THE TECHNICAL UNIVERSITY of CLUJ-NAPOCA

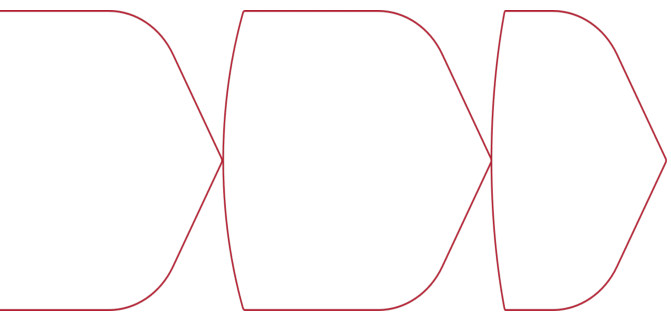
Online Client-Side Chess Game

OOP Project

Gavrila Robert-Daniel

Technical University of Cluj-Napoca

January 6, 2025



Content

Motivation

- **Problem Statement:** I had to implement a project whose goal was to reflect the studied OOP principles.
- **Why This Project?** I transposed my passion for chess into a useful app, trying to add an unique label.
- **Was it easy?** For me, it was a huge task to implement a real-time working app using Java. I started building a Spring Boot application, but it seemed to be a much more complex task that I expected, so I turned my app into a desktop Application running a TCP Server.
- **What I learned from this project?** It was a good change to learn good practices about OOP Principles and working with Java, Spring Boot, React and other tools trying to build my app.

Required Features

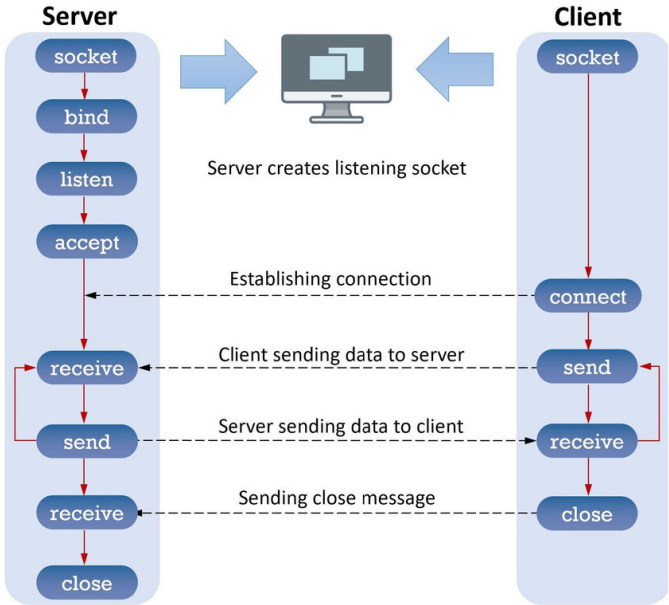
- **Real time communication with between clients.**
- **A Server-Side application which will manage the connections.**
- **The chess logic and the communication business.**
- **A client-side GUI application.**

Implemented Features

1. Play match with a connected random client.
2. Send real-time messages to the opponent.
3. Real-time updated GUI for each player with useful information.
4. Automatic color assignment for players.
5. Display the possible moves for the selected piece (Chess Logic).

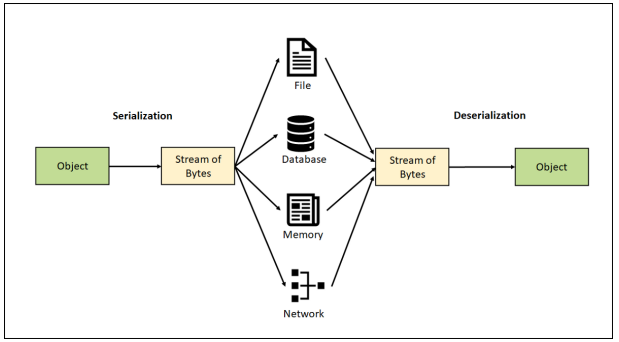
Play match with random client (1)

- I implemented only the feature to play with a pseudo-random player. You can start a new game by the pressing the button **Start** then, our server will search for another Client Connected.
- In technical terms, when we open a socket for communication with the server, it will search another client which is also searching for a game, and then it will match together our requests.



