COMP3211

Software Engineering

Group 1 – Software Requirement Specifications

Monopoly

Hong Kong Special Edition

Group 1	
Student ID	Student Name
	Kent Max CHANDRA
	Cheuk Tung George CHAR
	Tommaso Fabrizio COSTANTINI
	Xin DAI

CONTENTS

CONTENTS

I. Preface

Preface

This Software Requirements Specification (SRS) document outlines the requirements for the development of a digital-based Monopoly game, hereinafter referred to as "the game." This document is intended for the project team members, software developers, testers, and stakeholders involved in the implementation of the game.

The SRS is structured to ensure that all aspects of the game's functionality, user interactions, and system constraints are clearly defined and understood. It will serve as a foundation for the design and development phases, facilitating communication among team members and ensuring compliance with the rules and user stories outlined in the course materials.

Version history will be maintained to track modifications and updates to the document, with changes summarized to provide clarity on the evolution of the requirements. This SRS aims to be valid, consistent, complete, realistic, and verifiable, providing a comprehensive guide for the successful implementation of the game. Version 1.0 is the first and latest version of our project.

By adhering to the specified requirements and guidelines, the project team will ensure that the final product meets the expectations of its users while fulfilling the educational objectives of the course.

II. Introduction

Introduction

The purpose of this Software Requirements Specification (SRS) is to define the requirements for a Monopoly game. This game aims to provide an engaging and entertaining experience for players while adhering to the traditional rules and mechanics of Monopoly.

Need for the System

As a popular board game, Monopoly offers players the opportunity to engage in strategic decision-making, negotiation, and resource management. The digital version of this game will enhance accessibility, allowing users to play from any location without the need for a physical board or pieces. Enable players to enjoy the experience through the game.

System Functions

The game will support various functionalities, including:

- **Game Initialization**
- **Player Management**
- **Player Status Display**
- **Game Status Overview**
- **Turn Management**
- **Game Saving and Loading**
- **Gameboard Management**

III. Glossary

Glossary

Term	Full name	meaning
SRS	Software Requirements Specification	An official statement of what the system developers should implement
CLI	Command-line interface	a user interface that allows users to interact with computer programs by typing commands and parameters, rather than using a graphical interface. Users enter instructions in the command line, and the program executes the corresponding actions based on those commands.
GUI	Graphical User Interface	a visual interface that allows users to interact with software applications through graphical elements such as windows, icons, buttons, and menus. Unlike a command-line interface (CLI), which requires users to type commands, a GUI enables users to perform actions by clicking or tapping on visual elements, making it more intuitive and user-friendly, especially for those who may not be familiar with command syntax.
US	User story	an informal, natural language description of one or more features of a software system
FR	Functional Requirements	statements of services the system should provide
NFR	Non-Functional Requirements	constraints on the system’s services or functions

