ポートフォリオ

名前　　米満（よねみつ）　悠人（ゆうと）

学校　　福岡情報ITクリエイター専門学校
ゲーム・クリエイター学科
ゲームクリエイターコース

メール　　fko2247015@stu.o-hara.ac.jp

GitHub　　https://github.com/YYY0427

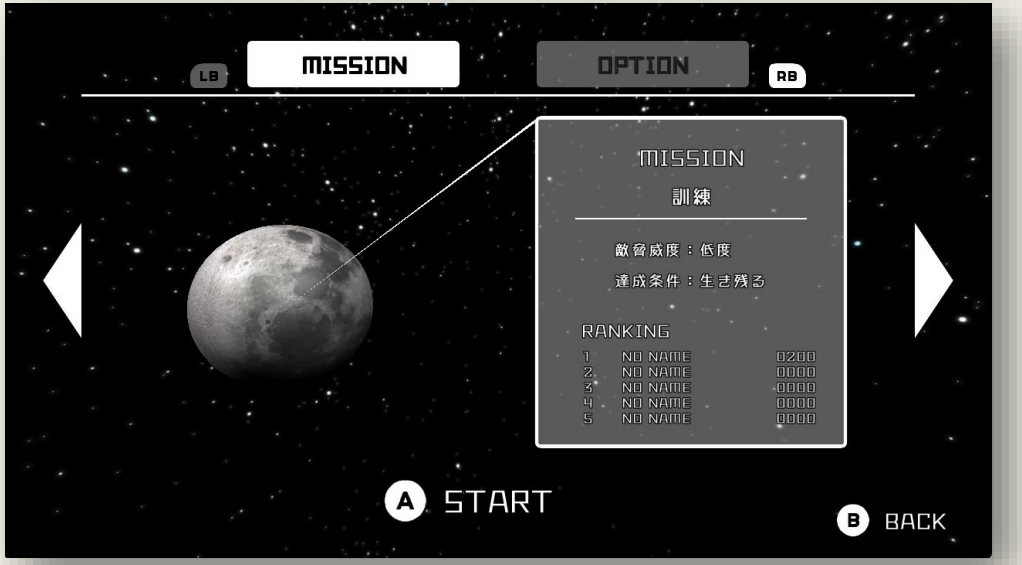
趣味　　旅行、めめちゃんのお世話

スキル

C、C++	２年
C#、Unity	１年
GitHub	１年半

最終制作作品

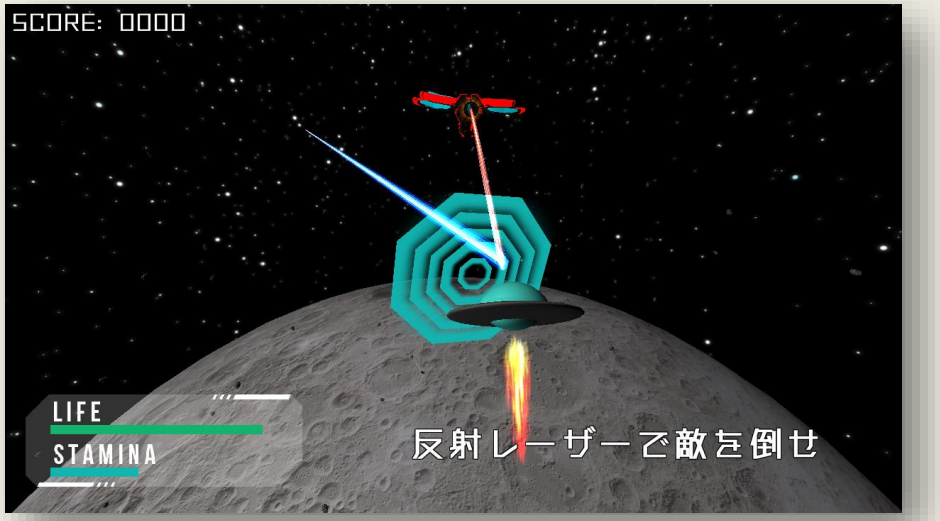
作品名	SPACE REFLECT
ジャンル	反射3Dアクション
制作期間	2023/12/17～2024/3/20
開発環境	C++、DXライブラリ、Unity
対応機種	Windows
制作人数	プログラマー１人
GitURL	https://github.com/YYY0427/REFLECT.git

