University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

stilltheory.org: a platform to practice chess openings with puzzles

Renan Krzesinski
Ygor Sad Machado

Final Essay

mac 499 — Capstone Project

Supervisor: Prof. Dr. Alfredo Goldman
Co-supervisor: Prof. Dr. Johanne Cohen

2022

Abstract

Renan Krzesinski, Ygor Sad Machado. **stilltheory.org: a platform to practice chess openings with puzzles**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

Solving puzzles is a very common and effective way to practice skills like tactics in chess. In recent years, puzzles are played mainly through internet services, but these services fail to offer puzzles that help players practice the opening phase of the game. In this work, new puzzle structures were developed specifically aimed at practicing the most required skills in the opening. To implement these puzzles, a web system was developed. For the data layer of the system, a graph-oriented database was built using free APIs from the Lichess platform. The web application consists of a backend capable of building and serving puzzles and a frontend that requests and consumes these puzzles, offering them online to users. The final application is available at https://stilltheory.org.

Keywords: Chess openings. Chess puzzles. Single-page application. Graph-oriented database.

List of abbreviations

API	Application Programming Interface
AQL	ArangoDB Query Language
AWS	Amazon Web Services
BFT	Breadth-first Traversal
CD	Continuous Delivery
CI	Continuous Integration
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DFT	Depth-first Traversal
EC2	Elastic Compute Cloud
ECO	Encyclopaedia of Chess Openings
EPD	Extended Position Description
FEN	Forsyth-Edwards Notation
FOSS	Free and Open Source Software
GPL	GNU General Public License
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JSONL	JSON Lines
RAM	Random Access Memory
SAN	Standard Algebraic Notation
SPA	Single-page Application

List of Figures

3.1 Excerpt from a chess game depicted as a graph. 6

5.1 Comparison of puzzles distribution by phase of the game in Lichess. . . . 16

6.1 Flowchart depicting how a Main Moves puzzle is played. 20
6.2 Flowchart depicting how a Survival puzzle is played. 21
6.3 Flowchart depicting how a Main Line puzzle is played. 21
6.4 Flowchart depicting how an All Theory puzzle is played. 22

8.1 Representation of the graph model, including attributes and their types. . 27

A.1 Screenshot showing a puzzle in the middle of the gameplay. 43
A.2 Screenshot of the search mechanism, used to choose the position users want to play with. 44
A.3 Screenshot of a puzzle of type Main Moves when the user loses it. 44
A.4 Screenshot of a puzzle of type Main Moves when the user wins it. 45
A.5 Screenshot of a puzzle of type Survival when the user loses it. 45
A.6 Screenshot of a puzzle of type Survival when the user wins it. 46
A.7 Screenshot depicting the modal containing instructions on how to play. . 46

List of Tables

8.1 List of attributes contained in a position. 26
8.2 List of attributes contained in a move. 27
8.3 Comparative of most popular graph-orientend databases. 29

9.1 List of endpoints available on puzzles-api. 35

List of Programs

1 Source code of a simplified build_graph function. 32
2 Source code of a simplified expand_graph function. 33

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Objective . . . 2
1.3 What was accomplished . . . 2
1.4 How the text is organized . . . 2

2 Methods **3**

3 Chess **5**
3.1 Phases of the game . . . 5
3.2 Mathematical model . . . 6
3.3 Complexity of the game . . . 7

4 The opening **9**
4.1 Complexity of the opening . . . 9
4.2 Breaking new ground . . . 9
4.2.1 Opening principles . . . 10
4.2.2 Calculation . . . 10
4.3 Engines . . . 11
4.4 Publications . . . 12
4.5 Opening Theory . . . 12

5 Online chess **15**
5.1 Opening explorers . . . 15
5.2 Puzzles . . . 16
5.3 Existing opening puzzles systems . . . 17
5.3.1 Lichess's opening explorer . . . 17
5.3.2 ChessOpenings . . . 18

6 Puzzle design **19**
6.1 Main Moves . . . 19
6.2 Survival . . . 20
6.3 Main Line . . . 21
6.4 All Theory . . . 22

7 Project decisions **23**
7.1 Free and Open Source Software . . . 23
7.2 Programming language . . . 23
7.3 Code quality . . . 23

8 Data **25**
8.1 Data modeling . . . 25
8.1.1 Nodes . . . 25
8.1.2 Edges . . . 26
8.1.3 Model . . . 27
8.2 Data acquisition . . . 28
8.3 Database management . . . 28

9 System modules **31**
9.1 db-tools . . . 31
9.2 graph-builder . . . 32
9.3 db-data . . . 34
9.4 puzzles-api . . . 34
9.5 puzzles-front . . . 35

10 Deployment **37**

11 Conclusion **39**
11.1 Personal perspectives . . . 39
11.1.1 krz . . . 39
11.1.2 sad . . . 40
11.2 Future work . . . 40

Appendixes

A Images of the user interface **43**

References 47

Chapter 1

Introduction

1.1 Background

Chess is one of the most played board games in the world, entertaining players for centuries and serving as a theme for books, movies and television series. Although in its common form it is intended to be played by two people, chess is fundamentally about solving problems using logic. So naturally many other ways to enjoy it have emerged.

One such alternative way of enjoying chess is solving puzzles, which consist of some kind of chess-related logic challenge. Because they are small excerpts, the puzzles are quick and simple when compared to complete games, and can work as an excellent introductory tool to learning chess.

The difficulty of a puzzle, which is what makes it enjoyable, can come from exercising any of the skills desirable for a good player. Today there are several online services that offer puzzles to players, but the puzzles offered tend to always exercise the same skills, mainly the ability to calculate and recognize tactical patterns on the board, and in some cases the speed of reasoning.

However, during the first moves of a chess game, the calculation and recognition of tactical patterns exert very little influence on the moves made. In this phase, known as opening, other skills are much more relevant, such as memorizing and applying principles.

So, in practice, the puzzles available on the big online chess services are not able to cover the initial phase of the game. As a result, many beginner chess players who use puzzles as a means of practicing end up developing their skills unevenly. They can reasonably navigate situations in the middle of the game, but get frustrated that they often cannot make it through the opening phase without losing immediately. And it's not just beginner players who are left helpless. More experienced players also play puzzles to stay sharp and could use them to practice their opening repertoires.

1.2 Objective

The objective of this work is to take better advantage of the great potential that puzzles have to help in learning chess, extending them to cover the opening phase as well. This objective can be divided into three main elements: first, to design puzzle structures that take into account the peculiarities of the initial phase of the game of chess. Then build an application capable of creating instances of these puzzles programmatically. And finally make this application available online for free and in the form of free and open source software.

1.3 What was accomplished

Taking into account the most demanded skills in the opening phase, 4 new types of puzzle were developed. These puzzles differ from traditional puzzles in several structural aspects, such as the flow of play or the number of correct answers.

To be able to serve these puzzles, the application needed to be grounded in a suitable chess database. Then a database was modeled according to the needs and an algorithm was developed to build this database from free data sources.

A complete system with server and client was developed to consume the data, algorithmically generate puzzles and make them available to users. The system is capable of serving two of the four types of puzzle developed.

When interacting with the system, the user can configure specific openings to practice, which color to play (black or white), difficulty level and the type of puzzle.

The application has been published online and is available at https://stilltheory.org. Furthermore, the generated data that feeds the application has also been made public and is available at https://gitlab.com/krz-sad-tcc/db-data.

1.4 How the text is organized

This text is organized in 11 chapters, linearly presenting the necessary knowledge to better understand the subsequent sections. Chapter 2 presents an overview of the tasks ahead and the tools available to face them. Chapter 3 is about chess in general and the mathematical aspects of the game. Chapter 4 is specifically about the opening and seeks to show what tools players have to go through this phase of the game, culminating in a definition for Opening Theory. Chapter 5 presents existing online chess applications and what they offer in relation to puzzles in general and Opening Theory. Chapter 6 contains proposals for opening puzzles. Chapters from 7 to 10 deal with the development of the application itself, with 7 presenting the general decisions, 8 covering everything that has to do with data, 9 contemplating in detail the constituent parts of the system, and 10 showing how the deployment was done. Finally, Chapter 11 presents ideas for continuing and expanding the project and gives the developers' personal perspectives on the work done.

Chapter 2

Methods

The chess game can be represented as a graph, and in the opening phase this modeling is particularly useful, as there is a close connection between the possible ways of reasoning about the opening and the ways of exploring a graph. For instance, the question "what are the possible moves in a position" is related to breadth-first traversal (BFT), whereas the question "will this line end in an advantage for either player" is related to depth-first traversal (DFT). Therefore, the first step is to carry out this modeling, which will serve as a basis both for proposing new types of puzzles and for building the system.

The next step is to build a database of chess openings, which should be a graph-oriented database. For this, it is necessary to select data sources, choose how to obtain these data and write the necessary algorithms to build the graph. It is desirable that the development of the application be accessible to anyone who wants to contribute, so such data must be free and easy to handle on personal machines.

For the application development part, it is necessary to make choices regarding architecture, programming languages, frameworks, auxiliary tools and ways to guarantee software quality. With such choices made, the construction of the actual constituent parts of the system begins.

Deployment-related tasks involve purchasing a domain, hiring a hosting service and carrying out the necessary infrastructure operations to make the application available online.

Chapter 3

Chess

Chess has had a long history of evolution, having certainly originated in the East and spread to the West via trade routes such as the Silk Road, and also through cultural exchanges that occurred during the occupation of the Iberian Peninsula by the Islamic Empire in the eighth century. Its closest ancestor is a game called Chatrang, played in Persia around the 5th or 6th century, which in turn derived from an Indian game called Chaturanga (Shenk, 2014). Since then its popularity has grown to become a universally known game, with approximately 605 million adults playing chess regularly (United Nations, 2022).

3.1 Phases of the game

It is usual to divide the game of chess into three phases: opening, middlegame and endgame. Even though there are no clear limits between these phases, they are still used as general guides to facilitate the study of the game, taking advantage of the fact that each of them presents certain peculiar aspects in relation to the others.

At the beginning of the game, all pieces are in their starting positions, away from the opponent's pieces. This phase is known as the opening. The goal is to bring the pieces into play, coordinating them to better face the direct confrontations that are about to start.. In the next phase, known as the middlegame, the pieces are already disputing space directly with the opposing pieces, and tactics begin to play a much more relevant role. When it's the case that a point is reached where most of the pieces have been exchanged, with only pawns and maybe a few pieces remaining, it is customary to say it's an endgame. This phase requires knowledge of specific techniques, and usually the objective is to promote pawns in order to checkmate the opponent.

