Process Book - Easier Openings

Contents

Overview and Motivation:

Basic Info

Project title: Easier Openings

Name: Ethan Stanley

Email: u1312964@utah.edu

uID: u1312964

Repository: https://github.com/ethanstanley3/dataviscourse-pr-easieropenings

Background and Motivation

This project is a tool to explore chess moves. Specifically, one can examine the most popular and effective moves from any board state. It can also filter the games in the dataset by skill level.

The goal of this project is to create a tool that can determine the effectiveness of chess openings for specific skill levels. Many chess players study openings that are theoretically strong (effective with high level play) openings, but these openings may not work well for beginner or intermediate players. They may be unforgiving to mistakes or fail to capitalize on mistakes that beginner or intermediate opponents are more likely to make. This tool can be used to find openings are easy to play at a given skill level.

I chose this project because I enjoy playing chess but I suspect I am not good enough to play advanced openings. I am also interested in game strategies that are suboptimal against an optimal opponent, yet more effective against imperfect opponents.

This project can be used to study chess or to answer questions about the relationship between chess skill, move popularity, and move win rate.

Related Work:

I was inspired by a visualization shown in class demoing a dynamic pie chart. When clicking on a wedge of the pie, the chart would show data contained in the pie chart wedge. I thought this was an interesting way to visualize a tree data structure. Thus, I figured I could apply this technique to chess's game tree. I also planned on using the concept of linked visualizations to create a powerful tool out of simpler components. We used this technique on many of the homework sets.

Questions:

The primary question I am trying to answer is: "what chess openings are most effective at lower skill levels?". I would like a visualization that allows the user to explore what moves are popular or successful from a given position and be able to choose the skill level of the games in the dataset. That way, one could filter for intermediate games to see which openings and moves are

effective at the intermediate level, as an example. A benefit is that people, including myself, could study and learn openings that consider the skill level of the players using them.

I am also curious to see if my visualization will reveal openings that are a) only effective at high levels b) only effective at lower levels or c) effective across all skill levels. This would be useful information to know when deciding which openings to study further. The visualization could answer questions such as “which openings will be effective right now?” and “which openings will continue to be effective as I improve at chess?”.

As the project evolved, a question I encountered is: “how much do opening moves affect the outcome of the game for each skill range?”. This is not easy to answer with traditional opening explorers because they only provide data on master-level play.

A question that evolved over the course of the project was “What are openings that are effective with high level play, but ineffective with weaker play?”. I found that there were fewer examples of these openings because advanced players have lower win rates in general (draws are more prevalent). As a result, I become more interested in the opposite question: “what are some openings that are widely used and effective with beginners but ineffective in advanced games?”. There are many more examples of this.

Data:

I collected my data from https://database.lichess.org/#standard_games.

This is an online database of all chess games played over the internet on lichess.com. There are currently around 3.7 billion games in the database. This dataset is in PGN format, which is a standard plaintext notation for chess games. The dataset is free for anyone to use.

I had to do a lot of data processing. I wrote a python script (process_data.py) that reads the PGN file containing 3.7 billion games and turns it into a JSON tree that powers the visualization (each node in the tree represents a state of the chess board). It does this by reading the N first games that match the specified filters (such as player skill level) and then iterates through each move of the game, adding nodes to the JSON tree when a novel move is played and updating a node’s win and usage rate when it already exists. Afterwards, I performed a depth first search algorithm to prune all the nodes in the tree that have were played only once. That way, only aggregate data is stored. This dramatically reduces the size of the data files while preserving important data.

I processed 500,000 games without filters (this file is loaded into the visualization by default), 50,000 beginner games, 50,000 intermediate games, and 50,000 expert games. Therefore, there are four unique trees that can be loaded into the visualization.

I chose to preprocess the data into JSON files rather than doing it dynamically because the raw data is far too large to upload to GitHub (it is over 250 GB). The JSON files, by contrast, are much smaller and can be loaded on demand without slowing down the visualization.

Exploratory Data Analysis:

I explored my data by generating a preliminary version of the JSON tree that powers my visualization. I logged it to the console and then expanded through the nodes to see if the amount of data I could feasibly process would generate a game tree of sufficient depth. I found that lines that are played by a high percentage of people can be explored very deeply (sometimes up to 50 moves). On the other hand, sequences of uncommon (bad) moves can run as low as 5 nodes deep. I think this is more than sufficient to explore openings because bad moves are almost certainly not worth exploring deeply anyway. I also tested generating trees of various sizes and found that the size of the JSON tree is logarithmic against the number of games processed. This assured me that I can process many games without creating a tree so large that it dramatically slows down the visualization.

I was confident that the construction of a JSON tree was a good way to approach this problem when I was able to get a preliminary version of the dynamic pie chart working.

Design Evolution:

Planned design:

I have included 5 sketches on the following pages: a brainstorm of elements to include, 3 prototypes exploring different aspects of the brainstorm, and a finalized version that incorporates the best components of the prototypes.

A general idea I explored in the brainstorm is how the user should interact with the visualization. First, I considered the most obvious mode of a chess board on which the player can move pieces. This would be the easiest input for a chess player to pick up, but it is hard to display meaningful data on the board. It gets cluttered too quickly. I explored this idea in prototype 2. Another option is to represent the possibilities in the chess game as a tree. The user could click on a node and the tree would dynamically expand to show subsequent possibilities. I could encode data (like move popularity or win rates) with the radius of the nodes, width of the edges, or color of the node. But this is problematic because it is hard to look at a path through a tree and relate it to a chess board. I explored this design choice more in prototype 1. Lastly, I considered a nested pie chart, where slices of the pie chart could be clicked and more slices would appear, designating possible moves. The area of the slices would encode the usage or win rate of the move. This idea was explored in prototype 3.

Prototype 1:

In this prototype, I explored using a tree to represent the moves of the chess game. This has the advantage of effectively showing all past moves, instead of just the current state of the board. The relevant data (move popularity or move win rate depending on what the user is interested in) would be encoded by the widths of the edges or the areas of the nodes. Width is one of the most effective visual encoders, so it would be effective. Area is not quite as effective, but it could be used to encode the metric that the user is less interested in. For example, area could encode win

rate, while the width of the edge could encode usage rate. That way, the marks (nodes and edges) would efficiently encode the relevant channels. One problem with this design is that the screen could become cluttered by having too many nodes. On average, there are 20 possible moves from a given chess position. I included a table at the bottom of the visualization that can include more relevant data about the most recently selected node. This design uses sliders to filter the skill level of the chess games being visualized.

Prototype 2:

This prototype explores using a chess board as the main visual component. The possible moves would be encoded by arrows on the board. The relevant variables would be encoded by the width of the arrow. For a less important variable, I could use the opacity of the arrow. Opacity is a much worse channel than area or width, so I think this design less effectively encodes the data. Additionally, it suffers from being cluttered. Many arrows would overlap. This would make it hard to select which move to explore. To overcome the limited effectiveness of the encoding channels, this design would need a table containing relevant data at the bottom.

