プログラマ志望
小貫　誠人

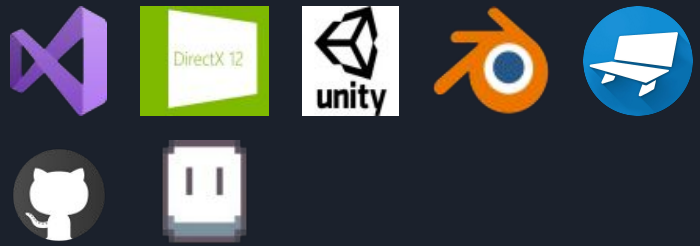
ポートフォリオ

自己紹介

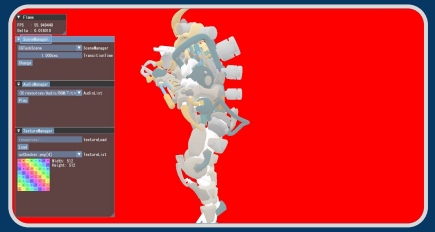
名前：小貫 誠人(オヌキ セイヤ)

使用言語：C++20, C#, Python

使用ツール



自作ゲームエンジン
SolEngine (DirectX12)



assimpライブラリを用いた、
マルチメッシュモデル読み込み

自作ECSによる
ゲームオブジェクトの処理

Unityのようなコンポーネント指向と、
ECSによるデータ指向 での開発が可能です。

コンポーネント単位で処理を区切ることで、
作業の分担などを行うことができ、開発速度の向上
に繋がりました。

C++の基礎力と実装速度に自信があります

- 関数実装は「短く、少ない処理で、単純に」が、
  ベストプラクティスであると認識しています。
- 「一つの処理を必ず行う関数」であるほど、
  強い信頼性と、単純さ故の再利用性が期待できます。
- **単純でわかりやすい が故の、**
  無駄な時間を省いた素早い実装 ができます。
- 短くわかりやすい関数のため、C++準拠のイテレータ仕様やメモリ管理、SIMDなどの知識を習得しました。

就活作品

ESC Survivor

ジャンル：2Dアクション(3D表現)
制作期間：8ヶ月(2024年 2月 ~ 11月)
制作人数：1人 (※1)
開発環境：DirectX12 (C++20)

外部ライブラリ：DirectXTex
nlohmann-json
Assimp
Dear ImGui

概要：

Soulstone Survivorを元にした、ローグライトゲームです
。

URL：https://github.com/Se-Onuki/DirectX12.git

※1 五月に個人パソコンが故障したため、学校のパソコンからのコミットがあります。

ECSの使い方

Componentとして利用する構造体の定義

```
struct ModelComp : IComponent {
    // モデルへのリソースハンドル
    SolEngine::ResourceHandle<SolEngine::ModelData> model_;
};
```

```
struct ModelAnimator : IComponent {
    // アニメーションのリソースハンドル
    std::array<SolEngine::ResourceHandle<SolEngine::Animation>, 4u> animateList_;
    // アニメーションの状態管理クラス
    SolEngine::AnimationPlayer animatior_;
};
```

Systemとして呼び出す処理の定義

必要なのは
・取得する型の情報
・型に対して行う操作が書かれた関数

```
class ModelAnimatorUpdate :public IJobEntity {
public:
    // 取得したいデータを指定する
    ReadAndWrite<ECS::ModelComp, ECS::ModelAnimator> readWrite_;
    // プログラムが型を解釈するためのヘルパ
    using DataBase = DataBase<decltype(readWrite_)>;

    // 自動的に呼び出される処理
    void Execute(const World *const, const float deltaTime) {
        // 保存されているデータを展開する
        const auto &[ ECS::ModelComp & model, ECS::ModelAnimator & animator] = readWrite_;
        // 取得したデータを用いて処理を行う
        if (animator.animatior_.GetDeltaTimer().IsFinish()) {
            animator.animatior_.SetAnimation( animation: animator.animateList_[0]);
            animator.animatior_.Start( isLoop: true);
        }
        animator.animatior_.Update(deltaTime);
    }
};
```