Differences between the game's phases give rise to different opinions as to which phase is more difficult and which one should be learned first by the novice player. Cuban Grandmaster José Raúl Capablanca, one of the most important in history, said that, because all 32 pieces are involved, the opening is the most difficult phase, and therefore players should start studying chess by the endgames (Capablanca, 1935). On the other hand, the American National Master Bruce Pandolfini, one of the most important chess teachers

today, argues that one should start studying chess by the opening, since all games have an opening and the player will always go through it, but not all games have endgames (Pandolfini, 1989).

3.2 Mathematical model

To have more tools at hand, it is useful to establish a mathematical model of the game of chess as a graph search problem. In this model, the nodes represent game states, each one corresponding to a position together with some situational aspects of the game, such as castling rights and en passant target. The edges correspond to moves that can be performed by players, leading from one state to another.

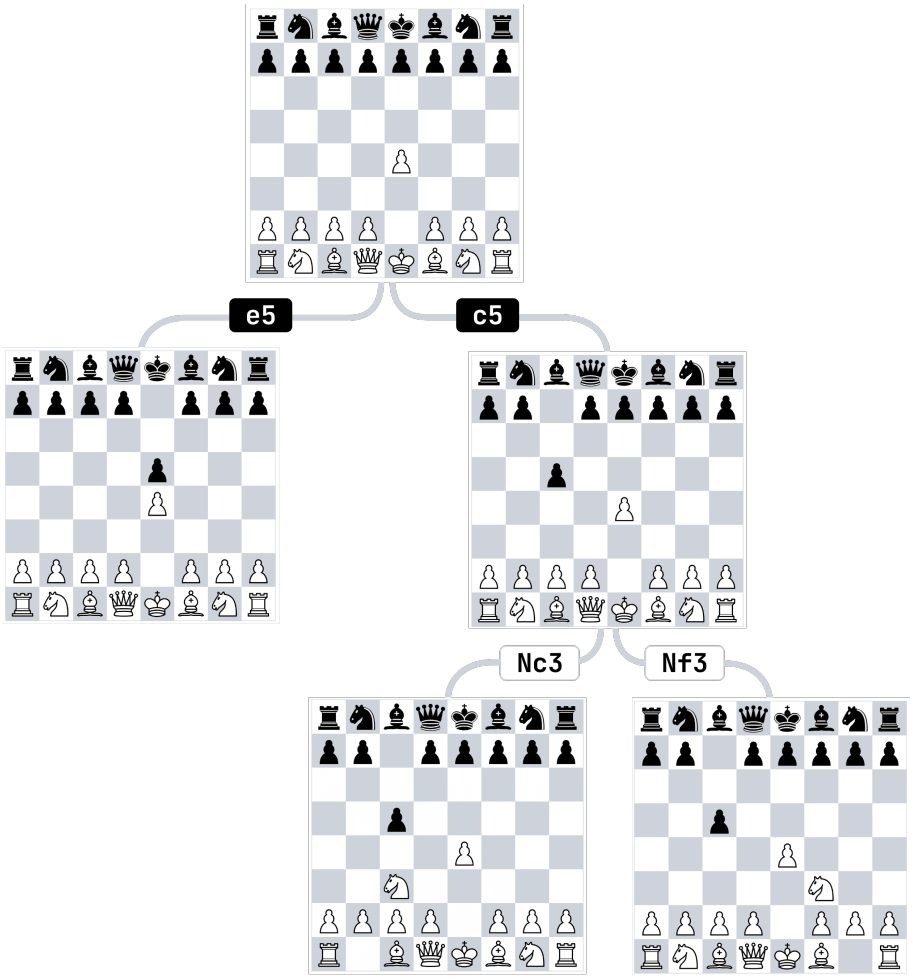

Figure 3.1: *Excerpt from a chess game depicted as a graph.* [1]

The initial state of the search problem is the initial state of the game, and the terminal states are situations where one of the players is checkmated or when the game is technically

[1] This work uses Standard Algebraic Notation (SAN) to display chess moves. More information at the Chess Programming Wiki.

a draw, for example when stalemate or a draw by the fifty-move rule occurs. Of course, a real game can end before a terminal state is reached, via a draw agreement between the players or the resignation of one of them.

3.3 Complexity of the game

The only part of the game of chess that has been solved to date is the ending. Great computational efforts were spent in the elaboration of exact solutions for positions that have a limited number of pieces in play. To arrive at such solutions, the search starts from terminal nodes towards intermediate nodes, exhaustively exploring the search graph backwards. The databases that compile such solutions are known as tablebases. Until 2018, there was a complete solution for all possible positions with up to 7 pieces (that is, one king for each side plus any 5 pieces). The most important tablebase is Syzygy. This is a free tablebase, and its total size is over 16 TB (Fiekas, 2022).

Tablebases are being developed for positions with up to 8 pieces. As the game of chess has 32 pieces at the beginning, possibly there will never be a complete solution for it. And precisely because of this, chess will probably remain an interesting game for a long time, both for humans and computers.

In 1950 Shannon (1950) proposed a simple and interesting calculation to capture the complexity of the game of chess. His method consists in approximately counting the number of variations, i.e. of different possible paths in the search graph, until the end of real games. This means that in addition to counting paths that end in terminal states of the search graph, paths ending in resignation and agreed draw are also counted, which actually are the vast majority of games in practice.

The calculation goes like this: chess is a game with a branching factor of approximately 30, which means that in each position it is possible to perform about 30 different moves. A fullmove, that is, one halfmove for each side, then generates about 10^3 paths. Then, taking into account that a typical game lasts about 40 fullmoves, it is possible to count the number of variations as $(10^3)^{40} = 10^{120}$.

This approximation is not intended to be precise, with the sole aim of making it clear that solving the chess game is an intractable problem.

Chapter 4

The opening

4.1 Complexity of the opening

By fixing the length of the games at 40 moves, the strategy used by Shannon seeks to be realistic in the task of approximating the number of possible variations in the entire game. It is possible to modify this calculation in order to get an approximation of the number of different variations in the opening. For that, two numbers need to be adapted: the depth of the search graph that is taken into account and the ramification factor.

Regarding the depth of the graph, it is necessary to establish how many moves long the opening phase is. According to Pandolfini (1989), openings can last between 10 and 15 moves, sometimes even more. For the purposes of this work, the most conservative number can be considered so that we have a tight upper bound for the complexity. This sets the length of the opening to 10 fullmoves.

When considering that every time a player needs to make a move there is an average of 30 moves available to choose from, we are taking into account all the legal moves, but this is clearly an exaggeration, as it includes bad moves and even blunders that would never be considered by a player. This number can be restricted to consider only "playable" moves. A reasonable choice is to consider that there are on average 3 relevant moves available, or about 10 moves per fullmove (Chernev and Harkness, 1945).

In this way, the total number of opening variations is about 10^{10}. Once again, this is just an approximation, and even though the concessions made were extended a little further, making the complexity analysis more distant from the mathematically precise structure of the game graph, it gives rise to a useful final number for better understanding the problem faced by a player in the opening.

4.2 Breaking new ground

Every time a game of chess is played, there is potential for knowledge to be accumulated about the positions that have emerged, either by the players who were in the game or by others who eventually come to study the game.

The opening has this cumulative characteristic in a much more pronounced way than the other phases, since, as the game always starts from the same position, the positions that appear in the opening are more likely to be repeated.

But even having accumulated knowledge about a number of positions, as a game progresses, the probability of reaching a position never seen before increases, and at some point, sooner or later, the player will be on her own. This is where the heavy use of tools such as opening principles and calculation begins.

4.2.1 Opening principles

When in doubt about what is the best move in an opening position, a good way to look at the situation is through the lens of opening principles.

Throughout history, players have realized that there are positional and geometric attributes that are desirable in most openings. They are not infallible rules, but general guidelines that, when followed, bring favorable elements to the game and avoid known pitfalls. These general rules are known as the Opening Principles. According to Emms (2006), the main opening principles are:

- Control of the center
- Rapid piece development
- King safety

Other commonly listed principles can be easily derived from these. For example, the famous principle "don't move the same piece twice" can be seen as a consequence of having rapid piece development. The opening principles eventually don't apply, but when coupled with well-done analysis and calculation, they begin to compose a powerful arsenal to get through the opening successfully.

4.2.2 Calculation

The next tool available to deal with the situations where memory doesn't help is calculation, which is used more heavily the further the players get into the game. Calculation is the act of mentally projecting the next steps from the current situation on the board. By imagining possible future board configurations, it's possible to evaluate them as good or bad, and then evaluate as good or bad the moves that lead there.

At each projected future step, a number of moves and opponents' responses are considered, configuring what can be seen as a subgraph of the game's search graph.

As a player gains experience, more skill is acquired to apply calculation systematically, but essentially, to apply it, it is enough to know the rules of chess.

4.3 Engines

Today there is one more element to help players learn and make decisions in openings: engines. Simply put, a chess engine is a computer program capable of playing chess (Chess Programming Wiki, 2022).

Engines have a history closely related to the history of computers themselves. In the 1830s, Charles Babbage already imagined a machine capable of playing chess. Later, the problem of building an engine capable of winning a match against the world champion became popular. That finally happened in 1997, when IBM's Deep Blue computer defeated the reigning world champion Garry Kasparov. This can be considered a turning point in the history of computer chess, from which it is possible to argue that computers became better than humans at chess.

Being able to make decisions, one of the functionalities of the engines is to evaluate positions and moves, so their use has become practically mandatory in the study of openings for serious chess players. This fact can be evidenced by the following quote from Grandmaster Judit Polgár's work "A Game of Queens":

> "Somewhere around 2004 to 2005 I experienced a general crisis in my opening preparation. I had a tough battle with myself to accept the fact that, whether I liked it or not, I had to use the engines on a daily basis, double-checking my old analysis as well as all my new ideas. I can say that the whole concept of preparation changed due to the intrusion of technology into chess!" (Polgár, 2014).

Engines evaluate positions more reliably than humans, but for several factors, when it comes to opening preparation, it is not advisable to take an engine by itself as the source of the best play. Primarily because of the lack of explainability. Especially in the case of engines based on neural networks, the best move indicated by an engine hardly adds any knowledge to the player, serving only as information to be memorized, and that cannot be applied in any other situation, rather than a real understanding of the position.

Another problem with using engines as a source of truth is that humans and engines have different approaches to exploring the game graph. It is possible that even if an engine points out that in a given position neither white nor black are better, in practice it may be much more difficult to play one side than the other (Gelfand, 2020).

