Port folio

ASOポップカルチャー専門学校　ゲーム学科
中村　天ノ助

目次

- Crave Moisture 二年生後期
- Re BestFight 二年生前期
- 今までの軌跡

Crave Moisture

制作時期　2年生後期

制作期間　3か月

制作人数　1人

ジャンル　3D対戦アクションゲーム

Grave Moisture

ゲームの概要

二年生最後の集大成として制作したゲームです。
デビルメイクライとエルデンリングを参考に制作したゲームで、
派手なアクションを駆使して強敵である「悪魔」を倒す
シンプルなゲームデザインです。重要な要素として、
攻撃のパリィやジャスト回避を実装しています。

Grave Moisture

ポリゴンによるエフェクト

プレイヤーの武器(剣)
から出る、紫色の軌跡を
ポリゴンの自動生成で
表現しました。

Grave Moisture

ポリゴンによるエフェクト

制作に至った経緯

制作の初期段階では、パーティクルエフェクトによって表現しようとしていましたが、軌跡の生成するごとに別々で制作しなければならず、時間がかかりすぎていました。
そこで、ポリゴンを自動生成することで、労力を減らすだけでなく自由度の高いエフェクトを簡単に生成できるようにしました。

Grave Moisture

ポリゴンによるエフェクト

自動生成の手順

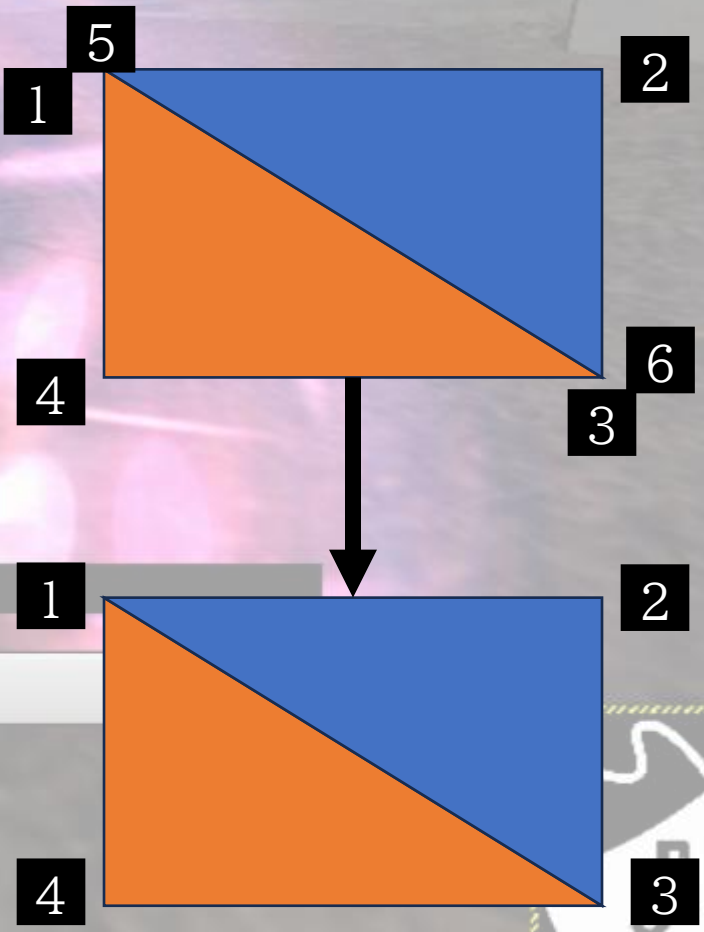
手順1::剣の先端と少し下の座標を取得
手順2::座標をそのまま頂点として使用
手順3::手順1の座標を動的配列に格納し、保持する
手順4::配列の情報を元にポリゴンを描画
手順5::以上1～4までをフレームごとに繰り返す
手順6::あらかじめ決めたタイマーで自動生成を終了

Grave Moisture

ポリゴンによるエフェクト

工夫した点_1

頂点インデックスを使って、重なっている
頂点の生成を省略することで、頂点数を
削減しました。
ポリゴンはタイミングによって100個を
超え、頂点数は600以上ですが、
工夫により400以下まで削減しています。



Grave Moisture

ポリゴンによるエフェクト

工夫した点_2

生成したポリゴンは、フレームごとに
外側に広がる処理をしています。
これにより、ポリゴンがよりエフェクトに
見えやすいようにしています。

Grave Moisture

カメラワークによる特殊演出

プレイヤーが「攻撃」「回避」
「ダッシュ」をした瞬間に
それぞれで特殊なカメラワーク
を実装しています。

Grave Moisture

カメラワークによる特殊演出

実装方法

・制御に必要なデータを外部に 保存しています。
・外部ファイルを読み込み、アクションをキーとして制御

Grave Moisture

カメラワークによる特殊演出

外部ファイルについて

・JSON形式のデータ
・”distance”::カメラとプレイヤーとの距離
・“speed”::カメラ制御の処理速度
・”relativeTarget”::カメラの視点の座標
・“cameraAngle”::カメラの角度
・“eventTime”::”eventName”の実行時間
・”eventName”::カメラシェイクなどの特殊処理のキー


```
"Direction_1" :{
    "distance" : [ 500.0 ],
    "speed": [ 2.0 ],
    "relativeTarget": [ 0.0, 0.0, 0.0 ],
    "cameraAngle": [ 0.0, 0.0, 0.0 ],
    "eventTime": [ 2.0 ],
    "eventName": [ "None" ]
},
  "Direction_2": {
      "distance": [ 1000.0 ],
      "speed": [ 5.0 ],
      "relativeTarget": [ 0.0, 0.0, 0.0 ],
      "cameraAngle": [ 0.0, 0.0, 0.0 ],
      "eventTime": [ 45.0 ],
      "eventName": [ "None" ]
```