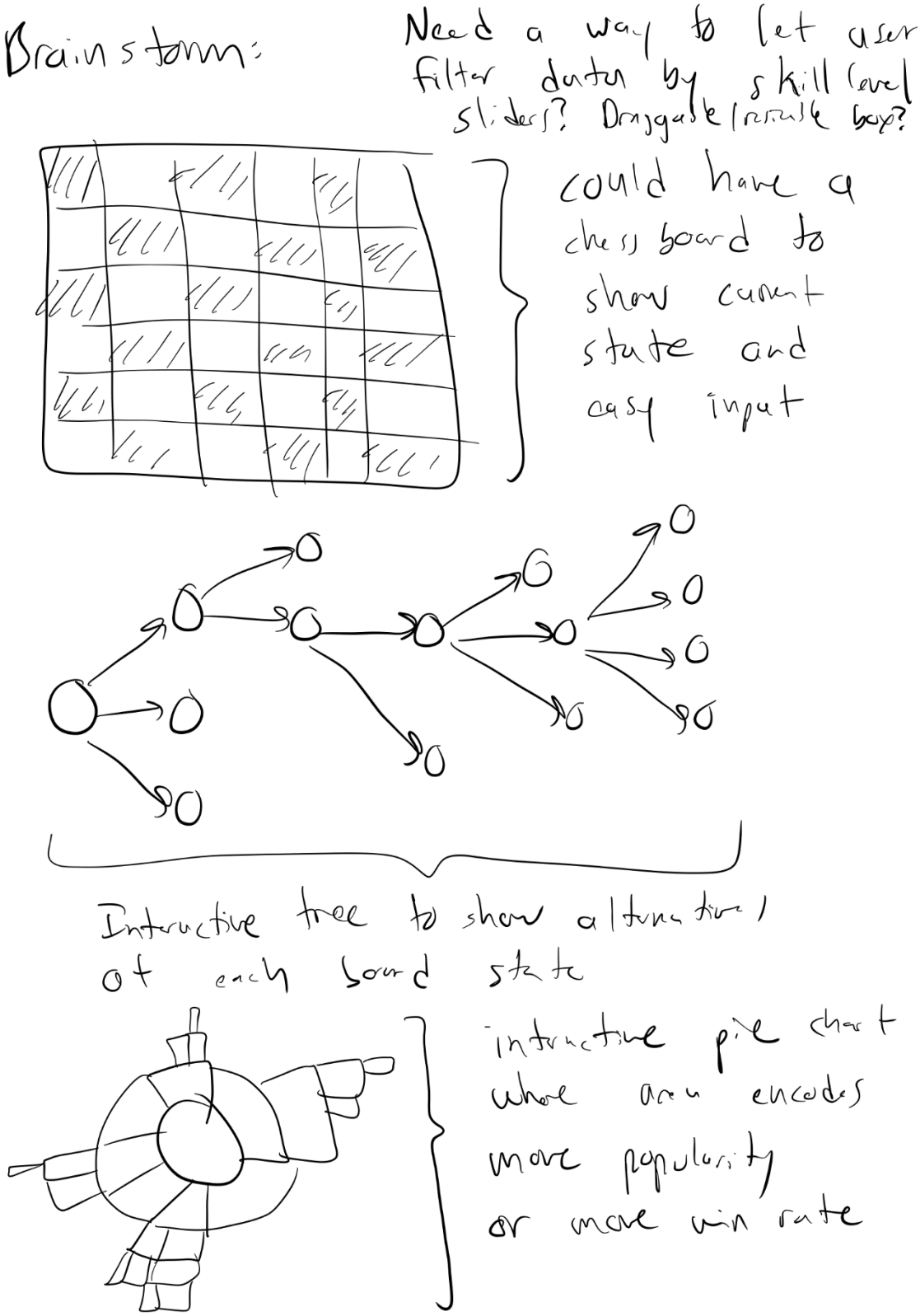
Prototype 3:

I explored a dynamic and layered pie chart in the design. Each slice of the pie chart represents a possible move, where the area of the slice encodes either the win rate or usage rate of the move (depending on the user's preference). When a slice is clicked, more slices appear above the clicked slice that encode possible moves. I have two channels to work with: slice area and slice color. These could be used to encode my two variables. I would use area to encode the more important variable. Although area is not as effective as width for encoding data, the slices of a pie chart will be easier to compare than the width of the edges of the tree found in prototype 1. This is because they share a common axis, unlike the edges of the tree.

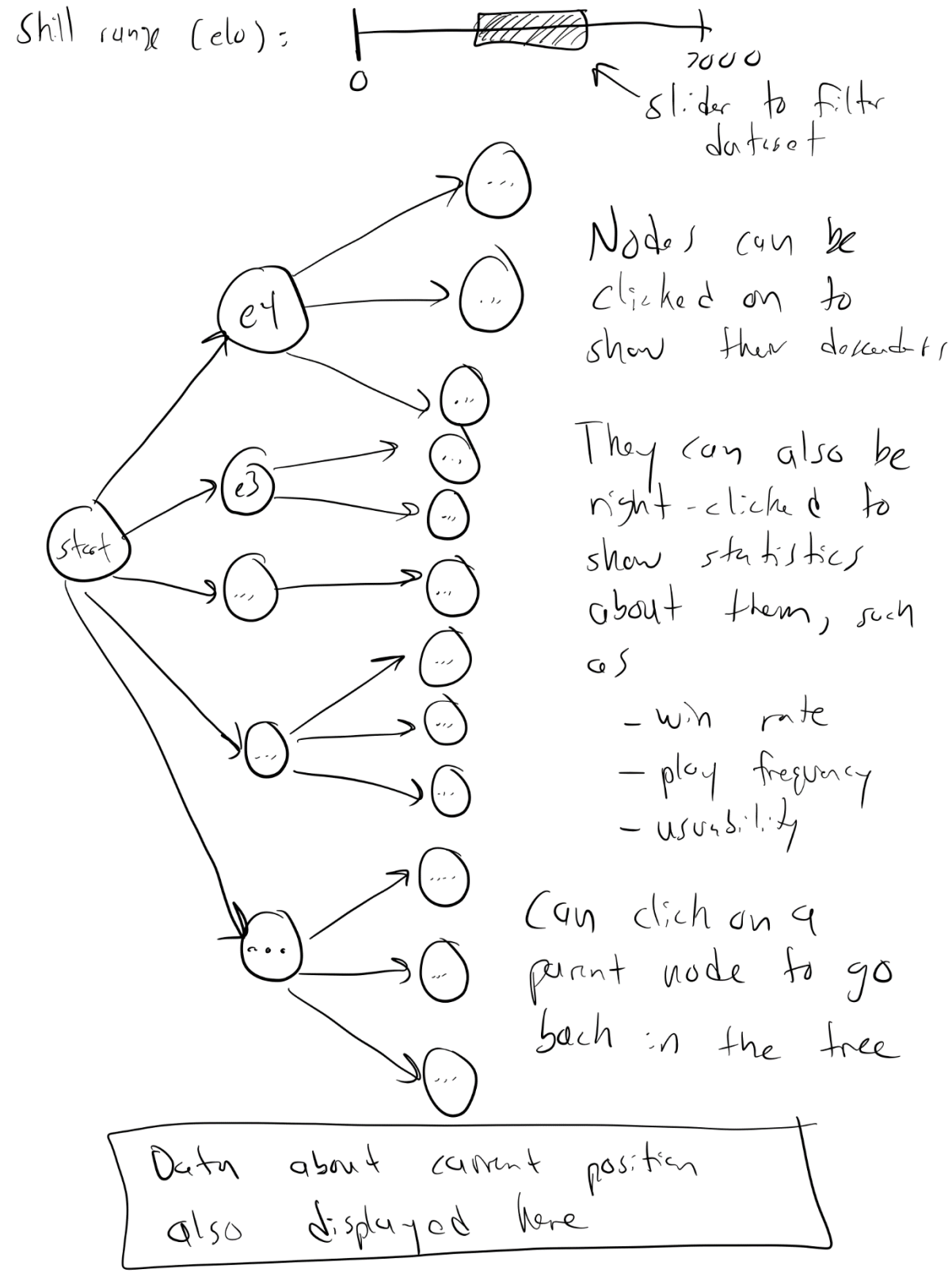
Finalization:

