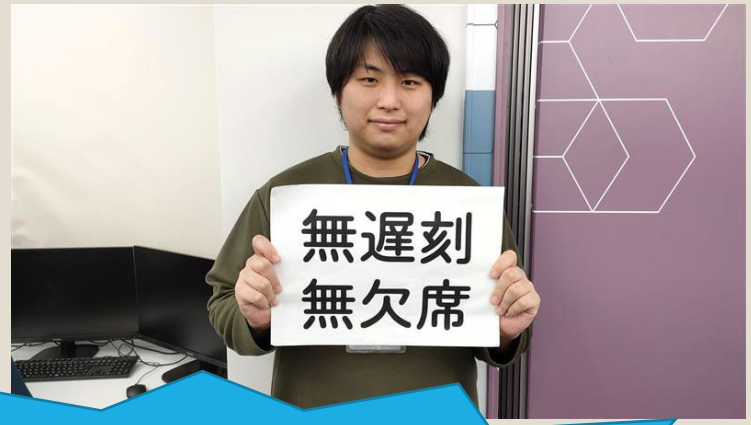
お米大好きプログラマー

- **名前**：河村 晃希（かわむら こうき）
- **学校**：福岡情報IT・クリエイター専門学校
  3年制 クリエイター学科　ゲームクリエイターコース
- **所属サークル**　：ゲーム制作部「ふゲだく！」
  役職：副部長 兼 会計
- **スキル**：C（3年）
  C++（2年半）
  C#（2年半）
  Unity（2年半）
  GitHub（2年半）
  Photoshop（2年半）
  Blender（半年）
  Maya（半年）

自己PR

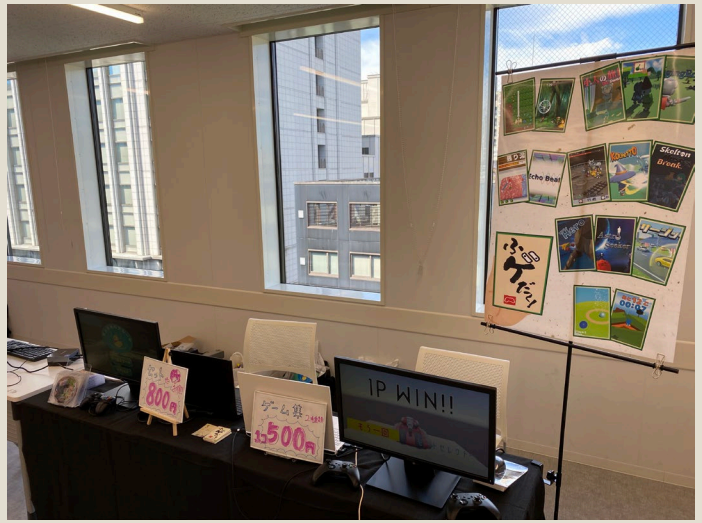
隙間時間に
Photoshopで作った
お米風プロフィール画像



毎日休まず
頑張ってます！！

ゲーム制作サークル「ふゲだく！」での活動

↓実際に遊んでもらっている様子



↑展示・販売ブースのセッティング

就活作品

最終作品

作品名　：**Tactical Power**
ジャンル：ダンジョン攻略型バトルアクション
開発環境：VisualStudio2022 C++
対応機種：Windows
制作期間：約５ヶ月
制作人数：1人
　担当　：モデル、サウンド、一部エフェクト以外

作品のGitHubリンク
https://github.com/kouki2564/Production_Storage.git
解説動画リンク
https://youtu.be/KuOo1NvDoxo

