Real-Time Turn-Based Chess-Like Game Using WebSocket Communication

### 1. Server-side Implementation (Node.js + WebSocket)

\*\*Install dependencies:\*\*

```bash

npm init -y

npm install express ws

```

\*\*Directory structure:\*\*

```

/project

/client

index.html

/server

server.js

```

\*\*`server/server.js`:\*\*

```javascript

const express = require('express');

const http = require('http');

const WebSocket = require('ws');

// Set up Express server

const app = express();

const server = http.createServer(app);

const wss = new WebSocket.Server({ server });

// Game variables

let players = [];

let gameState = {};

// Helper function to broadcast the game state

const broadcastGameState = () => {

const state = JSON.stringify({

type: 'game-state-update',

gameState,

});

players.forEach(player => player.ws.send(state));

};

// WebSocket Connection Handling

wss.on('connection', (ws) => {

if (players.length >= 2) {

ws.send(JSON.stringify({ type: 'error', message: 'Game full.' }));

ws.close();

return;

}

const playerId = players.length === 0 ? 'A' : 'B';

const player = { ws, id: playerId };

players.push(player);

if (players.length === 2) {

gameState = {

A: ['A-P1', 'A-P2', 'A-H1', 'A-H2', 'A-H2'],

B: ['B-H2', 'B-H1', 'B-P1', 'B-P2', 'B-P3'],

board: Array(5).fill(null).map(() => Array(5).fill(null)),

turn: 'A',

};

// Initialize positions on the board

gameState.board[0] = gameState.A;

gameState.board[4] = gameState.B;

broadcastGameState();

}

ws.on('message', (message) => {

const data = JSON.parse(message);

if (data.type === 'move' && gameState.turn === playerId) {

const { character, move } = data;

// Handle the move logic here (e.g., movement validation, combat logic)

// Simulate a successful move (move validation not implemented yet)

gameState.turn = gameState.turn === 'A' ? 'B' : 'A'; // Switch turns

broadcastGameState();

}

});

ws.on('close', () => {

players = players.filter(p => p.ws !== ws);

gameState = {};

if (players.length < 2) {

broadcastGameState(); // Clear game state if a player disconnects

}

});

});

server.listen(8080, () => {

console.log('Server is running on port 8080');

});

```

### 2. Web Client Implementation (HTML + JavaScript)

\*\*`client/index.html`:\*\*

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Chess-like Game</title>

<style>

table {

width: 200px;

height: 200px;

border-collapse: collapse;

}

td {

width: 40px;

height: 40px;

border: 1px solid black;

text-align: center;

vertical-align: middle;

}

</style>

</head>

<body>

<h1>Chess-like Game</h1>

<div id="status"></div>

<table id="game-board"></table>

<script>

let ws = new WebSocket('ws://localhost:8080');

let gameState = null;

let playerId = null;

ws.onmessage = function(event) {

const data = JSON.parse(event.data);

if (data.type === 'game-state-update') {

gameState = data.gameState;

renderGameBoard();

document.getElementById('status').innerText = `Current Turn: ${gameState.turn}`;

}

};

function renderGameBoard() {

const boardElement = document.getElementById('game-board');

boardElement.innerHTML = '';

gameState.board.forEach((row, i) => {

const tr = document.createElement('tr');

row.forEach((cell, j) => {

const td = document.createElement('td');

td.innerText = cell || '';

td.onclick = () => handleCellClick(i, j);

tr.appendChild(td);

});

boardElement.appendChild(tr);

});

}

function handleCellClick(row, col) {

if (gameState && gameState.turn === playerId) {

const character = gameState.board[row][col];

if (character && character.startsWith(playerId)) {

// Send a move command (implement move logic)

ws.send(JSON.stringify({ type: 'move', character, move: 'F' }));

}

}

}

ws.onclose = function() {

document.getElementById('status').innerText = 'Connection closed.';

};

</script>

</body>

</html>

```

### 3. Running the Project

1. Start the server:

```bash

node server/server.js

```

2. Open the client in a browser by navigating to the `client/index.html` file. You can open two browser tabs to simulate both players.

### Explanation

- \*\*Server-side:\*\* The server handles the game logic and communication between clients via WebSocket. It keeps track of the game state (character positions, turns) and broadcasts it to all clients after every valid move.

- \*\*Client-side:\*\* The web client renders the game board and communicates with the server via WebSocket. When a move is made, the client sends the move to the server, which processes it and sends back the updated game state.

### Next Steps

- \*\*Implement Move Logic:\*\* Extend the server-side code to handle move validation for different character types (Pawns, Heroes).

- \*\*Improve UI:\*\* Add better visual differentiation for characters, highlights for valid moves, and more interactive controls.

- \*\*Add Features:\*\* Implement game over conditions, dynamic team composition, spectator mode, or AI as per the bonus challenges.

This basic model can be scaled and improved based on the provided objectives.