From these drafts, I kept the dynamic pie chart because I believe it is the best way to compare all the moves from a given state. I also kept the chess board as a tool for inputting moves and displaying the current state of the board because this is what any online chess player would be most comfortable manipulating. To filter the games in the dataset, the user can drag and resize a rectangular brush too. This is critical to fulfill the visualization's purpose of showing effective openings for any skill level, rather than just experts. I believe this combination most effectively encodes the data about a given move while still being easy to manipulate.

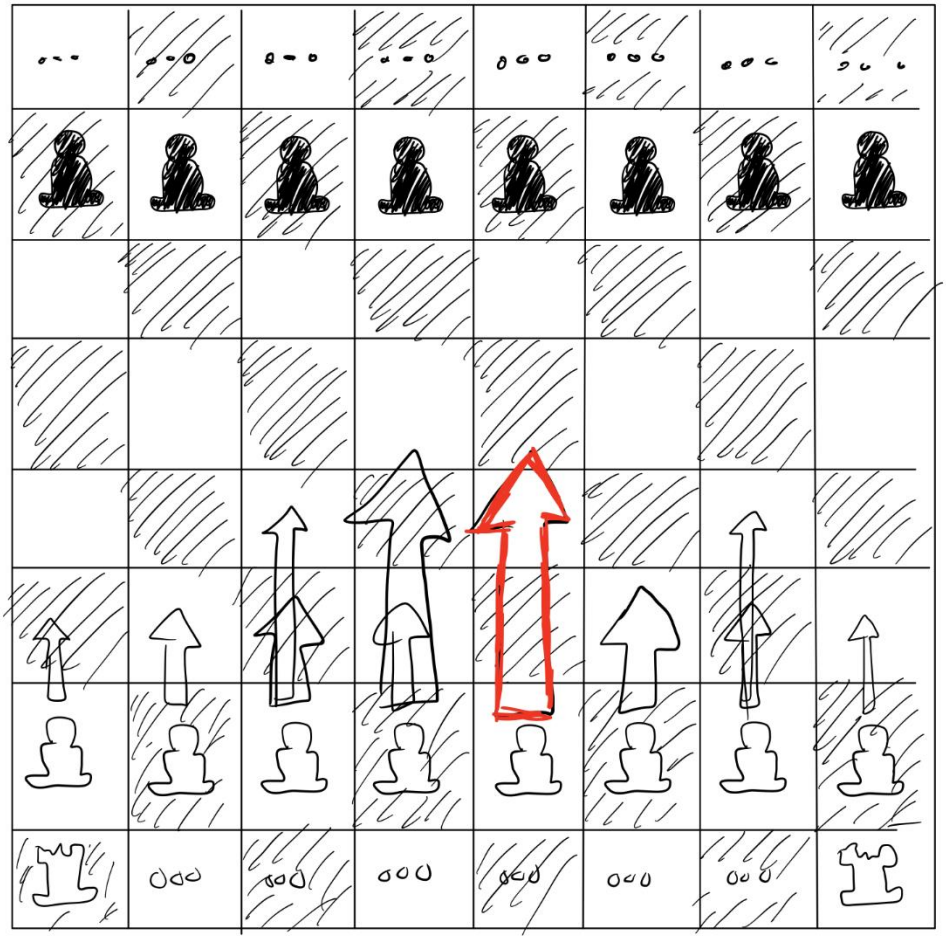
Brainstorm


Brainstorm:
Need a way to let user filter data by skill level sliders? Drop down/radio box?
could have a chess board to show current state and easy input
Interactive tree to show alternatives of each board state
interactive pie chart where area encodes more popularity or more win rate

Prototype 1


Skill range (elo):
0
2000
slider to filter dataset
e4
e3
start
Nodes can be clicked on to show their descendants
They can also be right-clicked to show statistics about them, such as
- win rate
- play frequency
- usability
Can click on a parent node to go back in the tree
Data about current position also displayed here