An example of such a situation is the following: an engine points out that there is perfect balance in position up to 5 moves ahead. For the white pieces, in each round ahead it is possible to make three or four "correct" moves, that is, moves that maintain the balance of the game. However, for the black pieces there are only non-trivial moves ahead, and in each round only one can be considered correct. This situation represents an easy game for white and a difficult one for black. For engines it is easy to explore a game graph under these conditions and safely state that there is a line that maintains the balance from the point of view of the black pieces, and hence it says the evaluation is equal. But for humans, the poor visibility characteristic of narrow path situations like this makes the game quite uncomfortable, and therefore unbalanced. Thus, it is safer to use engines to corroborate or reject previously made analyses, rather than to generate new ones.

4.4 Publications

Openings are a very common theme in chess literature in general, but their relevance is even more highlighted when we compare openings with the other phases of the game, a scenario in which this class of books is completely dominant (Capablanca, 1966).

To facilitate communication, throughout the history of chess some distinctive positions have been given names. For example, when a game is said to be a "Grünfeld Defense", everyone familiar with chess has an idea of what kind of game took place. Sometimes openings are named after a player who has studied them deeply, as in the case of the Philidor Defense (1. e4 e5 2. Nf3 d6), named after François-André Danican Philidor. Other times they are named after a place, as in the case of the Catalan Opening (1. d4 Nf6 2. c4 e6 3. g3).

Despite making conversations between humans much easier, these names are subject to problems. For instance, the existence of more than one name for the same opening (Petrov's Defense and Russian Defense) and some locale particularities that imply in spelling the names differently (such as Defense and Defence). These issues make it more difficult to use names as an accurate means of identifying openings. So a unified classification system for openings is of great value.

The main historical reference for classification of openings is the set of books called "The Encyclopaedia of Chess Openings" (ECO). It is a collection of 5 volumes classifying and describing the current knowledge about main lines and variants of the most diverse openings. It was first published in the 1980s and is currently in its fifth edition. The openings receive a code according to the volume in which they are located. For example, the opening "Queen's Gambit Declined; Chigorin Defence" can be found in volume "D – Closed Games and Semi-Closed Games". Its code is D07.

4.5 Opening Theory

Despite being a term widely used by people in chess, there is no precise definition for Opening Theory, with several authors giving different ideas about what it means.

One aspect in which there is consensus is that it is about previously acquired knowledge about openings. It may comprise publications, computational analysis and, most importantly, what good players actually play in their games.

As the present work deals with the construction of a computational system focused on the use of Opening Theory to assess moves, it is important that the definition of such a concept be done in a solid and systematic way. Next, we try to establish such a definition, justifying that the aspect that offers the best foundations is "what good players actually play".

The most interesting point of using a history of games to define Opening Theory is that players tend to use all the tools we described earlier: opening principles, calculation, evaluations given by engines, publications and even Opening Theory itself. Thus, Opening Theory would indirectly encompass and condense all these items, forming a cumulative knowledge base that tends to improve as it grows.

In order to use a history of games as a basis for a definition of Opening Theory, it is necessary to start from the premise that whoever plays a game of chess seeks to win and, therefore, strives to find the best moves. Of course, this is not always true. It is relatively common to see players who strive to only get a draw on well-known lines that lead to this result, or even situations in which a player makes a bad move on purpose just to confuse the opponent. We assume that, in the long run, such cases tend to be of little relevance.

From a technical point of view, this approach is also quite convenient, as it is related to the construction of databases, having the enormous advantage of being able to be carried out by computers.

As in a data science effort, a large volume of games is needed for it to make sense to extract information about the quality of moves, or to establish causality between moves made in the opening and the outcomes of the respective games.

With all this established, two fundamental questions arise:

- Which games are considered Theory?
- To what depth in the game graph is a path considered Theory?

As for the first question, if we consider that games played by any player can compose the Theory, it would also include poorly played games, which is clearly not useful. By intending a didactic function for Opening Theory – which is actually how it is used in practice – it seems natural to consider that Theory comprises games played by good players.

Now for the second question, if we establish that paths of any length can be considered Theory, at the limit, a game of chess is considered theory from beginning to end, which is useless. One approach is to consider that Opening Theory encompasses positions that appear a sufficient number of times in the records.

Therefore the matter of defining Opening Theory is narrowed down to determine what constitutes a good player, how many games is a sufficient number of entries and what are the records we are talking about, which can be seen as establishing the following:

- A database of chess games
- A minimum rating of players to be considered
- A minimum number of games a position must appear in to be considered

These choices will always contain a high level of arbitrariness, but give rise to a systematic definition of Opening Theory, in which three statements are enough to completely characterize it.

Chapter 5

Online chess

With the popularization of the internet, websites soon began to appear allowing players to play online against each other. In the year 2005, the websites "pogo" [1] and "Free Internet Chess Server" [2] were the most popular ones (Eade, 2005), and both are still online today, albeit with much lesser popularity. Nowadays the most popular website to play chess online is Chess.com [3], which in August 2022 was the 203rd website with the most hits in the world, followed by Lichess [4], which occupied position 265 in the same ranking (Similarweb, 2022).

Lichess was founded in 2010 by the French programmer Thibault Duplessis, and is of greater interest for this work because it is free software, with the source code available at https://github.com/lichess-org/lila (Lichess, 2022a). According to data provided by the founder, Lichess had over 11 million members in 2021 (Duplessis, 2021).

In addition to the possibility of playing against the computer or other players, chess websites provide several other features such as opening explorers and puzzles, which are of great interest to people looking to both have fun and practice specific aspects of their game.

5.1 Opening explorers

Opening explorers are tools built on top of game databases. They allow the user to navigate forwards and backwards in the game graph by making moves on a chess board. In the Lichess's Opening Explorer [5], for each position information is given about the number of times it has been reached, the number of times each move has been played, what is the distribution of the final results – i.e. white wins, draws and black wins –, among others.

[1] https://www.pogo.com

[2] https://www.freechess.org/

[3] https://www.chess.com/

[4] https://lichess.org/

[5] https://lichess.org/analysis#explorer

The databases used in Lichess are free and available at https://database.lichess.org/. There is a database of master games and a database of games played by users in Lichess itself. In addition to accessing databases via download, it is possible to obtain information about specific positions via an Application Programming Interface (API).

5.2 Puzzles

The vast majority of chess puzzles available are those known as "tactical puzzles". In chess, the term tactic refers to a move or sequence of moves that can guarantee a player some kind of gain without the other being able to avoid it, even with perfect play. For example, a "double attack" is a common type of tactic in which a player has the opportunity to carry out two threats at the same time, without the opponent having the possibility to defend both and therefore necessarily incurring some kind of loss.

Tactical puzzles can be built on top of positions that occurred in real games of chess, but there are also those where the situation on the board was manually designed especially for the purpose of creating a puzzle. In real games, tactics often appear in positions where a great opportunity arises from a blunder committed by the opponent in the previous move.

In this class of puzzles, either there is only one correct move or there is a single sequence of correct moves. Due to this structural characteristic, the opening is not a suitable phase to generate base positions for these puzzles, since in the opening the most common scenario is to have several moves or several sequences of moves that can be considered correct in each position.

In Lichess all tactics puzzles are built based on positions that occurred in games played by users of the platform. It is possible to filter tactics puzzles by the phase of the game in which they occurred. In doing so, the following results are shown:

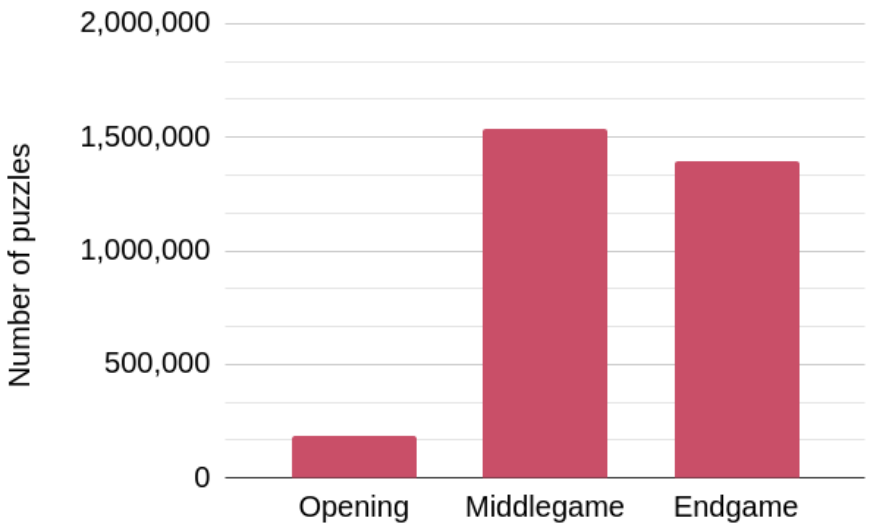

Figure 5.1: *Comparison of puzzles distribution by phase of the game in Lichess.*

The total number of puzzles available in these three classes is 3, 114, 263, of which 182, 372 come from the opening phase, that is, less than 6%, configuring a clear deficit of opening puzzles. And as these are all tactical puzzles, this small portion that comes from the opening only exists because, by chance or poor play, some sharp situations occur in the opening. But they still don't exercise the most important mental skills in this phase, which are memory and the application of general opening principles.

5.3 Existing opening puzzles systems

There are several projects that made efforts to develop tools for practicing openings in a more playful and interesting way than conventional forms. However, some of them impose access barriers, such as paywalls, being outside the scope of analysis of this work. Two existing systems will be analyzed, both of which can be used for free.

5.3.1 Lichess's opening explorer

In the Opening Explorer tool on the Lichess website, there is a feature that resembles an opening practice puzzle. The objective in this puzzle is to always make good moves while advancing in an opening of the user's choice. This mode can be accessed through a button called "Practice with computer" available in the menu. Being in any position available in the Opening Explorer, the user can click on this button and start a puzzle based on that position.

In this case an engine-based solution to the problem of telling whether an opening move is good or bad is used. According to the evaluation given by the engine for the player's moves, there are four possible feedbacks: "Good move", "Inaccuracy", "Mistake" and "Blunder".

The puzzle works like this: the user makes a move, receives feedback from Stockfish, then the computer responds with the best move, also according to Stockfish. The process then repeats. Whenever the user plays any move that is not considered the best by the engine, the best move is shown, providing instructive feedback.

