LOCI: A CHESS ENGINE FOR GENERATING GAME DATA

BY

ADEFOKUN AHIRA JUSTICE

185576

SUPERVISOR: MR. OGUNTUNDE

A REPORT SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF IBADAN, IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF A BACHELOR OF SCIENCE DEGREE IN COMPUTER SCIENCE

MARCH, 2019

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND OF THE STUDY

Games have been one of the most visible areas of progress in the AI space in the last few years. Chess, Jeopardy, GO and, very recently, Poker are some of the games that have been mastered by AI systems using breakthrough technologies. From that viewpoint, the success of AI seems to be really tied to the progress on game theory (Rodriguez, 2017).

While games are, obviously, the most visible materialization of game theory, it is far from being the only space on which those concepts are applied. From that perspective, there are many other areas that can be influenced by the combination of game theory and AI. The fact is that most scenarios that involve multiple "participants", collaborating or competing to accomplish a task, can be gamified and improved using AI techniques. Even though the previous statement is a generalization, I hope it conveys the point that game theory and AI is a way to think and model software systems rather than a specific technique.

Chess is a game that requires much creativity and sophisticated reasoning that it was once thought of as something no computers will ever be able to do. (Lai, 2015). It was frequently listed alongside activities like poetry writing and painting, as examples of tasks that can only be performed by humans. While writing poetry has remained very difficult for computers to this day, we have had much more success building chess-playing computers. (Lai, 2015)

In 1997, IBM's Deep Blue defeated the reigning World Chess Champion, Garry Kasparov, under standard tournament rules, for the first time in the history of chess (Lai, 2015). In the ensuing two decades, both computer hardware and AI research advanced the state-of-art chess-playing computers to the point where even the best humans today have no realistic chance of defeating a modern chess engine running on a smartphone. (Lai, 2015)

1.2 PROBLEM STATEMENT

Training data for deep learning chess engines is rare to come by. Some of these datasets are collated manually from tournaments played by humans and artificial intelligence (AI). Also, ready tools for the comparison of different evaluation function metrics are not readily available.

1.3 AIM AND OBJECTIVES

The aim of this project is to build a chess engine (**loci)** for generating game data.

The proposed system has the following objectives:

1. To design and model a chess engine that enables **AI to AI** gaming.
2. To implement a chess engine model that provides large chess game data.
3. To implement a chess engine model that allows for changes to engine evaluation functions.

1.4 METHODOLOGY

- System requirement analysis was done with review of literature on chess engines and an examination of the manual existing system.
- Design was done using use case diagram, activity diagram, and data flow diagram.
- Implementation was done with Python, a programming language, and with plain text data in portable game notation (PGN) as the data output format.
- Deployment and Testing was done on a desktop computer with the command line as its interface.

1.5 SCOPE OF THE STUDY

The proposed system aims to provide a chess engine that produces chess game data. The system is designed to implement two distinct evaluation functions to describe two artificial intelligences. The system is also designed to be extendable, with the varying of the distinct evaluation functions to investigate the effects of heuristics changes.

1.6 JUSTIFICATION OF THE STUDY

In the development of chess engines, the vast majority of engines use brute force-linear algorithms. In recent years, there has been a push to make chess engines smarter by "teaching" them to play. This is a new area of artificial intelligence research that uses neural networks and deep learning to "teach" a computer to play chess. This project implements a system that produces data that can be used to train these neural networks. Also, heuristic analysis is critical to the strength of chess engines. This system aims to provide data that can be used to determine the effects of changes made to evaluation functions.

CHAPTER TWO

LITERATURE REVIEW

2.1 PREAMBLE

Chess is one of the oldest and most popular board games in the world, played by two opponents on a checkered board with specially designed pieces of contrasting colors, which are usually white and black (Soltis, 2017). As Soltis (2017) explains, white makes the first move, after which the players swap/alternate turns in accordance with fixed rules. The aim of each player is to force the opponent's principal piece, the King, into checkmate—a position where it is unable to avoid capture. (Soltis, 2017)

Of most interest in this project is the field of computational chess, the methods by which chess engines are built, and the relevance of this system to computational chess researchers and enthusiasts. These methods will be discussed in this chapter.

2.2 DEFINITIONS AND ACRONYMS

There are a few terms that come from a specialist register used in this and subsequent chapters. Definitions are presented in this section. Subsequently, the attached acronyms will be used instead.

Artificial Intelligence (AI):

Merriam-Webster defines artificial intelligence to be

"1: a branch of computer science dealing with the simulation of intelligent behavior in computers.

2: the capability of a machine to imitate intelligent human behavior."

Portable Game Notation (PGN):

Wikipedia defines Portable Game Notation as

"a plain text computer-processible format for recording chess games (both the moves and related data), supported by many chess programs."

Integrated Development Environment (IDE):

Wikipedia defines an integrated development environment as

"An integrated development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools, and a debugger. Most of the modern IDEs have intelligent code completion."

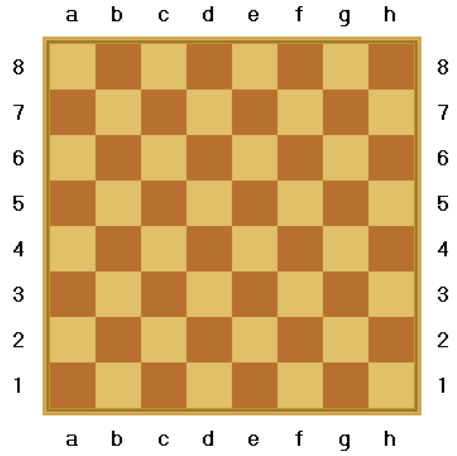
2.3 COMPUTER GAMES

Man has been fascinated by the idea that a computer can play games of skill ever since the advent of the electronic computer (Levy, 1983). The word skill implies some form of intelligence, and while it is implied that only humans can exhibit the type of intelligence needed to play games of skill, such as chess, bridge and poker, it so happens that the type of person who programs computers is also the type of person that enjoys playing games of skill (Levy, 1983). This similarity is important. As it turns out, the procedural analytical assessment made by players of games of skill is exactly that which is needed by computers to solve problems. As Levy (1983) explains, this similarity also accounts for the widespread interest amongst the computing fraternity in the programming of games of skill.

