BSc (Hons) Computer Science

A Beginner Friendly Java Chess Environment with In-game Assistant

By Vincenzo Scialpi-Sullivan
Supervised by Dr. Amna Eleyan

Table of Contents

Table of Figures

Table of Tables

Table of Equations

Declaration

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work.

27/04/2018

X Vincenzo Scialpi-Sullivan

Vincenzo Scialpi

Signed

Abstract

Currently in the world of virtual chess simulations, beginner players are neglected. The popular chess solutions expect the beginner player to learn through academic sources and provide no real in-game support. This project aims to fill this void by looking at effective non-chess intuitive User Interfaces and their support systems, deciphering what makes them so effective and applying those fundamentals to a Java virtual chess game environment. The chess game was created with a Java swift interface, and an Alpha-Beta pruning chess engine algorithm. The program was used by 7 evaluators with positive feedback given overall. The product has been considered successful however, future work can be undertaken to further explore in-game chess aids or improve the user interface in general.

Acknowledgements

I'd like to thank: my supervisor Dr. Amna Eleyan for her aid in completing this project, the people that evaluated my product for their appropriate & honest feedback, and my University for providing me with the knowledge and resources to complete this project.

1. Introduction

1.1 Project Background

Chess, like many other board games is a Strategy game. A game in which the users have a plan of action designed to achieve a long-term or overall aim. In the case of chess each player is aiming to win by checkmate. To place an opponent in checkmate, the opponent's king must be placed in an inescapable check, where it is being attacked by a piece. Whilst victory through checkmate is the main aim, if too many pieces have been lost by the player, a draw can still be salvaged from a losing game. There are many ways in which a player can force a draw; the player must use their pieces cunningly to achieve this (World Chess Federation, 2014). Over the years, chess has moved from a physical board to a virtual one, delivered through an application or a web service. This has tasked the game creators with making the most competitive, best performing chess engines, and creating the most user-friendly interfaces for their product.

Terminology

Chess Game – Once known as "checker", chess is a strategy game played with certain pieces on a special board (Chisholm, 1911).

Interactive Game – A game which employs two way communication between the player and the game itself. Both parties output information to each other.

User friendly game – A game which makes it easier for novices to play. Graphical User Interfaces are considered to be the most user-friendly interface.

Horizon Effect - The Horizon Effect is caused by the depth limitation of a search algorithm and manifests when a negative event is inevitable but delayable. Because only a partial game tree has been analyzed, it will appear to the system that the event can be avoided when in fact this is not the case (Chess Programming, 2017).

1.2 Aim

The Chess Game project aims to create an offline Java chess application which provides support during a match to better the user's decision making skills, an accessible Graphical User Interface to provide a comfortable and intuitive environment, and the design and implementation of an effective chess engine.

1.3 Objectives

To achieve the aim specified above, certain objectives must be met. These objectives are:

- Research existing chess simulation solutions and determine the positive and negative aspects of each.
- Research existing chess engines algorithms & criticise.
- Research game design fundamentals & how to keep user engagement high within a game environment.
- Design basic chess game interface & menu screens.
- Implement the basic chess environment in java IDE.
- Design Chess engine algorithm with the intention of a varying difficulties.
- Design user friendly 'Chess Aid', using research of important chess strategies.
- Implement designed AI opponent, with multiple stages of difficulty.
- Implement chess aid program.
- Present program to participants of the course and receive feedback in the form of pre-made questionnaire.
- Evaluate the chess algorithm and the chess aid's effectiveness in providing an engaging and improving experience for user.
- Implement Amendments based on participant feedback & finalise chess program.
- Produce final report & documentation.

1.4 Motivation and Contribution

The motivation for this project is the apparent lack of effective beginner-oriented chess software. Current beginner-oriented chess solutions rely solely on academic teaching in separate scenarios to improve a beginner player's performance. This chess project contributes to the field by providing a means of effect experiential learning – a learning method in which the practical space is picture as a learning environment that can foster personal development (Kolb, 2014). The Chess program is to provide a method of learning that can be defined with the following quotation, "Learning in which the learner is directly in touch with the realities being studied. It is contrasted with the learner who only reads about, hears about, talks about, or writes about these realities but never comes into contact with them as part of the learning process" (Tate & Keeton, 1978).

1.5 Report Structure

Title Page: Project title, author, formal title and supervisor.

