Finding Checkmate in N moves in Chess using Backtracking and Depth Limited Search Algorithm

Moch. Nafkhan Alzamzami 13518132
Informatics Engineering
School of Electrical Engineering and Informatics
Institute Technology of Bandung, Jalan Ganesha 10 Bandung
nafkhanalzamzami@gmail.com 13518132@std.stei.itb.ac.id

***Abstract*—There have been many checkmate potentials that players seem to miss. Players have been training for finding checkmates through chess checkmate puzzles. Such problems can be solved using a simple depth-limited search and backtracking algorithm with legal moves as the searching paths.**

Keywords—chess; checkmate; depth-limited search; backtracking; graph;

I. Introduction

There have been many checkmate potentials that players seem to miss. Players have been training for finding checkmates through chess checkmate puzzles. Such problems can be solved using a simple depth-limited search and backtracking algorithm with legal moves as the searching paths. The use of backtracking is if we found a move that can make the opponent escape from the checkmate in the given amount of limited moves, then we can scrap that move and try the other possible moves.

This algorithm can be used for creating and confirming the checkmate in N moves. This will make sure that the opponent cannot escape from the checkmate and so the puzzle is solvable with the moves that have to be found in N moves.

II. Base Theory

A. Chess

Chess is a board game for two players.[1] It is played on a square board, made of 64 smaller squares, with eight squares on each side. Each player starts with sixteen pieces: eight pawns, two knights, two bishops, two rooks, one queen and one king.[2] The goal of the game is for each player to try and checkmate the king of the opponent. Checkmate is a threat ('check') to the opposing king which no move can stop. It ends the game.[3][4]

During the game the two opponents take turns to move one of their pieces to a different square of the board. One player ('White') has pieces of a light color; the other player ('Black') has pieces of a dark color. There are rules about how pieces move, and about taking the opponent's pieces off the board. The player with white pieces always makes the first move.[4] Because of this, White has a small advantage, and wins more often than Black in tournament games.[5][6]

Chess is played on a square board divided into eight rows of squares called ranks and eight columns called files, with a dark square in each player's lower left corner.[8] This is altogether 64 squares. The colors of the squares are laid out in a checker (chequer) pattern in light and dark squares. To make speaking and writing about chess easy, each square has a name. Each rank has a number from 1 to 8, and each file a letter from a to h. This means that every square on the board has its own label, such as g1, f5 or b3. The pieces are in white and black sets. The players are called White and Black, and at the start of a game each player has 16 pieces. The 16 pieces are one king, one queen, two rooks, two bishops, two knights and eight pawns.[4] In this game out can get up to a quadruple pawn, king, knight, queen, and also the king although it is very rare.

The movement rules of chess are defined below:

1. The knight is the only piece that can jump over another piece.
2. No piece may move to a square occupied by a piece of the same color.
3. All pieces capture the same way they move, except pawns.
4. The king's move is one square in any direction. The king (K for short) may not move to any square where it is threatened by an opposing piece. However, the king can move to a square that is occupied by an opponent's piece and capture the piece, taking it off the board.

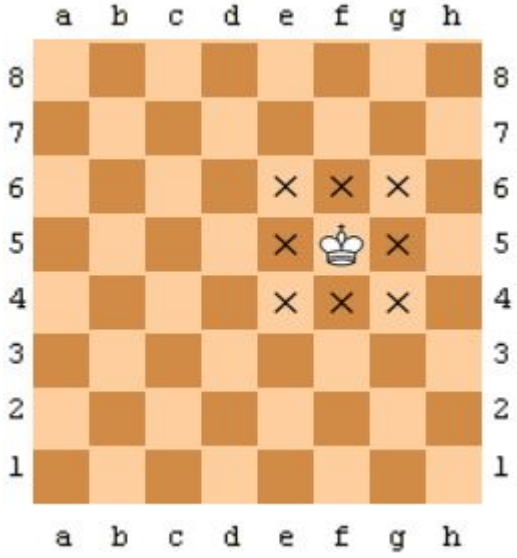

Fig. 1. Moves of the king
(source: https://simple.wikipedia.org/wiki/Chess)

