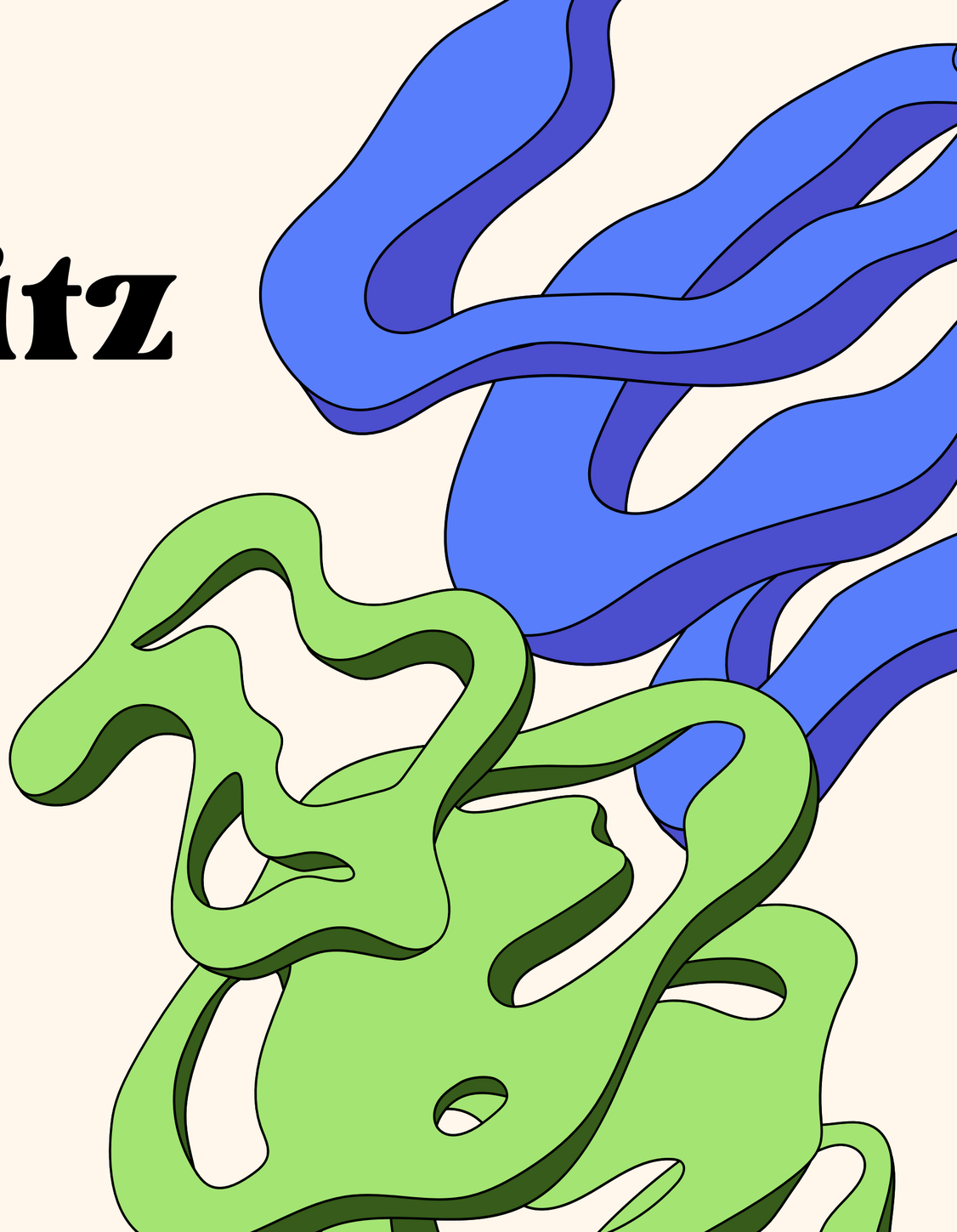
High Noon Blitz

A peer-to-peer Android game using the Bluetooth communication protocol

A.Y. 2023/2024

Alessandro Becci
alessandro.becci@studio.unibo.it

Luca Tonelli
luca.tonelli11@studio.unibo.it

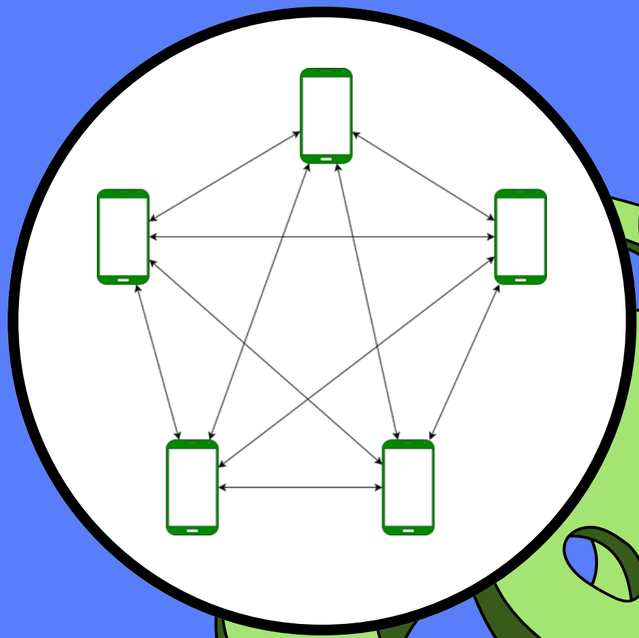


Objectives

- Create a multiplayer reaction game over Bluetooth.
- Measure reaction times despite Bluetooth latency.
- Fully peer-to-peer design with dynamic coordination.
- Recover from disconnections and coordinator failures without breaking the game.

