ChuzzGen - A guided system for Chess puzzle generation

Louis Winder

Abstract

This paper will detail the results from the creation a Chess puzzle-making system. The system can generate many different board configurations and evaluates them in order to produce positions whereby there are checkmate patterns for winning games. The goal of the system is to generate valid and "efficient" positions that can facilitate the study of game winning moves for human players to improve their skills. This report will introduce the system, how it works and highlight the result of generations. The system will be critically evaluated to assess its generation potential and determine how "creative" the generations are. They will be compared against rubrics from the field of Computational Creativity and given an evaluation based upon this. Finally, outlined will be the potential for expansion of the existing system and ideas for future work on the subject.

1 Introduction

Chess is universal. Established over 1500 years ago, it is played daily by many people. Arguably the most popular online Chess platform, Chess.com estimates that in February of 2023 there were around 37 million games played per day [1]. With this many players (including many professionals) it follows that there should be a concrete way to practice tactics and hone skills. This is where Chess puzzles come in. Chess puzzles in the general sense are puzzles that a player must solve by using the rules of chess (i.e. making moves) in order to win some kind of advantage over the other player. This typically includes checkmating patterns or moves that eventually win material, but can sometimes include moves that lead to positional advantages. Again, one the biggest places for puzzles is Chess.com. They claim to have a database of over 500,000 puzzles [2] that are generated mostly by deriving from real games played on the website.

Figure 1: An example puzzle taken from Chess.com. White to move must find the checkmate in 1 move (Queen to d8).

In contrast to this, the system outlined in this paper has no knowledge of positions in any kind of database or similar. Instead, generated board configurations are guided primarily by an index of probabilities and an "efficiency threshold". This way, generations could be said to exhibit creative tendencies and because Chess has so many different positions, can provide a different take on the problem to give variety in puzzles for players. The efficiency threshold will be determined by how many moves the generated position needs for a checkmate. Puzzles are created so that the player that is South of the board is the checkmate to be found for.

This report will describe the system that has been created to generate such puzzles. Outlined will be the overall design methodology, results from testing of the system and a critical appraisal of the generations including how seemingly creative they appear.

2 Background

One of the very first things I did before conducting this project was to analyse existing Chess puzzles. Chess.com puzzles were a good place to start for this as they have a database with over half a million different puzzles. I analysed many of these puzzles and took note of the recurring patterns and ideas that I could use as inspiration for my work. Analysing these puzzles allowed me to obtain a wide range of candidate solutions which could guide the generation of my own ones.

Further to this, I also looked at existing algorithms that have been made which produce puzzles. While researching this there appeared to be a main theme that instead of randomly generating the systems would traverse a database of real games and look for advantageous positions, which it would then save as a candidate for a puzzle. This method would likely generate more realistic puzzles however as generations are constrained by

real positions I argue they don't really fall under the umbrella of "creativity", hence why in this paper the system used creates puzzles in a different way. While there exists some examples of academic papers related to abstract puzzle generation [3, 4], there are very few that deal with Chess puzzles in particular — examples that I have come across are typically individual projects by single people. Probably the most convincing I found was a method similar to the Chess.com one (algorithm not publicly available) which is detailed in this GitHub repository.

While some inspiration was drawn from researching similar systems, overall my results were novel and unique which was the overarching goal.

3 Design methodology

To display a Chess board, one must first create a Chess game. Fortunately, having worked on a different project previously I already had an almost-complete Java project that implements the rules of the game. Thus, initially I had to modify this code so that it can be used to generate puzzles. This took a short amount of time. The application uses a Java Swing GUI to display the board and relevant statistics along with a Java-coded backend which deals with the (already implemented) game rules as well as the code that manages the puzzle generations.

Entire generation scheme

Before implementing the code for generation it was necessary to come up with a method for determining whether a generated position was puzzle-worthy. After a few iterations, the final process that was decided on for generating the puzzles follows the steps outlined in Figure 2.

