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

JUNE, 2018

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)

Although they differ in implementation, almost all chess engines in existence today (and all of the top contenders) implement largely the same algorithms. They are all based on the idea of the fixed-depth minimax algorithm first developed by John von Neumann in 1928, and adapted for the problem of chess by Claude E. Shannon in 1950. (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). These datasets are entered into online repositories for referencing (e.g. https://www.ficsgames.org/). This project allows researchers to generate the data needed for the "training" of this new breed of "intelligent" game engines.

Also, ready tools for the comparison of different evaluation function metrics are not readily available. This project intends to address this problem by allowing researchers provide custom evaluation heuristics directly to the engine, and then observe how it performs.

1.3 AIM AND OBJECTIVES

The aim of this project is to build a chess engine that produces data that can be used for the training of neural networks in deep learning chess engines, used in the analysis of chess engine strength, and be used to rapidly determine the effect of heuristic changes made to chess engine evaluation functions.

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 will be done using use case model, and activity diagrams.
- Implementation will be done with the python programming language with plain text data in portable game notation (PGN) as the data output format.
- Deployment and Testing would be done on a personal 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 provides the data necessary 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, commonly white and black. White makes the first move, after which the players alternate turns in accordance with fixed rules, each player attempting to force the opponent's principal piece, the King, into checkmate—a position where it is unable to avoid capture. (Soltis, 2017)

2.2 THEORETICAL BACKGROUND

2.2.1 CHESS

Chess is played on a board of 64 squares arranged in eight vertical rows called files and eight horizontal rows called ranks. 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)

2.3 CONTEXT OF WORK

2.4 REVIEW OF RELATED WORK

2.5 DEFINITION OF TERMS

Artificial Intelligence (AI):

Integrated Development Environment (IDE):

CHAPTER THREE

METHODOLOGY

3.1 OVERVIEW

In developing this system, the **prototyping paradigm** of software engineering will be used. Sommerville 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 (Sommerville, 2011). 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 amount 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 amount 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 and usability, 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).

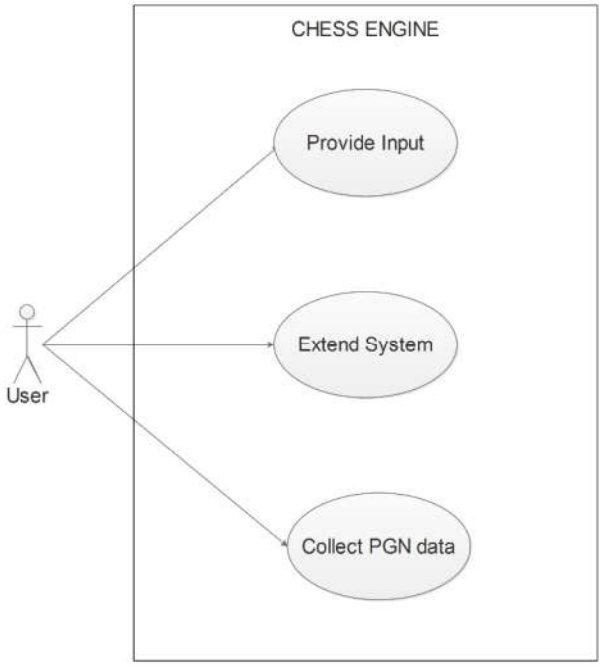
The functional requirements of this system are:

- A user shall be able to tell the system how many games they want 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.

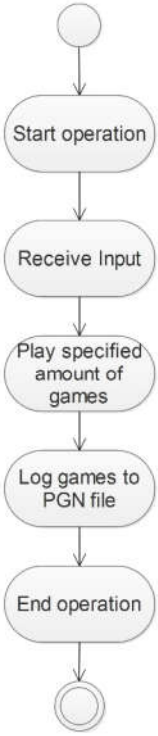
3.5 SYSTEM MODELING

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 is also produced.

3.5.1 USECASE DIAGRAM



3.5.2 ACTIVITY DIAGRAM



3.6 IMPLEMENTATION TOOLS

The language chosen for implementation is the **Python** programming language. 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 cdroms, 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 INTRODUCTION

4.2 DOCUMENTATION

4.3 TESTING

CHAPTER FIVE

CONCLUSION

5.1 SUMMARY

5.2 RECOMMENDATIONS

5.3 CONCLUSION

REFERENCES

APPENDIX