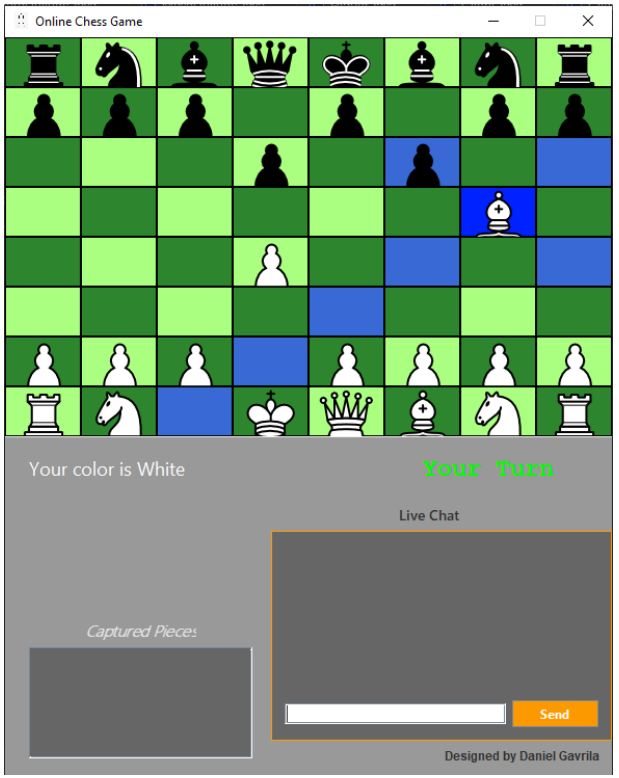
Play match with random client (2)

- For this implementation we have a Client class which consists of Socket, Object Input and Object Output Streams, and a bunch of other unique parameters.
- Using the same IP and Port as the server we can **Connect** and **Send** objects (Messages) through the Server to other clients.
- Each message has its own purpose and content, that is why we have some predefined types of messages such as *MOVE, CHAT, CHECK, etc.* Each message is serialized to be sent and de-serialized to be read.



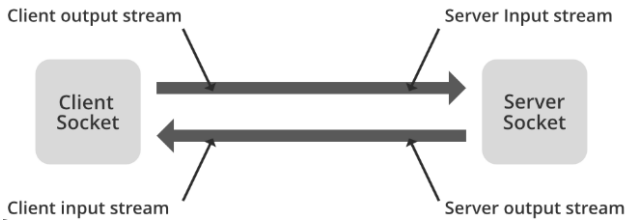
Play match with random client (3)

- After we have reached the connection between two clients, we start the game.
- As we select pieces and moves on the board, practically we send requests and information to our server, which sends back information we need.
- For example, when we select a piece, the server gives us as a response the available moves, as we can see in the picture.



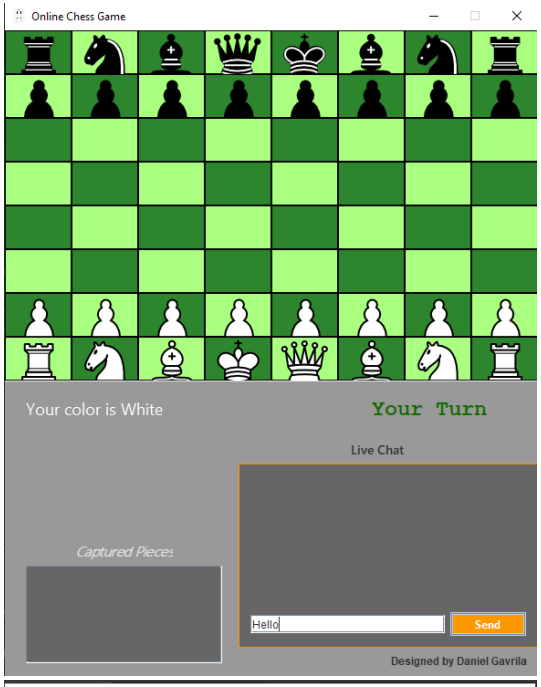
Play match with random client (4)