Grave Moisture

カメラワークによる特殊演出

外部ファイルについて

データの読み込みは、専用のロードクラスを制作して、
コンポーネント化しPlayerクラスで使用しています。
具体的なコードは「Player.cpp」の455行目に
記しています。

Grave Moisture

カメラワークによる特殊演出

実装例

・回避した瞬間にカメラを
斜めにして臨場感を表現

Grave Moisture

BehaviorBaseのAI

エネミーがなるべく状況に合わせて
臨機応変な行動をとれるようにAIを
作成しました。

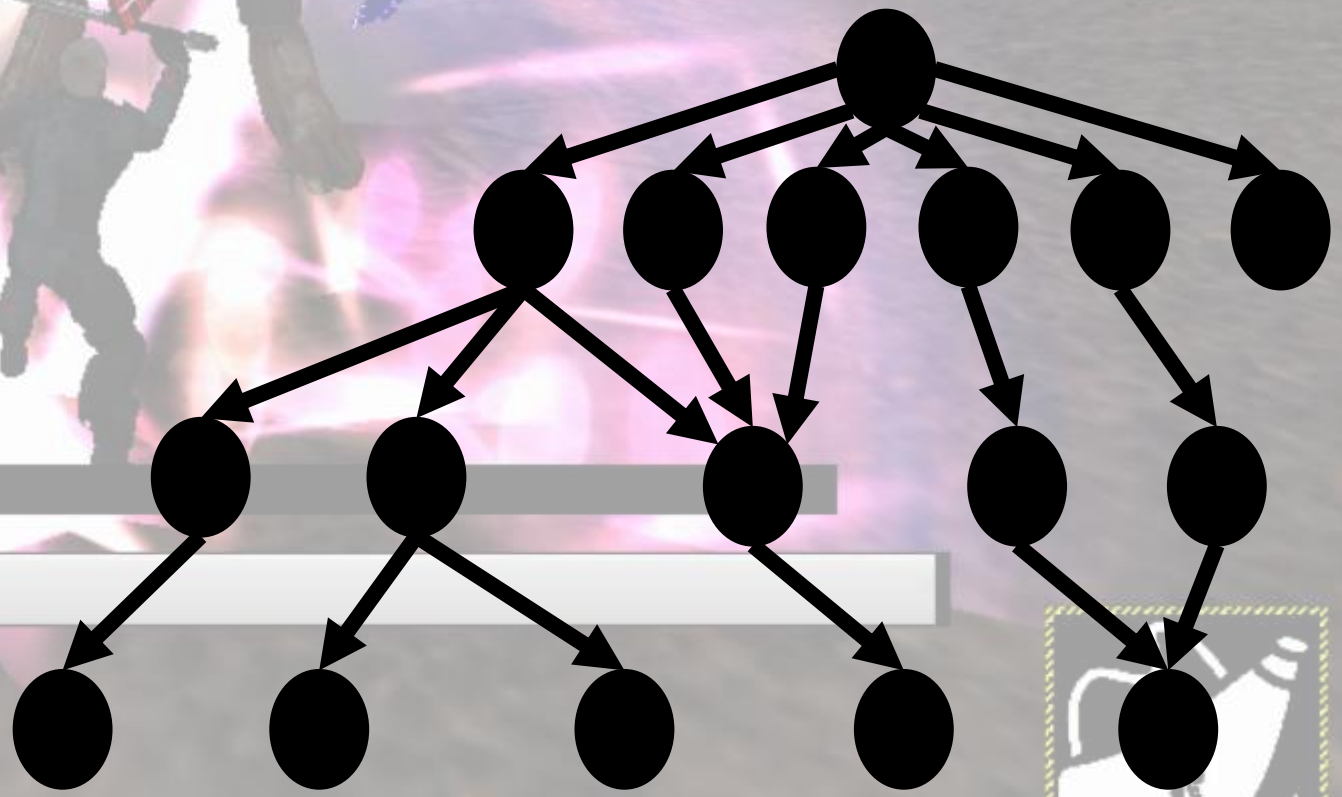
Grave Moisture

BehaviorBaseのAI

BehaviorTree

・定義::黒いマルを「振る舞い」
　矢印を「エッジ」とする。

・上から順に遷移して、終端に
　たどり着いたら、アクション



Grave Moisture

BehaviorBaseのAI

コードの詳細

・終端まで遷移し続けて、
インデックスを求めます。

・遷移条件によって遷移
しなかったら、そこで遷移終了

（「BehaviorBase.cppの179行目」）

```
179         bool isSearch = false;
180         //遷移後のインデックス
181         int nextIndex = NONE_INDEX;
182
183         while (!isSearch)
184         {
185             nextIndex = TransitionNextNode(nowIndex_);
186
187             //次のノードに遷移しなかった
188             //ループの終了
189             if (nextIndex == NONE_INDEX)isSearch = true;
190
191             //次のインデックスに遷移
192             else
193             {
194                 nowIndex_ = nextIndex;
195                 //AIのアクション終了を取り消す
196                 isEndAIAction_ = false;
197             }
198         }
```