5. The queen (Q) can move any distance in any direction on the ranks, files and diagonals.

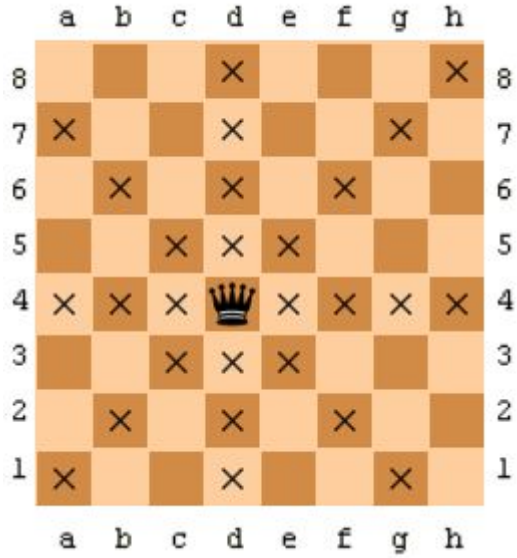

Fig. 2. Moves of the queen
(source: https://simple.wikipedia.org/wiki/Chess)

6. The rooks (R) move any distance on the ranks or files.[4]

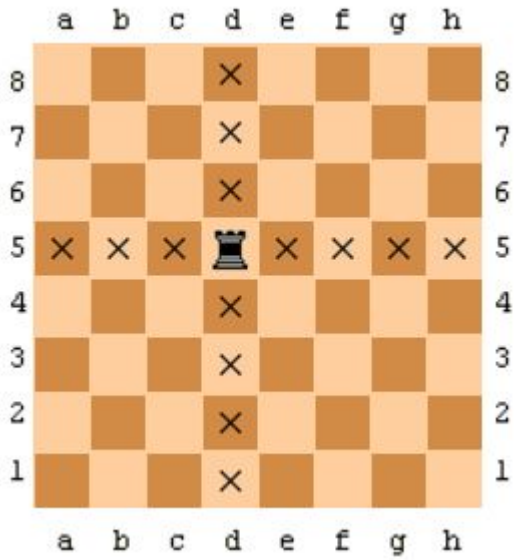

Fig. 3. Moves of the rook
(source: https://simple.wikipedia.org/wiki/Chess)

7. The bishops (B) move diagonally on the board. Since a bishop can only move diagonally, it will always be on the same color square.[7]

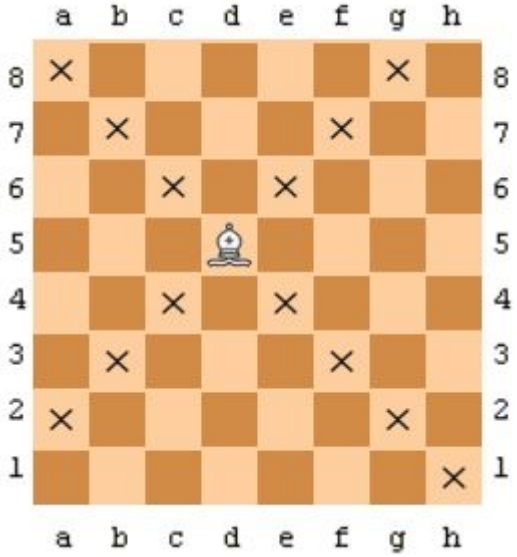

Fig. 4. Moves of the bishop
(source: https://simple.wikipedia.org/wiki/Chess)

8. The knights (Kt or N) move in an "L" shape. Each move must be either two squares along a rank and one square along a file, or two squares along a file and one square along a rank. It is the only piece that can jump over other pieces. Like the other pieces, it captures an opposing piece by landing on its square.

