**🌐 ✅ Your Setup Overview**

* **Backend (Spring Boot + WebSocket)** runs on your **laptop**
* **Piston (code executor)** also runs locally
* **Frontend** (HTML/JS) is opened by players from their own computers via your laptop’s IP

**🛠️ How to Set It Up**

**1. 🔍 Find Your Local IP**

Run on your laptop:

bash

CopyEdit

ipconfig (Windows)

ifconfig (Linux/Mac)

Look for your local IP (e.g., 192.168.1.42).

**2. 🧱 Run Backend on Local IP (Spring Boot)**

Edit application.properties:

properties

CopyEdit

server.address=0.0.0.0

server.port=8080

This allows your app to accept requests from any IP in the network.

**3. 🧠 WebSocket URL in Frontend**

Set the frontend WebSocket to:

javascript

CopyEdit

const socket = new WebSocket("ws://192.168.1.42:8080/ws/game");

Replace 192.168.1.42 with your actual local IP.

**4. 🧪 Access the Game from Other Devices**

* Run Spring Boot: mvn spring-boot:run
* On player computers, open a browser and go to:

cpp

CopyEdit

http://192.168.1.42:8080/

(Assuming your frontend is served from Spring Boot or in a static/ folder)

**5. ⚠️ Optional Firewall Config**

If other devices can't connect:

* On **Windows**, allow Java/Spring Boot in firewall
* On **Linux/Mac**, make sure port 8080 is open

**🖼 Frontend Distribution Options**

* Put frontend HTML/JS files in src/main/resources/static/ — Spring Boot will serve them automatically.
* Or you can copy the HTML file to each player’s machine and **manually set the IP** in the script.

**✅ Recap**

| **Component** | **Where it Runs** | **How Players Access** |
| --- | --- | --- |
| Backend (Java + Spring) | Your Laptop | http://<your-local-ip>:8080/ |
| Piston (local) | Your Laptop | Backend uses it via localhost:2000 |
| Frontend (HTML/JS) | Player’s Browser | Served from Spring Boot or copied manually |
| WebSocket | From frontend to your laptop | ws://<your-ip>:8080/ws/game |

Would you like:

* Help generating a test frontend that connects to WebSocket and sends a message?
* Or help setting up the Spring Boot WebSocket controller first?

4o

**You said:**

so now redo the full plan pls every thing with all the detiles from a to z

**ChatGPT said:**

Absolutely! Here's a complete, updated **A to Z roadmap** for your multiplayer coding strategy game **"Code and Conquer"**, hosted locally using **Java (Spring Boot, Maven, WebSocket)** and **Piston** as your code execution engine.

**🧠 "Code and Conquer" Full Plan**

**🌍 Game Summary**

A 2-player LAN game where players conquer a territory-based map by solving coding puzzles. In Phase 1, they race to solve puzzles. In Phase 2 (optional), the losing player can trigger a revenge phase to steal back land.

**🔧 Phase 0: System Requirements**

| **Component** | **Stack** |
| --- | --- |
| Backend | Java 17, Spring Boot, Maven |
| Frontend | HTML + JavaScript |
| Code Execution | [Piston](https://github.com/engineer-man/piston) (running locally via Docker) |
| Hosting | Local Network (LAN) on your laptop |
| Browser Support | Modern browser (Chrome, Edge, etc.) |

**📁 Phase 1: Project Structure**

text

CopyEdit

code-and-conquer/

├── backend/

│ ├── src/main/java/com/codeandconquer/

│ │ ├── config/ # WebSocket config

│ │ ├── controller/ # REST + WebSocket endpoints

│ │ ├── model/ # Game entities

│ │ ├── service/ # Game logic and Piston code execution

│ │ └── util/ # JSON handling, socket tools

│ └── resources/static/ # Frontend files (HTML + JS)

├── piston/ # Cloned piston repo (runs separately)