During the first 30 years in the history of computers, it could be argued that since computers were so expensive, it was frivolous to spend so much time on tasks like playing chess or poker (Levy, 1983). On the flip side, the scientific aspect of chess programming for instance showed that in performing a task usually associated with human intelligence, one could be said to be building artificial intellect (Levy, 1983). Levy (1983) also espouses another point in support of those who would devote efforts to writing such programs. He explains that, by writing successfully a program in which long-term planning is involved, one would devise methods which could be applied to other long-term planning problems, such as economic forecasting. This has proven to be true in many aspects of computing and algorithms.

2.4 CHESS

Chess is played on an 8×8 grid, a board of 64 squares arranged in eight columns called files and eight rows called ranks (Soltis, 2017). These squares alternate between two colors: one light, such as white, beige, or yellow; and the other dark, such as black or green. The board is set between the two opponents so that each player has a light-colored square at the right-hand corner. (Soltis, 2017). This is illustrated in the figure below.



2.4.1 ALGEBRAIC NOTATION

Individual moves and entire games can be recorded using one of several forms of notation. Algebraic (or coordinate) notation, identifies each square from the point of view of the player with the light-colored pieces, also called White (Soltis, 2017). Algebraic notation is by far the most widely used form of chess notation.

The eight ranks are numbered 1 through 8 beginning with the rank closest to White. The files are labeled a through h beginning with the file at White's left hand. The name of each square is a combination of its rank and file coordinate, such as h7 or g8. Additional concepts worthy of note are queenside and kingside. The portion of the board containing files a through d are referred to as the queenside, while files e through h is referred to as the kingside. (Soltis, 2017). This can be seen in the image above.

2.4.2 PIECES

KING

Each player has only one king. Soltis (2017) describes a King in his treatise. He says; “White’s king begins the game on e1. Black’s king begins the game at e8 and is opposite to White’s king”. Each king can move one square in any direction; e.g., From its starting position, White’s king can move from to d1, d2, e2, f2, or f1.

ROOK

Each player has two rooks. Soltis (2017) defines a Rook and its functions. He says; “rooks begin the game on the corner squares a1 and h1, and a8 and h8 for White and Black respectively”. A rook can move vertically or horizontally to any unobstructed square along the file or rank on which it is placed.

BISHOP

Each player has two bishops. As Soltis (2017) explains, Bishops begin the game at c1 and f1 for White, c8 and f8 for Black. A bishop can move to any unobstructed square on the diagonal on which it is placed. This results in a peculiar trait, each player has one bishop that travels only on light-colored squares and one bishop that travels only on dark-colored squares.

QUEEN

Each player has one queen. The queen combines the powers of the rook and bishop—i.e., vertical, horizontal and diagonal moves to unobstructed squares. This makes it the most powerful and movable piece. The White queen begins at d1 and the Black queen at d8.

KNIGHT

Each player has two knights. They start out the game on the squares between their rooks and bishops—i.e., at b1 and g1 for White and b8 and g8 for Black. The movement of the knight is unique. Soltis (2017) explains it thus; “the knight moves in an L-shape of two steps: first one square like a rook, then one square like a bishop, but always in a direction away from the starting square”. This means that, a knight at e4 could make a move to any of f2, g3, g5, f6, d6, c5, c3, or d2. The knight has the unique ability to jump over any other piece to a new square. The knight always moves to a square of a different color (Soltis, 2017).

PAWN

Each player has eight pawns. They all begin the game on the second rank closest to each player. White's pawns start at a2, b2, c2, through to h2, while Black's pawns start at a7, b7, c7, through to h7. Pawns are unique in several ways.

One way is that a pawn can move only forward; it can never retreat (Soltis, 2017). Also, a pawn moves differently than it captures. A pawn moves to the square directly ahead of it but captures on the squares diagonally in front of it. For example, a White pawn at f5 can move to f6 but can capture only on g6 or e6.

Another attribute of pawns is that an unmoved pawn has the option of moving one or two squares forward. Soltis (2017) says this is the reason for another unique property of pawns, called en passant—that is French for in passing. En passant is available to a pawn when an enemy pawn on an adjoining file advances two squares on its initial move and could have been captured had it moved only one square (Soltis, 2017). The first pawn can take the advancing pawn, as if it had moved only one square. An en passant capture must be made on the first move or not at all (Soltis, 2017). Only pawns can capture or be captured en passant.

Lastly, any pawn reaching the end of its file is to be promoted—that is, exchanged for—a queen, rook, bishop, or knight (Soltis, 2017).

2.4.3 MOVES

The board represents a battlefield of sorts, in which two armies fight to capture each other's king (Soltis, 2017). In this context, a player's army consists of 16 pieces that begin battle on the two ranks closest to that player, that is on a total of 16 squares (Soltis, 2017). The six different types of pieces: king, rook, bishop, queen, knight, and pawn; as defined above are distinguished by appearance and by how they move. The players alternate moves, White going first (Soltis, 2017).

CAPTURING

The king, rook, bishop, queen, and knight capture enemy pieces in the same manner that they move, the only exception is the pawn. For example, a White queen on d3 can capture a Black rook at h7 by moving to h7 and removing the enemy piece from the board. Pieces can capture only enemy pieces (Soltis, 2017).

CASTLING

The only exception to the rule that a player may move only one piece at a time is a compound move of king and rook called castling (Soltis, 2017). Soltis (2017) explains that a player castles by shifting the king two squares in the direction of a rook, which is then placed on the square the king has crossed. For example, White can castle kingside by moving the king from e1 to g1 and the rook from h1 to f1.

Castling is permitted only once in a game and is prohibited if the king or rook has previously moved or if any of the squares between them is occupied.

Also, castling is not legal if the square the king starts on, crosses, or finishes on is attacked by an enemy piece.

2.4.4 RELATIVE PIECE VALUES

Assigning the pawn a value of 1, the values of the other pieces are approximately as follows: knight 3, bishop 3, rook 5, and queen 9 (Soltis, 2017). The King is given an arbitrarily high value, say 100.

The relative values of knights and bishops vary with different pawn structures. Additionally, tactical considerations may temporarily override the pieces' usual relative values (Soltis, 2017). Material concerns are secondary to winning.

2.4.5 OBJECT OF THE GAME

When a player moves a piece to a square on which it attacks the enemy king—that is, a square from which it could capture the king if the king is not shielded or moved—the king is said to be in check. The game is won when one king is in check and cannot avoid capture or be shielded on the next move; this is called checkmate. A game also can end when a player resigns or acknowledges defeat, believing the situation to be hopeless.

There are three possible results in chess: win, lose, or draw (Soltis, 2017). These are assigned a value of 1, 0 and ½ respectively in PGN (Edwards, 1993).

