齊藤 芳紀
ポートフォリオ

プロフィール

名前

齊藤芳紀

取得している資格

基本情報技術者試験
応用情報技術者試験
CGエンジニア検定エキスパート
CGクリエイター検定エキスパート
画像処理エンジニア検定エキスパート
漢字検定2級

プログラミングが好きです

自宅でも趣味でやっています

学校で学ぶこと以外にも積極的に取り組みました

DirectX11, OpenGL, Python, C#, PyQt, STL, Lua, Unity, シェーダ　など

新しいことに挑戦するのは苦手ですが、粘り強く取り組んで、形にできていると思います

GitHubやってます

https://github.com/KanayaMaki/CPot

強みはマルチプラットフォームです

2年次はWiiとWindowsで動くゲームを作成し、
3年次は**DirectX11とOpenGL**でゲームを作れるライブラリを作成しました

ゲーム部分を完全に共通化することを実現しています

ゲームエンジンの制作に携わることが目標です

様々なプラットフォームへの深い知識、高度な設計、高速化など
多くの高いスキルが必要なゲームエンジン制作は、
とても**やりがいのある仕事**だと考えています

ですが、プログラム自体が好きなので、正直どんな仕事でもゲーム関係のプログラムができれば満足です
ゲームに関わりたいと思うのは、一般の人に楽しんでもらえることもやりがいにつながると思うからです

チームの為に行動できます

リーダー役がいなければその場を仕切る、誰かがリーダー役なら積極的に
行動し助ける。和を乱さず、**チームの為**に行動してきました

1・2年次に、チーム制作のリーダー経験があります

作品紹介

POTライブラリ

3年次制作。**DirectX11とOpenGL**で同じゲームを作成できるライブラリです

OminoJumper

1年次制作。コンソールを使用していて、**描画の高速化**を工夫しました

プルぷるプッシュ！！

2年次制作。WiiとWindowsで同じゲームがプレイできます

その他

Unityを使用した通信アクションゲーム、PyQtを使用したGUIアプリなど

POTライブラリ

DirectX11と**OpenGL**で同じゲームを作成できるライブラリです
コンポーネント指向も実装しています

使用言語：C++　　制作人数：1人
制作期間：3カ月　　制作時期：3年次

DirectX11

OpenGL

DirectX11とOpenGLの切り替えが可能です

ゲーム部分のコードを一切変えずに、
DirectX11とOpenGLを切り替えることができます

```
// 箱の読み込み
//
StaticTangentMeshModelCPU lBampMeshCPU;
BufferToMesh::Load(lBampMeshCPU, PathString("./Cube/cube.pmo"));

auto bampModel = std::make_shared<StaticTangentMeshModel>();
ModelCPUToModel::Load(*bampModel, lBampMeshCPU);


// ゲームオブジェクトの作成
//
GameObject* lObject = new GameObject;
lObject->SetName("Box");
lObject->AddComponent<StaticTangentModelRenderer>();
lObject->GetComponent<StaticTangentModelRenderer>()->model = bampModel;
lObject->AddComponent<AutoRotateComponent>();
```

これで3Dのモデルが表示できます

```
#ifdef CPOT_ON_DIRECTX11
 #include "./Pot/Render/DirectX11/texture2DDirectX11.h"
#elif defined CPOT_ON_OPENGL
 #include "./Pot/Render//OpenGL/texture2DOpenGL.h"
 #endif
```

