**Group 21**

**The Monopoly Game**

**Software Requirements Specification**

Chau Wing Yee 22082797D

Li Yuhan 22081625D

Liu Yim Shan 22084167D

**Preface:**

This document is for anyone who creating and using the Monopoly game, such as the development team and end users. It outlines the software requirements to ensure everyone understands the system's goals and what needs to be delivered.

Expected Readership:

* Developer: build the Monopoly game based on user requirements
* End Users: provide feedback on the improvement of the game

Version History: 1.0

* Rationale: provide a basic understanding of the game's requirements and functionalities
* Summary of the changes: created the document, including introduction, user requirements definition, system architecture, and system requirements specification

**Introduction:**

The Monopoly game is designed to create an engaging experience for board game fans even if they do not have the physical board on hand. It keeps the classic Monopoly gameplay while adding design features. It enables users to either play the game with friends and family or design a custom gameboard for their own. The Monopoly game will include the following functions:

* Start: Players can start a new game and play with 1-5 people. They will take turns rolling a pair of four-sided dice to move their position on the gameboard. They can buy properties and charge rent, depending on which position they landed.
* Game management: The system will keep track of players’ statuses. The one who has a negative income will lose the game. If one player is left or after 100 rounds, the game will end, with the winner having the highest money.
* Save and load game progress: Players can save and load their game's progress, allowing them to continue their games at any time.
* Design: Users can design a custom gameboard by modifying the squares. They can save the gameboard and load it for continuous modification.
* User interaction: Users can input commands and receive responses through a command-line interface.

**Glossary:**

* MVC: Model-View-Controller
* SRS: Software Requirements Specification
* Command Line Interface: a text-based user interface used to interact with the system
* Player: a person who can play the Monopoly game
* Designer: a person who can design the gameboard of the Monopoly game
* Square: a position on the gameboard. It can be property, chance, income tax, free parking, go to jail, just visiting or go

**User requirements definition:**

Functional requirement:

1.  As a player, I want to start a new game with an existing gameboard.

2.  As a player, I want to name the players with input or randomly generated strings to

better differentiate the players.

3.  As a player, I want to see the status of any specific player and all players.

4.  As a player, I want to see the status of the game, including the squares and the players’

positions on the gameboard.

5.  As a player, I want to query the next player.

6.  As a player, I want to save the current game to a file.

7.  As a player, I want to load a game from a file and continue the game.

8.  As a gameboard designer, I want to design a new gameboard based on an existing

gameboard by modifying the property squares (including their name, price, and rent).

9.  As a gameboard designer, I want to load an existing gameboard and customize it by

modifying its squares.

10.  As a gameboard designer, I want to save the gameboard I designed.

Non-functional Requirement:

1. Usability: The Monopoly game should be easy to use with clear instructions for user

actions. User manual should be provided to users to guide them on how to play

the game.

2. Performance: The system should respond to user actions within 1 second such as starting a

game, saving the current game state and loading a saved game.

3. Reliability: The system should ensure that the game runs smoothly without crashes or

errors. It should handle invalid inputs gracefully and provide appropriate error

messages.

4. Scalability: The system should be designed to handle an increasing number of players.

5. Portability: The system shall be work on different types of system such as Windows,

macOS.

6. Maintainability: The system should be easy to update and improve in the future, with well-

organized and understandable code.

7. Data integrity: The system should ensure that the game state is saved to a file accurately

and completely.

**System architecture:**

The Monopoly game will employ the Model-View-Controller (MVC) architectural pattern. It is divided into three main parts.

* Model: Encapsulates the game’s data and logic. It keeps track of the game’s data like the player’s position, money and properties owned. Also, it ensures that all the actions and events follow the game rules.
* View: Handles user interaction and displays results to the user through the command-line interface. It presents the game status, prompts for user input and shows the results of actions taken. For example: after the player rolls dice, the View will show the player how many squares he moves and where he lands.
* Controller: Processes user input and updates to the View and the Model. For example: if a player wants to buy a property, the Controller will update the Model to show the purchase and then update the View to display the remaining money the player has.

**System requirements specification:**

Functional Requirement:

* Game initialization:

1. The system shall allow players to start a new game by selecting predefined gameboards or creating custom ones to play.
2. The system shall allow players to name on their own or use a name which randomly generated.

* Player actions:

1. The system shall allow players to take turns rolling two four-sided dice and move positions based on the sum of the rolled dice.
2. The system shall allow players to buy properties if they land on properties which have no owner.
3. The system shall allow players to pay rent when they land on properties owned by other players.
4. The system shall implement rules for special squares when they land on specific square, e.g. Paying income tax when landing on the Income Tax square.
5. The system shall allow players to see the status of specific players or all players, displaying their current assets, properties owned, and positions on the board.
6. The system shall allow players to see the status of the game, including the squares and the players’ positions on the gameboard.
7. The system shall allow players to query the next player.

* Game saving and loading:

1. The system shall allow players to save the game to a file, preserving all relevant information, including player positions, total money, properties owned, and game round.
2. The system shall allow players to load a previously saved game from a file, retrieving all game data accurately for players to play continuously.

* End game:

1. The system shall determine the end of the game after 100 rounds or when only

one player left.

1. The system shall declare the winner based on the highest amount of money.

* Custom gameboard creation:

1. The system shall allow users to create a custom gameboard based on an existing

gameboard by modifying the property squares (including their name, price, and

rent).

1. The system shall allow users to save the gameboard they designed to a file.
2. The system shall allow users to load an existing gameboard from a file.
3. The system shall allow users to play with the gameboard they designed.

Non-functional Requirement:

1. Usability: User should be able to use the Monopoly game system after reading the user

manual for no more than 30 minutes. After reading the user manual, users

should no longer enter invalid command.

2. Performance: The system should respond to user actions within 1 second such as starting a

game, saving the current game state and loading a saved game.

3. Reliability: The system should ensure that the game runs smoothly without crashes or

Errors during 99% of the game sessions. It should handle invalid inputs

gracefully and provide appropriate error messages.

4. Scalability: The system should be designed to handle an increasing number of users

without performance degradation.

5. Portability: The system shall be compatible with Windows and macOS operating systems,

ensuring users can play the game on different platforms without issues.

6. Maintainability: The system should be maintainable with well-documented code, with

comments explaining the purpose and functionality of codes. This help

future developers to understand and modify the code easily.

7. Data integrity: The system should ensure that the game state is saved to a file accurately

and completely. When a user loads the saved game file, there should be no

loss of game data.