Soltis (2017) describes the six ways a draw can come about; a draw can be come about (1) by mutual consent, (2) when neither player has enough pieces left to deliver checkmate, (3) when one player can check the enemy king in perpetuity, (4) when a player who is not in check has no legal move (stalemate), (5) when an identical position occurs three times with the same

player having the right to move, and (6) when no piece has been captured and no pawn has been moved within a period of 50 moves.

In competitive events, a victory is scored as one point, a draw as half a point, and a loss as no points.

2.4.6 GAME NOTATION

Moves can be recorded by designating the initial of the piece moved and the square to which it moves (Edwards, 1993). For example, Be5 means a bishop has moved to e5. There are two exceptions: a knight is identified by N, and no initials are used for pawn moves. In the case of the pawn, 1 e4 means White's first move is a two-square advance of a pawn on the e-file, and for the knight, 1 … Nf6 means Black's response is to bring a knight from g8 to f6.

Castling kingside is indicated by 0-0, for both White and Black, while castling queenside is notated by 0-0-0 (Soltis, 2017).

Captures are indicated by inserting an x or: between the piece moving and the square it moves to. For pawn moves, this means dxe5 indicates a White pawn on d4 captures a piece on e5. En passant captures are designated by e.p. (Soltis, 2017).

Checks are indicated by adding ch or + at the end of the move, and checkmate is often indicated by adding # or ++ at the end of the move. Notation is used to record games as they are played and to analyze them in print afterward (Soltis, 2017).

When annotating (commenting) on a game, Soltis (2017) says;

> "an appended exclamation mark means a very good move, two exclamation marks are occasionally used to indicate an extremely good move, a question mark indicates a bad move, two question marks indicate a blunder, and the combination of an exclamation mark and a question mark on the same move indicates a double-edged or somewhat dubious move."

2.5 CHESS AS A COMPUTER GAME

Machines capable of playing chess have fascinated people since the latter half of the 18th century, when the Turk, the first of the pseudo-automatons, began a triumphal exhibition tour of Europe. Like its 19th-century successor Ajeeb, the Turk was a cleverly constructed cabinet that concealed a human master. The mystery of the Turk was the subject of more than

a dozen books and a widely discussed article written by Edgar Allan Poe in 1836. Several world-class players were employed to operate the pseudo-automatons, including Harry Nelson Pillsbury, who was Ajeeb during part of the 1890s, and Isidor Gunsberg and Jean Taubenhaus, who operated, by remote control, Mephisto, the last of the pseudo-automatons, before it was dismantled following World War I.

2.5.1 MASTER SEARCH HEURISTICS

The ability of a machine to play chess well has taken on symbolic meaning since the first precomputer devices more than a century ago. In 1890 a Spanish scientist, Leonardo Torres y Quevado, introduced an electromagnetic device—composed of wire, switch, and circuit—that was capable of checkmating a human opponent in a simple endgame, king and rook versus king. The machine did not always play the best moves and sometimes took 50 moves to perform a task that an average human player could complete in fewer than 20. But it could recognize illegal moves and always delivered eventual checkmate. Torres y Quevado acknowledged that the apparatus had no practical purpose. As a scientific toy, however, it gained attention for his belief in the capability of machines to be programmed to follow certain rules.

No significant progress in this area was made until the development of the electronic digital machine after World War II. About 1947 Alan Turing of the University of Manchester, England, developed the first simple program capable of analyzing one ply (one side's move) ahead. Four years later a Manchester colleague, D.G. Prinz, wrote a program capable of solving mate-in-two-move problems but not actually playing chess.

A breakthrough came in 1948, when the research scientist Claude Shannon of Bell Telephone Laboratories in Murray Hill, New Jersey, U.S., presented a paper that influenced all future programmers. Shannon, like Torres y Quevada and Turing, stressed that progress in developing a chess-playing program would have a wider application and could lead, he said, to machines that could translate from language to language or make strategic military decisions.

Shannon appreciated that a computer conducting an entire game would have to make decisions using incomplete information because it could not examine all the positions leading to checkmate, which might lie 40 or 50 moves ahead. Therefore, it would have to select moves that were good, not merely legal, by evaluating future positions that were not checkmates. Shannon's paper set down criteria for evaluating each position a program would consider.

This evaluation function is crucial because even a rudimentary program would have to determine the relative differences between thousands of different positions. In a typical position

White may have 30 legal moves, and to each of those moves Black may have 30 possible replies. This means that a machine considering White's best move may have to examine 30 × 30, or 900, positions resulting from Black's reply, a two-ply search. A three-ply search—an initial move by White, a Black reply, and a White response to that—would mean 30 × 30 × 30, or 27,000, different final positions to be considered. (It has been estimated that humans examine only about 50 positions before choosing a move.)

Turing's evaluation function was dominated by determining which side had more pieces in various future positions. But Shannon suggested that each position could be weighed using positional criteria, including the condition of pawns and their control of the centre squares, the mobility of the other pieces, and specific cases of well-placed pieces, such as a rook on an open (pawnless) file or on the seventh rank. Other criteria were used by later programmers to refine and improve the evaluation function. All criteria had to be quantified. For example, a human master can quickly evaluate the mobility of bishops or the relative safety of the king. Early programs performed the same evaluation by counting the number of legal bishop moves or the squares under control around a player's king.

2.5.2 COMPUTER CHESS

Computers began to compete against humans in the late 1960s. In February 1967 MacHack VI, a program written by Richard Greenblatt, an MIT undergraduate, drew one game and lost four in a U.S. Chess Federation tournament. Its results improved markedly, from a performance equivalent to a USCF rating of 1243 to reach 1640 by April 1967, about the average for a USCF member. The first American computer championship was held in New York City in 1970 and was won by Chess 3.0, a program devised by a team of Northwestern University researchers that dominated computer chess in the 1970s.

Technical advances accelerated progress in computer chess during the 1970s and '80s. Sharp increases in computing power enabled computers to "see" much further. Computers of the 1960s could evaluate positions no more than two moves ahead, but authorities estimated that each additional half-move of search would increase a program's performance level by 250 rating points. This was borne out by a steady improvement by the best programs until Deep Thought played above the 2700 level in 1988. When Deep Blue, its successor, was introduced in 1996, it saw as far as six moves ahead. (Gary Kasparov said he normally looks only three to five moves ahead, adding that for humans more are not needed.)

