ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Hanabi Multiplayer

Distributed Systems Final Project

Candidates: Gabriele Santi, Blagoja Savevski, Pengyue Xu

Supervisor: Prof. Andrea Omicini

Co-supervisor: Giovanni Ciatto

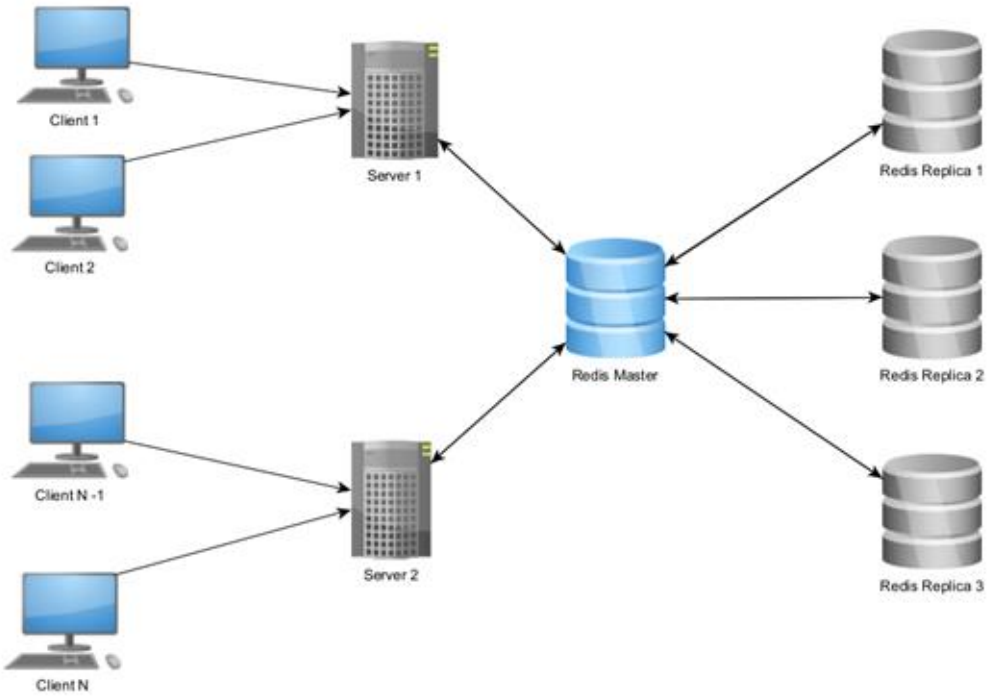
Master's Degree in Intelligent Embedded Systems cod. 6699
Academic Year 2025-26 | Department of Computer Science and Engineering

Distributed system architecture

Layered architecture : separation of concerns

- Presentation tier : Client
- Logic tier : Server
  - Domain level
  - Application level
  - Presentation level
- Data tier : Database

Infrastructure



- Client process on player machine - initial handshake, communication with spawned server
- Server process on server machine (container) - receives commands, sends events back
- Master process - single source of truth

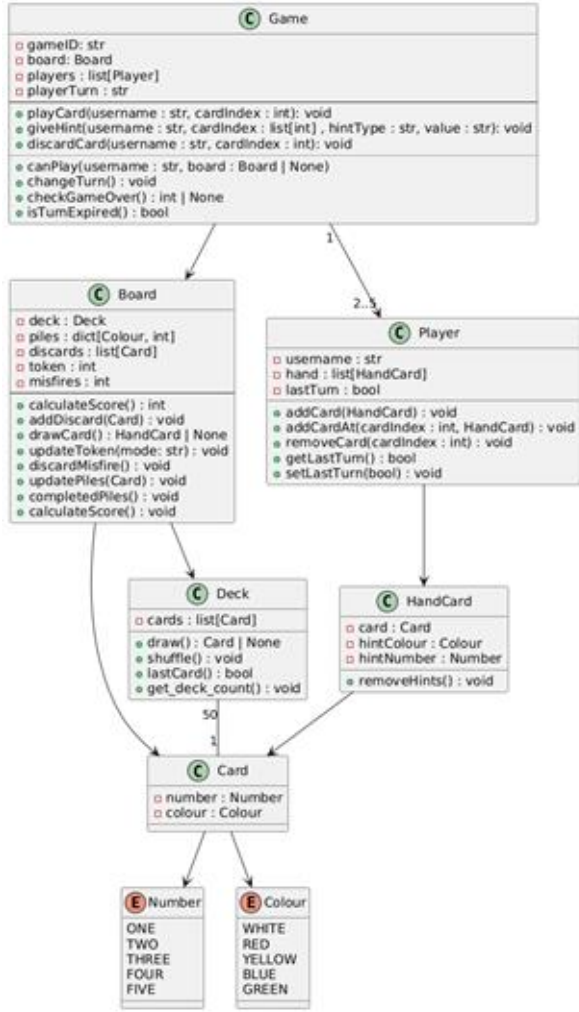
Motivations and reasons for distribution

