


Beijing-Dublin International College

COMP2008J : Software EngProject 1

Software EngProject 1

Assignment :Mahjong tiles Game (Beijing Version)

Ziang Chen	**Shuhan Wang**	**Lingrui Sun**	**Hongyue Chen**
22207260	22207210	22207240	22207233

Honesty Pledge:

I clearly understand the Academic Rules of Beijing University of Technology and University College Dublin, and I am aware of the punishment for violating the Rules of Beijing University of Technology and University College Dublin. I hereby promise to abide by the relevant rules and promise the originality of my work. If found violating the rules, I would accept the punishment thereof.

https://github.com/Mystour/mahjong.git

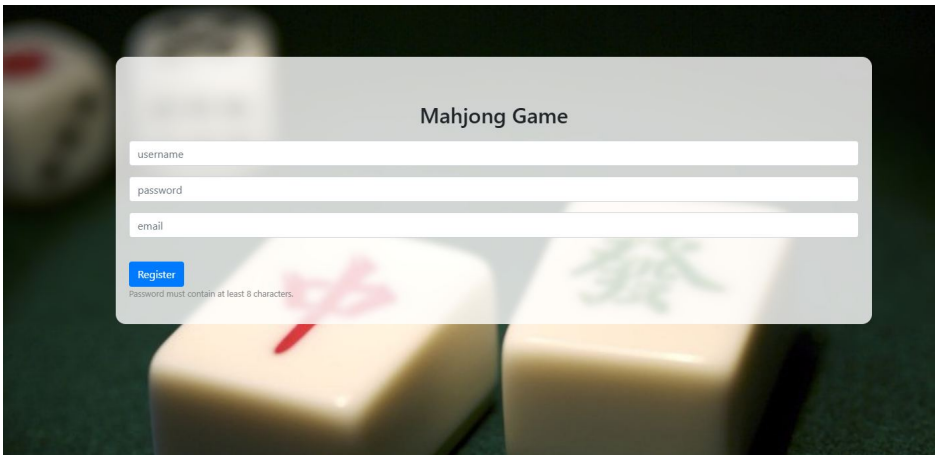
Date: 5/06/2024

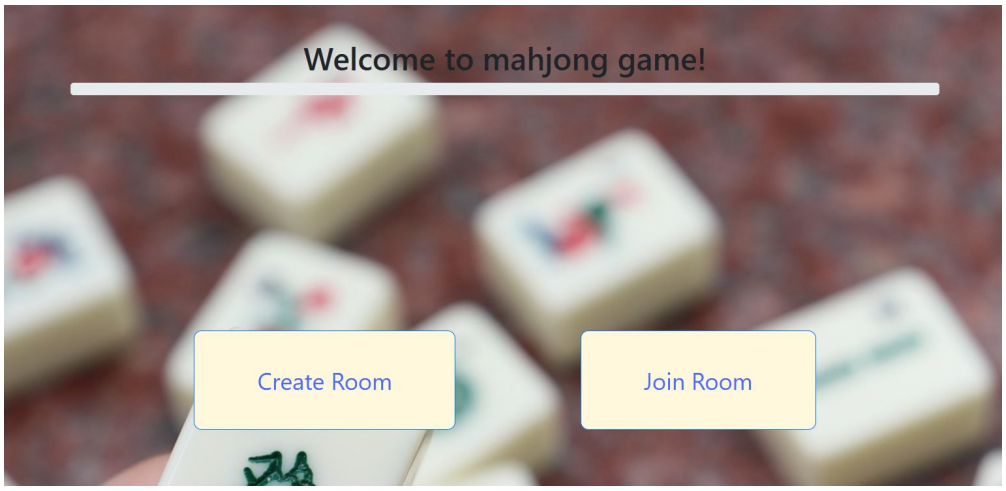
Contents

1 Project Explanation

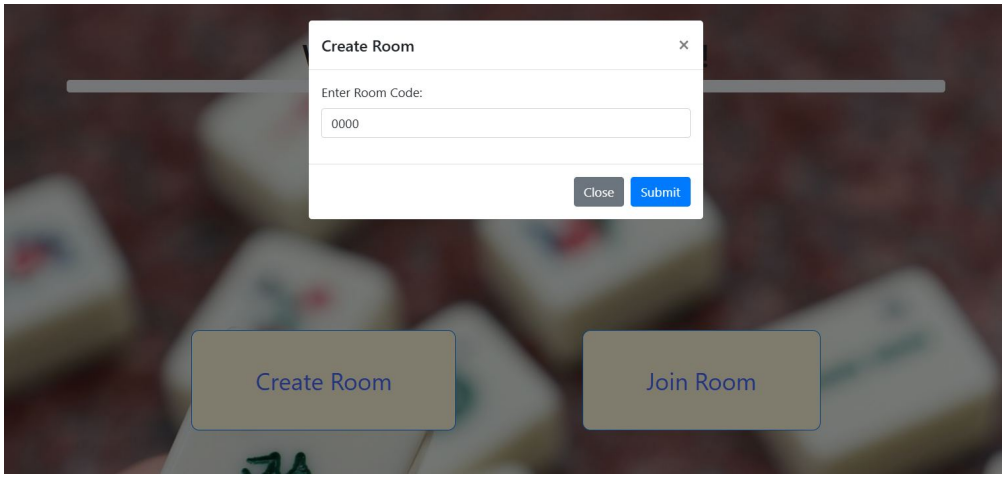
Our project focuses on developing the Beijing version of the Mahjong tiles game, which is deeply rooted in Chinese culture. The game involves strategically arranging tiles and is played by multiple players according to specific rules and scoring systems.

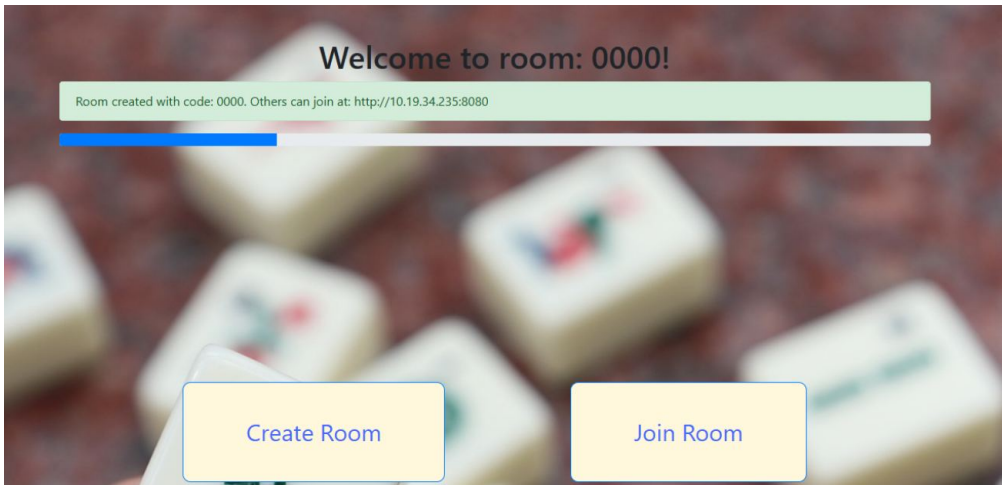
- **Game Launch**: Start the program on a computer and enter the login interface. Users can log in to the game system by registering a new account or using an existing account