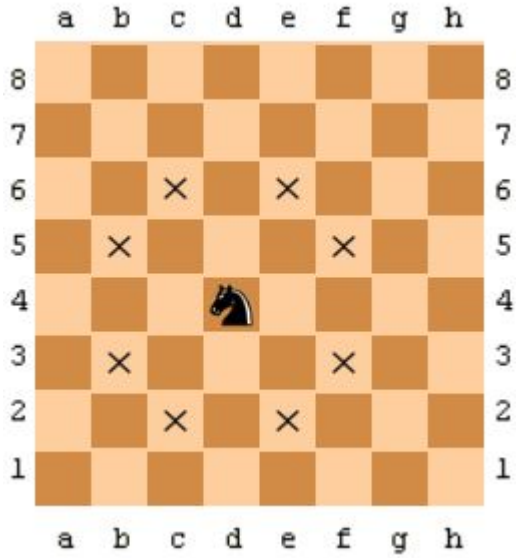

Fig. 5. Moves of the knight
(source: https://simple.wikipedia.org/wiki/Chess)

9. The pawns can only move up the board. On its first move a pawn may move either one or two squares forward. A pawn captures one square diagonally, not as it moves: see white circles on its diagram. Besides, in some situations pawns can capture opponent's pawns in a special way called en passant, which means in passing in French (see below).[4]

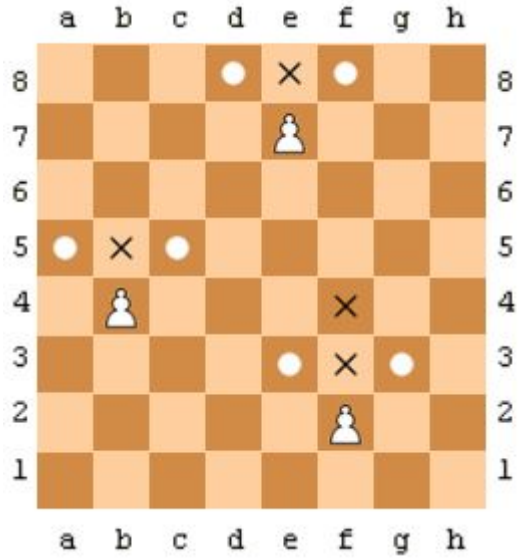

Fig. 6. Moves of the pawn
(source: https://simple.wikipedia.org/wiki/Chess)

10. Once in every game, each king can make a special move, known as castling. When the king castles, it moves two squares to the left or right. When this happens, the rook is moved to stand on the opposite side of the King.[9] Castling is only allowed if all of these rules are kept:[10]
    a. Neither piece doing the castling may have been moved during the game.
    b. There must be no pieces between the king and the rook.
    c. The king may not be currently in check, nor may the king pass through any square attacked by the opponent. As with any move, castling is not allowed if it would place the king in check.[4]
11. En passant ('in passing' in French) is a special capture. It is only available when a pawn moves forward two squares past an opposing pawn on an adjacent file. The opposing pawn must be on the 5th rank from its own side. Then the opponent's pawn can capture the double-mover as if it had only moved one square forward. This option is open on the next move only.[4] For example, if the black pawn has just moved up two squares from g7 to g5, then the white pawn on f5 can take it by en passant on g6. The en passant rule was developed when pawns were allowed to make their double move. The rule made it more difficult for players to avoid pawn exchanges and blockade the position. It kept the game more open.

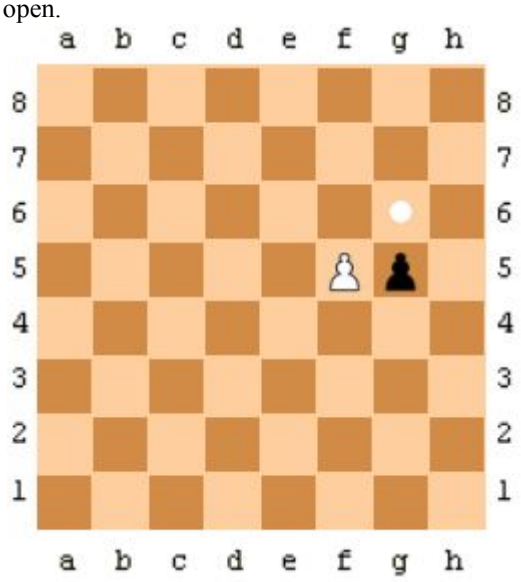

Fig. 7. En passant
(source: https://simple.wikipedia.org/wiki/Chess)