- Enable multiplayer gaming in a distributed environment scenario
- Resource sharing : consistency and replicated state; concurrent play
- Fault tolerance
  - Application level
    - Retry mechanism
    - Timeout mechanism
  - Data tier
    - Redis master and replica
- Coordination and consistency
- Scalability
- Security, Authentication, Trust
- Performance, responsiveness

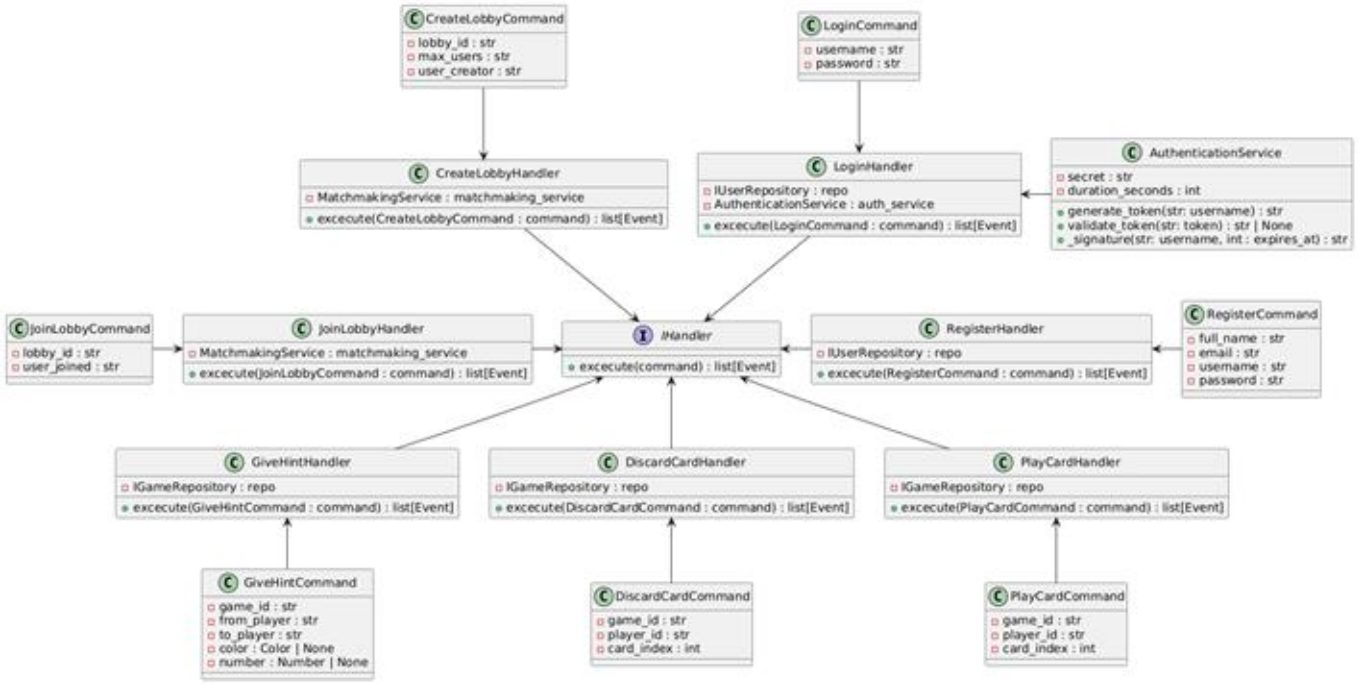
Domain

Division to address roles and responsibilities

- Game: execute moves based on the game's rules, update internal state
- Board: cards, misfires, tokens
- Player != User → dataclasses
  - User : authentication/authorization mechanisms
  - Player : User in an ongoing game
- Note at isTurnExpired()

