PORTFOLIO

ASOポップカルチャー専門学校
ゲーム・CG・アニメ専攻　ゲーム専攻

永冨 心

自主作品

目次

機人

k i z i n

作品概要

３Dアクションゲームで、敵を殲滅したら勝利というゲームです。操作を簡単にしてだれでも楽しめるようにチームで制作していきました。

担当箇所

主にグラフィック周りとカメラ制御を担当しました。
グラフィック周りの処理では、ポストエフェクトや影表現をしていきました。
モデルに対して、ディレクションライトを使って、影をつけたりモデルの影を落としたりしました。
ポストエフェクトについては、被写界深度の実装をしています。

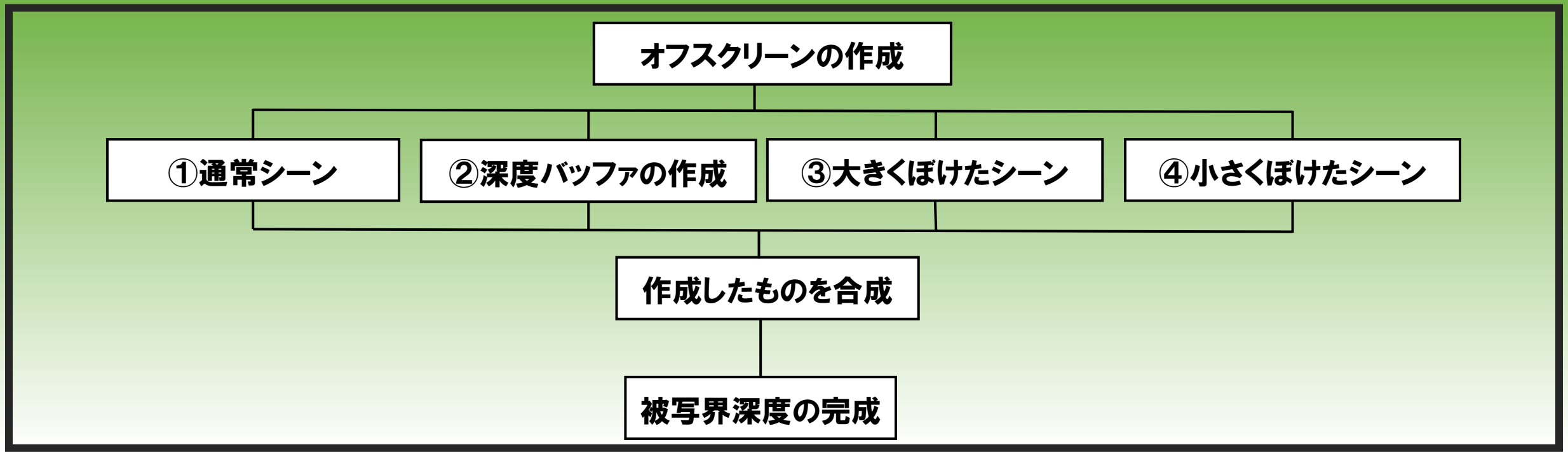
製作時期	３年後期
製作期間	４か月
チーム人数	５人
使用言語	C++/C#/HLSL
使用ライブラリ	DXライブラリ

ポストエフェクト

被写界深度

被写界深度を実装するにあたって、以下の工程で実装しました。
大きくぼけたシーンと小さくぼけたシーンを準備する理由は合成したときにより自然に見えるようにするためです。

①通常シーンはぼかしていないスクリーンを作成します。
②深度バッファは、作成したオフスクリーンの深度値を求めます。
③のぼけたシーンはオフスクリーンに強めのブラーをかけます。
④のぼけたシーンは③と同じくブラーをかけますが、弱めのブラーにします。