ゲーム概要

フロアにいる敵を倒してLvを上げ、
最奥のフロアに立ち構えるボスを倒せ！

面白さ①

TACTICAL POWER

適した武器を**使い分け**

最適解で敵倒せ！！

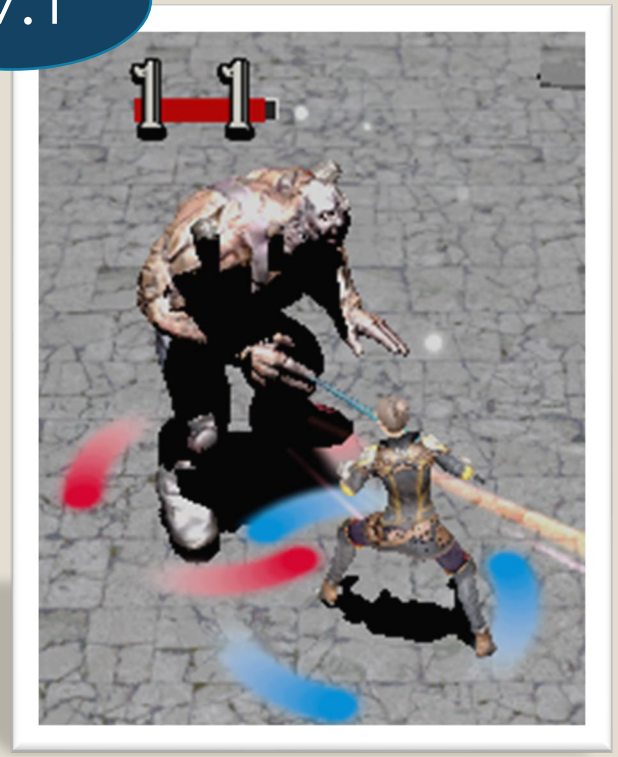
面白さ②

TACTICAL POWER

Lvを上げて**快適攻略！！**

目指せ**最速攻略！！**

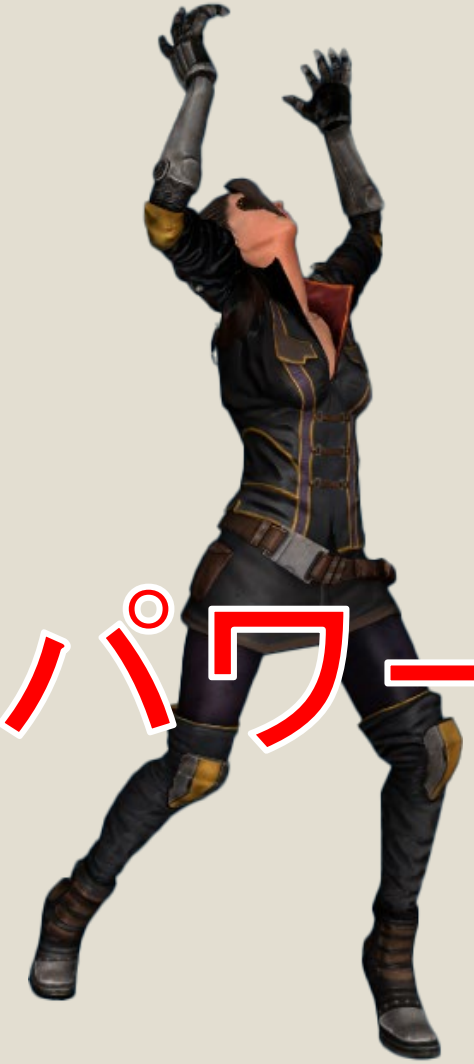
Lv.1



Lv.10

最後に

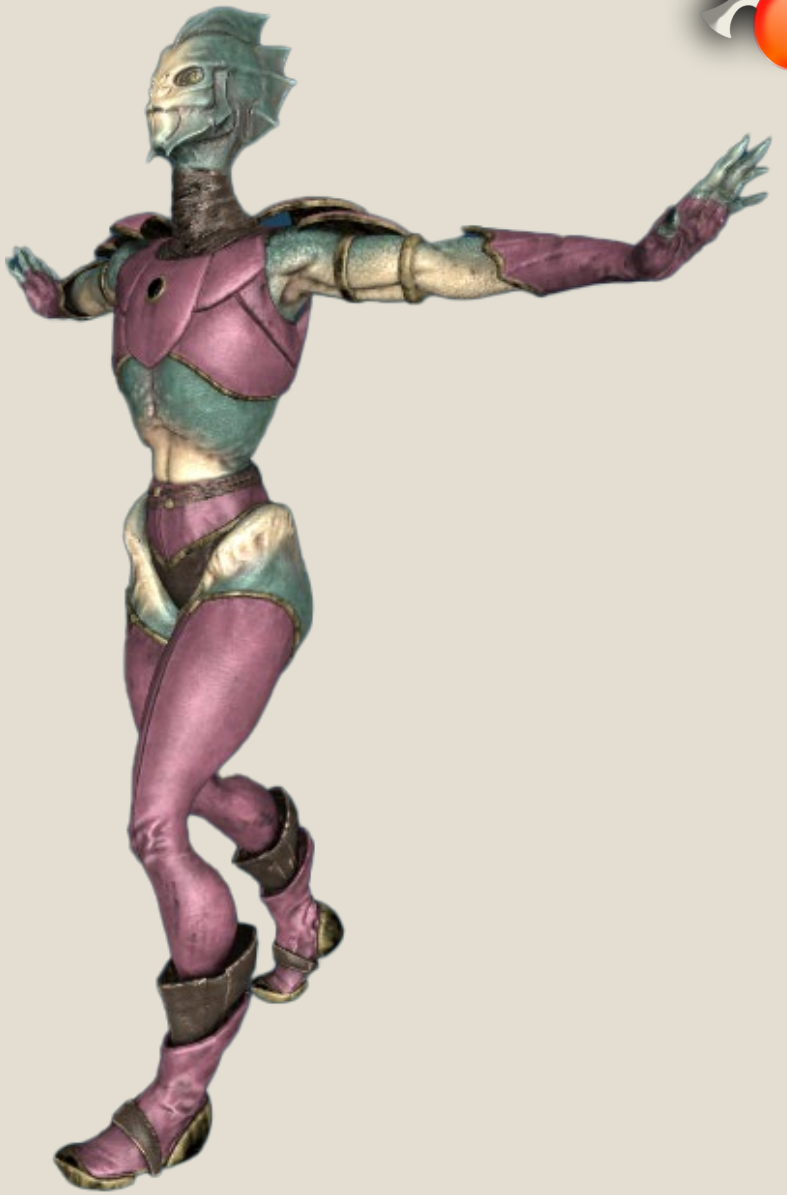
一手が勝敗を決める。
絞り出せ、最後の一撃のために…



――タクティカルパワー！

TACTICAL POWER

技術紹介

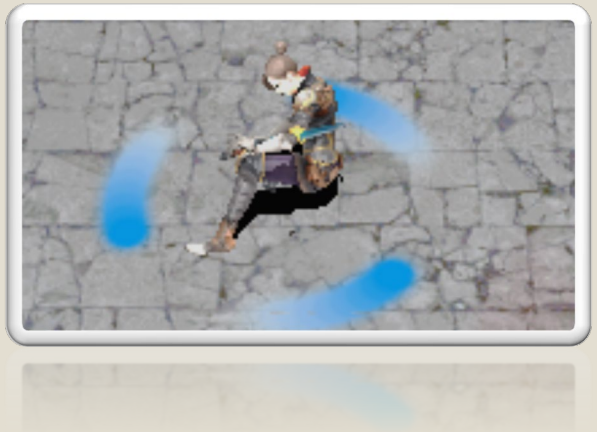


アピールポイント① ステートパターンの活用

StateBaseクラスを作り、
継承先の各**State**クラスにおいて
「その状態からどの行動に移ることが可能か」
という流れとして設計

実際の行動については
各オブジェクトに関数として持たせて実装
　改善点：各行動も**State**クラスに持たせてもよかった。

動作の管理が
しやすい！！



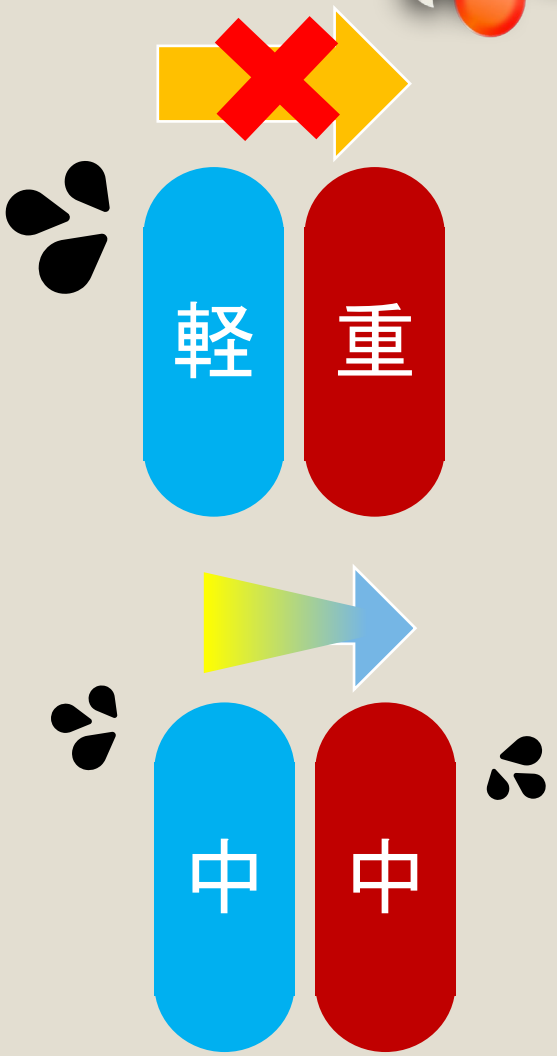
コード：Data/Scripts/State