├── docker-compose.yml # For Piston only

**🛠️ Phase 2: Local Network Hosting Setup**

1. 🔍 Get your local IP  
   Run ipconfig (Windows) or ifconfig (Linux/Mac) → get something like 192.168.1.42
2. ✏️ Update application.properties:

properties

CopyEdit

server.address=0.0.0.0

server.port=8080

1. 👨‍💻 Frontend JS WebSocket must use:

javascript

CopyEdit

new WebSocket("ws://192.168.1.42:8080/ws/game");

**🔌 Phase 3: Piston Setup (Code Execution)**

1. Clone & run Piston:

bash

CopyEdit

git clone https://github.com/engineer-man/piston

cd piston

docker compose up

1. Test the /api/v2/execute endpoint locally:  
   Piston runs on: http://localhost:2000
2. In Spring Boot, create a PistonService:

java

CopyEdit

public String runCode(String code) {

// Build JSON, send to localhost:2000/api/v2/execute, return output

}

**🎮 Phase 4: Game Design**

**Entities**

* **Player**: ID, name, owned territories
* **Territory**: ID, owner, puzzle
* **Puzzle**: ID, description, test cases
* **GameSession**: stores game state, players, current phase

**Game Phases**

1. **Conquer Phase**
   * Players choose any territory
   * Solve coding puzzle → territory is claimed
   * If both choose same, first to finish wins
2. **Revenge Phase**
   * Only losing player may trigger it
   * Each round, players alternate trying to steal a territory
   * Each round is one puzzle → first to solve steals it
3. **End Game**
   * After 6 revenge rounds, or full conquest → winner is the one with most land
   * Losing player may also surrender

**🖼️ Phase 5: Frontend (HTML + JS)**

Place in backend/src/main/resources/static/ so Spring serves it automatically.

**Pages**

* index.html — Join game screen (enter name)
* map.html — Interactive grid map with clickable tiles
* puzzle.html — Code editor popup (Monaco or CodeMirror)
* result.html — Shows winner, scores, option to "revenge" or "end game"

**Features**

* Map highlights territory ownership
* Players click a tile → puzzle opens in popup
* Live updates via WebSocket

**📡 Phase 6: WebSocket and REST Setup**

**WebSocket (/ws/game)**

Handles:

* Join game
* Select territory
* Broadcast code results
* Show map updates

**REST APIs**

http

CopyEdit

POST /api/code/execute → Submit code for checking via Piston

POST /api/game/start → Create new session

POST /api/game/submit → Player submits puzzle solution

POST /api/game/revenge → Start revenge phase

**🕹️ Phase 7: Game Flow**

text

CopyEdit

Player A and B join the game

→ Game starts (Conquer Phase)

→ Players click tiles, solve puzzles

→ Territory claimed on correct submission

→ When done, losing player decides:

[Revenge] or [End Game]

→ If Revenge:

→ 6 turn-based rounds of puzzle duels

→ Game ends → Show final map and winner

**🧪 Phase 8: Testing & LAN Access**

* Launch Spring Boot: mvn spring-boot:run
* Verify access from other devices:

cpp

CopyEdit

http://192.168.1.42:8080/

* Test WebSocket connection
* Test puzzle solving and code execution

**🌟 Optional Features to Add Later**

* Login + player avatars
* Admin dashboard to add/edit puzzles
* Offline single-player mode
* Ranking system and history tracking
* Code editor themes
* Sound effects / animations

**✅ Final Tips**

| **Task** | **Tool** |
| --- | --- |
| Real-time game state | WebSocket |
| Puzzle validation | Piston |
| Interactive UI | HTML + JS + WebSocket |
| Backend logic | Java Spring Boot |
| Multiplayer over LAN | Local IP hosting |

Would you like to:

1. Scaffold the Spring Boot backend structure now?
2. Build a test WebSocket + code execution flow?
3. Set up the frontend map and connection first?

Let me know how you'd like to begin!