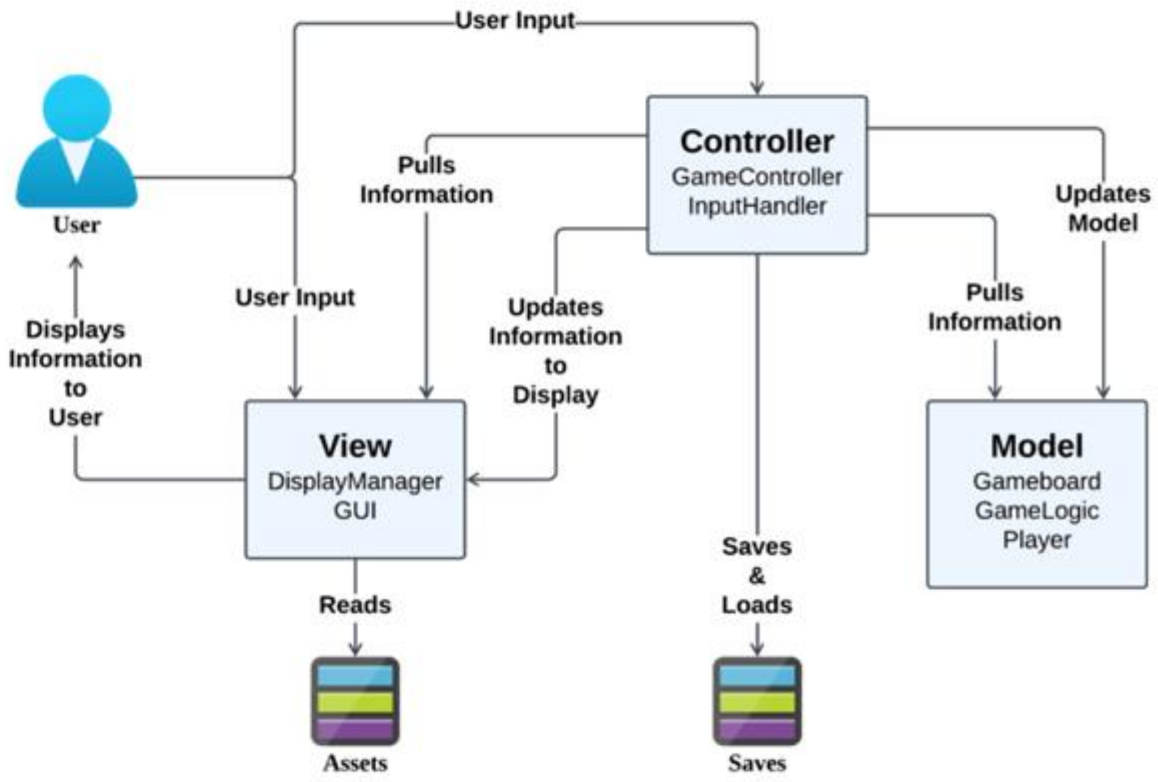
IV. User Requirements Definition

User Requirements Definition

User story (According to appendix C)	Content
US1	• The system shall allow players to initiate a new game using a predefined gameboard.
US2	• The system shall allow players to enter custom names for their profiles. • The system shall provide an option to generate random names for players if they choose not to enter custom names.
US3	• The system shall provide a method for players to view the status of a specific player. • The system shall provide a method to display the status of all players, including their balances and properties.
US4	• The system shall display the current status of the game, including player positions on the gameboard. • The system shall visualize the layout of the gameboard, showing all property squares and their ownership status.
US5	• The system shall provide a command for players to query which player's turn is next. • The system shall display the name of the next player to take their turn.
US6	• The system shall allow players to save the current game state to a file. • The system shall confirm that the game has been successfully saved.
US7	• The system shall allow players to load a previously saved game from a file. • The system shall restore the game state to the point at which it was saved, including player positions, balances and game board information.
US8	• The system shall allow gameboard designers to create a new gameboard based on an existing one by modifying properties. • The system shall enable designers to change property names, prices, and rent values for each square.
US9	• The system shall allow designers to load an existing gameboard for customization. • The system shall enable designers to modify any square on the loaded gameboard.
US10	• The system shall allow designers to save their newly designed gameboard to a file. • The system shall confirm that the gameboard has been successfully saved and is accessible for future use.

V. System Architecture

System Architecture



The system architecture of the Monopoly game is designed following the **Model-View-Controller (MVC) pattern**, which ensures a clear separation of concerns among the components.

1. **Model**:
   a. Contains hardcoded default game information.
   b. Manages game logic, player data, and gameboard configurations.
   c. Responsible for saving and loading game instances from the Saves folder.

2. **View**:
   a. Initializes by referencing all assets from the Assets folder.
   b. Starts without specific data displayed until populated by the Controller.
   c. Receives user input and updates its internal fields or calls appropriate methods.

3. **Controller**:
    a. Acts as the intermediary between the Model and the View.
    b. Pulls information from the Model to populate the View.
    c. Handles user input received from the View; it updates the Model when necessary.
    d. Manages game flow by determining the next action based on the input and Model state.
    e. Updates the View with new or modified information from the Model.

Interaction Flow

- User input is received by View, which then communicates with the Controller.
- The Controller assesses the input and may update the Model or determine the next action.
- Updated information from the Model is used to refresh the View, ensuring the user interface reflects the current state of the game.

