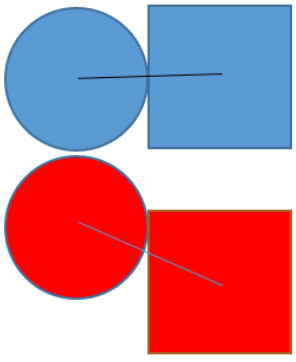
１番苦労したのは車の交差点移動処理です。街の中を車が走る、ただそれだけであれば私が苦労することは無かったと思います。全ての車が決まったルートを走るパターンを作ってみたのですが、レベルデザインの関係でマップを拡張した時に、そのプログラムは意味の無いものに変わりました。

指定位置情報、ルート番号に依存したプログラムではレベルデザイン変更を行いにくく、難易度調節の幅が狭まってしまいます。そのため、車が自分で道を決めて進み、ランダムに街の中を走るプログラムを作ろうと考えました。結果、マップを大幅に拡大しようと変更後のマップの新規交差点にボーンを入れ、そのボーンに対応する進行可能方向テーブルを更新すれば、自分の進行方向に交差点がある場合に進行可能方向の中からランダムで道を判定して右折なら交差点の奥で、左折なら手前で曲がるようになりました。

また、当たり判定にも悩みました。UE4 の当たり判定システムの生成リクエスト時に当たり判定をどのような形にするかを決めるのですが、それには球と直方体の 2 つのパターンがあります。そして、その球と直方体の当たり判定の方法でずっと迷っていました。インターネットで球の直方体の当たり判定を調べると、大体は角の座標をとって判定する場合でした。しかし、それだと角が球に触れていない場合は当たり判定が取れません。辺に触れる場合も当たり判定がとりたいと考えました。そして、最も近い球の表面の座標を求める方法を思いつきました。方法は以下の通りです。

- 球の位置座標から直方体の位置座標までの距離を三平方の定理で求めます。
- その距離の中から球の半径が占める割合を求めます。
- 球から直方体までの変位ベクトルを求めて、それに先ほど求めた割合をかけると、球の位置座標から最も直方体の座標に近い球の表面の座標までの変位ベクトルが手に入ります。

この方法で座標が手に入れば、あとは点と直方体の当たり判定なので出来ると思ったのですが、この計算で出された点が赤の右図の地点だとうまくいきません。この対策として、触れている場合に取れる直方体の向いている面の 4 点と、球の位置座標までの高さを用いて出した 4 角錐の体積は一定になる、という仮説を立てて実行していたのですが、どの方法もそれなりに問題点が出てきて、逆にすべての条件を通るものにすると処理回数が当たり判定リクエスト分なのもあり負担が多くなるという問題に直面しました。そのため、



妥協案ではありますが、二つ目の最もわかりやすく条件も網羅できていて処理も速い、球の表面座標を求めるものを採用ました。