機人 ~KIJIN~

制　作　時　期　：2022/9 ～ 2023/1
（制作期間 ： 4ヵ月）
制　作　人　数　：5人
動　作　環　境　：Windows10,11
使　用　言　語　：C++/HLSL
使用ライブラリ：DXライブラリ

作品概要

ロボットが主人公の無双ゲームです。
チームメンバー全員が初の3Dゲームの制作でしたが爽快感のあるアクションを実現することができました。
主な担当箇所はプレイヤー制御、アニメーション制御、UI制御です。
特に力を入れた場所はアニメーションに合わせた移動をアニメーションフレームで計算したところです。

その他概要

担当箇所

・全般担当

Keyboardクラス

Padクラス

Controllerクラス

PlayerAnimIndex.h

PlayerBehaviorクラス（デザインパターンは委託）

PlayerAttackBehaviorクラス（デザインパターンは委託）

PlayerSkillBehaviorクラス（デザインパターンは委託）

・一部担当

Mathクラス（Clamp()とDirNearAroundRad()を担当）

Animatorクラス（デザインパターンは委託、アニメーションブレンドはチームメイトと共同で制作）

シェーダー全般（ほかのチームメイトが担当していたが、勉強のために制作に一部立ち会った）

UI全般（制作後にほかのチームメイトがデザインパターンを変更した）

参考資料

・Web

気持ちのいいジャンプを目指して

https://qiita.com/odanny/items/297f32a334c41410cc5d

DXライブラリ　関数リファレンスページ

https://dxlib.xsrv.jp/dxfunc.html

NormalMap-Online

http://cpetry.github.io/NormalMap-Online/

・書籍

HLSLシェーダーの魔導書

Game Programming Pattern

実例で学ぶゲーム3D数学

・授業

Unityを使用したステージ生成の方法

プレイヤー制御

プレイヤーはアニメーションやエフェクトなどのパラメータを多数持っており、今まではその要素を管理するために膨大なコード量を必要としていました。

そのため、あらかじめパラメータをSTLのpairやtupleでまとめてコンテナに格納しておくことで、要素管理を簡潔にしました。
それぞれのプロセス中の起動判定を変えるだけで実行の有無も変えることができるようにしたことで、パラメータごとの管理も容易にしました。

例)Effect

エフェクト追加

```
void PlayerBehavior::AddEffect(FactoryID id)
{
    // エフェクトの種類取得
    auto& effect = effect_.at(static_cast<int>(id));
    // 座標格納
    auto& pos = std::get<1>(effect);
    pos = transform_->Pos();
    // 発動フラグ
    auto& flag = std::get<2>(effect);
    flag = true;
}
```

true判定されたIDのみ実行

```
void PlayerBehavior::Effect(ObjectManager& objectManager)
{
    // エフェクト関係
    for (auto& e : effect_)
    {
        auto& flag = std::get<2>(e);
        if (flag)
        {
            auto& factory = std::get<0>(e);
            auto& pos = std::get<1>(e);
            auto id = objectManager.CreateFromFactory(factory, ObjectID{}, pos);
            objectManager.Begin(id);
            flag = false;
        }
    }
}
```

アニメーション

モーションでは前に進んでいても、ボーンが固定されているためその場に留まってしまっているように見え、アニメーションとしては違和感がありました。

そのため、基準となるボーンのフレームごとの移動量を取得し座標に加算することで違和感のないアニメーションにしました。
また、アニメーションによって基準のボーンが違ったため、アニメーション変更時にあらかじめ基準のボーンを変更して対応しました。

```
if (playTime_ >= totalTime_)
{
    MV1SetAttachAnimTime(handle_, attachIdx_, totalTime_);
    nowPos_ = MV1GetAttachAnimFrameLocalPosition(handle_, attachIdx_, moveAnimFrameIndex_);
    pos_ = nowPos_ - prePos_;

    // 総再生時間まで再生したとき
    if (isLoop_)
    {
        // ループするとき再生時間を0にする
        playTime_ = 0.0f;
        MV1SetAttachAnimTime(handle_, attachIdx_, 0.0f);
        prePos_ = MV1GetAttachAnimFrameLocalPosition(handle_, attachIdx_, moveAnimFrameIndex_);
        MV1SetAttachAnimTime(handle_, attachIdx_, playTime_);
        nowPos_ = MV1GetAttachAnimFrameLocalPosition(handle_, attachIdx_, moveAnimFrameIndex_);
        pos_ = nowPos_ - prePos_;
    }
    else
    {
        // ループ再生しないときは総再生時間を入れとく
        playTime_ = totalTime_;
    }
    playTimeOver_ += delta * 60.0f;
}
else
{
    MV1SetAttachAnimTime(handle_, attachIdx_, playTime_);
    nowPos_ = MV1GetAttachAnimFrameLocalPosition(handle_, attachIdx_, moveAnimFrameIndex_);
    pos_ += nowPos_ - prePos_;
}
```

アニメーション中は現在座標からアニメーション初期座標を減算した座標を加算する（移動量）

ロボット特有の挙動

ロボットは人型のキャラクターと違い、ブーストダッシュの爽快感や待機状態の動作など、細かな部分にも気を付けないと違和感が生まれます。
例えば、このゲームでは回避アクションをダッシュに含めたため瞬間的なブーストで推進力が大きくなるようにしました。
単純に速度の数値を上げるだけでなく、ブーストエフェクトやカメラワーク、加速度などを変化させることで自然で違和感のない挙動にすることができました。
その他にも、ホバリングによる微妙な体の揺れや、必殺技発動時の反動で下がる動作などに注意して制作しました。

通常移動

ダッシュ