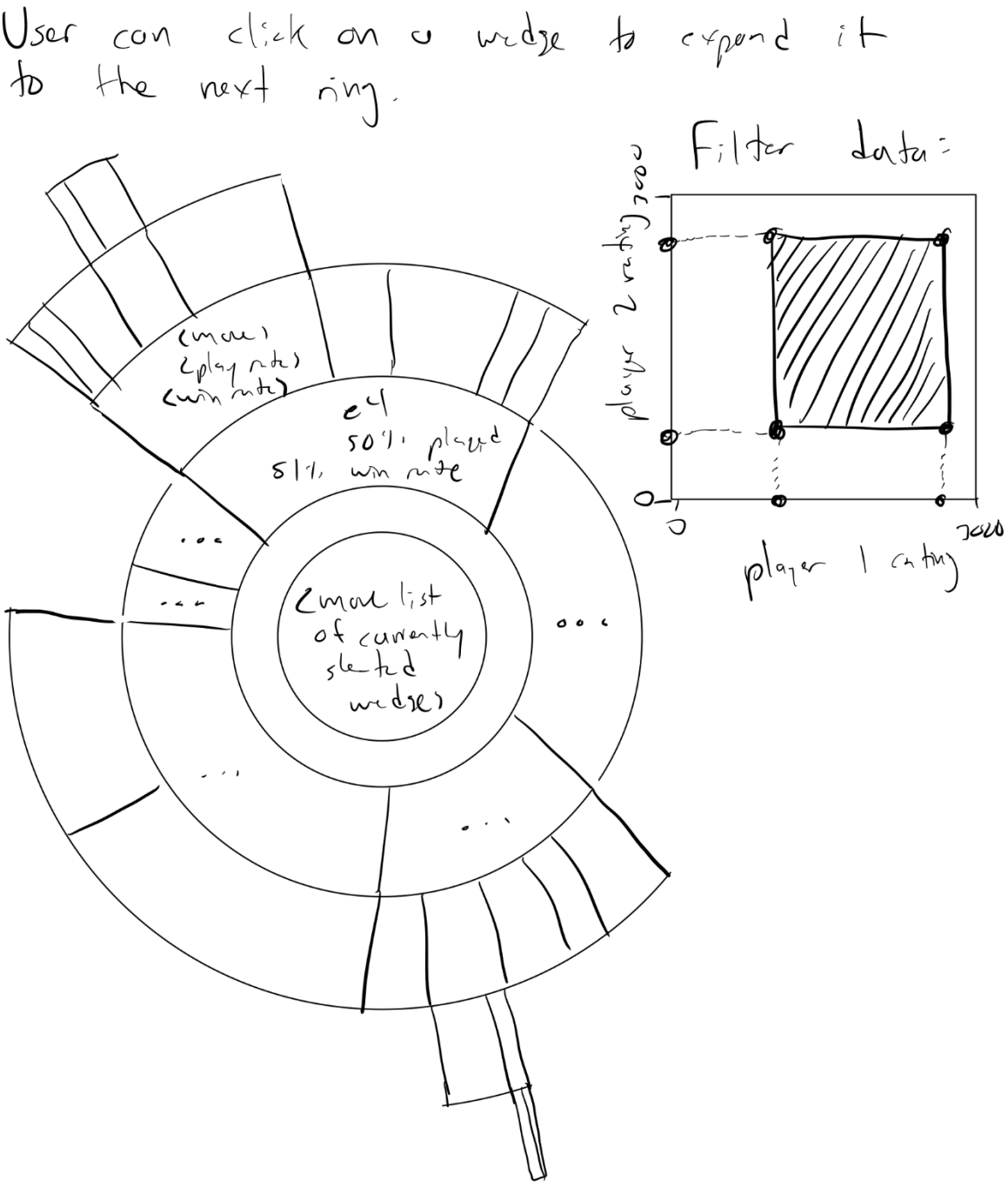
Prototype 2



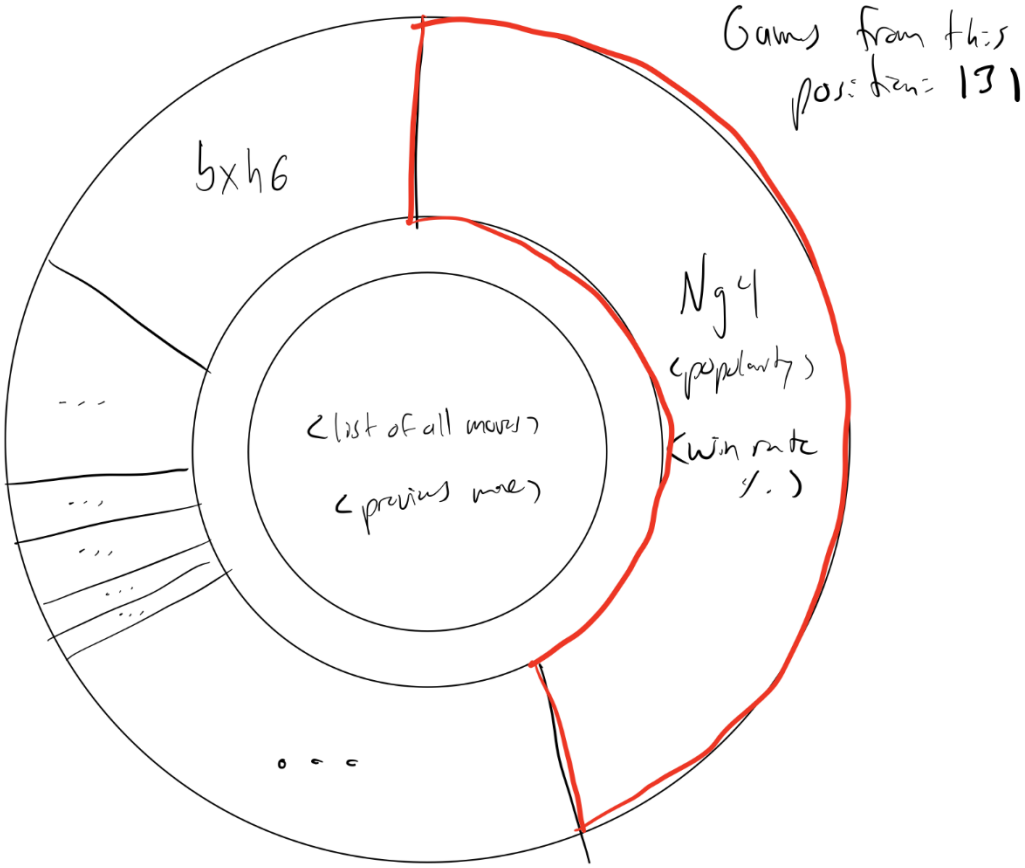
Popular moves

mal	# played Δ	success rate	avg elo of user →
e4	60	X	X
e3	10	X	
d4	20	X	
d3	..	X	
...	...	X	
...	..	X	

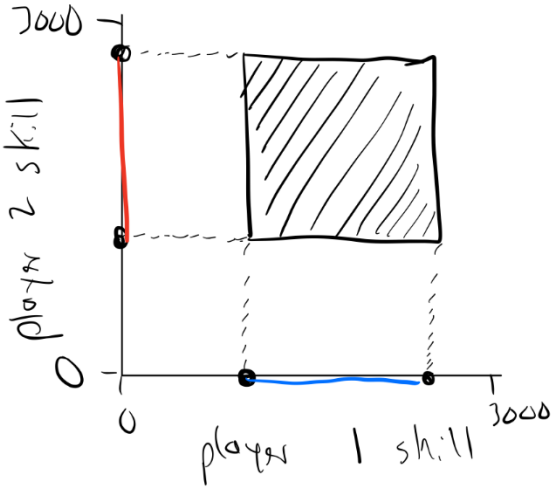
Prototype 3

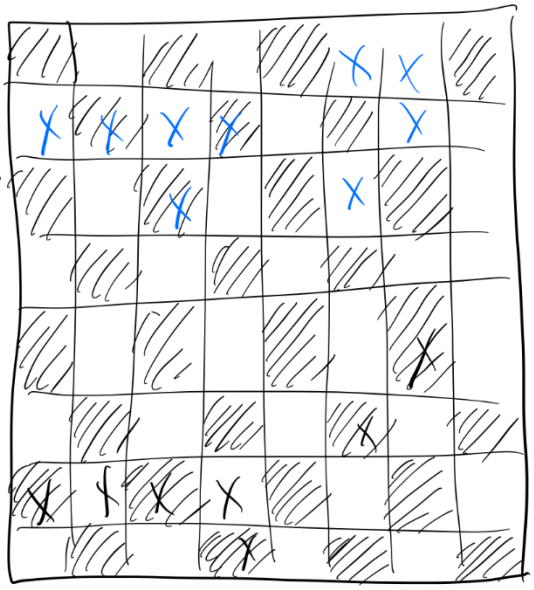

User can click on a wedge to expand it to the next ring.
(move)
(play rate)
(win rate)
e.g.
50% played
51% win rate
[move list of currently selected wedges]
Filter data:
player 2 rating
player 1 rating
0
3000