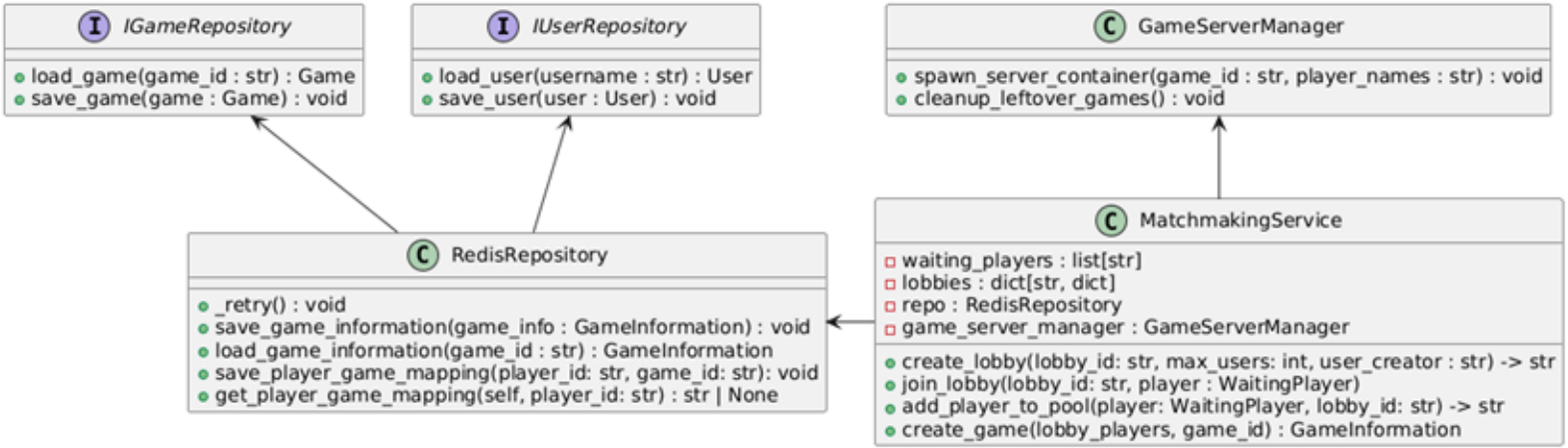


Application - 1



- Decoupling between action and logic.
- IHandler as a contract.
- Executing pure DTO commands

Application - 2



Handlers for interactions with the repository :