Grave Moisture

BehaviorBaseのAI

コードの詳細

・遷移はエッジデータに
基づいて行われ、遷移条件を
満たしていたら 遷移して終了

（「BehaviorBase.cppの150行目」）

```
150     for (const auto& edge : edges_)
151     {
152         if (edge.second->GetFromIndex() != fromIdx)continue;
153
154         //接続されているすべてのノードを検索
155         endNodeSize = edge.second->GetEndNodeNum();
156         for (int ii = 0; ii < endNodeSize; ii++)
157         {
158             if (!edge.second->IsCondition(ii))continue;
159
160             //コストが大きいほど優先度が高い
161             //コストが同じなら、後に格納されている順
162             if (nowWeight <= edge.second->GetCost(ii))
163             {
164                 nowWeight = edge.second->GetCost(ii);
165                 idx = edge.second->GetEndIndex(ii);
166             }
167         }
168
169         //エッジが見つかったなら終了
170         break;
171     }
```

Grave Moisture

BehaviorBaseのAI

コードの詳細

・ノードは、関数ポインタで
アクションを所持し、自身の
インデックスで識別する

（「Node.hの44行目」）

```
44      private:
45
46          //更新関数ポインタ
47          const ActionFunc action_;
48
49          //自身のノード番号
50          const int nodeIndex_;
51
```

Grave Moisture

BehaviorBaseのAI

コードの詳細

・エッジデータは遷移前と
遷移後のインデックス複数
を所持し、遷移条件を
関数ポインタで保持している

（「Edge.hの44行目」）

```
77  private:
78
79      //遷移条件
80      ConditionFunc condition_;
81
82      //フロムノードのインデックス
83      int fromIdx_;
84      //エンドノードのインデックス
85      std::vector<int> endIdxs_;
86
87      //エッジの重さ
88      std::vector<float> weight_;
```

Grave Moisture

シャドウマッピング

被写界深度を応用して、すべてのモデルに
影を落とすレンダリングを行いました。
HLSL言語を用いて独自のシェーダープログラムを
書きました。

Grave Moisture

シャドウマッピング

課題点

シャドウマッピングで使用した深度マップの解像度の低さが原因で、本来あるべき場所に影がないこと

Grave Moisture

シャドウマッピング

解決

光源とモデル頂点の法線とで内積をとり、
一定以上の大きさだった部分は影を落とさないようにすることで
解決しました。

Grave Moisture

シャドウマッピング

コードの詳細

法線とライトの位置から内積を求め、ライトの影響度を計算

ShadowModelPS.hlsl

```
// 法線マップを法線ベクトルに
float3 norm = (normalMapTexture.Sample(normalMapSampler, uv) * 2.0f - 1.0f).rgb;
norm = normalize(norm);

// ワールド空間上のライト位置と法線との内積を計算
float3 lightDirNorm = normalize(g_lightDir);
float NdotL = saturate(dot(norm, lightDirNorm));

// ワールド空間上の視点位置と法線との内積を計算
float3 viewDirNorm = normalize(g_camera_pos - PSInput.worldPos);
float NdotV = saturate(dot(norm, viewDirNorm));

// ライトと視点ベクトルのハーフベクトルを計算

float3 halfVector = normalize(lightDirNorm + viewDirNorm);

// D_GGXの項
float D = D_GGX(halfVector, norm, uv);

// F(フレネル反射項)
float F = Flesnel(viewDirNorm, halfVector);
```

Grave Moisture

シャドウマッピング

コードの詳細

深度マップから深度を取り出し、一定以上だった場合は頂点の色を暗くする

ShadowModelPS.hlsl

```
    // 深度テクスチャの座標を算出
    // PSInput.prjPos.xy は -1.0f ～ 1.0f の値なので、これを 0.0f ～ 1.0f の値にする
    DepthTexCoord.x = (PSInput.prjPos.x + 1.0f) / 2.0f;
    // yは更に上下反転
    DepthTexCoord.y = 1.0f - (PSInput.prjPos.y + 1.0f) / 2.0f;

    depth = shadowMap0Texture.Sample(shadowMap0Sampler, DepthTexCoord).x;

    if (PSInput.prjPos.z > depth + 0.0002f || d <= 0.545f)
    {
        color.rgb *= 0.45f;
    }


    return color;
}
```

