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 SRS covers the development of the **Tirandazi** game, which will provide players with an interactive 2D shooting experience. The game will feature player controls, enemy AI, scoring systems, and power-ups. The core mechanics will be developed using Python and Pygame. The game will include functionality for shooting, movement, enemy interaction, and a user interface for displaying score and lives.

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:
- Sound Design Best Practices:
- GitHub for Version Control

2. Overall Description

2.1 Product Perspective

Tirandazi is a standalone 2D action game designed to deliver an engaging and exciting experience where players control characters to shoot enemies while navigating obstacles. The

game features interactive mechanics like scoring, lives, power-ups, and player-enemy interactions.

2.2 Product Functions

- **Player Controls:** Enables movement and shooting through keyboard inputs.
- **Enemy AI**: Controls the movement and actions of enemy characters.
- **Scoring System:** Tracks player performance based on kills and actions.
- **Lives and Health:** Players have limited lives and health, which can be replenished under certain conditions.
- **Power-ups:** Grants temporary advantages like extra health or speed boosts.
- **Sound Effects**: Adds sound feedback for various actions such as shooting, enemy hits, and score changes.
- **Dynamic Game Speed:** Changes the speed of the game over time to increase difficulty.

2.3 User Classes and Characteristics

- **Players:** Individuals who interact with the game, control the character, and aim to achieve the highest score.
- **Game Admin**: Controls game settings, modifies difficulty levels, and manages content (e.g., game assets or power-ups)

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 interaction (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 enemies.
- 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 controls the character, interacts with enemies, and views the score and health.
- **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 all lives, showing the final score and options to restart or exit.

Not applicable.

3.1.2 Hardware Interfaces

- **Keyboard**: Players use the keyboard for movement (arrow keys/WASD) and actions (space bar for shooting).
- **Mouse (optional)**: If a mouse interface is implemented, players may use it for additional features or interactions (e.g., aiming or menu selection).

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 or achievements, 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

- **Character Controls:** The player must be able to control the movement of the game character using the keyboard (arrow keys/WASD) and shoot using the space bar.
- **Enemy AI:** Enemies must have predefined behaviors, moving and attacking the player within the game environment.

3.2.3 Scoring and Progression

- **Score Tracking:** The game should track the player's score based on kills, achievements, or time played.
- **Game Levels:** The game will include multiple levels with increasing difficulty.
- **Power-ups:** Items that enhance the player's abilities should appear randomly during gameplay, such as health boosts, speed increases, or special attacks.

3.2.4 Health and Lives

- **Health Bar:** Players must see their health represented visually (e.g., a health bar) on the screen.
- **Lives:** Players start with a set number of lives. If health reaches zero, the game ends unless extra lives are obtained through power-ups.
- **Respawn:** Players should be able to restart from the last checkpoint if they lose all their lives.

3.2.5 Game Ending

- **Game Over**: If the player loses all lives, the game should end and show a Game Over screen with an option to restart.
- **High Scores**: After completing or losing a game, players can see their final score and compare it to other 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.

3.2.7 Newsletter Subscription

- **Subscribe:** Players should have the option to subscribe to a newsletter for game updates, new features, or upcoming releases by providing their email address.

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:** A design approach that ensures a game interface works well across various screen sizes and devices.
- **User Interface (UI):** The visual elements that allow users to interact with the game, such as buttons, menus, and character controls.
- **API (Application Programming Interface):** A set of functions that allow the game to interact with external systems, such as databases or multiplayer services.
- **Game Engine:** The software framework used to create and develop the game, which handles graphics, physics, and sound.
- **Pygame:** A set of Python libraries used to develop 2D games, handling graphics, sound, and user input.
- **Python:** A high-level programming language used to write the game's core logic and mechanics.