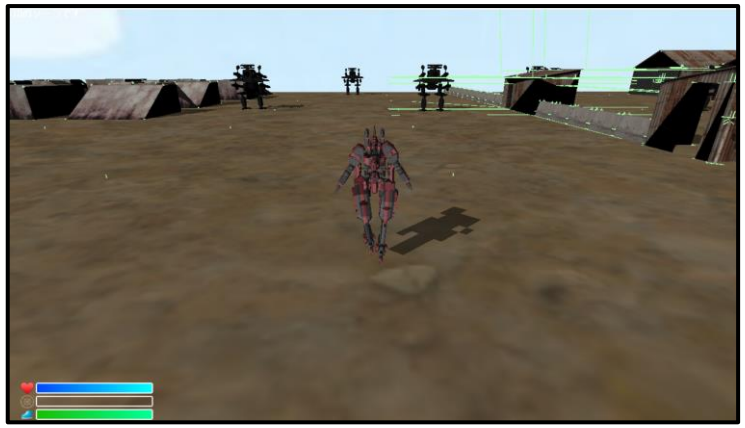
ポストエフェクト

深度バッファの作成

被写界深度を実装する際にカメラから見た深度バッファが必要となるのでカメラ空間でのZ値を描画した画像を用意します。このプログラムは範囲内か範囲外かをもとめて作成しました。

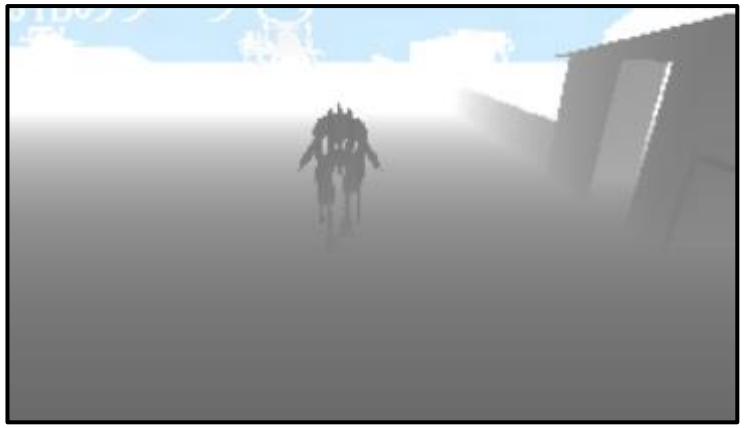
```
PSOutput output;
// 被写界深度の範囲内を0.0f～1.0fに変換
if (input.pos.z < DepthStart)
{
    output.color.x = 0.0f;
    output.color.y = 0.0f;
    output.color.z = 0.0f;
}
else if (input.pos.z > DepthEnd)
{
    output.color.x = 1.0f;
    output.color.y = 1.0f;
    output.color.z = 1.0f;
}
else
{
    output.color.x = (input.pos.z - DepthStart) * DepthScope;
    output.color.y = (input.pos.z - DepthStart) * DepthScope;
    output.color.z = (input.pos.z - DepthStart) * DepthScope;
}
output.color.w = 1.0f;
return output;
```

