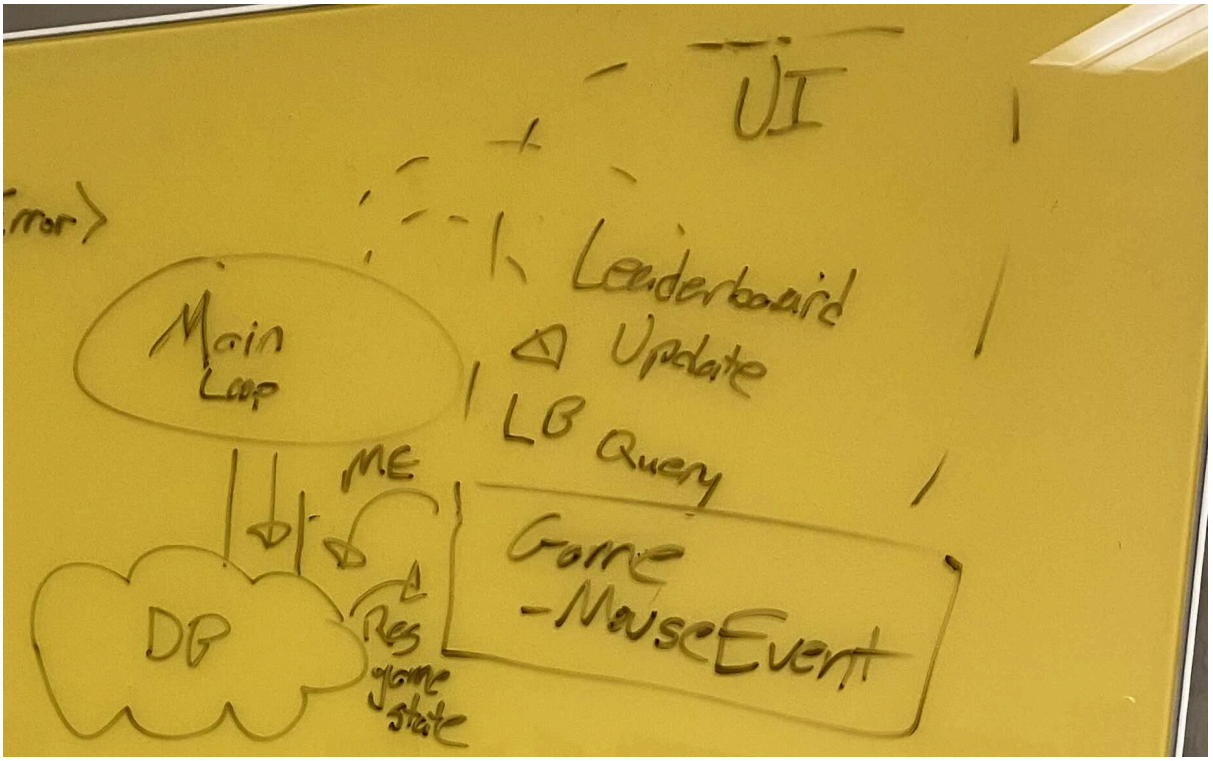
Project Overview (V.1.0):

This document was created after having an extensive conversation with Nick about everything we went over in the meeting on monday and more. Some of us were unable to attend, dropped in late, or simply didn't understand what was really is going on (this is **me**). Again, everything below is my understanding of the project and the direction we're all headed to, derived from one conversation with Nick. Please take this with a grain of salt. For the people who were lost like me, this might help.

Picture of what Nick went over with me:



From as far as I can tell, our big project here is primarily separated into 3 parts:

1. **Database (JSON Storage)** *we're just storing raw data
2. **Server/Main Loop (Logic Layer)** *where all our methods are going to go
3. **Subsystems (Game Logic, Matchmaking, GUI, etc.)** *querying what we want from server

The big central part of our project is the **Main Loop**. We're going to avoid referring to it as "Server" from our last meeting on monday as it sparked a lot of confusion for students in second year CPSC including me. The **Main Loop** acts as the processing unit that manages interactions between **Subsystems** and **Database**, while the **Database** stores all persistent player and game information. Below is my breakdown of each part, and its purpose.

1. Database (JSON Storage)

The **Database** provides a way to store all the persistent data across games and sessions.