Also helping computer progress was the availability of microprocessors in the late 1970s. This allowed programmers unattached to universities to develop commercial microcomputers that by the 1990s were nearly as strong as programs running on mainframes. By the late 1980s the strongest machines were capable of beating more than 90 percent of the world's serious players. In 1988 a computer, HiTech, developed at Carnegie Mellon University, defeated a grandmaster, Arnold Denker, in a short match. In the same year another Carnegie Mellon program, Deep Thought, defeated a top-notch grandmaster, Bent Larsen, in a tournament game.

HiTech used 64 computer chips, one for each square on the board, and was capable of considering up to 175,000 positions per second. Feng-Hsiung Hsu, a Carnegie Mellon student, improved on HiTech with a custom-designed chip. The result, Chiptest, won the North American Computer Championship in 1987 and evolved into Deep Thought, a program powerful enough to consider 700,000 positions a second. Although its evaluation skills were not as well developed as HiTech's—and far below that of a human grandmaster—Deep Thought was sponsored by International Business Machines Corporation (IBM) in an effort to defeat the world's best player by the mid-1990s in a traditional time limit.

At faster speeds even personal computers were able to defeat the world's best humans by 1994. In that year a Fritz 3 program, examining 100,000 positions per second, tied for first place with Kasparov, ahead of 16 other grandmasters, at a five-minute tournament in Munich, Germany. Later in the year Kasparov was eliminated from a game/25 tournament in London after losing a two-game match against Genius running on a Pentium personal computer.

In 1991 Deep Thought's team said the program, renamed Deep Blue, would soon be playing at the equivalent of a 3000 rating (compared with Kasparov's 2800), but this proved excessively optimistic. The main improvement was in the computer running the chess program. IBM developed, and used chess to test, a sophisticated new multiprocessing system (later used at the 1996 Olympic Games in Atlanta, Georgia, U.S., to predict the weather) that employed 32 microprocessors, each with six programmable chips designed specifically for chess. Deep Thought, by comparison, had one microprocessor and no extra chips. The new hardware enabled Deep Blue to consider as many as 50 billion positions in three minutes, a rate that was about a thousand times faster than Deep Thought's.

Deep Blue made its debut in a six-game match with PCA champion Kasparov in February 1996. The $500,000 prize fund and IBM's live game coverage at their World Wide Web site attracted worldwide media attention. The Kasparov–Deep Blue match in Philadelphia was the first time a world champion had played a program at a slow (40 moves in two hours)

time format. Deep Blue won the first game, but Kasparov modified his style and turned the later games into strategic, rather than tactical, battles in which evaluation was more important than calculation. He won three and drew two of the remaining games to win the match 4–2.

In a six-game rematch held May 3–11, 1997, in New York City, an upgraded Deep Blue was able to consider an average of 200 million positions per second, twice its previous speed. Its algorithm for considering positions was also improved with advice from human grandmasters.

By adopting a new set of conservative openings, Kasparov forced Deep Blue out of much of its prematch preparation. After resigning the second game, in a position later found to be drawable, Kasparov said he "never recovered" psychologically. With the match tied at one win, one loss, and three draws, Deep Blue won the decisive final game in 19 moves.

2.5.3 COMPUTER EXTENSION OF CHESS THEORY

Computers have played a role in extending the knowledge of chess. In 1986 Kenneth Thompson of AT&T Bell Laboratories reported a series of discoveries in basic endgames. By working backward from positions of checkmate, Thompson was able to build up an enormous number of variations showing every possible way of reaching the final ones. This has been possible with only the most elementary endgames, with no more than five pieces on the board. Thompson's research proved that certain conclusions that had remained unchallenged in endgame books for decades were untrue. For example, with best play on both sides, a king and queen can defeat a king and two bishops in 92.1 percent of the initial starting positions; this endgame had been regarded as a hopeless drawn situation. Also, a king and two bishops can defeat a king and lone knight in 91.8 percent of situations—despite human analysis that concluded the position was drawn. Thompson's research of some five-piece endgames required considering more than 121 million positions.

Because of their ability to store information, computers had become invaluable to professional players by the 1990s, particularly in the analysis of adjourned games. However, computers have severe limits. In the 1995 PCA championship, Kasparov won the 10th game with a heavily analyzed opening based on the sacrifice of a rook. According to his aides, the prepared idea was tested on a computer beforehand, and the program evaluated the variation as being in the opponent's favor until it had reached the end of Kasparov's lengthy analysis.
The availability of top-notch microcomputers poses a major problem for postal chess. A principal difference between over-the-board chess and all forms of correspondence chess is

that in the latter players are permitted to analyze a position by moving the pieces and by consulting reference books. By the 1990s most serious postal players used a computer database containing thousands of games categorized by opening moves. However, if the use of computers is extended to finding the best moves in the middlegame or endgame, postal chess becomes computer chess. The International Correspondence Chess Federation said in 1993 that "the existence of chess computers is a reality and for correspondence chess the use of chess computers cannot be controlled."

2.5.4 CHESS GAME DATA AS PGN

2.6 SIMILAR SYSTEMS

2.6.1 SUNFISH

Sunfish is a simple, but strong chess engine, written in Python, mostly for teaching purposes. Without tables and its simple interface, it takes up just 111 lines of code. (Ahle, 2017)

Because Sunfish is small and strives to be simple, the code provides a great platform for experimenting. People have used it for testing parallel search algorithms, experimenting with evaluation functions, and developing deep learning chess programs. Fork it today and see what you can do. (Ahle, 2017)

2.6.2 DEEP PINK

Deep Pink is a chess AI that learns to play chess using deep learning. Here (http://erikbern.com/2014/11/29/deep-learning-for-chess/) is a blog post providing some details about how it works. (Bernhardsson, 2017)

There is a pre-trained model in the repo, but if you want to train your own model you need to download pgn files and run parse_game.py. After that, you need to run train.py, preferably on a GPU machine since it will be 10-100x faster. This might take several days for a big model. (Bernhardsson, 2017)

2.7 LOCI AND THE THEORY FOR CHESS ENGINES

Next, we examine the theory that is relevant to building a basic chess AI. Although there is variation between engines, almost all chess engines today implement the same algorithms (Lai, 2015). They are all based on the idea of the fixed-depth minimax algorithm developed by John von Neumann in 1928 (Neumann, 1928). It is contentious what it is that fundamentally sets these engines apart, though it is conjectured that the evaluation function is responsible for the major differences between the top chess engines (Levy, 1983).