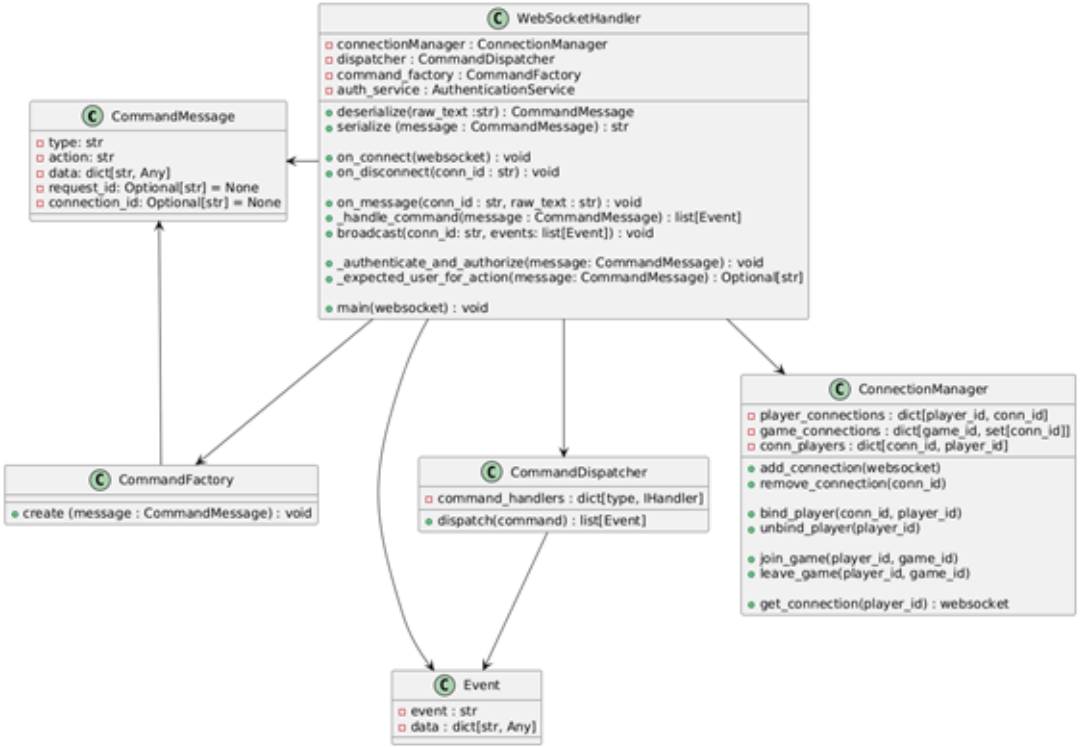
- IGameRepository
- IUserRepository

or external services :