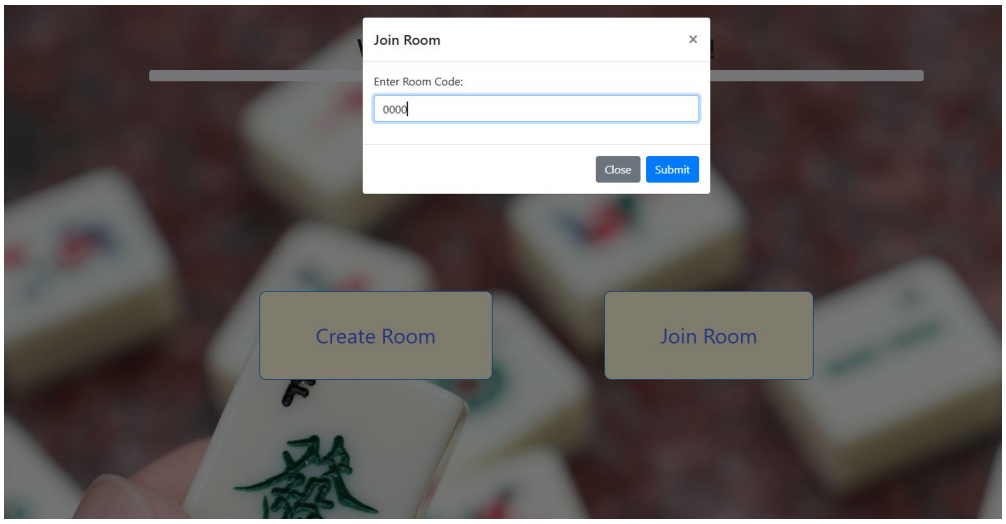


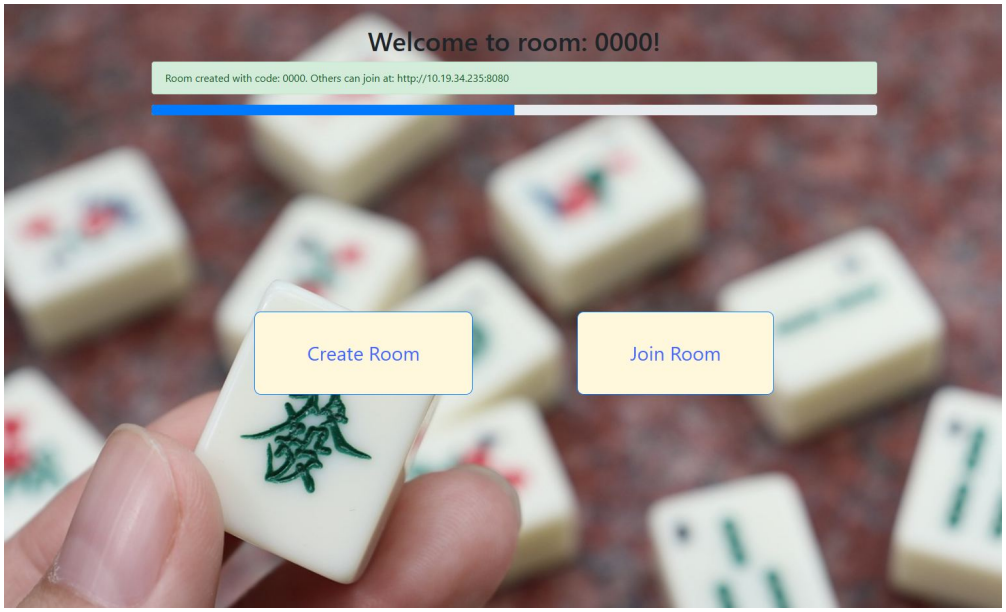
- **Create a room and invite other players**:After successful login, users can choose to create a new gaming room. After entering the room number created by the user, a URL is displayed to invite other players to join the room. Within the same LAN, other players can join the game room by accessing this URL and entering the room number.



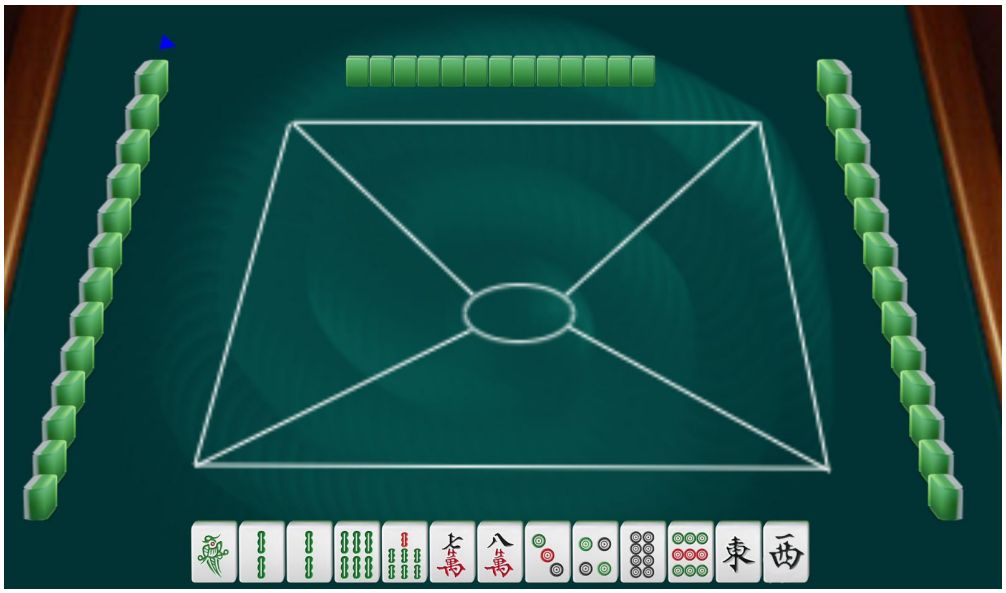


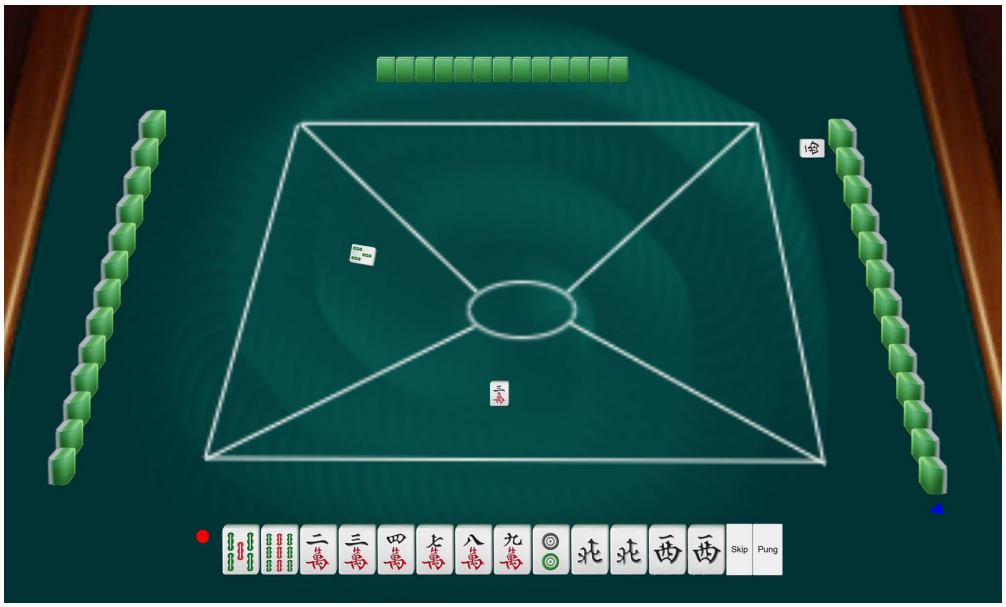
- **Join the room**: Other players can join the game by accessing the URL provided, entering the room input screen, and entering the room number. Each time a player successfully joins a room, the progress bar on the room screen advances by 1/4. When the number of players reaches four, the system automatically starts the game.