元の画像



深度バッファに変換

深度バッファ



ポストエフェクト

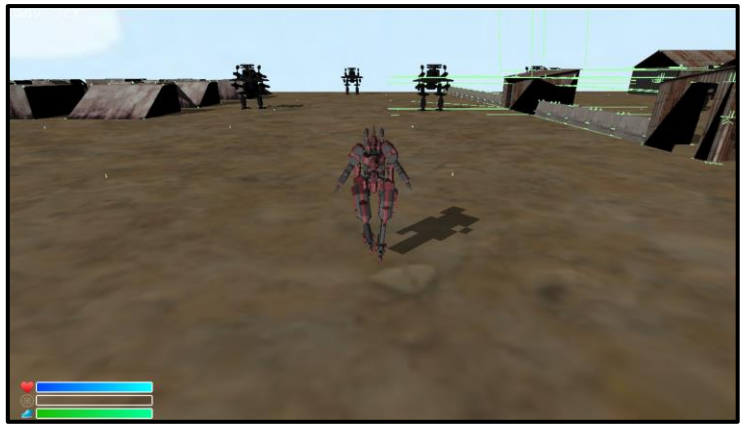
被写界深度の実装

先ほど取得した深度バッファと元のスクリーンバッファ、強めのブラーと弱めのブラーを合成して被写界深度を実装しました。
DepthStartとDepthEndの間はブラーをかけないようにし、ブラーをかけるところでもfadeを使い0～1の中で0.5までは弱いブラーをかけて、それ以上は強いブラーをかけるようにして実装していきました。

```
// 深度値を取得する
float d = depth.Sample(sam,input.uv);

// フォーカス情報からぼやけ率を算出
if ( d < DepthStart)
{
    fade = 1.0f - d / DepthStart;
}
else if (d < DepthEnd)
{
    fade = 0.0f;
}
else
{
    fade = (d - DepthEnd) / (1.0f - DepthEnd);
}
// ぼやけ率から色を算出
if (fade < 0.5f)
{
    color1 = tex.Sample(sam, input.uv);
    color2 = boke1.Sample(sam, input.uv);
    blend = fade / 0.5f;
}
else
{
    color1 = boke1.Sample(sam, input.uv);
    color2 = boke2.Sample(sam, input.uv);
    blend = (fade - 0.5f) / 0.5f;
}
output.color = lerp(color1,color2,blend);
return output;
```

元の画像



被写界深度の実装

被写界深度の実装

ポストエフェクト

FOGの実装

被写界深度のみだと空気感が伝わりにくかったので、フォグを一緒に出すことにより空気感が出るようにしました。

深度値を参考に徐々に空と同じになるようにしていき、被写界深度のようにぼやけるようにもしていきました。

ぼやけ率から表示する色を決定

```
float4 depthOfColor(float fade, float4 color1, float4 color2)
{
    float4 color1_, color2_;
    float blend;
    // 取得した範囲から色をとる
    if (fade < 0.5f)
    {
        color1_ = color1;
        color2_ = float4(color2.xyz, fade);
        blend = fade / 0.5f;
    }
    else
    {
        blend = (fade - 0.5f) / 0.5f;
        // この2つの処理が空の色にする
        color1_ = lerp(color1, float4(color2.xyz, 0.5f), 1);
        color2_ = float4(color2.xyz, fade);
    }

    return lerp(color1_, color2_, blend);
}
```

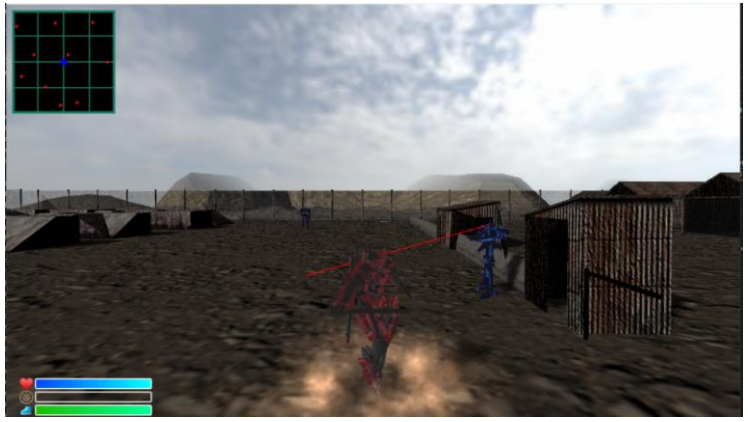
徐々に空の色に近づくようにした

```
// メインのスクリーンと縮小バッファとのラープ
float4 lerpColor(float fade, float4 color1, float4 color2)
{
    float4 color1_, color2_;
    float blend;
    if (fade < 0.5f)
    {
        color1_ = color1;
        color2_ = color1;
        blend = fade / 0.5;
    }
    else
    {
        color1_ = color1;
        color2_ = color2;
        blend = (fade - 0.5f) / 0.5f;
    }
    return lerp(color1_, color2, blend);
}
```

被写界深度のみ

被写界深度の実装

FOG＋被写界深度



Sneaking Mission

チームで制作したステルスアクションゲームです。
敵に見つからずに目標を達成することがクリア条件です。

主にギミックの制御やキーコンフィグの実装、ステージを作成して、そのステージを読み込めるようにしました。ギミックでは、監視カメラとプレイヤーの当たり判定（扇形と円の当たり判定）や発見時のイベントなどをしました。

製作時期：3年前期
製作期間：4か月
チーム人数：4人
使用言語：C++/HLSL/rapidXml
使用ライブラリ：DXライブラリ

キーコンフィグの実装

まず最初に、キーコンフィグの情報をファイルに吐き出すために、保存するときにファイルがない場合はキーコンフィグを保存するファイルを作成します。(*1)保存するときにキーボードかコントローラーを分けて保存するようにしています。

