プログラム解説

１：製作期間

約５か月（11 月下旬～４月下旬）

〔具体的な作業の内訳〕
＜11 月下旬～１２月上旬＞
・プレイヤーの通常処理、移動や回転などのプレイヤー関連の処理を実装。
・敵の通常処理、追いかける処理やランダムに動く処理を実装
・当たり判定の実装
・ステージ制作、実装
＜１２月中旬～１月上旬＞
・プレイヤーにくっつく物質の制作と実装。
・プレイヤーの動きに応じたカメラの距離計算の制作と実装。
＜１月中旬～2 月中旬＞
・タイトル、ゲームプレイ、ゲームオーバー、ゲームクリアの実装。
・エフェクトの実装。
・サウンドの実装。
・プレイ中の UI の実装
＜2 月下旬～3 月上旬＞
・スコアの実装、調整。
・制限時間を計測する処理を実装
・新しい敵の追加
＜3 月中旬～現在＞
・様々な意見を参考に改良
・ポインタからスマートポインタに変更
・プログラムの処理内容を意識した改良。
・敵の出現処理を改良

次のページに続きます。

２：プログラムを組む上で工夫したこと・注意したこと

・State パターンを活用し状態を柔軟に変えられるようにしました
・シングルトンを用いて唯一性を保持し、間違えて複数実装することが無くなりました。
・スマートポインタを用いて解放忘れをなくしました。
・条件の整理をし、早期リターンなどを活用し可読性を向上させました。
・プログラムを書いた意図を示すようにしました。
・for 文を回す際、余計なコピーが作成されないようにしました。
・ポリモーフィズムを意識するために、派生クラスで共通する処理は基底クラスで記述するようにしました。
・参照することが目的の変数などは極力 const 修飾子を付けて、変数の変更をしないことが明確に分かるように意識しました。
・関数ポインタを使ってプログラムを短くできそうな部分は関数ポインタを使うようにしました。こうすることで、関数の呼び出し側のコードがシンプルになり可読性も向上しました。

３：プログラムのアピールポイント

・シーン
（場所）”Src/Scene/SceneManager.cpp”
“Src/Scene/Scene.h”
各シーンは次に遷移するシーンの数字の情報だけを伝えられるようにしました。
ラムダ式などを組み合わせた各シーンを生成するための関数を配列に格納しておくことでシーンの生成を簡略化し、コードの可読性とメンテナンス性を向上させています。

・敵の管理と生成
（場所）”Src/Enemy/EnemyManager.cpp”
“Src/Enemy/EnemyParameter.h”
敵の種類が増えるごとに同じような敵を生成する処理を書かなければならなかったので、可読性の高さと拡張性を持たせたいと思い、全ての敵の種類を `enum class ENEMY_LIST` で定義し、それぞれ配列で一か所に纏めてあるモデルパス、敵の最大数、出現間隔、生成方法を追加するだけで新しい敵を実装できるようにしました。

次のページに続きます。

・くっつく処理

（場所）”Src/Substance/SubstanceBase.cpp”

99 行目

プレイヤーにくっつく丸い物体の処理を作るにあたって、くっついたとき、プレイヤーの回転に追従する処理について考える必要がありました。

まず、当たった瞬間にその物体から自分自身までのベクトルを作成し、その後は、そのベクトルを基準にプレイヤーの回転と同じ回転をさせ続けることで、全ての方向から当たっても追従できる処理を作りました。

また、「別の物体に当たってくっつく」という抽象的な処理を記述しているので、その処理だけのクラスを作成し、他のクラスと合成して使用もできます。

・自作の行列クラス、ベクトルクラスの活用

（場所）”Library/Vector3D.cpp”

”Library/Matrix3D.cpp”

”Src/Substance/SubstanceBase.cpp

ゲーム制作の際に行列計算とベクトル計算を行うことが頻繁にあったので、

自作のベクトルクラスを作成しました。そのクラスでは、演算子のオーバーロードや

その他の関数を実装することでベクトルクラスをより便利にすることが出来ました。

自作のベクトルクラスでは Unreal Engine のベクトルを参考にしています。

・カメラ　（場所）”Src/Camera/Camera/Camera.cpp”

カメラの距離が固定だとプレイヤーが移動しづらいという意見があったので、プレイヤーの動きに合わせて自動的にカメラの距離を近づけたり遠ざけたりしました。

・状態管理クラス

（場所）”Src/StateMachine/ StateMachine.h”

“Src/StateMachine/ StateBase.h”

テンプレートを組み合わせることで、汎用的な状態の基底クラスと、その状態を管理するクラスを作成しました。特にコードの美しさや直感的な操作性にこだわっており、

C++でよく見るような以下の形式で状態を追加できるように実装しました。

AddState<追加したい状態クラス>(コンストラクタの引数);

次のページにコード例を書いています。

(前のページの続き)

コード例：
stateMachine_ = std::make_unique<EnemyBase::StateMachineType>();

//状態の追加
stateMachine_->AddState<Idle>(ENEMY_STATE::IDLE);
stateMachine_->AddState<Chase>(ENEMY_STATE::CHASE);

//ゲーム開始時の状態の設定
stateMachine_->ChangeState(ENEMY_STATE::CHASE);

４：プログラムでの反省点と次回への目標

今回の制作では、プログラムの技術的な情報の勉強ばかりに時間を費やしてしまい、肝心のステージの数や敵の種類等のゲームのボリュームが少なくなってしまいました。
先生にも「1 週間に 1 回は見た目を変えたほうが楽しいしモチベーションも上がるよ」とアドバイスをいただき、実際に変えてみたところ、その言葉を痛感し
ゲームは遊ぶ人を楽しませるためのものが必要だと改めて認識しました。

設計方法が悪い部分が多く、一つのクラスに機能を持たせすぎてしまいました。
余裕を持ち、まとめられる部分はまとめて、プログラムの可読性を向上させたいです。
テストの段階でマジックナンバーを入力することが多く、放置されていたものもあるので、こまめにマジックナンバーを消し、外部から簡単に変更できるものにするように心がけたいです。

現在は、先生から教えて頂いたプログラムの本を読んだり、Unity 公式の資料を見たりして、改めてコーディングの基礎の部分や、設計を見直しています。
また、グループで卒業作品を制作中でもあり、自分にはない考えや感覚を積極的に取り入れるよう心掛けながら制作しています。