Technical Design Document – Outline

# Title Page

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author(s) | Changes |
| 0.3 | 14-04-2024 | Anfer Molina | Final Preview |
|  |  |  |  |
|  |  |  |  |

# Table of Contents

# Game Summary

"Slippery Lands" is a puzzle game that challenges players to navigate through levels by sliding on ice and pushing crates to activate switches and open doors. Inspired by the ice puzzles of classic Pokémon games on the Gameboy, "Slippery Lands" introduces unique mechanics where players must strategically push crates while managing their sliding movement on ice.

# Development Environment

## Development Hardware

It was developed in two laptops, the first one a Toshiba Laptop with 16 GB of RAM, intel i7, and 512 GB SSD (RIP). The newer laptop is an Asus TUF with 16GB of RAM, i7 13th Gen, 512GB SSD, and Nvidia RTX 4050 graphics card to handle SFML.

## Programming Languages

C++ (preview – latest).

## Development Tools

Microsoft Visual Studio for C++ development, Simple and Fast Multimedia Library (SFML) for graphics and audio, Git for version control, and Aseprite.

## External Code

SFML Library (v2.6.1): Used for rendering game graphics, handling window events, and managing audio. Official site: [SFML](https://www.sfml-dev.org/).

## Game Engine

The game utilizes a 2D game engine provided by the educational course, modified to support grid-based movement and collision detection essential for handling the unique ice-sliding mechanics and interactions with crates.

# Architectural Analysis

## Classes

Describe the classes that will have to be implemented. For each class, provide:

* Its responsibilities
* How it collaborates with other classes

|  |  |  |
| --- | --- | --- |
| Class | Responsibilities | Collaborations |
| Scence\_Ice | Manages the game scene for levels with ice. It handles initialization of the level from a file, player movements, collision handling, updating game objects, rendering the scene, and playing music and sound effects. | GameEngine: Receives events and updates from the central game engine.  EntityManager: Manages all entities within the scene, including updating and rendering them.  MusicPlayer & SoundPlayer: Handles background music and sound effects.  Assets: Provides access to game assets like textures and animations. |
|  |  |  |
|  |  |  |

Present class diagrams that show the relationships between classes. Show only the most essential attributes and methods for each class.

## Behavioral Analysis

Present statecharts, flow charts (activity diagrams), sequence diagrams, etc. that model complicated behavior. If your game has actors that implement a state machine, this would be the section where you’d present the statechart.

## Game Loop

Initialization: On starting a level, the Scene\_Ice class loads level layout from a configuration file, initializes the game view, and sets up the player and obstacles.

Player Movement and Interaction:

Handles real-time input from the player to move the character in four directions.

Adjusts movement based on whether the player is on ice or snow, affecting speed and sliding behavior.

Plays corresponding animations and sound effects based on the terrain.

Collision Detection:

Regularly checks for and resolves collisions between the player, obstacles, and boundaries.

Updates game state based on interactions (e.g., player reaching the door or triggering a switch).

Rendering: Continuously updates the visual display of the game scene, drawing the background, entities, and UI elements like the timer.

Audio Management: Plays and manages background music and sound effects appropriate to player actions and game events.

# Technical Risks

List all technical risks that could make it difficult or impossible to complete the game. Examples:

* Uncertainty on how to implement a certain feature
* Uncertainty on if a certain feature can be executed fast enough in real time
* First time using a certain library

|  |  |  |
| --- | --- | --- |
| Risk | Severity | Mitigation (what is to be done to eliminate or minimize this risk) |
| Complexity in Collision Handling | Given the mixed terrain types and movable objects, ensuring robust and bug-free collision detection is challenging. | Thorough testing on all levels, especially under various scenarios where player and object interactions can be complex. |
|  |  |  |
|  |  |  |
|  |  |  |