```
enum class State
{
    MOVE,
    ATTACK,
    DODGE,
    ADJUST,
    KNOCK,
    JUMP,
    STAN,
    DEATH
};

class StateBase
{
public:
    StateBase(){}
    ~StateBase(){}

    State GetState() { return state; }

    bool GetIsOnDamage() { return isOnDamage; }
    bool GetIsMove() { return isMove; }
    bool GetIsMoveDir() { return isMoveDir; }
    bool GetIsKnockBack() { return isKnockBack; }
    bool GetIsAttack() { return isAttack; }
    bool GetIsDodge() { return isDodge; }
    bool GetIsJump() { return isJump; }

protected:

    /* 各状態移行の許可項目 */
    // ダメージをうけるか
    bool isOnDamage = false;
    // 動けるか
    bool isMove = false;
    // 動作方向を変えられるか
    bool isMoveDir = false;
    // ノックバックを受けるか
    bool isKnockBack = false;
    // 攻撃に移れるか
    bool isAttack = false;
    // 回避できるか
    bool isDodge = false;
    // ジャンプできるか
    bool isJump = false;

    State state = State::MOVE;

};
```

アピールポイント② 当たり判定

当たり判定のためのColliderは
[球],[カプセル],[マップ] を実装

さらに各Colliderに
[重],[中],[軽],[貫通] の４段階で重さを設定可能
重いと押せない、
軽いとそのままのベクトルで押し出し、
同じ重さだと重さを感じるような押し出しを実装



アピールポイント③

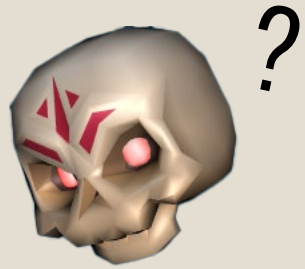
カメラ動作

カメラの視点を３視点実装

TPS視点　　：視点座標を中心に本体座標を軸回転、
　　ターゲットロック機能の実装

クォータービュー視点　：カメラ方向は固定して
　　視点座標を追尾するように本体座標を移動

ハイアングル視点　　：視点座標の上に本体座標を置いて見下ろす



その敵、
上から見るか?
横から見るか?