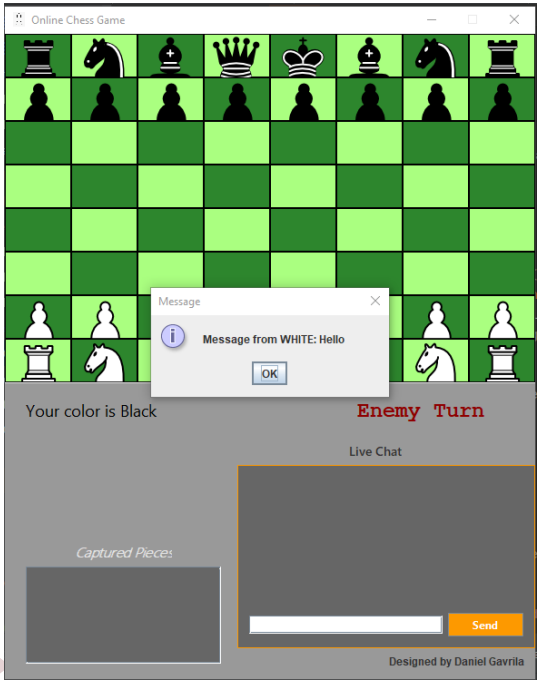
- Our messages have different types, based on the requests we want to process the server.
- These messages are processed by some threads which Listens our commands.
- If we want to create a connection, we have to listen the connection thread for two clients, and then to match them together and to lock the connection.
- If a player leaves the game, our server will respond with a message and will interrupt the connection with previous client.



Send Real-Time messages

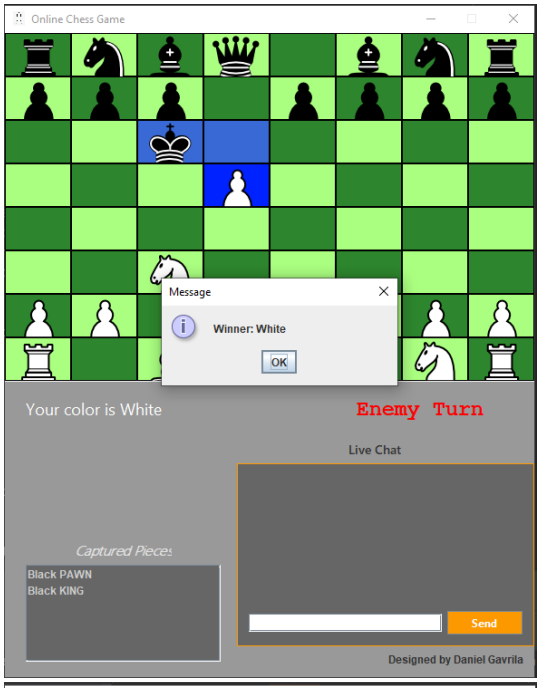
- This is a very useful and entertaining feature which I wanted to add to my project. Basically it allows as to send real-time messages to the opponent as notifications.
- When we press **Send** the GUI saves the Message and sends is to the Client as a notification Pane.
- The implementation is pretty much similar with the other types of messages, but here we send a notification on the GUI.
- The messages are not stored anywhere, because it is not a priority for our game.