As mentioned before, the engine can be an excellent aid in studying openings, but when it is the only source of truth there are problems. In the case of Lichess's Opening Explorer, this can be clearly observed. Only the moves with the highest ratings given by the engine are considered "Good moves", but as this rating depends on the depth of the analysis, moves that are actually good may not receive the due credit. For example, for the first move of the game, an analysis with a depth of 53 is available in the cache of the application. The moves that are considered "Good moves" are:

- c4
- e4
- g3
- Nf3

In this case, the move d4, known to be an excellent initial move, is considered "Inaccuracy", evidencing the limitation of the computational analysis.

5.3.2 ChessOpenings

ChessOpenings is another project that has the advantages of being free to use and, despite not being clearly licensed, having its source code available, which can be found at https://github.com/Clariity/ChessOpenings. The website has a section called "Train" that allows users to practice some openings in the form of puzzles (Gregory, 2022).

In the case of this tool, anyone can submit opening lines, constituting puzzles with their respective answers. Therefore, the evaluation of the moves is carried out previously, manually by the people who submitted the puzzle.

How the puzzle works is as follows: the user must select one or more openings from a list of names, as well as a color to play with. Thus, a set of puzzles is assembled to be played in a training session. These selected positions will work as targets to be reached in each puzzle. At the beginning of each puzzle, the starting position on the board is shown and the player must always play the main move that leads to the target position. After each correct user move, the computer responds with the main line, and so on. When the user misses a single move, the puzzle ends and the next puzzle of the training session begins.

The first problem that stands out with this type of puzzle is that the evaluation of the moves is done solely via comparison with the "main line", accepting only one correct move at each interaction. This type of evaluation ends up being very limited, far from real life, since in actual games players can almost always play several moves.

Another problem is that, implicitly, the puzzle assumes that the opening ends exactly at the chosen position, which is a position that has a name. There is no possibility to practice moves ahead of the named position.

Finally, there is an issue with the construction of the puzzles, which is not automated. The project relies on manual contributions from the community to add new openings, making arbitrary demands like "The opening must be equal in terms of advantage for both Black and White" and "The opening should be at least 7 moves deep". That is, it can be said that the system works as a computerized version of the study style that already existed in books and classes.

Chapter 6

Puzzle design

The entire puzzle development stage was carried out taking into account what are the main skills to be exercised in this phase of the game, that is, memorization of theory and application of opening principles.

There are several structural questions common to all puzzles, such as what is the objective of the puzzle, what is the role of the position chosen by the user, and what type of traversal in the game graph is the puzzle associated with. Therefore, it is opportune to present the types of puzzle within a framework common to all.

6.1 Main Moves

Puzzle question: given a position, what are the main theoretical moves?

Type of exploration of the game graph: in breadth.

Role of the position chosen by the user: to prevent puzzles from being too static, or a training session from being repetitive, it's desirable to have some level of randomness when fetching a new puzzle. For this reason, the position chosen by the user works as a reference, determining a subtree of the game tree. From this subtree, the actual positions on which puzzles are based are extracted at random.

How the difficulty level influences the puzzles: the higher the degree of difficulty, the further away from the reference position will be the base positions of the puzzles, that is, the deeper the theory that the user must know to complete the puzzle.

How it works: each move is evaluated as correct or incorrect, and then undone so that another can be performed. If all main continuations are discovered, the puzzle ends in success, but if there are 3 mistakes, the puzzle ends in failure.

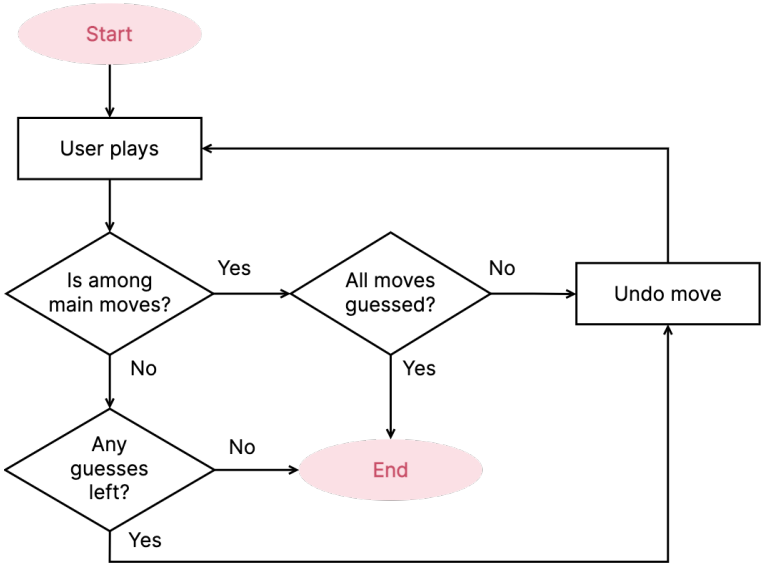

Figure 6.1: *Flowchart depicting how a Main Moves puzzle is played.*

6.2 Survival

Puzzle question: at each turn in a game against the computer, what is the single main theoretical move?

Type of exploration of the game graph: in depth.

Role of the position chosen by the user: the chosen position is precisely the position from which the puzzle will start.

How the difficulty level influences the puzzles: in this type of puzzle, the difficulty levels are designed by hand, being differentiated by the probability distribution that governs the answers given by the computer. At the easy level, the computer can only respond with the most popular moves. At the medium level, the probabilities of answers must match those of answers given by masters in real games. At the hard level, all moves that can be considered theory are equally likely to appear. In this way, the higher the difficulty level, the more likely the player will come across rare lines.

How it works: at each correct move the puzzle advances with an answer from the computer; at each error, the move is undone. If a position is reached where the computer does not have the data resources to provide a theoretical answer, it's said that the computer "ran out of theory", and the puzzle ends in success. If there are 3 mistakes, the puzzle ends in failure.

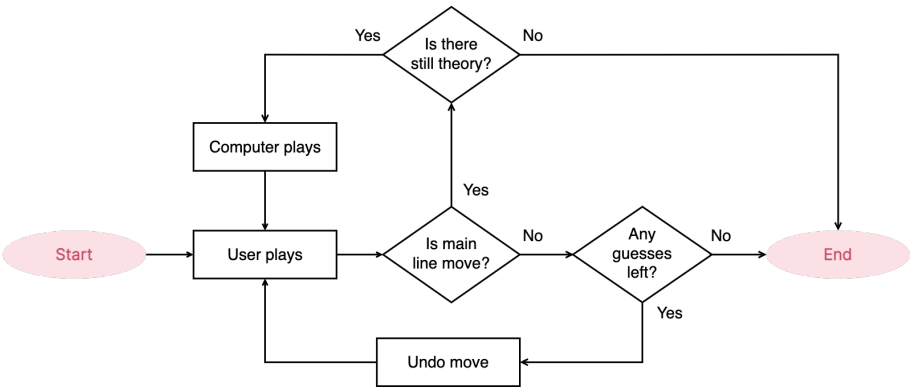

Figure 6.2: *Flowchart depicting how a Survival puzzle is played.*

6.3 Main Line

Puzzle question: given a position, what is the main line leading to it?

Type of exploration of the game graph: in depth.

Role of the position chosen by the user: the chosen position is the target to be reached starting from the initial position.

How the difficulty level influences the puzzles: it doesn't, so in this case the option to set the level would be grayed out. In this type of puzzle, the level of difficulty would have to do with knowing different variations leading to the target position. Problems arise as not all positions have more than one variation leading to it, and when they do, it's not easy to sustain that they are worth remembering. This also justifies "Main Line" as the name chosen for the type of puzzle.

How it works: the user plays both the black and the white pieces, always trying to follow the main line leading to the target position.

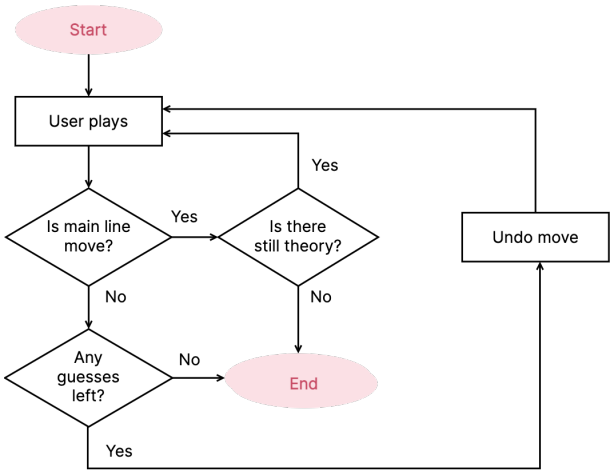

Figure 6.3: *Flowchart depicting how a Main Line puzzle is played.*

6.4 All Theory

Puzzle question: given a position, what is all the theory on it?

Type of exploration of the game graph: in breadth and in depth, breadth-first.

Role of the position chosen by the user: the chosen position is precisely the position from which the puzzle will start.

How the difficulty level influences the puzzles: in any puzzle that involves breadth exploration, there can be a relevance parameter that determines what the main moves are. Variation of this parameter allows us to control how minimally known the answer-moves will be. It is possible to associate the variation of this parameter with the difficulty level of the All Theory puzzle. The higher the desired relevance for the answers, the easier the puzzle. On the other hand, if a lower relevance is set, rarer moves will also be considered answers and the puzzle becomes more difficult.

How it works: the logic is the same as in a breadth-first traversal. Given a position, the user plays a Main Moves puzzle, but at each correct move played, the resulting position is pushed into a queue so it can be the base for a new Main Moves puzzle later.

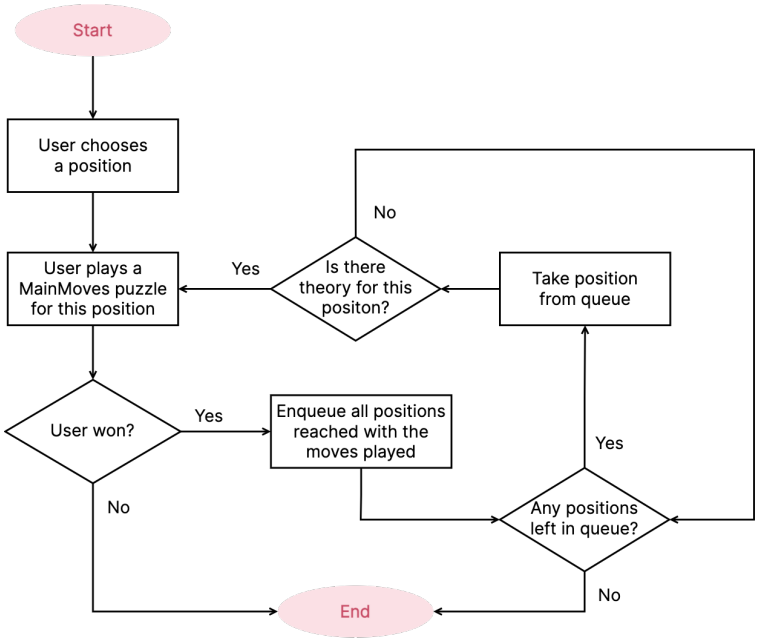

Figure 6.4: *Flowchart depicting how an All Theory puzzle is played.*

Chapter 7

Project decisions

7.1 Free and Open Source Software

As a general guideline for the project, it was decided that the choice of tools should prioritize Free and Open Source Software (FOSS). The definition adopted for FOSS is that given by the prominent portal "It's FOSS", which reads as follows:

> "FOSS means Free and Open Source Software. It doesn't mean software is free of cost. It means that source code of the software is open for all and anyone is free to use, study and modify the code. This principle allows other people to contribute to the development and improvement of a software like a community." (Prakash, 2022).