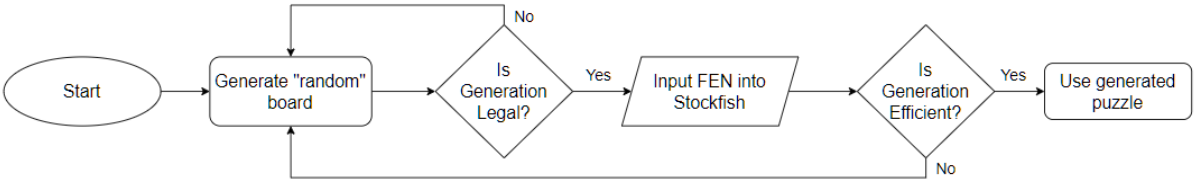

Figure 2: Flowchart showing how the puzzles are generated.

Firstly, a board is configuration is generated using certain heuristics to guide it (see Generation of board positions). When a board is generated we then check if it is a legal position according to the rules of the game (i.e. no checks, too many pieces etc.). If it is legal, we can move on to efficiency verification but if it is illegal we generate a new position. After the legality check, positions are then checked for their "efficiency". This is done by comparing the generation against an pre-defined efficiency threshold and only passing those generations that meet this criteria. Generations that pass both legality and efficiency verification will be utilised.

Generation of board positions

In order to guide generations so that they are meaningful and not entirely random, a method was designed to guide the generation scheme. The board is assigned two different types of probability weightings: one for Pawns and another for all other piece types (King, Bishop etc.). These probabilities are used only to influence the *rows* that pieces are able to generate on. The result of using probability weightings to constrain generation to certain rows means generated board states appear much less random and like they are more similar to real positions one may encounter during a game of Chess. Pawns were given a separate list of probability weightings as they have a different subset of rows they can be on (they can't generate on rows 1 and 8 for example).

Next, we initialise the pieces that are to be placed in their positions on the board. The rules of piece generation (for each colour) are defined as:

- Each King has a 100% chance of generating
- Each Queen has a 50% chance of generating, but only once
- Castles, Bishops and Knights have a 50% chance of generating once, and a 25% chance of generating twice
- There is a 100% chance for 1 Pawn to generate, and for each new Pawn (up to a maximum of 8) this chance reduces by 12.5%
- Additionally, Pawns have a 0% chance to generate on rows 1 and 8

Pieces are then assigned their respective positions. As mentioned previously, row placement is determined using a weighted probability function however column placement is assigned in a random manner (squares 1 through 7).

Legality verification

Due to the somewhat random nature of the board generation, positions may be illegal. To make sure all generated positions are legal an internal check is performed on the board. This does three things: it checks if either King is in check, verifies no Kings are next to each other and also checks that Bishops of the same colour are on opposite-coloured squares. If either one of these checks fail on a candidate board it is discarded and a new one is created. The process repeats until a generation satisfies this constraint.

Efficiency evaluation

Once a generated position has passed its legality check it must also pass an efficiency check, used to determine if the generation is puzzle-worthy. We define an efficiency threshold as the number of moves to checkmate which requires any generation must be a smaller number of moves than (or equal to) it. For example, a threshold of 5 means for generated positions there must exist a checkmate in a maximum of 5 moves. The user has the option to choose between 3 different efficiency thresholds, called "difficulties": "Easy" (mate in <= 3 moves), "Medium" (mate in <= 5 moves) or "Hard" (mate in <= 10 moves).

The minimum number of moves to checkmate is determined by inputting the board position[1] into a strong open-source chess engine called Stockfish[2]. This can analyse the current position and calculate the fastest checkmate. We compare this minimum number of moves against the efficiency threshold and if it's lower than (or equal to) it then this generation becomes the puzzle! Once the program is initiated many different puzzles can be created, and the difficulty can be adjusted as necessary. FEN notation can include information about various game states such as whether or not either side can castle and how many moves have been made but to simplify here we do not allow any side to castle and also set the move counts to 0.

4 Results

The resulting generations from testing with the system were generally positive. A fair amount of trial-and-error was required to determine the parameters which yielded the best results (probabilities for rows and pieces). After a few iterations of the testing process suitable values for these parameters were settled upon. Due to the way suitable generations are produced, it can take anywhere from almost instantly to 10s of seconds to generate puzzles, the main reason for this being down to the search that is needed using the Stockfish engine. While ideally puzzle generation should be fast, it shouldn't be too much of a problem as it's unlikely we would need to deploy this in any kind of real-time application.