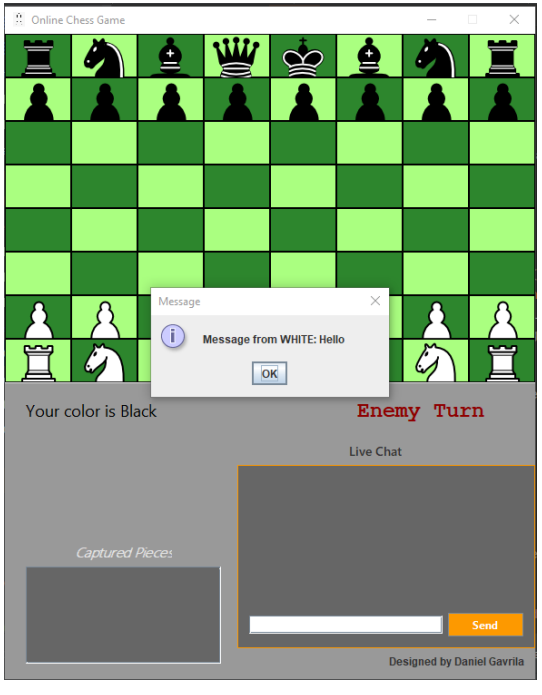




GUI

- For GUI, I used NetBeans Apache GUI Swing Editor, because I found it more helpful and easy to use compared to the Intellij Swing Editor.
- To update the GUI, I used a method of repainting the GUI when a modification tooks place, such as selecting a piece and moving it to a certain position.
- We receive notifications when a player is in Check or when a player wins the game.
- Also on the bottom of the GUI page we have a text field used to send messages, a Section of **Captured Pieces** where we can see which pieces we have captured and we are notified when it is our turn to move.



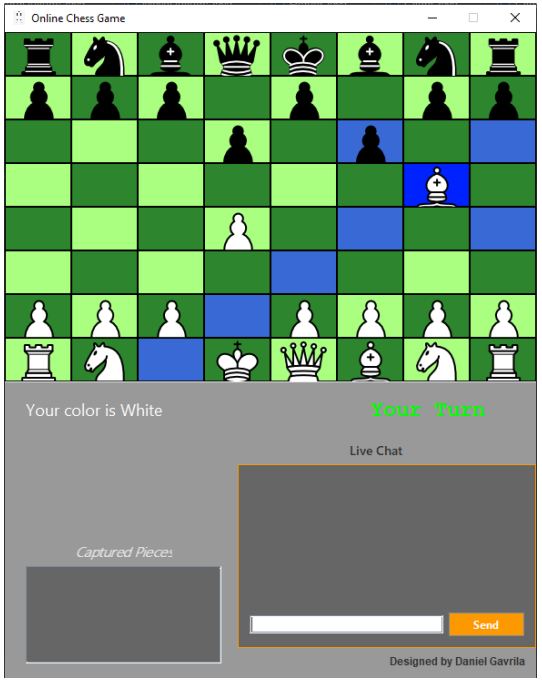


Color Assignation

- After the connection between two clients is made, the server (ClientPairingThread) will assign a color to the clients.
- This color is assigned in the following way: the Client who is the first to establish a connection will have the **WHITE** color, and the other player will have the **BLACK** color.
- This is caused by the logic behind of matching players, because we iterate through all connected Server Clients to match a connection, and the first ones will be in front of the newer requests.

Display the possible moves for a piece (Chess Logic)