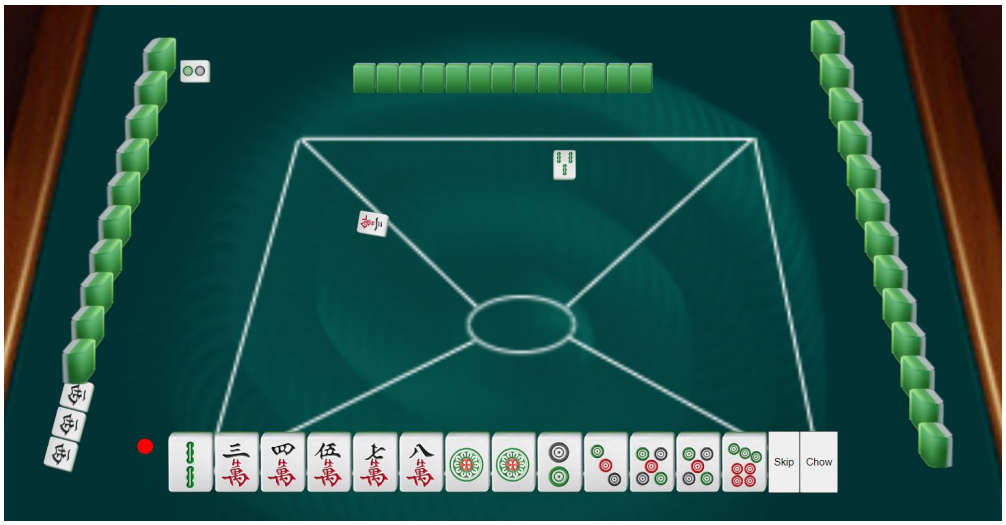




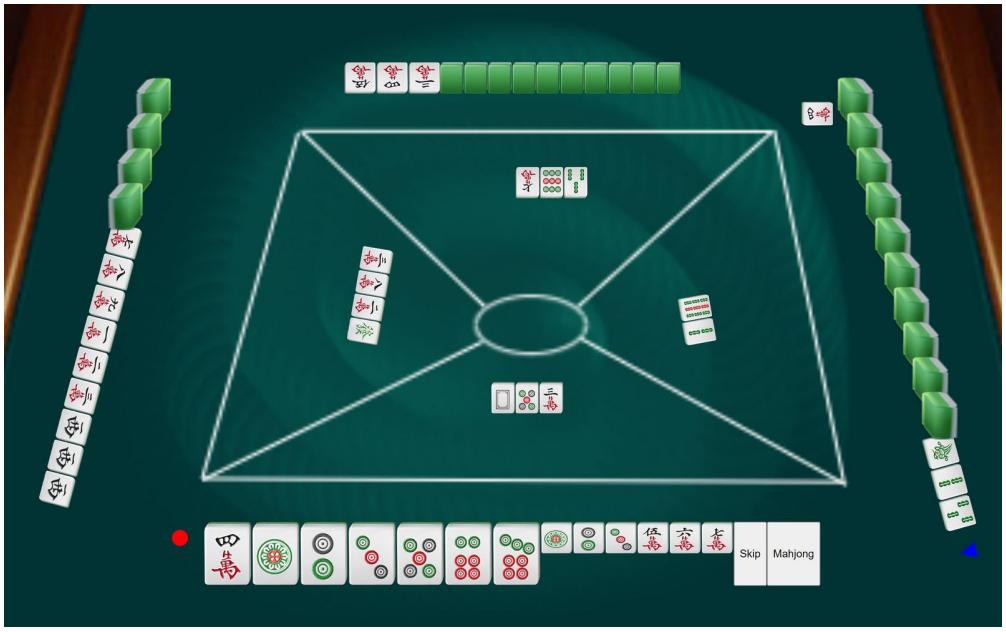
- **Playing Mahjong**: The game is played according to the Beijing version of mahjong rules, including grabbing cards, playing cards, Chow, Pung, Kung, Mahjong and other operations. The game status is updated in real time, and the interface of all players synchronously displays the current game status and their hand.

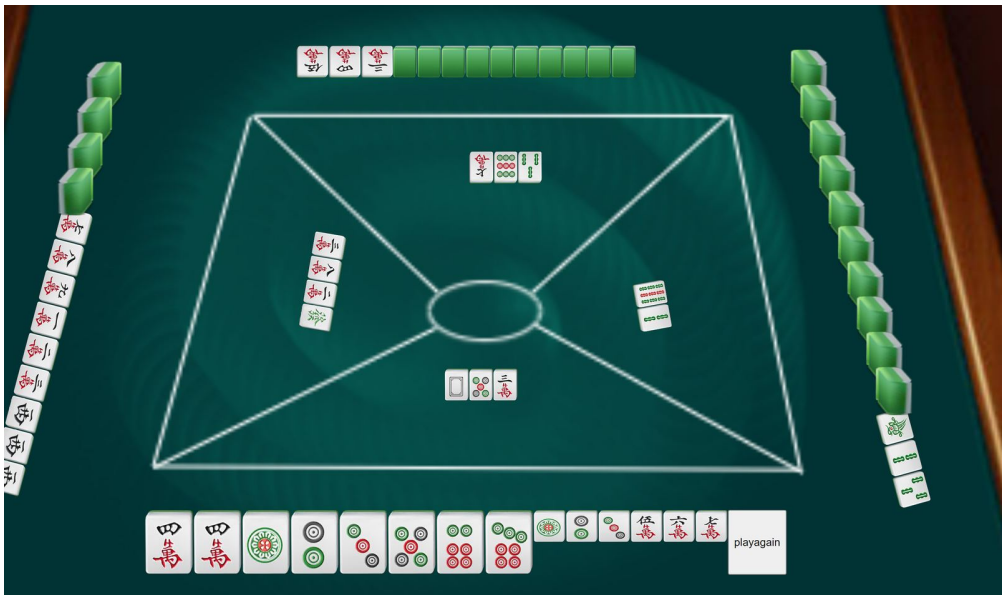




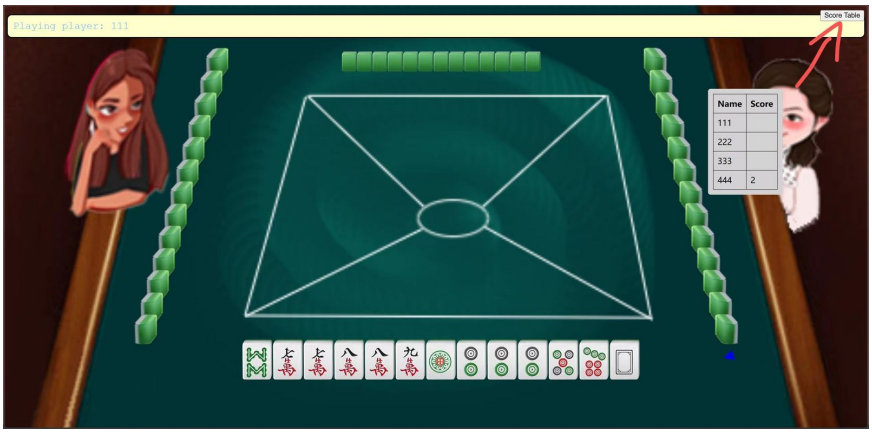


After any player Mahjong, the game ends





There is a button in the upper right corner of the game interface, which can be clicked to view the accumulated scores of each player



2 Implementation of game requirements

Requirement Analysis:

- **Game Rules**: We thoroughly analyzed the detailed rules of the Beijing version of Mahjong, including tile types (bamboo, character, circle, wind, dragon), the number of tiles, the game objective (forming four melds and a pair), and special rules (such as pong, kong, and winning).
- **User Requirements**: Ensured that users could play the game smoothly, with a user-friendly interface, supporting multiple players, and enabling the resumption of unfinished games.
- **System Requirements**:The system was implemented using Java, following software engineering guidelines, supporting multiplayer local network gameplay, providing a good user experience, and featuring game state saving and resumption capabilities.