ECSを用いた大量のオブジェクトの処理

関数が変数を保持しないため､INとOUTのみを意識したシンプルな実装が可能です。

```
class ModelAnimatorUpdate :public IJobEntity {
public:
    ReadAndWrite<ECS::ModelComp, ECS::ModelAnimator> readWrite_;
    using DataBase = DataBase<decltype(readWrite_)>;

    void Execute(const World *const, const float deltaTime)
    {
        const auto &[ ECS::ModelComp & model, ECS::ModelAnimator & animator] = readWrite_;
        if (animator.animatior_.GetDeltaTimer().IsFinish()) {
            animator.animatior_.SetAnimation( animation: animator.animateList_[0]);
            animator.animatior_.Start( isLoop: true);
        }
        animator.animatior_.Update(deltaTime);
    }

};
```

Templateを使って、
取得するコンポーネントを指定する

指定したコンポーネントへの参照

主な過去作

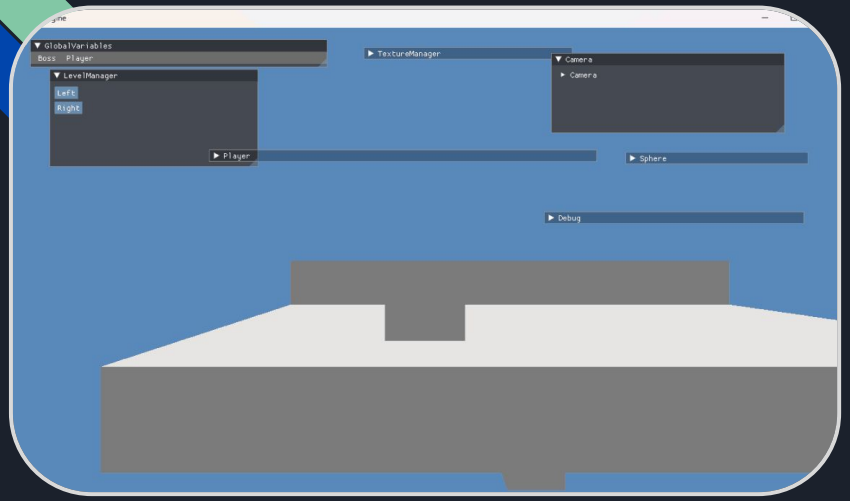
コスモスピナー

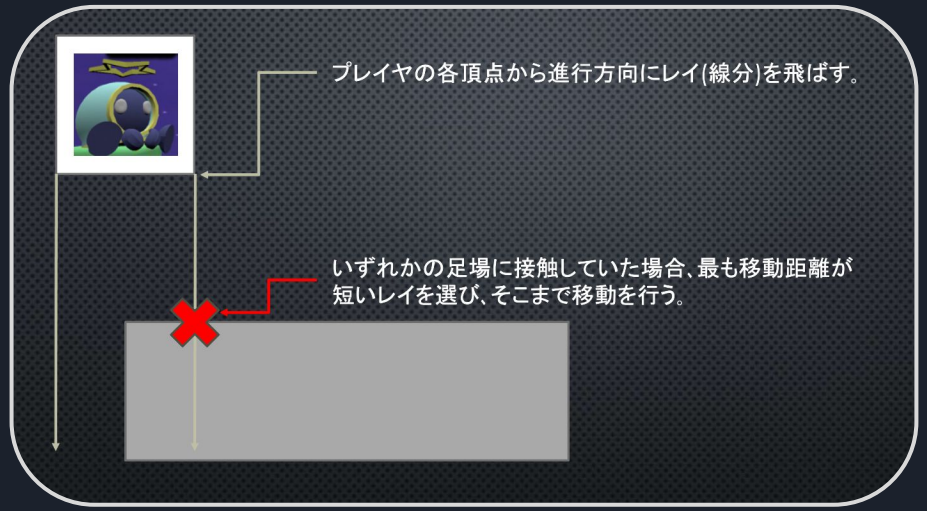
制作期間： 1ヶ月
開発環境： 自作エンジン
開発人数： 3人 (PL 1 / PG 2)
担当箇所： メインPG/インゲーム全般/
エンジン/レベルエディタ