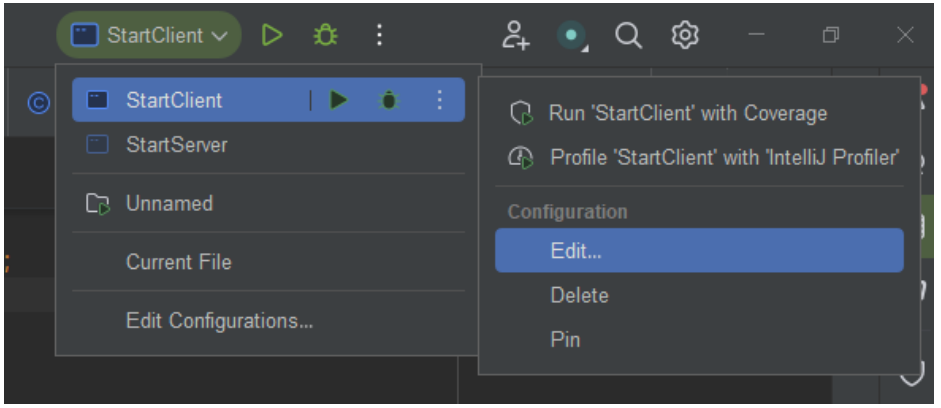
- To highlight on table the path we use a boolean parameter which indicates if the tile is the next move of the selected piece and if it is, it will be colored in blue.
- This is basically the chess logic, because we have to validate each move for each piece.
- We have defined all possible paths for each piece, and then we verify the edge cases for each piece and also the Check states.
- The chess logic is very common and known, I don't think is necessary to explain it on larger scales.

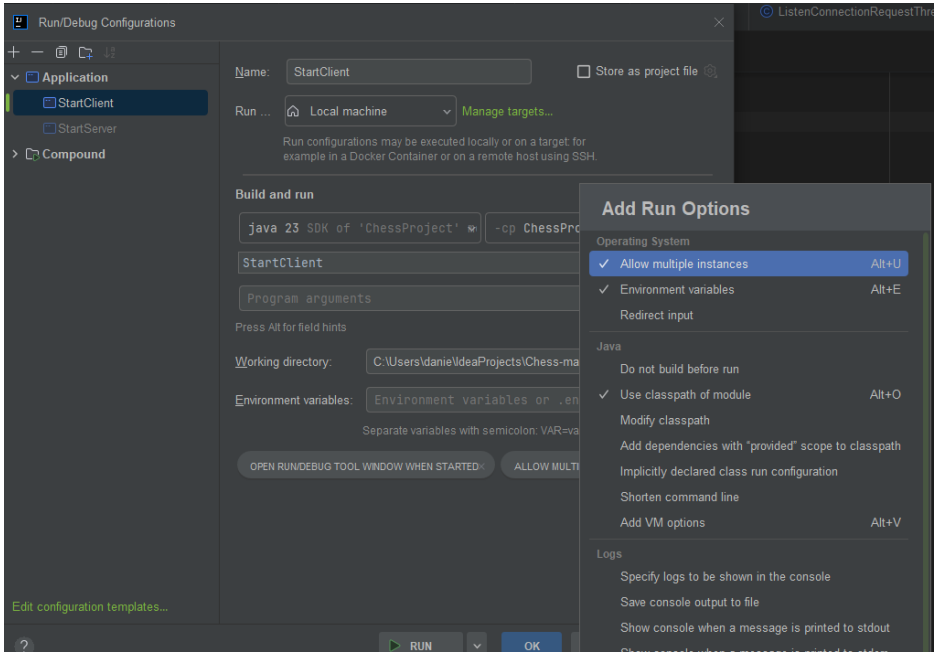


How to run the Application? (1)

Modifications needed

- In the first step, if we use IntelliJ IDEA IDE, we have to configure our Profile/Running configuration for **StartClient** to allow running multiple instances.
- To allow this, we have to select our current configuration, go to **Edit Configuration** $\rightarrow$ **Modify Options** $\rightarrow$ **Allow multiple instances** (or `Alt + U`).