Re:Best Fight

制作時期　2年生前期

制作期間　3か月

制作人数　1人

ジャンル　3D対戦アクションゲーム

Re:Best Fight

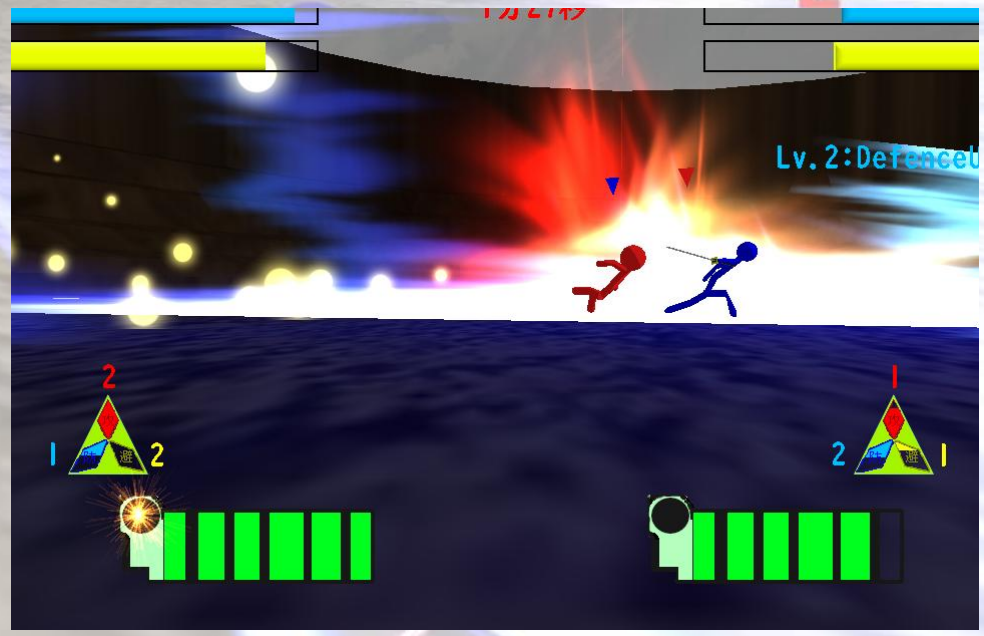
ゲームの概要

二年生で、人生初の3Dゲームを制作しました。
ジャンプフォースを参考に制作したゲームで、
二人プレイ推奨ですが、一人でも遊べるようにAI
を実装しています。必殺技もかなりこだわって
制作しています。

Re:Best Fight

こだわった部分::必殺技

キャラクターは全部で3種類用意し、
それぞれに特有の必殺技演出を
実装しています。
カメラワークは「CraveMoisture」
と同じ仕組みですが、一体につき
2週間以上かけて試行錯誤を
重ねています。



Re:Best Fight

技術的なアピールはCraveMoistureですが、
Re:Best Fightは私のこだわりの強さを反映した作品です。
ぜひ、以下のURLから動画を見てみてください。

https://youtu.be/BhkPpED-Jnc

今までの軌跡

専門学校に入学をしてから今までに制作してきたゲームをそれぞれ掲載しています。（以降、15ページあります）

宇宙戦艦防衛戦

詳細

プラットフォーム　PC

使用言語　C＋＋

使用ライブラリ　DxLib

制作人数　1人

制作期間・時期　3か月・1年生前期

ジャンル　2Dシューディングゲーム

説明

初めて制作したオリジナルゲームです。
企画、実装まですべて担当しました。
スクリーンが1つ～3つに分かれており、
スクリーンを切り替えながら、敵を砲台で倒し、
制限時間まで戦艦を守り切ることでクリアです。
敵の種類に応じたスコアも実装しました。

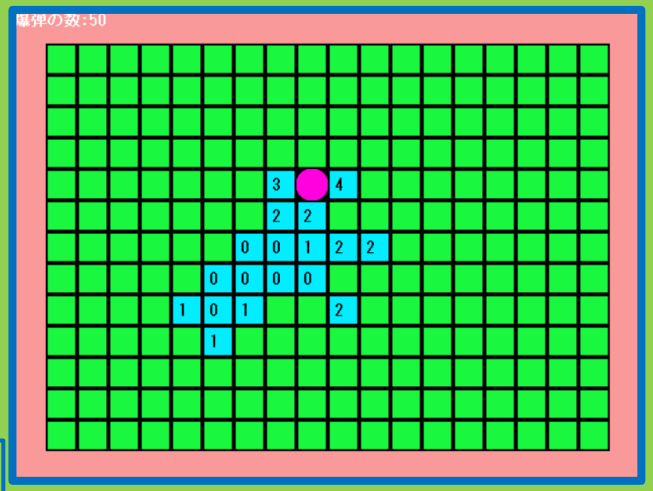
スコアは外部ファイルに記録しており、
ランキングとしてゲーム内に表示できます。

アイテムを8種類実装しており、
一定時間でランダムに3つ取得・使用できます。

使用言語・ライブラリなどすべて初めてで、
まずは完成させることを目標に制作しました。

MineSweeper

詳細



プラットフォーム　PC

使用言語　C＋＋

使用ライブラリ　DxLib

制作人数　1人

制作期間・時期　1日・1年生前期

ジャンル　MineSweeper

説明

プログラムの配列の計算を効率化する手段を学ぶために制作しました。
ゲームはGoogleで遊べるMineSweeperと同じ仕様です。

配列で格子状に表現しています。計算量を削減し、処理速度を上げるために、
配列の初期化時に先頭に地雷を個数分置き、
初期化終了時に、ランダムな二つの要素番号で配列をswapする処理を配列の要素数分行うアルゴリズムを実装しました。