Finalization



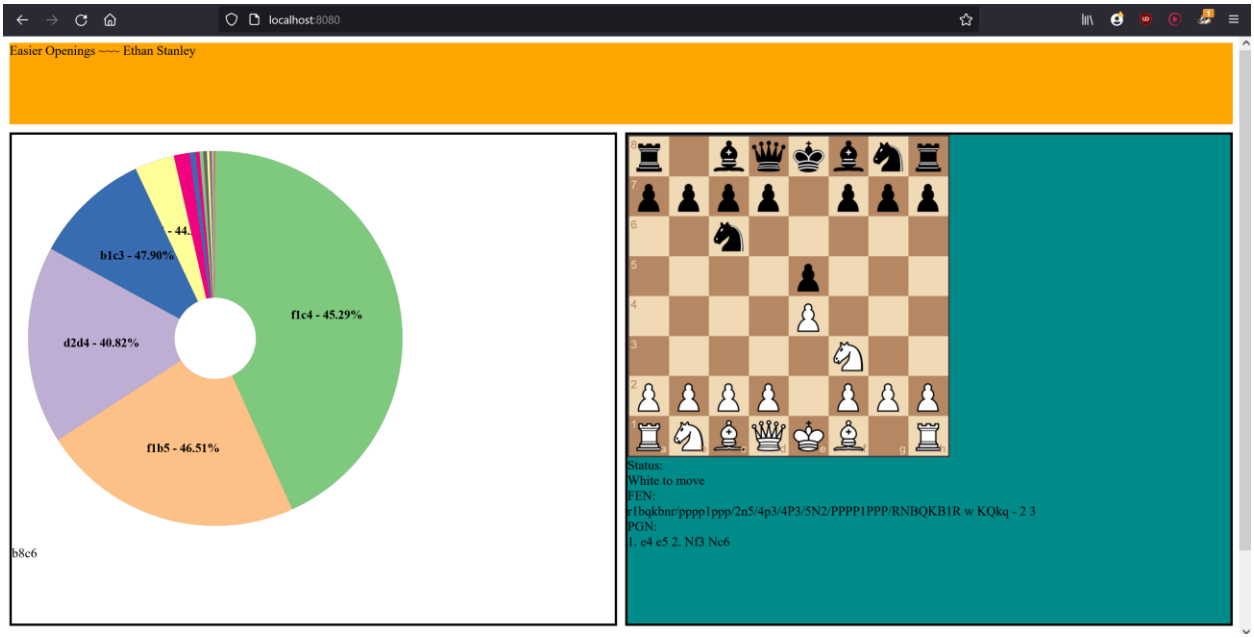
User can click on a wedge on it becomes the outer ring, or make a move on the chess board.





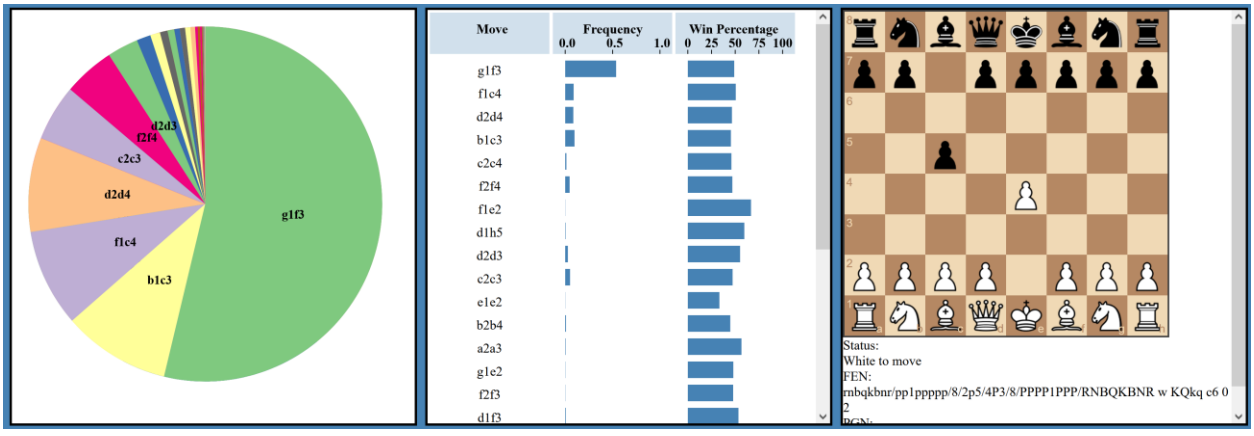
Changes in design while implementing:

I followed the design plan of the finalization until I reached this point:



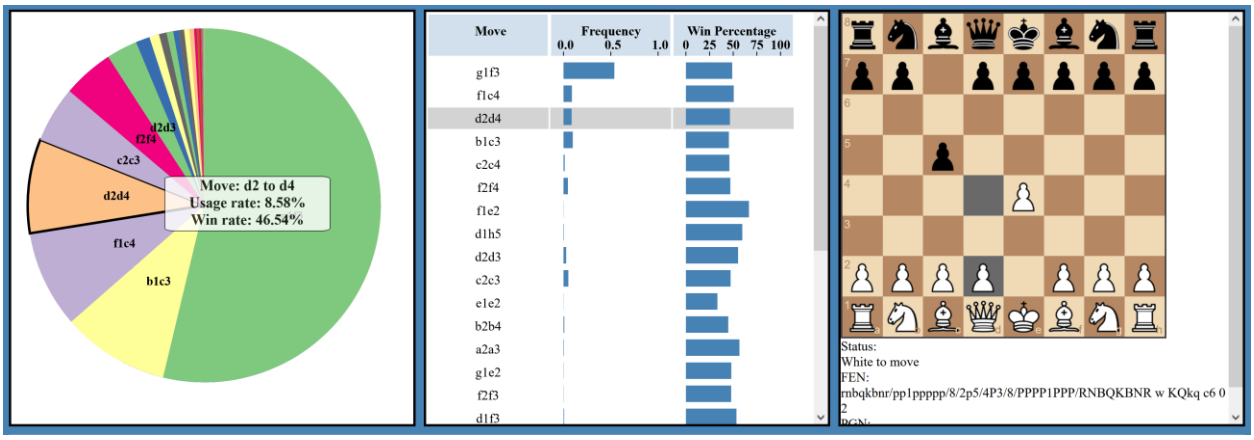
At this time, I had my project review meeting and was suggested that adding a table would improve the visualization. This turned out to be a crucial improvement. The table would allow sorting based on the attributes of the moves and the use of more powerful visual variables (bar charts inside of the table would use position, which is a more powerful visual encoder than angle).