Chapter 1: Introduction
Aims and objectives of project, and a brief overview of the report.

Chapter 2: Literature Review
Relations between the project and previous work, Analysis of current chess games, Technical details necessary for comprehension of work. Explanation of the reasoning for the project based on the failings of previous works.

Chapter 3: Design
Identification of requirements for the project's product. A display of design diagrams with an explanation for the product.

Chapter 4: Implementation
A Description of the work undertaken and the result of it. A showing of code from the product itself to aid in an explanation of the work done.

Chapter 5: Evaluation
Comparison of the state of the product against the original objectives and aims. An examination the completed work and its effectiveness. Results of the user evaluation and a comparison of the end result compared to the literature survey and design.

Chapter 6: Conclusion
A Brief restatement of the work taken. A summarisation of any findings or recommendations, the limitations of the project and an explanation what can be done with future work to improve upon it.

2. Literature Survey

Chapter Summary: This chapter investigates related works to the chess game project, games that are not necessarily chess related but have good intuitive User Interface and provide a good User Experience. What is lacking in the current chess game scene and how improvement could be made on the existing chess solutions.

2.1 Background & User Interface

Before the chess game was created, research was carried out into what existing chess engines and chess interfaces were present already. As an effective User Interface was to be designed, the literature surrounding interface design was investigated. Some general heuristics for designing an effective User Interface are defined follows (usability.gov, 2017):

- Keep the Interface Simple – The best interfaces aren't thought about by the user, they avoid unnecessary elements.
- Create consistency – The user feels more comfortable navigating a consistent layout, once a skill is learnt in the program it should be able to be applied throughout.
- Be purposeful in layout – the special relationships between items are important, careful placement of items can help draw attention to certain aspects.
- Strategically use colour – Attention can be drawn or redirected using colour and light contrast.
- Typography – Certain typefaces conjure certain results. Legibility and scan-ability are important when choosing fonts.
- Make sure the system is responsive – The user needs to constantly be updated on what is going on in the program, otherwise they can easily become lost
- Think about defaults – By carefully anticipating the user's goals, the process of using the system can be sped up greatly.

If the above heuristics are followed an effective User Interface will be created.

2.2 Chess Engine Techniques

2.2.1 Chess Engines: General Methodology

A technique which must be used in the chess game project is a chess engine. Fundamentally all modern chess engines follow the same two steps, position evaluation followed by an optimal move search. (Zaifrun, 2010) This method can be explained in the pseudo code below.

```
EvaluatePosition() {
     Identify material on board
     evaluate important board squares
     ...
      // the chess engine takes into account many factors,
     each factor being weighted by an amount specified to give
     the best move.
}
PositionTreeTraversal() {
     Evaluate the most immediate nodes
     follow nodes which return the highest advantage
     continue down the tree a specified amount // greater
     depth will make the chess engine choose a better move
}
```

The chess engine evaluates a position similarly to how humans do when they play chess. However, the chess engine performs much better than their human counterparts as they can evaluate hundreds of positions per second.

2.2.2 Stockfish: Powerful Open Source Engine

An example of a chess engine written in this manner is Stockfish. Stockfish is an open Source chess engine written in C++ (Costalba, et al., 2017). Stockfish regularly is ranked in the top 2 chess engine rating lists, with currently only one chess engine, Komodo, having a higher rating. The initial release of this engine was November 2nd, 2008. It has been constantly updated an improved over time with the previous major release being November 1st, 2016. The most recent version of this engine can be viewed at the stockfish GitHub as the code is open to public viewing. Stockfish is a very powerful chess engine; feasibly an improvement on this engine cannot be made in the chess game project. Nevertheless in the chess game project a ‘stockfish-like’ chess engine must be produced. Java has been chosen over C++ for the creation of this engine for the following reasons: Java’s platform independency, the project will be able to run on all platforms that support java without the need for recompilation. Java’s safer and more stable environment, Java performs garbage collection and doesn’t allow pointer arithmetic, which completely removes two common sources of error in C++. Finally, and most importantly, Java has various powerful libraries supporting it

which allow for easy creation and maintenance of a user friendly interface – the contribution to be made to this field in the chess game project.