動的に格子の数と地雷の数を決められるように
vector配列を使用して実装しています。

TypingGame

詳細

プラットフォーム　PC

使用言語　C＋＋

使用ライブラリ　DxLib

制作人数　1人

制作期間・時期　2日・1年生前期

ジャンル　TypingGame

説明

自分のタイピング速度を上げたいと思い制作しました。
内部でロードしてある文字列データをランダムに選択され、それをローマ字で打つ普通のタイピングゲームです。

UIを充実しており、打ち間違いがわかりやすいようにその文字の色だけ変えたり、確定ボタンを押した際の正解・不正解の演出を実装しました。

入力文字の比較処理で、計算効率を上げるために入力した瞬間だけ、入力回数に対応した文字と比較するようにしています。

Obvious/Strange

詳細

プラットフォーム　PC

使用言語　C＋＋

使用ライブラリ　DxLib

制作人数　2人

制作期間・時期　3か月・1年生後期

ジャンル　謎解き脱出ゲーム

説明

友人と一緒に二人チームで制作しました。
私はプログラムを担当しました。
ギミックのある部屋が4つあり、ギミックをすべて解くことで出口を開き、脱出できます。

ギミックは磁石や化合など、「科学」を基にしています。磁石の極を切り替えたり、アイテムを化合し、別のアイテムを作り、使用する機能を実装しました。

ギミックに使用するアイテムは10種類を超えており、それらを管理するためのシステムを構築しました。

初めてのチーム制作で、コミュニケーションを積極的に取りながら認識の違いが起きないように政策を進めました。

お祭り射的ゲーム

詳細

プラットフォーム　PC

使用言語　C＋＋

使用ライブラリ　DxLib

制作人数　1人

制作期間・時期　1週間・2年生前期

ジャンル　3D射的ゲーム

説明

3Dゲームの当たり判定やベクトル操作を学ぶために、制作しました。
一定時間内に、配置されている景品をできるだけ多く撃つことで高いスコアを目指すゲームです。

射的の弾は発射時に、銃の向いている角度に合わせて、前方向のベクトルを計算しています。

当たり判定はDxLibライブラリの関数を使用していますが、当たり判定の必要な情報だけを取得するために、情報取得用の関数を自力で実装しました。

また、当たり判定は景品の3Dポリゴンと銃の発射時のベクトルで計算し、処理速度と精密性を向上させる工夫をしています。

CoockingGame

詳細

プラットフォーム	PC
使用言語	C＋＋
使用ライブラリ	DxLib
制作人数	4人
制作期間・時期	2日・2年生後期
ジャンル	闇鍋スコアアタック

説明

福岡学生ゲームジャムという組織のイベントに参加した際に、ランダムに決められたチームで、テーマ「ごちゃまぜ」で制作しました。私は、チームの中でリードプログラマーとして携わりました。

1フェーズで上空から食べ物がランダムで落下していき、それをマウスで拾い、猫の鍋に入れることでポイントを得ます。そして、2フェーズのルーレットの結果によってポイントを乗算し、スコアが決まります。

制作の中では企画に最も時間をかけ、チーム内でのゲームの形を統一することを意識しました。

2日間のゲームジャムということもあり、実装に限りがありました。いかに効率よくプログラムを組めるかを常に考えて作業し、完成させました。


これ以降、授業作品です

1年生

制作時期	1年生前期
使用言語	C++
使用ライブラリ	DxLib

1年生で最初に制作したゲームです。簡単な2Dのシューティングゲームになっています。
アニメーションや矩形の当たり判定など、ゲーム作りにおける基本的なことを学びました。
縦にスクロールする処理など、通常のプログラミングとは違う、ゲーム制作においてのプログラミングの考え方を学びました。