The next step was to create the table and have it display the data properly. This is what it looked like:



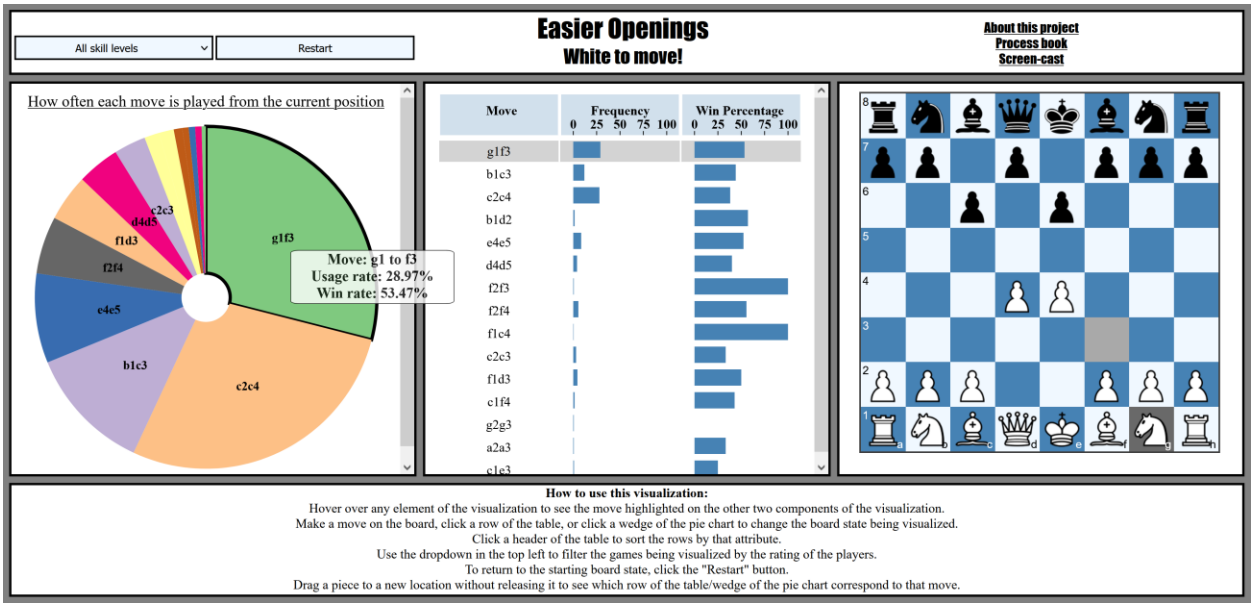
I used bar charts inside of the table because it is easier to compare bar lengths than many numbers. The exact percentages are also provided by the pie chart tooltip.

From there, I connected all the components of the visualization so any of them can be used to select the next move. Additionally, you can hover over any element and all components will have appropriate highlighting. Here is what the highlighting looks like:



Another deviation I made from the planning phase is the controls for the filtering. In my sketches, I had a brush that would allow the user to select a precise skill level range of both players. Unfortunately, this is not aligned with the reality of the data processing I had to do. To get a meaningfully large tree, I had to leave my (albeit underpowered) computer running for several hours. Obviously, this could not be done on demand. Thus, I would have to process the data into a pruned tree ahead of time. Another road block was that data processing takes too long with overly precise filtering because it takes along time to ignore games. Thus, I could only have a few major groups (beginner games, intermediate games, expert games, and unfiltered games). I decided that a dropdown menu makes more sense for this task than a brush.

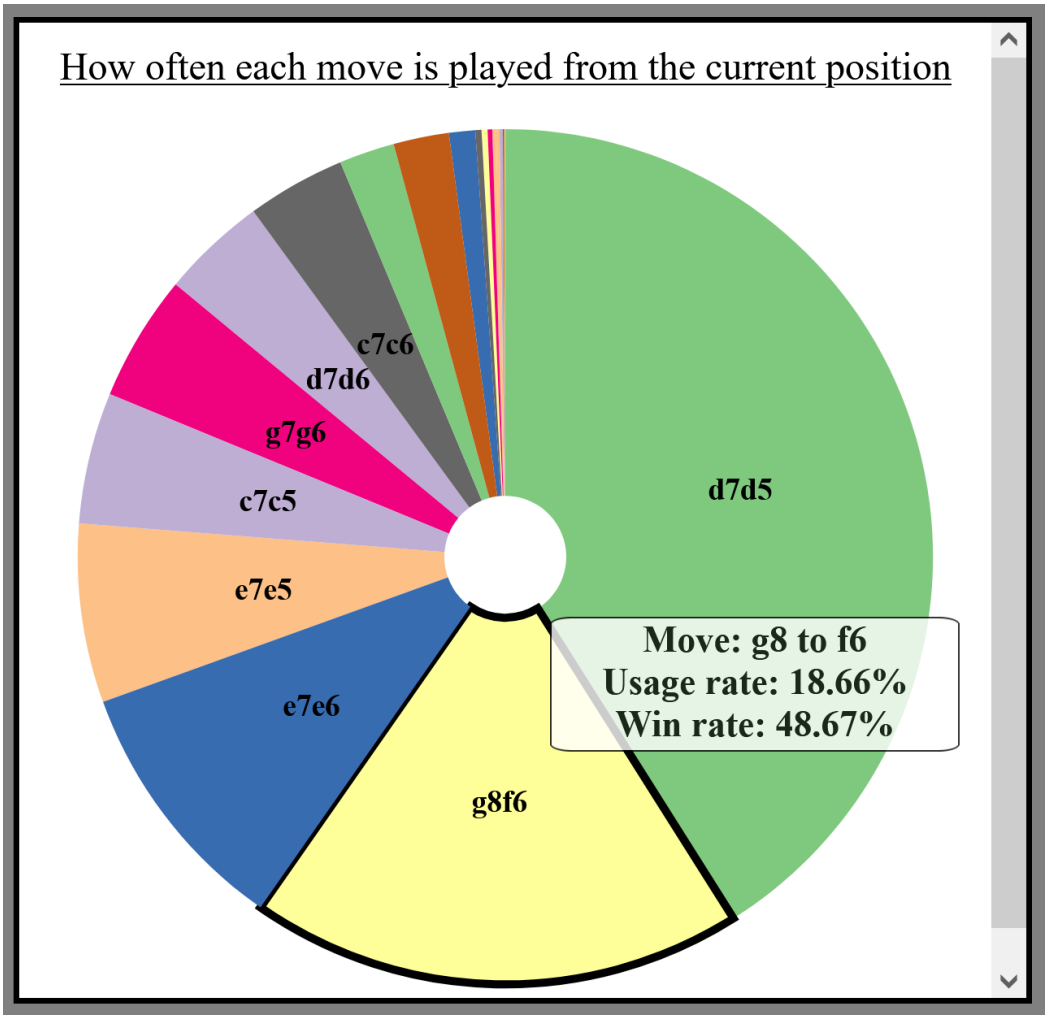
All that was left to do was implement the filtering and improve the GUI. Here is the final state of the visualization:



