Rubamazzo

PROJECT FOR THE COURSE OF DISTRIBUTED SYSTEMS

MADINA KENTPAYEVA

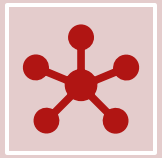
MADINA.KENTPAYEVA@STUDIO.UNIBO.IT

Introduction

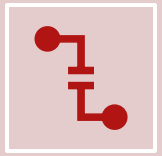
What is Rubamazzo?

A traditional Italian card game where players capture cards from the table by matching values or stealing opponents' decks.

The Idea Behind Rubamazzo Distributed



Multiplayer over network



Fault-tolerant & reconnectable

Managed via CLI + REST APIs

Design

System Architecture

Akka ActorSystem

RESTful API

Modular Logic

RESTful API Design

/game/createGame

/game/join/{gameId}/{playerName}

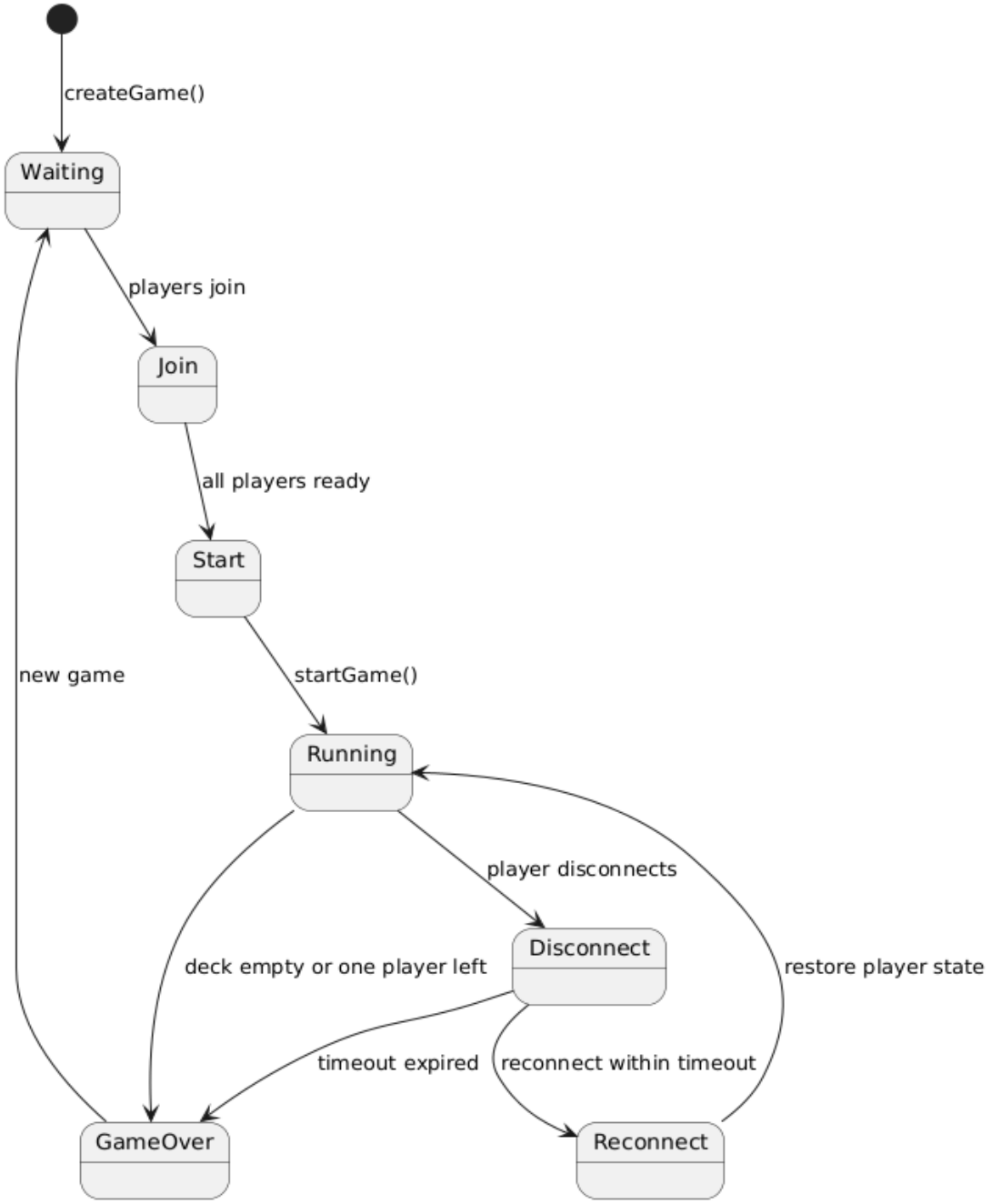
/game/start/{gameId}

/game/makeMove/{gameId}

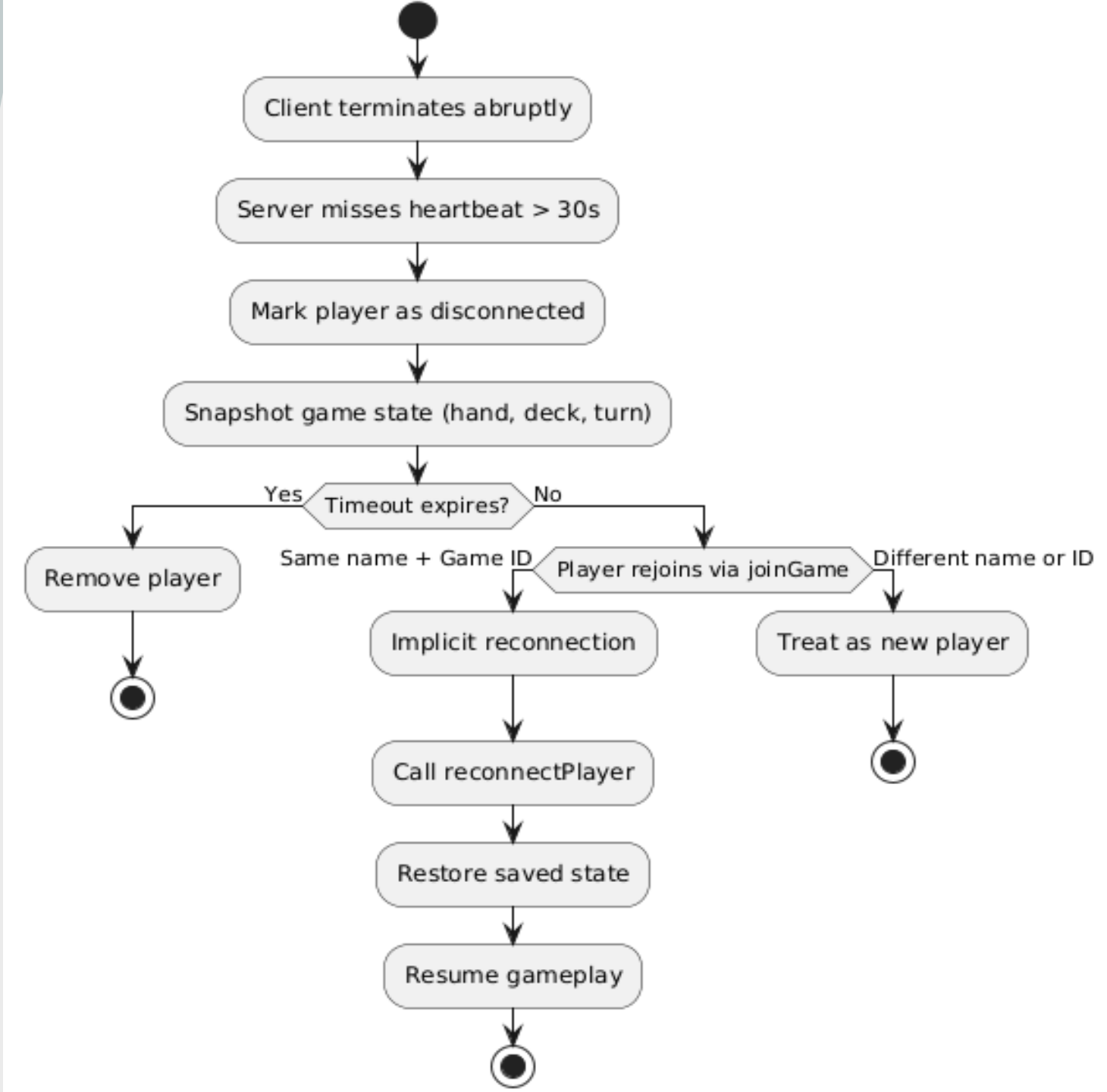
/game/gameState/{gameId}/{playerName}

/game/disconnectPlayer/{gameId} &
/reconnectPlayer/{gameId}

Game Flow


createGame()
Waiting
players join
Join
all players ready
Start
startGame()
new game
Running
player disconnects
Disconnect
deck empty or one player left
restore player state
timeout expired
reconnect within timeout
GameOver
Reconnect