回転する立体的な足場を用いた3Dパズルゲームです。

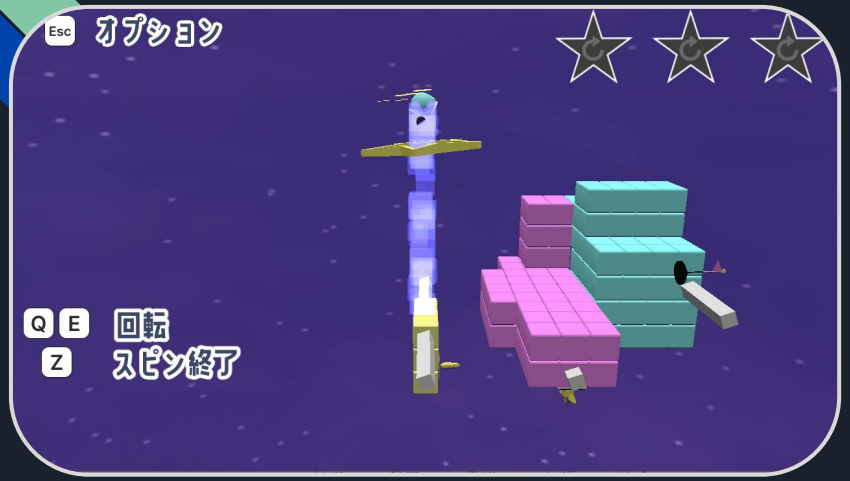
3D描画だけではなく、挙動の面でも3Dにすることを
挑戦しました。

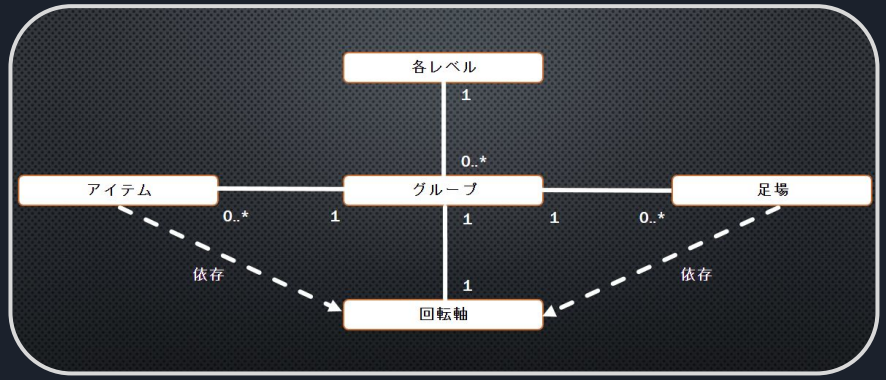
当時、完全に3D空間で歩き回れるゲームを作ったことがなかったため、
当たり判定などの挙動から開発を行いました。



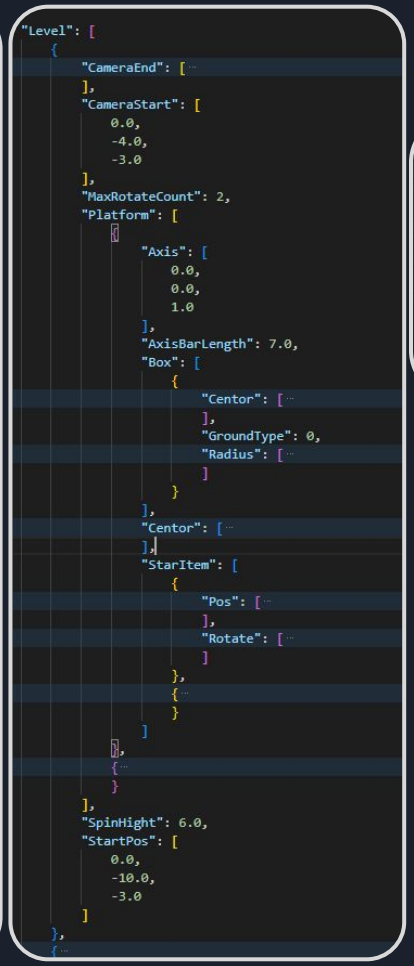


立体的な挙動を実装するために、地形への当たり判定を開発しました。
地形へ線分(Ray)を飛ばし、接触した時の係数(t)を算出して
それを元に移動量(Velocity)を調整するように実装しました。