Implementation:

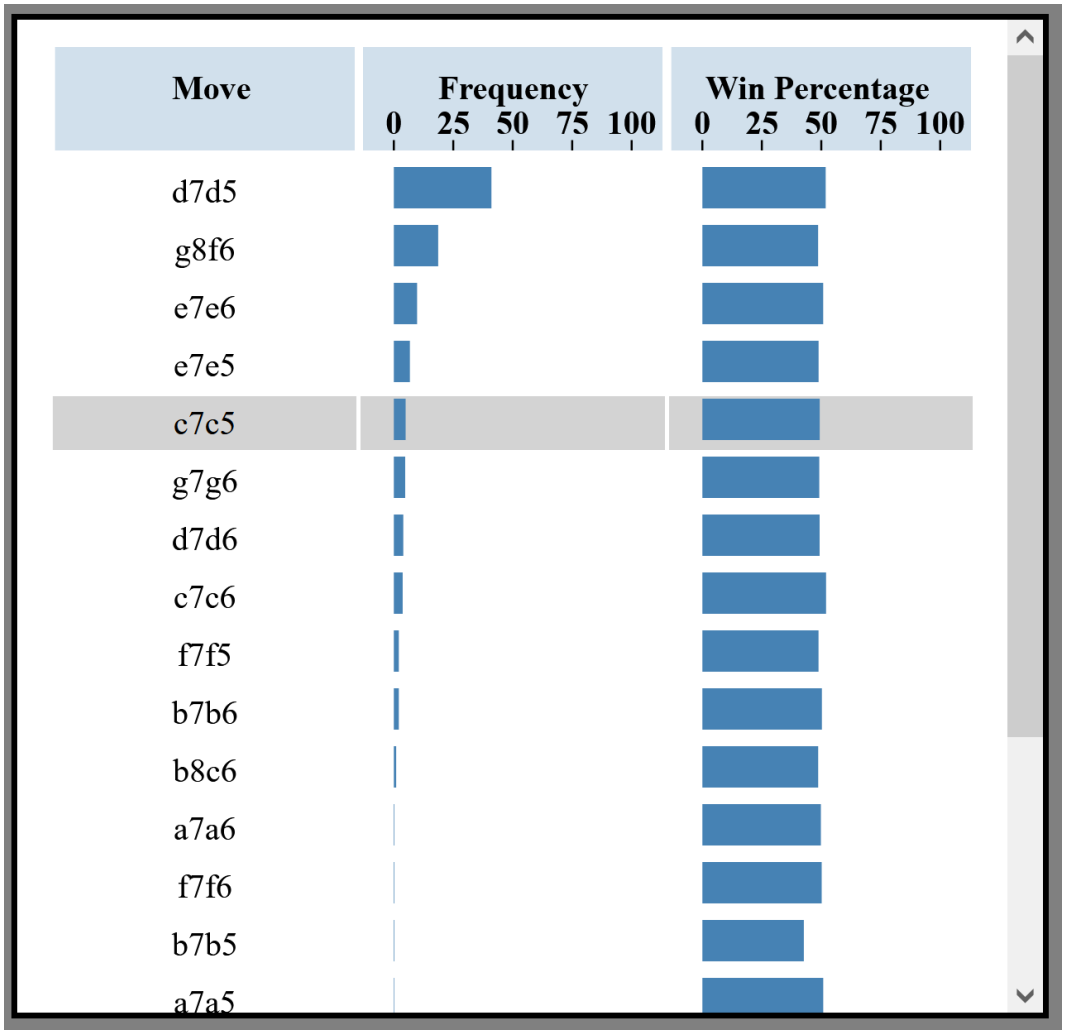
My project contains three interactive linked visualizations: a dynamic pie chart, a table, and a chess board. I will go into each component in detail:

Pie chart:



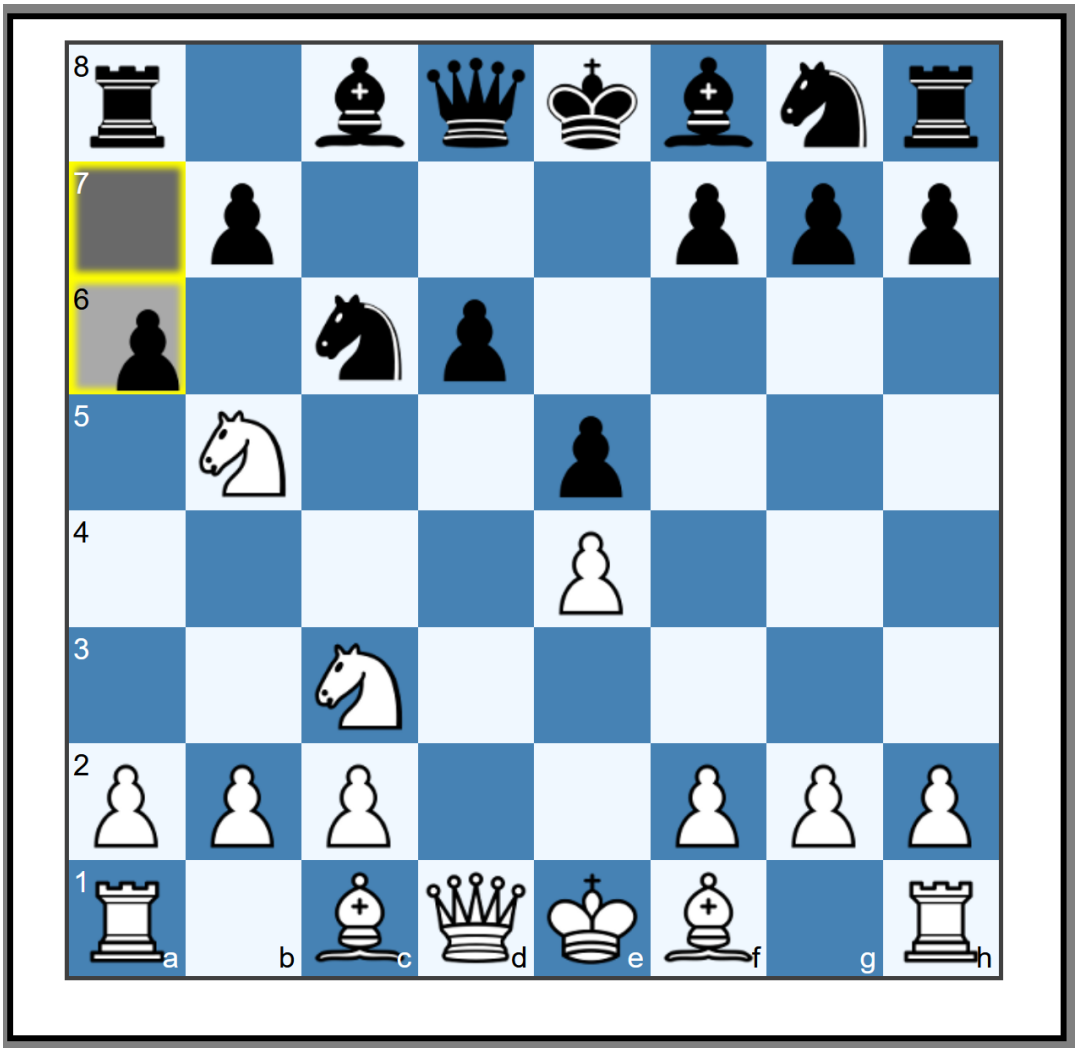
This pie chart shows every move that has been played from the current board state in the database. The size of the wedge corresponds to the usage rate of the move (what percentage of the time this move was played from this position). Hovering over a wedge emphasizes the border of the wedge and brings up a tooltip providing additional numerical information about the move (in the image, g8f6 is being hovered). The tooltip lists the move (useful for wedges that are too small to have a label), the usage rate (what is encoded by the wedge size), and the win rate of the move (how often a player that made this move ultimately won the game). The colors of the wedges do not have semantic meaning – they just distinguish adjacent wedges. Clicking on a wedge updates all three components of the visualization to represent the new board state (a move is played on the board). The pie chart is useful to quickly see which moves are the most dominant.