2.7.1 THEORY FOR CHESS ENGINES

Neumann's minimax algorithm was adapted to the problem of chess by Claude E Shannon in 1950 (Shannon, 1950). In Shannon's solution, he describes the minimax algorithm alongside α-β Pruning.

MINIMAX

The minimax algorithm is a simple recursive algorithm to score a position based on the assumption that the opponent thinks like we do, and also wants to win the game (Lai, 2015).

In its simplest form, minimax can be expressed as

```
function minimax(position) {
      if position is won for side to move:
            return 1
      else if position is won for the opponent:
            return -1
      else if position is drawn:
            return 0

      bestScore = -∞

      for each possible move mv:
            subScore = -minimax (position.apply(mv))
            if subScore > bestScore:
                  bestScore = subScore

      return bestScore
}
```

This algorithm works in theory, and also in practice for simpler games like tic-tac-toe (game tree size of at most 9! or 362880) (Lai, 2015). Tic-tac-toe is a solved game. This means that for any given game, the end can always be computed. Chess is not solved, and we might never be able to. The search tree size of chess is estimated to be about 10^{123} (Allis, 1994) which is more than one hundred orders of magnitudes higher than what is computationally feasible using modern computers (Lai, 2015). For comparison, the estimated total number of atoms in the known universe is 10^{78} to 10^{82}. Therefore, a chess engine must decide which parts of the game tree to explore (Lai, 2015).

The most common approach is a fixed-depth search

CHAPTER THREE

METHODOLOGY

3.1 OVERVIEW

In developing this system, the **prototyping paradigm** of software engineering was be used. Sommerville (2011) describes software engineering to be the production of software from the early stages of system specification through to maintenance of the system after it has been deployed. For this project, the development of the software system includes the following activities:

- System requirement analysis
- Design
- Implementation
- Deployment and testing

3.2 EXISTING SYSTEMS

The existing systems, by which a developer of deep learning chess engines could collate data for the training of its neural network is largely a mammoth task of obtaining precomputed data of chess games from online repositories and normalizing or parsing such data into usable formats either by manual means or automated scripting or both.

Also, there are no systems of note, which permit evaluation function changes to the degree of studying effects of heuristics changes.

The system requirements are obtained both from an examination of the existing systems and by considerations of how it can be extended upon and made better.

3.3 PROPOSED SYSTEM

The proposed system generates game data dynamically, as games played in-the-moment by two distinct “artificial intelligences” and stored as portable game notation (PGN).

The proposed system is very processor intensive and requires a considerable amount of computational power to run efficiently. This is the most crucial non-functional requirement of

the system. Since the system does not deal with the management of a database, security as a non-functional requirement is not a primary concern.

The system is designed to receive input from the user on the number of games to compute. It is also designed to receive configuration input specifying the value of individual game pieces. These configuration files prescribe the behavior of the artificial intelligences that will play against each other. After the above-specified input has been supplied by the user, computation proceeds for a given period of time, hours or days even. The time spent in computation is a factor of the number of games played by the engine and the computational power of the deployment environment.

The proposed system has a **client-server architecture**, with a command-line interface as the client frontend, and the chess engine as the server backend.

This system is a niche application and thus requires some expertise with computational chess to be used effectively.

3.4 SYSTEM REQUIREMENTS SPECIFICATION

Software system requirements can be classified as functional requirements or non-functional requirements (Sommerville, 2011). Functional requirements describe the services that a system provides, how it reacts to certain input, and how it behaves in given situations (Sommerville, 2011). Non-functional requirements are the constraints on the services the system offers. This includes timing constraints, development constraints, and constraints imposed by standards for the software system (Sommerville, 2011)

3.4.1 NON-FUNCTIONAL REQUIREMENTS

For this system, the primary non-functional requirement is **performance**. The system requires a considerable amount of computational power to produce results within reasonable time limits, as it is very process bound and computationally complex system. This requires a computer with a very fast processor and huge amounts of memory.

Correctness is also a key non-functional requirement of the system. The correctness of the game data stored in portable game notation (PGN) depends on the correctness of each stored move. Any errors produced by the system would affect the integrity of the produced data.

Requirements like security, while considered, are not important requirements for this system.

Hardware Requirements

Recommended specifications

Processor Intel® Core™ i7-7700HQ processor Quad-core 2.80 GHz

Memory 16 GB DDR4 RAM Memory

Software Requirements

Windows 7, 8, 8.1, 10

Python 3.x

Python Chess

Pygame

Tools

Visual Studio Code

Python

3.4.2 FUNCTIONAL REQUIREMENTS

The functional requirements of a system describe what that system should do, along with its primary functions (Sommerville, 2011).

The functional requirements of this system are:

- A user shall be able to tell the system how many games they want to be computed.
- A user shall be able to extend the system and implement their own evaluation functions.
- A user shall be able to access the data of computed games in portable game notation (PGN) format.
- The system shall be able to receive input specifying the number of games to be played, and validate the correctness of this input.
- The system shall be able to receive configuration input describing the evaluation functions for the AI that will play the specified number of games, and validate the correctness of this input.
- The system will play the specified amount of games and produce the data of those games in portable game notation (PGN) format.