Furthermore, the license chosen for the project was a license for free software. According to criteria such as code base size and program type, it was chosen to license all system modules where a license makes sense with the GNU General Public License version 3 (GPLv3) license or any later version (Free Software Foundation, 2022).

7.2 Programming language

The main programming language chosen for system development is JavaScript. The reason for this is its extremely high popularity, which gives developers an enormous amount of tools, frameworks and libraries at their disposal. Furthermore, with the advent of the Node.js runtime, it has become possible to write JavaScript on both the frontend and the backend, which makes application development a more concise process.

7.3 Code quality

Whenever possible and reasonable, a choice has been made to program in TypeScript, a superset of JavaScript that provides a useful typing layer. This has been particularly important during the development of the backend codebase to prevent bugs.

Also, a good part of the backend code is covered by automated tests. When writing tests, preference was given to the parts that work directly with data, as they are more stable and more critical. The tool used to run the tests automatically is the Jest framework.

The entire codebase, in addition to the generated data, is kept under version control with git since the beginning of development. The chosen hosting service was GitLab. The project's various modules can be found at https://gitlab.com/krz-sad-tcc.

The Continuous Integration and Continuous Delivery (CI/CD) services provided by GitLab are used to ensure the desired level of code quality on the main branch. Tasks such as linting, testing, and deployment are performed automatically when a push is performed on the main branch.

Chapter 8

Data

8.1 Data modeling

The way by which the project represents its data is one of the most important aspects in system design. As stated in Chapter 3.2, chess games can be easily represented as graphs. This implies that a convenient way to model data on chess games is one that allows its overall graph shape to be kept and traversed.

8.1.1 Nodes

The modeling of each state of a game must take into account the configuration of the pieces on the board and some other information that helps in the application of game rules. Usually, states are represented using Forsyth–Edwards Notation (FEN), which is a string containing 6 fields:

- Full description of the board, with position of pieces and empty spaces.
- Whose turn is it, White or Black
- Possibility of performing long and short castling for both colors
- En passant target
- Halfmove clock
- Fullmove number

An example of a FEN string is as follows:

```
rn1qkbnr/pp2ppp1/2p5/3pPb1p/3P3P/8/PPP2PP1/RNBQKBNR w KQkq h6 0 5
```

FEN is a very informative description of the game situation, sometimes even providing the ability to differentiate variations by which the same position was reached. For applications that demand that positions be represented in the same way, regardless of the history that led to them, there is another notation, the Extended Position Description (EPD).

An EPD string is much like a FEN string, but without move numbers, and containing the en passant target field only when an en passant is legal in that position. Below is the EPD string representing the same position as the FEN in the previous example:

rn1qkbnr/pp2ppp1/2p5/3pPb1p/3P3P/8/PPP2PP1/RNBQKBNR w KQkq -

It is clear that by the nature of the game graph, which contains cycles, there can be more than one way to reach the same position. Therefore, for the same EPD there may be several FEN. So, as a unique identifier of positions in the context of a puzzle application, EPD works best. But as a FEN is required as an input by some chess tools, a FEN also had to be included in each graph node.

Furthermore, another highly relevant piece of information for a graph node is the number of games that passed through the position. This is the main measure of relevance of an opening position.

The name and ECO code of the position, when applicable, are not crucial data, but were included in the modeling as they are good aids to give context to users when faced with positions on the board.

The last field that a node contains is metadata that says whether it is a leaf node in the graph or not. Without this field, some operations would be prohibitively inefficient.

The final schema of a position is as follows:

Property	What it is	Data type
`eco`	Encyclopaedia of Chess Openings code of the position	`string`
`epd`	Description of the position in Extended Position Description notation	`string`
`fen`	Description of the position in Forsyth-Edwards Notation	`string`
`isLeaf`	Indicates whether a node is a leaf or not	`boolean`
`name`	Name of the position	`string`
`nGames`	Number of games that reached the position	`integer`

Table 8.1: *List of attributes contained in a position.*

8.1.2 Edges

Changes of state in a chess game happen when players make moves. The most important information when modeling the moves are the initial state and final state, so identifiers for these states have been included in the edges with the labels "from" and "to".

The next relevant piece of information is the weight of the edge, that is, the number of times the move has been played from the position identified as "from". Note that, for efficiency purposes, it is important to maintain "number of games" properties both in nodes and edges, as knowing one does not immediately imply knowing the other. This is because, as the graph contains cycles, nodes can have two parents. So knowing the number of times a move has been made does not reveal how many times the resulting position has been reached, as one might think. To know how many times a position was reached, it is necessary to add up the moves that led to it starting from all its parents.

The last property included in the modeling is the identifier of the movement performed in SAN.

The final schema of a move is as follows:

Property	What it is	Data type
`from`	ID of the starting node	`string`
`to`	ID of the ending node	`string`
`san`	Move in Standard Algebraic Notation	`string`
`nGames`	Number of games	`integer`

Table 8.2: *List of attributes contained in a move.*

8.1.3 Model

The data model is shown in the following figure:

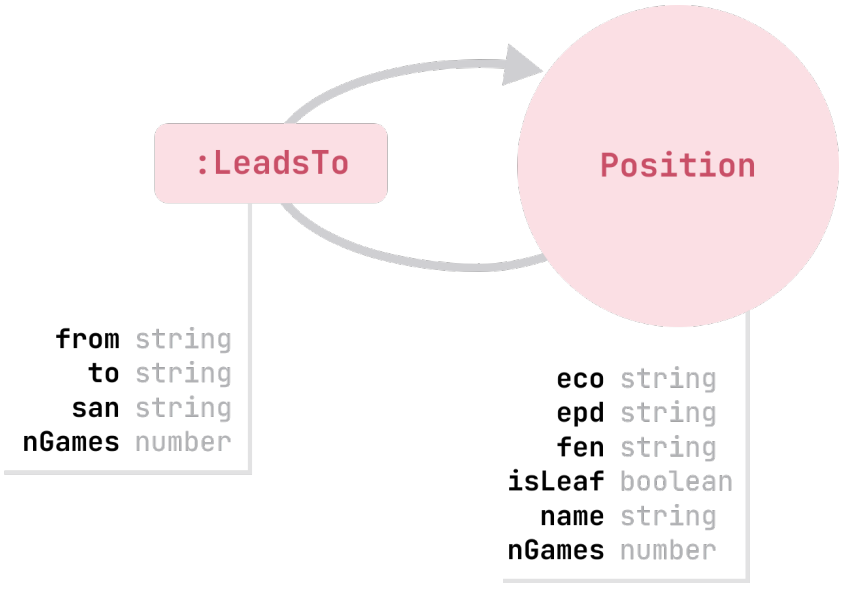

Figure 8.1: *Representation of the graph model, including attributes and their types.*

8.2 Data acquisition

First, it's important to state that the project does not create any data on its own, relying entirely on existing data coming from sources made publicly available by Lichess. In particular, two sources are currently used: the "Masters database" and the "Chess opening names" database.

The first one is a cluster of information about games played by chess masters, and is provided either via direct download (https://database.lichess.org/) or in the form of an online API (https://lichess.org/api#tag/Opening-Explorer). As the file available for download also includes games played by all players on the Lichess platform, the size is excessively large, making it difficult to manipulate the files on certain personal machines. Therefore, in order to facilitate an easy start in the development of the project, it was decided to get the data via the online API, even if it takes longer.

The second source is a set of tables containing regular information about named positions, such as EPD, ECO, and a name by which they are known (https://github.com/lichess-org/chess-openings). This dataset is substantially smaller in comparison to the previous source, but that's expected since only a few positions are relevant or popular enough to be granted a name. As a result of being smaller and provided in the form of tables, it is possible here to download data and consume it locally.

8.3 Database management

With the advent of the graph-oriented databases, storing graph-like structures with all their properties became a viable option. This approach features a number of advantages, the most notable of which is how trivial it makes it to implement queries that depend on graph traversals. For instance:

Given a position, find all the paths that start at the initial position and end there. Given a position, find whether it is at least N steps away from the border of the graph. Given an unnamed position, find its closest named ancestor.

Although it makes querying the database more convenient and idiomatic, the decision of using a graph-oriented database does not come for free, as it also brings new issues to consider, such as lack of support and the use of lesser-known query languages.

The decision of which database management system to use comes down to evaluating the available options of graph-oriented databases, among which preference is given to those with a solid community and frequent updates. Table 8.3 shows the most popular options available.

Considering the project's goal to favor FOSS solutions, it made sense to rule out Amazon Neptune from the options. Ultimately, the decision was to discard Neo4j as well and use ArangoDB, given that the former has a complicated licensing system with multiple amendments, that, in the end, result in a limited free version compared to the paid version.

Criterion	Neo4j	ArangoDB	Amazon Neptune
Technology	Purely graph-oriented	Doc-oriented with support to graphs	Purely graph-oriented
Licensing	GPLv3 with aditional stricter licenses	Apache	Proprietary
Pricing	Community edition: Free Enterprise edition: Paid	Community edition: Free Enterprise edition: Paid	Paid

Table 8.3: *Comparative of most popular graph-orientend databases.*

ArangoDB is a hybrid database management system, with simultaneous support for document, key-value and graph data models. Data is stored using JavaScript Object Notation (JSON) documents, giving it the flexibility to handle data with many levels of nesting, as well as data without a fixed schema. This is similar to how MongoDB and other popular NoSQL databases store data. So the choice for ArangoDB is also justified by the possibility of, in a future iteration of the project, storing information that is not represented by graphs. By way of comparison, Neo4j is not able to do this, being restricted to graph orientation.