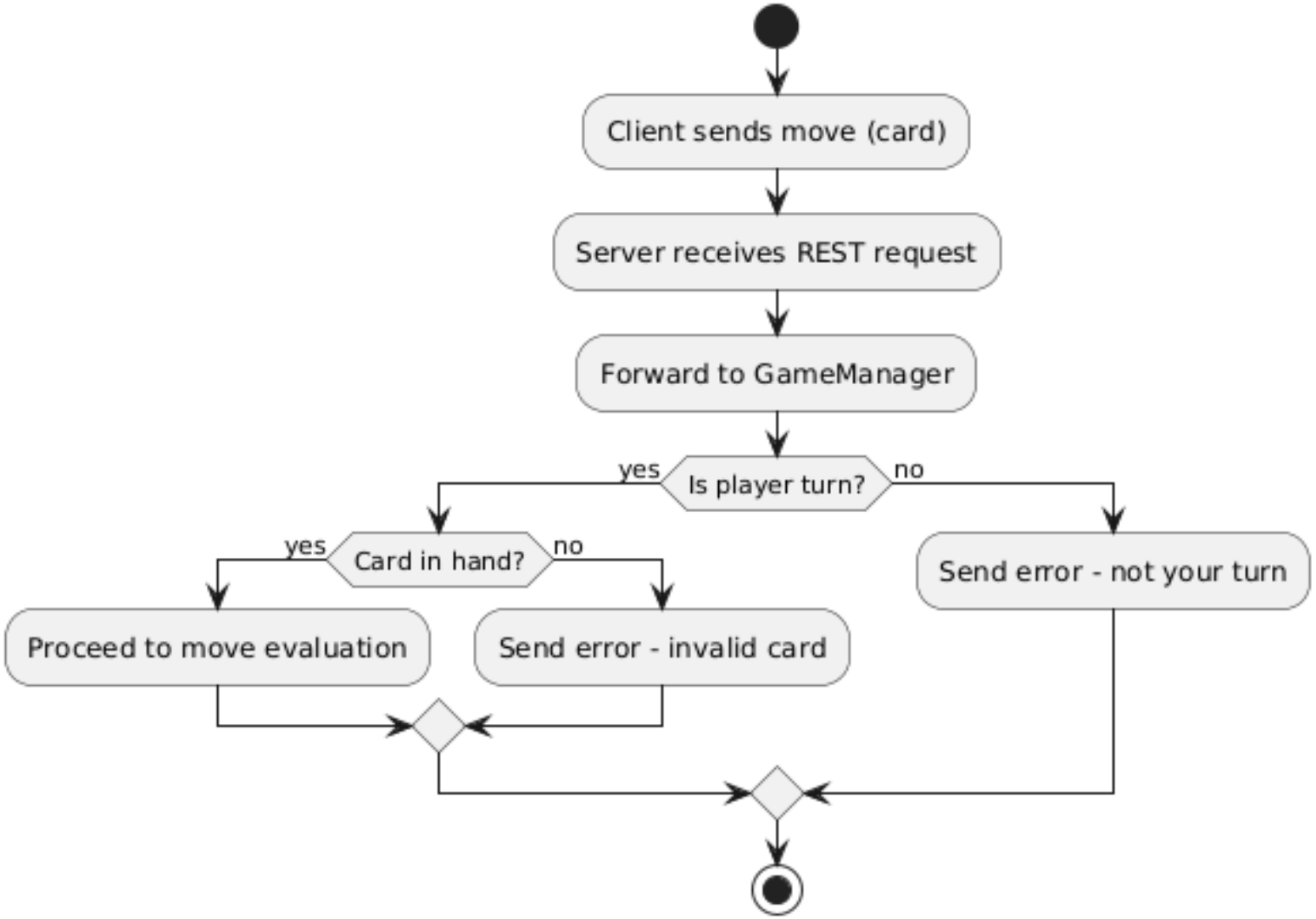
Disconnection & Reconnection Lifecycle



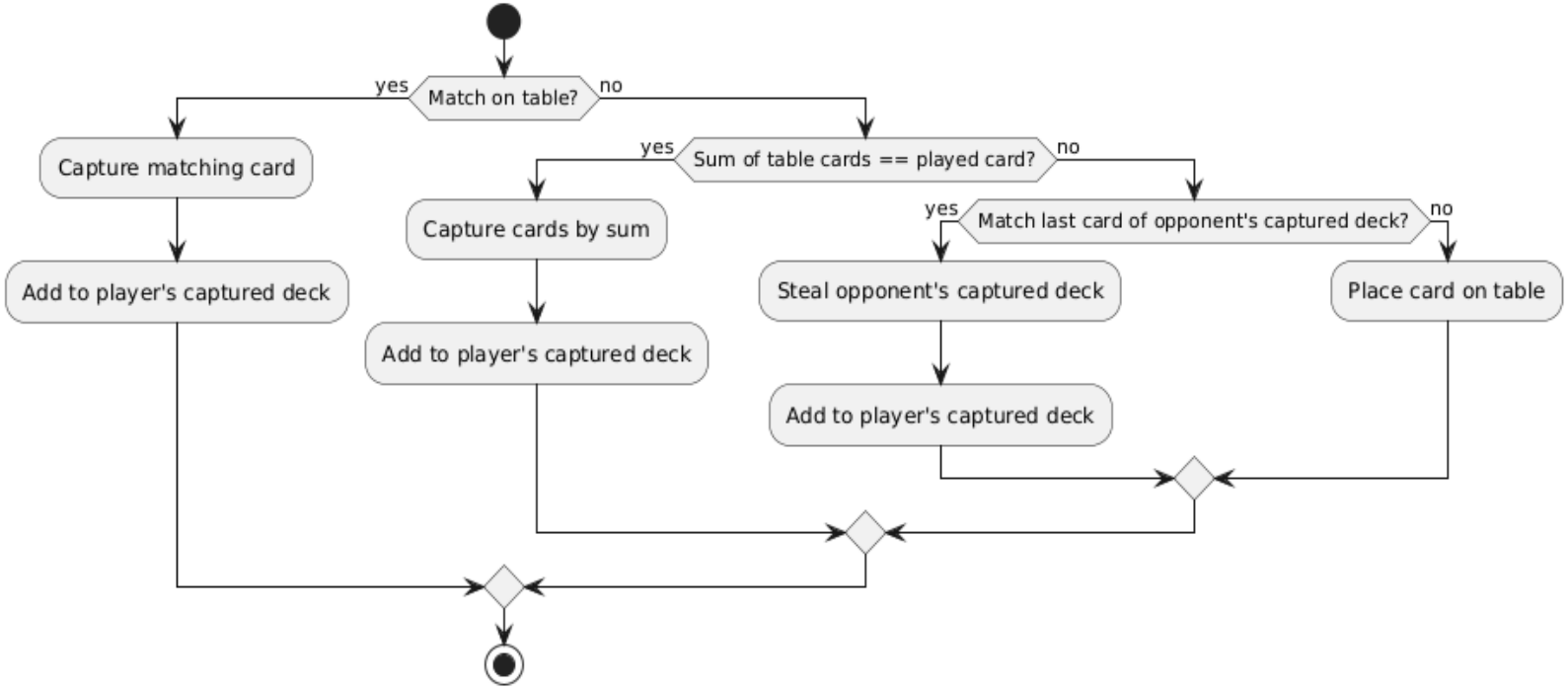
Fault Tolerance Strategy

- Heartbeat mechanism to track client activity
- Disconnected players stored with state snapshot
- Timeout triggers redistribution or game termination
- reconnection restores original state