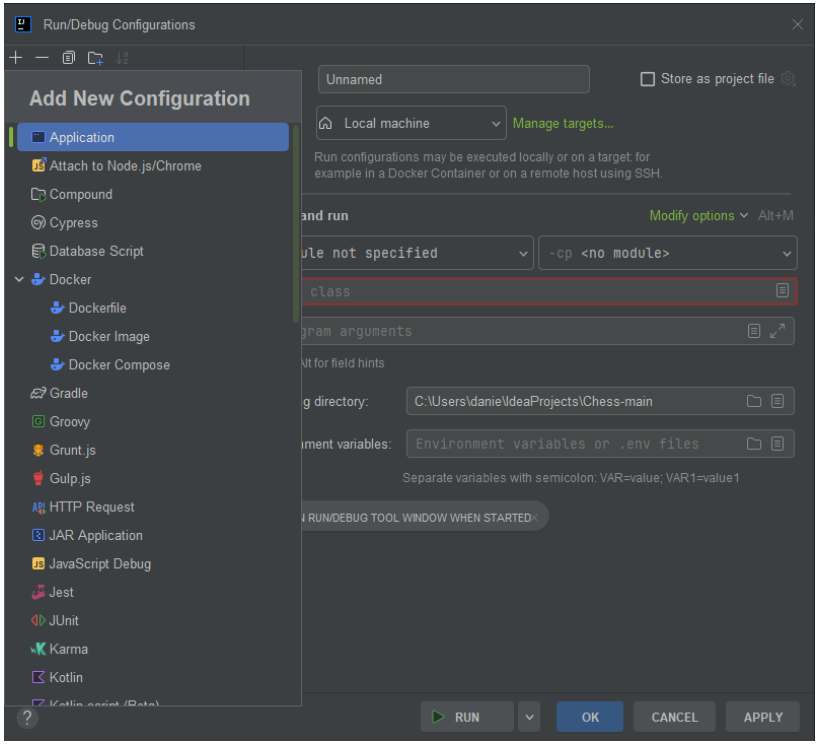


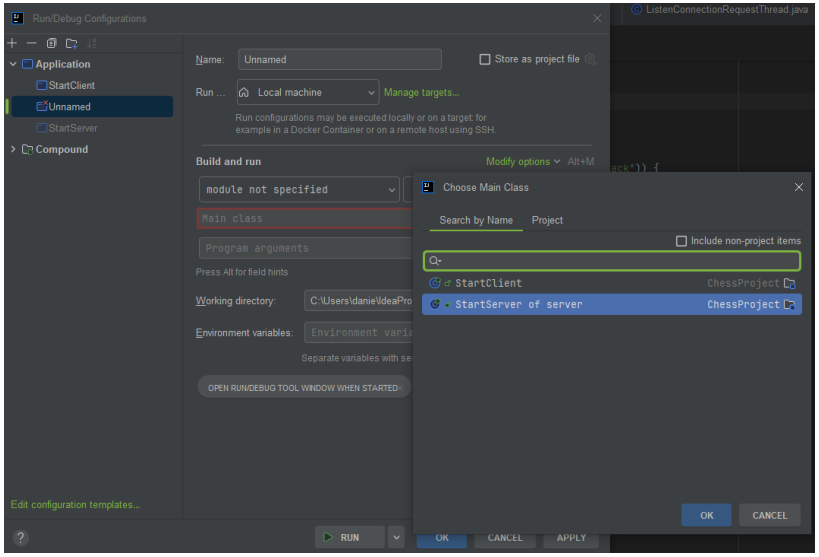


How to run the Application? (2)

Modifications needed

- Secondly, we have to create a profile to start the server as well.
- We have to navigate again to **Edit Configurations** → **Click on +** → **Click on Application** → **Main Class** → **StartServer**

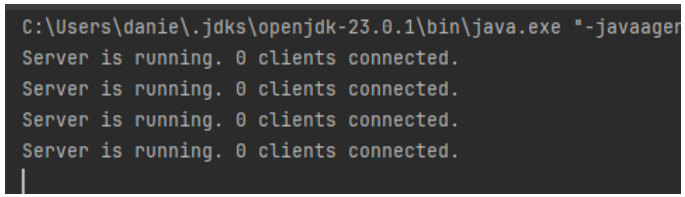




How to run the Application? (3)