2.1 User and System Requirements

2.1.1 Gameplay

Our system successfully implements the Beijing version of the Mahjong tiles game using Java, following strict software engineering guidelines.The game accurately enforces the Beijing Mahjong rules, supports multiple players in a single game, and facilitates local network multiplayer gameplay.

The entire system is implemented using Java, adhering to software engineering guidelines for robust and maintainable code. The game logic is designed to strictly enforce the rules of the Beijing version of Mahjong. The system supports multiple players in a single game and facilitates multiplayer gameplay over a local network.

2.1.2 Interaction

The system provides a user-friendly interface for game interaction. The system provides a user-friendly interface for game interaction.

The game runs smoothly, ensuring a good user experience. The interface is designed to be user-friendly, making it easy for players to interact with the game. The system provides clear error messages and gracefully handles exceptions to ensure uninterrupted gameplay.

2.2 Functional Requirements

- **Game Setup** The system allows setting up a Mahjong game with four players. Tiles are shuffled and distributed evenly among players, with the system deciding the seating arrangement.
- **Gameplay** Players take turns drawing tiles from the wall, making valid moves by either drawing a tile from the wall or claiming a discarded tile from another player. Players can declare winning hands when they have a valid Mahjong hand.
- **User Interface** The system provides a user-friendly interface displaying relevant information such as players' hands, the current state of the game, and any notifications or alerts.
- **Multiplayer Support** The system facilitates multiplayer gameplay, allowing multiple users to connect and play together online, handling communication between players, including actions taken and messages exchanged during the game.

2.3 Non-functional Requirements

- **Usability Standards** Players should only interact through the GUI, preventing manipulation of the code layer.
- **Performance Constraints** The system runs smoothly on various devices, with acceptable response times for users.
- **Maintainability** The system is developed in Java following software engineering principles, ensuring ease of maintenance and future upgrades.

3 Desgin pattern (Key Concepts)

In our Mahjong game project, we implemented several design patterns to ensure a robust, maintainable, and scalable architecture. Below are the patterns we used and their specific implementations within the project:

3.1 Structural

3.1.1 Composite

The Composite pattern allows us to compose objects into tree structures to represent part-whole hierarchies. It enables clients to treat individual objects and compositions of objects uniformly. In our Mahjong game, the Composite pattern is used to handle different types of Mahjong tiles and their relationships.

Composite Pattern Components:

- **Component**: This is an interface for all objects in the composition, both composite and leaf nodes. The `Tile` class serves as the component with methods like `getTileType()` and `getTileValue()`.
- **Leaf**: These are the individual objects in the composition that do not have any children. Include `CharacterTile`, `DotTile`, `BambooTile`, and `WindTile`. Each of these classes extends the `Tile` class and represents a specific type of Mahjong tile.

Listing 1: Tile Code Example

```
public enum TileType {
    BAMBOO, CHARACTER, DOT, DRAGON, WIND
}
```

- **Composite**: These are the nodes that can have children and are composed of leaf nodes. In the context of our diagram, the `Hand` class can be seen as a composite as it manages multiple `Tile` objects.

Composite Pattern Usage:

- **Tile Management**: By treating both individual tiles and collections of tiles uniformly through the `Tile` class, the Composite pattern simplifies the management of different tile types. This approach allows the game to handle various tile operations, such as drawing or discarding tiles, in a consistent manner.
- **Hand Composition**: The `Hand` class contains multiple `Tile` objects and provides methods to validate the Mahjong hand. This exemplifies the composite structure where a `Hand` can be composed of multiple individual `Tile` objects, enabling complex operations on the hand as a whole.

Listing 2: Code Example:Hand class manage multiple Tile Objects through ArrayList

```
public class Hand {
    private List<Tile>[] handcard;
    private final List<Tile> discards;
    private final List<Tile> pungs;
    private final List<Tile> chows;
    private final List<Tile> kongs;
```

```
    public Hand() {
        pungs = new ArrayList<>();
        chows = new ArrayList<>();
        kongs = new ArrayList<>();
        handcard = new List[5];
        for (int i = 0; i < 5; i++) {
            handcard[i] = new LinkedList<>();
        }
        discards = new ArrayList<>();
    }
    // Other methods ...
    public List<Tile>[] gethandcard() {
        return handcard;
    }
    public List<Tile> getKongs() {
        return kongs;
    }
    public List<Tile> getPungs() {
        return pungs;
    }
    public List<Tile> getChows() {
        return chows;
    }
    public List<Tile> getDiscards() {
        return discards;
    }
    // Additional methods for adding, sorting, discarding tiles...
}
```

By utilizing the Composite pattern, the Mahjong game can efficiently manage the relationships and operations on different types of tiles, ensuring that the game logic remains modular and scalable.

3.2 Creational

3.2.1 Singleton Pattern

The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. We used this pattern for configuration classes and game state management.

Game State Management and Configuration Classes

- *GameHandler and GameService:* These classes are implemented as singletons to manage the game state across multiple users and sessions.
- *Configuration Classes:* Classes like `SecurityBeansConfig`, `WebSecurityConfig`, and `WebSocketConfig` are also singletons to ensure they are only instantiated once.
- *GameController:* The `GameController` class, annotated with `@Controller`, is a singleton in the Spring context, ensuring it is instantiated only once and can be globally accessed.

Singleton Pattern Components

- *Singleton:* The Spring container manages these configuration and controller classes and their beans, ensuring each is a singleton.
- *Global Access Point:* Beans defined in these classes are accessible globally within the Spring application context.

3.2.2 Abstract Factory Pattern

The Abstract Factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes. This pattern was used to create different types of game objects such as tiles. In the game package, the TileFactory class is responsible for creating different types of tiles (bamboo, character, circle, wind, dragon).

1. Tile type

Code Structure The code defines a base class `Tile` and several subclasses representing different types of Mahjong tiles:

- `BambooTile`
- `CharacterTile`
- `DotTile`
- `DragonTile`
- `WindTile`

Each subclass inherits from the `Tile` base class and implements the `toString()` method to return a string representation of the tile.

Factory Method Pattern Components

- *Abstract Product:* The `Tile` class serves as the abstract product. It defines the interface that all concrete products must implement.
- *Concrete Product:* The concrete products are the subclasses of `Tile` (`BambooTile`, `CharacterTile`, `DotTile`, `DragonTile`, `WindTile`).
- *Creator:* The creator in this context is implicit. Each subclass of `Tile` can be viewed as a factory that creates instances of specific tile types.