アピールポイント④　攻撃手段と敵の多様性

プレイヤーの攻撃は３種類
剣　　：３段コンボが出来て、空中攻撃も可能！
斧　　：ダメージが高く、スタン効果！
魔法　：溜め時間は長いが、うまく当てると大ダメージ！

敵は４種類＋ボス
素早い敵　：素早い！
重い敵　　：ノックバック無効！
魔法を使う敵：プレイヤーに魔法を発射！すぐ逃げる！
浮遊敵　　：プレイヤーに弾を発射！浮いてる！
ボス　　：強い！デカい！そして強い！

作品総括

TACTICAL POWER

より良い戦う操作感が作れた！！

だがしかし…

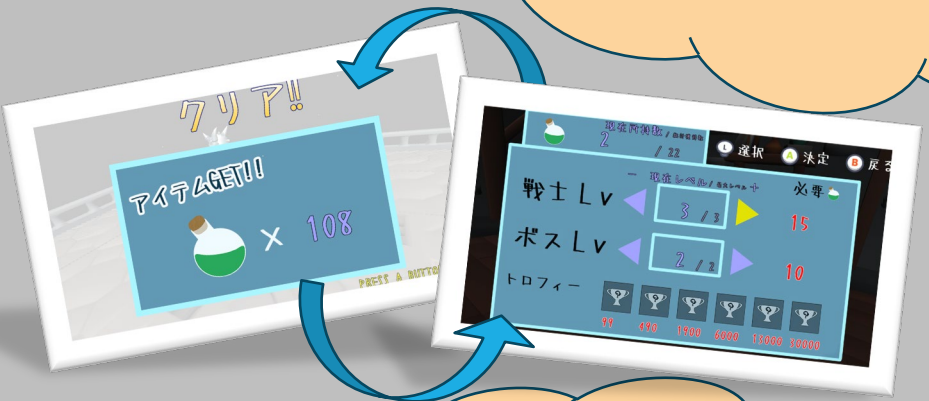
見た目がダメ…！

エフェクトをもっとつけて、
ライティングも調整して
見た目を良くして行きます！！

その他作品

Quaternionクラスの実装
移動ベクトルを入れることで
平行移動もできるように実装

```
class Quaternion
{
public:
    Quaternion()
    {
        Qu.w = 1;
        Qu.x = 0;
        Qu.y = 0;
        Qu.z = 0;
        vec = VGet(0, 0, 0);
    };
    virtual ~Quaternion() {}

private:
    struct Q
    {
        float w;
        float x;
        float y;
        float z;

        Q operator * (const Q& _q) const
        {
            Q tempQ;

            //クオータニオンの掛け算//
            //公式通りです。
            tempQ.w = w * _q.w - x * _q.x - y * _q.y - z * _q.z;//実部
            tempQ.x = w * _q.x + x * _q.w + y * _q.z - z * _q.y;//虚部x
            tempQ.y = w * _q.y + y * _q.w + z * _q.x - x * _q.z;//虚部y
            tempQ.z = w * _q.z + z * _q.w + x * _q.y - y * _q.x;//虚部z

            return tempQ;
        }
    };

    Q Qu;
    VECTOR vec;

public:
    /// <summary>
    /// 移動量の設定
    /// </summary>
    /// <param name="_angle">1フレーム当たりの角度（ ラジアン値 ）</param>
    /// <param name="_axis">回転の軸ベクトル</param>
    /// <param name="_moveVec">平行移動ベクトル</param>
    void SetMove(float& _angle, VECTOR& _axis, VECTOR& _moveVec)
    {
        Qu.w = cos(_angle / 2.0f);//実部
        Qu.x = _axis.x * sin(_angle / 2.0f);
        Qu.y = _axis.y * sin(_angle / 2.0f);
        Qu.z = _axis.z * sin(_angle / 2.0f);
        vec = _moveVec;
    }

    /// <summary>
    /// クォータニオンによる実移動
    /// </summary>
    /// <param name="_rotPoint">回転する中心の座標</param>
    /// <param name="_pos">回転するものの座標</param>
    /// <returns>回転後の座標</returns>
    VECTOR Move(VECTOR& _rotPoint, VECTOR& _pos)
    {
        Q qPos, qInv;
        VECTOR vPos;

        //3次元座標をクオータニオンに変換
        qPos.w = 1.0f;
        qPos.x = _pos.x - _rotPoint.x;
        qPos.y = _pos.y - _rotPoint.y;
        qPos.z = _pos.z - _rotPoint.z;

        //回転クォータニオンのインバースの作成
        //逆クォータニオンを出すのは大変なので、
        //3次元だと同じ値になる共役クオータニオンで作成(虚部だけマイナス反転)
        qInv.w = Qu.w;
        qInv.x = -Qu.x;
        qInv.y = -Qu.y;
        qInv.z = -Qu.z;

        //回転後のクオータニオンの作成
        qPos = Qu * qPos * qInv;

        //3次元座標に戻す
        vPos.x = qPos.x + _rotPoint.x;
        vPos.y = qPos.y + _rotPoint.y;
        vPos.z = qPos.z + _rotPoint.z;

        // 回転後に移動
        vPos.x += vec.x;
        vPos.y += vec.y;
        vPos.z += vec.z;

        return vPos;
    }
};
```



