Technical Design Document (TDD)

Game Title: The Record of Lies: Crown's Formation
Version: v0.1
Date: 23/02/2025
Author(s): Esset Axis

---

1. Overview

- **Purpose**: This document defines the technical specifications, architecture, and development guidelines for The Record of Lies: Crown's Formation, a 2D RPG game developed as part of a university project. Its purpose is to provide a clear and structured guide for developers, designers, artists, and testers involved in the project, ensuring an efficient and consistent development process.
- **Scope**: The game is built on a modular architecture with a custom C++ engine. This document covers key aspects such as rendering, audio, physics, artificial intelligence, optimization, and development tools. It also includes details on the workflow, external tool integration, and collaboration strategies for game production.
- **Audience**: This document is intended for:
    - **Engineers and programmers:** To implement the game's architecture and core systems.
    - **Game designers:** To understand the technical limitations and capabilities of the engine.
    - **Artists:** To adapt their assets to the required formats and engine specifications.

2. Target Platform

2.1 Hardware Baseline

- **Platform(s)**: PC
- **Minimum/Recommended Specs**:
    - **CPU**: Intel Core i3-350M
    - **GPU**: Integrated Graphics
    - **RAM**: 1GB minimum, 3GB recommended
    - **Storage**: 2GB
- **Resolution & Performance**:
    - **Target FPS**: 60 FPS
    - **Resolution**: 1260x720

2.2 Cross-Platform Considerations

- Keyboard, controller.

3. Architecture

3.1 High-Level Diagram

3.2 Core Systems & Modules

- **Game Engine**: Custom C++ Engine.
- **Subsystems**:
    - Rendering.
    - Audio.
    - Input.
    - Assets.
    - Localization.
    - Physics.
    - Window.
- **Class Structure**:
    - Key classes:
        - Engine.
        - Module.
        - Interfaces:

        - IRendereable.
        - IUpdateable.
        - IActivable.
        - ICleaneable.
        - IInitializable.
    - Relationships [UML diagram].

3.3 External Libraries & SDKs

- **Physics**: Box2D.
- **AI**: A* Pathfinding.
- **Graphics**: SDL2.
- **Other**: PugiXML, NlohmannJson, Tracy.

4. Technology Choices

4.1 Programming Language

- **Primary**: C++.
- **Scripting**: Dialogue Node based.

4.2 Codebase Structure

- Folder hierarchy:
    - Assets: `Assets/`
    - Code: `src/`
- Naming conventions:
    - `PascalCase` for classes.
    - `camelCase` for variables.
    - `_camelCase` for constructorVariables.

4.3 Programming Style

- Code standards: C++ Core Guidelines.
- Tools: Visual Studio.

5. Physics & Collision Detection

- **Physics Engine**: Box2D.
- **Collision Layers**: Player, Npc, Obstacle, Ground, Interactive, Items
- **Optimization**: None.

6. AI & Pathfinding

- **Pathfinding**: A*.
- **AI Behaviors**: Finite State Machines (FSM).
- **NPC Decision-Making**: Behavior Trees.

7. Optimization Strategies

- **Performance**:
    - GPU: Occlusion culling, texture atlases.
    - CPU: Async methods.
- **Memory**: Object pooling.

8. Tooling & Workflow

- **Engines & IDEs**: Visual Studio.
- **Asset Pipeline**: .wav/.ogg/.png importers.
- **CI/CD**: GitHub Actions.
- **Debugging**: Profilers (Tracy), VS debugging tools.

9. Version Control & Collaboration

- **VCS**: GitHub.
- **Branching Strategy**: GitLabFlow.
- **Collaboration**:
    - Code reviews (Pull Requests).
    - Issue tracking (Jira).

10. Delivery Process

- **Build Pipeline**:
    - QA testing stages (Vertical Slice, Alpha, Gold).
- **Deployment**:
    - Distribution platforms (Github).

11. Technical Considerations

- **Risks**: Performance bottlenecks.
- **Dependencies**: LE-Dialogue-Editor.
- **Scalability**: DLC.