AI Puzzle Solving

Final Report for CS39440 Major Project

Author: Joshua Cant (joc105@aber.ac.uk)

Supervisor: Myra Wilson (mxw@aber.ac.uk)

03/05/2024

Version 1.0

This report is submitted as partial fulfilment of BSc degree in Computer science (G400)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

Name: Joshua Cant

Date: 09/04/2024

Consent to share this work.

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Joshua Cant

Date: 09/04/2024

Acknowledgements

I am grateful to my supervisor Myra Wilson for overseeing the project and keeping me on the right track and sharing her knowledge with me.

I am also grateful to my second marker Tossapon Boongoen for his helpful advice during the mid-term demonstration.

Abstract

The AI Puzzle generator is an application designed to generate puzzles of varying difficulty for the user to solve. Each puzzle is different each time you play the game keep the playing engaged. This report describes the process, problems and other issues that occurred during the development of the project.

The project involved creating a chess playing and puzzle creating AI in Python programming language. The program was developed using an agile development methodology.

The purpose of the project was to create a program that could create and solve random puzzles with a difficulty factor taking into account. It involves creating a basic command line interface for which the user can interact with.

The report itself contains solutions to problems that many AI engineers encounter such as performance issues and optimisation. The report is able to tell you in detail how the algorithm works and how it could be improved on.

Contents

Figures

Tables

1. Background, Analysis & Process

1.1 Background

The earliest attempts of creating a chess-playing algorithm arose in the 1940s and 50s [1], at the same time as the first electronically digital computer was made [2]. From this time until now chess-playing algorithms have changed a lot, along with the computational powers of computers. The UNIVAC 1 operates at a speed of 2.25 MHz, while in contrast my CPU has a clock speed of 3.7 GHz on six cores [3]. The size of memory also increased over time, with computers from the fifties having memory sizes in the kilobytes, and modern computers have memory sizes in the gigabytes. This means that as computers progressed, more complicated and more computationally expensive algorithms could be used for chess-playing algorithms. The first chess-playing algorithm to beat a world champion was Deep(er) Blue in 1997 [1], this was a famous game among the chess community.

My project is to create a puzzle creator and a puzzle solver. The main issue for this project is how to quantify the puzzle difficulty, and to have my puzzle solver and creator efficient. The two ways in which I am quantifying puzzle difficulty, is through how many moves it takes to get to check mate, and the total number of possible moves. Number of moves to checkmate is a good way to quantify a puzzle as it can be less obvious to find the correct move if checkmate is further away, as it is difficult to think that far ahead in the game. Total number of possible moves is also a good way to quantify how difficult a puzzle is as it can be difficult to find the correct move if there are many moves to choose from. My project will allow the user to attempt to solve the puzzle in the same amount of moves that the puzzle solver solved it in. If the user can solve the puzzle, the program will move to the next puzzle, if the user fails to solve the puzzle, the program will show the user the solution it found using the puzzle solver.

I have always been interested in how things work, especially video games, which I took a keen interest in from an early age. I was particularly interested how enemies worked against the player, and how they were able to make intelligent choices. I was very into football so one my favourite games were FIFA, which has an impressive AI that can play as an entire team. I have been casually interested in chess for a few years, playing it during lockdown during the boom of chess on twitch. I also have a keen AI in general, similarly to my interest in game AI, I am also interested in how Ais like ChatGPT function and Ais that control robots and self-driving cars. Combining my love for games, chess and Ai seemed like an obvious choice for my major project, and making my program also create chess puzzles gives to more challenge than just a chess player.

Throughout my university journey I have taken many modules, many of which I have taken a keen interest in. The two main modules which interested me the most was CS31920 Advanced Algorithms [4] and CS26520 Artificial Intelligence [5]. These two modules give me a firm foundation on which to complete this project.

1.2 Analysis

At first glance, chess can be a very simple game, however upon further research, chess is extremely complicated, not just in playing but the algorithms behind chess engines. Chess algorithms need to search the entire problem space, which for chess is large. The average branching factor of chess is said to be 35. After statistical analysis of over 2.5 million games, the branching factor was revealed to be closer to 31 [6]. This means that there is an average of 35 possible moves in any configuration. This makes algorithms that follow every branch on every node such as a brute force search, computational expensive. Brute force searches suffer from exponentially increasing number of nodes, which will lead to combinatorial explosion if other methods are not implemented to limit the

search space [6]. There are many different methods that can be used to limit the search space which I will discuss later.

1.2.1 Chess Engine

The need for a chess engine in this project is vital, I need the chess engine to interact with the game, to create the pieces, the board, check for checkmate etc. I needed to decide whether to create my own chess engine or to use a premade python chess engine. I quickly decided that creating my own chess engine along with the rest of my project would be too much work. There are a few good chess libraries which I could use. I landed on using python-chess [7] for my chess engine. This library is comprehensive compared to its counterparts such as Chessnut [8]. This engine will give my project a firm foundation and will allow me to focus entirely on my puzzle solver and puzzle creator without having to waste time creating a chess engine.

1.2.2 Programming Language

Any language could be used for this project, each with their benefits and drawbacks. I ultimately decided to use python for many reasons. Python is simple and very flexible; it has an easy-to-read syntax making it easy for me to implement features and not have to worry about complicated syntax [9]. It will also make the code more readable and maintainable in the future. Python also has an extensive number of libraries and frameworks, and many are tailored for AI development [9]. The chess engine mentioned previously is a good example of a library that is perfect for my project. There are also other libraries such as NumPy, SciPy, math, pandas, TensorFlow, PyTorch and Keras. These libraries provide powerful tools for data manipulation, statistics, machine learning and deep learning which all could be used for this project. Python is also cross platform, meaning it can be run on any operating system [10]. This means my project can be run on any machine or device the user wants.

1.2.3 Search Algorithm

The most important part of the puzzle solver is the search algorithm. There are many algorithms and techniques to choose from. The most common algorithm is the minimax algorithm. Minimax is a recursive decision rule algorithm which is used in Artificial Intelligence, game theory and statistics [11]. The minimax algorithm alternates between maximising the score for itself and minimising the score for the opponent. In other words, it minimises the possible loss for the worst-case scenario [11]. Minimax by itself is a brute force algorithm. As mentioned previously, brute force algorithms are computationally expensive. The time complexity for minimax is O(b^d), where b is the branching factor, and d is the search depth. This is computationally expensive, for example if we use a depth of 5 with a branching factor of 35, the total number of nodes would be 52521875. This is a lot of nodes to evaluate especially if you have a slow evaluation function.

Luckily there are methods we can use to reduce the search space, such as alpha-beta pruning. Alpha-beta pruning is a variation of the minimax algorithm, but it stops evaluating a node when at least one possibility has been found to that proves the move to be worse than a previously examined node. This means than further nodes do not need to be evaluated further as we know it would give a worse result [12]. This means it returns the same value as a normal minimax algorithm but prunes away branches that will not affect the final decision [12]. The algorithm stores two values, alpha and beta. Alpha represents the minimum score that the maximizing player is guaranteed to get, and beta represents the maximum score that the minimizing player can is guaranteed for [12]. Alpha is represented as minus infinity, and beta is represented as positive infinity, basically meaning both players start with the worse possible moves. Using this, the algorithm will know whenever the maximum score that the minimizing player is guaranteed to become less than the minimum score that the maximizing player is guaranteed of. This means the maximizing player does not need to

consider looking further down this branch. To give a real-world example, if the player has identified two good moves, move A and move B. Move A will improve the player's position and so will move B, but if the player plays move B, the opponent will be able to force checkmate in a couple of moves. This means the outcomes and further possible moves from playing move B no longer need to be considered as they know the opponent can force checkmate [12]. The worst-case of alpha-beta pruning is the same as the brute force minimax, which being O(b^d) [12]. The best-case performance of alpha-beta pruning is O($\sqrt{b\hat{}d}$) [12].

1.2.4 Other Algorithms

Another popular algorithm to use is the Monte Carlo tree search. The Monte Carlo method employs random sampling to tackle deterministic that are considered challenging or even impossible through other methods. In 1987, Bruce Abramson combined the standard minimax algorithm with an "expected-outcome model based on random game playouts until the end" [13]. This is different to the standard minimax algorithm which employs a static evaluation function. In 1992, B. Brügmann created a Go playing program which employed Monte Carlo method. The term "Monte Carlo tree search" was coined in 2006 by Remi Coulom [13]. In a chess AI, a Monte Carlo search tree would simulate many random game moves from the current state to estimate the value of a move.

