Assignment 2: OctaFlip with Network

Problem Description

Implement a **server** and a **client** for an 8×8 board game called OctaFlip. Two clients register as **Red (R)** and **Blue (B)**, then take turns moving their pieces according to the rules below. Your server acts as the referee.

(OctaFlip rules are the same as those in Assignment 1: ...)

Use cJSON, Jansson, etc. to handle JSON payloads (e.g., {"type": "...", ...})

Client to Server

- **register**
    - The server checks whether the given username already exists and, if not, adds it to the game's waiting list.
        - Payload: {
        - "type": "register",
        - "username": {your_username},
        - }
        - Response: (server to client)
            - If register success, {"type": "register_ack", }
            - If game already in progress, {"type": "register_nack", "reason": "game is already running"}
- **Move: (Automate your algorithm by implementing a `move_generate` function that takes the current board state as input and returns target coordinates `(tx, ty)` for the payload)**
    - Description: The client informs the server of the source and target coordinates for a move.
        - Payload: {
        - "type": "move",
        - "username": {your_username},
        - "sx": {integer},"sy": {integer},"tx": {integer},"ty":{integer},
        - }
        - Response: (server to client)
            - When the move is valid, {"type":"move_ok","board":[...],"next_player":"Bob"}

- When the move is invalid, {"type":"invalid_move","board":[...],"next_player":"Bob"}
- When a time out occurs, {"type":"pass", "next_player":"Bob"}

Server to Client

- **game_start**
    - Description: Once two players have registered, the server broadcasts that the game is starting, indicates who goes first, and provides the initial board state.

      ([
      [“R”, “.”, “.”, “.”, “.”, “.”, “.”, “B”,],
      [“.”, “.”, “.”, “.”, “.”, “.”, “.”, “.”,],
      [“.”, “.”, “.”, “.”, “.”, “.”, “.”, “.”,],
      [“.”, “.”, “.”, “.”, “.”, “.”, “.”, “.”,],
      [“.”, “.”, “.”, “.”, “.”, “.”, “.”, “.”,],
      [“.”, “.”, “.”, “.”, “.”, “.”, “.”, “.”,],
      [“.”, “.”, “.”, “.”, “.”, “.”, “.”, “.”,],
      [“B”, “.”, “.”, “.”, “.”, “.”, “.”, “R”,],
      ])
        - Payload: {
        - “type”: “game_start”,
        - “players”: [{username1}, {username2}],
        - “first_player”: {username1},
        - }
- **your_turn**
    - Description: The server sends the current board state to the player along with the remaining time for this turn. The board is represented as an array of 8 strings, each containing exactly 8 characters (e.g., `char board[8][8]`).
        - Payload: {
        - “type”: “your_turn”,
        - “board”: [...],
        - “timeout”: 5.0,
        - }
- **game_over**
    - Description: When the game’s termination condition is met, the server sends the final results to both players.
        - Payload: {

- “type”: “game_over”,
- “scores”: {
- {username1}: {user1_score}, {username2}:{user2_score}
- },
- }

Constraints

- You must implement your logic in C (not C++).
- **The client must accept command-line options (e.g., via `argv`) for specifying the server IP address and port.**
  ./client -ip {ip} -port {port} -username {username}
- Communication must use TCP sockets and JSON payloads.
- Your code should not produce segmentation faults for any valid/invalid input.
- Maintain modular and readable function definitions.

Grading Scenario: Please generate a client, server-visible log that records each board state and the corresponding move

- **Player Registration**
  - user: “Alice” connects and sends a `register` request.
  - user: “Bob” connects and sends a `register` request.
- **Game Start Notification**
  - As soon as two players are registered, the server broadcasts a **game_start** message (including the initial board and the first player).
- **Turn Handling: Automate move execution with a function**
  - The server sends **your_turn** to the current player, including the latest board state and a `timeout`
  - The player must send a **move** before the timeout expires.
- **Move Validation and Feedback**
  - Upon receiving the **move**, the server validates it.
  - If valid: respond with **move_ok**, updated board, and `next_player`.
  - If invalid: respond with **invalid_move**, current board, and `next_player`.
  - If the player fails to move before the timeout, the server sends a **pass** and advances to the next player.

    - Repeat the cycle (Turn Handling → Move Validation and Feedback) until the game ends.

- **Game Over**
    - The server sends **game_over** to players

Submission Guidelines

- Submit a zip file that consists of `server.c` and `client.c` file containing your implementation.
- Make single purpose function (one can understand the purpose with function names)
- File naming format: `hw3_YOURSTUDENTID.zip`
  Example: `hw3_200012345.zip`
- Upload your code via LMS before June 2 Monday, 11:59:59 PM.
- You are granted a total of **two days** of grace period, which can be used for either Assignment 1 or Assignment 2, or split between both.
    - For example, you may submit Assignment 1 two days late and Assignment 2 on time, or each assignment one day late. No additional extensions will be given beyond this shared grace period.

Implementation Guidelines

- Prevent a user who is already registered from registering again.
- Registration order determines play order: the **first player** to register **goes first**.
- The **game starts** only when **two players** have successfully **registered**; the turn timer (5 s) begins immediately.
- Before accepting a move, **verify** that it's the **current player's turn**.
- If a player **fails to move** within the 5 s timeout, automatically **pass** the turn to the other player.
- **Reject** any move to an **invalid position**.
- Support exactly two players; store each player's details (username, socket, etc.).
- If **one client disconnects**, automatically **pass** their turn to the remaining player.
- In a fatal case (e.g., both clients disconnect), terminate the game.