クリアで手に入るアイテムを用いて
ゲーム難易度を自分で調整可能
お遊び要素としてトロフィーも実装

2024年8月~ 10月
１マップ型
3Dバトルアクション

Ordeal Maze

1vs1でボスバトルを行い、
強化して自身で難易度を調整しつつ、繰り返しバトルを行うゲーム。

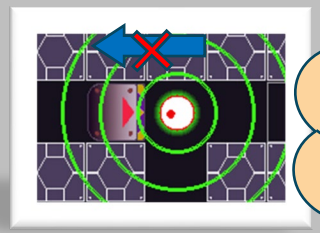
この作品からカメラの回転にはQuaternionを採用。

また、Effekseerというエフェクトの制作ツール、ライブラリを使用してより臨場感を付与。

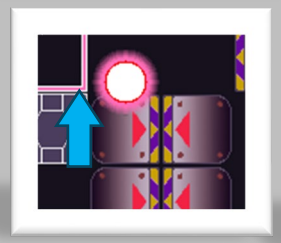
エフェクトやライトによるゲームの重さの課題を抱え、作っていて自身の勉強となった作品。

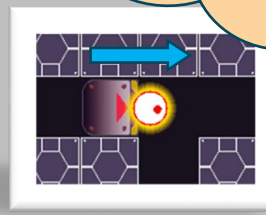
バイナリデータの
読み書きを行って、
回数記録の
セーブ＆ロードを実装

```cpp
// セーブ
void File::Save()
{
    // ファイルの枠作成
    FILE* Writefp = nullptr;

    // == 0  でファイル存在
    if (fopen_s(&Writefp, "TimeData", "wb") == 0)
    {
        for (int i = 0; i < 5; i++)
        {
            if (m_timeRank[i] > 0)
            {
                fwrite(&m_timeRank[i], sizeof(int), 1, Writefp);
            }
        }

        fclose(Writefp);
    }
    else
    {
        assert(false);
        return;
    }
}

// ロード
void File::Load()
{
    FILE* Readfp = nullptr;
    // == 0  でファイル存在
    if (fopen_s(&Readfp, "TimeData", "rb") == 0)
    {
        // 配列個数分のintを入れる
        for (int i = 0; i < 5; i++)
        {
            fread(&m_timeRank[i], sizeof(int), 1, Readfp);
        }
        fclose(Readfp);
    }
    // ファイルが無い(初プレイ)時の初期データ作成
    else
    {
        CreateData();
        Save();
        Load();
        return;
    }
}
```