The major questions is, which algorithm is faster and which algorithm is more applicable to my project? The run time of Monte Carlo tree search is represented as O(C x b^d) [14]. Where C is the average number of simulations performed, b is the branching factor and d is the depth of search. It is also important to note that the time complexity is often represented with number of simulations and not nodes as it uses sampling to find the most promising route instead of searching the entire tree. While on paper this makes alpha-beta pruning look more efficient than Monte Carlo, but this may not be the case as alpha-beta will need to use an evaluation function for each node, which if inefficient, will greatly increase runtime of the algorithm. It is also important to look at the time of problem the algorithm will solve. As my algorithm will need to solve a puzzle of a certain depth, the minimax algorithm is better suited. Depending on the difficulty selected by the user, I know that the solution will either be with 1, 2 or 3 depths. This means an exhaustive search will be better than random sampling, as the random sampling may not return the 100% correct move every while an exhaustive search will. While Monte Carlo would be better and more efficient for a chess AI that must play from the beginning of the game, this is not needed for my project.

It also worth noting that Monte Carlo tree search is a relatively new algorithm, it has been implemented in many chess engines such as Stockfish [15] and AlphaZero [16]. These engines use deep neural networks to greatly lower the number of evaluations, which greatly improves performance. If I had more time for this project, I would use a neural network which means I could possibly use Monte Carlo over minimax.

1.2.5 Evaluation function

The evaluation function is the backbone of the minimax algorithm. The evaluation function is a function that used to estimate the value of a position in a game tree [17], in my case, the function will evaluate the configuration of the board. The functions only look at the current configuration of the board, and not previous or future configurations. On surface level chess evaluation is rather simple, but diving deeper it becomes considerably more complicated. In chess Ais, the units returned by an evaluation are called "pawns" [17]. In chess piece evaluation, a pawn has a value of 1, so if a white was up by a pawn, then the evaluation would be +1, if black were up by a pawn, the evaluation would be -1. Other pieces are also valued in pawns. The most common way valuing pieces is as follows: Pawn = 1, Knight = 3, Bishop = 3, Rook = 5, Queen = 9 [18]. This means that a knight is worth

3 pawns. In chess the value of piece can change depending on the position that they are in and the stage in which the game is in, i.e. start game, middle game, end game. Larry Kaufman produced this as more accurate values for the middle game: Pawn = 1, Knight = 3.5, Bishop = 3.5, Rook = 5.25, Queen = 10 [18]. This is not massively important for my Ai as it only looks at one stage of play.

Some positions on the board are more favourable than others, for example it is more valuable to have a knight in the centre of the board than to have them on the sides. For this we use piece-square tables. Piece-square tables are essentially squares that store how good or bad a position is for a piece to be standing there [19]. Again, using a knight for example, the position of C3 would return +0.1, and the position of A2 would return -0.4. The squares for white and black are essentially the same, just flipped, and for black you would need to subtract the value. Using piece-square tables helps the chess Ai keep some structure on the board if there are no taking moves to be played. There are some general rules for each piece that the Ai will then follow. It will attempt to advance pawns up the board and will discourage the Ai leaving central pawns in the middle of the board [19]. For knights it will attempt to keep them in the centre and avoid the edges. For bishops it will attempt to avoid the corners and edges. Squared b3, c4, b5 and d3 are also more favourable than others [19]. Rooks are encouraged to be centralised and avoid the a and h flanks [19]. Queen is a bit different as there are technically no good squares, but only bad squares, so the queen is encouraged to stay out of the corners and more central. This is an example of a piece-square table, this image was taken from chessprogramming.org:

```
// knight
-50,-40,-30,-30,-30,-30,-40,-50,
-40,-20,  0,  0,  0,  0,-20,-40,
-30,  0, 10, 15, 15, 10,  0,-30,
-30,  5, 15, 20, 20, 15,  5,-30,
-30,  0, 15, 20, 20, 15,  0,-30,
-30,  5, 10, 15, 15, 10,  5,-30,
-40,-20,  0,  5,  5,  0,-20,-40,
-50,-40,-30,-30,-30,-30,-40,-50,
```

Figure 1: White knight piece-square table.

Another key part of an evaluation function is checking for checkmate. For this I made it so that if the evaluation function detects the board is in checkmate, it will return a high number for white, or a small number for black. This mean that when the minimax algorithm detects checkmate, it will always know to move towards it.

I also did a lot of research on other parts of chess evaluation. For example, I looked at pawn structure, king safety, centre control, mobility, and game phases. I even implemented some of these, but ultimately, I noticed that they did not affect the moves that the puzzle solver plays, so I removed them from the code.

1.2.6 Puzzle Generator

There are three main ways in which to generate a chess puzzle. One way is to use a data set of games, and search for games that satisfy the requirements, and then role the last moves back until the desired length of puzzle is created. For example, I could use an API such as Lichess [20] and

search through the games they had on the database. I decided not to use this approach as I would have no use for my puzzle solver then, I wish for my project to create and solve the puzzle itself and not use outside sources. Another way to generate puzzles is through machine learning techniques. An artificial neural network can be trained on games in a database. This network can then be given a set of requirements and create a chess puzzle based on the learning data. I decided not to do this as I believe I did not have enough time to implement this along with the puzzle solver. The third way and the way in which I am using, is to do it randomly. This way is the simplest and involves moving pieces randomly until the requirements are met. The main issue with this is the bored will not look natural due to its random nature, there are ways to combat this which I will talk about later.

1.3 Project Objectives

Project Objectives:

1. The program will create a random puzzle varying in difficulty.
2. The program will solve the puzzle.
3. The program will display the puzzle to the user.
4. The program will allow the user to attempt to solve the puzzle.
5. The program will allow the user to view the solution.

1.3.1 Use Case Diagram

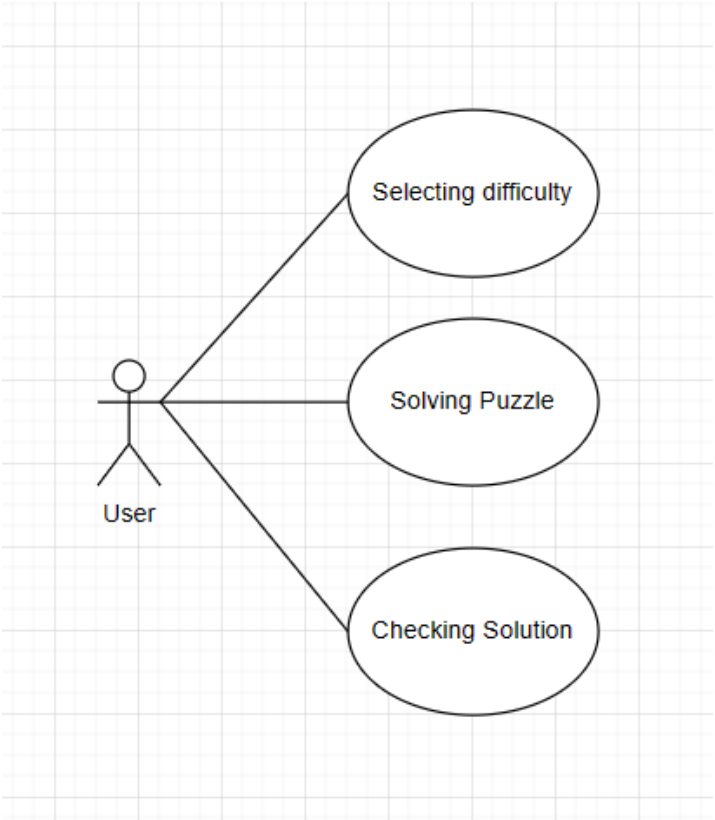

Figure 2 Use Case diagram.

1.4 Process

Before starting this project, I considered various methodologies before deciding on one to use. I researched agile methodologies such as Scrum, Kanban and feature driven development. I also investigated doing waterfall too. I am more familiar with a plan driven approach such as waterfall methodology and this would boast the most familiarity. Despite the lack of experience with an agile methodology, I thought this would be a good opportunity to gain experience and understanding into

how an agile methodology works and functions, this will be beneficial for future projects and for in the computer industry.

