ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

MindRoll: Distributed Multiplayer Dice Game

Yiming Li, Tianyu Qu, Jing Yang

Catalogue

Overall Design

Turn-based multiplayer dice game

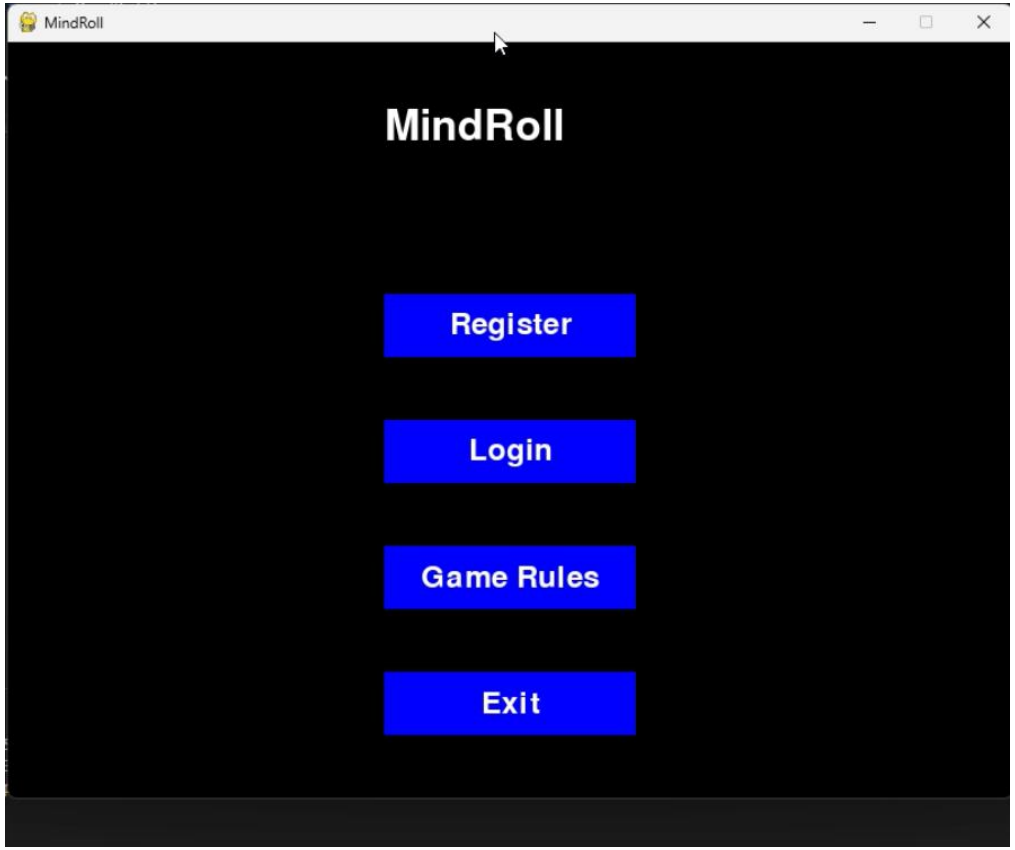
🎮 **Graphical User Interface:** Built with Pygame for an engaging user experience.

👥 **Multiplayer Support:** Players can join rooms and play together in real time.

🔐 **User Authentication:** Secure registration and log in with password hashing and token-based sessions.

💾 **Persistent Storage:** User data and session tokens are stored in MongoDB.