Advantages

- Encapsulation of Creation Logic: The logic for creating specific types of tiles is encapsulated within each concrete class.
- Open/Closed Principle: The design adheres to the open/closed principle, allowing new types of tiles to be added without modifying existing code.

Improvement To better adhere to the Factory Method Pattern, a dedicated factory class can be introduced to manage the creation of tile objects. Below is an example of such a factory class:

Listing 3: TileFactory Class

```
public class TileFactory {
    public static Tile createTile(TileType type, int number) {
        return switch (type) {
            case BAMBOO -> new BambooTile(type, number);
            case CHARACTER -> new CharacterTile(type, number);
            case DOT -> new DotTile(type, number);
            case DRAGON -> new DragonTile(type, number);
            case WIND -> new WindTile(type, number);
            default -> throw new IllegalArgumentException("Unknown tile type");
        };
    }
}
```

Using the factory class, client code can create tile objects as follows:

Listing 4: Client Code Example

```
Tile bambooTile = TileFactory.createTile(TileType.BAMBOO, 0);
Tile characterTile = TileFactory.createTile(TileType.CHARACTER, 1);
```

3.3 Behavioral

3.3.1 Observer pattern

The Observer pattern is used to notify multiple objects about state changes. This is crucial for a real-time multiplayer game like Mahjong. In the game package, classes handling game state changes, such as GameHandler or GameService, notify observers (e.g., player interfaces) about game state changes.

JavaScript Code

The following JavaScript code demonstrates how to subscribe to different topics using the STOMP protocol. This code represents the observer part of the Observer pattern on the client side.

Listing 5: JavaScript Code for Observing Messages

```
stompClient.subscribe('/game/room/' + roomCode, function (message) {
    console.log('Message received: ' + message.body);
    updateAllPlayersHandCards();
});

stompClient.subscribe('/topic/roomDataChanged/' + roomCode, function (message) {
    console.log('Message received: ' + message.body);
    updateAllPlayersHandCards();
});

stompClient.subscribe('/topic/drawTileFeedback/' + roomCode, function (message) {
    console.log('Message received: ' + message.body);
});
```

Java Code

The following Java code snippets demonstrate how to implement the observer pattern on the server side using Spring Boot. The server processes events and sends messages to different topics that the clients subscribe to.

Spring Boot Controller

This controller handles messages related to drawing tiles and room data changes.

Listing 6: Spring Boot Controller for Handling Messages

```
@MessageMapping("/drawTile/{roomCode}")
public void handleDrawTileMessage(@Payload String message,
                                  @DestinationVariable String roomCode) {
    System.out.println(message);
    boolean success = gameService.handleDrawTileMessage(message);
    String feedback = success ? "Message received and processed successfully."
                              : "Message processing failed.";

    // Send feedback message to the topic
    simpMessagingTemplate.convertAndSend("/topic/drawTileFeedback/" + roomCode,
                                         feedback);
    // Correctly use the room code to call handleRoomDataChanged method
    handleRoomDataChanged(roomCode, feedback);
}

@MessageMapping("/roomDataChanged/{roomCode}")
@SendTo("/topic/roomDataChanged/{roomCode}")
public void handleRoomDataChanged(@DestinationVariable String roomCode,
                                  String message) {
    // Notify browser that data has changed
    System.out.println("Room data has changed in room: " + roomCode);
    simpMessagingTemplate.convertAndSend("/topic/roomDataChanged/" + roomCode,
                                         message);
}
```

Observer Pattern Analysis

Java Code Analysis

The Java code uses Spring Boot's WebSocket messaging support to implement the server side of the Observer pattern. Here's how it works:

Subject: `handleDrawTileMessage` Method

- **Event Handling**: When a draw tile message is received, this method processes it.
- **Notification**: Based on the success of processing the message, it sends feedback to the topic `/topic/drawTileFe` using `simpMessagingTemplate.convertAndSend`. This acts as the notification mechanism for observers subscribed to this topic.

- **State Change**: It calls `handleRoomDataChanged` to notify about the change.

Subject: `handleRoomDataChanged` **Method**

- **State Change Notification**: This method sends a message to the topic `/topic/roomDataChanged/{roomCode}` to notify all observers about the change in room data.

JavaScript Code Analysis

The JavaScript code on the client side subscribes to these topics to receive notifications.

- **Subscription to** `/game/room/{roomCode}`: The client subscribes to this topic to receive messages related to the game room. When a message is received, it logs the message and calls `updateAllPlayersHandCards` to update the hand cards of all players.
- **Subscription to** `/topic/roomDataChanged/{roomCode}`: This topic is used to receive notifications about changes in room data from the server. When data changes, the client receives a notification and updates all players' hand cards.
- **Subscription to** `/topic/drawTileFeedback/{roomCode}`: The client subscribes to this topic to receive feedback from draw tile actions. When a feedback message is received, it logs the message to the console.

Summary

The Observer pattern implementation shown here uses Spring Boot for the server-side logic and JavaScript for the client-side subscriptions. This pattern ensures that clients are notified in real-time about server-side events, allowing for efficient and flexible communication between the server and clients.

4 Test

4.1 Development Testing

4.1.1 Unit Testing

Objective: Test the functionality of individual program units or object classes.
Details:

- **Testing Goal**: Validate the functionality of each individual method and object class.
- **Involved Code**:
  - `ScoringSystemTest.java`: Primarily tests various methods of the `ScoringSystem` class, such as `uniformTile()`, `oneDragon()`, `isPairSequence()`, and `sevenPairs()`.
  - `HandTest.java`: Primarily tests various methods of the `Hand` class, such as `testAddCard()`, `testDiscard()`, `testCanChow()`, and `testIsValidMahjong()`.
- **Test Descriptions**:
  - `uniformTile()`: Tests the logic for whether all tiles in the hand are of the same type.
  - `oneDragon()`: Tests the logic for whether the hand forms a sequence from 1 to 9 in one suit.
  - `isPairSequence()`: Tests the logic for whether two tiles form a pair sequence.
  - `sevenPairs()`: Tests the logic for whether the hand consists of seven pairs.
  - `testAddCard()`: Tests the functionality of adding a card to the hand.
  - `testDiscard()`: Tests the functionality of discarding a card from the hand.
  - `testCanChow()`: Tests whether a "Chow" (a sequence of three consecutive numbers in the same suit) can be made.
  - `testIsValidMahjong()`: Tests whether the hand constitutes a valid Mahjong hand.

4.1.2 Component Testing

Objective: Test the interfaces of components.
Details:

- **Testing Goal**: Validate the functionality of component interfaces.
- **Involved Code**:
  - `HandTest.java`: Primarily tests the interface methods of the `Hand` component, such as `testDeepCopyHandca` `testAllCard()`, `testExecuteChows()`, and `testExecutePung()`.
- **Test Descriptions**:
  - `testDeepCopyHandcard()`: Tests the deep copy method of the hand, ensuring that the original hand and the copied hand are different objects but have identical content.
  - `testAllCard()`: Tests the functionality of retrieving all cards in the hand, including those added to Chow, Pung, and Kong.
  - `testExecuteChows()`: Tests the method for executing a "Chow" operation (forming a sequence of three consecutive numbers in the same suit).

- `testExecutePung()`: Tests the method for executing a "Pung" operation (forming a triplet of the same number).
- `testExecuteKong()`: Tests the method for executing a "Kong" operation (forming a quadruplet of the same number).