Stockfish also importantly provides the option to have a selected difficult out of 20. This is an aspect of stockfish that provides a lot of utility, as, at its best, Stockfish is above all human grandmasters. The difficulty levels are required so that a human could feasibly defeat the engine. Stockfish scales its difficulty by changing the fixed depth, the further it moves down the tree the more effective it is at choosing advantageous moves. It is very important that a chess engine provides a sufficient but not overbearing level of difficult for its players. This aspect of stockfish should be implemented in the chess program. Overall the Stockfish technique of having weighted factors contribute to the evaluation of a position, followed by a traversal of a tree can be applied to the solution.

2.2.3 KC Chess: Simpler Engine

An older and less effective implementation of a chess engine is shown in the pseudo code below (Phillips & Bruce, 1990).

```
CutoffSearch (Player, Depth, CutoffValue);
    If Depth = 0 Then
        return (0);
    Else
        BestScore := -infinity;
        Generate the move list for Player as per current board
setup;
        For each legal move in the move list do
            Make the current move and get MoveScore;
            SubTreeCutoffValue := MoveScore - BestScore;
            Score := MoveScore - CutoffSearch (enemy of
Player, Depth - 1,

SubTreeCutoffValue);
            UnMake the current move;
            If Score > BestScore then BestScore := Score;
            If BestScore >= CutoffValue then exit the For
loop;
        End For;
        Return (BestScore);
    End If;
End CutoffSearch;
```

A problem with this program is the extremely simple evaluation of a given move. The score is determined exclusively by the number of value points assigned to the piece that is taken. If no piece is taken then no points are awarded. As implemented, the positional evaluation plays only a small role in selecting a move. A better positional evaluator, which takes into account such things as attack strategy and special situations requiring special actions, should be

combined with a better capture value to calculate the value at all nodes in the search tree. A better capture value would take into account the fact that the material value of the piece varies with the stage and balance of the game. The combination of material and position values should be consistent with the strategy and circumstances of the game. Another deficiency with this program is that it suffers from the horizon effect (Phillips & Bruce, 1990).

Whilst this chess engine implementation works, it is not ideal as explained above. Preferably for the chess engine project, a chess engine more akin to the likes of Stockfish will be implemented. However, due to the focus of the project being more for user experience, the chess engine algorithm specifically does not require as intense a focus therefore a simpler chess engine such as KC chess could be implemented.

2.3 Related Works & Existing Solutions

2.3.1 Firaxis: Civilization, Feedback & Engagement

Firaxis are a strategy video game development company lead by Sid Meier. They place great emphasis on creating user friendly interfaces and have found great success with their games, especially with the Civilization series. In 2010, Sid Meier gave a talk the Games Developers Conference in which he explained how to make an impression on the player within the beginning of the game and maintain user engagement over time. “It’s important to be very careful with the setback that the player experiences, it’s important that the player understands why those things happened and especially how to prevent that from happening the next time. And anytime you can plant that seed of the “the next time” in the players head, you’re well on your way to this idea of replay-ability” (Meier, 2010). This philosophy can be found throughout Firaxis’ games. As shown in figure 1 where a player’s overall empire happiness has reached -10. The player can very clearly understand why their empire has reached a point of such unhappiness. The colour coding shows the positive factors raising the happiness in green and the negative in red. A player can easily understand that the main source of unhappiness is generated by their population. The list even shows the population icon so that the player may easily find it elsewhere. The effects of the unhappy state of the empire are also explained at the top of the overview. Fundamentally, the game is giving the player feedback on their current performance and showing them in a clear manner the steps needed to correct it. In this example issuing a social policy could resolve the player’s issue, as currently there is no social policy in place generating happiness.

Figure 1 Civilization V’s effective player feedback method (Meier, 2010)

This technique of giving important feedback to player when they have mistakes to aid replay-ability should be implemented as retention is an important part of improving one’s proficiency within the chess game. It will directly give them pointers to improve and as Meier states, implant the idea of replay ability – a major part of improving in chess. “As a mere pastime chess is easily learnt, and a very moderate amount of study enables a man to become

a fair player, but the higher ranges of chess-skill are only attained by persistent labour." (Chisholm, 1911). As explained by this, chess mastery can only be really achieved through constant playing. Therefore player retention is an important part making the chess game project successful. Another concept Meier discusses is progression through difficulty levelling. Essentially, Meier states that more difficulty levels are always better for keeping a player engaged as they can continue playing the game, defeating each higher difficulty as if it is a premade challenge. As stated before, the chess engine to be created in this project should include varying difficulty levels by following the stockfish design. With this in mind we can view the following 2 related works to see how they attempt to help the players improve their chess game, and how they hold the player's attention.

