就職作品資料

ＨＡＬ大阪３年　西口煌大

目次

・作品概要と制作背景
・技術的な挑戦と工夫
・今後の制作予定

作品概要と制作背景

作品概要と制作背景

この作品は出されたお題を次々クリアしていく2Dのミニゲームアクションです。
代表的な作品である「メイドインワリオ」の分析を行い、技術的な挑戦と独自的な工夫を凝らし、没入感の高いゲーム体験を目指しました。

制作期間

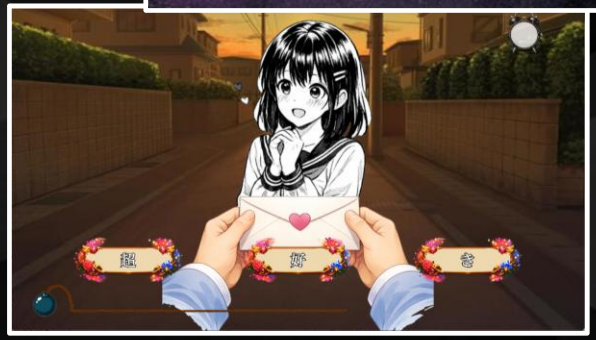
2025年10月 ～ 2026年1月 約3ヶ月

基本操作

左クリック　：アクション
マウス移動：移動

制作の狙い

反射で遊ばせる没入感
保守性の高いアセット管理
拡張に強いシーンフロー設計



技術的な挑戦と工夫

技術的な挑戦と工夫

システム設計

- 債務分離アーキテクチャ
- Sceneスタックによるフロー管理
- リソース管理の明確な分離
- 型安全なオブジェクト生成

パフォーマンス最適化

- DX11描画基盤自前構築
- オフスクリーン描画パス整理
- リソースキャッシュ管理
- フレーム非依存の時間制御

UI/UX

- デバッグUI
- UI合成設計
- 入力統合
- テンポ制御

描画基盤/表示制御

- 品質管理・保守性の作法

システム設計

技術的な挑戦と工夫

システム設計

➢ 債務分離アーキテクチャ

Gameクラスに「初期化・更新・描画・解放」の実行順を集約し、Scene／Objectは処理内容の実装に集中させました。
これにより、**フローの見通し**と**拡張時の事故防止**を狙っています。

➢ Sceneスタックによるフロー管理

メイドインワリオは「ミニゲームをするシーン」と「お題を待機するシーン」を高い頻度で行き来します。
ミニゲームと待機のシーンが**多対一の関係**であり、共通の待機シーンを軸に、複数のミニゲームを差し替えて管理できる構成が必要だと考え実装しました。

技術的な挑戦と工夫

システム設計

➢ リソース管理の明確な分離

リソース管理をマネージャーに集約し、Object は「使うだけ」に制限しました。
これにより、生成漏れ・解放漏れを構造的に防いでいます。

➢ 型安全なオブジェクト生成

すべての描画オブジェクトは Object を基底とし、必要なリソースはマネージャーから取得します。
生成経路を固定することで、安全性を担保しています。

パフォーマンス最適化

技術的な挑戦と工夫

パフォーマンス最適化

➢ DX11描画基盤自前構築

「なぜ描けるのか」を説明できる状態を目標に、DX11 を用いて描画基盤を一から構築しました。
各ステージの責務を分解して実装しています。

➢ オフスクリーン描画パス整理

オフスクリーン描画を前提とすることで、描画結果をテクスチャとして扱えるようにしました。
デバッグ UI での可視化を想定した設計です。

技術的な挑戦と工夫

パフォーマンス最適化

➢リソースキャッシュ管理

Mesh・Texture・Shader をマネージャでキャッシュ管理し、**生成・共有・寿命管理**を Object から**分離**しました。
実運用を想定したリソース管理基盤です。

➢フレーム非依存の時間制御

実際にプレイ検証を行う中で、ゲーム体験の質を左右する最重要要素が**リズムの安定性**と**再現性**であると判断しました。そこで、まず最小単位となる Tick 情報を時間管理の基盤として定義・集約し、その上にリズム同期用の構造体および関数群を設計・実装しています。

UI/UX

技術的な挑戦と工夫

UI/UX

➢ Debug UI

ImGui によるデバッグUIで、**内部状態**と**描画結果**を可視化し、
調整・検証・原因切り分けの速度を上げています。

➢ UI合成設計

ゲーム画面はオフスクリーンに描画し、UI／演出は後段で合成しています。
描画順を固定することで、**UI追加**や**遷移演出**を**破綻させない構成**にしました。

技術的な挑戦と工夫

UI/UX

➢ 入力統合

キーボード／マウス／ゲームパッド入力を同一インターフェースに統合し、
ゲーム側は**入力デバイス差**を**意識せず**に実装できるようにしました。

➢ テンポ制御

Delta Time に基づくフレーム非依存の時間管理に、BPM/Beat/Tickの概念を導入し、
「テンポに同期した**状態遷移**と**演出制御**」を実現しました。

技術的な挑戦と工夫

UI/UX ： テンポ制御

while分のループ回数　＝　そのフレームで**経過した拍数**として算出します。
更新毎に行い、インゲームのループ、イベントに利用していきます。

```
// 経過秒数をTickに変換してカウントアップ
m_TickCounter += m_Beat.ticksPerSecond * deltaTime;

// 1Tick以上進んでいたらTick数を進める
while (m_TickCounter >= 1.0f)
```

設定した上限の拍数を基に「**残り何拍**」「**経過拍数**」の値を返す関数を作成しました。
値を参照時「どこを基準にするか」を**直感的にコーディング**できるようにしました。

```
// 残りの拍数を取得
return m_BeatRest ;

// 経過した拍数を取得
return m_GameBeats - m_BeatRest ;  // 上限の拍数 - 残りの拍数 で算出
```

描画基盤・表示制御

技術的な挑戦と工夫

描画基盤・表示制御

➢品質管理・保守性の作法

HLSLは命名と責務を統一し、C++側の定数バッファと対応づけています。

シェーダ運用の**破綻**を防ぐ設計です。

同一フレームワークで扱えるよう、基盤側を先に設計しており、VS/PS/GS/CS まで対応させています。

シェーダーステージを「命名規約」によって自動判別し、**描画基盤側**で**責務**を**完結**させている設計

```
return (m_HlslName.rfind( "VS_", 0) == 0) ? new VertexShader()   : nullptr;
return (m_HlslName.rfind( "PS_", 0) == 0) ? new PixelShader()    : nullptr;
return (m_HlslName.rfind( "GS_", 0) == 0) ? new GeometryShader() : nullptr;
return (m_HlslName.rfind( "CS_", 0) == 0) ? new ComputeShader()  : nullptr;
```

今後の制作予定

今後の制作予定

・作成したテンポ基盤を用いたミニゲームの追加

2月10日を最終期限とし、
既存のテンポ制御基盤を活かしたミニゲームを追加します。
ゲームとしての完成度向上を目的としています。

ご覧いただきありがとうございました。