It was difficult to land on one agile methodology as they all have their strengths and weaknesses. All agile methodologies revolve around a team of developers and constant face to face meeting with each other and with the product owner. This would have to be adapted due to me being a solo developer. Ultimately, I landed on doing sprints. This was the best option for a solo developer and would help me keep on my target. I find sometimes without clear structure or direction I can fall into a rut of not doing work or pushing it away saying, “I’ll do it another time”, using sprints I am unable to do this as I must finish all the work assigned in the sprint.

1.4.1 Planning and Iterations

My project was split into two major parts. The puzzle solver and the puzzle generator. This meant I need a development methodology that would allow me to design my project in the short term and in the long term. I attempted to break the problem down into small steps which could be accomplished within each sprint. This meant I could focus on what the project did after each sprint and how that pushed towards the final project coming together.

I created a document before each sprint that contained the following timeboxes, Plan, Design, Implementation, Testing, Review and Retrospective. Sprints that where just focused on writing of report did not include testing. This allowed me to effectively plan my sprints to minimise time wastage.

The plan time box discusses what will be achieved by the end of this sprint. It looks at the overall progress of the project and what is the best next step.

The design time box discusses how I will implement what is discussed in the plan. It goes over the research that will be needed for the implementation. I discuss the algorithms that will be used.

Implementation briefly discuss what has been implemented in the sprint and how I implemented it.

Testing describes how I tested what has been implemented in the sprint.

The review looks at the code and how the overall sprint went. It discusses what went well and what went badly. This can be difficult as a single developer as mentioned above. It can be exceedingly difficult to find issues within your own work and it is challenging to criticise yourself. It is exceedingly difficult to know whether these reviews where helpful to me or not, I like to think I am very critical of myself meaning I can review my own work with no bias.

After the review I had a retrospective. In this timebox I tried to look for things on which I could improve. For example, I know that the first scrum I was unable to get all the work achieved as I gave myself too much work to do. So, in my retrospective I mentioned how I needed to improve the amount of work I assign myself to keep it within the sprint. Again, the usefulness of the retrospective is heavily reliant on the review, meaning that it may not always be as useful as a solo developer as it would be for a team of developers.

1.4.2 Changes to methodology

As mentioned previously, all agile methodologies focus on frequent meetings between the team and alongside the product owner. These meeting could not happen as there was no product owner and no other team members to hold meetings with. I attempted to simulate these meetings as best as possible, I sat down with myself and tried to figure out what I was going to do that day. I treated my

meeting with my supervisor similarly to a meeting with the product owner which helped keep me on course.

2. Design

2.1 Overall design

The overall design of my project can be split into 3 sections, the puzzle solver, puzzle creator and driver. The puzzle solver will implement the minimax algorithm with an evaluation function as mentioned in section 1.2.3 and 1.2.5 respectively. The puzzle creator will implement an algorithm that will randomly create the puzzle. The driver code is what the user will interact with. I wished to create my puzzle solver using minimax first, which can be tested using premade tests, and then create the puzzle creator second, as this will allow me to test the puzzles created with the puzzle solver. Lastly the main driver code will be implemented to bring both together and allow the user to interact with the program.

2.2 Programming language and libraries.

For this project I decided to use python as mentioned in 1.2.2, more specifically I have used python version 3.12, as this was the most recent version of python when beginning the project. I also used python-chess [7] as mentioned in 1.2.1. Another library that I have used is random [21], this library will be used by the puzzle creator when choosing random moves. The final library used was unittest [22]. This library was used to create unit tests for my project.

2.2 Early design

In my early design I had implemented the minimax algorithm along with an overly complicated evaluation function. When I started designing and implementing my evaluation function, I was creating it as if it were playing from the beginning of the game. My evaluation function included piece evaluation, pawn structure, piece-square tables, king safety, game phases and checkmate and mobility.

The initial design for my puzzle creator was quite simple. It was going to make random moves along with an evaluation function. The evaluation function would take the board and evaluate if it is a valid puzzle by checking the moves to checkmate. If the board were a valid puzzle, the puzzle would then be solved by the minimax algorithm. These moves would be saved and then compared against the users moves to check if they played the correct move or not.

2.3 Changes and later designs

Quickly I noticed that evaluation function was too computationally expensive and unnecessary. I noticed that as my algorithm only needs to be able to solve puzzles, most of the is unnecessary and only works in real game situations. The most important evaluation for the evaluation function to make is to check for checkmate. I decided to keep piece evaluation and piece-square tables, as these give the algorithm enough structure to be able to play the game at a decent level and solve puzzles without being too computationally expensive.

The design for the puzzle creator has changed too. I noticed when a puzzle was created using random moves, the board did not look natural, it did not look as if it was played by two players. To remedy this, I altered so that it my algorithm chooses a random opening to play, and then the minimax algorithm will play against itself for so many moves before attempting to play random moves. This way the board will regain some structure before performing random moves.

Another change that was needed was when checking for checkmate, the way I was doing it was employing the puzzle solving algorithm, which played moves depending on the difficulty, for example if the difficulty was 3, meaning the moves needed to check mate was 3, the evaluation function would play 3 moves using the minimax algorithm, if the board was then in checkmate, the board was a valid moves and the last moves where popped. This is a very inefficient way of calculating if the puzzle is valid. A faster way to implement this is to just allow the minimax algorithm to play moves until checkmate is found after the random moves are played. This way the last moves can be popped instead of checking for checkmate after every move.

2.5 The Minimax algorithm

As mentioned previously I chose to use the minimax algorithm with alpha beta pruning. The minimax algorithm is a decision-making algorithm which is perfectly equipped for a chess Ai as it explores every possible move, and returns the best move, if the opponent plays optimally too. It works recursively to build the game tree, where each node is a possible move that the player or the opponent could make. As mentioned in 1.2.3 it alternates between maximising and minimizing players. For each node the minimax algorithm calls an evaluation function. This function will return how favourable a particular game state is for the maximising player. This function is vital to the minimax algorithm as this value is how the algorithm decides what move to take. To optimize my minimax algorithm, I also implemented alpha-beta pruning. This is where a branch of the tree is cut off if we know it guaranteed to return a worse result than a previously explored tree, thus there is no need to explore it. This helps reduce the number of nodes in the tree. This is spoken about in more detail in section 1.2.3.

Here is a flow diagram of the code:

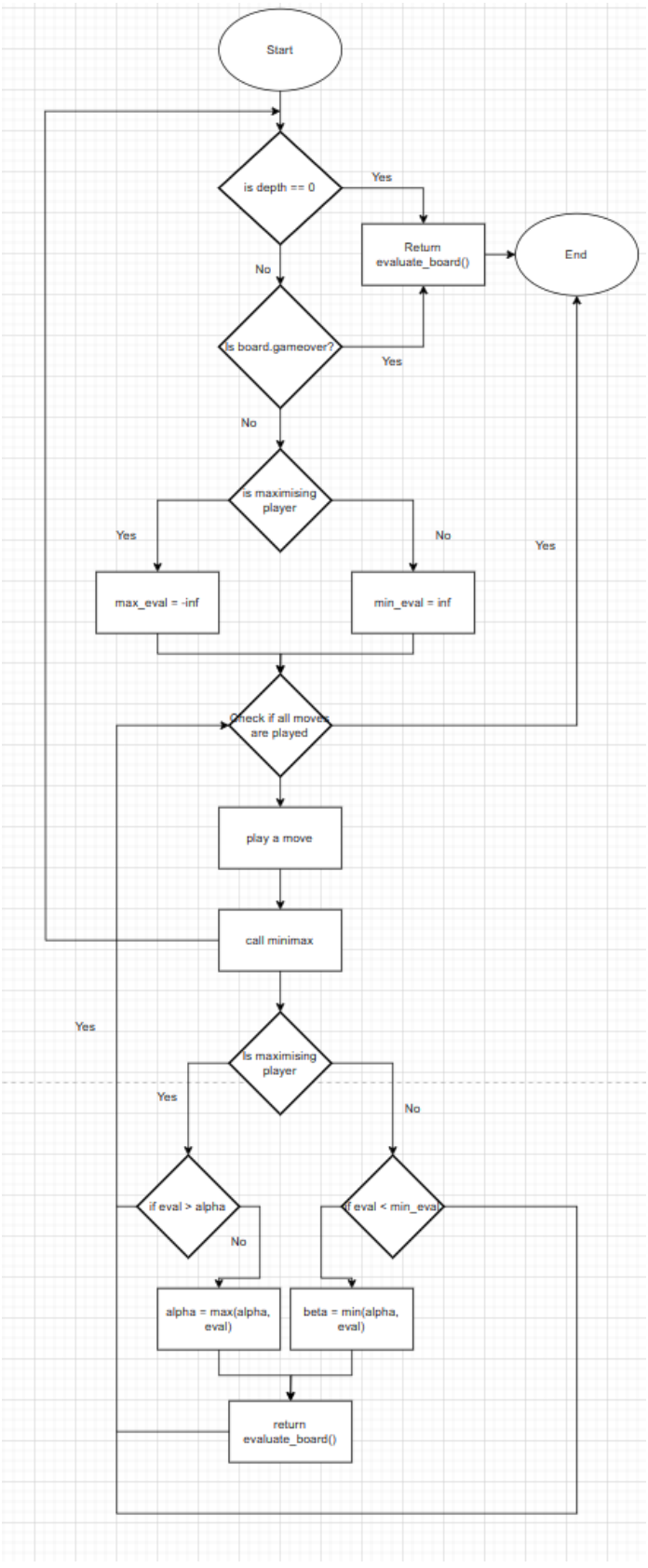

Figure 3 - Alpha-beta flow diagram.