インターフェースを同じにしてマクロで分岐することで
実現しています

頂点モーフィングや、**レンダーテクスチャを使用したシェーダ**も使用できる、
細かい操作の可能なライブラリです

将来的にはLinuxとWindowsでマルチプラットフォームを可能にしたいです

問題点と対応

OpenGLでテクスチャを使用する際に、画像が逆転してしまう

原因：OpenGLでは、テクスチャデータが左下から格納されている

対応1：モデルのUV座標を上下逆転させる

→トゥーンテクスチャのUV値などは上下逆転させる必要があり、シェーダを自動で共通化させることが難しいので、不採用でした

対応2：テクスチャを作成するときに上下逆転させる

→レンダーテクスチャなどはおそらく上下反転できないので、不採用でした

対応3：シェーダからテクスチャを参照するときにUV値を上下反転させる

→シェーダを共通化させるときにも一律で上下反転すればいいので、対応可能です。この方法が一番いいと採用しました

各プラットフォームによって対応している機能に差がある

対応：各プラットフォームの機能を調べ、最大公約数的な機能を実装しました

→一番下の機能に合わせる必要があり、あまりよくなかったかもしれないです
力不足を感じるので、プログラマとしてもっと成長したり、他の人と協力することで、解決していきたいと考えています

コンポーネント指向を実装しています

Unityのように、ゲームオブジェクトに**コンポーネント**を追加できます

```
//これでコンポーネントを定義
class PresenComponent : public Updater {
    CPOT_TYPE(PresenComponent, Updater)

public:
    void OnUpdate() override {}
};
CPOT_REGIST_TYPE(PresenComponent);


void PresenFunc() {

    GameObject* lObject = new GameObject;
    lObject->SetName("Saito");  //ゲームオブジェクトに名前を付ける

    lObject->AddComponent<PresenComponent>();   //コンポーネントを追加
    lObject->GetComponent<PresenComponent>();   //コンポーネントを取得
    lObject->AddComponent("PresenComponent");   //文字列からでもコンポーネントを追加できる
}
```

簡単にコンポーネントを作成・追加できます

```
//派生クラスで宣言する
#pragma region Extend

#define CPOT_TYPE(THIS, SUPER)                                  ¥
    using This = THIS;                                          ¥
    using Super = SUPER;                                        ¥
public:                                                         ¥
    static const Type& SGetTypeName() {                         ¥
        static Type t(CPOT_NAME_EXTEND(THIS));                  ¥
        return t;                                               ¥
    }                                                           ¥
    BOOL CanCast(const Type& aTypeName) const override {        ¥
        if (THIS::SEqualType(aTypeName)) return true;           ¥
        return SUPER::SEqualType(aTypeName);                    ¥
    }                                                           ¥
    BOOL EqualType(const Type& aTypeName) const override {      ¥
        return THIS::SEqualType(aTypeName);                     ¥
    }                                                           ¥
    const Type& GetTypeName() const override {                  ¥
        return SGetTypeName();                                  ¥
    }                                                           ¥
                                                                ¥
protected:                                                      ¥
    static BOOL SEqualType(const Type& aTypeName) {             ¥
        return  SGetTypeName() == aTypeName;                    ¥
    }                                                           ¥
private:
```

