Project Documentation: Mahjong Game Development

Jingwang Li, Jie Mao, Yixin Niu, Lanyun Xiao

29th April 2024

1 Introduction

This document outlines the development plan for a computer-based Mahjong game, detailing user and system requirements, division of labor, functional and non-functional requirements, design patterns, and project structure.

2 User and System Requirements

2.1 User Requirements

- Visual interface for user interaction: login, game preparation, gameplay actions (draw, discards, chow, pong, kong), score checking, and dice rolling.

2.2 System Requirements

- Desktop game application.
- Password encryption using hash codes for security.
- Database integration for user data management.
- Game logic for determining winners, turn management, and scoring.
- Graphical User Interface (GUI) using JavaFX.

3 Functional and Functional Requirements

3.1 Functional Requirements

- User registration and login.
- Gameplay mechanics including card playing and decision-making for chow, pong, kong, and hu.

- Score display and dice rolling functionalities.
- Database interactions for authentication and user data management.
- Game state management including turn and role determinations.
- Automatic card organization and default play on timeout.

3.2 Non-Functional Requirements

- Code standardization and conventions.
- High responsiveness and compatibility across multiple devices.
- Simple and intuitive user interface.
- Robust error handling and feedback mechanisms.
- Enhanced maintainability and modularity of the code.
- Support for English language only.

4 Division of Labor and Weekly Work Plan

4.1 Division of labor

- System Implementation: Li Jingwang
- GUI and Database Design: Mao Jie, Li Jingwang
- Module Design and Implementation: Niu Yixin, Xiao Lanyun

4.2 Weekly Work Plan

4.2.1 Week 10 to 12 (Deadline: 05/13/2024)

– Create a project on github and invite members of my group to join the project.
– Build the framework of the game project
– Start testing game framework, ensuring correctness of basic functionalities.

4.2.2 Week 13 (Deadline: 05/20/2024)

– Complete major functionality implementation of the game, including generation, arrangement of tiles, and player interactions.
– Start implementing game victory conditions and scoring system.
– Continue implementing and testing game victory conditions and scoring system.

4.2.3 Week 14 (Deadline: 05/27/2024)

- Optimize game user interface and interaction experience.
- Enhance visual and audio effects of the game.
- Perform final functionality testing and debugging.

4.2.4 Week 15 (Deadline: 06/07/2024)

- Finalize project documentation and reports, including project summary and reflections.
- Organize and submit all project files to GitHub repository.
- Ensure GitHub repository is fully uploaded and complete.
- Include GitHub repository link in the submitted PDF document.

5 Project Structure

The project is structured into three main packages, each with its own set of classes and interfaces that work together to create the game application's framework.

5.1 System Package

- **GameManager Class**
  - Attributes: ruleScreen, gameScreen, loginScreen, scoreScreen, menuScreen, mahjongGame
  - Methods: showRuleScreen(), showGameScreen(), showLoginScreen(), showScoreScreen(), showMenuScreen(), run(), exitGame()
  - Design Pattern: Facade Pattern, simplifying interaction with the game system as the main controller.
- **Screen Interface**
  - Attributes: Height, Width, BackGround
  - Methods: getCanvas(), paint()
- **Game Interface**
  - Methods: startNewGame(), isPaused(), swop(), isGameOver(), checkForPause(), isRoundEnd()
- **Screen Enum**
  - Values: RULE_SCREEN, GAME_SCREEN, LOGIN_SCREEN, SCORE_SCREEN, MENU_SCREEN