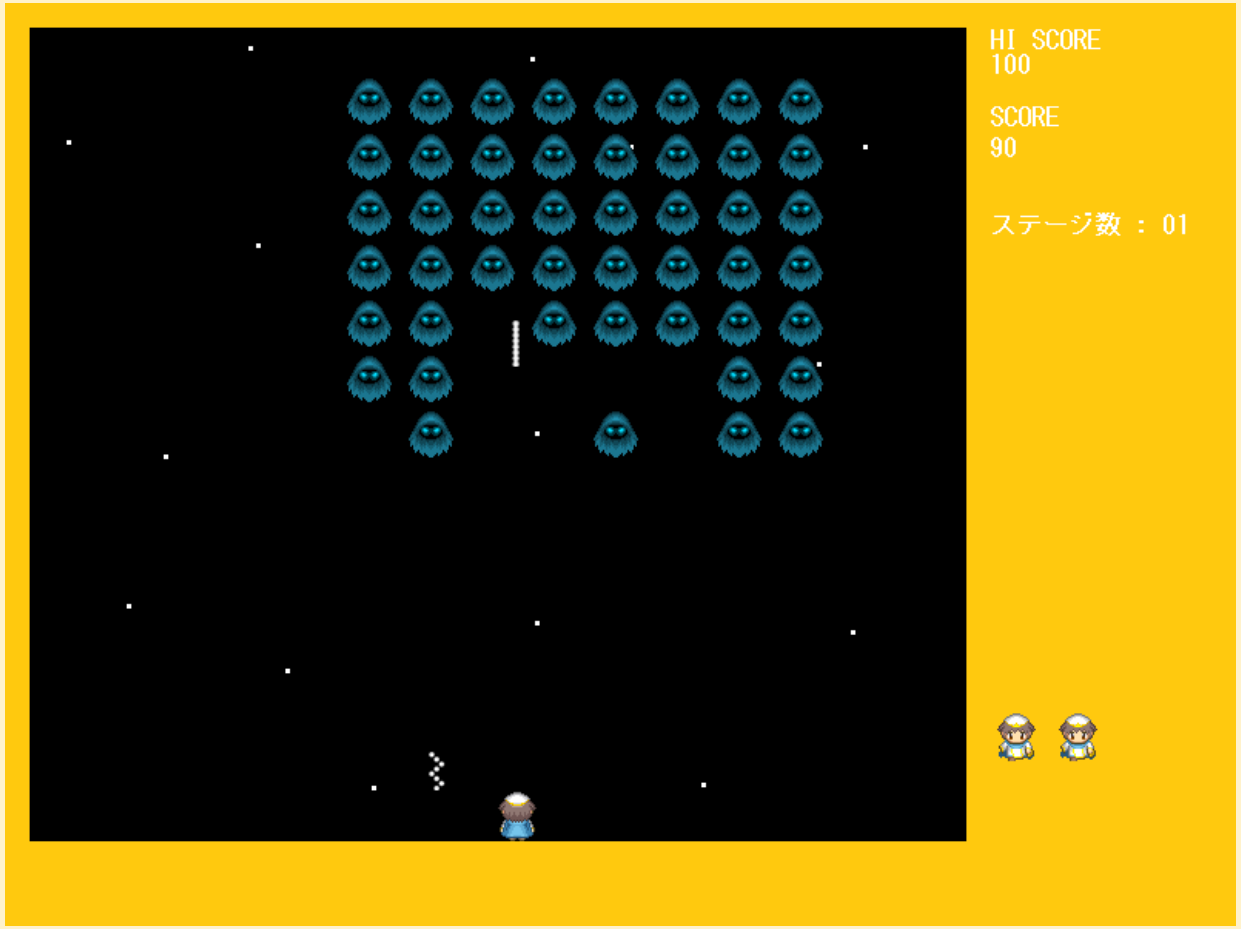
制作時期　　1年生前期

使用言語　　C++

使用ライブラリ　　DxLib

エネミーの状態を管理するという課題でした。

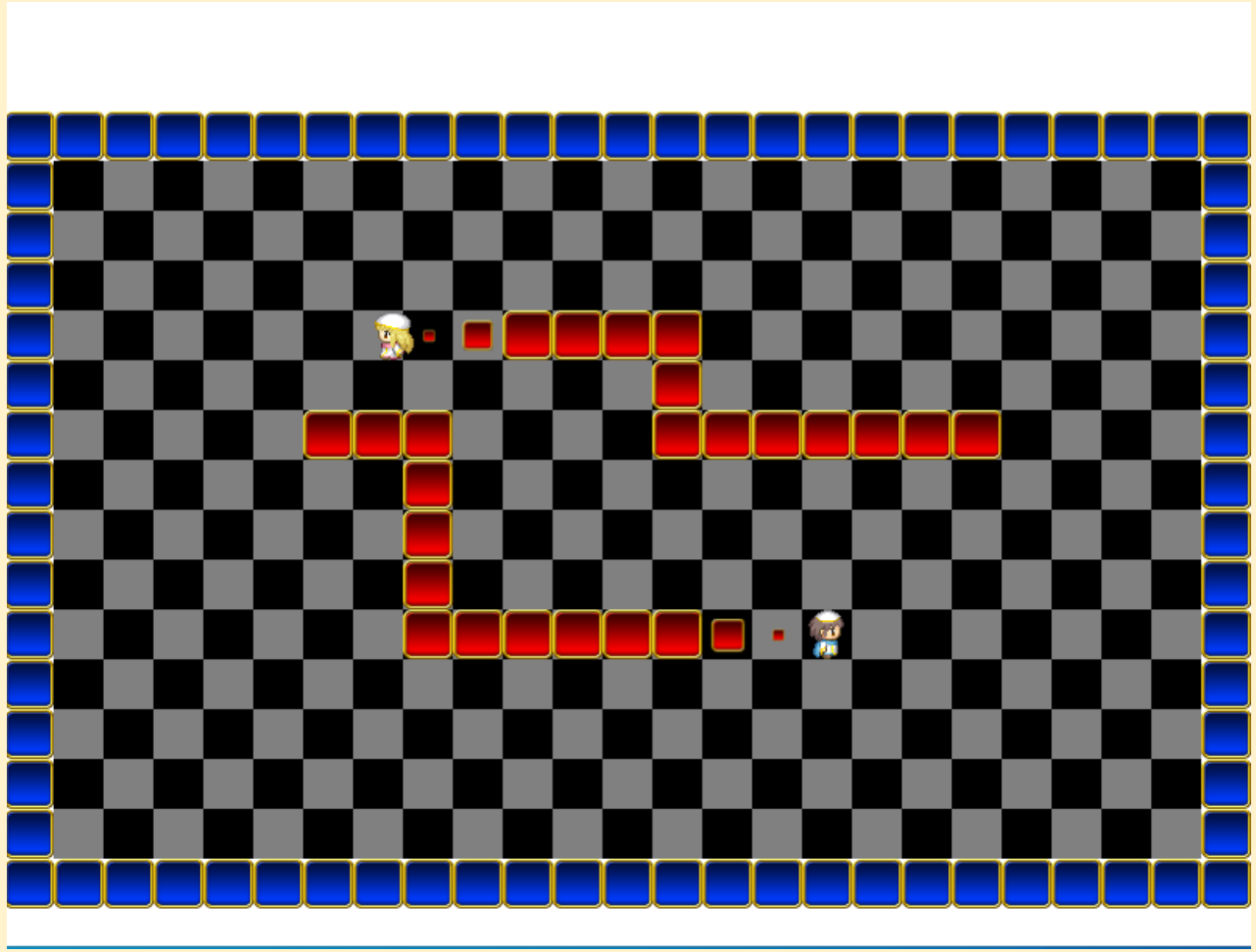
エネミーを個ではなく集団で制御をしています。
また、集団の中でもプレイヤーによって倒されているかで
描画をするか決める使用を実装しています。
また、エネミーをすべて倒したら次のステージに進めるや
ハイスコアをゲームプレイ中だけ保存し、ゲームとして楽しめ
るようにしました。