2.6 Puzzle Creator

The puzzle creator is quite simple, and it utilises the minimax to choose correct moves for one side while random moves for the other side. It has a 2D list of openings from which it will randomly choose. After the opening is played, the minimax algorithm will play as both sides for five moves each. These two features alongside each other will bring the board to a nice middle ground, where most of the pieces will be developed, giving the board a certain amount of structure before the game begins to play itself. After this, the algorithm will play five random moves each. This is to create some randomness on the board to create the puzzle. As mentioned previously, there is an average of 31 to 35 moves for any given position, this means there should be a wide variety of boards after these ten moves. After these moves the board is in a position where we can start to push for an end game. To do this the minimax algorithm will play one side, while the other side will play randomly. The minimax algorithm will play the best move at a given time; this means that it will be able to push for end game versus the random player. This will carry on until checkmate is found. Then the algorithm will pop the last moves depending on difficulty. For example, difficulty two means checkmate in two moves, so four moves will be popped (two for player, two for opponent).

Choosing the numbers for how many moves the minimax and the random algorithm was all trial and error. I noticed if the minimax algorithm played too many moves, it took too long to create a puzzle. I also noticed that if too many random moves where played the board looked too unnatural, and it took a long time for the minimax algorithm to find end game as the board was so random. So, I chose five minimax moves each, and five random moves each to progress the game enough but to not run into any of these issues.

2.7 Main Driver code

The main driver code design is quite simple. This is the code that the user will interact with. It will ask the user for a difficulty that they wish to play, and it will then call the puzzle creator class to create the puzzle. The puzzle will then be displayed to the user for the user to attempt to solve. If the user does not solve the puzzle in the correct number of moves, they will be shown the solution. After they are shown the solution or they successfully complete the puzzle, they will be asked if they want to continue, quit, or change the difficulty.

2.9 Structure

The overall structure of the project is quite simple, the main driver code needs a puzzle creator, the puzzle creator needs a minimax class, and a puzzle solver needs an evaluation class. I decided to create an individual class for each one of these segments. I created the evaluation class when it was much more complicated than it is now, it could now be refactored to be including within the minimax class. Another reason I had originally had the evaluation in its own class was because I was going to use it within in the puzzle generator, which now I know was not the case. Structuring my project like this made sense to me and it will mean that no data is lost between the classes.

One thing to note is that python does not have private or public variables and methods, everything is public. For ease of reading any method or attribute that I wished to be private, I began the name with an underscore [23], this is common practise for python programmers.

2.9.1 Class Diagram

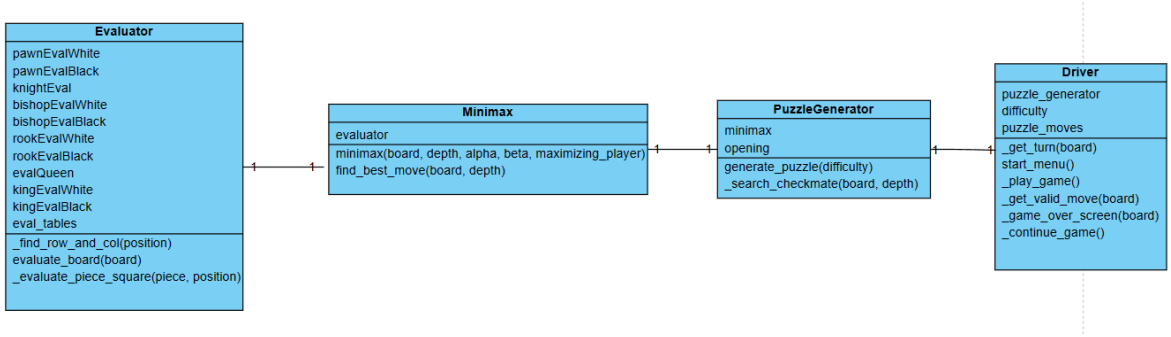

Figure 4 - Class diagram.

3. Implementation

3.1 General Approach

My project can largely be broken down into three sections as described in the design section. I decided to implement the puzzle solver with minimax first. This made the most sense to me as it is the backbone of my project, it will aid in the solving and the creating of puzzles so is needed before the puzzle creator is implemented. The main driver code is implemented last as it relies on the other two sections being implemented completely before it can be implemented.

A design was created each scrum which I tried to follow as closely as possible.

3.2 Implementation 1 – basic minimax with basic evaluation function.

3.2.1 Design

The first design was to implement the minimax algorithm along with a basic evaluation function. I wished to get the algorithm searching and returning correct values from an evaluation function that simple searches the board and returns the sum of the pieces using the values discussed in section 1.2.5.

3.2.2 Implementation

I initially began by creating my evaluation class and evaluation function in the class. The initial implementation of the function was quite simple, it looks over the game board, checking the pieces colour and type, and then adding or subtracting (add for white, subtract for black) to the evaluation number, this value is then returned. I decided to create the evaluation function first as I knew it was needed by the minimax algorithm. I initially though my evaluation function did not work, as when I tested it with a default board (all the pieces in the starting position) the evaluation function returned 0. I quickly realized that this was correct as both sides had the same number of pieces on the board meaning 0 was the correct value.

Unlike the evaluation function, the initial implementation of the minimax function was a bit more complicated. It was my first time writing a search of this magnitude. Due to my research, I knew the minimax algorithm was recursive, so I knew it needed a base and a base case. The base case for most recursive searches is when the depth is 0, meaning it reached the end of the depth specified. There is also a second base case for my minimax algorithm, which is when the board is in a game over state. This means if it finds checkmate, there is no point searching further down the tree, obviously. The next obvious step is to implement the maximizing and minimizing player section. This is just a simple if statement from a value passed into the function, what changes is on the recursive call. If it is the maximizing players turn, the next minimax call will pass false as an argument, making it the

minimizing players turn. On the other side if it is the minimizing players turn, the next minimax call will pass true as an argument, making it the maximizing players turn. This is how you get the flip between the maximizing and minimizing player. My algorithm then loops over the possible legal moves and pushes them to the board. It then calls itself again, passing the new board with the new move on. Also, on this call the depth has 1 subtracted to it, otherwise it will never reach the base case. After the recursion has finished and the boards have been evaluated, the moves are then popped from the board to return to the original game state. Depending on the evaluation of the board, the highest evaluating move will be returned. As the evaluation function is very basic, the moves where a little out of the ordinary, but I expected this and knew why it was happening so was not an issue.

3.2.3 Review

Overall, this first implementation was a success, while the evaluation function is not overly complicated, the main aim was to get the minimax algorithm searching correctly which I managed to do. I noticed that the algorithm was quite slow and would take a long time to search a depth of 3 plus. This will be remedied in the next iteration.

3.3 Implementation 2- Alpha-beta pruning and more complex evaluation.

3.3.1 Design

As noted in the review of the last section, the algorithm took a long time to search above depth 3. To improve this in this implementation I implemented alpha-beta pruning.

I also wished to improve the evaluation function; in this implementation I implemented piece-square tables as discussed in section 1.2.5.

I also wished to implement returning a larger value when checkmate was found.

