Specification for Connecting a Tetris Game to TetrisServer

In this specification, you will learn how to establish a connection between your Tetris game and an external server (`TetrisServer`). The server plays as an external player, making optimized moves based on the game state your game sends. The communication between the Tetris game and `TetrisServer` happens over `localhost:3000` using JSON objects. Here are the details on how to connect and communicate with the server.

Communication Protocol

1. **Client Request**:

   - The Tetris game (client) sends a **JSON string** representing the current game state via a socket connection to `localhost:3000`.
   - The request contains a serialized `PureGame` object, including:
     - **`width`**: The width of the Tetris board.
     - **`height`**: The height of the Tetris board.
     - **`cells`**: The current state of the board (2D array).
     - **`currentShape`**: The tetromino that is currently falling.
     - **`nextShape`**: The next tetromino to be played after the current one.

2. **Server Response**:

   - The server will respond with a **JSON string** representing an `OpMove` object.
   - The `OpMove` object contains:
     - **`opX`**: The optimal X-position where the current tetromino should be placed.
     - **`opRotate`**: The optimal number of rotations to apply to the current tetromino.

Additional Interpretation of `OpMove` Values

- **If `opX` is `0`**: This means that the tetromino should be placed in the **left-most position** on the board.
- **If `opRotate` is `0`**: This means that the tetromino **does not need to rotate** and should be placed as is.

Steps to Connect to the Tetris Server

1. **Establish a Socket Connection** to `localhost:3000`.
2. **Send the game state** (a serialized `PureGame` object) to the server in JSON format.
3. **Receive the optimized move** (a serialized `OpMove` object) from the server in JSON format.
4. **Interpret and apply the move**:
   - If `opX` is `0`, place the tetromino in the left-most position.
   - If `opRotate` is `0`, do not rotate the tetromino.

Example Code Snippet for Socket Communication

This code demonstrates how to send the `PureGame` object and receive the `OpMove` object using a socket connection and JSON serialization.

Client Code: Sending `PureGame` and Receiving `OpMove`

```java
import com.google.gson.Gson;
import java.io.*;
import java.net.Socket;

public class TetrisClient {

    private static final String SERVER_HOST = "localhost";
    private static final int SERVER_PORT = 3000;

    public static void main(String[] args) {
        PureGame game = new PureGame(); // Assuming you have filled game state

        // Step 1: Establish a socket connection to the server
        try (Socket socket = new Socket(SERVER_HOST, SERVER_PORT);
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

            // Step 2: Convert PureGame object to JSON
            Gson gson = new Gson();
            String jsonGameState = gson.toJson(game);

            // Step 3: Send the game state to the server
            out.println(jsonGameState);
            System.out.println("Sent game state to server: " + jsonGameState);

            // Step 4: Wait for the server's response (OpMove)
            String response = in.readLine();
            System.out.println("Received response from server: " + response);

            // Step 5: Convert the JSON response to an OpMove object
            OpMove move = gson.fromJson(response, OpMove.class);
            System.out.println("Optimal Move: X=" + move.opX() + ", Rotations=" + move.opRotate());

            // Step 6: Apply the move based on the opX and opRotate values
            if (move.opX() == 0) {
                System.out.println("Place the piece at the left-most position.");
            } else {
                System.out.println("Move the piece to X=" + move.opX());
            }

            if (move.opRotate() == 0) {
                System.out.println("No rotation needed.");
            } else {
                System.out.println("Rotate the piece " + move.opRotate() +
" times.");
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Class Definitions

Here are the provided class definitions for `PureGame` and `OpMove`.

```
import java.util.Arrays;

public class PureGame {
    private int width;
    private int height;
    private int[][] cells;
    private int[][] currentShape;
    private int[][] nextShape;

    @Override
    public String toString() {
        return "PureGame{" +
                "width=" + width +
                ", height=" + height +
                ", cells=" + Arrays.deepToString(cells) +
                ", currentShape=" + Arrays.deepToString(currentShape) +
                ", nextShape=" + Arrays.deepToString(nextShape) +
                '}';
    }

    // Getters and Setters...
}

public record OpMove(int opX, int opRotate) {
}
```

Communication Example

Here’s an example of what the JSON strings might look like during communication:

Request (`PureGame` in JSON format):

```
{
  "width": 10,
  "height": 20,
  "cells": [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
    ...
  ],
  "currentShape": [
    [1, 1, 0],
    [0, 1, 1]
  ],
  "nextShape": [
    [1, 1, 1, 1]
  ]
}
```

Response (`OpMove` in JSON format):

```
{
  "opX": 3,
  "opRotate": 1
}
```

Interpreting the `OpMove` Response

- **If `opX = 0`**: The tetromino should be placed in the **left-most position** of the board.
- **If `opRotate = 0`**: The tetromino does not need to be rotated.

For example:

- **`opX = 0` and `opRotate = 1`**: Move the piece to the **left-most position** and rotate it **once**.
- **`opX = 3` and `opRotate = 0`**: Move the piece to **X = 3** without rotating.

Requirements and Setup

- **Java Sockets**: For communication over TCP.
- **Gson (or another JSON library)**: To handle JSON serialization and deserialization.
  - Add Gson to your project using Maven or Gradle, or download the jar file.

Error Handling

- **Socket exceptions**: Handle cases where the server is unreachable, the connection is lost, or the server returns unexpected data.
- **Invalid JSON responses**: Add checks to validate that the server's response is in the correct format.

Summary

1. **Connection**: The Tetris game connects to `TetrisServer` via a socket at `localhost:3000`.
2. **Communication**: The game sends the current state (as a `PureGame` JSON object) to the server and receives the optimized move (as an `OpMove` JSON object).
3. **Interpreting `OpMove`**:
    - If `opX` is `0`, place the tetromino in the left-most position.
    - If `opRotate` is `0`, do not rotate the tetromino.
4. **Next Steps**: Implement the move in your game based on the received `OpMove` data.

This approach allows your Tetris game to communicate with the external server and make intelligent moves based on the server's optimized decision-making.