ギミックとして移動方向制限床を実装
プレイヤーの進行方向から通過可能かを判定





```cpp
namespace
{
    // マップチップの情報
    constexpr int kChipWidth = 48;
    constexpr int kChipHeight = 48;

    // チップのマス数
    constexpr int kChipNumX = 19;
    constexpr int kChipNumY = 14;


    ////////////////////
    // マップチップの配置情報 //
    ////////////////////////////////////////////////////////
    //  0 : 空白                                          //
    //  1 : 壁                                            //
    //  2 : プレイヤー                                    //
    //  3 : 色変更床 (30:赤 31:緑 32:青 33:黄 (34:白))    //
    //  4 : 着色床 (40:赤 41:緑 42:青 43:黄 (44:白))      //
    //  5 : 一方通行床 (50:上, 51:左, 52:下, 53:右)       //
    //  6 : ゴール                                        //
    ////////////////////////////////////////////////////////
    constexpr int kStage[Game::kStageNum][kChipNumY][kChipNumX] =
    {
        // ステージ0(サンプルステージ)(本編に実装ましない)★★★★
        {
            {1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 },
            {1 ,30,0 ,0 ,0 ,51,0 ,0 ,40,1 ,41,0 ,0 ,53,0 ,0 ,0 ,31,1 },
            {1 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,1 },
            {1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 },
            {1 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,1 },
            {1 ,43,0 ,0 ,0 ,50,0 ,0 ,1 ,0 ,1 ,0 ,0 ,50,0 ,0 ,0 ,42,1 },
            {1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,2 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,1 ,1 },
            {1 ,41,0 ,0 ,0 ,52,0 ,0 ,1 ,0 ,1 ,0 ,0 ,52,0 ,0 ,0 ,40,1 },
            {1 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,1 },
            {1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 },
            {1 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,1 },
            {1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,6 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 },
            {1 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,1 },
            {1 ,32,0 ,0 ,0 ,51,0 ,0 ,42,1 ,43,0 ,0 ,53,0 ,0 ,0 ,33,1 },
            {1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 },
        },
        // ステージ1(色変え)☆
        {
            {0 ,0 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,40,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,1 ,0 ,1 ,1 ,1 ,1 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,1 ,0 ,1 ,0 ,0 ,1 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,1 ,0 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,0 ,0 },
            {0 ,0 ,0 ,1 ,32,0 ,0 ,0 ,40,0 ,0 ,0 ,0 ,42,1 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,0 ,1 ,1 ,1 ,1 ,1 ,0 ,0 ,0 },
            {0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,30,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,6 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,2 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
            {0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 },
        },
```

マップチップの配置は
namespace内にて数字で割り当てて管理、参照
当時はcsvファイルを使う発想が無く…

2023年10月~2024年2月
1画面型
2Dパズルアクション

Glow Ruler

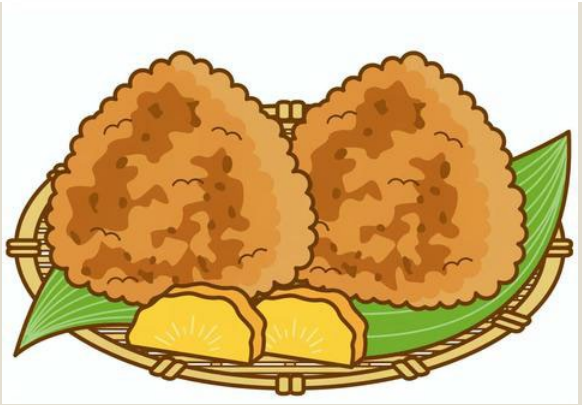
円状の装置で色を取得し、対応する色の床を光らせる。全て光らせたらゴールを目指し、その移動回数を測るゲーム。

「失敗やゲームオーバーは作らない」「繰り返し挑戦してもらう」
という考えを元に制作。

データファイルの読み込み、書き込みを実装。この作品では主に移動回数の記録として使用。

背景や横の模様などは、アニメーション付きの画像などが見つからず、自力で描画。

今後の目標



オブジェクト周りの動作制御なら
とりあえず任せろ！！

そんな自分になりたいです。

これまでのゲーム制作を振り返ると、プレイヤーや敵の動作を調整したり作り込んだりしている時間が、最も楽しく感じられました。

これからも、このような動作処理周りのスキルをさらに磨いていきたいと思います。