12. When a pawn moves to its eighth rank, it must be changed for a piece: a queen, rook, bishop, or knight of the same color (player's choice).[11] Normally, the pawn is queened, but in some advantageous cases another piece is chosen, called 'under-promotion'.[4]

B. Backtracking

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).[12]

C. *Depth-Limited Search*

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.[13]

Depth-limited search can be terminated with two Conditions of failure:

1. Standard failure value: It indicates that the problem does not have any solution.
2. Cutoff failure value: It defines no solution for the problem within a given depth limit.

III. Implementation

In the implementation of the algorithm, the algorithm receives a chess board state and a number of limited moves as a puzzle to solve. The algorithm is then searching for the moves that leads to a checkmate. The algorithm as pseudocode can be seen as below:

```
function getAnswerMove(numberOfMoves: integer) do
    foreach (move in all legal moves as answer move
candidates) do
        execute move to the board
        answerFound <-
isOpponentCannotEscape(numberOfMoves)
        undo the move
        if (answerFound) do
            return move
        end
    end
    // Move not found. Opponent's king can escape
    return null
end

function isOpponentCannotEscape(numberOfMoves:
integer) do
    if (no legal move available) do
        // Either stalemate or checkmate, return
true if checkmate, false if stalemate
        return is opponent move attacked
    end else if (numberOfMoves = 1) do
        // Opponent can escape in the given number
of moves
        return false
    end
    foreach (move in all legal moves from opponent
pieces) do
        execute move to the board
        answerMove <- getAnswerMove(numberOfMoves -
1)
        undo the move
        if (answerMove = null) do
            return false
        end
    end
    // Passed all possible moves, opponent cannot
escape.
    return true
end
```

The search searches each possible move as a candidate answer move and tests all the possible moves by the opponent after each candidate answer move. The search goes as a depth-limited search with the limit is defined from the number of moves needed to do the checkmate. The number of moves is occured as the answer moves count only, not including the opponent's moves.

The search will be backtracked when the candidate answer can create opponent moves that make it not possible to checkmate in the number of moves to checkmate. Therefore, the candidate answer can not be the answer and has to be cut off from the search.

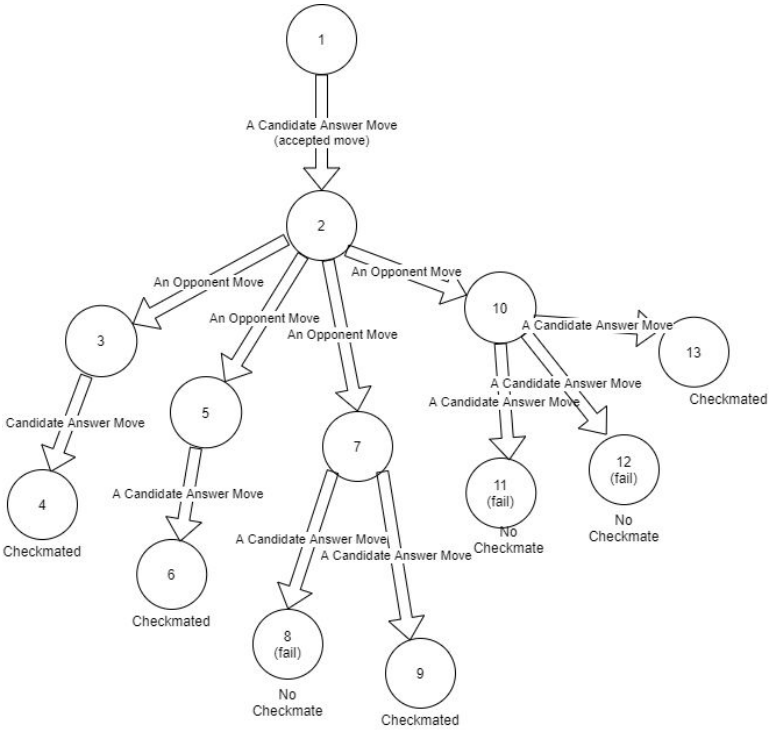

Fig. 8. Example of an accepted candidate move in a visualised search tree

A candidate answer move can be proven to be the answer move if both of the conditions below are fulfilled:

1. For every possible move the opponents make, there is at least one answer move that can lead to a checkmate.
2. Opponent is checkmated.