- **Purpose**: We're designing it to store structured but flexible data. This is going to be things like player profiles, game history, and leaderboard rankings. These are just from the top of my head.
- **Role**: The database supports all other components by storing raw data. For example:
    - The **Matchmaking System** queries the Database for player skill ratings and match history.
    - The **Game Logic** queries I actually don't know. Peter will decide how he wants it structured.

I think the whole purpose for using the JSON format and this whole database thing is because it keeps gameplay logic separate from data. Our updates to game mechanics and everything else won't really disrupt and mess up the underlying data storage.

Example JSON Structure for Player Data:

```
{
  "playerID": "12345",
  "username": "PlayerOne",
  "rating": 1500,
  "gameHistory": [
    {"opponentID": "67890", "result": "win", "date": "2024-10-10"},
    {"opponentID": "11223", "result": "loss", "date": "2024-10-12"}
  ],
}
```

From this example:

The rating could be used by **Matchmaking System** to pair players. I'm planning to match players within a certain +- 30 range for example.

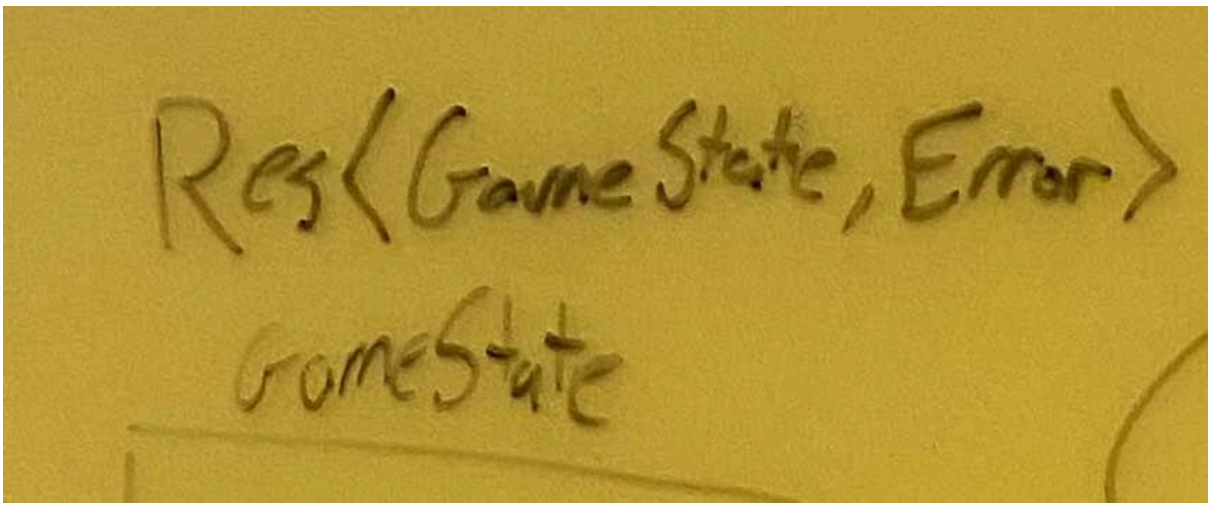
The gameHistory could be queried to update a player's skill rating after each match.

2. Main Loop (Logic Layer)

The **Main Loop** (previously referred to as “Server”) acts as sort of a ‘filter’ for all game logic, continuously coordinating between various components and handling player interactions.

What I’m trying to get at here is the UI is taking everything. It doesn’t care. If the UI shows a prompt to enter how many **rows(int)** do you want for your connect 4 board, and the user enters “Jamaican Bobsled Team”. The UI is completely chill with this. It’s up to the server to use logic and filter out whatever is not cool and return an error.

Pic from the conversation:



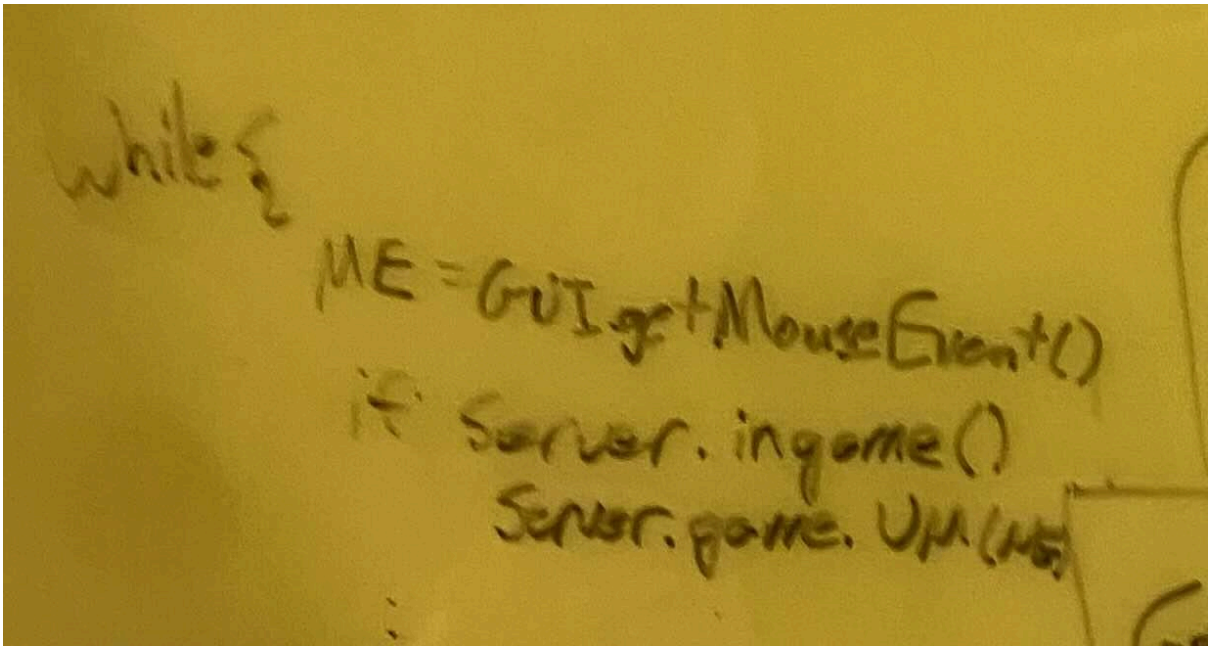
^^This is what the server will return when the user enters some nonsense.

- **Purpose**: Renaming it to “Main Loop” clarifies that this isn’t a web server but a continuously running system to manage game processes. We can just think of this as a big java class.
- **Components and Operations**:
    - **Game Logic**: Processes rules, state transitions, and interactions (e.g., valid moves).
    - **Matchmaking and Leaderboard(My Team ♥)**: Manages skill-based matchmaking and ranks players.
    - **Sub-Loops** for **Game State** and **Server Update**: Checks game state and retrieves data for UI updates. I’ll talk about these sub-loops later with a picture.
- **Role**: Acts as an intermediary between raw data (Database) and subsystems like GUI and Matchmaking. For example:
    - The **Matchmaking System** will request data (e.g., player ratings) from the Main Loop, which it then uses to pair players or update leaderboard stats.
    - The **Game Logic** could request player move histories for verification against game rules.

I believe we're doing this for the sole purpose of minimizing the pain experienced by everyone. Essentially, all of our subteams are the subsystems, which means we'll only have to rely on the public interfaces of the Main Loop, simplifying integration. There's less need to constantly badger every subteam for this and that.

Example Code for the Main Loop:
This is what I cooked up from my understanding of Nick's picture below the typed code below.

```
public class MainLoop {
    private Database database;
    private GameLogic game;
    private MatchmakingSystem matchmaking;
    public void run() {
        while (true) {
            MouseEvent ME = GUI.getMouseEvent();  // Retrieve mouse events
            if (game.inProgress()) {              // Check if a game is active
                GameState updatedState = game.update(ME);  // Update game state based on
event
                GUI.render(updatedState);                  // Render new state to UI
            }
            matchmaking.findMatch();
            database.saveGameState(game.getGameState());
        }
    }
}
```



What I want to get at with this example code:

- The **Main Loop** listens for mouse events (ME), sends them to game.update(), then retrieves the new state for rendering.
- The matchmaking.findMatch() call constantly checks the queue for compatible players to start a match.
- database.saveGameState() ensures game progress is persistently stored.