In ArangoDB, creating a graph simply means establishing two collections of documents, one for vertices and one for edges. A vertex is just a regular document, with no special requirements. An edge, however, is a particular type of document with additional attributes, `_to` and `_from`, which define a directional relationship.

On top of that, ArangoDB provides a wide range of additional tools, such as a full-text search mechanism, support for indices and transactions, data analysers, aggregation tools, and so forth, making it an excellent choice for this project.

Chapter 9

System modules

The final system consists of five separate modules with their own responsibilities. Of these, only two, puzzles-api and puzzles-front, are intrinsically connected to the web application. They are the ones directly in charge of building and presenting the webpage to the users. As for the other three modules, they work as tools meant to facilitate the development process.

The web application proper is organized using client-server architecture, which is an "umbrella term for any application architecture that divides processing among two or more processes" (Oracle, 2022). In this particular case, the division takes place in the form of backend – code tightly integrated with the database – versus frontend – related to presentation and user interaction. Even though they are both deployed to the same physical machine, they run as separate services, independent of each other. This is a great solution, because it allows them to eventually be adapted so that disruptions in one service do not affect the other.

In common, most of the modules have the fact that they use the "chess.js" library, which is a collection of useful functions, types and tools to handle chess logic (Hlywa, 2022). This not only makes it faster but also safer to develop the application, since there is no need to reimplement, from scratch, all the extensive and complex logic inherent to chess games. By doing so, it becomes trivial to perform tasks such as identifying which moves are valid, if a piece can be captured, whether there is a check or checkmate configuration, among others.

9.1 db-tools

This is a useful higher-level layer built upon ArangoDB's official JavaScript tools. It aims to provide a consistent way of approaching the database, abstracting common operations, as well as providing TypeScript bindings and types that are widely used throughout the project.

Being exported as a JavaScript library, it can be imported in whichever project it is needed, removing the necessity to duplicate code related to database connection. Given its implementation as a library, this module is distributed using GitLab's Package Registry, a safe space where libraries can be published and used as dependencies in downstream projects (GitLab, 2022).

Some of its responsibilities are: providing additional context to exceptions thrown by ArangoDB; exporting a convenient, type-safe query function that abstracts the process of reading from the database; exporting a function that encapsulates the process of connecting to the database; defining constants containing the name of collections and graphs used.

9.2 graph-builder

As previously mentioned, the project uses data collected from multiple sources that are parsed and organized in the most convenient way before being fed to the database. This is the module responsible for putting together the pieces needed during this phase.

The process to populate the database was devised in such a way that it is possible to do it across multiple sessions, gradually expanding the dataset as desired. This is necessary for many reasons:

- The script relies on Lichess's online APIs, which only accept a limited number of requests per minute, limiting how many positions can be queried in a given timeframe;
- Given the size of the graph containing all possible chess games and the limitation above, it is not feasible to process every one of the graph's nodes in a single session;
- The database has to always be in a consistent state, regardless of any possible connection issues, either to Lichess or between graph-builder and the database. As a consequence, should any potential corrupted or invalid data be identified, the process stops, the faulty piece of information is not saved, and the process can be restarted exactly where it stopped.

The algorithm used when populating the database is as follows:

```
function build_graph(min_games: number) {
  if (!root_in_database()) {
    add_root()
  }
  expand_graph(min_games)
}
```

Listing 1: *Source code of a simplified build_graph function.*

The `build_graph` function is basically a wrapper around the more generic function `expand_graph` whose intent is to handle the special case of the root position. Since the root is not a child of any other position, it has to be manually inserted at the very beginning. After this case is handled, the function to expand the graph enters the scene: `expand_graph` receives `min_games`, a parameter that says how far the graph should be expanded. Positions that have more than `min_games` games in the data source are considered. The others are rejected. The function `expand_graph` is as follows:

```
function expand_graph(min_games: number) {
  const leaves = get_expandable_leaves(min_games)
  const q = new_queue(leaves)

  while (!empty(q)) {
    const parent = queue_head(q)
    let lichess_data = fetch_lichess_data(parent)

    for (const M of lichess_data.moves) {
      let child = get_vertex_from_db(M)
      if (child !== null) {
        create_edge(parent, child)
      }
      else {
        child = create_vertex_from_move(M)
        add_position_name(child)
        create_edge(parent, child)

        if (child.games >= min_games) {
          queue_push(q, child)
        }
    }

    if (!empty(lichess_data.moves)) {
      mark_as_non_leaf(parent)
    }
  }
}
```

Listing 2: *Source code of a simplified expand_graph function.*

This procedure works by executing a breadth-first construction of the game graph. It starts by fetching every leaf position already in the database and putting them into a queue.

At this point, it is convenient to remark that the algorithm assumes every position to be a leaf, until it and all of its children are correctly inserted into the database. This is a safety mechanism, which is meant to make sure that, if a position has not had all of its children processed correctly, this can be done in a future run of `expand_graph`.

Once the queue is filled with the leaf positions, one at a time, they have a list of their child positions fetched from Lichess. Then, for each of these children, an edge is created between it and the parent position. After every one of these children has been processed, the original position is marked as no longer being a leaf, and the process moves to the next position in the queue.

9.3 db-data

This module works akin to a backup service, holding snapshots of the database. Data stored here uses the format JSONL – a text file in which every line has to be a valid JSON object (JSONL, 2022) – and is exported from the database using a script provided by the db-tools module.

Data in the repository is organized using the parameter `nGames`. That means every time a new dataset is uploaded, the commit message of such an update includes the number of games used when said data was generated. This parameter is the same one as explained on the graph-builder module.

The relevance of this module comes mainly from the fact that, because it takes a long time to generate the data that's put into the database, the expansions are performed sporadically. The latest version of the dataset, for instance, was reached after more than 50 expansions that took place over almost two months. Therefore, it makes sense to regularly back up the most recent version achieved, so that it is possible to restore it faster and reliably without regenerating everything from scratch.

Apart from that, data from this repository is also used as seed to fill the database when running the tests pertaining to the puzzles-api module. This ensures that these tests run against a realistic dataset, helping to catch bugs.

9.4 puzzles-api

Following the client-server architecture, this module holds the codebase for the application's server and has been written using TypeScript. It consists of a Hypertext Transfer Protocol (HTTP) API that transits standardized JSON objects, and it is built on top of the popular framework Express.

A considerable amount of thought was put onto the decision for Express and mainly relies on the fact that, being a minimal and non-opinionated framework (OpenJS Foundation, 2022), it adds very few abstraction layers to the codebase. This allows developers to still have full control over the inner workings of the application, without compromising their ability to make their own decisions as to how the source code should be organized. Apart from that, it is also among the most popular backend JavaScript frameworks as of 2021, being known by 97% and used by 81% of the respondents for the State of JavaScript survey, which gives it a huge community to benefit from (Greif, 2021).

The codebase is organized in such a way that each puzzle type is associated to its own endpoints, containing backend logic responsible for bundling every piece of information required to display and play that sort of puzzle. This makes it easier to apply fixes, fine-tune or even rewrite a puzzle's logic without affecting the others.

The module's API is composed by the following endpoints:

Endpoint	Description
`/puzzles/main-moves`	Organizes a survival puzzle, using the parameters received, according to the rules exposed on Chapter 6.
`/puzzles/survival`	Organizes a survival puzzle, using the parameters received, according to the rules exposed on Chapter 6.
`/positions`	Looks for a specific position in the database using its EPD.
`/positions/history`	Given a position EPD, performs a backwards traversal in the graph's main line until the root. In the end, the path taken is returned, representing the most likely sequence of moves played during a game.
`/positions/search`	Uses a term to search for positions in the database. Both name and FEN fields are used when matching records.
`/positions/:key/info`	Given the ID of a position in the database, tries to gather theory information for it by crawling data from the Wikibooks portal on Chess Opening Theory.

Table 9.1: *List of endpoints available on puzzles-api.*

9.5 puzzles-front

Just like the previous one, the last of the system components is also written using TypeScript, and it is responsible for its frontend. That means rendering the user interface, dealing with user interactions and communicating with the backend API.

This component uses the Single-page Application (SPA) approach, in which the browser retrieves all the necessary code with a single page load (Flanagan, 2006), and avoid new page loads at all costs. This choice pairs quite nicely with the client-server architecture. Here, the frontend service acts as a client that uses a connection to the backend server in order to fetch new data, which is then used to change the page's content. This also contributes to the goal of building an actual web application rather than a simple document-based website, since the final outcome is a highly dynamic and responsive interface.

Since performing dynamic changes to the page content using plain JavaScript can become quite tedious, the application has been written using the React framework. Backed by Facebook and countless other individual contributors (Meta Open Source, 2022), React has been in the market for a long time, being as of 2021 actively used by 80% and

recognized by strikingly 100% of the respondents for the State of JavaScript survey (Greif, 2021). Among the reasons to use React are:

- Its huge community and ecosystem, providing fast support and plenty of ready-made libraries and pieces. Not to mention its stability;
- The fact that it tries as much as possible to stick to plain JavaScript syntax, ditching usage of templating languages. This puts the entirety of JavaScript, as well as its patterns and abstractions, at the developers' hands.

For styling purposes, the choice resides on Tailwind Cascading Style Sheets (CSS). Being a utility-first CSS framework, it does not come with fully implemented components – buttons, text fields, combo boxes –, but rather with atomic, general-purpose CSS classes that can be used across the whole application (Tailwind Labs, 2022).

```
<button class="btn btn-primary" />

<button class="padding-1 margin-1 bg-white color-red border-gray
               border-1 border-solid shadow-8 uppercase
               focus:bg-pink focus:color-black" />
```

When comparing the two pieces of code above, the initial impression is that the one using Tailwind CSS appears to be more cluttered and hard to read. It however has the advantage of being clear as to which CSS definitions are currently in use. This completely avoids of the most common problems with CSS: using the cascade [1] and the specificity [2] algorithms to determine what to apply when competing selectors specify values for the same properties in the same elements.

It is also important to remark that the puzzles-front project does not reimplement a chess board itself. Rather, the option was to use Lichess's own implementation, which is a FOSS chess board known as "Chessground" (Lichess, 2022b). Given that it has been developed with an important application in mind, it has not only the stability and security necessary to this project, but also the completeness in terms of features available.

Images showing details of the current state of the user interface can be found in the Appendix A.

[1] https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade

[2] https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity

Chapter 10

Deployment

Deploying an application requires two fundamental steps: obtaining a domain name and obtaining a hosting service for the website.