3.3.2 Implementation

Alpha-beta pruning was relatively easy to implement as it utilises the standard minimax algorithm with some changes. Essentially it will break out of the loop of pushing moves onto the board if we know the moves are going to be worse than previously evaluated moves. To do this we need to two new values called alpha and beta. These values are past into the function when it is called. Alpha stores the minimum possible value that the maximizing player can obtain, and beta stores the maximum value the minimizing player can obtain. Alpha is set to -infinity and beta is set to positive infinity. The alpha value is updated using the max of the current alpha value and the value returned by the next minimax call. Beta is updated using the min of the current beta value and the value returned by the minimax call. If beta is ever bigger than beta, we know the opponent will be able to get have better moves with we explore this path, thus we no longer need to explore down it.

Piece square tables where quite easy to implement. I used the values specified on chessprogramming.org [19]. The squares themselves are 2D lists, where each row and column are a position on the board. Each piece type (i.e. white bishop) has its own square with unique values for the positions. I also created a 2D dictionary which holds the colour and then the correct evaluation table. This means looking up the correct table is much easier, it can be achieved with one line, eval_tables[piece colour][piece type] instead of having a long if statement that checks every colour and piece type. The only other function I needed was to translate the position of the piece into coordinates. The chess engine stores positions from 1-64, but my piece tables where in a 2D list, meaning I needed to translate the number into a position. The piece square function is then called by the evaluation function, and the value is added for white and subtracted for black.

Within the evaluation function, I also added then when checkmate was found, if the checkmate was for white 90 was added to the evaluation value, and for black -90 was added to the evaluation function. This meant that if checkmate were found, this would always be the biggest or smallest score possible.

3.3.3 Review

This implementation was an enormous success, I noticed a great reduction in time for the minimax algorithm, being able to search to depth 5 in a reasonable time. The implementation of piece square tables also means the algorithm is now able to make moves that will progress pieces into better positions, this means the algorithm is able to play progressive moves even when no take moves are available.

3.4 Implementation 3 – Improved evaluation function

3.4.1 Design

The design for this implementation was to improve the effectiveness of the evaluation function. I wanted to keep improving the evaluation function so that the minimax algorithm would play better moves. The plan was to implement king safety, by checking if it has pawn protection in front of it, pawn structure, by checking for things like connected pawns and double pawns. I also implemented checking for mobility of pieces.

3.4.2 Implementation

To implement king safety, I basically just needed to check in front of the king to check if there is a pawn barrier in front of the king. It checks directly in front of the king, and the two diagonals in front of the king too. If there is a pawn barrier, the evaluation number was increased by 0.5 for white and -0.5 for black for each pawn in the barrier.

To implement mobility, I simple counted the number of possible moves that each player had. In chess it is a general rule that the more legal moves you could play, the stronger your position [24]. For each legal move, I would add 0.1 for white, and take away 0.1 for black.

To implement pawn structure, I began by checking for doubled pawns. A doubled pawn is a pawn that is stuck on the same the same file as another pawn of the same colour [25]. I implemented this by simply looking at the square in front of the pawn, if there is a pawn of the same colour there, then apply the penalty of -0.5 for white, and 0.5 for black.

To implement connected pawns, I checked for pawns that defend other pawns. For this I created a function that look diagonally forward from the original pawn. If the pawn is defending another pawn, then return 0.1 for white, and -0.1 for black.

3.4.3 Review

While on paper this iteration looks like a success, I also noticed that the evaluation function began to take a little bit longer, this became an issue when used in conjunction with the minimax algorithm because its evaluating so many distinct positions. I also concluded that most of this evaluation was not needed. As at this point in development, I only needed my minimax algorithm to solve puzzles of maximum of 3 depth. Meaning most of this evaluation was not needed. I also noticed that I had only used my minimax on white, and when I tried to use to find blacks best move, it returned strange moves.

3.5 Implementation 4- Major refactoring

3.5.1 Design

Using the issues mentioned previously, I wished to refactor the evaluation function and the minimax function to alleviate these issues. To do this I wanted to create a new function that would be called instead of the minimax function, which would then check what turn it is, and then call the minimax function.

3.5.2 Implementation

The refactoring of the evaluation function was very easy, as I had deemed the king safety, mobility, and pawn structure all useless, they could simply be removed from the function.

To correctly call the minimax function, I created a new function that checked whose turn it is when the function is called. This will change whether the maximising player is true or false for the minimax algorithm. If it is whites turn, then false is passed into the minimax algorithm, so the next moves evaluated are the minimizing players, and if its blacks turn, then true is passed into the function so the next moves are the maximising players.

3.5.3 Review

This implementation was a great success, the removal of parts of the evaluation function improved performance without hindering the effectiveness to find a solution for a puzzle. The minimax algorithm could now be used for black moves, this means going forward I can create puzzles that use white and black without any issues.

3.6 Implementation 5 – Puzzle generator

3.6.1 Design

The initial design for this implementation was to create a puzzle using completely random moves, and then solve it using my puzzle solver. I wanted the game to choose a random opening, and then play random moves on either side until it was deemed an acceptable puzzle through an evaluation function.

I soon realized this was not a suitable way to create a puzzle. The evaluation function took too long to evaluate, and the board looked too random for my liking. I changed the design so that the minimax algorithm will play some moves before the random moves are played, this will give the board a more natural structure.

3.6.2 Implementation

The puzzle generator was a bit finicky to implement, not matter what I tried I could not get it how I liked it. Initial implementation played completely random moves after an opening was played, this made the board, and the moves look unnatural and uneasy to find the best move. To remedy this, I changed it so that the minimax algorithm will play five moves each after the opening is played. This gave the board some more structure before the random moves are played. The algorithm then checks if the game is over, if this is false, then a one side will play random moves while the other side will be played by the minimax algorithm. This allows the randomness while still pushing for checkmate. The random and minimax moves are played until the game is in checkmate. Once the game is in checkmate, the moves are popped from the board depending on the difficulty. For example, if the difficulty is three, then 6 moves will be popped from the board (three for white, three for black).

3.6.3 Review

While the puzzle generator can produce puzzles, they are not entirely how I want them. I wanted the board to feel natural with moves that each side making sense, but this is not the case under current implementation. While it works perfectly for one move to check puzzles, two or three moves start to feel unnatural. While the minimax move will be optimal, the other side is random, so it can be difficult to find the solution to the puzzle as the algorithm will play a completely random move that the user will not be expecting. One method I tried to do to fix this, was have the minimax algorithm play itself until checkmate, but as the minimax algorithm playing itself is so similar in performance, one side would end up taking all the pieces of the other side leaving only the king. This does not make an incredibly fun and enjoyable puzzle to try and solve. Ultimately, I believe the way I have attempted to implement puzzle generator might be nearly impossible. I will discuss this in the critical evaluation section.

3.7 Implementation 6 – Driver code

3.7.1 Design

The design for the driver code is very simple. It simple generates the puzzle, displays it to the user, allows the user to solve it, show the user the solution and the ask the user if they wish to continue.

I also wish for the driver code to have some error mitigation on the user inputs, for example if they accidentally type the move in wrong then the program will ask them to reinput their move instead of crashing.

3.7.2 Implementation

The driver code begins but displaying a welcome message to the user, which is followed promptly by an option to select the difficulty. The user can specify 1 for easy, 2 for medium, 3 for hard and 4+ for custom. The maximum difficulty is 10, so if the user inputs something over 10 then, then the difficulty will be set to 10. This then calls the _play_game function. This function is main function the user will interact with. It calls the PuzzleGenerator class generate the puzzle and to get the solution. The function then loops for the length of the solution, printing the board, and asking the user for their move. The move is checked to see if its valid, if it is not then the user is asked to input another move, this will repeat until a valid move is inputted. When a valid move is added, it pushed to the board. Depending on the difficulty, it will depend on the amount of moves the user has to input. For example, difficulty one, the user will have to input one move, difficulty two is two moves, and difficulty three is three moves. Ones all the moves have been played; the board is checked to see if it is in checkmate. If the board is in checkmate and the puzzle is solved, a message saying “Puzzle solved!:)” is output, if the puzzle is not in checkmate and the puzzle is not solved then a message saying “Puzzle failed:(“ is output. The use is then asked if they wish to see the AI solution or not. If they user input “Y”, then the solution is showed to the user. After the solution is shown the user is asked if they wish to continue or change the difficulty, if the user changes the difficulty or continues, the game will recall _play_game function and start again.

3.7.3 Review

Overall, this implementation was a success, I was able to implement the main driver code and test it for bugs quickly and effectively. This was the final implementation of the program, and this was vital to pull every part of the code together.