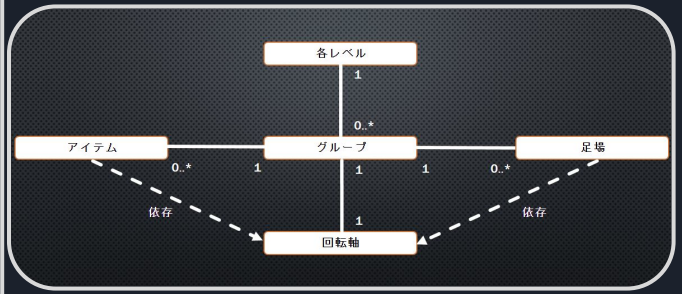




ステージの形状が回転により変わるゲームなので、
レベル内のオブジェクト同士の関係性を強く意識して実装しました。

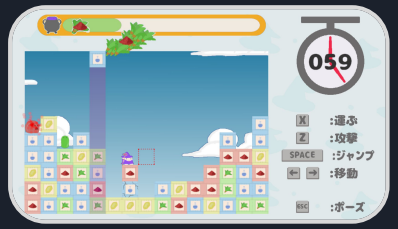


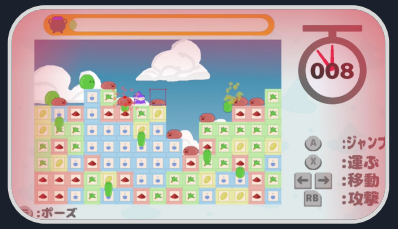
```
"Level": [
    {
        "CameraEnd": [
        ],
        "CameraStart": [
            0.0,
            -4.0,
            -3.0
        ],
        "MaxRotateCount": 2,
        "Platform": [
            {
                "Axis": [
                    0.0,
                    0.0,
                    1.0
                ],
                "AxisBarLength": 7.0,
                "Box": [
                    {
                        "Centor": [
                        ],
                        "GroundType": 0,
                        "Radius": [
                        ]
                    }
                ],
                "Centor": [
                ],
                "StarItem": [
                    {
                        "Pos": [
                        ],
                        "Rotate": [
                        ]
                    },
                    {
                    }
                ]
            },
            {
            }
        ],
        "SpinHight": 6.0,
        "StartPos": [
            0.0,
            -10.0,
            -3.0
        ]
    },
    {
```

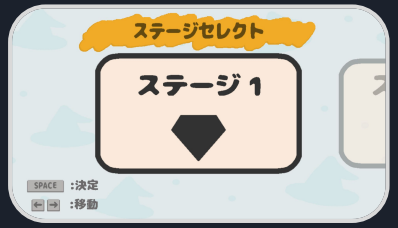


制作したレベルエディタと、
それを用いて開発したレベルデータの
Jsonファイルです。

ゲーム内で作成したクラスの関係を上記の図の様に単純化して、
それをJsonで書き起こしたものを
実際にゲーム内外で
読み書きをできるようにローダーを作成しました。





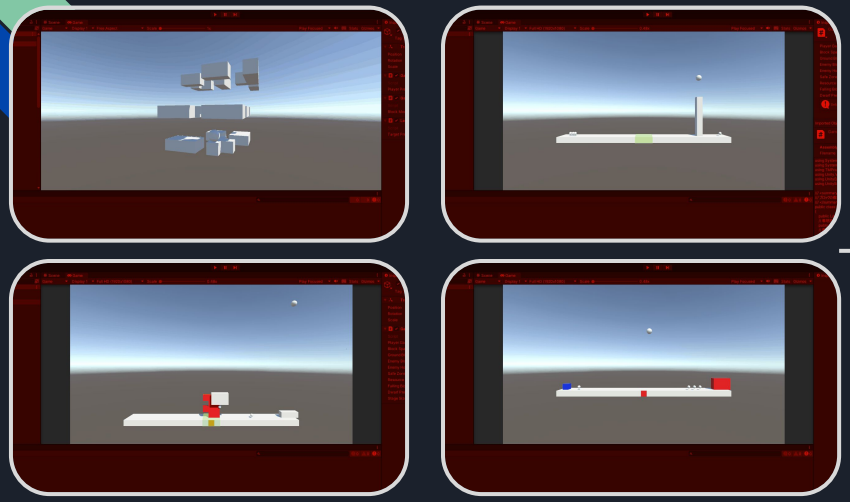


チェイン魔女娘

制作期間： 3ヶ月半
開発環境： メンバーのエンジン
開発人数： 4人 (PL 1 / PG 3)
担当箇所： メインPG/インゲーム全般/
エディタ/プロトタイプ開発

