Chess Visualizer Process Book

4th June 2023

Emma-Andreea Chirlomez
Ioan-Alexandru Tifui
Theodor-Pierre Moroianu

Project Goal

As amateur chess players, we wanted to create a tool one can use to see the effectiveness of various openings, strategies or individual moves throughout chess games and player ***ELOs***.

An opening, a move or a game board that might lead with a high probability to a win for low-rated players might be disastrous against skilled players, or vice-versa. This implies there is no universal ranking of potential moves given a chess board, when taking into account the skill (***ELO***) of the two players. With ***ChessViz***, we aim at providing a straightforward, easy to use visualization tool, where players of any rating can explore historical data of chess games, for their particular ***ELO*** range. For this purpose, our visualizations often include a chess board, where players can examine various chess data.

Architecture and Used Tools

The application we developed follows the conventional architecture of a single-page web application. Its components are:

1. The backend API server, which computes and sends to the UI data to be visualized. The server is written in Python 3, and heavily relies on:
    - ***Flask***, a simple web framework we use for communicating with the clients.
    - ***Python-Chess***, a Python package for parsing ***PGN*** files (chess archives).
2. The frontend server, serving static assets to browser clients, and proxying ***API*** requests to the backend server. This component is generated automatically, and we only wrote its configuration file.
3. The web UI, written with ***Typescript*** in ***React***. For the visualizations, we use:
    - ***D3***, for which we built a custom React wrapper.
    - ***Chessground***, a chess board visualizer created by lichess.org, for which we also built a custom React wrapper.
    - ***BlueprintJS***, an UI toolkit for a nicer user interface.
    - ***Apexcharts***, an UI toolkit for representing heatmaps.
    - ***Recharts***, an UI toolkit for interactive pie-charts.
    - ***React-Simple-Maps***, an UI toolkit for interactive maps.

Backend Server Design

The backend is implemented in ***python 3,*** the reasoning behind this choice being that ***python*** contains really good libraries for both running an ***API*** server and parsing large amounts of ***PGN***

files (**P**ortable **G**ame **N**otation). A good alternative would have been ***node.js***, but we decided to use ***python*** because of our familiarity with the language.

The first iteration of the server was written shortly after ***Milestone 1***, for which we had already written some code, by simply wrapping it within an ***API*** service. The current design is quite flexible, allowing the addition of new data / statistics by:

1. Defining the internal format used between the ***API*** and the ***Web UI*** to transmit the data.
2. Add code that computes the wanted statistics, based on the parsed games (which requires an initialization stage called once, a generation stage called for each parsed game, and a saving stage called at the very end which dumps the generated data).
3. Expose an ***API*** route that serves the new data.

This flexible server architecture allowed us to use an Agile-like development lifecycle, adding new features as they were required by the ***Web UI***.

Frontend UI Design

For having an organized, easy-to-follow, and strongly typed project, we opted to use a combination of ***React*** and ***Typescript***. As mentioned above, we used a large variety of existing tools and modules, taking advantage of the amazing ***npm*** and ***React*** ecosystem.