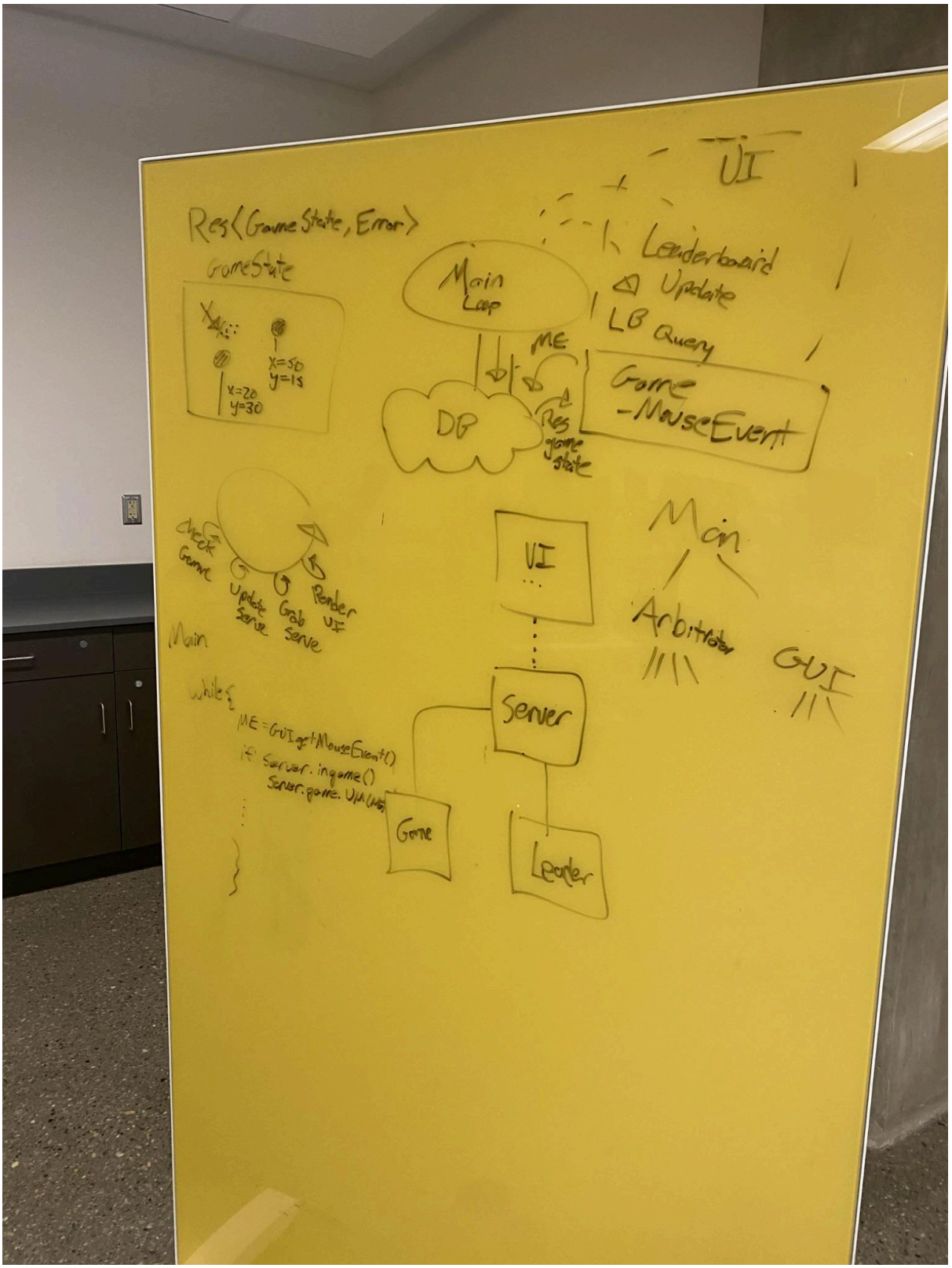
3. Subsystems (Game Logic, Matchmaking, GUI, etc.)