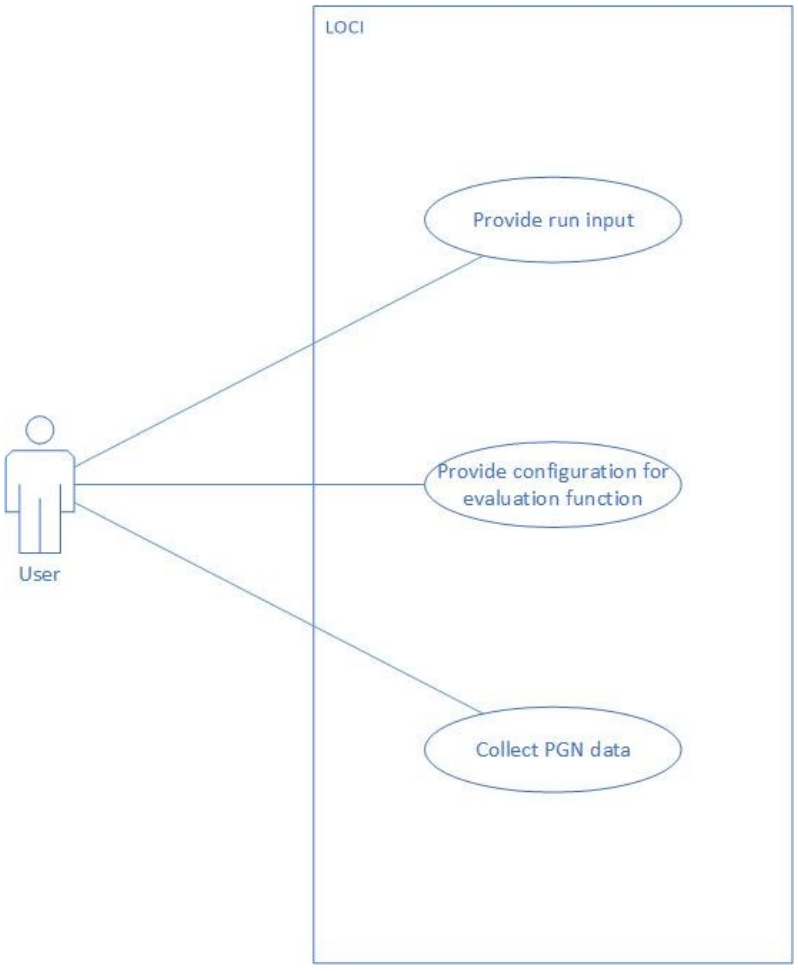
3.5 SYSTEM MODELING

In designing this system, some system models are used to describe the various aspects of the system's operation. For the interaction model, a use case diagram is produced. An activity diagram along with a development view of the system architecture is also produced.

3.5.1 USE CASE DIAGRAM

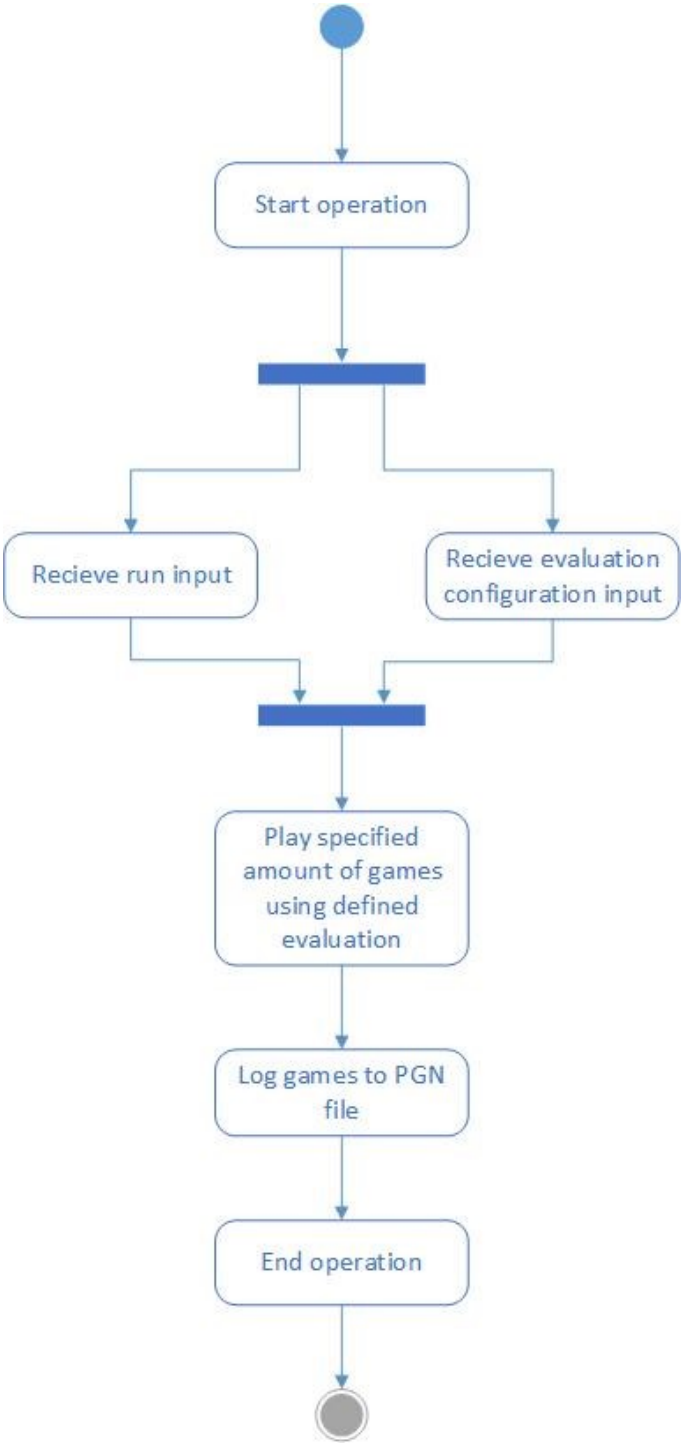
The use case model is used to describe a simple scenario of what a user expects from a system (Sommerville, 2011).

The use case diagram presented below makes a simplification of the above stated functional requirements, and depicts it in terms of how the user interacts with the system.



3.5.2 ACTIVITY DIAGRAM

Activity diagrams are used to show all the activities that are included in a system process and the flow of control from one activity to another (Sommerville, 2011).