5.2 Module Package

- **mahjongGame Class**
  - Attributes: player, mahjongTable, playerListener, gameStatus
  - Methods: mahjongGame(), getInstance(), startNewGame(), isPaused(), swop(), isGameOver(), checkForPause(), isRoundEnd(), shuffleTiles()
  - Design Pattern: Singleton Pattern, ensuring the uniqueness of game logic.
- **Player Class**
  - Attributes: location, playerTileWall, name
  - Methods: player(), chow(), Pong(), Kong(), disCard(), getName(), getPlayerTileWall(), getLocation()
- **TileWall Class**
  - Attributes: playerTileWall, TileFactory, TableTileWall, discardWall
  - Methods: TileWall(), getPlayerTileWall(), getTableTileWall(), GetDiscardWall()
  - Design Pattern: Composite Pattern, representing a wall composed of multiple tiles.
- **TileFactory Class**
  - Methods: TileFactory(), createTile()
  - Design Pattern: Factory Pattern, representing a factory that produces tiles.
- **NumberTile Class**
  - Attributes: rank
  - Methods: NumberTile(), getRank()
- **WindAndDragonTile Class**
  - Methods: WindAndDragonTile()
- **Tiles Class**
  - Attributes: suit
  - Methods: Tiles(), getSuit(), setSuit()
- **Dice Class**
  - Attributes: number
  - Methods: dice(), roll()
- **PlayerLocation Enum**
  - Values: WEST, EAST, SOUTH, NORTH

5.3 Display Package

- **GameScreen Class**
  - Attributes: Pause_gameButton, Continue_gameButton, Chow_button, Pung_button, Kong_button, Win_button, Hand_tilesButton
  - Methods: loadGameWindow()
- **LoginScreen Class**
  - Attributes: Login_button, background
  - Methods: loadLoginWindow()
- **RuleScreen Class**
  - Attributes: Return_toMenuButton
  - Methods: loadRuleWindow()
- **ScoreScreen Class**
  - Attributes: Return_toMenuButton
  - Methods: loadScoreWindow()
- **MenuScreen Class**
  - Attributes: Start_gameButton, Exit_gameButton, Rules_explanationButton Score_inquiryButton
  - Methods: loadMenuWindow()
- **playerListener Class**
  - Methods: playerListener(), addListener(), delectListener(), notify(), isPlayerClick()
  - Design Pattern: Observer Pattern, managing event listeners and notifications.
- **ButtonDisplay Class**
  - Attributes: Login_button, Start_gameButton, Exit_gameButton, Rules_explanationButton Score_inquiryButton, Return_toMenuButton, Pause_gameButton, Continue_gameButton, Chow_button, Pung_button, Kong_button, Win_button, Hand_tilesButton
- **ApplicationStart Class**
  - Methods: start(), main()

6 Design Patterns

The following design patterns have been applied within the project to enhance maintainability, scalability, and loose coupling of the code, making it clearer and easier to understand.

- **Singleton Pattern**: Applied to mahjongTable and mahjongGame classes to ensure a single instance of these key components within the application.
- **Observer Pattern**: Applied to playerListener class to allow objects to notify other dependent objects when their state changes.
- **Facade Pattern**: Applied to GameManager class to provide a simple interface to the more complex subsystems of the game application.
- **Composite Pattern**: Applied to TileWall class to allow objects to be composed into a tree structure to represent part-whole hierarchies.
- **Factory Pattern**: To be applied to the TileFactory class. This pattern is particularly useful when the system needs to be independent of how its objects are created, composed, and represented.

7 UML design

To accurately reflect user requirements, the project framework, and the sequence of program operations, we have provided a use case diagram, class diagram, and sequence diagram. Please note the following color coding in the class diagram: red indicates the system package, green denotes the module package, and blue represents the display package.