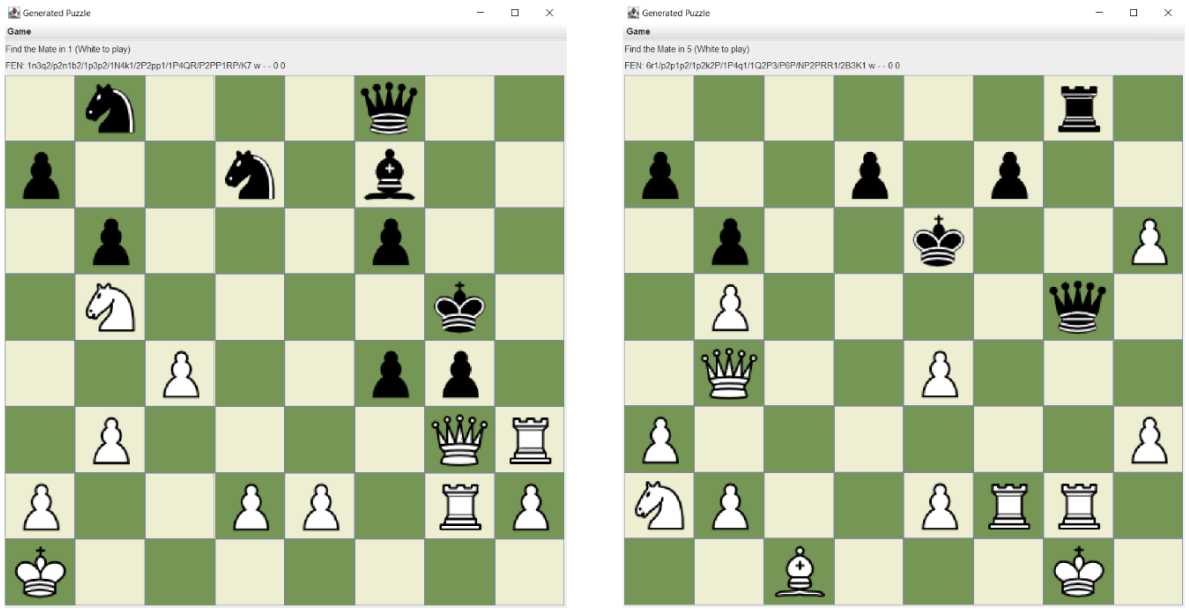

Figure 3: Example generations for different difficulties. Left: "Easy" (mate in <= 3 moves), Right: "Medium" (mate in <= 5 moves).

[1]Positions are input using a standard notation called the Forsyth-Edwards Notation (FEN) so that they can be read and understood by engines.

[2]Stockfish is an open-source Chess engine developed by multiple authors. It is used in this project only for calculation of fastest checkmates. Source code can be found at: https://github.com/official-stockfish/Stockfish

Figure 4: Example generation for difficulty "Hard" (mate in <= 10 moves).

Users can use the 'Game' menu to generate new puzzles, change the difficulty of the generated puzzles and can copy the puzzle FEN to the clipboard for input into Chess engines such as Stockfish for detailed analysis, as shown in Figure 5.

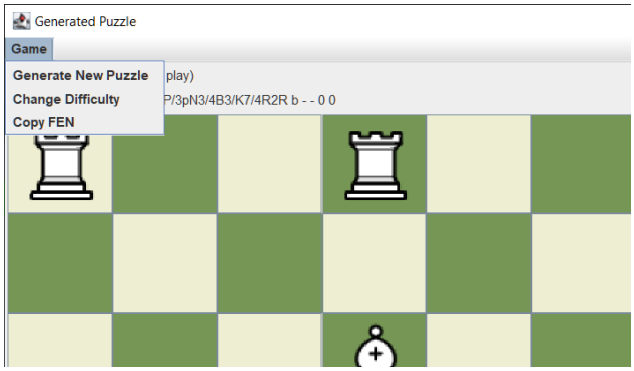

Figure 5: Users can choose to generate a new puzzle (with the current difficulty), change the puzzle difficulty or copy the puzzle FEN to the clipboard.

5 Evaluation

The nature of this work lends itself well to both a subjective and objective evaluation of its overall creativity. The best way to evaluate the generations is to produce many different examples and observe their overall quality in terms of how novel and valuable they are. To have a concrete framework to compare against I will utilise Colton's Tripod [5], comparing my system against the three explicit qualities of *skill*, *appreciation* and *imagination*.