ロードするときにファイルがある場合は(*2)で読み込むようにしています。

ファイルがない場合はデフォルトを読み込むようにしています。

(*1)キーコンフィグをセーブ

```
bool InputConfig::Save(const std::string& fname, InputCode& in)
{
    std::ofstream ofs(path + fname);
    if (!ofs)
    {
        return false;
    }
    Header h[];
    std::vector<Data> vec;

    // マップからベクターに格納する
    for (InputID id = InputID::Dash; static_cast<int>(id) < static_cast<int>(InputID::Max);
    {
        vec.push_back({ id,in[id] });
    }

    // ヘッダ情報を作る
    h.size = vec.size();
    h.ver = 1.0f;
    for (auto& v : vec)
    {
        h.sum += static_cast<int>(v.id) * v.code;
    }
    // 書き込み
    ofs.write(reinterpret_cast<char*>(&h), sizeof(h));
    ofs.write(reinterpret_cast<char*>(vec.data()), sizeof(vec[0]) * vec.size());
    return true;
}
```

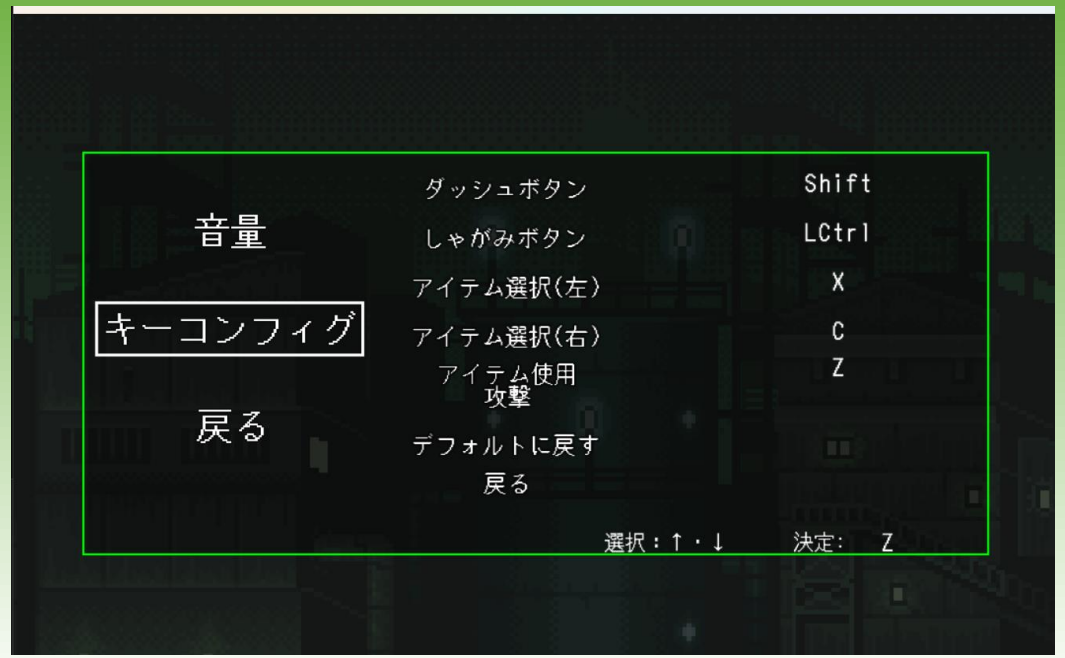
(*2)キーコンフィグを読み込む

```
bool InputConfig::Load(const std::string& fname, InputCode& in)
{
    std::ifstream ifs(path + fname);
    if (!ifs)
    {
        return false;
    }
    Header h{};
    std::vector<Data> vec;
    // ヘッダー情報を読み込む
    ifs.read(reinterpret_cast<char*>(&h), sizeof(h));
    // 設定データを読み込む
    vec.resize(h.size);
    ifs.read(reinterpret_cast<char*>(vec.data()), sizeof(vec[0]) * h.size);
    int sum = 0;
    // 以下sum値チェック
    for (auto& v : vec)
    {
        sum += static_cast<int>(v.id) * v.code;
    }
    if (sum != h.sum)
    {
        return false;
    }
    // マップに入れる
    for (auto& v : vec)
    {
        in.emplace(v.id, v.code);
    }
    return true;
```