マクロなどを使って継承関係を取得しています

GetComponentでは、**継承関係にあるコンポーネント**も取得できます

将来的には**外部のファイル**から、ゲームに登場するオブジェクトを定義できるようにしたいと考えています

OminoJumper

コンソールで動く2Dアクションゲームです
描画の高速化に取り組みました

使用言語：C++　　制作人数：1人
制作期間：1カ月　　制作時期：1年次

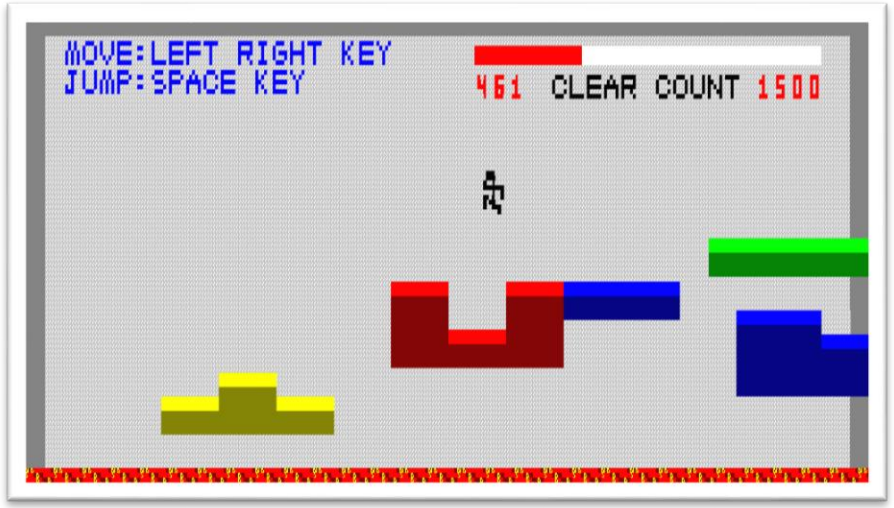
コンソール画面に描画しています

コマンドプロンプトのような**コンソール画面**に描画しています

1年次の学校のコンテストでは、コンソール画面を使う必要がありました

その**制限の中で、高い表現を可能にする**方法を実現しました

これが



これに

どのように描画しているのか

フォントサイズを変更し、**解像度が高く**なるようにしています
カラーバッファを操作することで、文字色と背景色を変えています

問題点と対応

描画がちらつく

原因：ゲーム中、即時更新をしていたので、描画が完了するまでにタイムラグがありました
対応：ダブルバッファを実装して、フレームの最後に一気に描画するようにしました

描画が遅い

原因：コンソールへの出力が遅いです。さらに、フォントサイズを小さくすると文字数が多くなり、0.1fpsしか出ませんでした
対応：直接コンソールのメモリを設定する関数を使用しました。また、描画が必要な部分だけ更新する処理を実装しました

ゲームについて

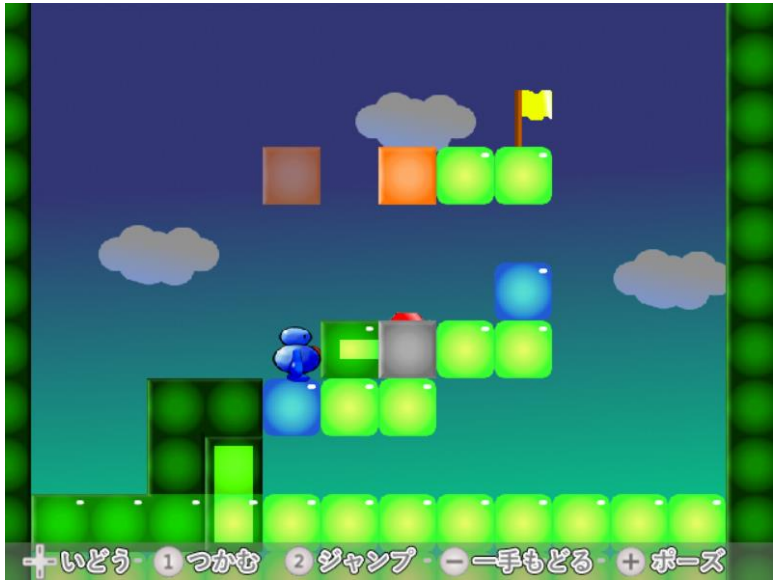
死亡時に**10万個**の血のパーティクルが飛び、移動するブロックに付着します

また、ステージは全て**テキストファイルで定義**されていて、ステージ数、
ブロックのスピード、ブロックの形などを変更することが可能です

プルぷるプッシュ！！

Wiiと**Windows**で動く2Dのパズルゲームです

使用言語：C++　　制作人数：6人（プログラムは私1人が担当）
制作期間：1カ月　　制作時期：2年次



WiiとWindowsのマルチプラットフォームです

スプライト描画、サウンド、入力を対応させました
ゲームのコードを共通化させることができます

```
//共通部分のテクスチャ
image->SetFileName(kTexture_White, "./Data/Image/Union/white.tga");
image->SetFileName(kTexture_Black, "./Data/Image/Union/black.tga");

//タイトル
image->SetFileName(kTexture_TitleLogo, "./Data/Image/Game/titleLogo.tga");
image->SetFileName(kTexture_PushStart, "./Data/Image/Game/pushStart.tga");
```