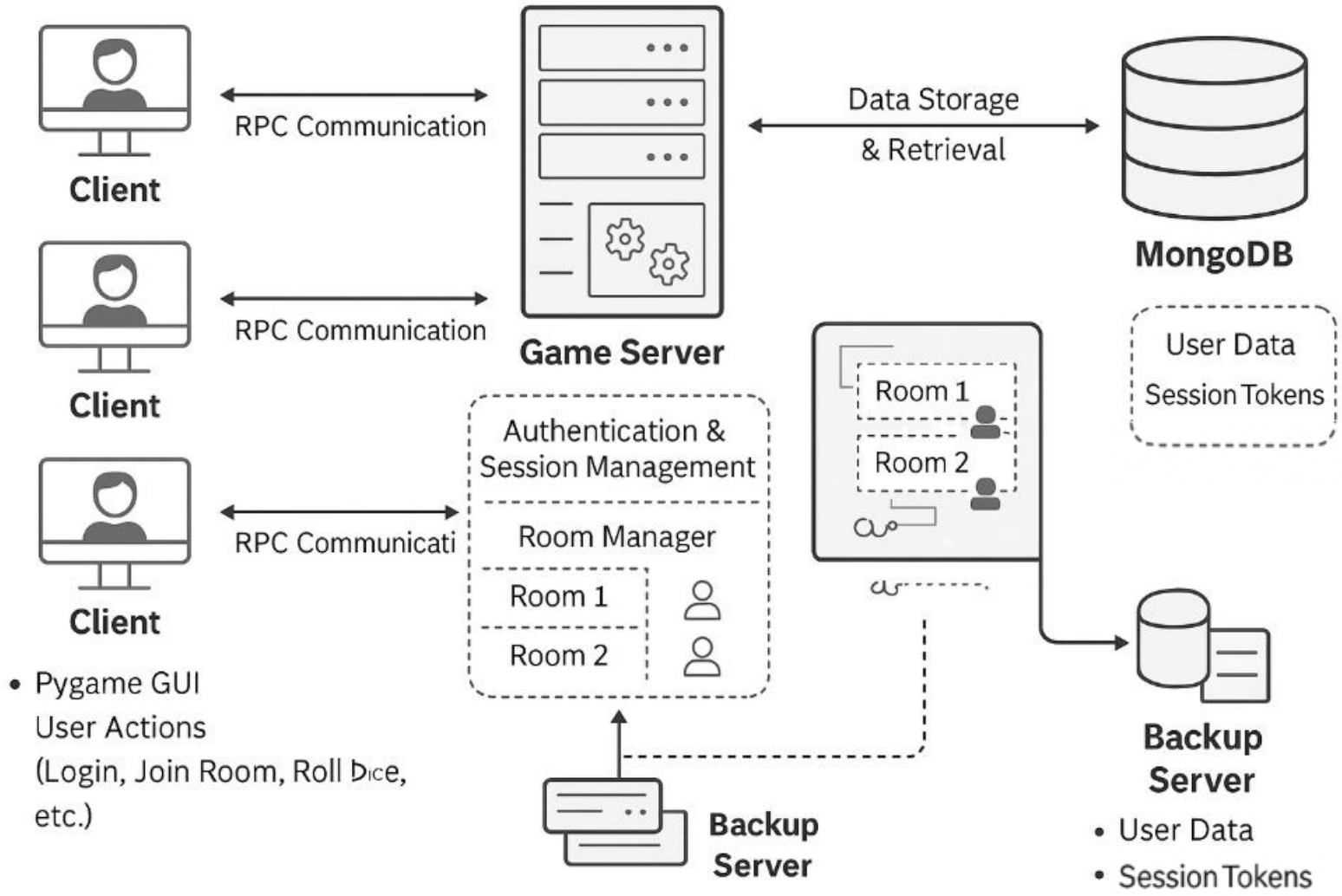
🔁 **Reconnection Support:** Players can reconnect and resume their game if disconnected.