4. Testing

4.1 Approach to testing

As I was following an agile approach, I would complete tests during each sprint to make sure that the code was bug free for release. I constantly manually tested my project through development, helping me to find bugs and errors before writing unit tests closer to the end of the sprint. Unit tests were written for each function at the end of the sprint, if any failed occurred these where fixed so that they passed the tests.

4.2 Test planning

Before I carried out each test. I read my code and identified areas that had already been tested and areas that needed to be tested. Each class has its own set of tests, for example the evaluator class has its own set of tests, and the minimax algorithm also has its own set of tests. The main objective for each test is to make sure that the code was running correctly without any errors.

4.3 Manual testing

Manual testing was a big part of the initial testing for each sprint. I will constantly check that values where correct and often give the give the program a wide range of inputs to make sure that the functions were performing correctly.

4.4 Unit tests

Unit testing was an important part of my project, near the end of each scrum after all the code was implemented, I created tests for each function with a range of test data to limit test my program.

4.5 User testing

Unfortunately, I was unable to utilise any user testing in my project. This was due to the fact that the driver code was created last at the very end of the project, meaning that I did not have enough time for me to find someone to act as my user.

After the user had used the program for a little bit, I would ask them to fill in a questionnaire which asked them questions like, was the interface easy to use? Where the puzzles difficult enough? Where there any bugs? Was the algorithm efficient enough? Questions like these would allow me to identify where the interface and the algorithm can be improved.

4.6 Testing the minimax algorithm.

Testing the minimax algorithm needed some work around, I needed to find puzzles to test the algorithm with. I am not good enough at chess to create my own puzzles. Which means I had to use puzzles I found of chess.com and compare the outputted result with the correct result on the puzzle.

4.7 Final test table

This is the final test table that tests all the final features:

Table 1 - Final test table

Test ID	Description	Input	Pass/Fail	Comment.
PS 1	Test that a white rook is evaluated to 0.0 if on square 7	A board with all the pieces in the starting position. The position to search: 7.	Pass.	The returned value is 0.0

PS 2	Test that a Knight is evaluated to -0.4 if on square 57	A board with all the pieces in the starting position. The position to search: 57.	Pass	The returned value is -0.4
PS 3	Test that a knight is evaluated to 0.1 if on square 21.	A custom board with a knight on square 21: "3r4/8/8/8/8/5N2/8/8 w - - 1 1" The position to search: 21.	Pass	The returned value was 0.1
PS 4	Test that a black rook is evaluate to 0.5 if on square 59.	A custom board with a black bishop on square 59: "3r4/8/8/8/8/5N2/8/8 w - - 1 1" The position to search: 59,	Pass	The returned value was 0.5
PS 5	Test that the black queen is evaluated to 0.05 if on square 43.	A custom board with a black queen on square 43: "8/8/3q4/8/8/8/8/8 w - - 0 1" The position to search: 43.	Pass	The returned value was 0.05.
PS 6	Test that a white bishop is evaluated to -0.1 if on square 5.	A custom board with a white bishop at position 5: "8/8/8/8/8/8/8/5B2 w - - 0 1" The position to search: 5.	Pass	The returned value was -0.1
MM 1	Test that when given a puzzle from chess.com, the minimax algorithm can find the correct move. Puzzle type: Mate in two Expected move: "c4d5"	Input the board taken from chess.com: "r1b2bkr/ppp3pp/ 2n5/3qp3/2B5/ 8/PPPP1PPP/RNB 1K2R w - - 0 1".	Pass	The returned value was "c4d5"
MM 2	Test that when given a certain puzzle from chess.com, the minimax algorithm	Input the board taken from chess.com: "r3r3/pp3p1k/6p p/	Pass	The returned value was "h4h3".

	can find the best move. Puzzle type: Mate in one Expected move: “h4h3”	4b3/2BpP2q /PQ1P3P/1P3P2/2 R3RK b - - 0 1”		
MM 3	Test that when given a certain puzzle from chess.com, the minimax algorithm can find the best result. Puzzle type: Mate in two Expected results: “e8e1”, “b1a2”, “e1a1”	Input the board taken from chess.com “4r3/7R/6B1/8 /5k2/1Pb2P2/ 2P3PP/1K6 b - - 0 1	Pass	The returned values where “e8e1”, “b1a2”, “e1a1”
MM 4	Test that when given a certain puzzle from chess.com, the minimax algorithm can find the best result. Puzzle type: Longer defence Expected results: “h8e8”.	Input the board taken from chess.com. “7Q/7p/8/3K2q1/ 2P5/8/7k/8 w - - 0 1”	Pass	The returned value was “h8e8”. As this was a longer defence, further depth was needed to find the best moving meaning it took a long time to find the move.
MM 5	Test that when given a certain puzzle from chess.com, the minimax algorithm can find the best result. Puzzle type: Mate in two Expected results: “f8f2”, “g2h3”, “f1h1”.	Input the board taken from chess.com. “5r2/4R2P/2p1R3 /1pk5/p5B1/2PP2 P1/PP4K1/5r2 b - - 0 1”	Pass	The returned values where “f8f2”, “g2h3”, “f1h1”.
EV 1	Test that the evaluation function returns 0 when a default board is passed in. Expected Result: 0	Default board, where all positions are starting positions	Pass	The returned value was 0
EV 2	Test that the evaluation function returns 1.05 when a black pawn is	Board where starting positions are normal except a pawn has been	Pass	The returned value was 1.05

	removed from square h7. Expected result: 1.05	removed from square h7. “rnbqkbnr/ppppppp1/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1”		
EV 3	Test that the evaluation function returns 1.4 when a black pawn has been removed, a white knight is on e5, a white pawn is on e4, and a black pawn is on d5 Expected result: 1.4	Board where some moves have been played. “rnbqkbnr/ppp2ppp/8/3pN3/4P3/8/PPPP1PPP/RNBQKB1R b KQkq - 0 3”	Pass	The returned result was 1.4
PG 1	Test that when a puzzle is created with a difficulty of 1, there is 1 move to checkmate by checking how many moves are in the move list. Expected result: 2	The difficulty: 1	Pass	The returned value was 2.
PG 2	Test that when a puzzle is created with a difficulty of 2, there is 2 moves to checkmate by checking how many moves are in the move list. Expected result: 4	The difficulty: 2	Pass	The returned value was 4
PG 3	Test that when a puzzle is created with a difficulty of 3, there is 3 move to checkmate by checking how many moves are in the move list. Expected result: 6	The difficulty: 3	Pass	The returned value was 6
Dr 1	Test to see if the get turn function returns	A default board where all the	Pass	The returned values were

	the correct values given a chess board. Expected results: “White”, “Black”	positions are starting positions.		“White” and “Black”.
Dr 2	Test to see if the valid move function works when given a default board and “e2e4” is inputted. Expected Result: “e2e4”	A default board where all the positions are starting positions. The user inputs “e2e4”	Pass	The returned value was “e2e4”
Dr 3	The puzzle is correctly displayed to the user	The difficulty: 1	Pass	The puzzle is displayed for the user to see
Dr 4	The user is able to input a move which is then pushed to the board	The move to play, i.e. “e2e4”	Pass	The move is correctly pushed to the board.

5. Critical Evaluation

5.1 Future improvement.

There are many improvements that could be made to my project, whether it be the puzzle creator or puzzle solver. The main issue with the minimax algorithm is the depth at which to search at. Searching too shallow and you will not find an optimal solution, searching too deep will be computationally very expensive. To combat this problem, you could implement iterative deepening. Iterative deepening performs multiple Depth first searches at an increasing depth [26]. It usually starts at a depth of 0 and continues until a condition is met [26]. It combines the benefits of breadth first search and depth first search, for example it is complete, meaning it guarantees to find a solution if it exists, where a purely depth first search does not. It allows for better time and memory management because the search stops when a time limit is reached, or the solution is found.

Another huge improvement that could be made to the minimax algorithm is the use of transposition tables. Due to most chess algorithms brute force approach, they encounter the same positions over and over again just from a different sequence of moves. This is what is known as a transposition [27]. A transposition table is a database of previously stored performed searches and evaluations. There is no point reevaluating the entire search again, so it is beneficial to remember what the outcome of the previous search in this position was. When the search algorithm encounters a transposition, it can use the transposition table to look up the previous outcome. This saves a large amount of time on the search algorithm, greatly improving performance.