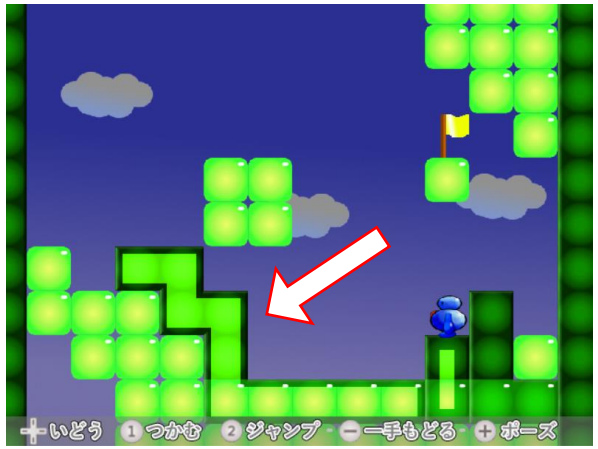
Windowsでの、テクスチャデータの設定

```
//ワープブロックの描画
void WarpBlock::Draw(GameData& data) {

    //描画する四角形を計算する
    Vector2D vecLoc = Stage::IntVecToVector2D(loc);
    Rectangle drawRect;
    drawRect.loc = vecLoc;
    drawRect.large = kWarpDrawLarge * scale;
    drawRect.rotate = -rotate;

    data.image->Set(kTexture_Warp); //テクスチャの設定
    data.camera->Draw(data, drawRect, kDrawTypeStage);  //描画
}
```

ゲーム中からは、リソースはenumで参照します

ゲームについて

UV座標を変更したスプライトを使用することで、
硬い状態から軟らかい状態に戻っていくブロックを表現しました



ブロックの表現

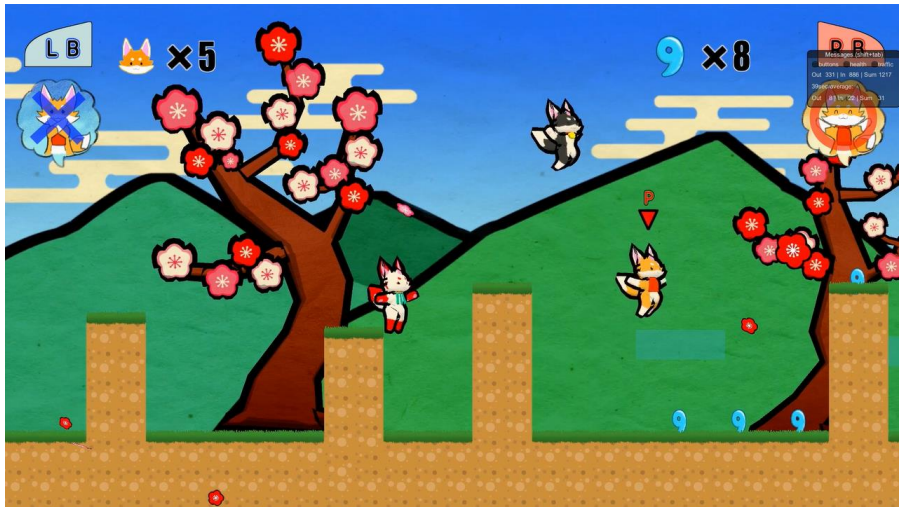
3種類のギミックや、1手戻る機能(最大100回連続で戻れる)を実装しました

チームの皆にデバッグをしてもらったおかげで、
他の生徒に遊んでいただく発表会では**バグが出ません**でした

きつねっとわーく

3人のプレイヤーが協力して進む、通信2Dアクションゲームです
全体の設計、ギミックとアイテムの実装、通信の補助などを行いました

使用言語：Unity + C#　　制作人数：8人（内プログラマ4人）
制作期間：1カ月　　制作時期：3年次



WordSearcher

語感の似ている単語を検索することができるGUIアプリケーションです
自分で音のルールを作成し、検索に使用することができます

使用言語：Python3 + PyQt　　制作人数：1人
制作期間：1カ月　　制作時期：2年次