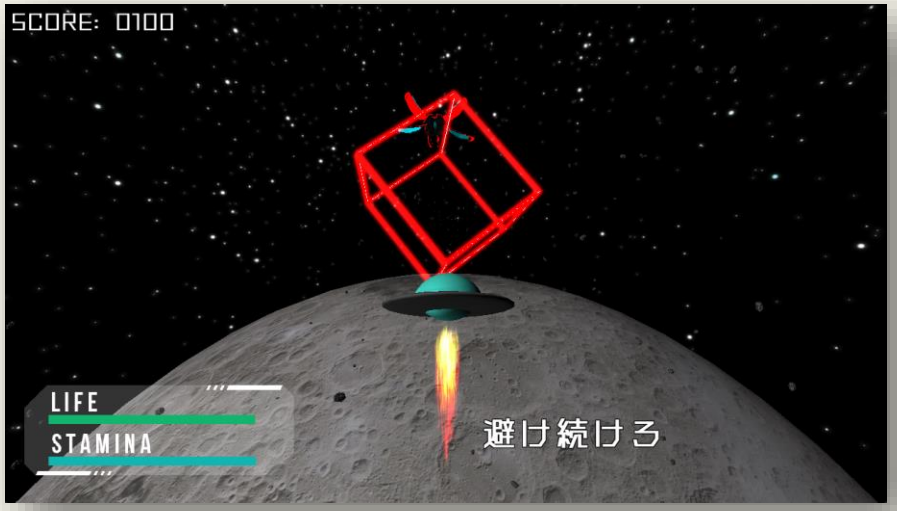


企画・概要

自分からは攻撃ができない

敵のレーザーを反射して敵を倒そう





キューブ型のレーザーは反射できない

避け続けよう！！

エイムアシスト

レーザーを反射するとき、純粋な反射だけだと

敵に反射したレーザーを当てるのが難しすぎる為、エイムアシストを導入

エイムアシスト仕様

①反射したレーザーと１番近い敵を検索

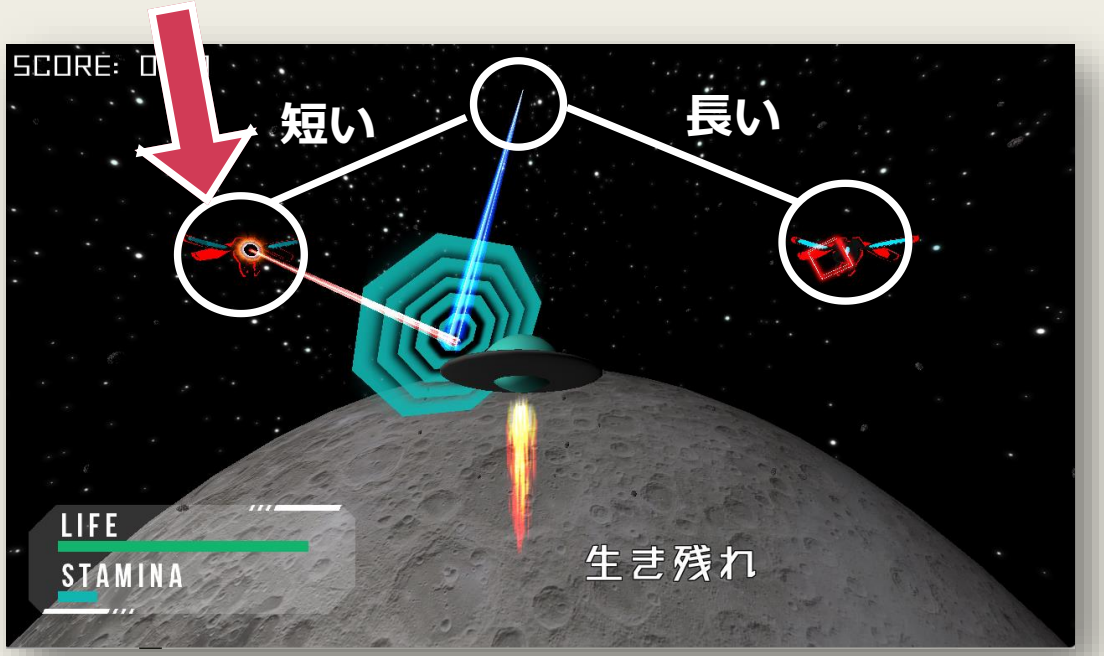
②検索した敵と反射したレーザーの距離が一定範囲以内なら
　エイムアシストを有効にする