ブロックを動かし、繋げたブロックをまとめて壊す
パズルゲームです。

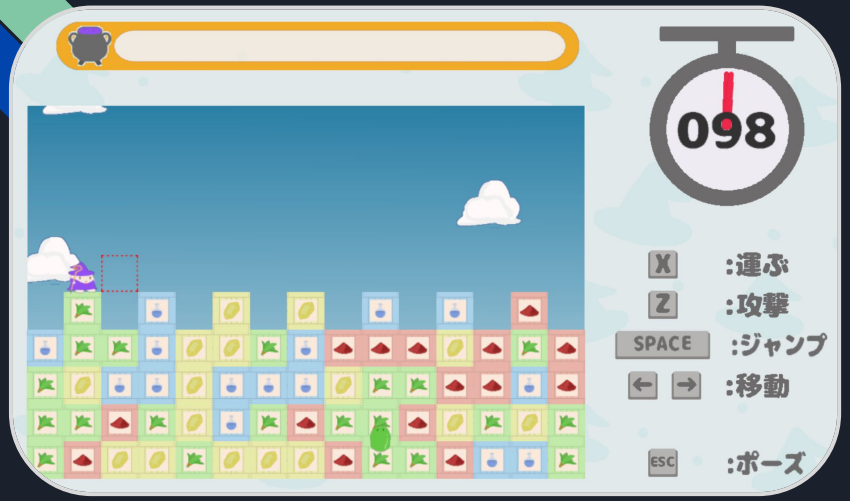
ステージ内にブロックがどのように配置され、どのように連結
しているかをDFS(深さ優先探索)で調べたり、
空中に浮かぶブロックなどのオブジェクトの落下処理と当たり
判定などを実装しました。

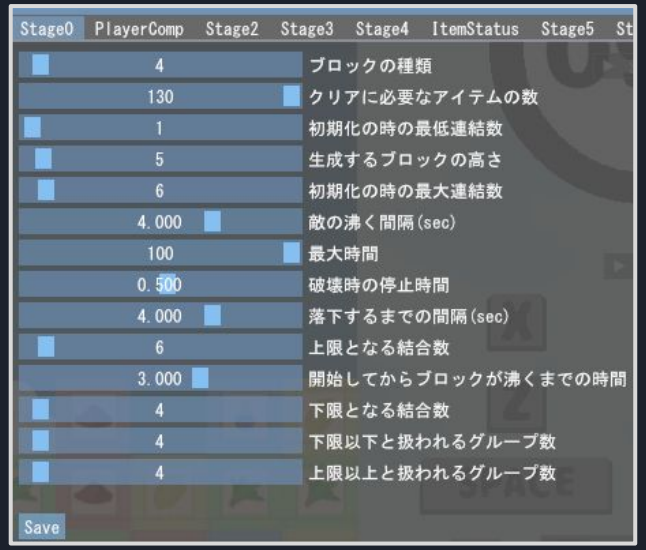


この作品では、ゲーム自体の面白さを確認してから肉付けを行うということを意識しました。

開発効率の点から一度Unityを利用して、プロトタイプをプランナーと相談しながら制作しました。
メンバーが開発していたエンジンに移植する際のことを考え、
描画以外の処理を自作し、コンポーネントの処理をそのまま移植できるように工夫しました。

隣のブロックが同じ色なのかを再帰関数で調べ、
その箱が連続しているかを判断してブロックの破壊を行えるようにしました。





ゲーム内のパラメータの調整項目の実装では、
プランナーからの要望に応えて各種オブジェクトを調整できるようにしました。

中でも「ゲーム開始時の箱の接続数」の項目では、破壊時と同じ「連結数の検索関数」を流用するなど、1関数1機能を心がけた事による恩恵を感じました。