キーコンフィグの実装

キーを変更するときにそのキーが他の動作に使用しているときはそのキーと変更前のキーを入れ替えるようにしています。(*1)キーボードと(*2)コントローラーどちらも入れ替えれるようにしています

キーコンフィグ画面



(*1)キーボードの変更

```
void InputConfig::SwapKeyInputCode(InputID id, int code)
{
    int tmpCode[ keyInputCode_[id] ];
    for (auto& keyCode : keyInputCode_)
    {
        if (keyCode.second == code)
        {
            keyCode.second = tmpCode;
        }
    }
    keyInputCode_[id] = code;
}
```

(*1)コントローラーの変更

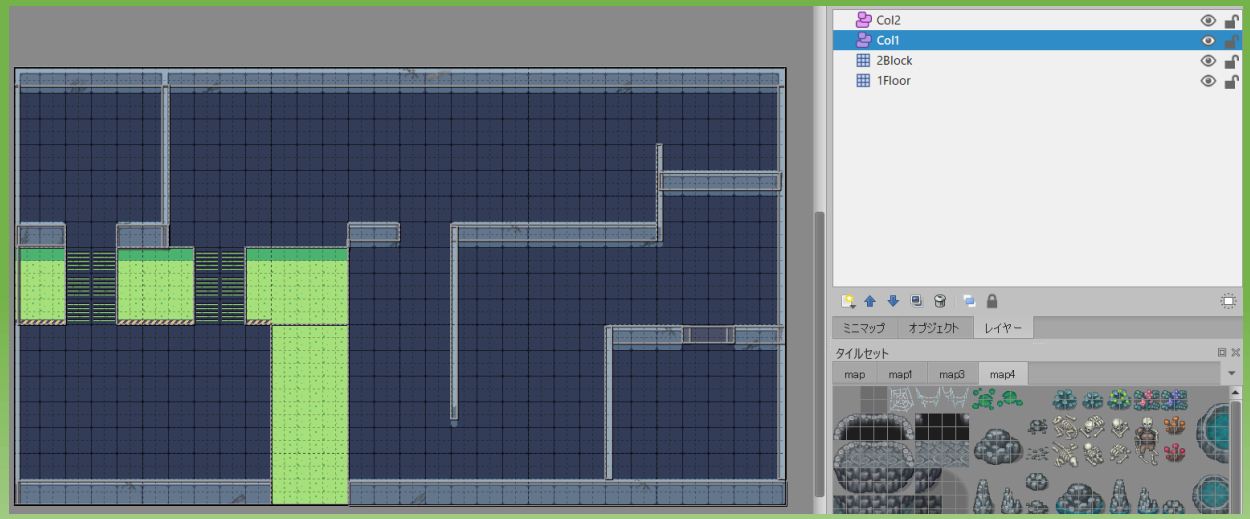
```
void InputConfig::SwapPadInputCode(InputID id, int code)
{
    int tmpCode[ joypadInputCode_[id] ];
    for (auto& padCode : joypadInputCode_)
    {
        if (padCode.second == code)
        {
            padCode.second = tmpCode;
        }
    }
    joypadInputCode_[id] = code;
}
```

ステージの描画

ステージの描画をするときにTiledを使って作成したマップにプレイヤーとエネミーが通れない場所を設定します。(*1)

その情報をrapidXmlを使用してステージ情報を読み込むようにしました。その中で、Tiledを使って設定したコリジョン情報を使ってレイを飛ばしてプレイヤーやエネミーが当たった時に押し出すようにして壁の中に埋まらないようにしました。コリジョンの情報を分けることでギミックが作動しているときは通れなくて、ギミック解除後に通れる壁を用意したりしました。