③エイムアシストの強さは0.0～1.0
(1.0の場合は敵を自動追従、ゲーム内では0.45)

```
// Z軸を無効にする
Vector3 directionPos = m_directionPos;
directionPos.z = 0;
enemyPos.z     = 0;

// 敵とレーザーの向いている方向との距離が一定範囲内なら
Vector3 aimAssistVec[];
if (enemyPos.Distance(directionPos) < aim_assist_range)
{
    // エイムアシストを有効にする
    aimAssistVec = (enemyPos - directionPos).Normalized();
}

// 反射ベクトルを作成
Vector3 goalPos = Vector3::Reflect(
    m_pLaser->GetDirection(), Vector3::FromDxLibVector3(m_pShield->GetVertex().front().norm));
Vector3 reflectVec = (goalPos - m_directionPos).Normalized();

// ベクトルの取得
float reflectVecPower = 1.0f - aim_assist_power;
Vector3 moveVec = ((reflectVec * reflectVecPower) + (aimAssistVec * aim_assist_power)) * move_speed;
```

（Game/Laser/ReflectLaser.cppを参照）

**反射したレーザーと１番近い為
この敵にエイムアシストがつく**



外部ファイル化

ステージデータ、ウェーブデータ、敵の行動データを外部ファイル化

外部ファイル化することによって、レベルデザインの調整を効率化！

ステージデータ（１つのステージにウェーブが複数ある場合、次の行にウェーブデータのファイル名を書く）

	A	B	C	D
1	敵のデータファイル			ボス敵の種類
2	TutorialWave1	TutorialWave2	TutorialWave3	2

ウェーブデータ（１つのウェーブに複数体の敵が出てくる場合、次の列にウェーブデータを書く）

	A	B	C	D	E	F	G	H	I	J	K
1	スクリーン座標x	スクリーン座標y	プレイヤーを0としたZ座標	敵の種類	HP	攻撃力	拡大率	移動速度	行動データのファイルパス		
2	640	0	1200	0	100	10	1.25	10	TutorialAction1		

敵の行動データ（行動が複数ある場合、次の列に敵の行動データを書く）

	A	B	C	D	E	F	G	H	I	J	K
1	目的地x	目的地y	目的地 z	目的地に到達してから 次の目的地に向かうまでの待機フレーム	目的地に到達したら レーザーを撃つか	レーザーを撃つ場合 どのレーザーを撃つか	チャージフレーム	レーザーを撃つ場合 目的地に到達してから レーザーを撃つまでの待機フレーム	レーザーを撃つ場合 レーザーの移動速度	レーザーを 何フレーム発射し続けるか	レーザーを撃つ場合 レーザーがプレイヤーを追従するかどうか
2	640	200	1200	80	1	2	0	0	8	300	1
3	640	-100	1200	0	0						