The domain name chosen was "Still Theory" in honor of grandmaster Ben Finegold, who sometimes uses this phrase with joking intentions when the situation in a game is absurdly far from any chess theory. As ".org" domains are intended to be used by non-profit organizations, this is an appropriate domain extension for our application. The domain was then purchased from Google Domains, which was chosen because it offered a lower price than its competitors.

For the hosting service, the company Amazon Web Services (AWS) was chosen because it offers a multitude of facilities, configuration possibilities and, especially, the first year free of charge for a service that is suitable for our application. This service is Amazon Elastic Compute Cloud (EC2), through which we have access to a virtual machine with 1 virtual Central Processing Unit (CPU) with i386/x86-64 architecture and 1 GB of Random Access Memory (RAM).

This virtual machine runs the Debian 10 operating system, and all the necessary programs have been installed on it. Among them, the NGINX web server stands out, which is used as a reverse proxy. As already mentioned in the chapter on code quality, the deployment task is performed automatically as soon as a new version of the code is published in the main branch of the git repository.

Chapter 11

Conclusion

11.1 Personal perspectives

In this section, opinions will be given about difficulties and lessons learned during the development of the project from the points of view of the two developers, Renan Krzesinski (krz) and Ygor Sad Machado (sad).

11.1.1 krz

During development there were several times when I only conceptually knew how things worked. In these cases it was necessary to choose an existing tool, learn to use it and make things work. This process of getting started with new tools is painful, but the knowledge and experience gained are enormous.

The most relevant specific cases in this sense are the ArangoDB database management system and the ArangoDB Query Language (AQL), the Jest testing framework and the great depth that I needed to carry out in the React library to contribute to the construction of the frontend.

Also with respect to the frontend, I had my first contact with the world of design, which Ygor has already mastered. In all the weekly meetings he taught me something new regarding the construction of user interfaces at the level of tools, usability and beauty.

The experience of building an entire system from scratch has been very rewarding. Especially since it's a system that I dreamed of myself, with features that I wanted to exist on other platforms so that I could practice chess openings with puzzles, which I couldn't. Now, while we have something useful for these purposes, it's easy to understand why such puzzles didn't exist before: it wasn't a trivial task.

11.1.2 sad

Playing chess has never been one of my strong interests. Even though I admired those who dared to try and learn the inner workings of such a complex game, I myself never felt tempted to do it. That however changed as I was invited by Renan to proceed with this project.

Facing such, for me, unexplored land certainly felt like a challenge worth facing. That didn't come without struggles, though. More often than not, it felt as if I would never understand so many concepts and topics, especially those named after trivial things - engine, theory, lines. But patiently Renan introduced me to everything I should know. If I forgot something, he would promptly remind me and answer all my questions. This made the subject less mysterious and much more approachable. For sure, to this day, I can name things I still do not fully comprehend or can not explain, but at least I feel confident, even if just a little, to talk about chess.

I feel proud that we could deliver a well-tested application built according to what is considered the best practices for web development. Proud of building something that allowed me to put every little piece of knowledge I gathered throughout my university years, from graph algorithms to web services. In retrospect, I see now that there could not be a more inspiring, exciting and fascinating project to finish my undergraduation.

11.2 Future work

As with all software projects, this project is not expected to be finished. It is natural that development is constant, providing maintenance, fixing bugs, etc. But beyond this common future work, there certainly are some specific aspects yet to be developed that are worth naming.

The first major shortcoming of the project in its current form is the absence of any feedback collection from potential users. It would be much healthier to guide the development of the application using the tips given by a large number of chess players informing what they find interesting and what they don't. In addition, it can also be very useful to collect metrics such as usage time, with which it is possible to have confirmation regarding the success or failure of a piece of software. This type of feedback collection has the potential to make development cycles faster and the resulting application better.

Keeping that in mind, some yet-to-be-implemented features that seem quite interesting for the project are:

- The two puzzle types that have been designed but not implemented yet
- User registration to allow tracking progress
- A specific version for mobile devices
- Make the server more robust, as some queries involving graph exploration are computationally heavy
- Use docker to facilitate the deployment

Furthermore, it is important to state that Lichess is the main source of data, ideas and inspiration for this project. If at some point it becomes clear that Still Theory is really an interesting application, one idea we have is to reach them and propose an integration in the form of new features for Lichess. That would involve transferring all the technology we have on this project to their technology stack. That is, starting a graph-oriented database in the Opening Explorer module, modifying the server code and adapting the user interface. This would be a really big step, but an excellent way to conclude this contribution.

Appendix A

Images of the user interface

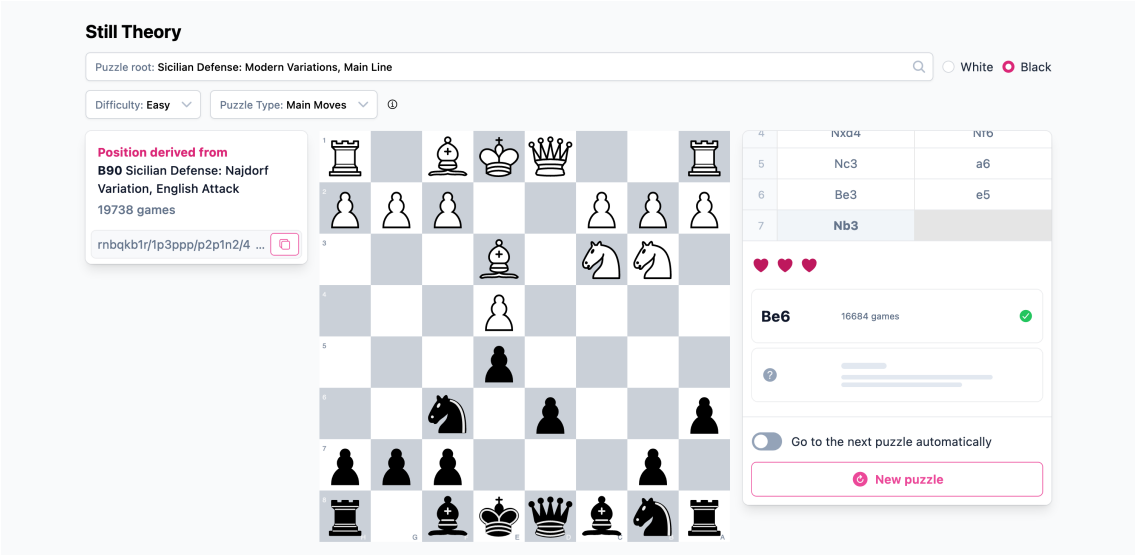

Figure A.1: *Screenshot showing a puzzle in the middle of the gameplay.*

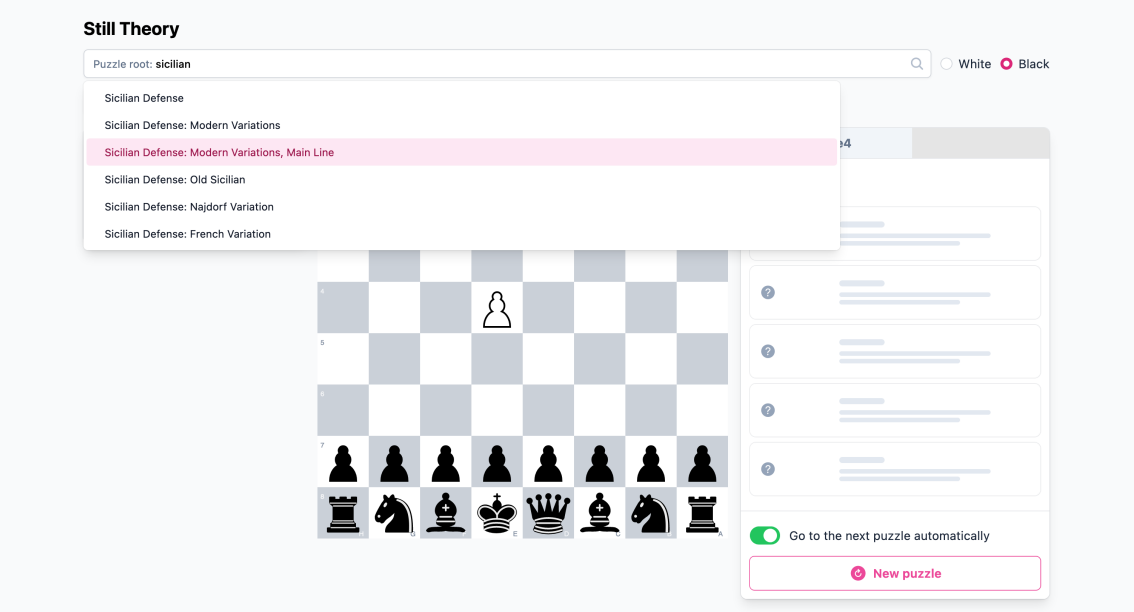

Figure A.2: *Screenshot of the search mechanism, used to choose the position users want to play with.*