4.2 Release Testing

4.2.1 System Testing

Objective: Test the behavior of the entire system.
Details:

- **Testing Goal**: Ensure that all components and modules of the system work together to meet the overall functional requirements.
- **Involved Code**:
  - `LoginControllerTest.java`: Tests the user login and registration functionalities.
  - `GameControllerTest.java`: Tests the game creation and room management functionalities.
- **Test Descriptions**:
  - `testLogin()`: Tests the display and response of the login page.
  - `testRegister()`: Tests the display and response of the registration page.
  - `testDoRegister()`: Tests the functionality of registering a new user, ensuring correct storage of user data and proper redirection.
  - `testCreateRoom()`: Tests the functionality of creating a room, ensuring correct creation and proper redirection.

4.3 User Testing

4.3.1 Alpha Testing

Objective: Conduct initial testing by internal staff or users in a controlled environment.
Details:

- **Testing Goal**: Perform testing by the development team or company internal staff in a controlled environment to identify and fix major issues.
- **Involved Code**: The entire system, including all functionalities and components.
- **Test Descriptions**: Conduct comprehensive testing in a controlled environment, simulating real usage scenarios to identify and fix potential issues.

4.4 Beta Testing

Objective: Conduct testing by real users in a production environment.
Details:

- **Testing Goal**: Allow real users to use the system in a real environment, collecting feedback and issue reports.

- **Involved Code**: The entire system, including all functionalities and components.
- **Test Descriptions**: Release the system for real users to use, collect user feedback and issue reports, and further improve and optimize the system.

5 Refactoring

5.1 Duplicate code

1. Original Code

In the original code, the `startGame()` and `endGame()` methods contain duplicate code for initializing the game state, including creating the tile pile, shuffling tiles, and distributing initial tiles.

Listing 7: Original Code

```
public void startGame() {
    createPlayers();
    creatTilePile();
    shuffleTiles();
    distributeInitialTiles();
    Random random = new Random();
    dicenum = random.nextInt(10) + 2;
    indexnum = dicenum % 4;
    checknum = indexnum;
    players[indexnum].setIsbanker(true);
}

public void endGame() {
    for (Player player : players) {
        player.playAgain();
    }
    drawTile = null;
    discradTile = null;
    creatTilePile();
    shuffleTiles();
    distributeInitialTiles();
    Random random = new Random();
    dicenum = random.nextInt(10) + 2;
    indexnum = dicenum % 4;
    checknum = indexnum;
    players[indexnum].setIsbanker(true);
}
```

Issues with the Original Code

- **Code Duplication**: The same logic for initializing the game is repeated in both `startGame()` and `endGame()` methods.
- **Maintenance Effort**: Any changes to the initialization logic need to be made in multiple places, increasing the risk of errors.

2. Refactored Code

The refactored code introduces a new method `initializeGame()` that consolidates the duplicated initialization logic. Both `startGame()` and `endGame()` methods call this new method, ensuring that the game initialization logic is defined in one place.

Listing 8: Refactored Code

```
public void startGame() {
    createPlayers();
    initializeGame();
}

public void endGame() {
    for (Player player : players) {
        player.playAgain();
    }
    drawTile = null;
    discradTile = null;
    initializeGame();
}

private void initializeGame() {
    creatTilePile();
    shuffleTiles();
    distributeInitialTiles();
    Random random = new Random();
    dicenum = random.nextInt(10) + 2;
    indexnum = dicenum % 4;
    checknum = indexnum;
    players[indexnum].setIsbanker(true);
}
```

Benefits of Refactored code

- **Reduced Code Duplication**: The common initialization logic is consolidated into a single method, reducing redundancy.
- **Easier Maintenance**: Changes to the initialization logic only need to be made in one place, reducing the effort required to maintain the code.
- **Improved Consistency**: Using a single method for initialization ensures that the game state is consistently set up in both `startGame()` and `endGame()` methods.
- **Enhanced Readability**: The refactored code is cleaner and easier to read, making it more understandable for developers.

5.2 Switch (case) statements

1. Tile type function

The original code is as follows:

Listing 9: Original Code with if-else Statements

```
public static TileType transTypeFromMessage(String s) {
    String string = s.substring(0, s.length() - 1);
    System.out.println(string);
    if (string.equals("BAMBOO")) {
```

```
        return TileType.BAMBOO;
    } else if (string.equals("CHARACTER")) {
        return TileType.CHARACTER;
    } else if (string.equals("DOT")) {
        return TileType.DOT;
    } else if (string.equals("WIND")) {
        return TileType.WIND;
    } else if (string.equals("DRAGON")) {
        return TileType.DRAGON;
    }
    return null;
}
```

Issues with the Original Code

The primary issues with the above approach are:

- **Readability**: The sequence of `if-else` statements can be difficult to follow, especially as the number of conditions increases.
- **Maintainability**: Adding new cases or modifying existing ones can be error-prone and cumbersome.

Refactored Code with Switch Statement

According to best practices discussed in our lectures, the `if-else` chain can be refactored into a `switch` statement for better clarity and performance.

Listing 10: Refactored translateType Method

```
public static int translateType(TileType type){
    switch (type) {
        case BAMBOO:
            return 0;
        case CHARACTER:
            return 1;
        case DOT:
            return 2;
        case DRAGON:
            return 3;
        case WIND:
            return 4;
        default:
            System.out.println("Non-existent tile, what happen?");
            return -1;
    }
}
```

Benefits of Using Switch Statements

- **Readability**: Switch statements are more readable compared to a series of `if-else` statements, especially when dealing with multiple cases.
- **Maintainability**: It is easier to add new cases and modify existing ones.

- **Performance**: Switch statements can be more efficient in terms of execution, particularly when there are many cases to evaluate.

Conclusion

Refactoring the code to use switch statements simplifies the structure, improves readability, and enhances maintainability. This exercise demonstrates the value of refactoring and adhering to best practices in software development.

6 Git Repositories

https://github.com/Mystour/mahjong.git

7 Timeline

Time	Task arrangement
Week 8	Discuss game requirements, standards, implementation, etc., and complete Phase-1 assignment, configure GitHub environment and create public project
Week 9	Make the main part of the game, make the basic GUI page, and ensure the most basic functions of the game
Week 10	Improve the GUI page of the game, improve the content of the game, such as score statistics, ranking and other functions
Week 11	Add animation to the main game screen, add the start screen, score board screen,teaching screen, etc
Week 12	If we have more time, we decide to make a robot player, which can simulate the basic operation of ordinary players, let players can play single
Week 13	Add basic database and network functionality to make players can play online
Week 14	Make overall game reports of Phase-2, add comments to the all code, and check for bugs
Week 15	Finally check everything and upload the project to Brightspace

8 Member contributions & Weekly Update

8.0.1 Chen Ziang (Group Leader)

1. **Network and WebSocket Deployment**: Responsible for configuring and deploying WebSocket for real-time updates, ensuring seamless communication between the server and clients.

2. **Project Framework Design and Setup**: Led the design and initial setup of the project framework, ensuring a solid foundation for development.

3. **Database Design and Implementation**: Designed and implemented the database schema, ensuring efficient data storage and retrieval. Also responsible for database-related configurations.

4. **Frontend Implementation**: Worked on the frontend design and development, ensuring a user-friendly and visually appealing interface.

8.0.2 Wang Shuhan

1. **Backend Game Logic Implementation**: Developed the core game logic, including rules for drawing and discarding tiles, and ensuring the game state transitions correctly.
2. **API Development for Frontend Integration**: Created backend endpoints and services to provide interfaces for the frontend, facilitating smooth data flow between the client and server.
3. **Partial Frontend Logic Implementation**: Assisted in implementing frontend logic to ensure proper integration with backend services.