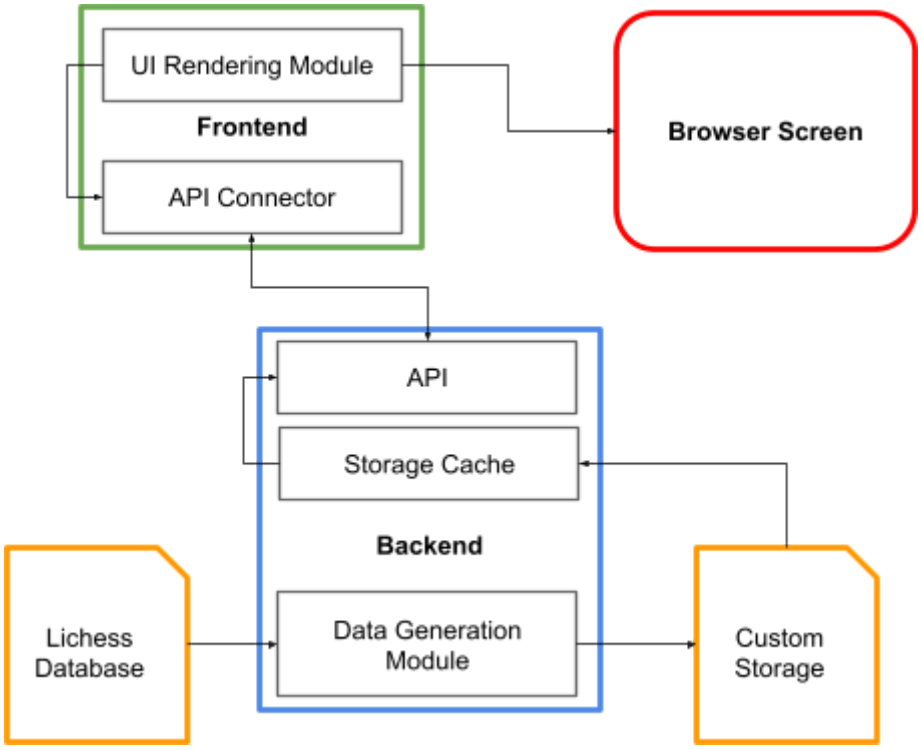
Due to the well-defined hierarchy of ***React*** components, offering a good compartmentalization of the project, we were able to each work on different tasks, without getting conflicts between us. The ***Web UI*** grew organically throughout the semester, each of us working whenever we could.

One notable mention is that the entire website is made from scratch. We did not use a premade template, but rather designed and implemented the pages and tabs as we thought it best fit our vision. We believe the final design is intuitive and easy-to-use, while keeping a professional and appealing appearance.

Generating The Data

Due to the sheer size of the dataset we are using (***24GB*** of chess games, given in the ***pgn*** standard format), the ***API*** cannot generate the requested statistics on-the-fly, and the ***Web UI*** is even less.

To address this issue, we parse the dataset during a generation phase, and convert it to our own internal representation, which can easily be serialized and deserialized to and from disk. Together with a cache implemented in the server, this design allows for an efficient and responsive ***API***, able to serve compact preprocessed data to the ***Web UI***.



Due to the inherent slowness of ***python***, the computational complexity of generating the statistics and data we are interested in, and more generally the sheer amount of data to process, we opted to run the computation in a ***Cloud-Based VM***. We are currently using an ***Ampere A1 Compute*** host, with ***4 ARM*** cores and ***24GB*** of ***RAM***.

Target Audience

The target audience of this website are people interested in learning more about chess. We are not chess experts nor competitive chess players, but thanks to the data we are able to extract from chess games, we believe anyone can benefit from the visualizations we provide. Some of the information we were able to infer are that:

- As players climb up the ***ELO*** ladder, they tend to draw more often. This can be explained by the fact that better players make fewer mistakes and chess tends towards a draw if both players play close to perfection.
- The games of lower-rated players tend to be shorter. A plausible explanation is that weak players often make major blunders (mistakes) in the early stages of the game, which lead to quick losses. This should encourage beginners to study the opening principles in order to improve.
- Low-rated players often play openings such as the ***Wayward Queen Attack*** or the ***Van't Kruijs Opening,*** which might lead to quick checkmating attacks against unprepared

opponents. These kinds of openings are, however, never used at a higher level as they lead to a difficult position to play if the opponent does not fall for the trick.
Good players go for well-studied openings such as the ***Queen's Gambit*** or the ***King's Indian Defense*** which can have hundreds of variations.

- Looking at the heatmap of piece placement along a game we get perhaps the most interesting insights. For instance, low-rated players play ***E4*** more often than ***D4*** on the first move.
  This explains some apparently strange facts about piece placement, like the ***white*** developing their ***dark-squared bishop*** on average much later in the game than the ***light-squared bishop***. The visualization also shows the best squares to place certain pieces, for example, ***knights*** are rarely placed on the edge of the board. Finally, the heatmap also reveals information about endgame strategies of high-rated players, such as ***king activation***, avoiding pawn islands or placing the rook on the 7^{th} or 2^{nd} rank in the opponent's half of the board.

Peer Assessment

The project was developed in brainstorming sessions, peer coding f2f or over video conferencing, and small discussions throughout the semester, so while we believe we contributed equally to the project is it quite hard to give an exact breakdown. Additionally, metrics such as commits / ***LOCs*** are not representative, as the ***Git*** repository contains both code and data, backend is harder to develop, frontend ***CSS*** / styling is time-consuming, configs are tough to set but only result in a few lines of code, many parts of the project such as brainstorming do not result in visible artifacts, etc.