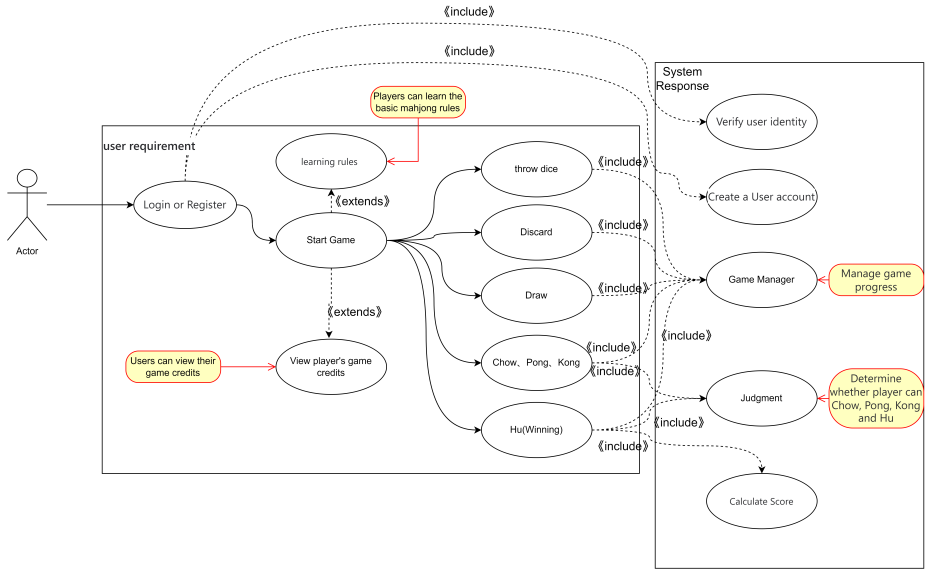

Figure 1: Use Case Diagram

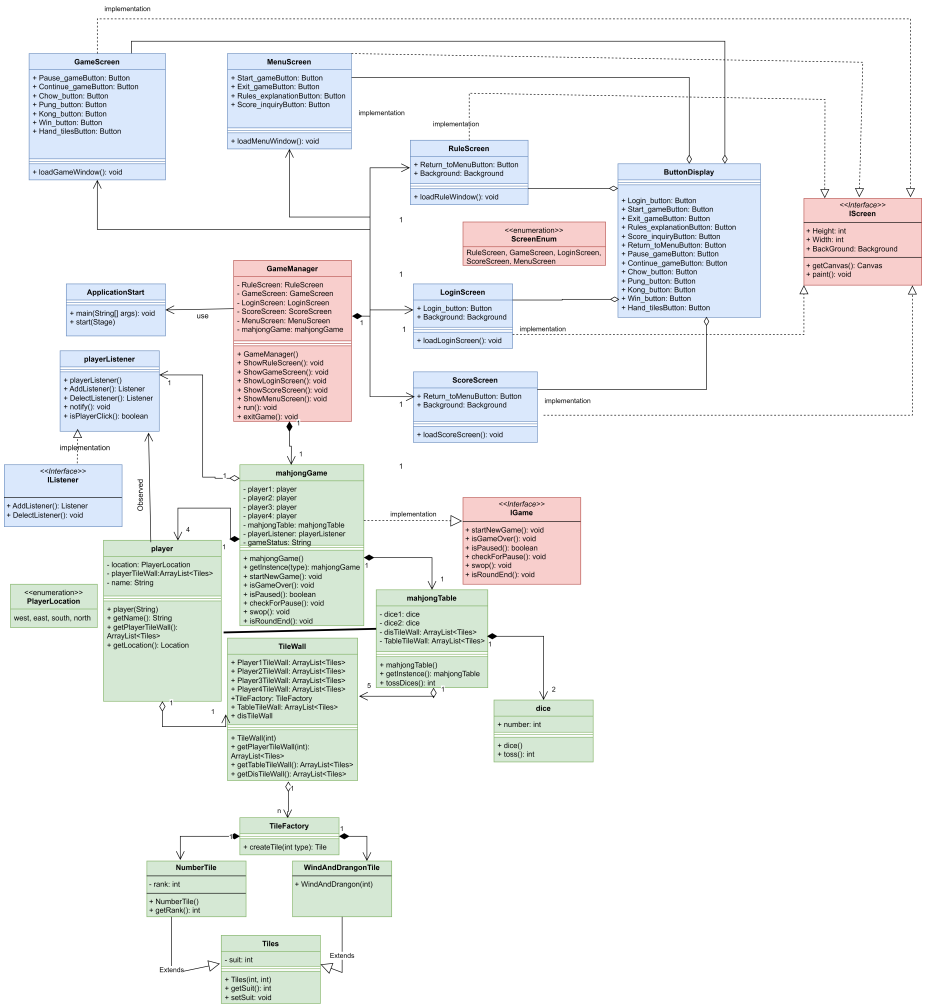

Figure 2: Class Diagram, red indicates the system package, green denotes the module package, and blue represents the display package