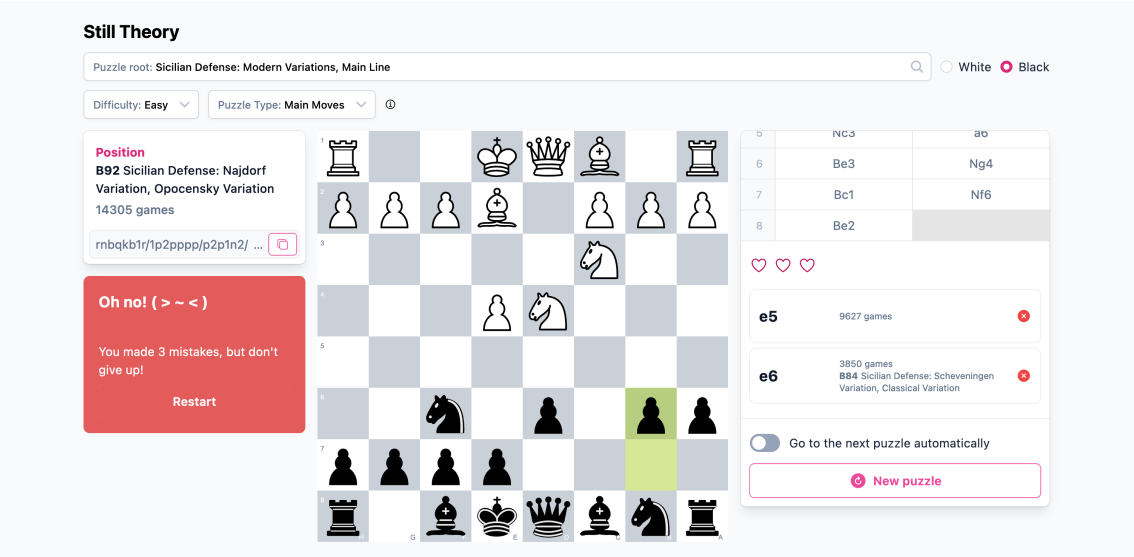

Figure A.3: *Screenshot of a puzzle of type Main Moves when the user loses it.*

Figure A.4: *Screenshot of a puzzle of type Main Moves when the user wins it.*

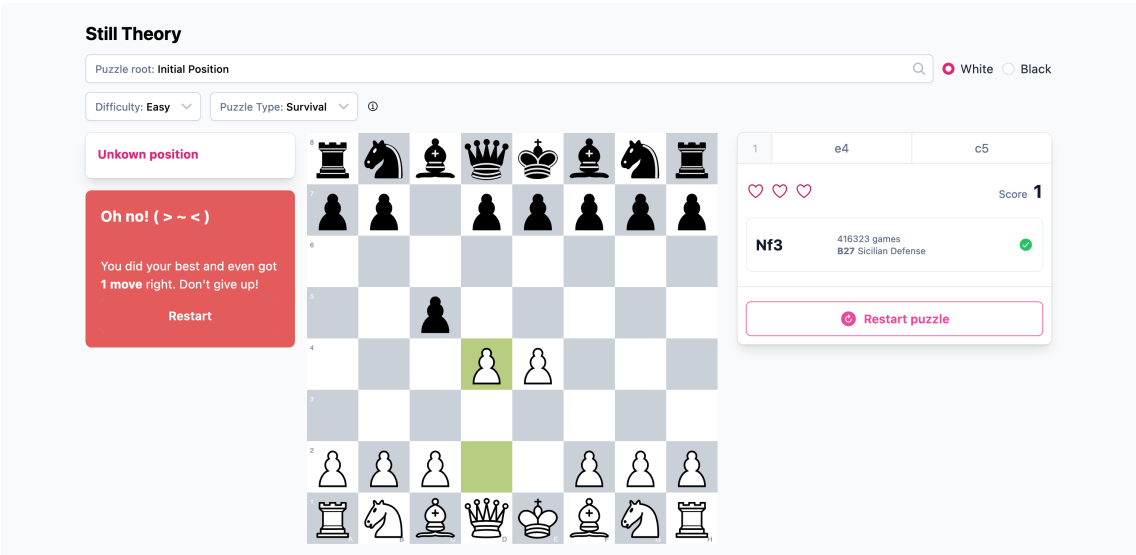

Figure A.5: *Screenshot of a puzzle of type Survival when the user loses it.*

Figure A.6: *Screenshot of a puzzle of type Survival when the user wins it.*

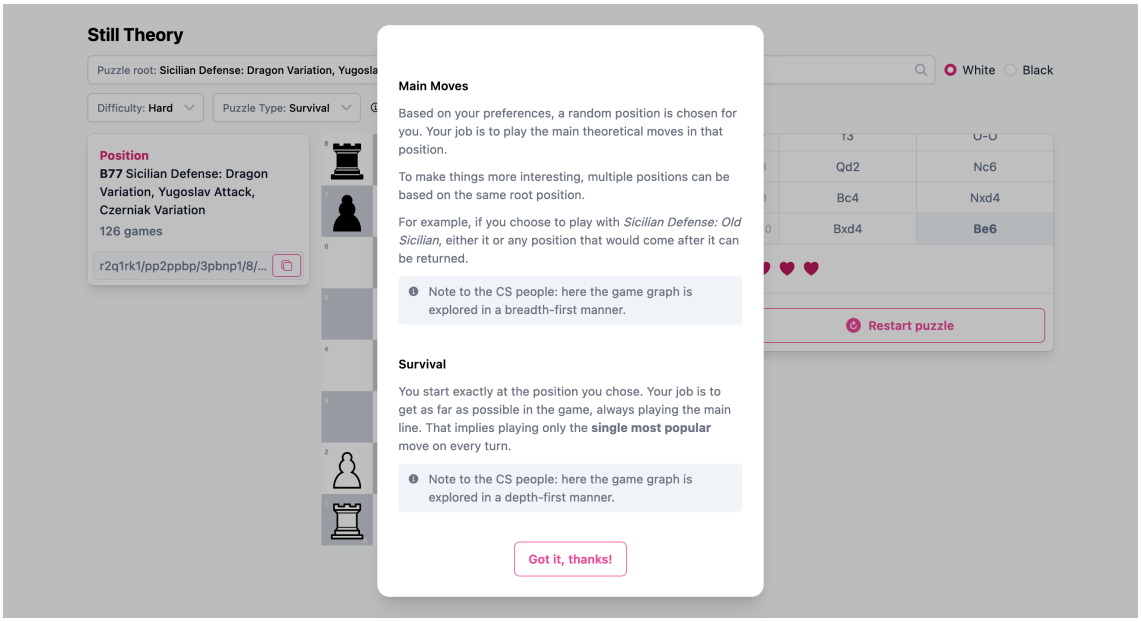

Figure A.7: *Screenshot depicting the modal containing instructions on how to play.*

References

[Capablanca 1935] José Raúl Capablanca. *A Primer of Chess*. Harcourt Brace Jovanovich, 1935 (cit. on p. 5).

[Capablanca 1966] José Raúl Capablanca. *Capablanca's Last Chess Lectures*. London, UK: Herbert Jenkins Ltd, 1966 (cit. on p. 12).

[Chernev and Harkness 1945] Irving Chernev and Kenneth Harkness. *An Invitation to Chess. A Picture Guide to the Royal Game*. New York City (NY), USA: Simon and Schuster, 1945 (cit. on p. 9).

[Chess Programming Wiki 2022] Chess Programming Wiki. *Engines*. 2022. URL: https://www.chessprogramming.org/Engines (visited on 12/21/2022) (cit. on p. 11).

[Duplessis 2021] Thibault Duplessis. *How many members does Lichess have?* 2021. URL: https://lichess.org/forum/general-chess-discussion/how-many-members-does-lichess-have#10 (visited on 12/21/2022) (cit. on p. 15).

[Eade 2005] James Eade. *Chess for Dummies. A Reference for the Rest of Us!* 2nd ed. Hoboken, (NJ) USA: Wiley Publishing, Inc, 2005 (cit. on p. 15).

[Emms 2006] John Emms. *Discovering Chess Openings. Building a Repertoire from Basic Principles*. London, UK: Gloucester Publishers Limited, 2006 (cit. on p. 10).

[Fiekas 2022] Niklas Fiekas. *Syzygy endgame tablebases*. 2022. URL: https://syzygy-tables.info (visited on 12/21/2022) (cit. on p. 7).

[Flanagan 2006] David Flanagan. *JavaScript. The Definitive Guide*. 5th ed. O'Reilly, 2006 (cit. on p. 35).

[Free Software Foundation 2022] Free Software Foundation. *How to Choose a License for Your Own Work*. 2022. URL: https://www.gnu.org/licenses/license-recommendations.html#software (visited on 12/21/2022) (cit. on p. 23).

[Gelfand 2020] Boris Gelfand. *Technical Decision Making in Chess*. Glasgow, UK: Quality Chess UK Ltd, 2020 (cit. on p. 11).

[GitLab 2022] GitLab. *Package Registry*. 2022. URL: https://docs.gitlab.com/ee/user/packages/package_registry/ (visited on 12/19/2022) (cit. on p. 32).

[Gregory 2022] Ryan Gregory. *ChessOpenings*. 2022. URL: https://chessopenings.co.uk/train (visited on 12/21/2022) (cit. on p. 18).

[Greif 2021] Sacha Greif. *The State of JS Survey 2021*. 2021. URL: https://2021.stateofjs.com/en-US/ (visited on 12/21/2022) (cit. on pp. 34, 36).

[Hlywa 2022] Jeff Hlywa. *chess.js*. 2022. URL: https://github.com/jhlywa/chess.js (visited on 12/19/2022) (cit. on p. 31).

[JSONL 2022] JSONL. *JSON Lines Documentation*. 2022. URL: https://jsonlines.org (visited on 12/21/2022) (cit. on p. 34).

[Lichess 2022a] Lichess. *About Lichess*. 2022. URL: https://lichess.org/about (visited on 12/21/2022) (cit. on p. 15).

[Lichess 2022b] Lichess. *Lichess Chessground*. 2022. URL: https://github.com/lichess-org/chessground (visited on 12/21/2022) (cit. on p. 36).

[Meta Open Source 2022] Meta Open Source. *React Documentation*. 2022. URL: https://reactjs.org/ (visited on 12/21/2022) (cit. on p. 35).

[OpenJS Foundation 2022] OpenJS Foundation. *Express Documentation*. 2022. URL: https://expressjs.com/ (visited on 12/21/2022) (cit. on p. 34).

[Oracle 2022] Oracle. *Distributed Application Architecture*. 2022. URL: https://web.archive.org/web/20110406121920/http://java.sun.com/developer/Books/jdbc/ch07.pdf (visited on 12/19/2022) (cit. on p. 31).

[Pandolfini 1989] Bruce Pandolfini. *Chess Openings: Traps and Zaps*. Simon and Schuster, 1989 (cit. on pp. 6, 9).

[Polgár 2014] Judit Polgár. *A Game of Queens*. Glasgow, UK: Quality Chess UK Ltd, 2014 (cit. on p. 11).

[Prakash 2022] Abhishek Prakash. *What is FOSS*. 2022. URL: https://itsfoss.com/what-is-foss/ (visited on 12/21/2022) (cit. on p. 23).

[Shannon 1950] Claude E. Shannon. "Programming a computer for playing chess". *Philosophical Magazine* 41.314 (Mar. 1950) (cit. on p. 7).

[Shenk 2014] David Shenk. *The Immortal Game. A History of Chess*. Souvenir Press, 2014 (cit. on p. 5).

[Similarweb 2022] Similarweb. *chess.com vs. lichess.org Ranking Comparison*. 2022. URL: https://www.similarweb.com/website/chess.com/vs/lichess.org/#overview (visited on 12/21/2022) (cit. on p. 15).

[Tailwind Labs 2022] Tailwind Labs. *Tailwind CSS Documentation*. 2022. URL: https://tailwindcss.com/ (visited on 12/21/2022) (cit. on p. 36).

[United Nations 2022] United Nations. *World Chess Day*. 2022. URL: https://www.un.org/en/observances/world-chess-day (visited on 12/21/2022) (cit. on p. 5).