For example for the number of moves is 2, as shown on figure 8, a candidate answer move is selected on node 2 and is then checked for every possible opponent counter move, those are node 3, 5, 7, and 10. Then, every opponent move is then checked if there is at least one answer move exists. Node 3 has an answer move node 4, node 5 has an answer move node 6, node 7 has an answer move node 9, and node 10 has an answer move node 13. So, after every opponent's counter move for candidate answer move node 2 is checked that they all have at least one answer move, candidate answer move on node 2 is proven that it is an answer move.

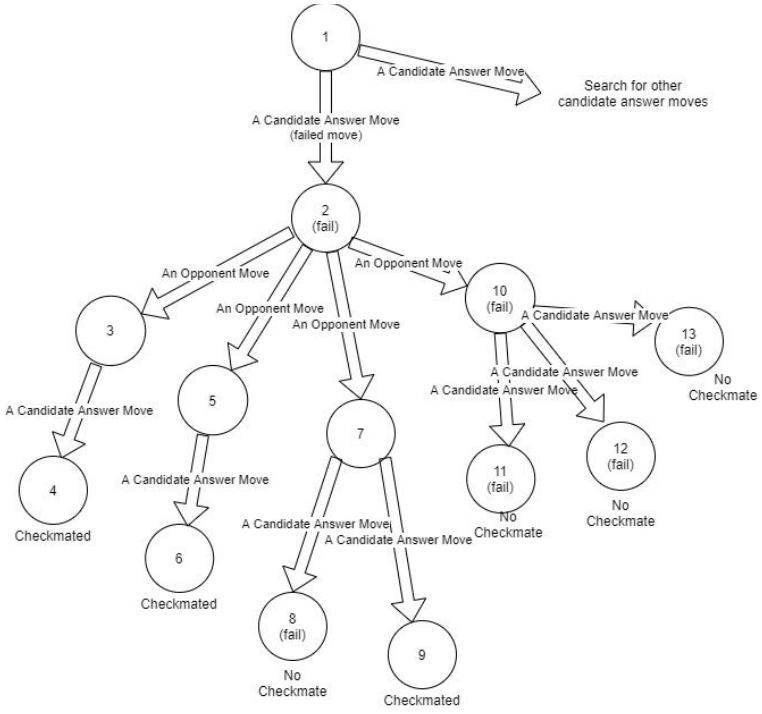

Fig. 9. Example of a failed candidate move in a visualised search tree

Example of a failed candidate answer move is shown on figure 9. Like before, a candidate answer move is selected on node 2 and is then checked for every possible opponent counter move, those are node 3, 5, 7, and 10. Then, every opponent move is then checked if there is at least one answer move exists. Node 3 has an answer move node 4, node 5 has an answer move node 6, node 7 has an answer move node 9, but node 10 doesn't have any move that leads to a checkmate. That means the selected candidate answer move is failed to checkmate the opponent since the opponent can escape from a checkmate through certain moves. The algorithm then has to find another candidate answer move and check if it is a valid answer move. If every possible candidate answer move is not valid, then it is not possible to checkmate the opponent on the given chess state and number of moves.