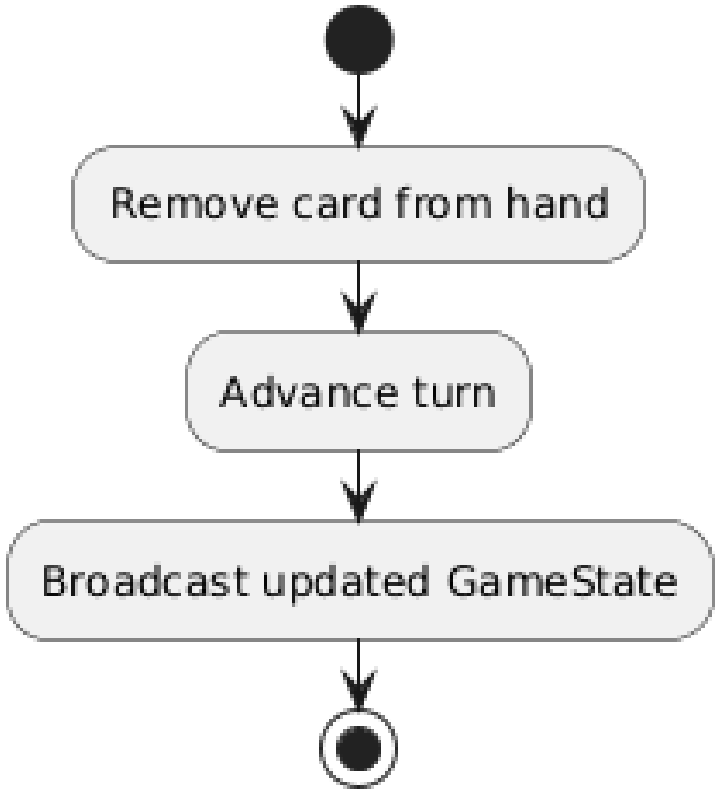
Move Logic Lifecycle: Input & Validation