It is also worth noting that Civilization V also implements an 'advisor' system, the advisors give the player basic objectives to implement which in the long run will help them to win the game. This is an effective system as the entire grand strategy does not have to be academically explained to the user, they can just follow the short term tactics and strategies which will in turn make up the grand strategy that will help them win the game in the end.

2.3.2 Chess.com: Limited Feedback & UX Shortcomings

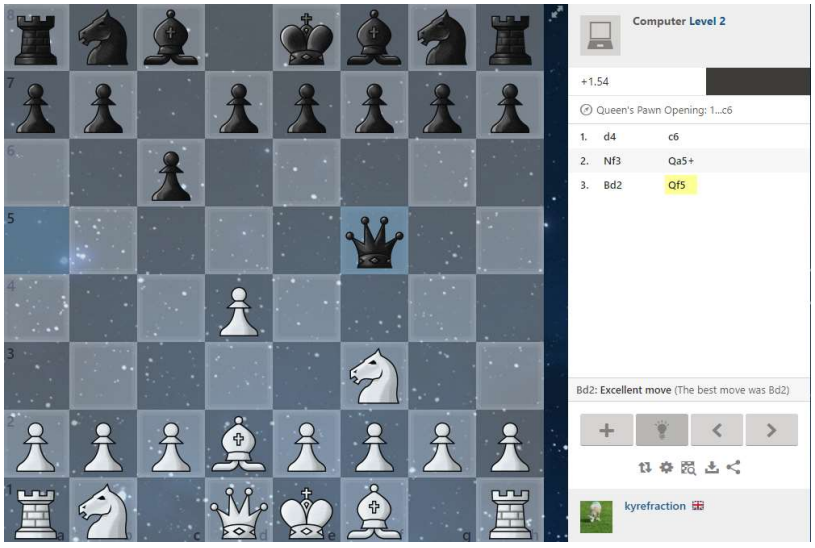


Figure 2 Chess.com's main game interface (Allebest & Severson, 2007)

Figure 2 shows the chess.com game interface. The Board and pieces have excellent graphical representations, with the user having the ability to change how they look to fit their preference. The interface keeps a log of all the moves performed on the right. The AI opponent has a scalable difficulty, from 1 to 10. In figure 2, a chess.com option named "coach mode" has been enabled. This adds to the interface a turn by turn review of each move the user performs. In figure 2 this reads, "Bd2: Excellent move (The best move was Bd2)". Whilst this information is somewhat helpful, it doesn't give any reasoning as to why the move

performed was an excellent move. There are 4 possible move evaluation results; the classifications of moves are as follows:

"**Excellent move"**- Best move available.
"Inaccuracy" - The computer evaluates that this move resulted in a position that is at least 0.3 points worse than the position resulting from the best move available.
"Mistake" - This move is at least 0.9 points worse than the best move available.
"Blunder" - This move is at least 2 points worse than the best move available.

Figure 3 a Chess.com daily puzzle (Allebest & Severson, 2007)

It is helpful to tell the user when they have made a mistake, however unlike figure 1 which shows the user a comprehensive and understandable list of factors which are affecting their performance, chess.com gives the user no information about those factors. Furthermore, the names of the classifications and what their outcomes mean for the game aren't completely clear. For example, "if a blunder is performed, should the player resign to save themselves from attempting to win an unwinnable game?" Unfortunately, with this interface the player does not receive any of this information which would be crucial in helping them to improve.
Chess.com does include some non-conventional methods of aiding people in improving their chess game. Figure 3 shows a daily exercise in which the player is placed in a premade scenario and must discern their most effective move. In this example the player cannot possibly win the game and must try for a stalemate. Unfortunately this information left for the user to figure out, this makes the daily challenges less accessible for newer chess players as there is no description of what they have been tasked to do. The win condition is unclear.