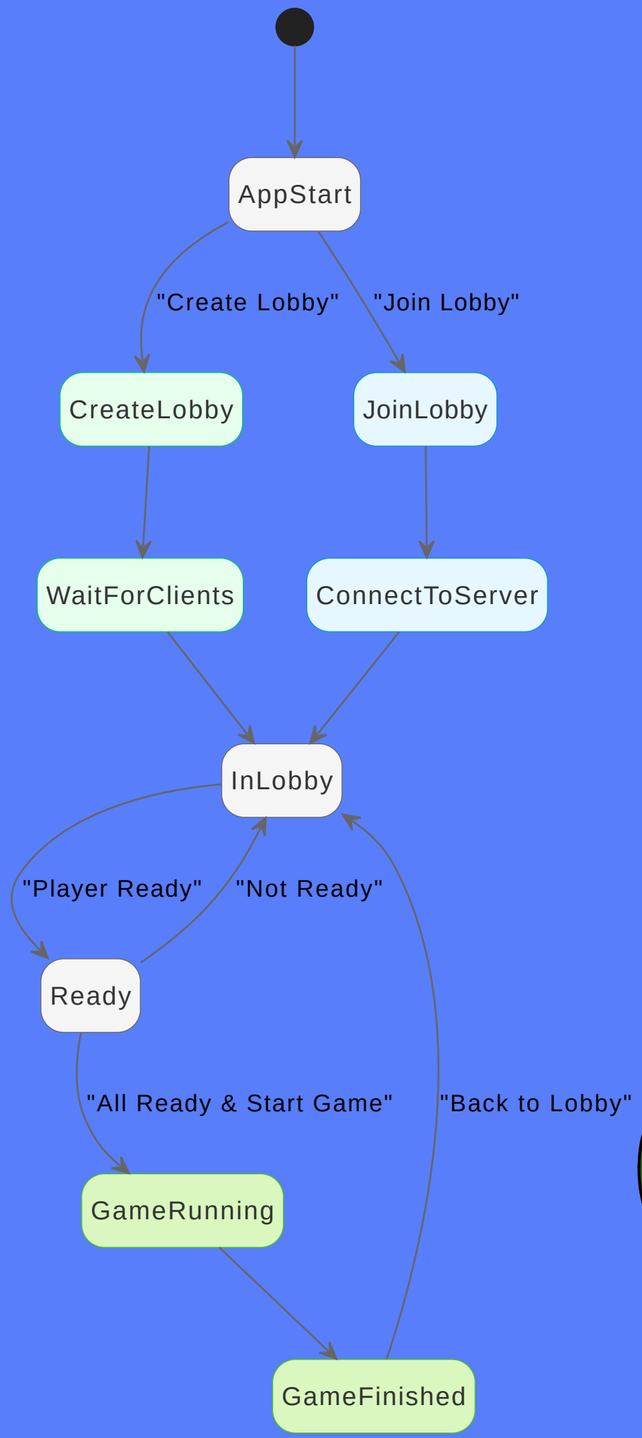
Network Architecture

- Fully connected Bluetooth mesh (server is just a starting point).
- No permanent central server – roles can switch dynamically.
- Nodes can connect to both the server and each other.
- Nodes share network info and elect a new leader when needed.



Game Flow

Normal Game Flow Demo



States

Device Status	Lobby Status	Game Status