(*1) コライダーの指定



Ghost Hunt

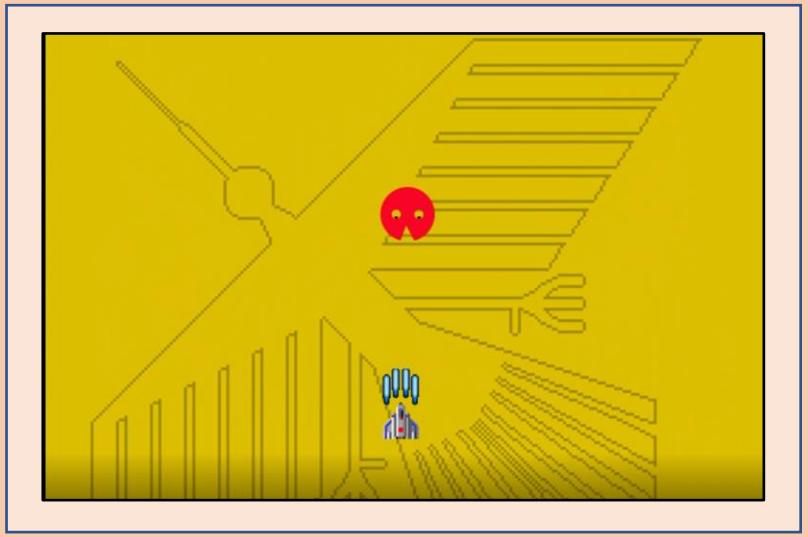
ステージ上にある武器を拾って、敵を倒していき最後に出現したボスを倒すことでクリアとなるゲームです。

ステージに出現する武器をランダムに出現するようにしたり、エネミーが幽霊なのでプレイヤーは通れないけど、エネミーは通れるようなものを作成しました。rapidXmlを使って、Tiledで作成したステージを使えるようにして、Tiledでコリジョンを設定してプレイヤーやエネミーが通れる道を設定したりしました。

製作時期	：2年後期
製作期間	：4か月
チーム人数	：1人
使用言語	：C++/rapidXml
使用ライブラリ	：DXライブラリ

授業作品

1年生



2Dシューティングゲーム

使　用　言　語：C言語
使用ライブラリ：DXライブラリ

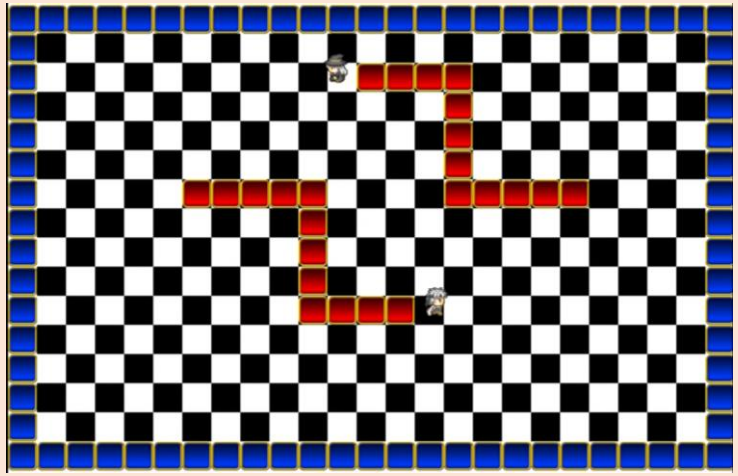
入学して初めての作品です。当たり判定や描画の仕方を学びました。ゲーム制作の基礎的な部分を知ることができました。