Many newer engines like stockfish [15] and AlphaZero [16] implement an artificial neural network to perform evaluations for their algorithms. Chess evaluations can be overly complex, and trying to evaluate a board linearly can be complicated and not very efficient. ANNs (artificial neural networks) can approximate complex non-linear functions leading to faster evaluation. ANNs can also learn and be trained from previous games. Training an ANN on high level games will allow it to mimic the evaluation of such games, its important to train the ANN on high level games, otherwise it’ll learn to mimic a poor evaluation [28]. Another benefit of this training, the ANN will be able to detect similar positions that it was trained on, meaning it will be able to detect a move faster than traditional linear

evaluation. But most importantly, due to their ability to learn ANNs excel at pattern recognition. This means that you can train you ANN to spot patterns that is not possible, or extremely difficult in a standard evaluation function.

5.2 Analysis and research

My project is very research heavy at the beginning. The first few weeks dedicated purely to research and understanding what I needed to achieve in this project. This stage was a huge success, luckily for me, chess AI has been a very large topic for AI development since the 1940s. This meant that there was a huge amount of research and documentation for me to read and gain a good understanding of how to build my project. It also helped put the project into scale for me, when I entered the project, I wished to create a chess playing AI that could play chess to a high level and could rival other chess engines. The research taught me that I was not going to be able to create an Ai that could rival stockfish within such a short time by myself. Chess is very complicated, especially computer chess, which would require a whole team to be able to build an AI as good as I wanted to. This helped clear my mind and put the project into perspective.

This allowed me to create a project that was had realistic requirements, instead of trying to create the best chess AI. This is especially important due to the small amount of time that could be dedicated to development. If I had more time, I would be able to implement more features.

Overall, the research and analysis of this project was a huge success and allowed me to create a strong base of information for me to carry on to the other parts of the project. It allowed design and implementation to run much smoother as I had the knowledge to fix errors that occurred.

5.3 Design

Due to the large amount of research and analysis I was able to perform, the design section of this project was simple and effective. The minimax algorithm is largely considered the best algorithm for a chess Ai, meaning I could research it deeply making design quite simple. I also had a lecture on it in my “Artificial intelligence” [29] module in year 2, meaning I already understood how it worked. Due to the agile nature that I had adopted, design for each section was done just before implementation, I believe this was a good approach as it enabled the design to be fresh in the mind for implementation, meaning that I didn’t have to go back and keep reading a design document.

The design for the puzzle creator was more challenging and underwent a more changes compared to the minimax algorithm. The way I wanted to go about it had far less research, meaning the design was more up to my own interpretation than industry standard. Having this higher level of freedom was nice and challenging at the same time. It meant I could design the algorithm how I wanted, but there was no guarantee that said design was going to be good. Ultimately, I believe the design was not good enough, and I should approach it in another way, I will discuss this in more detail in section 5.4.

Despite this, there will always be a debate on whether a design is good or not, especially as people opinions differ. I attempted to keep design simple and easy to understand, eventually stripping details I deemed unnecessary.

5.4 Implementation

The implementation of the minimax and evaluation function was a success. There is room for improvement which I mentioned in section 5.1, but ultimately it did the job I wanted it to do within

the time frame. I believe the code is written to a high standard and is easy to read, maintain and build upon.

The implementation of the puzzle generator was not successful. The work I managed to accomplish was not how I wanted it to be. The board will still feel unnatural due to its random nature. There is also a high possibility of bad puzzles being created, for example if the minimax algorithm decides to take all the pieces of the opponent leaving only the king, this is not a good puzzle. There is a possibility that the way I decided to implement this is nearly impossible. I do not think there is a way to create a puzzle randomly while also making it look natural, its just not possible. I found a reddit thread close to the end of the project that seems to think this way of implementing it is impossible [30]. Many of the comments suggest using previous games to create the puzzle, which is one format I looked into, but ultimately decided against as I wanted to use my minimax algorithm to solve the puzzles. I know reddit is not necessarily a good source, but if many of the comments are saying that it is difficult or near impossible to program then it gives an insight that maybe this is the wrong approach or needs more research and time to properly implement.

I also wished to have a basic user interface for which the user can play the game through. Unfortunately, I ran out of time to implement this feature.

5.5 Testing

While overall my testing was thorough and evaluated all possibilities, there was definitely some improvement to be made in the way I tested. Firstly, as mentioned in section 4.5, I wish to have implemented user testing once my project was coming to an end, unfortunately I did not finish in time meaning I had no time for user testing.

I also believe I could have kept better track of my testing. I did not keep any test documentation which would have helped a lot in future weeks, as I would have known what had already been tested instead of having to read through all my unit tests. It also would have helped on the write up.

It also would have been better if I had a clearly testing strategy. My strategy was to just simply create tests at the end of each sprint. If I had planned each testing session better, my testing may have been more efficient and effective.

5.6 Methodology

The methodology that I landed on was using scrums. This means I had a set plan for each scrum and held meetings and reviews at the end of each sprint. This helped me to keep a clear direction on what I was doing, what I had accomplished, what needed improvement and future development. Given the agile nature of scrums, it focusses on having working code at the end of each sprint. This focus on code instead of documentation allowed me to focus on producing high quality code that was tested and refactored before the end of each sprint.

Despite overall being a success, there was still some elements of scrum and agile development that proved to be more hindrance. The idea of holding meetings daily and at the end of each scrum was a huge pain to have to try and simulate being a solo developer. There was no product owner to communicate with, was I supposed to act as the product owner? In the end I treated my meetings with my supervisor as a replacement for this, this helped me keep a clear goal in mind. Trying to simulate a normal meeting would be pointless, as there would be no information to share between colleagues because there is only one person doing this project. I did not attempt to perform such meeting for this reason. Instead of this, I attempted to have a “review meeting” instead of a standard team meeting. Ultimately whether this was useful or not is up to interpretation, it helped a little bit and made planning for the future easier. Reviewing you own work especially your own code can be

challenging, especially as you believe you have done everything correctly but there could be a huge glaring issue that another developer might notice but you will not.

It can be hard as a solo developer to remain focused and motivated on your project. When you work as part of a team you will always have your supervisor or another colleague to push you along and help when you get stuck. When solo developing, you can find yourself getting bogged down in errors and it can be difficult to pull yourself out of these ruts. This means a high level of discipline is required to stay focussed and motivated on the project.

Overall, I believe choosing an agile methodology was correct as it enabled me to learn much about how the process worked in practise and how it would be implemented in a real-world scenario. While it was annoying to have to work around the lack of meetings and supervision, I ultimately believe holding a scrum each week helped keep me on track and helped me produce quality code within the time scale. Using scrum meant that I had to achieve the work I had designated for that scrum instead of pushing it back which could have been possible with other methodologies such as waterfall.

5.7 Final reflection

Overall, I believe this project has been a success, while some elements have been missed, and there is definitely room for improvement on many aspects, it has been a very good learning and developing experience for me. It was my first time attempting to work in an agile way, which can be difficult to learn and adapt to. This project has also taught me a new level of professionalism. It has taught me properly how to analyse a problem and where to spend more time and when time can be cut short. Many of the previous work in my university career I could slap together a week before its due, I knew this would not be the case for this project which helped me stay motivated throughout.

Despite this there are still some improvements that I could make to the way I work. I spend too much time on research and analysis, especially on the puzzle creator. I spend a lot of time researching fields which turned out to be dead ends, leaving not very much time for development and implementing my own way of creating a puzzle. While my programming skills are good, that does not matter when you leave yourself too little time to implement a key part of your project, you could be the best coder in the world, but give too little time, then it just does not matter.

In conclusion, I am still very pleased with my project, the work I managed to accomplish and lessons I learnt along the way. I really enjoyed my project, and I wish to continue it on outside of university and continue to develop it into something better.

Appendix A Third Party Code and libraries

python-chess [7] **–** A chess library for python. Library was used as a chess engine, it was used to generate the board, legal moves and play these moves. The software was used without modification. The library was used under the GPL 3 license.

random [21]**-** This library implements pseudo-random number generators. Used to select random moves from a list of moves. The software was used without modification. The library was used under the Python Software Foundation License.

unittest [22]**-** A unit testing framework for python. Used to create unit tests. This software was used without modification. The library was used under the MIT license.

Appendix B – Example scrum report:

Plan

This two-week period will be the initial programming and testing of the puzzle solver (minimax with alpha beta pruning). I wish to have the minimax algorithm correctly implemented alongside a detailed evaluation function that will correctly assist the minimax algorithm in finding the correct move. To begin with I wish to implement the evaluation function first and then my minimax algorithm.

