つたえる

しゅうしょくさくひん

HAL

ゲーム４年制学科３年
西口 煌大

自己紹介

西口　煌大
（ニシグチ コウタ）

生年月日	２００４年１１月１９
趣味	カラオケ・卓球
制作実績	コンソール x 1　DirectX 11 x 2 Unity x 2　UnrealEngine5 x 1

どんなプログラマ？

遊んだ時の**感触**を意識する

体験志向プログラマ

クリエイターとしての見解

1 体験志向である事の重要性

ユーザーを最優先に考えられる
「正しさ」より「伝わり方」を判断基準にできる

2 違和感の活用

体験を阻害している要素は削減
印象を強めるための調整にもつかえる

就職作品について

目標	①体験の流れが途切れないゲーム ②1/31までに完成させる
作品内容	メイドインワリオの再現
使用言語	C++20/DirectX11
制作期間	1ヶ月（個人制作）

メイドインワリオの理由

1. 目標に近いゲーム
2. テンポ / ゲームループ設計が明快

体験を基準にした実装の優先度

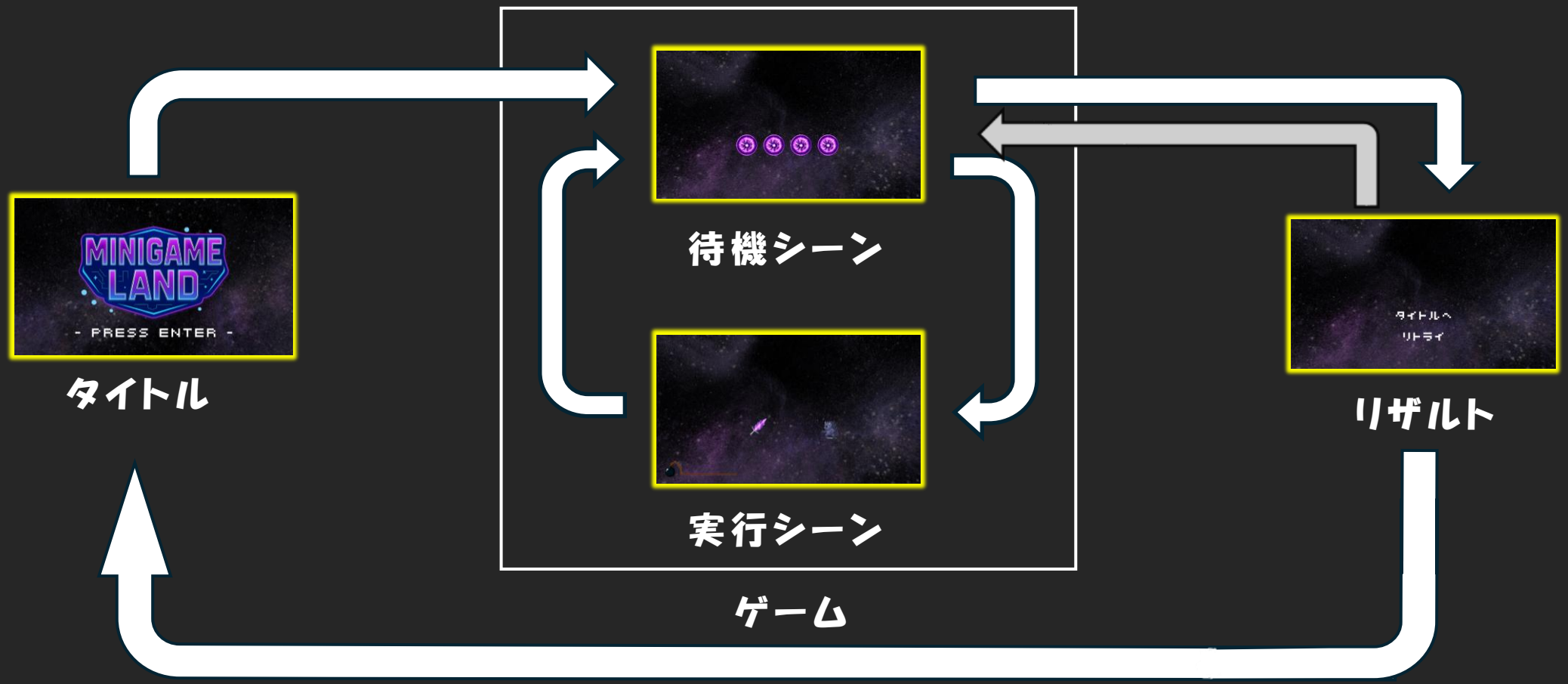
ゲームループ完成

ゲーム成立の条件

リズム感の実装

体験を損なわないため

ゲームフロー


MINIGAME
LAND
- PRESS ENTER -
タイトル
待機シーン
実行シーン
ゲーム
タイトルへ
リトライ
リザルト

違和感を抑えるための設計

リズム感

① 待機→実行の遷移を3小節固定

② 早期クリア時の体験破綻を防止

リズムをとるために

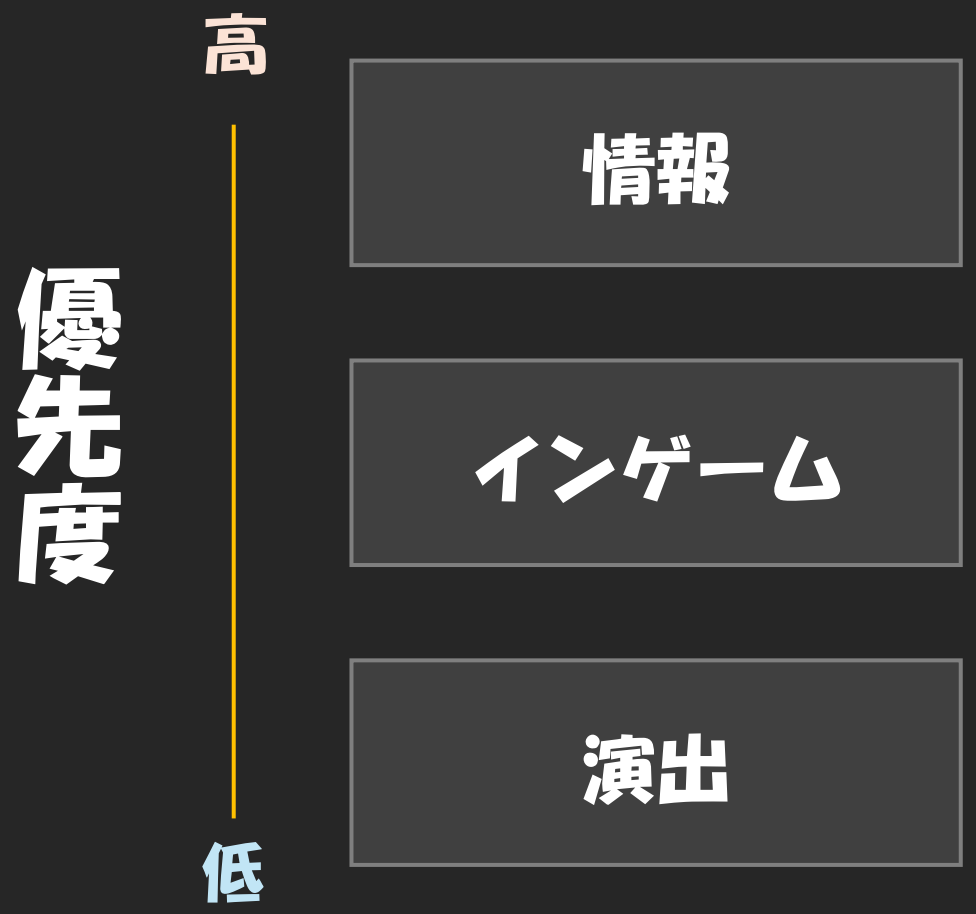
```cpp
struct RhythmBeatConst
{
    // 入力値
    float bpm = 120.0f;         // BPM (Beats Per Minute)
    int   beatUnit = 4;         // 拍子の分母 (4 = 4分音符, 8 = 8分音符, etc.)
    int   ticksPerBeat = 16;    // 1拍を何分割するか

    // 計算結果
    float secondsPerBeat = 0.0f; // 1拍の長さ (秒)
    float secondsPerBar  = 0.0f; // 1小節の長さ (秒)
    float secondsPerTick = 0.0f; // 1Tickの長さ (秒)
    float ticksPerSecond = 0.0f; // 1秒あたりに進むTick数

    // Setup (セットアップ) :
    //     設定をまとめて反映する関数名によく使われる
    void Setup(float bpm_, int beatsPerBar_ = 4, int ticksPerBeat_ = 16)
    {
        bpm = bpm_;
        beatUnit = beatsPerBar_;
        ticksPerBeat = ticksPerBeat_;

        // 1拍の長さ (秒)  = 60 / BPM
        secondsPerBeat = 60.0f / bpm;