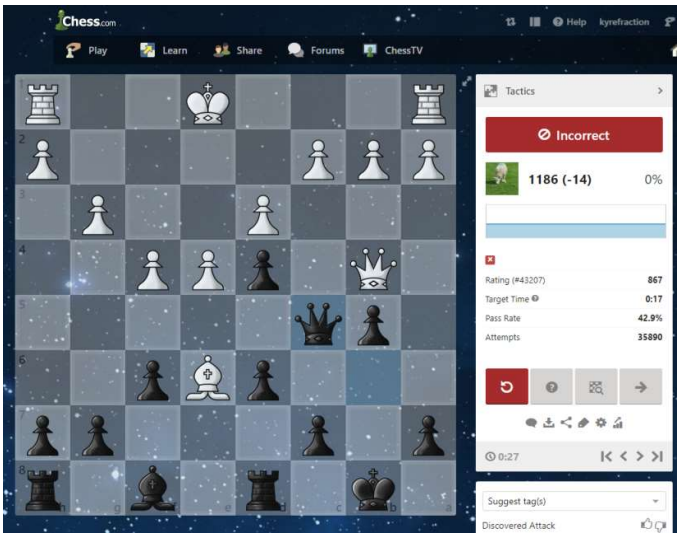


Figure 4 Chess.com 'learn' mode (Allebest & Severson, 2007)

Chess.com also has a tab entitled, "learn", which once pressed brings up a scenario like the one shown in figure 4. Similarly to the daily exercise, the user must attempt to find the one best possible move as chosen by the site's chess engine. The problem with this method is that, during the chess midgame, there are many moves the user could perform which would not objectively be better than another. Some players prefer to trade pieces, some prefer to protect their pieces and prey on enemy blunders. This subjectivity is removed in these scenarios. If the user performs a move which does not match the chess engine's move, it is labelled, "incorrect" and points are deducted from the user's score. If we reconsider Meier's quote, "It's important to be very careful with the setback that the player experiences, it's important that the player understands why those things happened and especially how to prevent that from happening next the time". It is very clear that chess.com does not help the user to understand why their actions are not optimal, which massively discourages the user from continuing to learn.

To improve upon chess.com's implementation, the 'Coach Mode' option should be expanded. An understandable list of positive and negative factors which affect an evaluation of a position should be made available to the user. This would aid the user in improving as they get to an intermediate level. More importantly, if a user is completely new to chess, chess.com would be too overwhelming; it does not accommodate the beginner player. An explanation of the pieces, standard chess vocabulary and book moves would help the player greatly, however, they are not present in this execution. (Allebest & Severson, 2007)

2.3.3 Chesscademy: Improved Feedback & New Approach

Chesscademy is an implementation of a chess game which is tailored to beginner users. It features a syllabus which provides a beginner chess player with all the information necessary to begin playing the game. Chesscademy's method of introducing new players to the game is very effective. They provide the user with lessons explaining each piece, how that piece moves and how it takes in an enclosed environment. It then later explains more advanced chess concepts such as avoiding hanging pieces. This is an excellent way in getting a new player aware of the more complicated concepts in chess (Ng, et al., 2015).

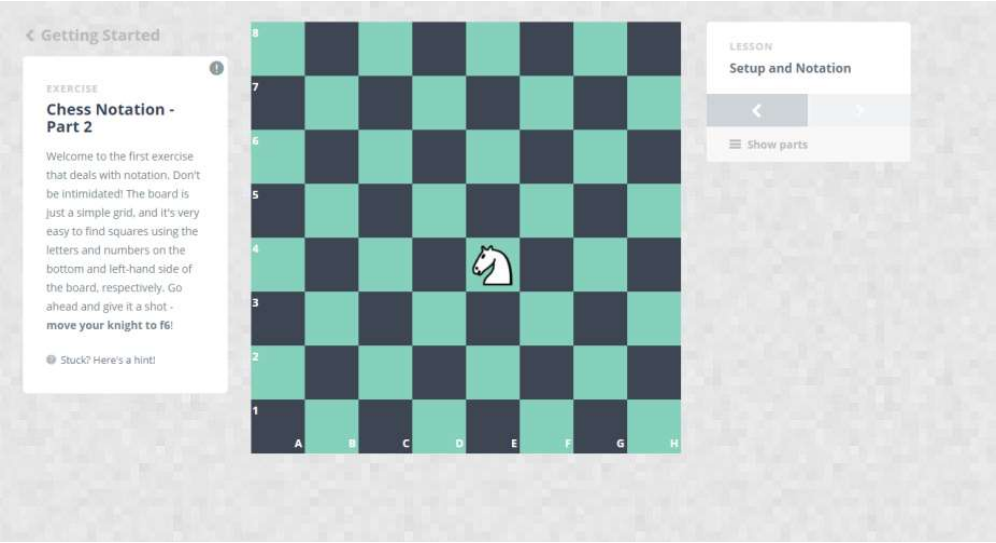


Figure 5 Chesscademy lessons (Ng, et al., 2015)

Figure 5 shows one of Chesscademy's enclosed lessons. In this example it is explaining how to move a knight. It has tasked the player with moving the knight to f6, which also helps the user pick up on chess movement notation.