- MatchmakingService
- AuthenticationService

Note the _retry method for interacting with Redis : Exponential backoff + jitter

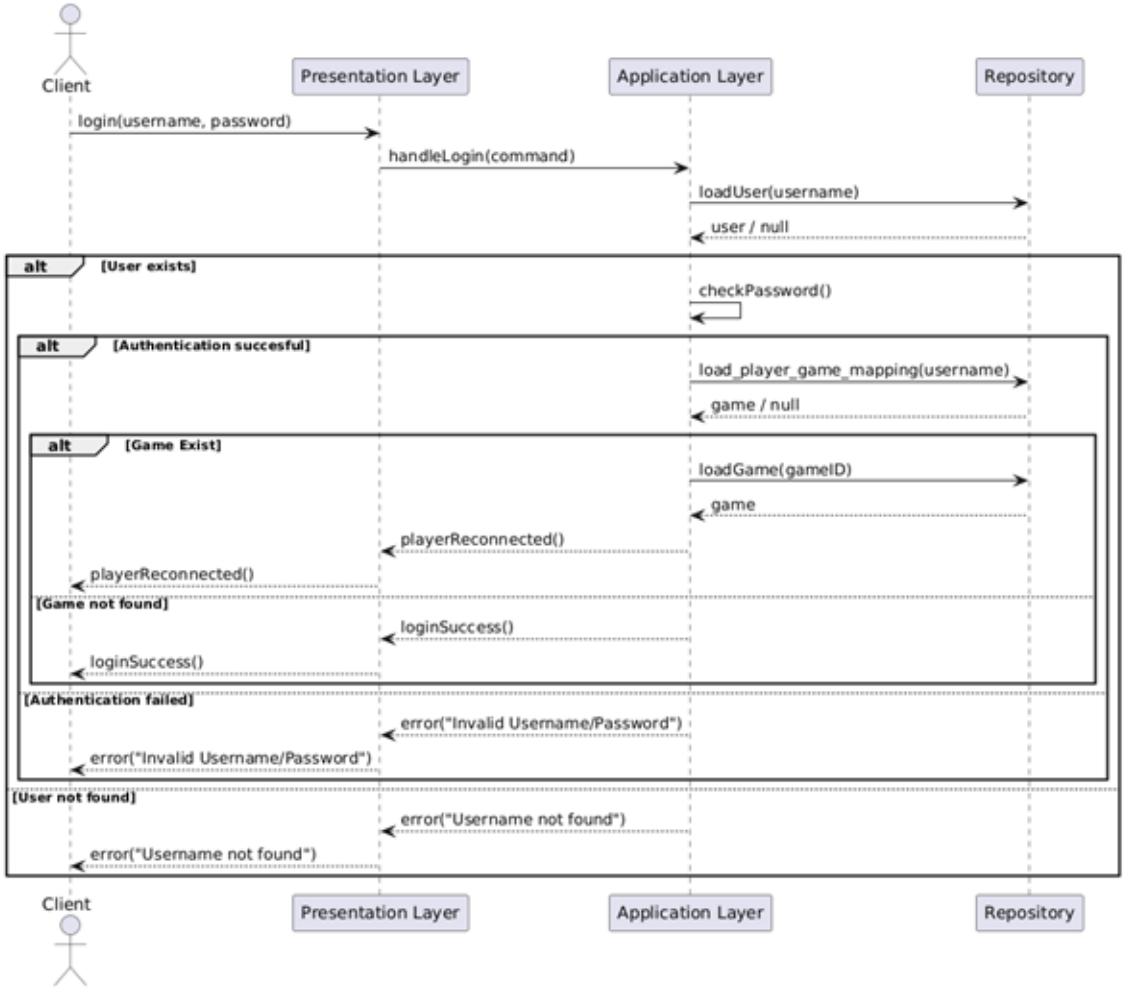
Presentation



- Pattern dispatcher : avoid if/else
- Pattern factory : delegate creation of objects

- WebSocketHandler : communication orchestrator

- Command : generic DTO that captures raw data
- Event : representation of the system's reactive output



Login sequence diagram

Reconnection mechanism is bounded

Game exist → reconnection

Game doesn't exist → lobby selection

Fault tolerance & Availability

- Persistent Websocket connection - fault tolerance in communication
- Timeout mechanism in Game to prevent blocking due to inactivity
- Retry mechanism when reaching Redis
- Sentinel promotes replica to master if master is determined shut down

- Caching a local copy in client
- Load distributed to multiple server instances
- Priority to consistency over availability in the case of network partition

Implementation

- Websockets communication protocol - persistent, reliable, manageable
- Docker container network for backend components
- JSON in-transit data format ex. serialization of Game
- Redis KV database
- Python, FastAPI backend, Uvicorn server
- Pytest unit, integration tests
- React, TypeScript, Vite front end
- Docker Compose to deploy backend

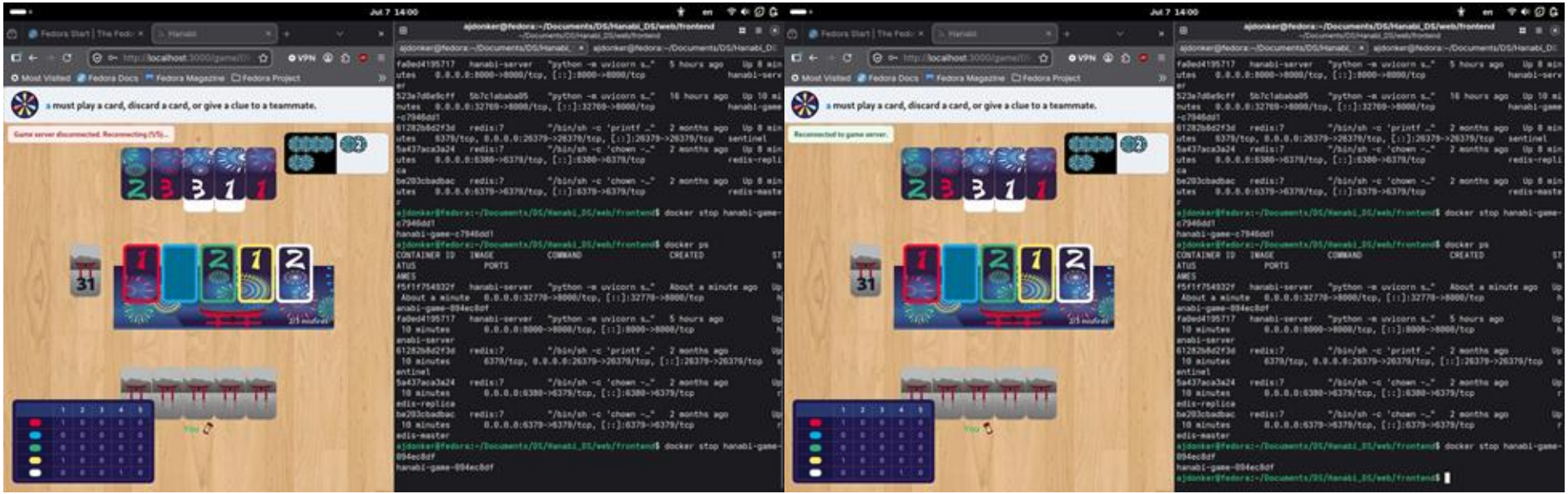
Validation

Layered architecture allows for easy boundaries for unit testing.