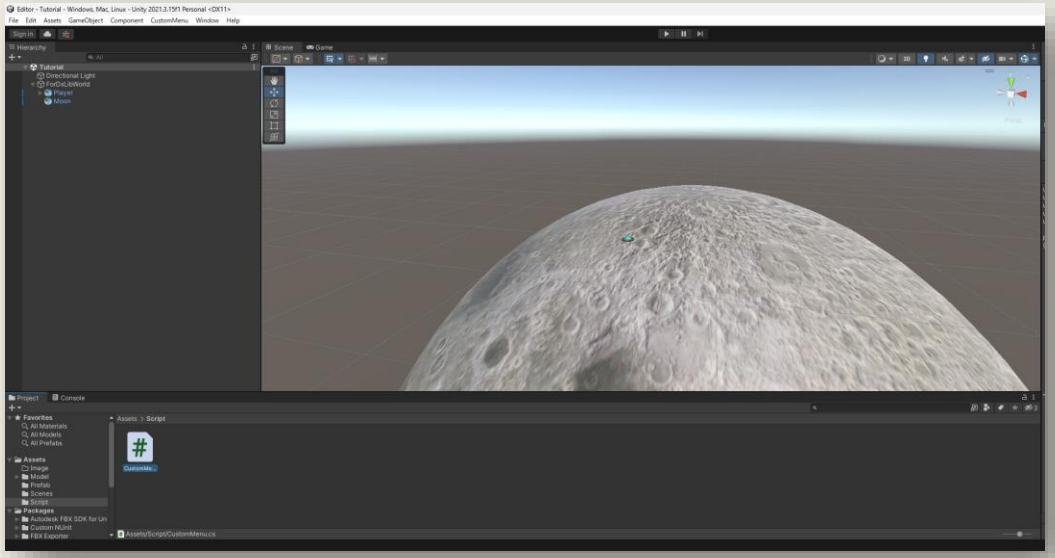
ステージの配置

Unityを外部エディターとして使用

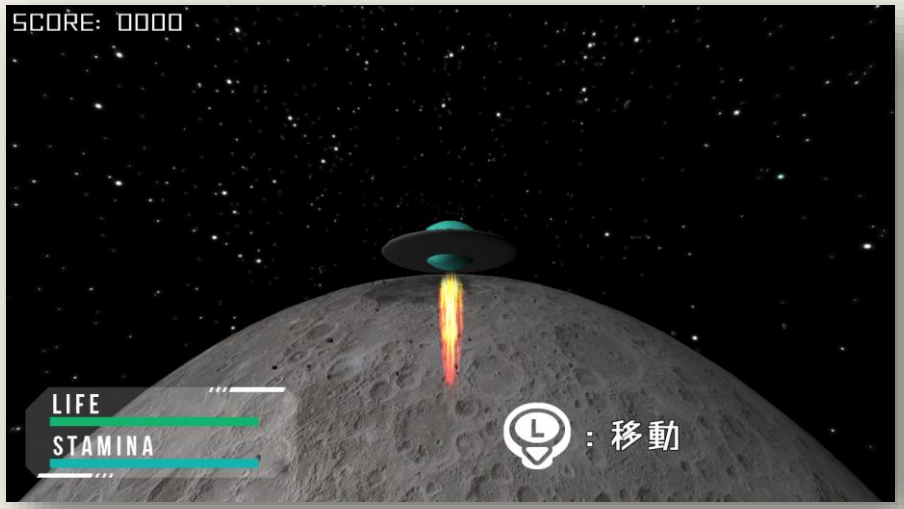
座標、回転、拡大のそれぞれのデータをバイナリファイルでUnityで出力し、

作成したバイナリファイルを読み込み、ステージの作成を高速化

Unityの画面



ゲーム画面



カメラワーク

ゲームスタート時、ゲームクリア時に

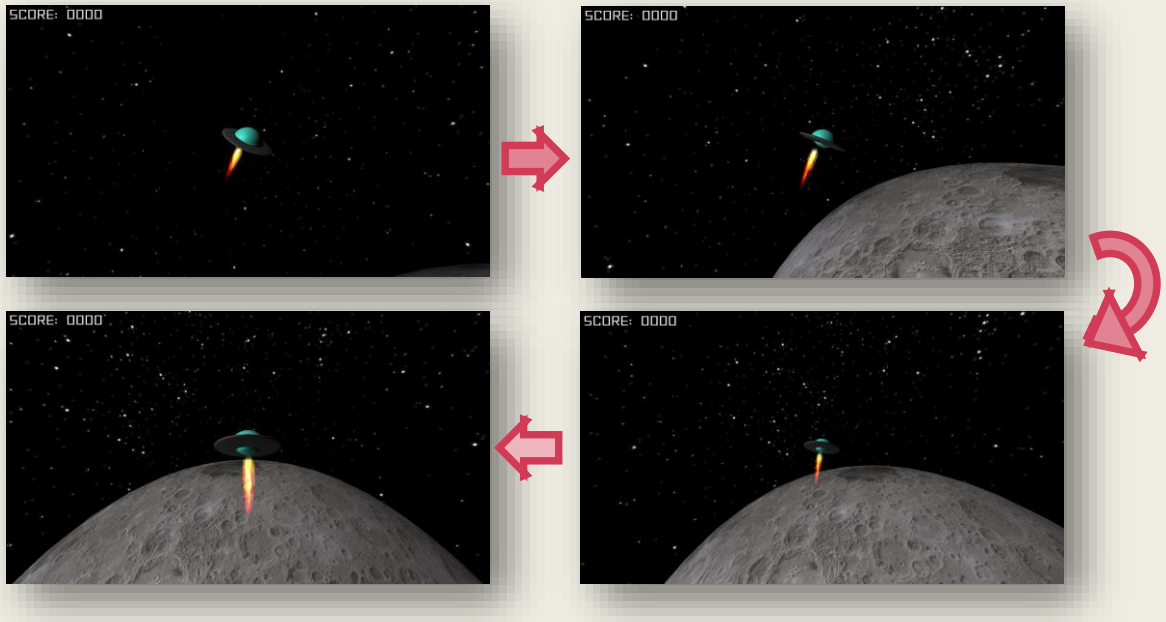
カメラがプレイヤーの周りを回る演出を

エルミート曲線を使って実装

(エルミート曲線を使うことで、滑らかなカメラの挙動を表現)

```cpp
// エルミート補間
Vector3 Vector3::Hermite(const Vector3& startPos,
                         const Vector3& startTangent,
                         const Vector3& endPos,
                         const Vector3& endTangent,
                         float value)
{
    float t2_3 = value * value * value;
    float t3_2 = value * value;
    float h1 = 2.0f * t2_3 - 3.0f * t3_2 + 1.0f;
    float h2 = -2.0f * t2_3 + 3.0f * t3_2;
    float h3 = t2_3 - 2.0f * t3_2 + value;
    float h4 = t2_3 - t3_2;

    return startPos * h1 + endPos * h2 + startTangent * h3 + endTangent * h4;
}
```

(Math/Vector3.cppを参照)




```cpp
// プレイヤーの位置がカメラの位置より前に来たら
if (playerPos.z > m_pos.z + camera_start_animation_start_pos_z)
{
    // エルミート曲線の値を増やす
    m_hermiteFrame     = (std::min)(++m_hermiteFrame, camera_start_animation_frame);
    float hermiteValue = static_cast<float>(m_hermiteFrame) /
                         static_cast<float>(camera_start_animation_frame);

    // カメラの位置をエルミート曲線で移動させる
    m_pos = Vector3::Hermite
        (
            m_pos,
            camera_start_animation_start_direction,
            playerPos + camera_start_animation_end_pos,
            camera_start_animation_end_direction,
            hermiteValue
        );

    // エルミート曲線の値が1.0fを超えたら
    if (m_hermiteFrame >= camera_start_animation_frame)
    {
        m_isStartAnimation = true;
        m_hermiteFrame = 0;
    }
}
```

(Game/Camera.cppを参照)

