Software Requirement Specification

Maryam Mohammadi

Kharazmi University

Fall 1403

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to outline the requirements for the development of the Tirandazi game. This document will provide a detailed description of the functional and non-functional requirements and serve as a guide for the development team to ensure the successful design, development, and deployment of the game.

1.2 Scope

This Software Requirements Specification (SRS) outlines the development of the 2048 game, a single-player sliding block puzzle game. The game will feature a 4x4 grid where players combine numbered tiles to reach the 2048 tile. The core mechanics will be developed using Python and Pygame. The game will include functionality for tile movement, combination, scoring, and a user interface for displaying the current score and game status.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS: Software Requirements Specification**
- **UI: User Interface**
- **UX: User Experience**
- **AI: Artificial Intelligence**
- **Pygame: Python library used to create 2D games**
- **FPS: Frames Per Second**
- **Python: Programming language used for game logic and development**

1.4 References

- Python Documentation
- Pygame Documentation:
- GitHub for Version Control

2. Overall Description

2.1 Product Perspective

2048 is a standalone single-player sliding block puzzle game designed to provide an engaging and challenging experience. Players combine numbered tiles on a 4x4 grid to reach the 2048

tile. The game features interactive mechanics such as scoring, tile movement, and player interactions.

2.2 Product Functions

- **Tile Movement:** Players can slide tiles up, down, left, or right using keyboard inputs.
- **Tile Combination**: Identical tiles that collide during a move will merge into a single tile with the combined value.
- **Scoring System:** Tracks the player's score, which increases when tiles merge.
- **Game Over Condition:** The game ends when no moves are possible, either due to the grid being full or no adjacent tiles having the same value.
- **Undo Functionality:** Allows players to revert to the previous game state.
- **Save and Resume**: Enables players to save their progress and resume the game later.

2.3 User Classes and Characteristics

- **Players**: Individuals who interact with the game, control the tiles, and aim to achieve the highest score.
- **Game Admin:** Manages game settings, modifies difficulty levels, and oversees content updates.

2.4 Operating Environment

- **Client Side:** The game will run on platforms that support Python and Pygame, including Windows, macOS, and Linux.
- **Server Side:** If multiplayer functionality is integrated, the game may require a backend to handle data and player interactions (though this is optional for the initial release).

2.5 Design and Implementation Constraints

- The game must be optimized for performance to ensure smooth gameplay, even with complex actions or many moving tiles.
- The game will use the Pygame library for handling graphics, sound, and input.
- The game must operate on all systems that support the latest version of Python and Pygame.

2.6 Assumptions and Dependencies

- Players will have access to the required hardware (keyboard and display) and an operating system compatible with Python.
- The game will be hosted and distributed on platforms such as itch.io or Steam (if applicable).

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

• **Game Interface**: The main game screen where the player combines numbered tiles, moves them using directional inputs, and views the score.

• **Start Menu**: A menu for starting a new game, viewing instructions, or accessing the game settings.

• **Pause Menu**: A menu that allows players to pause the game, resume, or restart.

• **Game Over Screen**: Displayed when the player loses (i.e., no moves possible), showing the final score and options to restart or exit.

3.1.2 Hardware Interfaces

• **Keyboard**: Players use the keyboard for tile movement (arrow keys).

• **Touchscreen (optional)**: If the game supports mobile, players can swipe to move tiles.

3.1.3 Software Interfaces

• **Python Interpreter**: The game will be executed using the Python runtime environment.

• **Pygame Library**: This game uses the Pygame library to handle graphics, sounds, and player inputs.

3.1.4 Communication Interfaces

• **Not Applicable**: Since this is a standalone, single-player game, there is no need for external communication protocols like HTTP/HTTPS.

3.2 Functional Requirements

3.2.1 User Authentication

• **Login**: If the game includes user accounts for saving high scores, players will need to log in via a form with username/email and password.

• **Logout**: Players must be able to log out securely from their account.

• **Registration**: New users can create an account to save progress and track scores.

3.2.2 Game Mechanics

• **Tile Movement**: The player must be able to slide the numbered tiles using the keyboard or touchscreen.

• **Tile Combination**: Identical numbered tiles should merge into a new tile when they collide during movement.

3.2.3 Scoring and Progression

• **Score Tracking**: The game tracks the player's score, increasing as tiles merge to form higher numbers.

• **Game Levels**: The game consists of a single level, with difficulty increasing as the grid fills with higher-numbered tiles.

3.2.4 Lives and Health

• **Not Applicable**: There are no lives or health systems in 2048, as it's a puzzle game.

3.2.5 Game Ending

• **Game Over**: If no moves are available, the game ends, and a Game Over screen is displayed with options to restart.

• **High Scores**: After a game ends, players can view their final score and compare it with their personal or global high scores.

3.2.6 User Reviews and Ratings

- **Submit Review:** Players should be able to submit reviews and ratings for the game, including comments on gameplay, graphics, difficulty, and overall enjoyment.
- **View Reviews:** All users should be able to view reviews and ratings submitted by other players, providing insights into the game's quality.

3.3 Performance Requirements

- **Load Time:** The game should load within 5 seconds after being launched to ensure a quick start-up time.
- **Response Time:** Player actions (such as moving, shooting, or interacting with the environment) should register within 100ms to ensure smooth gameplay.

3.4 Design Constraints

- **Responsive Design:** The game interface must be adaptive, ensuring that it works well on different screen sizes, from desktop monitors to mobile devices.
- **Consistent Aesthetics:** The design should maintain a cohesive look and feel, including a consistent color scheme and interface elements throughout the game.
- **Framework Standards:** The game should follow best practices for design and development to ensure it can be easily maintained and updated in the future.

3.5 Software System Attributes

3.5.1 Reliability

- The game should have an uptime of 99.9%, ensuring minimal disruptions and smooth gameplay.

3.5.2 Availability

- The game should be available for play at any time, ensuring players can access it on-demand.

3.5.3 Security

- **Data Security**: Implement encryption for any user data, such as account information, if applicable.
- **Secure Transactions**: If the game has in-game purchases or accounts, they must be protected with secure payment systems.

3.5.4 Maintainability

- The game's codebase should be well-documented and adhere to coding standards, making it easy to update, fix bugs, or add new content in the future.

3.5.5 Portability

- The game should be playable on multiple platforms, including PC, macOS, and mobile devices (iOS, Android), ensuring cross-platform compatibility.

4. Appendix

4.1 Glossary

- **Responsive Design:** An approach to web design that ensures a game interface adapts seamlessly to various screen sizes and orientations, providing an optimal user experience across devices.
- **User Interface (UI):** The means by which players interact with the game, including visual elements like buttons, menus, and game boards, as well as input methods such as keyboard or touchscreen controls.
- **API (Application Programming Interface):** A set of protocols and tools that allow different software components to communicate with each other. In game development, APIs enable the integration of various functionalities, such as graphics rendering and sound processing.
- **Game Engine:** A software framework used for the development and creation of video games. It provides the necessary tools and features, such as physics simulation, graphics rendering, and sound, to build and run games.
- **Pygame:** A set of Python modules designed for writing video games. It provides functionalities for creating games, including handling graphics, sound, and input devices.
- **Python:** A high-level programming language known for its readability and versatility. In game development, Python is often used for scripting game logic and integrating various game components.