Table:



The second visualization is the table shown above. It is an additional encoding of the data shown in the pie chart, except it shows the win percentage explicitly rather than interactively like the pie chart. The rows can be sorted by any attribute by clicking the headers. This makes the table a much more powerful tool to find the moves with high or low win percentages, and uses a more powerful channel (position) so the attributes of the moves can be compared more effectively.

Chess board:

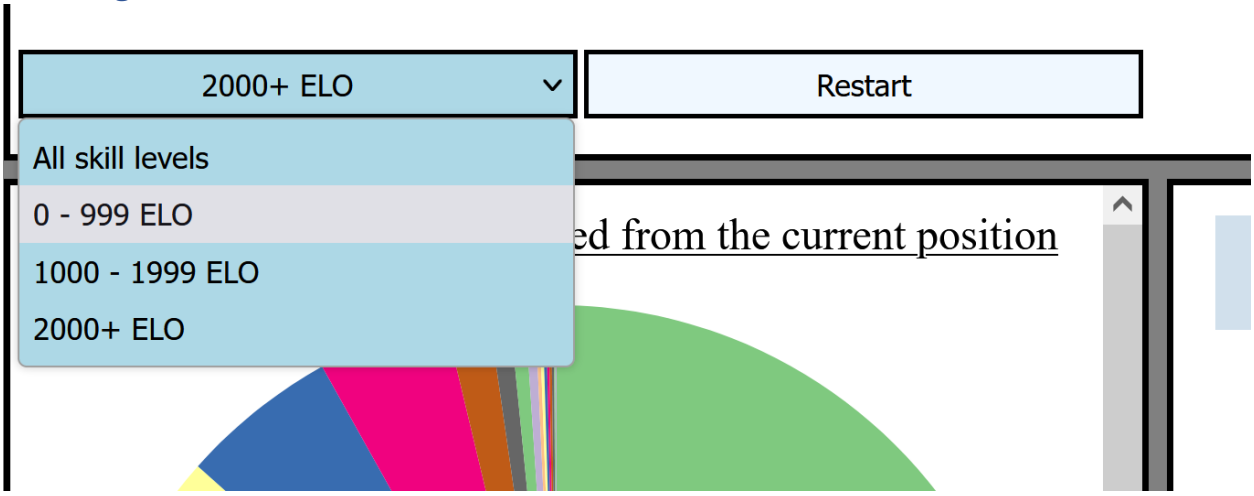


The final component of the visualization is the board. This displays the current state of the chess game and allows the user to play moves and update the other visualizations. The board does not encode data but it is necessary to make the visualization easy to use and to give context to the moves being explored on the pie chart and table.

Interactivity:

All three of the visual components are synchronized. Hovering over a wedge of the pie chart, a row of the table, or a move on the chess board will highlight that move on the other components. This combines the strengths of each individual component into a more powerful tool (e.g. instead of previewing moves one by one to see which one has the highest win rate, one can use the table to find the move with the highest win rate, hover over it, and see it highlighted on the board).

Filtering



A final element of interactivity is the ability to filter which games are used to generate the data in the visualization based on the rating of the players. The intent of this is to allow a user to compare the effectiveness and popularity of an opening at low or high skill levels.

Evaluation:

While there are many questions that could be addressed by this visualization, the one I had in mind was: “Are there openings that are theoretically strong (effective with high level play) but don’t work well for beginner or intermediate players?”. I found examples of these openings by

filtering for high level games, using the sort function of the table to find openings with a high win rate, and then comparing this to the win rate when filtering for beginner/intermediate games. An example is the King's gambit. Advanced players have a 48% win rate with the King's gambit opening whereas intermediate players have a 43% win rate with it. This difference is especially great considering that advanced players have draws much more frequently. This suggests that the King's gambit, for whatever reason, is a difficult opening to play effectively and thus it is not the best to learn as a beginner or intermediate chess player.

Another thing I learned from interacting with the visualization is that the choice of opening is far less consequential in beginner games than advanced games. Almost all opening moves in beginner games have a win rate between 45% and 55%. In contrast, some openings have a win rate as low as 30% among advanced players. An example is c3 (or the Saragossa opening). It has a win rate of 36% in advanced games and 49% in beginner games. A takeaway from this is that it is perfectly fine to not bother studying openings and experiment if you are a beginner chess player.

Another question I answered is: "what openings are effective across a range of skill levels?". The King's knight opening is popular and has roughly a 50% win rate across all skill levels. One can use the visualization to find many other popular and effective openings so they know what to study (they are very likely to face these openings no matter who they are playing).

I think my visualization is effective in answering these questions. The table (which was added at the suggestion of a TA), makes it much easier to find moves of interest (high/low win rates, high/low usage rates, moves with a notable combination of these attributes). The pie chart is good for making quick judgements about the distribution of possible moves in a given position. For example, if the pie chart is dominated by a single wedge, then this move is "obvious" because many people play it. This has implications for how easy an opening is to play because obvious moves reduce the likelihood of mistakes.

The visualization works well. It runs smoothly and has no bugs that I am aware of.

If I were to further improve the visualization, I would add information about how often games result in draws, and I would add functionality to undo moves rather than returning to the initial board state and replaying the game.