インベーダーゲーム

使　用　言　語：C言語
使用ライブラリ：DXライブラリ

敵の動きを個別で移動するのではなく、集団で移動する処理を学びました。
スコアの更新ができるようになり、
シーン遷移を実装することができました。

1年生



スネークゲーム

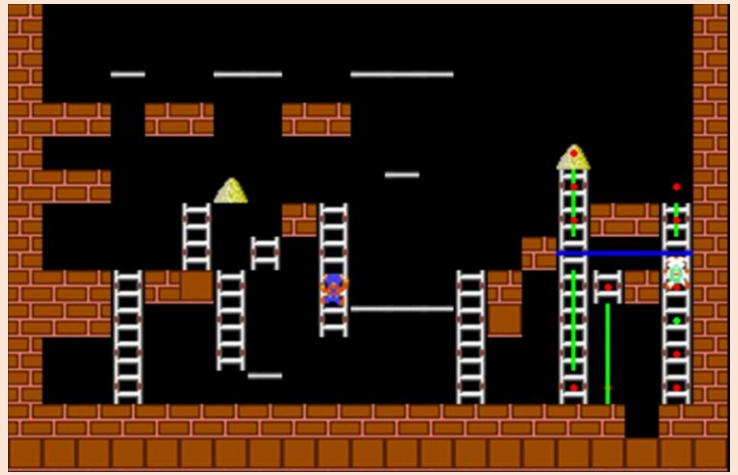
使　用　言　語：C言語
使用ライブラリ：DXライブラリ

初めてC++を用いたゲーム制作をしました。
前回までの制作で関数がいろいろとあふれていましたが、各クラスにまとめることができたことに感動しました。

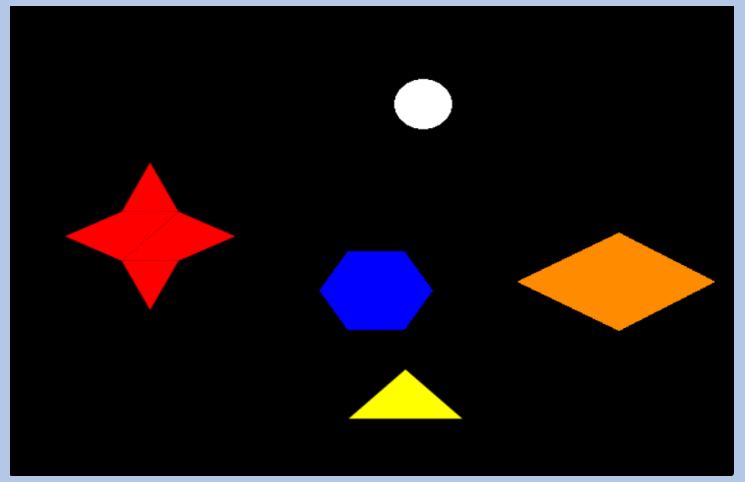
ロードランナー

使　用　言　語：C＋＋
使用ライブラリ：DXライブラリ

オブジェクト指向の基本を勉強しました。継承やデザインパターンのシングルトンパターンを利用して管理することを学びました。



2年生



当たり判定

使　用　言　語：C＋＋
使用ライブラリ：DXライブラリ

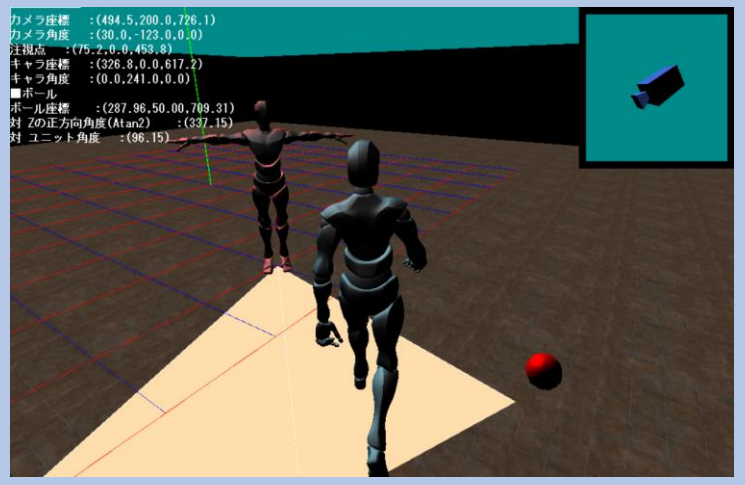
様々な形の図形を描画して当たり判定の勉強をしました。

2Dアクションゲーム

使　用　言　語：C＋＋
使用ライブラリ：DXライブラリ

レイを用いた地面や重力の制御やXMLファイルを利用したリソースの外部管理等を行った。コマンド入力についての学習もしました。

2年生



3Dビューワー

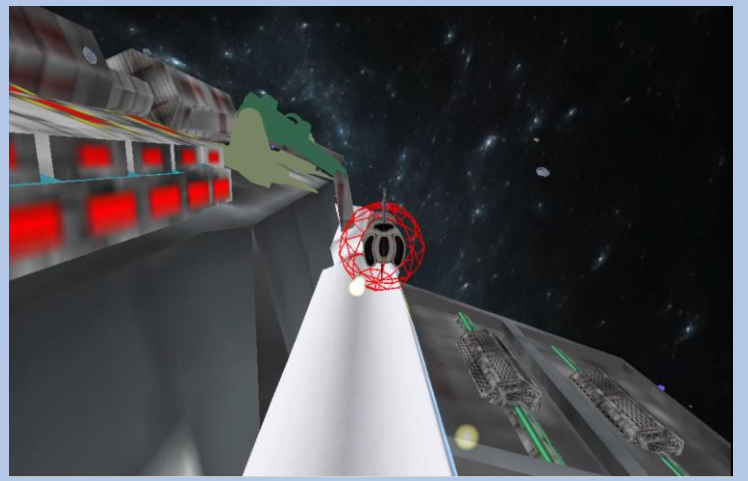
使　用　言　語：C＋＋
使用ライブラリ：DXライブラリ