Move Logic Lifecycle: Capture or Placement



Move Logic Lifecycle: State Update & Turn Progression



Implementation

Diagramma Architetturale: Client & Server Setup

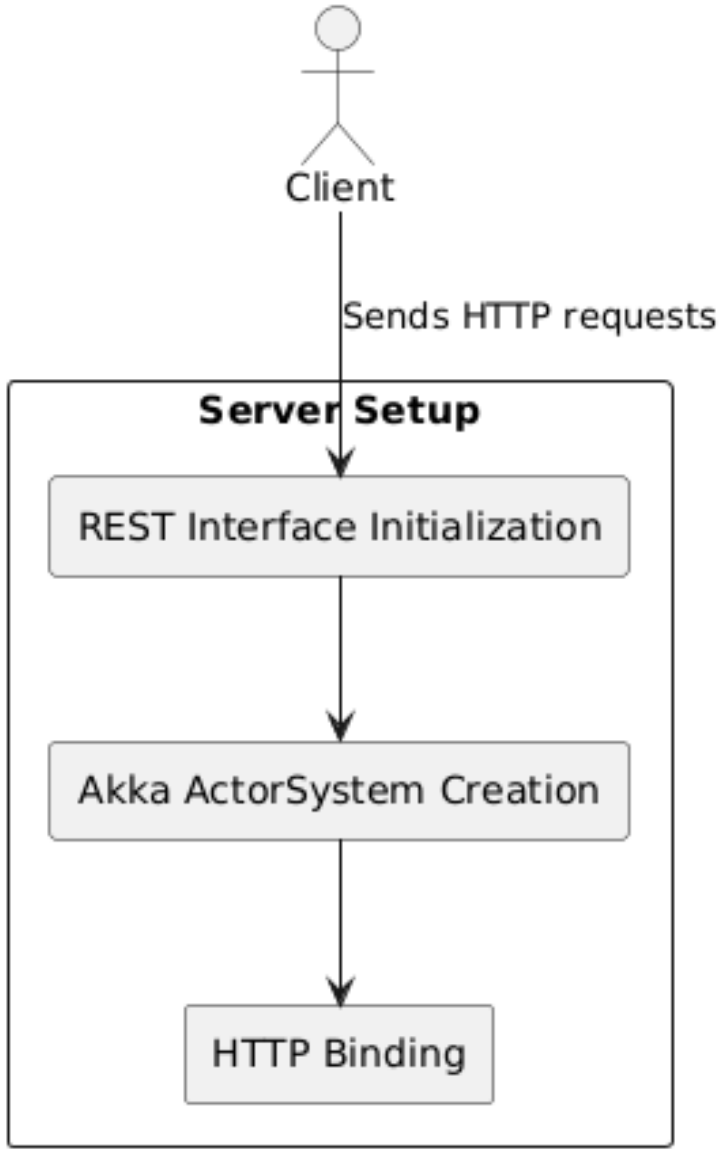


Diagramma Architetturale: Client ↔ REST Routes

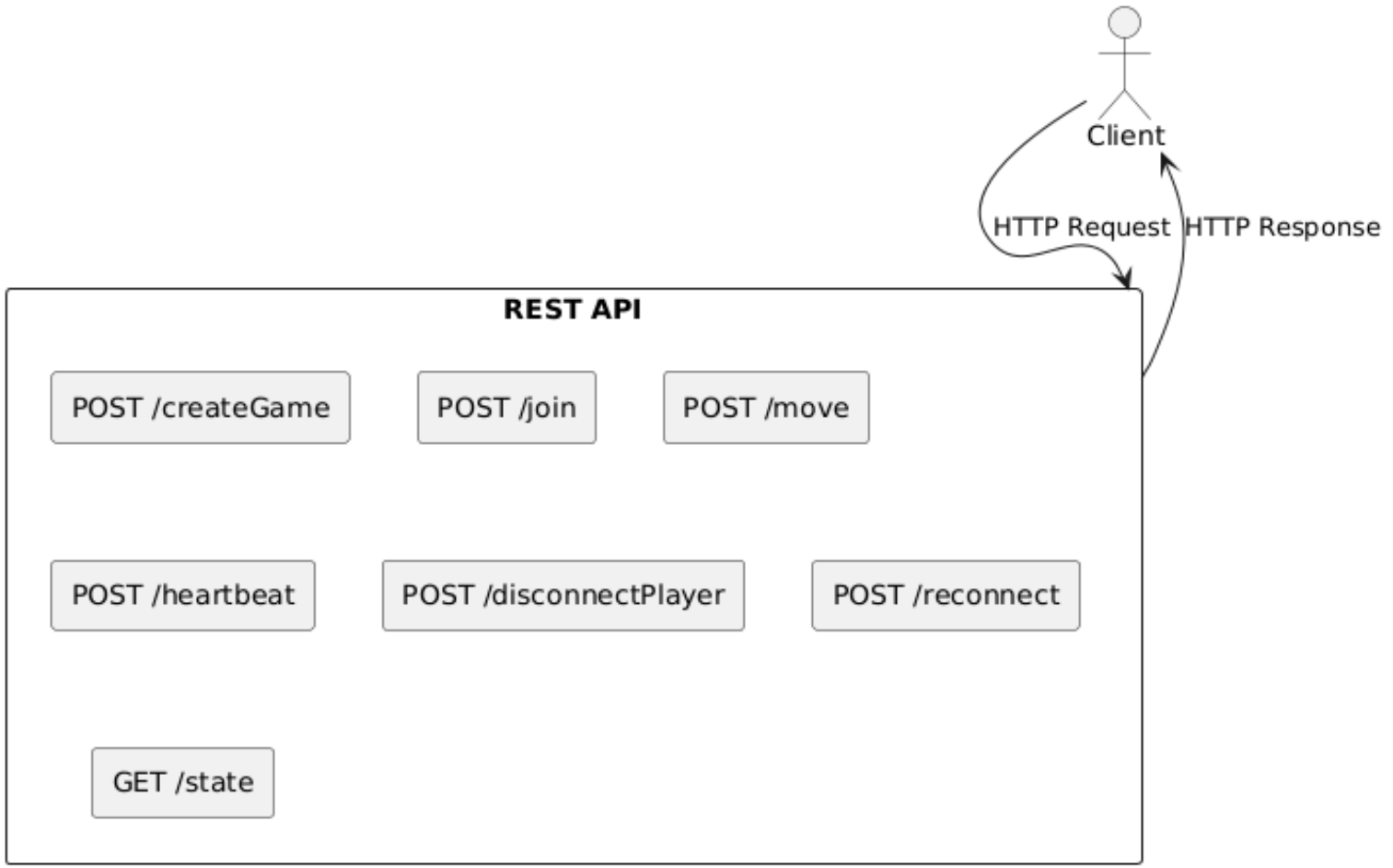
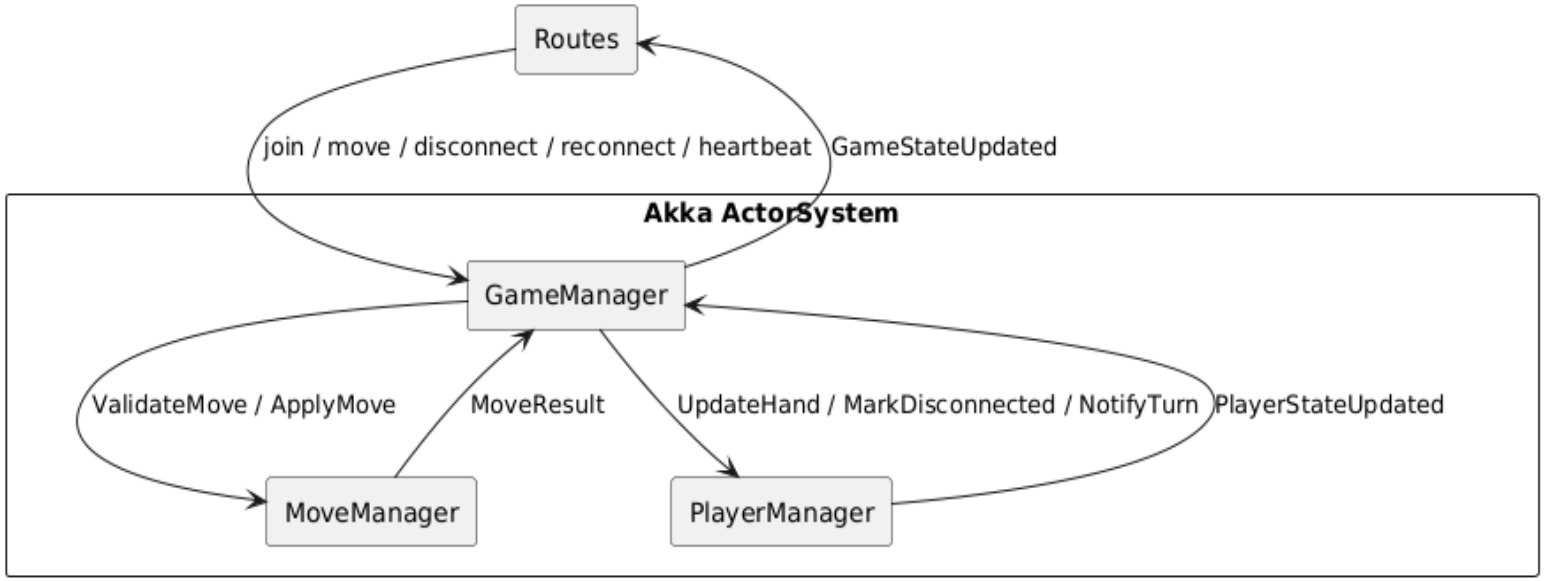


Diagramma Architetturale: REST Routes ↔ GameManager



Testing & Validation

- **Unit Test**: MoveManager, GameManager, PlayerManager
- **Test di integrazione**: full game simulation
- **Test di resilienza**: disconnections, timeouts, reconnections
- **Strumenti**: ScalaTest, curl, script Bash

Conclusions

Final Reflections

Project Strengths:

- **Robust architecture** with clear modular design
- **Immutable game-state model** ensures thread-safe, consistent updates
- **Separation of concerns** across move logic, state management, and disconnection handling
- **Responsive and scalable** under normal load thanks to Scala + Akka HTTP

Future Enhancements

- Develop a **graphical or web-based UI** → Improve accessibility and user engagement
- Add **persistent storage layer** (e.g., database) → Safeguard game state against crashes or restarts
- Address **centralized server bottlenecks** → Adopt Akka Cluster for distributed scalability

With these enhancements, Rubamazzo is ready to evolve from a prototype into a scalable, production-grade platform.

Thanks!