• NOT_INITIALIZED • IN_LOBBY • READY • IN_GAME • IN_GAME_WAITING	• CREATED • STARTED	• STARTED • FINISHED_NORMAL • FINISHED_TIMEOUT

Messaging System

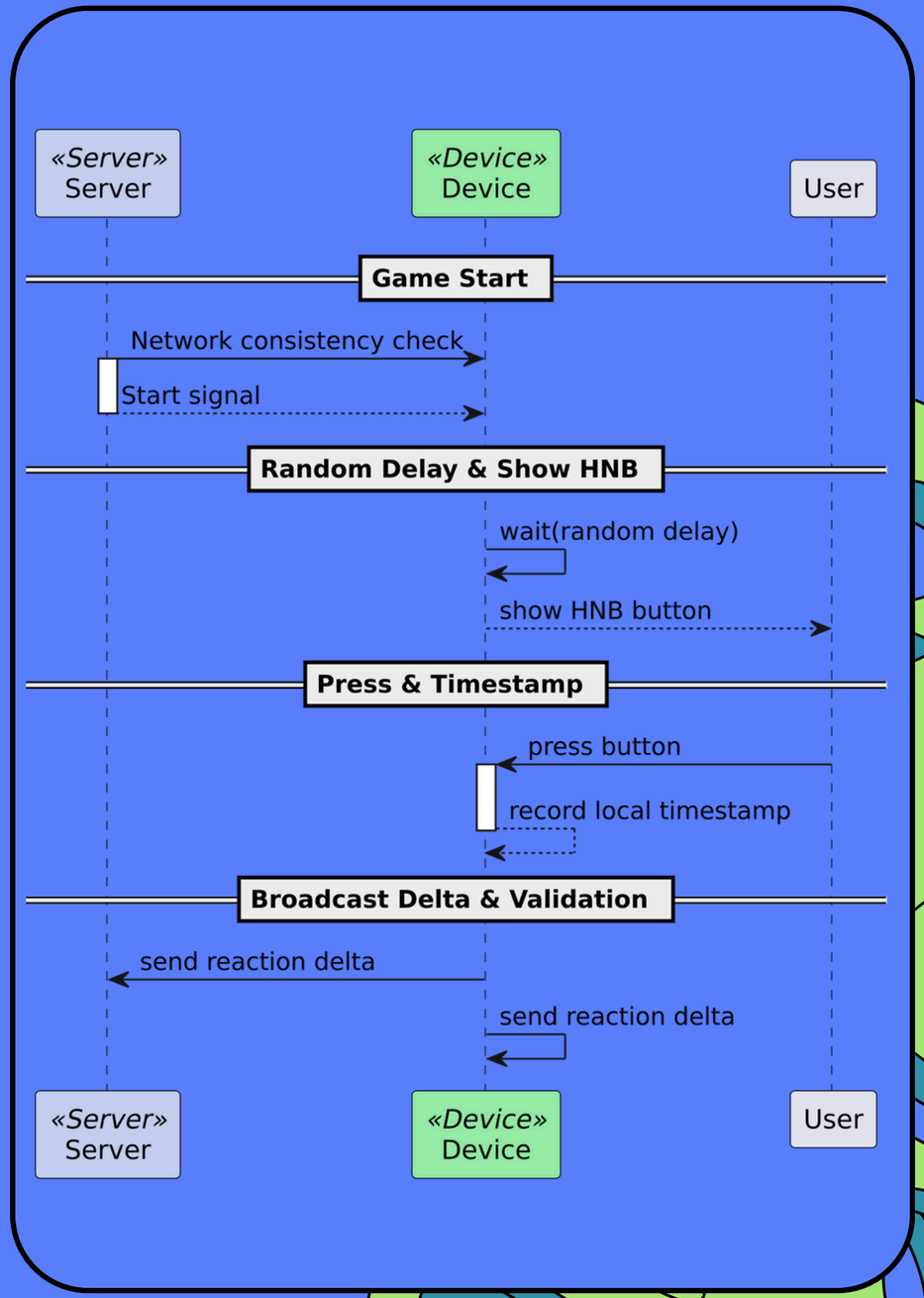
Networking Messages	Lobby Control Messages	Election Protocol Messages	Game Control Messages
• NETWORK_ACK • CONFIGURATION • INFO_SHARING	• READY • CONSISTENCY_CHECK_REQUEST • CONSISTENCY_CHECK_REPLY	• ELECTION_REQUEST • ELECTION_ACK • ELECTION_INGAME_FINISHED	• GAME_START • END_GAME • BACK_TO_LOBBY_GAMELIST