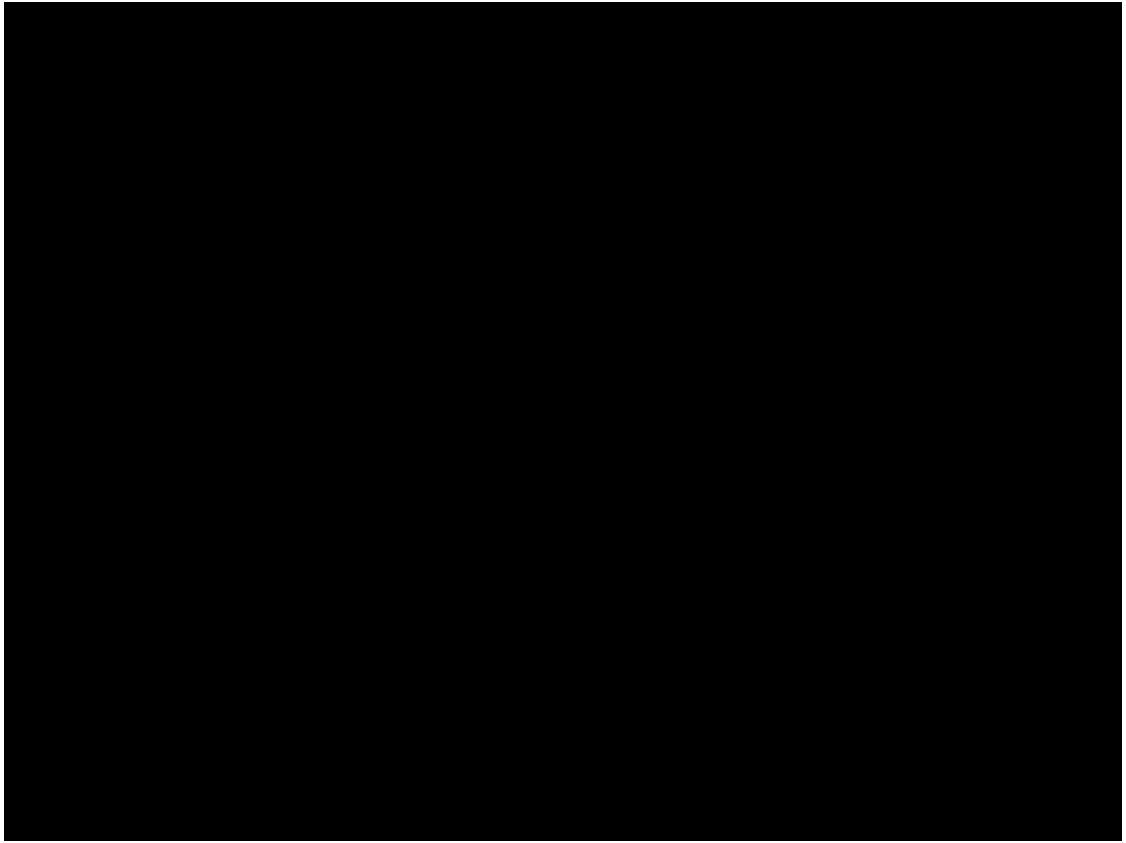
- Unit tests for domain classes to validate game rules are properly implemented
- Unit tests for matchmaker operations with mocked classes that interact with it
- Client-server communication flow completely tested in all possible scenarios
- Integration test of MatchmakingService + GameServerManager
- End to end (manual) test of complete flow; demo provided
- Manual integration acceptance tests that check multi component scenarios ex. checking whether game state update is received by all clients

Demo

Container failure into reconnection



Server temporarily unreachable, client reconnection



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Thank you for your attention

Gabriele Santi, Blagoja Savevski, Pengyue Xu