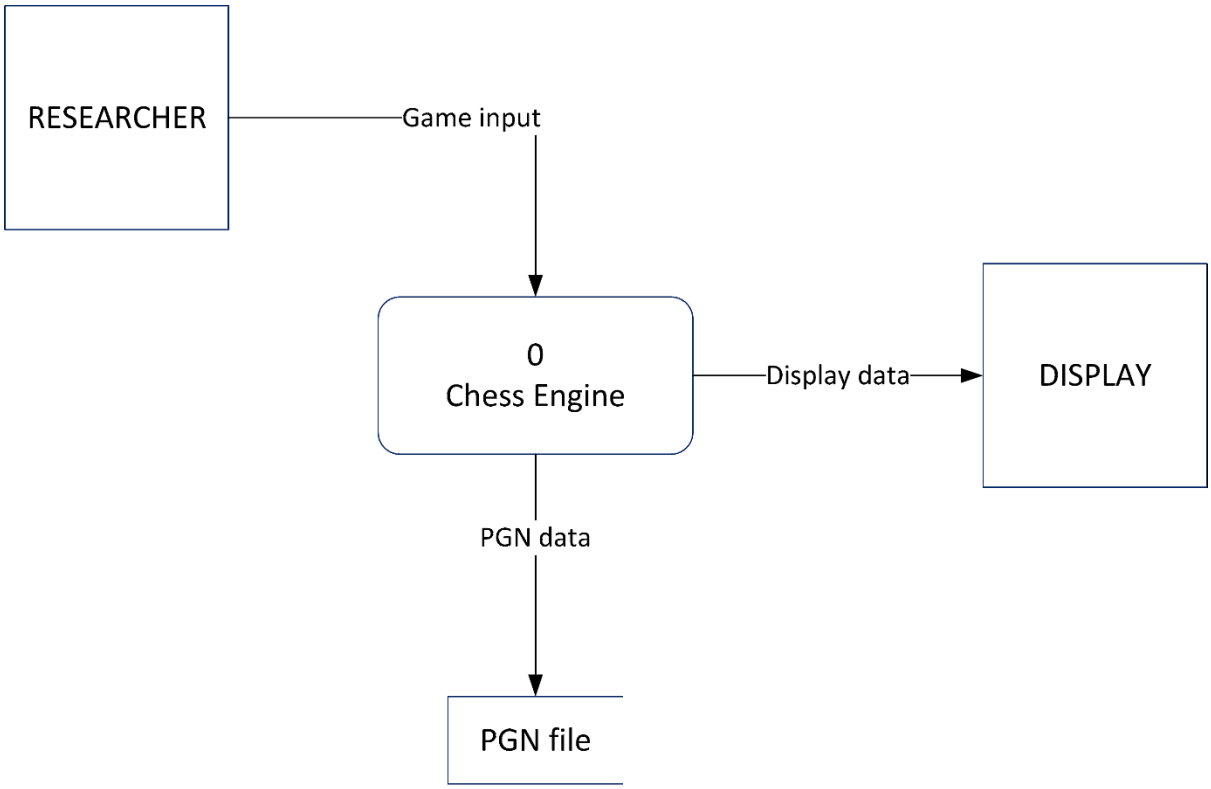


3.5.4 DATA-FLOW DIAGRAM

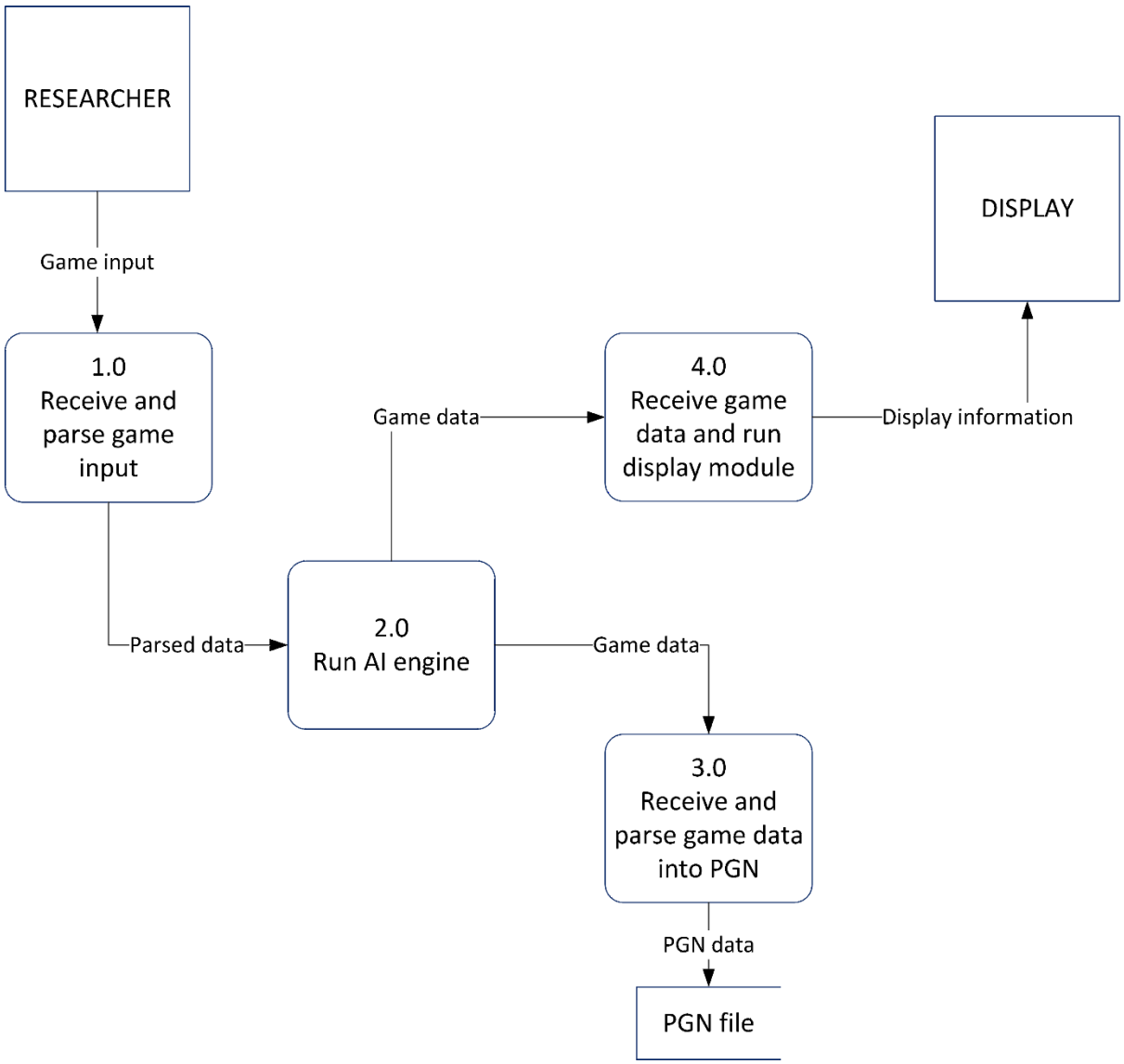
Data-flow diagrams are used in the process modeling of a system. Valacich et al (2012) describe a data-flow diagram as a graphic/image that illustrates the movement of data between external entities, the processes, and the data stores within a system.

There are two levels of abstraction of the DFD that are of interest in the design of this system, the **context diagram** and the **level-0 diagram**.

A context diagram shows the boundary or scope of the system, and the system's relationship to its environment (Valacich, George, & Hoffer, 2012). The context diagram has only one process. Below is the context diagram for this system.