Design

My first method to implement is the evaluation function. This function will be quite complicated and needs a lot of research to which evaluations will be used. Currently I am going to implement piece value of each side, king protection and mobility.

I am also implementing the minimax algorithm. This is going to search all the possible game states and alongside the evaluation functions the algorithm will be able to choose the best move.

I will also be implementing alpha beta pruning with the minimax algorithm. This will help limit the depth of the algorithm helping to save time on the search.

Implementation

I have implemented two classes. One being an evaluator class and the other being the minimax algorithm. The Minimax class simply implements the minimax algorithm and nothing else. The Evaluator class contains all the evaluation functions that the minimax algorithm will call to evaluate the board.

Testing

Unfortunately, due to time restrictions, no testing has been conducted. The next scrum will be for testing and tweaking of the algorithm.

Review

This sprint has been a success and a failure at the same time. While I have managed to get a large amount of work done and the quality of the work is high, I have not got any testing done due to running out of time. Unfortunately, this will push my plan back and I will need to spend the next sprint testing and reviewing the code.

Retrospective

Despite not getting the timings correct for this sprint, it has taught me how to plan better and how to divide my time better. This will mean that future sprints should better planned, and I will be able to get all the work done in the time. While the timing may have been bad, having this sprint has pushed me to work hard and produce quality software. I believe that giving myself a strict time frame has helped me produce more software than if I was not using sprint ideology.

Bibliography

[1] "Computer chess," Wikipedia, 10 April 2024. [Online]. Available: https://en.wikipedia.org/wiki/Computer_chess#:~:text=1950%20%E2%80%93%20Claude%20Shannon%20publishes%20%22Programming,of%20chess%20(dubbed%20Turochamp)..

Wiki page that gives a history into chess. Also includes brief discussion on algorithms

[2] P. A. Freiberger, "Atanasoff-Berry Computer," Britannica, [Online]. Available: https://www.britannica.com/technology/Harvard-Mark-I.

Website that describes the history of the Harvard-Mark-1 Computer

[3] "UNIVAC I," Wikipedia, 12 April 2024. [Online]. Available: https://en.wikipedia.org/wiki/UNIVAC_I.

Wiki page that gives the history and functionality of the UNIVAC_1 computer.

[4] "CS31920," Aberystwyth University, [Online]. Available: https://www.aber.ac.uk/en/modules/deptfuture/CS31920/AB1/. [Accessed 27 April 2024].

Webpage that describes the module information for Advanced Algorithms as Aberystwyth University

[5] "CS26520," Aberystwyth University, [Online]. Available: https://www.aber.ac.uk/en/modules/deptfuture/CS26520/AB2/. [Accessed 27 April 2024].

Webpage that describes the module Information for Artificial Intelligence at Aberystwyth University.

[6] "Branching Factor," Wikipedia, 18 March 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Branching_factor&action=history.

Wiki page that briefly describes branching factor. Includes branching factor for games like chess and Go

[7] N. Fiekas, "python-chess: a chess library for Python," 01 01 2024. [Online]. Available: https://python-chess.readthedocs.io/en/latest/.

Official documentation for the Python-chess library. Helpful for learning how the chess library functioned.

[8] Cloudjos, "chestnut 1.0.0," 11 March 2023. [Online]. Available: https://pypi.org/project/chestnut/.

Pipy page to the python library chestnut.

[9] A. Ryabtsev, "8 Reasons Why Python is Good for AI and ML," 26 March 2024. [Online]. Available: https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/.

A webpage which describes why python is a good language for AI and machine learning. Useful for when choosing a programming language.

[10] M. Walburg, “10 Best Cross-Platform Programming Languages,” 17 August 2021. [Online]. Available: https://binarapps.com/10-best-cross-platform-programming-languages/#:~:text=Is%20Python%20a%20cross%2Dplatform,and%20even%20Android%20and%20iOS..

Webpage that describes that python is a cross platform language. Useful for when choosing a programming language

[11] “Minimax,” Wikipedia, 2024 April 7. [Online]. Available: https://en.wikipedia.org/wiki/Minimax.

Wiki page that describes in detail the minimax algorithm. Very useful for initial research and implementation.

[12] “Alpha–beta pruning,” Wikipedia, 24 April 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Alpha%E2%80%93beta_pruning&action=history.

Wiki page that describes in detail alpha-beta pruning. Very useful in research and implementation.

[13] “Monte Carlo tree search,” Wikipedia, 15 April 2024. [Online]. Available: https://en.wikipedia.org/wiki/Monte_Carlo_tree_search.

Wiki page that describes in the detail the Monte Carlo search tree. Useful for when comparing different algorithms.

[14] S. B. Yifan Jin, “CME 323, Report”.

A report that details distributed algorithms and optimization. Important for looking up time complexity of the Monte Carlo search tree.

[15] “Stockfish,” Stockfish, [Online]. Available: https://stockfishchess.org/. [Accessed 02 05 2024].

Home page to the chess engine Stockfish.

[16] “AlphaZero,” Wikipedia, 24 March 2024. [Online]. Available: https://en.wikipedia.org/wiki/AlphaZero.

Home page to the chess engine AlphaZero.

[17] “Evaluation function,” Wikipedia, 13 February 2024. [Online]. Available: https://en.wikipedia.org/wiki/Evaluation_function.

Wiki page that described evaluation functions and how it is used along side search algorithms.

[18] “Chess piece relative value,” Wikipedia, 5 April 2024. [Online]. Available: https://en.wikipedia.org/wiki/Chess_piece_relative_value#:~:text=Piece%20valuations%20have%20no%20role,9%20points%20to%20a%20queen..

A wiki page that described the value of chess pieces and how they are used in an evaluation function. Useful for when implementing my evaluation function.

[19] "Simplified Evaluation Function," chessprogrammingwiki, 23 February 2021. [Online]. Available: https://www.chessprogramming.org/Simplified_Evaluation_Function.

Wiki page that describes a simple evaluation function. Was useful in getting values for the piece square tables.

[20] "Lichess API reference," Lichess, [Online]. Available: https://lichess.org/api.

Home page for the Lichess API, useful for research into alternative methods.

[21] "random — Generate pseudo-random numbers," Python, [Online]. Available: https://docs.python.org/3/library/random.html.

Complete documentation for the random library. Useful to know what functions to use.

[22] "unittest — Unit testing framework," Python, [Online]. Available: https://docs.python.org/3/library/unittest.html.

Complete documentation for unittest framework.

[23] A. Sumich, "Class Variables, Attributes, and Properties," diveintopython, 18 04 2024. [Online]. Available: https://diveintopython.org/learn/classes/variables#:~:text=In%20Python%2C%20private%20variables%20are,the%20variable%20or%20property%20name..

Webpage teaching the basic of python class syntax. Was used when needing to know what to name a private method.

[24] "Mobility," chess programming wiki, 2 July 2021. [Online]. Available: https://www.chessprogramming.org/Mobility.

Wiki page describing mobility in chess and how it is used in an evaluation function. Useful for when implementing my evaluation function.

[25] "Doubled Pawn," chess programming wiki, 27 June 2020. [Online]. Available: https://www.chessprogramming.org/Doubled_Pawn.

Wiki page describing doubled pawn in chess and how it is used in an evaluation function. Useful for when implementing my evaluation function.

[26] "Iterative deepening depth-first search," Wikipedia, 24 November 2023. [Online]. Available: https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search.

Wiki page that describes how iterative deepening works. Useful for looking at methods to improve my algorithm.

[27] "Transposition Table," chess programming wiki, 28 April 2022. [Online]. Available: https://www.chessprogramming.org/Transposition_Table.

Wiki page that describes how transposition tables works. Useful for looking at methods to improve my algorithm

[28] "Neural Networks," chess programming wiki, 12 March 2022. [Online]. Available: https://www.chessprogramming.org/Neural_Networks#:~:text=ANNs%20in%20Games,ordering%2C%20selectivity%20and%20time%20management..

Wiki page that describes how neural networks work in chess. Useful for looking at methods to improve my algorithm and future improvements.

[29] M. Wilson, *Uninformed Search,* Aberystwyth, 2022.

Lecture by Myra Wilson which describes the minimax algorithm.

[30] aristo999, "Create chess problems with python," Reddit, [Online]. Available: https://www.reddit.com/r/learnpython/comments/o5zk72/create_chess_problems_with_python/. [Accessed 2024 April 24].

Reddit page that discusses how it is difficult to create a puzzle generator. Useful for when evaluating my puzzle generator.