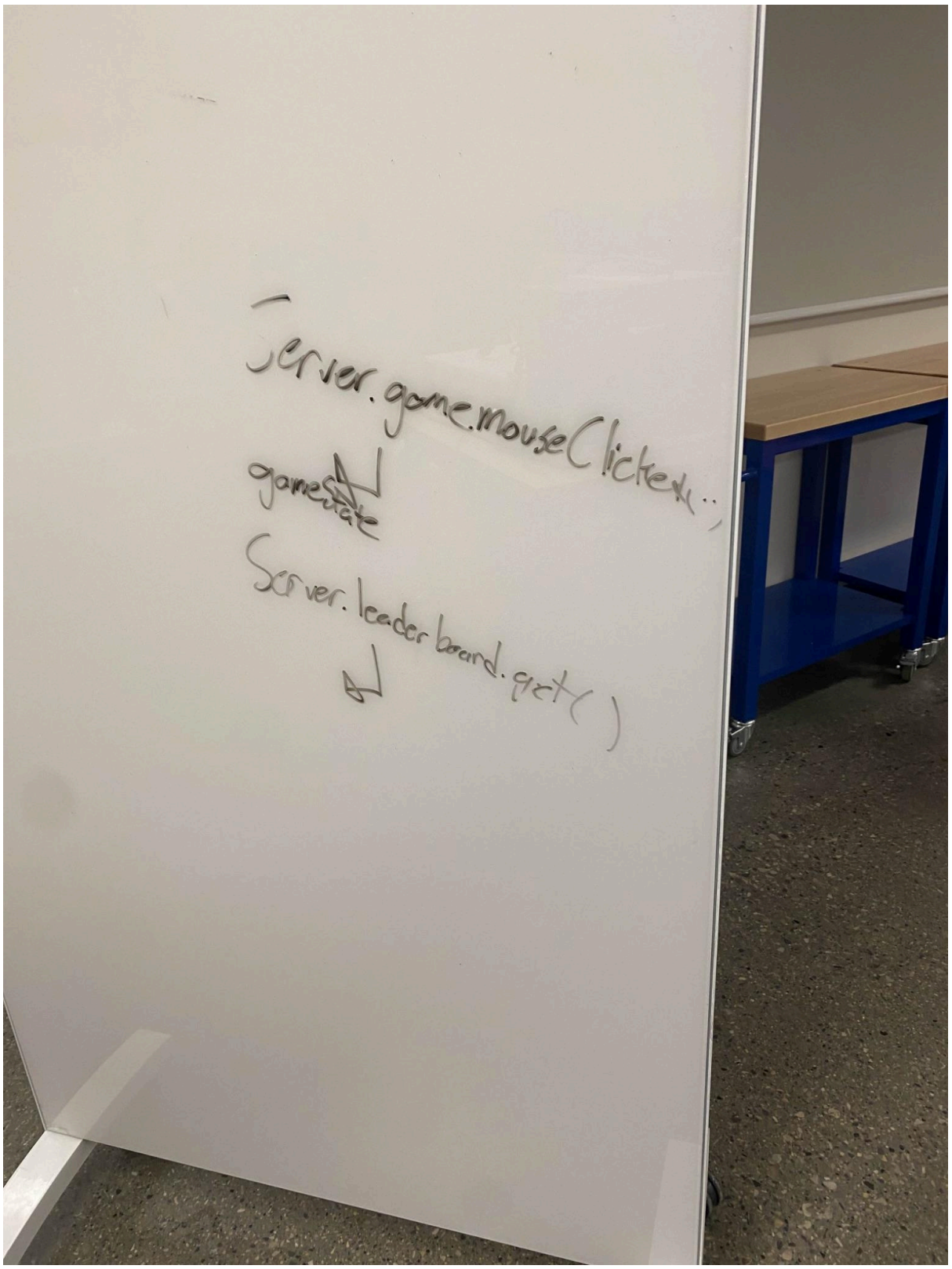
This is basically the subteams. Each subsystem performs a focused task but relies on the Main Loop to gather information or update the game.

- **Purpose**: Allow each team or subsystem to work on a specific functionality, like game rules, matchmaking, or UI.
- **Role**:
    - **Game Logic**: Implements IGameLogic and contains game rules and state logic.
    - **Matchmaking System (My Team ♥)**: Pairs players based on skill, manages queues, and updates the leaderboard via queries to the Main Loop.
    - **GUI**: Displays the GameState retrieved from the Game Logic to provide a seamless, interactive experience.
    - **Networking** (I think this is what they do, I have no clue about Network): Manages communication between clients and the Main Loop, if the game supports multiple clients.

Already mentioned above, it's to make our lives easier. This is a pretty modular design process at least in my books, so it does simplify development. Each subsystem can be developed independently and focus on its area of expertise but still query the Main Loop for necessary data.

Sources (lol)


UI
Res<GameState, Error>
GameState
X=50
y=15
x=20
y=30
Main Loop
ME
DB
Res game state
Leaderboard
Update
LB Query
Game
-MouseEvent
Check Game
Update Server
Grab Server
Render UI
Main
while{
ME = GUI.getMouseEvent()
if Server.inGame()
Server.game.UM(ME)
VI
Server
Game
Leader
Main
Arbitrator
GUI


server.game.mouseClicked(...)
gameState
Server.leaderboard.get()