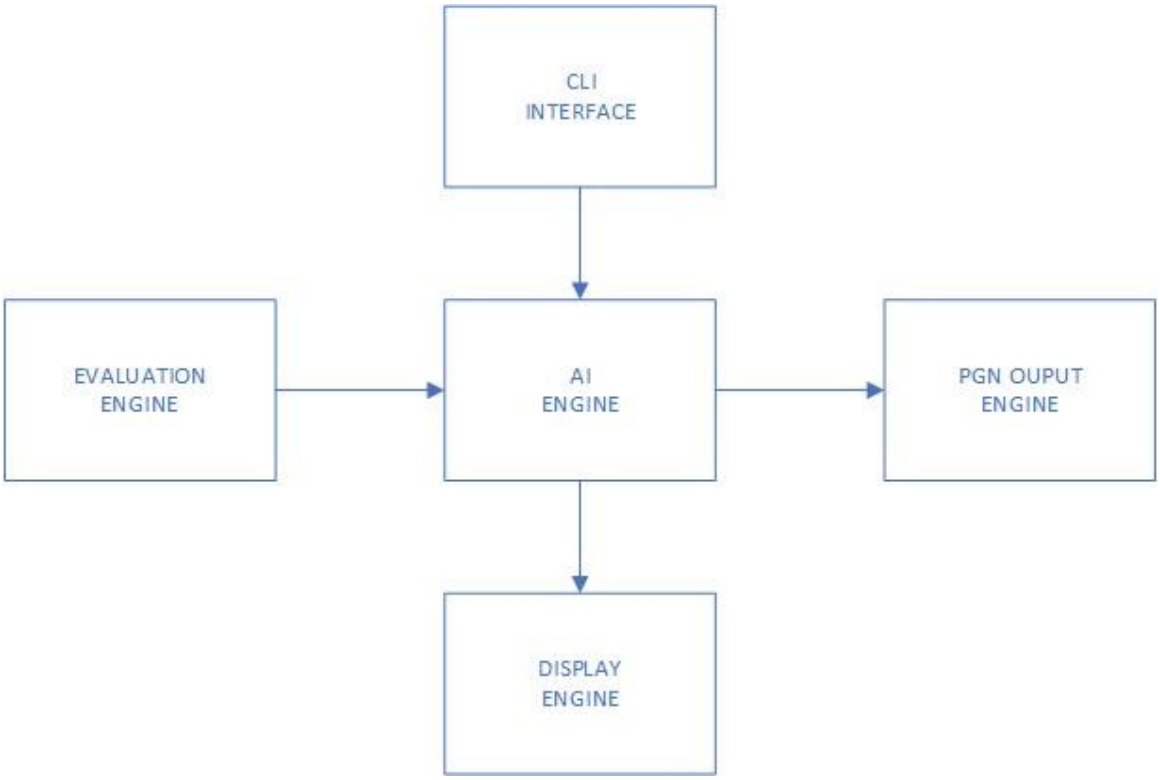
The level-0 diagram represents the basic individual processes in the system at the highest level of abstraction (Valacich, George, & Hoffer, 2012). Below is the level-0 diagram for this system. There are 4 primary processes described.



3.5.4 SYSTEM ARCHITECTURE

As Somerville (2011) discusses, architectural modelling can be done by examining what views or perspectives are useful when designing and documenting a system. Somerville (2011) goes on to describe four views that are relevant to system architectures; *a logical view, a process view, a develoment view, and a physical view*.

For the purposes of this system, I desribe a **development view** of system architecture below.



3.6 IMPLEMENTATION TOOLS

The language chosen for implementation is **Python**. Python is desirable for use in this project for some reasons.

One reason is that code written in Python is easy to read as it has a very English-like syntax. This is important because one of the functional requirements is user extendibility of the system. Python is a scripting language and this makes it easier for a third-party to add functionality to the code.

Another reason is that python has suitable APIs and libraries that make the development of this system easier. One of those libraries is **python-chess**, a pure Python chess library with move generation, move validation and support for common formats (Fiekas, 2019). The version of python-chess used with this project is *python-chess 0.23.8*.

The visualization GUI will be built using **pygame**. Pygame is a Python wrapper module for the SDL multimedia library. It contains python functions and classes that will allow you to use SDL's support for playing CD-ROMs, audio and video output, and keyboard, mouse and joystick input (Shinners, Dudfield, Appen, & Pendleton, 2018).

The chosen IDE is **Visual Studio Code**.

CHAPTER FOUR

IMPLEMENTATION

4.1 OVERVIEW

This chapter focuses on discussing the results obtained after applying the steps and methods discussed in the previous chapter. Furthermore,

4.2 DOCUMENTATION

4.3 TESTING

CHAPTER FIVE

CONCLUSION

5.1 SUMMARY

5.2 RECOMMENDATIONS

5.3 CONCLUSION

APPENDIX

REFERENCES

Ahle, T. D. (2017). *Sunfish*. Retrieved from GitHub: https://github.com/thomasahle/sunfish

Bernhardsson, E. (2017). *Deep Pink*. Retrieved from Github: https://github.com/erikbern/deep-pink

Edwards, S. J. (1993). *Standard: Portable Game Notation Specification and Implementation Guide.* Retrieved from http://www.saremba.de/chessgml/standards/pgn/pgn-complete.htm

Fiekas, N. (2019, January). *python-chess*. Retrieved from https://pypi.org/project/python-chess/

Lai, M. (2015). *Giraffe: Using Deep Reinforcement Learning to Play Chess.* Imperial College London, Department of Computing, London.

Levy, D. (1983). *Computer Gamesmanship: The Complete Guide to Creating and Structuring Intelligent Games Programs.* New York, New York, United States of America: Simon & Schuster, Inc.

Neumann, J. v. (1928). Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 295-320.

Rodriguez, J. (2017, February 21). *Game Theory and Artificial Intelligence.* Retrieved from Medium: https://medium.com/@jrodthoughts/game-theory-and-artificial-intelligence-ee8a6b6eff54

Shannon, C. E. (1950). XXII. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 256-275.

Shinners, P., Dudfield, R., Appen, M. v., & Pendleton, B. (2018, July). *Pygame*. Retrieved from https://pypi.org/project/Pygame/

Soltis, A. E. (2017, September 26). Chess. *Encyclopædia Britannica*. Encyclopædia Britannica, inc. Retrieved February 12, 2019, from https://www.britannica.com/topic/chess

Sommerville, I. (2011). *Software Engineering* (9th ed.). United States of America: Pearson Education, Inc.

Valacich, J. S., George, J. F., & Hoffer, J. A. (2012). *Essentials of Systems Analysis and Design.* Pearson Education, Inc.