Skill

Skill is described as "the ability to show that the system understands and has some mastery of the area it is working in", essentially saying its outputs are deliberately and methodically crafted to progress towards a good solution. In terms of my system, I think there are some indicators that this is the case. For example, the guidance of generations (i.e. placement probabilities) and the quantitative engine evaluation indicates to the system which generations are good and which ones are not. Despite this, it could be said to be lacking in an overarching direction or aim as most of the artefacts it is crafting are done in a relatively stochastic manner. It just so happens to *sometimes* produce generations that are valuable.

Appreciation

Appreciation is "the ability for a system to critically assess the value of what is produces". In some regards it could be said that my system fulfils this criteria to an extent, as generations are analysed for their legality and efficiency, however in the grand scheme of creativity as compared to how a human might evaluate their work this kind of low-level evaluation I believe is not enough to satisfy the criteria of adequate appreciation. This is not to say that there is lost potential here — in fact far from it. There are many ways the system can be improved to where it has the ability to critically appraise itself, such as by comparing to real Chess games in a database for example.

Imagination

Imagination is "the potential for the system to be innovative in how it creates its objects". This leg of the Tripod encapsulates one of the key principles in creativity: novelty. In my opinion, through having a fair deal of knowledge on the subject of Chess as a whole, the generated positions from my system are certainly surprising. There is an interesting fact of Chess (and surely other similar games) that despite there being an immensely large space for legal positions [6], the grand majority of games will never actually explore them, just because there are so many and because typical Chess games play out in a similar fashion. An argument could therefore be made against a heavily-random system such as this in favour of a more guided, database-driven one for the sole reason that in real games these positions just would not occur and they therefore would hold little use as a tool for learning. A devil's advocate stance would be that even though these positions may not actually appear in real games, this doesn't matter as the principles are still the same. With the adoption of new ideas into heavily-established fields being met with so much obstinacy, I believe it's the novel take that is the most beneficial.

Finally, I believe the results obtained overall were very novel and somewhat valuable. I say very novel because the generations are very unique in comparison to other similar systems and I say somewhat valuable because, while I believe they can have some use, there are definitely areas where the generations are lacking, in particular to do with how there is little self-evaluation. It is troubling that it could be described as "mere generation", but at the same time the wide potential for enhancements quells some of these qualms.

6 Conclusions and future work

In conclusion, I believe the solution creates interesting and unique puzzles. Whether there is much creativity pertaining to what's happening "under the hood" is up for debate however I believe the generations would certainly appear intriguing to an outside observer.

In terms of future expansion, I believe there is ample room for further contributions to this idea. One example of an improvement would be to facilitate the generation of puzzles that don't just provide checkmating patterns but also ones that result in a material advantage. Doing this would allow for more variety and more interesting puzzles to be generated. Evaluating for a material advantage was attempted for this work but was abandoned because it was difficult to get working well, therefore naturally it follows this should be the next implementation idea were the project to be continued. Another potential improvement could be to utilise a machine learning approach to generate the board positions. For example, a neural network with an additional efficiency evaluation function could be used which would train on existing puzzles/game states and can learn to produce its own, unique puzzles. It may also be beneficial to utilise database such as done by Chess.com for an enhanced realism compared to the puzzles generated for this paper.

Overall, I believe this method works well for generating "creative" positions where other methods may not. Despite this, for long-term learning it may be more beneficial to have something more concrete in terms of what it generates, so that users don't keep asking themselves *"but would this ever happen in a real Chess game?"*. Perhaps, therefore, a hybrid approach is the most reasonable for this sort of task.

References

[1] Leon Watson. Chess.com hits 1 billion games played in february. https://www.chess.com/news/view/chess-boom-1-billion-games-played-in-february.

[2] Chess.com. How we built a puzzle database with half a million puzzles. https://www.chess.com/blog/CHESScom/how-we-built-a-puzzle-database-with-half-a-million-puzzles.

[3] Barbara De Kegel and Mads Haahr. Procedural puzzle generation: A survey. *IEEE Transactions on Games*, 12(1):21–40, 2019.

[4] Alec Thomson. *A system for procedurally generating puzzles for games*. PhD thesis, Massachusetts Institute of Technology, 2013.

[5] Simon Colton. Creativity versus the perception of creativity in computational systems. In *AAAI spring symposium: creative intelligent systems*, volume 8, page 7. Palo Alto, CA, 2008.

[6] Claude E Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.