System Architecture

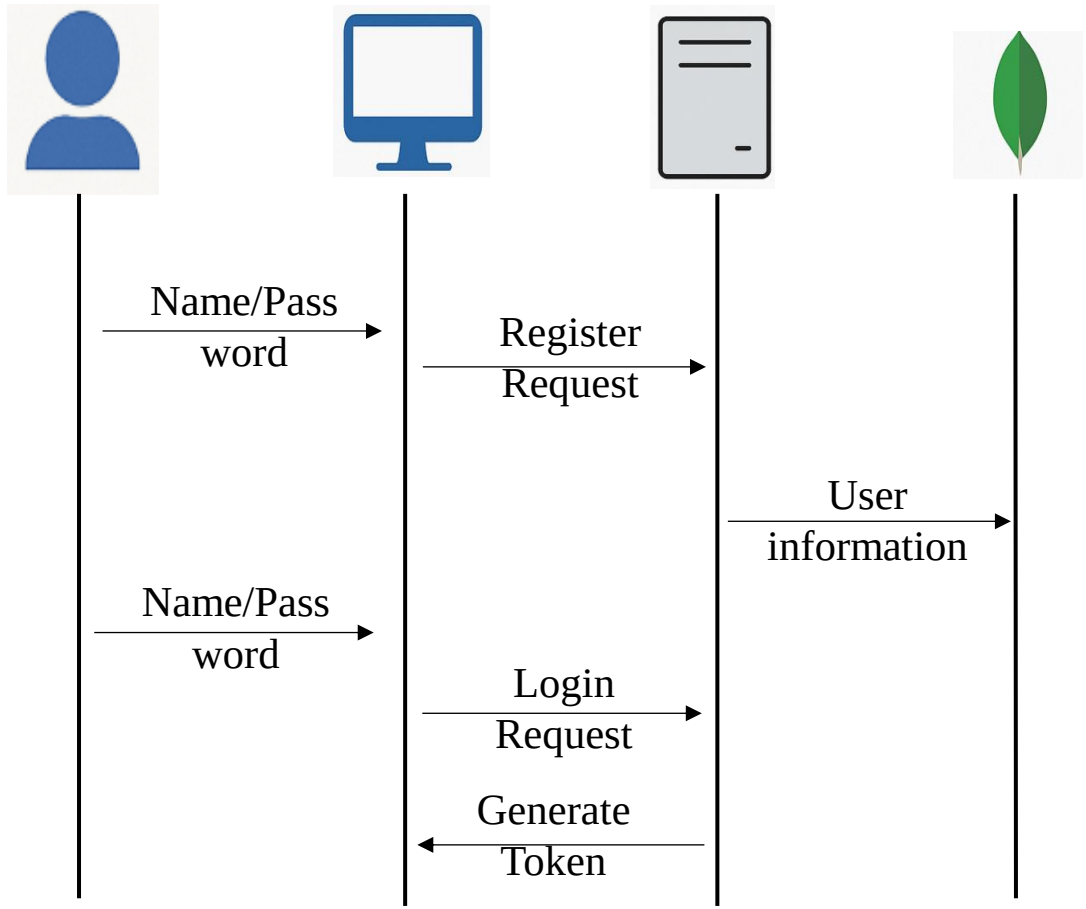
Client Server Architecture via RPC

•**Client:**
Handles user interface (Pygame GUI)
Sends user actions to the server
Receives game state updates from server
•**Server:**
Manages game logic, rooms, player states
Authenticates users and manages sessions
Stores persistent data in MongoDB
Handles reconnection and backup logic