デザインパターン

Stateパターンでの敵ボスの行動管理
(簡単に敵ボスの行動を追加できる)

```
private:
    // ステート
    enum class State
    {
        // 基本
        ENTRY,      // 登場
        IDLE,       // 待機
        BARIER,     // バリア
        DIE,        // 死亡
        GAME_OVER,  // ゲームオーバー

        // 攻撃
        MOVE_HOMING_LASER_ATTACK,   // 移動しながらホーミングレーザー攻撃
        CUBE_LASER_ATTACK,          // キューブレーザー攻撃
    };
```

Factory Methodパターンでのレーザーの管理
(インスタンスの生成に必要な処理を隠蔽化し、可読性の向上)

```
// レーザーの追加
int LaserManager::AddLaser(LaserType type,
                           const std::shared_ptr<EnemyBase>& pEnemy,
                           const int laserChargeFrame,
                           const int laserFireFrame,
                           const float laserSpeed,
                           const bool isPlayerFollowing)
{
    // レーザーのデータを作成
    LaserData laserData;

    // レーザーの種類を設定
    laserData.type = LaserType::NORMAL;

    // レーザーのインスタンスを作成
    laserData.pLaser = std::make_shared<NormalLaser>(
        pEnemy, m_pPlayer, laserChargeFrame, laserFireFrame, laserSpeed, isPlayerFollowing);

    // Keyの設定
    laserData.key = CreateLaserKey();

    // レーザーリストに追加
    m_pLaserList.push_back(laserData);

    // Keyを返す
    return laserData.key;
}
```

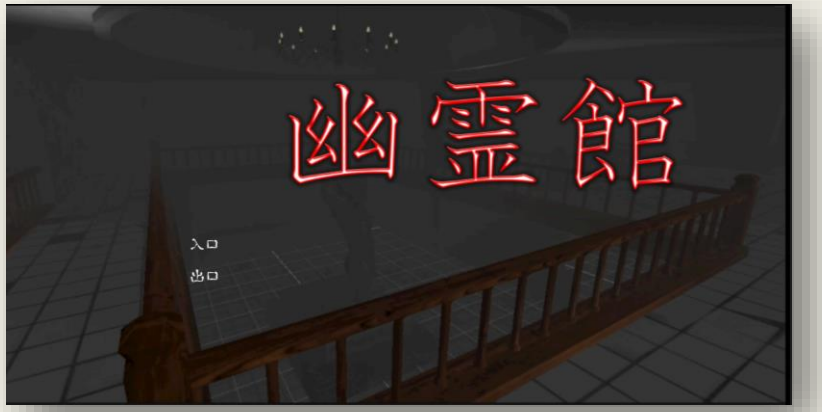
Singletonパターンでのサウンドの管理
(1つのクラスでサウンドの機能を集中管理)

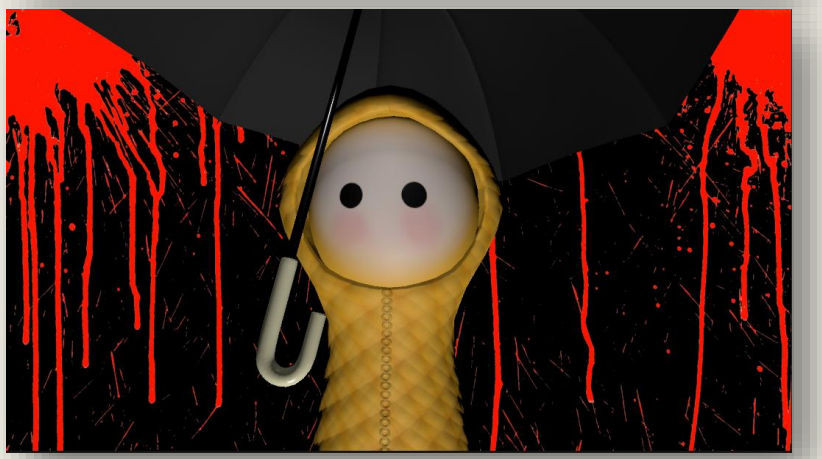
```
private:
    /// <summary>
    /// コンストラクタ
    /// シングルトンパターンなのでprivate</summary>
    SoundManager();

    /// <summary>
    /// コピーコンストラクタ禁止
    /// </summary>
    /// <param name="soundManager">コピー元</param>
    SoundManager(const SoundManager&) = delete;

    /// <summary>
    /// 代入演算子禁止
    /// </summary>
    /// <param name="soundManager">代入元</param>
    void operator = (const SoundManager&) = delete;
```

```
// 唯一のインスタンスを返す
SoundManager& SoundManager::GetInstance()
{
    static SoundManager instance;
    return instance;
}
```