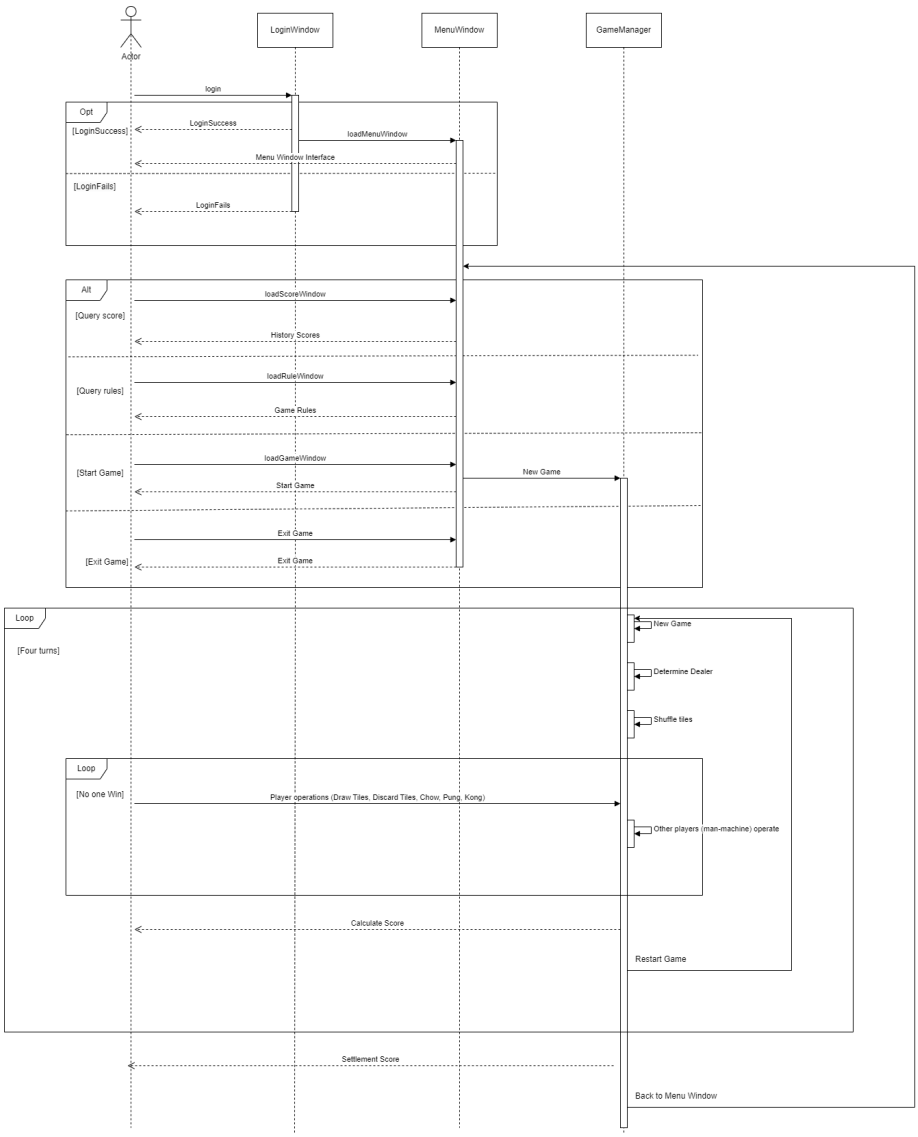

Figure 3: Sequence Diagram