As mentioned above, we believe we worked equally on the project. However, a (very rough) breakdown of the project components is:

- Project idea (chess statistics) - Alexandru.
- ***Milestone 1*** - Everybody, over a video conference.
- Project setup (creating and configuring a ***POC*** skeleton project connecting the backend and the frontend) - Theodor.
- ***Backend API*** and ***data generation*** - Everybody.
- ***Home*** page - Theodor.
- ***Game of Chess*** page - Emma.
- ***ELO Rating*** page - Theodor.
- ***Advanced Analysis*** page:
  - Page menus configuration - Emma.
  - ***General Information*** tab - Emma.

- ***Openings Frequency*** tab - Emma.
- ***Player Victory Rate*** tab - Theodor.
- ***Pieces Placement Through Games*** tab - Alexandru.
- ***Representative Player Evolution*** tab - Alexandru.

We stress once again that the division made above is extremely rough, and does not account for peer programming, code review, exploring new ***UI*** libraries, trying new concepts, help between the team, code changes in components written by another teammate etc.

User Interface of Chess Visualizer

We opted to have a standard ***Server - Client*** architecture as it allows for a better isolation between data collection and processing, and visualizations. This makes deployment more difficult, as the ***Web UI*** is dynamic, and cannot rely solely on static assets, which makes it impossible to deploy it as a ***Github*** page. Thus, we manually deployed it to our own domain, and self-hosted it on the same cloud machine we use for generating the data. The website is accessible at preausor.cf. Please note that because the scope of the project is not software engineering, and users provide no sensitive input to the page, we did not bother to generate and register a ***TLS*** signature, and the website is only available through ***HTTP*** (instead of the more secure ***HTTPS***).

The ***Web UI*** itself (i.e. the content sent to the client when the page is loaded) does not include any data, making it load and render very fast. When a visualization is generated by the user, the app makes ***API*** calls, requesting from the ***API*** only the required data, making the rendering visualizations very fast and responsive, especially considering we are parsing over ***24GB*** of chess data for our project.

The application has a menu at the top, where users can select one of multiple views. When a user selects a visualization, the main content of the page is replaced by the visualization. Additionally, some visualizations are split-screen, where the left part of the screen is an interactive visualization users can click on to see additional information on the right part of the screen, such as moves on the chessboard or another visualization.

Creating The Visualizations

For representing our chess data in a meaningful way, we took inspiration from the following lectures:

- ***Histograms. Pie charts, Line charts*** - Lecture "*Do and don't in viz*".
- ***D3.js Library*** - Lecture "*D3.js*".

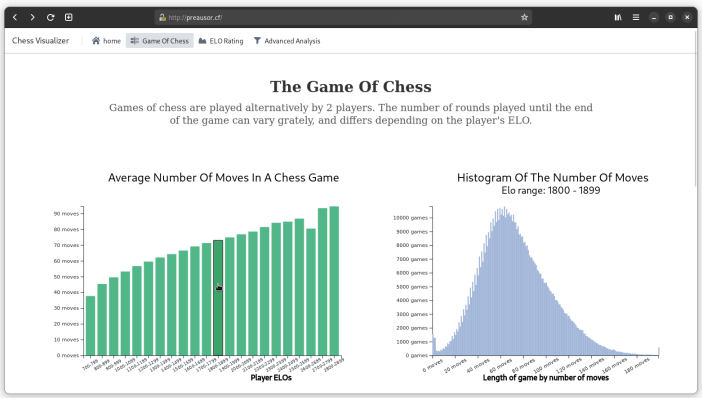
- ***D3 interactive components*** - Lecture "*Interactions*", Lecture "*More interactive d3.js*".
- ***Formatting and colors*** - Lecture "Perception colors".

The visualizations are either written in ***D3*** (for which we implemented a custom ***React*** wrapper layer), or using native ***React*** libraries, such as ***ApexCharts*** or ***Recharts***. This allows us to embed them within ***React*** components natively, making the transition between pages or views more fluid and, when applicable, animated.