Game Timing

Problem: Bluetooth latency (~200ms) impacts fairness in reaction-based games

Solution:

1. Server performs network check before sending simultaneous start signal
2. HNB button appears after randomized delay on each device
3. Local timestamps record precise button press timing
4. Reaction time deltas broadcast to all devices for validation



Client-Server Connection

- **First Client Connection (Client2)**
  - Client2 connects directly to server
  - Server sends CONFIGURATION with empty MAC list
  - Client2 sets server as MasterDevice
- **Second Client Connection (Client1)**
  - Client1 connects directly to server
  - Server sends CONFIGURATION with MAC list including Client2
  - Client1 sets server as MasterDevice
  - Client1 connects to Client2 directly using MAC address
  - Client2 shares server info via INFO_SHARING

Client-Client Connection

- **First Client Connection (Client2)**
  - Client2 connects directly to server
  - Server sends CONFIGURATION with empty MAC list
  - Client2 sets server as MasterDevice
- **Client-to-Client Discovery**
  - Client1 discovers and connects to Client2 first
  - Client2 sends INFO_SHARING with server address
  - Client1 learns about server
- **Server Connection**
  - Client1 connects to server using INFO_SHARING data
  - Server sends CONFIGURATION with MAC list
  - Client1 sets server as MasterDevice

DEMO

Bully Election Algorithm

Key Concepts

- Each device maintains an election list with MAC addresses and UUIDs
- Fully connected topology enables direct monitoring
- Connection exceptions replace traditional heartbeats

DEMO-LOBBY ***DEMO-GAME***

Election Process

1. Detection: Device detects coordinator failure via Bluetooth exception
2. Initiation: Device sends ELECTION_REQUEST to higher UUIDs
3. Processing: Higher UUID devices respond with ELECTION_ACK
4. Selection: Highest UUID device becomes coordinator
5. Recovery: If coordinator fails, next highest UUID device takes over

Conclusions

This project was a challenging yet rewarding journey that allowed us to bridge the gap between theory and practice.
We developed a Bluetooth-based multiplayer game, focusing on aspects such as network management, device communication, and overall user experience.

Throughout the development, we deepened our understanding of Kotlin and the intricacies of Bluetooth communication on Android. We also explored fault-tolerant networking strategies and navigated the often complex world of Android permissions.