8.0.3 Chen Hongyue

1. **Frontend Logic Implementation**: Developed the main frontend logic, including handling user interactions and updating the user interface based on game state changes.
2. **JavaScript File Logic**: Implemented and maintained JavaScript files to manage client-side logic and interactions, ensuring a responsive and dynamic user experience.
3. **Backend Implementation**: Assisted in backend development, including handling game state changes and database interactions.

8.0.4 Sun Lingrui

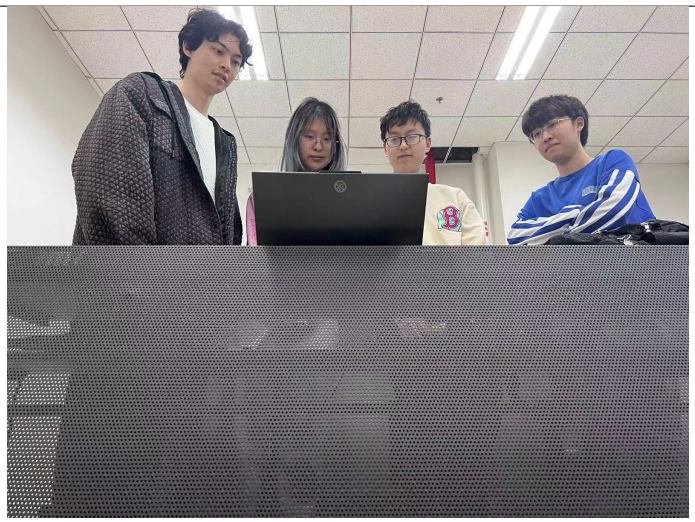
1. **Frontend Implementation**: Worked on the frontend design and development, ensuring a user-friendly and visually appealing interface.
2. **Backend Implementation**: Contributed to backend development, focusing on game state management and server-side logic.
3. **Testing and Debugging**: Conducted thorough testing of both frontend and backend components, writing test cases and debugging issues to ensure the stability and reliability of the application.

Our team demonstrated excellent collaboration and division of labor. Chen Ziang took the lead on WebSocket integration, ensuring real-time updates and a seamless user experience. In Week 5, the team focused on implementing the multi-player room functionality where players could join a room to play Mahjong and the progress bar updated as each player joined, showcasing the real-time capabilities. Each member took on specific responsibilities while ensuring seamless integration of their work with the rest of the team. Our combined efforts led to the successful completion of the Mahjong game project, complete with real-time updates, robust game logic, and a user-friendly interface.

Table 1: Weekly Update

Group06 Week7 Complete the Game Requirement	
Item	**Description**
1	What our group has done since last week: • The direction and purpose of the overall project was discussed and the plan was to develop the Beijing version of the mahjong system, determines the language and framework for writing the code, and formulates the game requirement.
2	Zang Chen: Identify and write the user requirement. Shuhan Wang: Identify and write the system requirement. Lingrui Sun: Identify and write the functional requirement. Hongyue Chen: Identify and write the Non-funtional requirement.
	
3	We have the following questions that we would like to discuss: • How to design a mahjong system with a Beijing version of the mixed card. • How to design a nice front-end
4	What we plan to do next week: • Complete the UML design • Refine each person's division of labor

<table>
<tr><td colspan="2">Group06 Week8

Complete uml diagrams and create github projects</td></tr>
<tr><td>Item</td><td>Description</td></tr>
<tr><td>1</td><td>What our group has done since last week:
We discussed various standards and rules of mahjong game, discussed the logic of writing code, classification, made various UML diagrams, completed Phase-1 assignment, and configured GitHub environment, and create public project</td></tr>
<tr><td>2</td><td>Ziang Chen: Design the case and sequence diagram. Create Github environment.
Shuhan Wang: Design the class diagram. Make a pdf of project requirements.
Lingrui Sun: Design the State diagram. Customize timeline.
Hongyue Chen: Design the flow chart. Create Github environment.</td></tr>
<tr><td colspan="2">  </td></tr>
<tr><td>3</td><td>We have the following questions that we would like to discuss:
How to achieve four computers online play mahjong game
What kind of data structure is used to store mahjong sets</td></tr>
<tr><td>4</td><td>What we plan to do next week:
Refine the specific classes in the project code, briefly carry the framework.
Begin to overcome the code requirements.</td></tr>
</table>

Group06 Week9

The division of labor between the front and back end of the code, Preliminary writing code

Item	Description
1	What our group has done since last week: • We created each pack, class, and initially decided that our mahjong game would be played by four people on four computers, not four people on one computer. Refine each person's division of labor
2	Ziang Chen: Create the package, the class name, Make an online server,Create databases Shuhan Wang: Write the code in the Tile package, improve the overall code. Lingrui Sun: Write mahjong scoring system, improve the overall code. Hongyue Chen: Write the code in the player package, improve the overall code.
	
3	What we plan to do next week: • Continue to complete the unfinished code.

<table>
<tr><td colspan="2">Group06 Week10

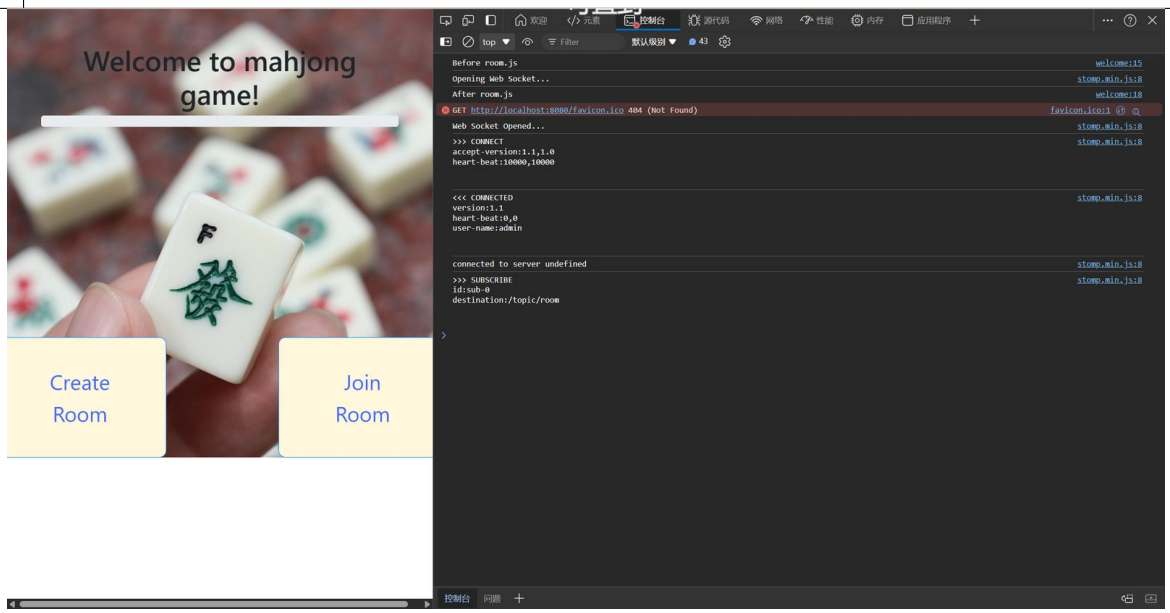
Core Functionality Development</td></tr>
<tr><td>Item</td><td>Description</td></tr>
<tr><td>1</td><td>What our group has done since last week:
Developing core backend functionalities and integrating login/registration systems.</td></tr>
<tr><td>2</td><td>Ziang Chen: Configured WebSocket for real-time updates and implemented both server-side and client-side WebSocket logic to handle real-time game updates.
Shuhan Wang: Started working on the game logic, including the rules for drawing and discarding tiles.
Lingrui Sun: Started working on the game logic, including the rules for is valid mahjong.
Hongyue Chen: Started working on the game logic, including the rules for player actions like chow pung kung.</td></tr>
<tr><td colspan="2">  </td></tr>
<tr><td>3</td><td>What we plan to do next week:
Implement real-time updates and WebSocket integration.</td></tr>
</table>