IV. Use Cases

For testing the chess puzzle, I implemented the algorithm using Java programming language version 13.0.1 and a java chess library by bhlangonijr called chesslib. The implementation of the algorithm in java source code is shown below:

con.nafkhanzam.checkmatefinder.Answer

```
package com.nafkhanzam.checkmatefinder;

import java.util.HashMap;
import java.util.Map;

import
com.github.bhlangonijr.chesslib.move.Move;

public class Answer {
    private Move answerMove;
    private Map<Move, Answer> nextAnswer;

    public Answer() {
        nextAnswer = new HashMap<>();
    }

    public Move getAnswerMove() {
        return this.answerMove;
    }

    public void setAnswerMove(Move move) {
        this.answerMove = move;
    }

    public void setNextAnswer(Map<Move, Answer>
nextAnswer) {
        this.nextAnswer = nextAnswer;
    }

    public Map<Move, Answer> getAnswers() {
        return nextAnswer;
    }

    public void putAnswer(Move move, Answer
answer) {
        nextAnswer.put(move, answer);
    }

    public Answer getNextAnswer(Move move) {
```

```
        return nextAnswer.get(move);
    }

    public boolean end() {
        return nextAnswer.isEmpty();
    }

}
```

con.nafkhanzam.checkmatefinder.CheckmateFinder

```
package com.nafkhanzam.checkmatefinder;

import com.github.bhlangonijr.chesslib.Board;
import
com.github.bhlangonijr.chesslib.move.Move;
import
com.github.bhlangonijr.chesslib.move.MoveGenerat
or;
import
com.github.bhlangonijr.chesslib.move.MoveGenerat
orException;
import
com.github.bhlangonijr.chesslib.move.MoveList;

public class CheckmateFinder {
    private Board board;

    public CheckmateFinder(Board board) {
        this.board = board;
    }

    public Answer findAnswer(int depth) throws
MoveGeneratorException {
        Answer answer = new Answer();
        _answerMove(answer, depth);
        return answer;
    }

    private boolean _answerMove(Answer answer,
int depth) throws MoveGeneratorException {
        for (Move move :
MoveGenerator.generateLegalMoves(board)) {
            board.doMove(move);
            boolean found =
_opponentMove(answer, depth);
            board.undoMove();
            if (found) {
                answer.setAnswerMove(move);
                return true;
            }
        }
        return false;
    }

    private boolean _opponentMove(Answer answer,
int depth) throws MoveGeneratorException {
        MoveList availableMoves =
MoveGenerator.generateLegalMoves(board);
        if (availableMoves.size() == 0) {
            return board.isKingAttacked();
        } else if (depth == 1) {
            return false;
        }
        for (Move move : availableMoves) {
            board.doMove(move);
            Answer next = new Answer();
            boolean found = _answerMove(next,
depth - 1);
            board.undoMove();
            if (!found) {
                return false;
            }
            answer.putAnswer(move, next);
        }
        return true;
    }

}
```

And for testing the puzzle, I created a simple main console app program for input and output the moves. The program receives a number of moves needed to search and the chess board state in Forsyth–Edwards Notation.

I tested the puzzles from a user named TripleXDooM on chess.com forum. I downloaded the board state as a Forsyth–Edwards Notation.

1. Test-Case 1

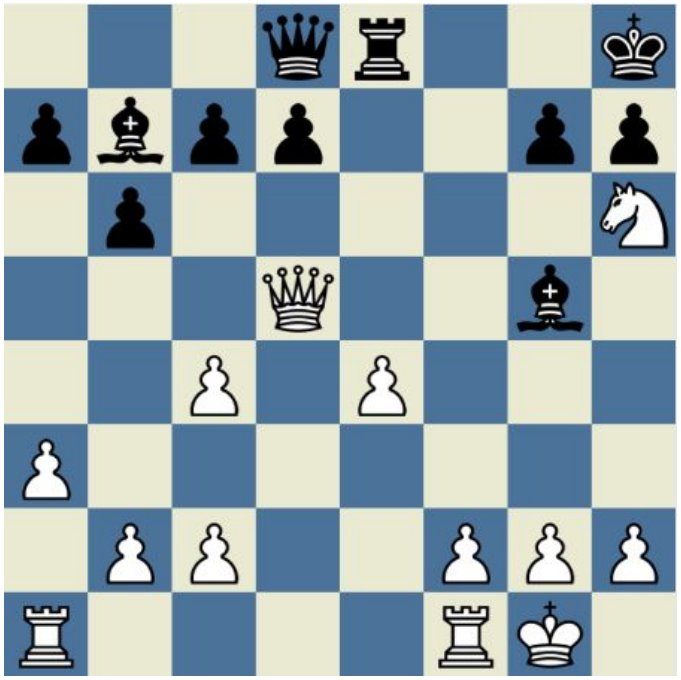

Fig. 10. A mate in 2 chess puzzle test-case 1 the starting state.
(source:
https://www.chess.com/forum/view/more-puzzles/300-checkmate-puzzles-puzzles-1---50)