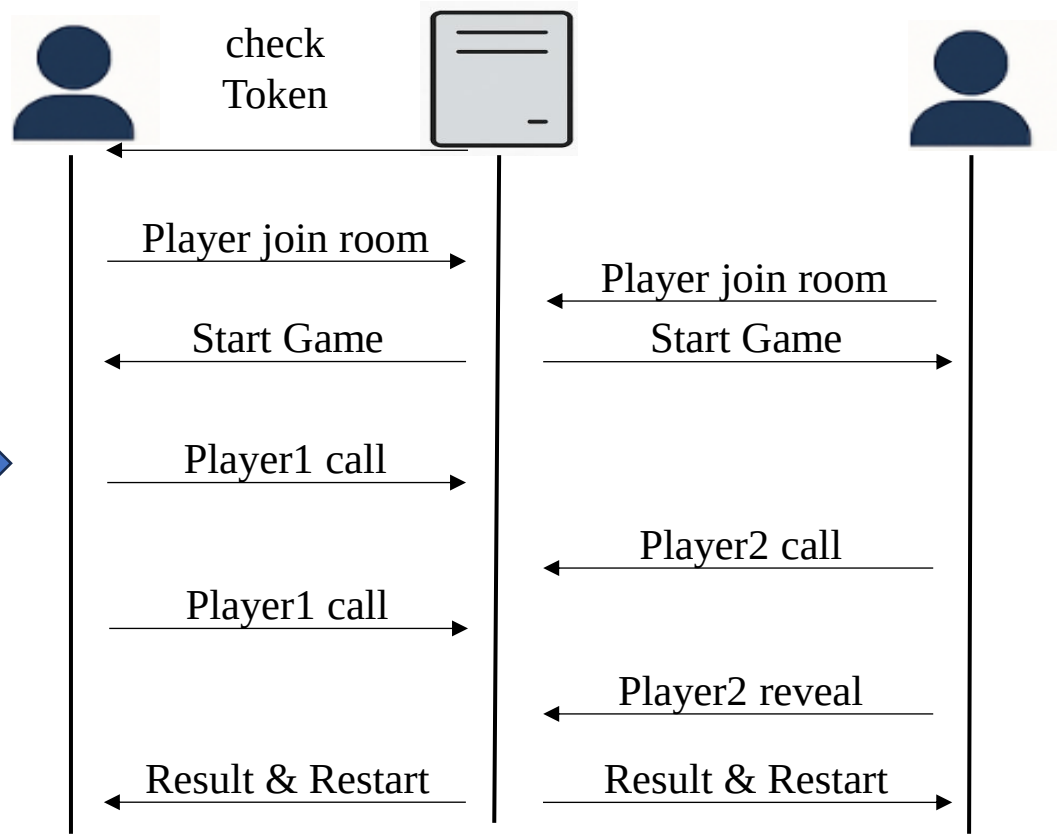


Game Logic and Interaction

Login Logic



Game Logic



Core Distributed Concepts

Authentication

Token-Based Authentication

Session Management:

•Tokens have expiration (TTL) to enhance security.

•In memory session state

Password Hashing with Salting

RPC

Client-Server Communication:

•All interactions between client and server use RPC.

•Communication Protocol: TCP

•Message format: JSON

•Client-side polling to simulate real-time updates

Decoupling

Sync

Game State Management:

•The server maintains the authoritative state for each game room.

•When a player takes an action (e.g., rolls dice), the server updates the state and broadcasts.

Consistency:

•Ensures all players see the same game state.

•Handles late joiners and reconnections.

Fault Tolerance

Reconnection Support:

•If a client disconnects (e.g., network failure), it can reconnect using its token

•120 seconds timeout

•The server restores the player's state and allows rejoin.

Backup Server:

•A backup server deployed to take over in case the main server fails.

•Real-time sync with primary server

•Failover-ready architecture

Test & Validation

Unit Testing:

•Individual modules and functions are tested in isolation.

•Examples: authentication logic, game state transitions, utility functions.

Integration Testing:

•Tests interactions between components, client-server communication and database operations.

•Ensures modules work together as expected.

End-to-End Testing:

•Simulates real user scenarios from registration to gameplay.

•Validates the complete workflow and user experience.

Manual Testing:

•Edge Case like empty username, incorrect password, server crash, reconnect, leave room.

•Validates the complete workflow and user experience, also test UI experience .

Test Coverage：

•Authentication and token management

•Game logic (dice rolling, turn management, win/loss conditions)

•Room creation and joining

•Client-server message exchange

•Reconnection and state recovery

Demo

https://drive.google.com/file/d/15Pi-7NfqE3VXJuQ4OeM9TMEQ1zieuIZN/view?usp=sharing

Thanks for listening