List Of The Visualizations

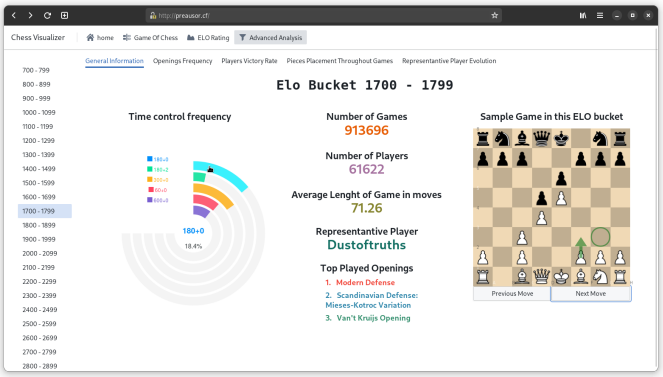
Throughout the project, we implemented a wide range of visualizations, including:

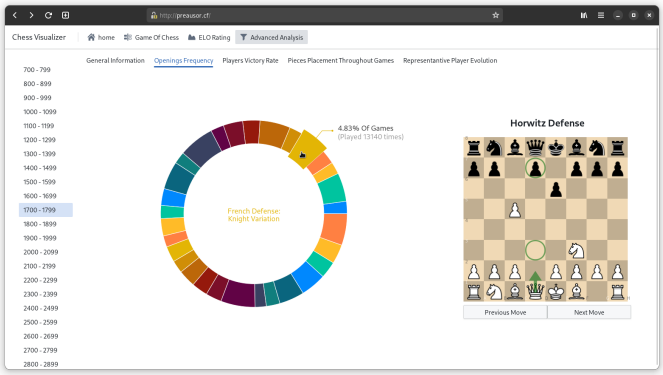
- An interactive map displaying for each country the highest rated player ever, with data taken from parsing to ***FIDE*** statistics.
- An interactive wrapper over a chess board view, allowing users to view chess games.
- Histograms with the number of games / number of moves per ***ELO*** rating range. The histogram is clickable, displaying for each bucket a random game played between two players within the ***ELO*** range.
- A pie chart displaying the most common openings for each ***ELO*** bucket.
- A histogram displaying how players of different ***ELOs*** defeat each other within a single ***ELO*** bucket.
- A dual histogram, displaying for each piece and state of the game the average placement of that piece from the perspective of both white and black.
- Three pie charts that show the win/loss/draw ratio for a

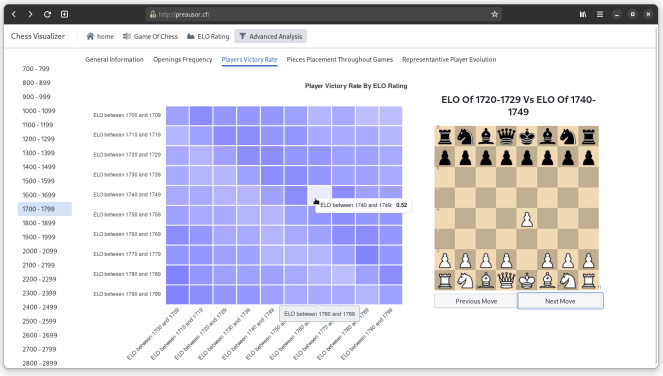


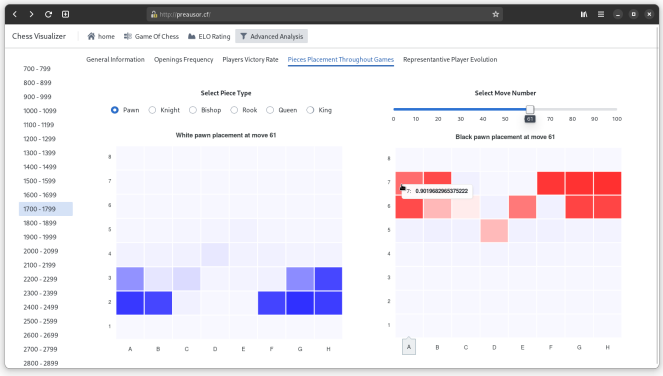
representative player of each ***ELO*** bucket, when playing with the white pieces, the black pieces and in general.