Starting the Server

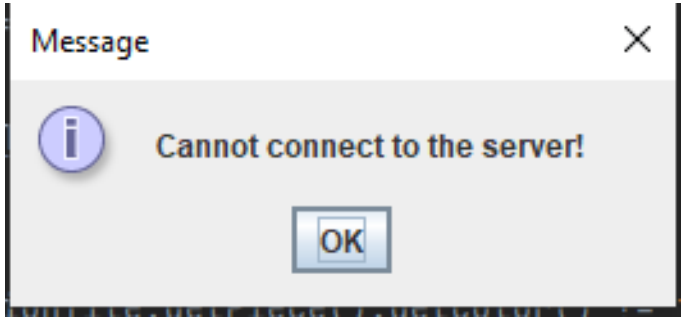
- After we finished all these modifications, we have to start firstly the server.
- So, we have to choose the **StartServer** Configuration and hit the start button.
- The console will start printing the statement of the Server and the number of clients connected.

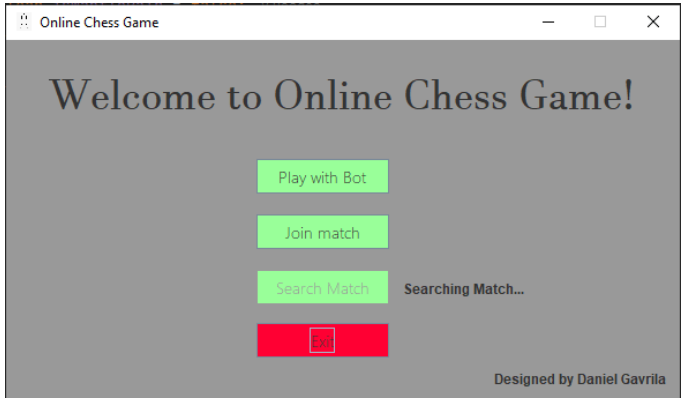


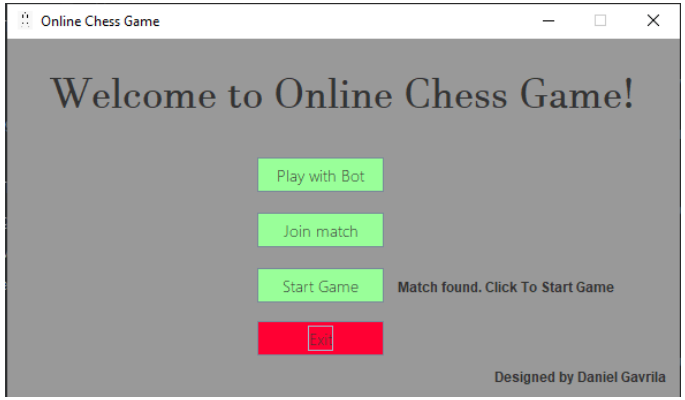
How to run the Application? (4)

Starting the Client

- To start a new client, we have to change the configuration to **StartCLient** and to hit the Start button.
- If we start a new client, before the server is running, we will not be able to enter the application, because we will receive a notifications with the message **Cannot connect to the server!**
- When the menu starts, we have to choose the **Search Match** button, and when a matchmate is found we will press **Start Game** and the game will start.







Future Updates

Disclaimer

- The other buttons as well as the login panel and Database Management are not finished yet, so they are not working until this point.
- There are a few bugs, which I did not managed to solve yet, for example when start a game session and a client leaves it, the server will continue to match it with another client.
- I will do my best to solve and to implement all features in the future, to be a complete project.
- I also want to finish the other projects I started, such as the **BitBoards Chess** and the **Online Spring Boot Chess Application**.
- I have to mention that there are some parts where I took inspiration from another projects on GitHub or GeeksForGeeks.

Final Words

- Thank you very much for your kindly attention!
- I hope that my project was according to your expectations.
- Thank you, Andrei, for your support and for supervising my OOP Final Project!