In test-case 1, the Forsyth–Edwards Notation is as below:

```
3qr2k/pbpp2pp/1p5N/3Q2b1/2P1P3/P7/1PP2PPP/R4RK1 w - - 0 1
```

The input and output according to the opponent's move on chess.com is as below:

```
Number of moves: 2
Chess board state in FEN: 3qr2k/pbpp2pp/1p5N/3Q2b1/2P1P3/P7/1PP2PPP/R4RK1 w - - 0 1
Answer found in 129ms.
Move d5g8!
Opponent's move: e8g8
Move h6f7 and checkmate!
```

Fig. 11. Input/output for test-case 1.

The final move resulted in a checkmate as shown below:

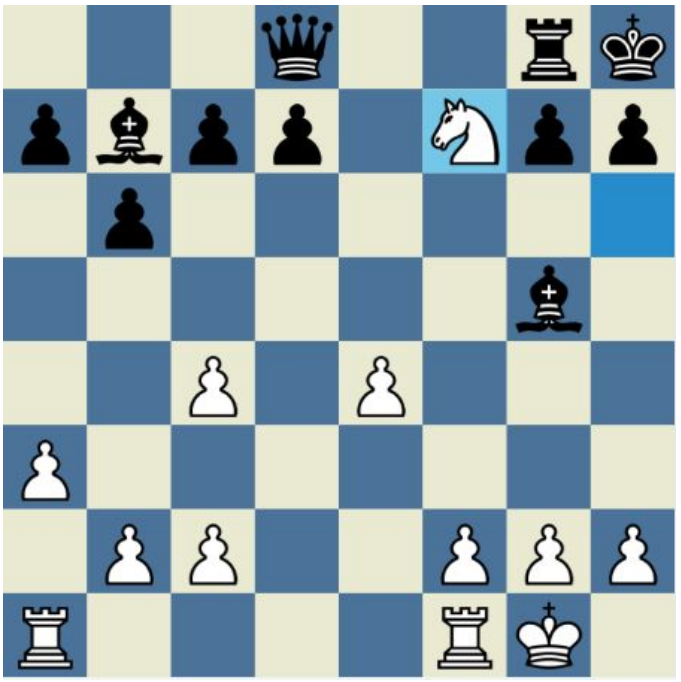

Fig. 12. A mate in 2 chess puzzle test-case 1 the final state.
(source:
https://www.chess.com/forum/view/more-puzzles/300-checkmate-puzzles-puzzles-1---50)

2. Test-Case 2

Fig. 13. A mate in 2 chess puzzle test-case 2 the starting state.
(source:
https://www.chess.com/forum/view/more-puzzles/300-checkmate-puzzles-puzzles-1---50)

In test-case 2, the Forsyth–Edwards Notation is as below:

r1bq2k1/ppp2r1p/2np1pNQ/2bNpp2/2B1P3/3P4/PPP2PPP/R3K2R w KQ - 0 1

The input and output according to opponent's move on chess.com is as below:

```
Number of moves: 2
Chess board state in FEN: r1bq2k1/ppp2r1p/2np1pNQ/2bNpp2/2B1P3/3P4/PPP2PPP/R3K2R w KQ - 0 1
Answer found in 105ms.
Move d5f6!
Opponent's move: d8f6
Move h6f8 and checkmate!
```

Fig. 14. Input/output for test-case 2.