        // 1小節の長さ (秒)  = 1拍 * 拍数
        secondsPerBar = secondsPerBeat * static_cast<float>(beatUnit);

        // 1Tickの長さ (秒)  = 1拍 / Tick数
        secondsPerTick = secondsPerBeat / static_cast<float>(ticksPerBeat);

        // 1秒あたりのTick数 = 1 / secondsPerTick
        ticksPerSecond = 1.0f / secondsPerTick;
    }
};
```

FPS、BPM、拍、から **拍の間隔の値を算出**

```cpp
class RhythmBeat
{
private:
    RhythmBeatConst m_Beat{};
    float           m_TickCounter = 0.0f;       // 経過時間の蓄積
    int             m_TickIndex   = 0;          // 現在のTick数
    int             m_Advance     = 0;          // 現在の拍子インデックス
public:
    RhythmBeat() = default;
    // 初期化
    void Initialize(const RhythmBeatConst& config);
    // 更新
    int Update(float deltaTime);

    // 現在のTickを取得
    int GetTotalTick() const
    {
        return m_TickIndex;
    }

    // 現在の拍子インデックスを取得
    // 今が何泊目かを返す
    int GetBeatIndex() const
    {
        return m_TickIndex / m_Beat.ticksPerBeat;
    }

    // 現在の拍子内のTick位置を取得
    // 今の拍子の中で何Tick目かを返す
    int GetTickInBeat() const
    {
        return m_TickIndex % m_Beat.ticksPerBeat;
    }

    const RhythmBeatConst& GetBeatConst()const
    {
        return m_Beat;
    }

    int GetAdvance() const {
        return m_Advance;
    }
};
```

現在の拍数を監視・更新

実装の優先順位



ゲームの成立

遊びの成立

完成度の向上

現状と残タスク

現状

ひととおり遊べる状態

「タイトル」-「ゲーム」-「リザルト」

残タスク

情報	インゲーム	演出
スコア表示	敵挙動	パーティクル
UIの調整	難易度調整	スピードアップ

完成の見通し

完成条件を満たす構造は実装できており

残っているのは体験の完成度を高める要素です

ご覧いただきありがとうございました。