制作実績

作品名	幽霊館 (Dark Deceptionをモチーフ)
ジャンル	3Dアクションゲーム
開発環境	Unity、C#
制作期間	2023/7/31～ 2023/9/13
制作人数	プログラマー4人
GitURL	https://github.com/YYY0427/DarkDeception.git
担当箇所	敵全般、ロード画面
学習内容	・Unityの基礎知識 ・チーム制作経験 ・チーム制作でのGitHubの利用 ・NavMeshの知識





制作実績

作品名	ボディーガード
ジャンル	3Dアクションゲーム
開発環境	DXライブラリ、C++
制作期間	2023/6/15～2023/9/15
制作人数	プログラマー１人
GitURL	https://github.com/YYY0427/FPS.git
学習内容	・3Dの当たり判定 ・Effekseerの利用 ・FPS視点

制作実績

作品名　これくとコイン
(パックマンをモチーフ)

ジャンル　ドットイートゲーム

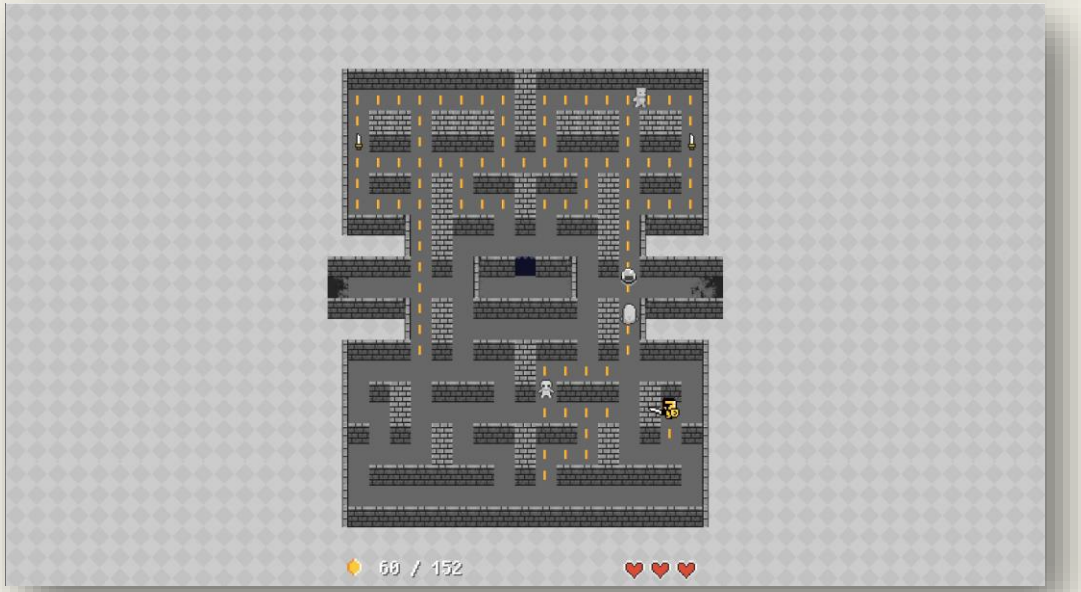
開発環境　DXライブラリ、C++

制作期間　2023/1/27～ 2023/3/25

制作人数　プログラマー１人

GitURL　https://github.com/YYY0427/CollectCoin.git

学習内容
- A*アルゴリズム
- 2Dの当たり判定
- 継承
- 外部ツール「Platinum」を使用してマップを作成



今後の目標

今後はまだ触れたことのないUnreal Engineの勉強や
Unityをさらに深いところまで触っていきます！