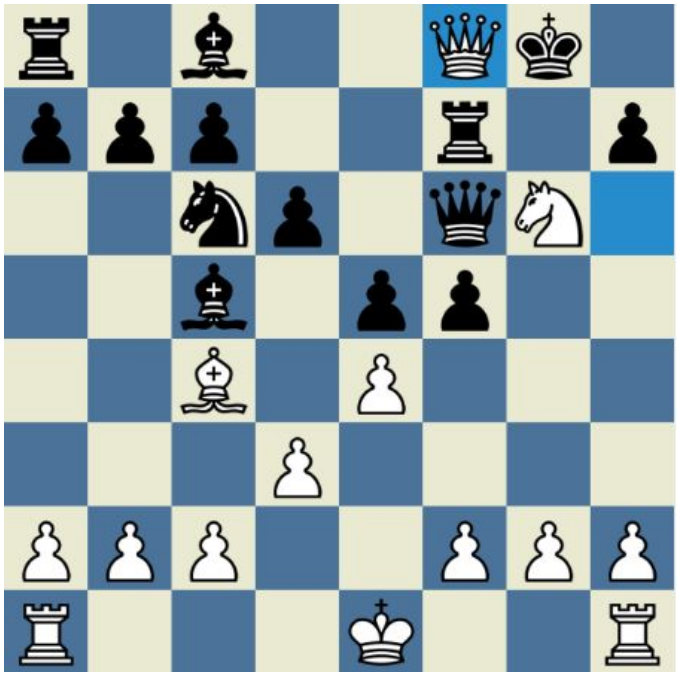

Fig. 15. A mate in 2 chess puzzle test-case 2 the final state .
(source: https://www.chess.com/forum/view/more-puzzles/300-checkmate-puzzles-puzzles-1---50)

V. Conclusion

Chess is a game that needs a lot of depth in thinking on which move is not a blunder or inaccurate. With the great evolution of technology, machines can defeat humans in the decision of making a move in chess games. Finding a possible checkmate in a fast amount of time in a complex chess game state is something a machine can do with correct algorithms. Backtracking and depth-limited search is one of the alternatives algorithms.

Video Link at Youtube

https://youtu.be/Q7R9EY7lpFs

Acknowledgment

I would like to thank Mr. Rinaldi Munir for their guidance in teaching algorithm strategies course in his lecture so that I'm able to understand many new algorithms including backtracking and depth-limited search.

References

[1] Abate, Frank R. (ed) 1997. The Oxford desk dictionary and thesaurus. ISBN 0-19-511214-8

[2] Costello, Robert E. et al. (eds) 2001. Macmillan dictionary for children. Simon & Schuster, New York. ISBN 0-689-84323-2Costello, Robert E. et al. (eds) 2001. Macmillan dictionary for children. Simon & Schuster, New York. ISBN 0-689-84323-2

[3] Paton, John et al. (eds)1992. The Kingfisher children's encyclopedia. Kingfisher Books, New York. ISBN 1-85697-800-1

[4] "Laws of Chess". FIDE. Retrieved 2008-11-26.

[5] Chessgames "Chess Opening Explorer". Chessgames.com. Retrieved 2010-05-25.

[6] Rowson, Jonathan (2005). Chess for Zebras: thinking differently about black and white. Gambit Publications. p. 193. ISBN 1-901983-85-4.

[7] "Chess Moves - How chess pieces move - chess piece movements". gamesinfodepot.com. Retrieved 1 April 2010.

[8] "Chess [[wikt:basic|Basics]]". chesslab.com. Retrieved 1 April 2010.

[9] "Castling, by Chess Corner". chesscorner.com. chess corner. Retrieved 1 April 2010.

[10] Reuben, Stewart 2005. The chess organiser's handbook. 3rd ed, incorporating the FIDE Laws of Chess. Harding Simpole, Devon.

[11] Robert Harrison. "Chess tips: How to promote a pawn". helium.com. Retrieved 25 May 2010.

[12] https://www.geeksforgeeks.org/backtracking-introduction/

[13] https://www.javatpoint.com/ai-uninformed-search-algorithms

STATEMENT

With this, I hereby state that this paper is my own writing, not a translation and not a plagiarization of someone else.

Bandung, 3rd May 2020

Moch. Nafkhan Alzamzami 13518132