3Dゲームに挑戦する前段階として制作したものです。3Dでの座標制御や回転などを勉強しました。

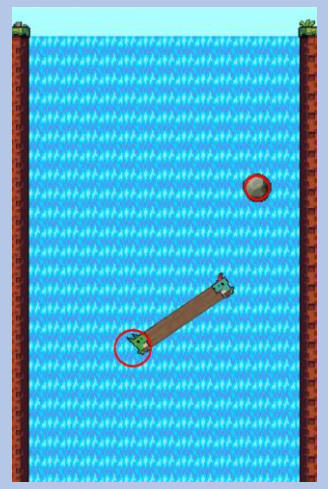
3Dシューティングゲーム

使　用　言　語：C＋＋
使用ライブラリ：DXライブラリ

3Dの回転を主軸にプレイヤーや敵機体の制御を行いました。最初は想定していた方向に移動しなかったりと大変なこともありましたがとてもいい勉強になりました。



2年生



丸太運びゲーム

使　用　言　語：C＋＋
使用ライブラリ：DXライブラリ

カプセル形状と円の当たり判定を使ったゲーム。
この学習の中で様々な形状と円の当たり判定の基礎を学んでこれからの制作に役立てようと思いました。

弾幕シューティングゲーム

使　用　言　語：C＋＋
使用ライブラリ：DXライブラリ

三各関数やベクトルを利用して、一年次に制作しシューティングのリメイクみたいなのを制作しました。
様々な種類の弾幕を生成することができました。

2年生

3Dアクションゲーム

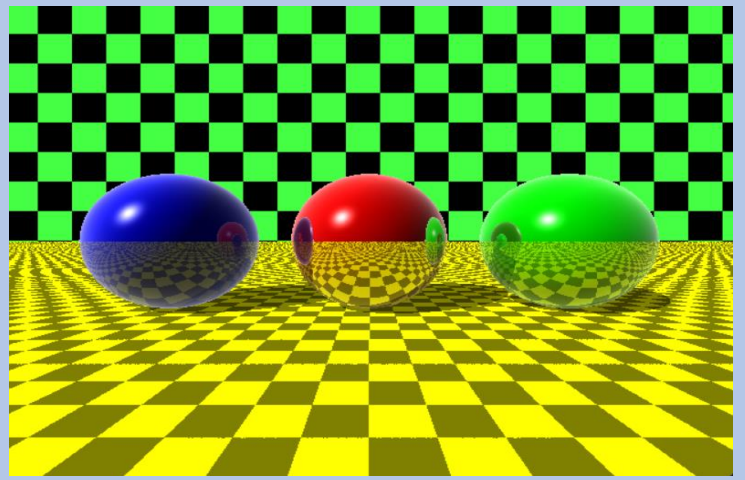
使　用　言　語：C＋＋
使用ライブラリ：DXライブラリ

マリオギャラクシーのように重力判定やカメラ制御の勉強をしました。様々な重力に対しての処理を学ぶことができたので良かったです。

レイトレーシング

使　用　言　語：C＋＋
使用ライブラリ：DXライブラリ

マテリアルの三要素の計算処理やレイの扱いなどの基礎を学ぶことができました。ゲームではありませんが非常に勉強になる学習でした。



Thank you!