COSC2780: Intelligent Decision Making
Course Project Draft
Modelling Chess

Team ABP*
Axel Thor Ahmer (s3855518)
Bharti Sanjeebkumar Sinha (s3861921)
Kee Chew Huang (s3808292)

April 13, 2022

1 Overview and Literature

We plan to model the game of chess to allow for arbitrary goal planning, generating novel checkmate patterns, and concretely evaluating endgame positions. Constraint logic programming (CLP) and answer set programming (ASP) will be investigated. Chess has been researched and implemented in Prolog previously [1, 2]. Two famous chess puzzles (the n-queen problem, and the knight's tour problem) have been solved using ASP [3], however both these puzzles were relatively simple - only involving a single piece type from one player. Analysing results of the n-queens problem will help set the initial understanding of how chess problems have been approached [4]. An overview of how ASP can be applied to problems in different systems and applications will shape our perspectives and approach also [5].

We are planning to explore more complex combinatorial problems than the aforementioned, involving many *different piece-types*, and planning *with and without an opposition* that can make moves. Chess positions are often represented conveniently in 'Forsyth–Edwards notation (FEN)[6], which we plan to leverage to graphically display our software and solutions with various chess websites and chess GUIs.

We hope that our software can help chess players interactively find attacking ideas in a style different to conventional chess engines by planning with individual pieces and reverse engineering goals to obtain desired positions.

ABP stands for Axel - Bharti - Peter. However, it is also a hilarious wordplay on *ASP* - Answer Set Programming.

1.1 Topic Ideas

1. **Goal Planning**

   A checkmating example: given some board size (e.g. $4 \times 4, 5 \times 5, 6 \times 6$) and the position of chess pieces on that board, determine all paths that a piece can take in order to achieve a checkmate on the opposition's king.

   Other goal's could be: reaching a desired square, capturing a specific piece type, or constructing some type of piece structure.

2. **Generating 'Checkmate in N' Positions**

   Given some arrangement of opposition pieces, generate solutions where a defined set of white pieces need to be placed in order to allow for mate in N moves. The set of pieces to be placed can be constrained by pieve type, number of pieces, or some maximum human-value. Checkmate in 1 move will be easiest to implement as the opposition has no influence on the game. If the complexity of adding opposition defenive moves is too complex they could be ignored. Ignoring special rules such as en passant[1] and castling[2] may also significantly reduce the complexity of this problem.

3. **Evaluate Endgame Positions**

   Given an endgame situation (determinable by a tablebase[3]), use ASP to compute whether a game is winnable assuming it was played perfectly. The constraints and difficulties laying ahead could possibly be the exponential nature of the chess game-tree. Apart from that, modelling the problem in a dynamic way where both players taking turns rather than a static problem similar to N-queens problem would be a tricky part to get over.

2 Learning Objectives

1. Using a declarative programming language, express a model for the game of chess.

2. Integrate our solution with existing chess websites and resources to validate our software and represent solutions graphically.

3. Identify how CLP and ASP can be applied to uncover knowledge in chess and their limitations.

4. Become more familiar with the Prolog CLP(FD) library and ASP concepts.

5. Explore how different constraints effect the computation of solutions:

   - Varying chessboard dimensions.

[1] `https://en.wikipedia.org/wiki/En_passant`
[2] `https://en.wikipedia.org/wiki/Castling`
[3] `https://www.chessprogramming.org/EndgameTablebases`

- Constraining the number of pieces.
- Constraining the type of pieces.
- Constraining the maximum human-value of pieces.
- Using new, alternate chess pieces.

6. Build a better understanding of current chess programming concepts such as: game-tree exploration, min-max search, alpha-beta pruning, conventional chess engines, and how chess tablebases are generated.

7. Become better chess players!

3 Schedule

3.1 Milestones

1. Define the board squares.

2. Define piece movement for the six chess pieces: King, Queen, Bishop, Knight, Rook, Pawn.

3. Define the concept of a piece existing on a board square

4. Decide on any rule simplifications to include to reduce the project scope, then define any desired special movement rules, e.g.: disallow a king to move into an attacked square; disallowed any piece to move to a square such that it would cause it's own king to be attacked; en passant rule (hard); and castling (hard).

5. Define the concept of checkmate:

   - If one piece is attacking the enemy king: the piece cannot be taken, be blocked, and the king cannot move to safety.
   - If two piece's are attacking the king: the king cannot move, and one piece moved so as to unveil the attack from the other piece.
   - Note: three or more pieces cannot simultaneously attack a king - it is impossible by the rules.

6. Define the concept of time, i.e. a single player making consecutive moves on a single board to some goal.

7. Define the concept of time as it pertains to two players.

8. Apply the game model to investigate our three topic ideas.

3.2 Timeline

- **Week 7:** Gather ideas and concepts about the game of chess and how it can be modelled using logical programming. Divide work between team-members. Work on milestones.
- **Week 8:** Construct minimal viable solution towards basic modelling problems - board and piece definitions. Run small evaluations to check the basic logic.
- **Week 9:** Use the game model to generate solutions to simple single-player combinatorial problems, and investigate how constraining the game effects solution generation.
- **Week 10:** Use the game model to generate solutions to complex two-player combinatorial problems, and investigate how constraining the game effects solution generation.
- **Week 11:** Determine which problems should be mentioned in the report, and run experiments accordingly to generate answers. Generate graphics and begin writing report.
- **Week 12:** Generate graphics and finalize report.

References

[1] Tuhola H. *Modeling of Chess in Prolog*. `https://boxbase.org/entries/2018/nov/19/modeling-chess-in-prolog/`. Accessed: 2022–04-12. 2018.

[2] Barney Pell. "Metagame in symmetric chess-like games". In: (1992).

[3] Edeilson Milhomem da Silva Carvallho et al. "Two Classic Chess Problems Solved by Answer Set Programming". In: ().

[4] Jordan Bell and Brett Stevens. "A survey of known results and research areas for n-queens". In: *Discrete mathematics* 309.1 (2009), pp. 1–31.

[5] Enrico Pontelli. *A 25-Year Perspective on Logic Programming*. Springer, 2010.

[6] *Forsyth-Edwards Notation - Chessprogramming wiki*. `https://www.chessprogramming.org/Forsyth-Edwards_Notation`. [Online; accessed 2022-04-05]. 2022.