Component Access

- The Controller is the central component that has access to both the Model and the View.
- The Model and View do not directly interact with each other or the Controller; they operate independently, relying solely on the Controller for communication.

This architecture promotes a modular design, enhancing maintainability and scalability while ensuring a cohesive user experience.

VI. System Requirement Specification

Functional Requirements

1. Game Initialization

- **FR1.1**: The system shall allow players to start a new game using an existing gameboard.
    - **FR1.1.1**: Players will be presented with a graphical menu to select from predefined gameboards.
    - **FR1.1.2**: There will be a button that allows the user to start the game by pressing it.

2. Player Management

- **FR2.1**: The system shall allow players to enter custom names for their profiles.
    - **FR2.1.1**: A text input field will be provided for each player to enter their name.
    - **FR2.1.2**: The system shall generate random names if players choose not to enter custom names, displayed in the GUI.

3.Player Status Display

- **FR3.1**: The system shall provide a feature to view the status of a specific player or all players.
    - **FR3.1.1**: During the game, each player's information, including their money, properties, and names, will be displayed on the screen.

4.Game Status Overview

- **FR4.1**: The system shall display the current game state after each turn.
    - **FR4.1.1**: The gameboard will be visually represented in the GUI, showing player positions and property ownership.
- **FR4.2**: The system shall provide real-time updates on game status.
    - **FR4.2.1**: The GUI will refresh automatically after each action to reflect the latest game state.

5. Turn Management

- **FR5.1**: The system shall allow players to see whose turn it is next.
    - **FR5.1.1**: The current player will have an indicator to show that it is their turn.

- **FR5.2**: The system shall end the turn of the current player at the appropriate time.
    - **FR5.2.1**: The game system will automatically end the turn of the current player after all actions are completed, allowing the next player to start their turn.

VI. System Requirement Specification

6. Game Saving and Loading

- **FR6.1**: The system shall allow players to save the current game state to a file.
  - **FR6.1.1**: A "Save Game" button shall available in the GUI, prompting players to enter a filename and save the game.
  - **FR6.1.2**: The system shall confirm the successful save operation through updating the save record displayed on the interface.
- **FR6.2**: The system shall allow players to load a previously saved game from a file.
  - **FR6.2.1**: A "Load Game" button will provide access to saved game files.
  - **FR6.2.2**: The system shall restore the game state, including player positions, balances and game board details, upon loading.

7. Gameboard Management

- **FR7.1**: The system shall allow gameboard designers to create a new gameboard based on an existing one.
  - **FR7.1.1**: Designers will have access to a graphical interface for modifying property names, prices, and rent values.
- **FR7.2**: The system shall enable designers to load and customize existing gameboards.
  - **FR7.2.1**: A "Load game board" button will provide access to saved game board files.
  - **FR7.2.2**: The system shall restore the game board state, including the details of each property, such as name, price, and rent, as well as the other fixed squares, upon loading.
- **FR7.3**: The system shall allow designers to save their newly designed gameboards to a file.
  - **FR7.3.1**: A "Save game board" button shall available in the GUI, prompting players to enter a filename and save the game board details.
  - **FR7.3.2**: The system shall confirm the successful save operation through updating the save record displayed on the interface.

VI. System Requirement Specification

Non-Functional Requirements

1. Usability

- **NFR1.1**: The GUI shall be easy to use. User shall be able to use all the system functions after half an hour of training. After this training, users can find and use all function in the game by their own without reading the user manual.

2. Performance

- **NFR2.1**: The system shall respond to user inputs (e.g., clicks, selections) within 1 second to ensure a smooth and engaging gaming experience.
- **NFR2.2**: To maintain fluid user experience, during system operation, the GUI interface shall maintain a frame rate of over 30 frame per second.

3. Reliability

- **NFR3.1**: The system shall handle unexpected inputs gracefully. For example, input string to the property price when designing the game board. Providing informative error messages and guidance to users rather than crashing or freezing.

4. Robustness

- **NFR4.1**: The average time to restart after a failure shall not exceed 2 minutes to minimize downtime.
- **NFR4.2**: The percentage of events causing failure shall be less than 0.1% during normal operation.