NETWORKING STRATEGY

This Ivanhoe game uses a **Client-Server** networking architecture. A single, centralized Server is connected to multiple clients, and communicates with each of them using **JSON Strings**. The JSON Strings describe player objects, cards, and game state, and are used to indicate turns and moves.

All of the game logic and model objects are stored on the server side. The client side stores no information, and receives a complete update of the game state after every turn & move (including player hands and displays). This is advantageous because it prevents synchronous issues and can deal with a certain degree of faulty networking.

The server encompasses the **Model** and the **Controller** elements of the game. The client encompasses the **View**. It is essentially a GUI that displays elements based on inputs from the controller. Is also has event handlers (such as buttons and selectors) that relay user choices back to the server. The server then relays this information to the controller, which modified its model and decides on what response to send back to the client(s).

--- PROS

- **Centralized Model.** The server is the only place that keeps track of the game state. Even if communication goes temporarily awry between server and clients, the client can be quickly synchronized with an update.

- **Security.** Since all important information is stored on the server, it is very difficult to cheat.

- **Flexibility.** By using JSON strings, clients (and the server) can be written in any language (and not just Java). This means that if the game were ever to be implemented for web or mobile platforms, the communication would not have to change. This offers more flexibility than serializing objects, which is limited to Java-to-Java communication.

--- CONS

- **Network Congestion / Scalability.** For only five players and a single game, a single server can deal with the volume of requests. However, if the amount of games & players were increased, eventually the server would be unable to deal with the volume of requests.

- **Robustness**. If the server fails without warning, it is impossible to continue the game and the clients will not be disconnected gracefully. It is also impossible to play a game without the server.