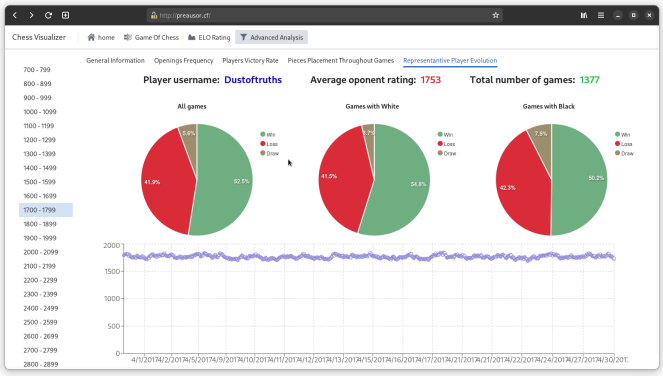
- A line chart that showcases the rating evolution of a representative player for each ***ELO*** bucket.







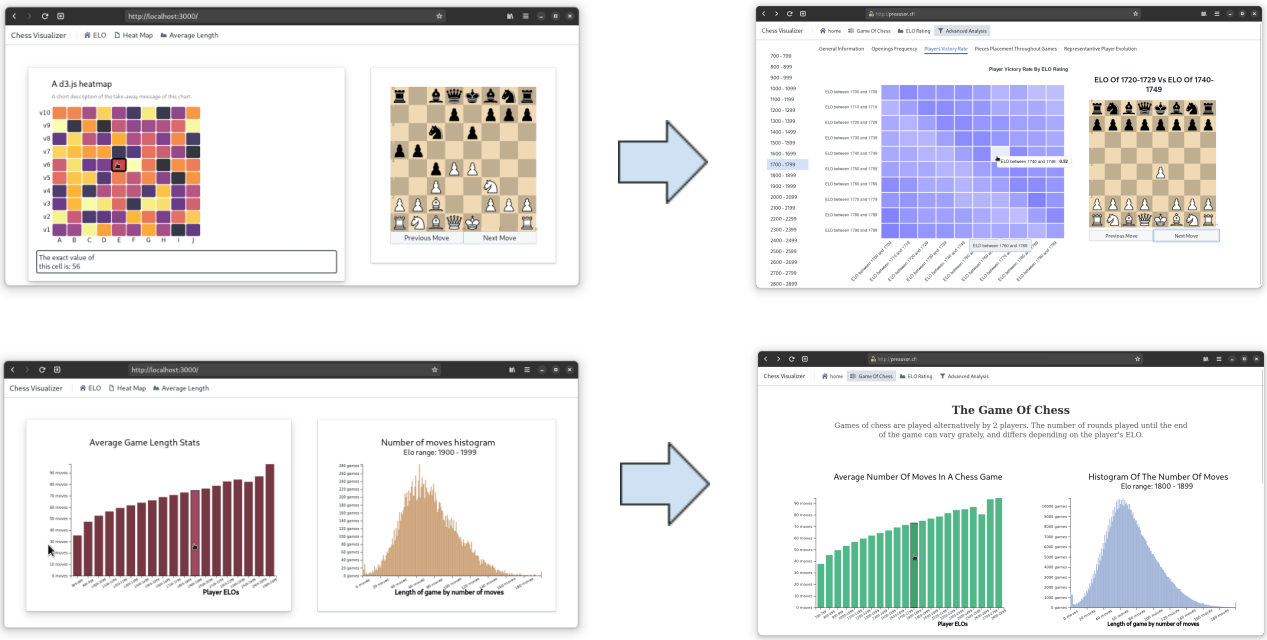




Differences From Milestone 2

The project changed quite drastically from ***Milestone 2***. The biggest difference is simply the number of added visualizations, and the quantity of data we process.

One nice example is how we switched from ***D3*** to ***ApexCharts*** and ***Recharts***, for obtaining more polished, easier to use visualizations.



Running The Project

The easiest way to view the project is to go to http://preausor.cf/.

If you wish to run it locally, you can follow the instructions provided in the ***readme.md*** file within the ***Git*** repository.

1. Clone the repository.
2. Start the backend server by running:
   ```
   > cd backend && python src/main.py run-api
   ```
3. Start the frontend server, by running:
   ```
   > cd frontend && npm i && npm run start
   ```

The app should then be accessible on http://localhost:3000/.