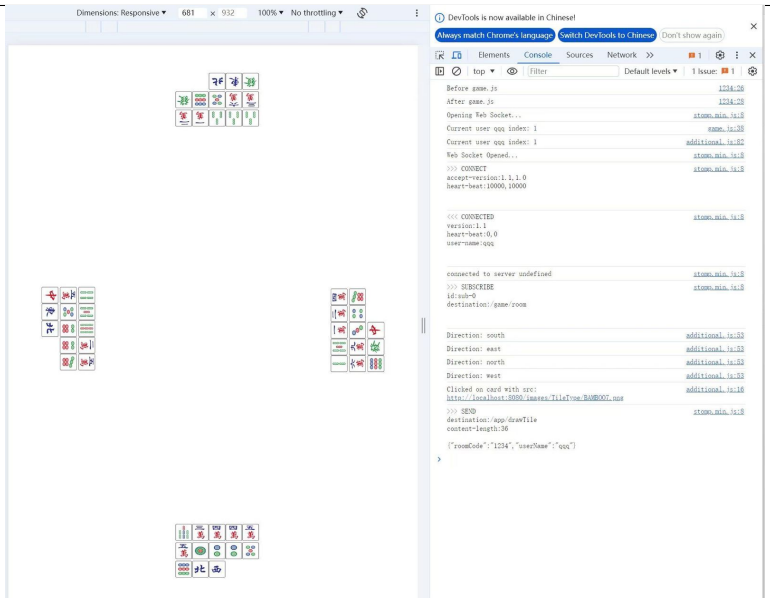
<table>
<tr><td colspan="2">Group06 Week11

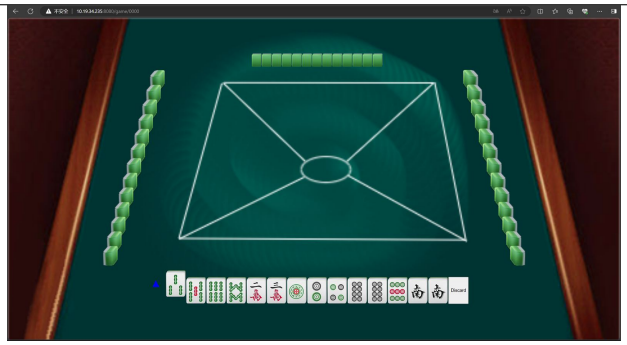
WebSocket Integration and Real-Time Updates</td></tr>
<tr><td>Item</td><td>Description</td></tr>
<tr><td>1</td><td>What our group has done since last week:
Implementing WebSocket for real-time game updates.</td></tr>
<tr><td>2</td><td>Ziang Chen: focusing on game state management and WebSocket integration.
Shuhan Wang: Developed the game controller logic to docking backend logic
Lingrui Sun: Assisted in the integration of WebSocket with the game logic.
Hongyue Chen: Developed the game controller logic to manage player actions and game state changes.</td></tr>
<tr><td colspan="2">  </td></tr>
<tr><td>3</td><td>What we plan to do next week:
Implement multi-player room functionality and progress tracking</td></tr>
</table>

Group06 Week12

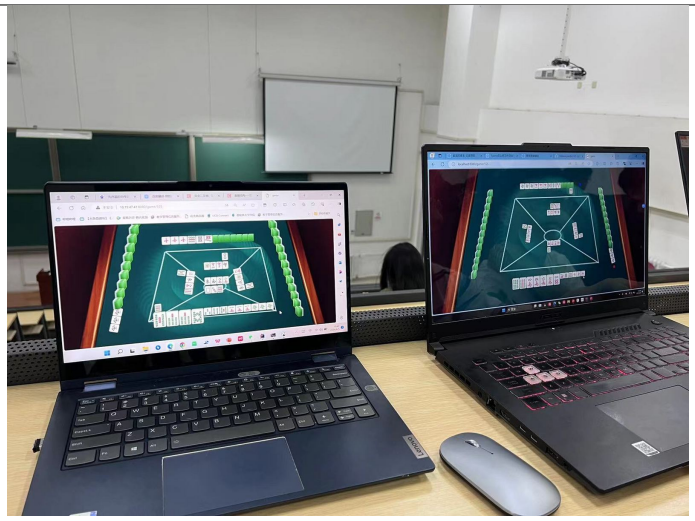
Multi-Player Room and Progress Tracking

Item	Description
1	What our group has done since last week: • Implementing multi-player room functionality and progress
2	Ziang Chen: Developed the real-time progress bar that updates as each player joins the room, using WebSocket for real-time updates. Shuhan Wang: Worked on the game logic to ensure it handles multiple players correctly. Lingrui Sun: Assisted in testing the multi-player functionality and ensuring smooth transitions as players join the room. Hongyue Chen: Implemented the functionality for players to enter a game room and start playing Mahjong.
	
3	What we plan to do next week: • Implement multi-player room functionality and progress tracking

Group06 Week13 Multi-Player Room and Progress Tracking	
Item	**Description**
1	What our group has done since last week: • Implementing multi-player room functionality and progress
2	Ziang Chen: Implemented the functionality for players to enter a game room and start playing Mahjong. Developed the real-time progress bar that updates as each player joins the room, using WebSocket for real-time updates. Shuhan Wang: Assisted in testing the multi-player functionality and ensuring smooth transitions as players join the room. Lingrui Sun: Assisted in testing the multi-player functionality and ensuring smooth transitions as players join the room. Hongyue Chen: Worked on the game logic to ensure it handles multiple players correctly.
	
3	What we plan to do next week: • Design and develop the user interface.

Group06 Week14 User Interface Development	
Item	**Description**
1	What our group has done since last week: • User Interface Development
2	Ziang Chen: Designed the user interface for the game lobby, player dashboard, and game board using HTML, CSS, and JavaScript. Integrated the UI with the WebSocket for real-time updates. Shuhan Wang: Worked on integrating the game board UI with the backend game logic. Lingrui Sun: Developed the UI for displaying player stats and game history. Hongyue Chen: Worked on improving the responsiveness and usability of the user interface.
	
3	What we plan to do next week: • Conduct extensive testing and debugging.

<table>
<tr><td colspan="2">Group06 Week15

Testing and Debugging(some Refactoring)</td></tr>
<tr><td>Item</td><td>Description</td></tr>
<tr><td>1</td><td>What our group has done since last week:
Conducting comprehensive testing and fixing bugs.
Refactoring some duplicated codes.</td></tr>
<tr><td>2</td><td>Ziang Chen: Conducted unit testing for the backend logic, WebSocket communication, and frontend interactions. Fixed identified bugs.
Shuhan Wang: Create project reports
Lingrui Sun: Create project reports
Hongyue Chen: Focused on testing the game logic and ensuring the correctness of game state transitions.</td></tr>
<tr><td colspan="2">  </td></tr>
<tr><td>3</td><td>What we plan to do next week:
Commit the project, yeah</td></tr>
</table>