制作時期	1年生前期
使用言語	C++
使用ライブラリ	DxLib

スネークゲームを制作しました。マップチップの配置を動的に行うことが課題でした。

プレイヤーは進行方向の情報をあらかじめ持ち、自動で進みます。1マスごとに方向転換できるように制御を行いました。またゲームの終了条件は、
「配置されたマップチップに衝突」、「壁に衝突」、
「キャラクター同士が衝突」と複数あるため、それらをまとめて管理するクラスを作成しました。



制作時期　　1年生前期

使用言語　　C++

使用ライブラリ　　DxLib

簡単なRPG風のスコアアタックゲームです。

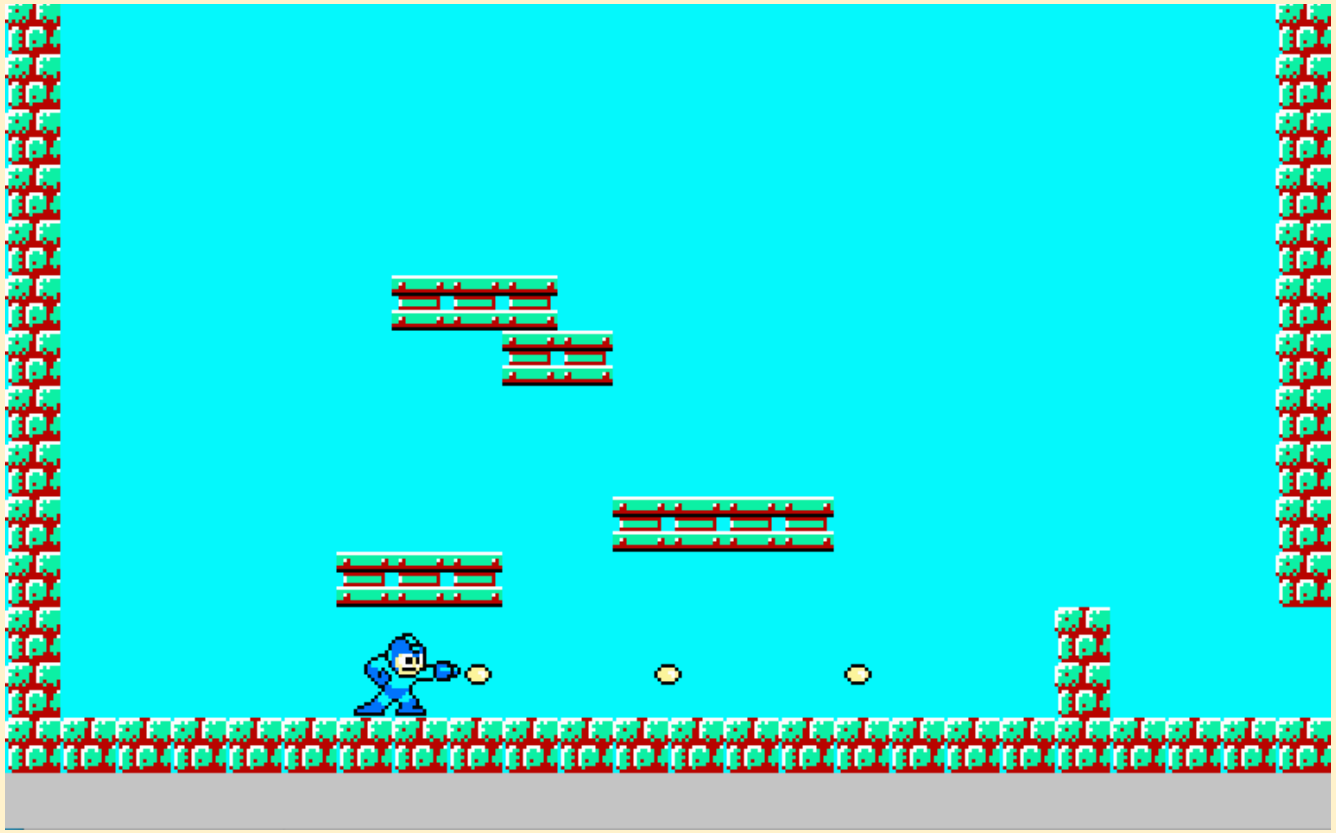
外部データから取り込んだ情報を元に、ステージを作成する課題でした。
CSV形式のデータからステージの広さに応じて動的に、プログラム内の配列に取り込み使用しました。
また、動きがそれぞれで異なる複数のエネミーを独立して動かしつつ、一元管理することにも挑戦しました。
左上にある階段に触れると、エネミーのいない
別のステージに移動することもできます。

制作時期　　1年生前期

使用言語　　C++

使用ライブラリ　　DxLib

初めてのアクションゲームの制作をしました。
1年生前期の集大成で、今までに習ってきたことを
全て活用して制作しました。
プレイヤーに対する重力や体の部位に合わせた矩形
の当たり判定を実装しています。具体的には、
プレイヤーの足のつま先がブロックに接していると
つま先立ちができるようにしています。
撃っている銃弾や種類の異なるブロックなどを
なるべく細かくクラス分けし、オブジェクト指向を
意識しながら制作しています。



2年生

制作時期　　2年生前期

使用言語　　C++

使用ライブラリ　　DxLib

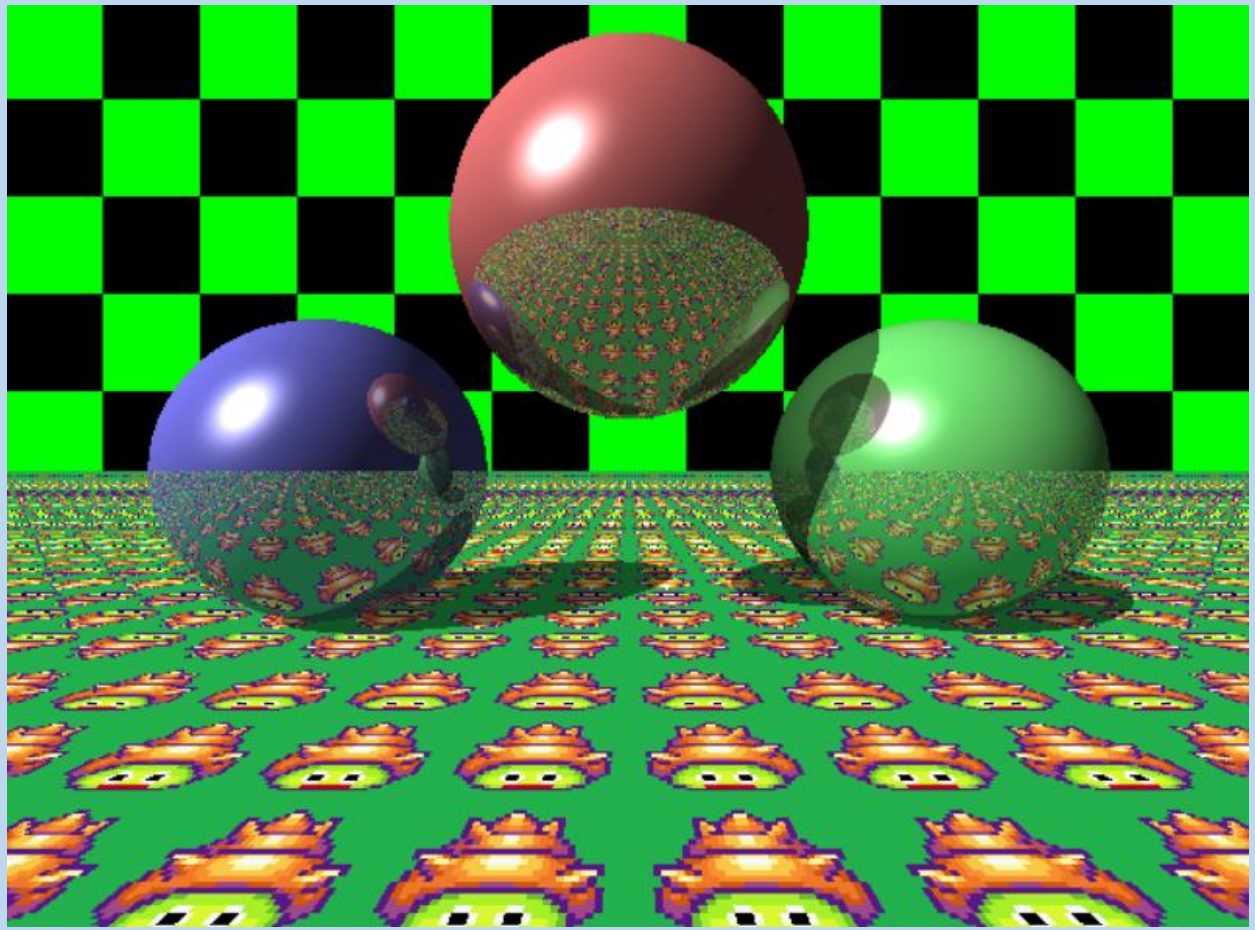
2年生の初期に、3Dのゲーム制作の授業を受けました。
DxLibライブラリにおける、3D画面上のカメラやオブジェクトの基本的な扱い方を学びました。
プレイヤーの地面に対する当たり判定を正確に行うために、ベクトルと3Dオブジェクトとの当たり判定を使用するなど、見た目の違和感をなるべく減らすことを意識しました。また、カメラをプレイヤーに追従させる機能も実装しています。プレイヤーが動くと同時にカメラを同じ方向に動かし、またキー入力でカメラのアングルを変えることもできるようにしました。

制作時期　　2年生前期

使用言語　　C++

使用ライブラリ　　DxLib

前回に制作した3Dゲームをさらに改良して、HLSL言語を用いた様々なレンダリングを実装しました。
今まではDxLibライブラリが自動で行ってくれていたのですが、それらをあえて無くし、独自のシェーディングを行っています。
メタリックやリムライト、水面の揺らぎを独自でシェーディングしています。またステージ全体の陰影処理を実装しました。
被写界深度など、授業だけでは理解が難しかったものもあり、オリジナルゲームの制作をしながら理解を深めていきました。



制作時期	2年生後期
使用言語	C++
使用ライブラリ	DxLib

HLSL言語によるシェーダープログラムを使用せずに、レイトレーシングをc++側で実装するという授業でした。3D上の情報を元に2D画面で交差判定を取り、球を描画しています。球の陰影やスペキュラーはもちろん、床の鏡面反射もc++側で行っています。

数学の知識が多く必要で、交差判定の数学的な理屈を理解することにとても苦労をしました。

この授業以降では、より深い理解で3Dモデルのレンダリングを行うことができるようになりました。