Figure 6 shows Chesscademy's implementation of the daily challenges. This version is similar to chess.com, but improves upon it by providing an explanation which becomes available once the challenge is attempted. In this example, "Black's rook is pinned, allowing mate in two from the pawn". This gives the user the necessary amount of information needed to understand the problem and they can apply this knowledge further in a real chess game scenario.

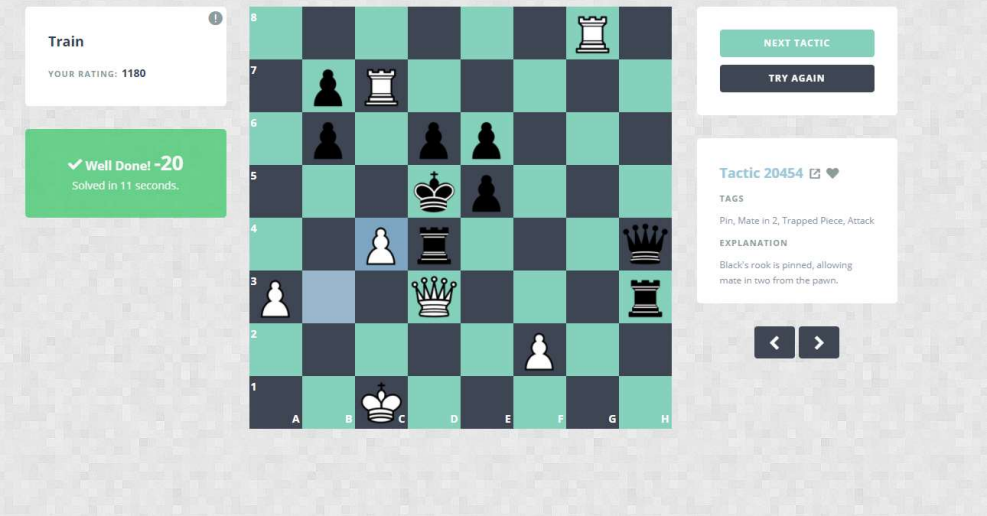


Figure 6 Chesscademy 'learn' mode (Ng, et al., 2015)

Figure 7 shows Chesscademy's play versus AI mode. It is very similar to chess.com's implementation. However, Chesscademy's 'play versus AI' mode completely lacks a 'Coach Mode'. It also has no audio sound cues for pieces moving or a player being placed in check like chess.com has. Similarly to chess.com, this is a part where Chesscademy fails to succeed. For the Chess Game project, it is apparent that a more advanced 'Coach Mode' should be implemented.

In conclusion, considering this project's desiderata, Chesscademy is a more proficient example of a related work. The beginning videos, explanations of pieces and challenges are perfect for introducing a beginner to chess and helping an intermediate player to improve. The main part where both chess.com and Chesscademy fall short is in the actual chess versus AI modes. It can become very difficult for a player to evaluate a position in this mode; this is where the more advanced Coach Mode would be come into effect. The Chess engine AI opponent should present a level of difficult chosen by the player. This is important as if the game isn't sufficiently difficult, "then on the strategy layer a lot of things don't kick in. The player just doesn't have to engage with the systems." (Solomon, 2016) A chess engine similar to the stockfish engine algorithm would be ideal.

Figure 7 Chesscademy's main game interface (Ng, et al., 2015)

2.3.4 Comparison of Existing & Related Systems

Chesscademy in comparison to chess.com is much more suited to the beginner chess player as it is aimed for that market and provides the user with academic lessons for the user to carry out before beginning their actual chess journey. However, for real-time in game aid, chess.com's solution is more effective. Whilst the chess.com feedback is very limited, Chesscademy provide no in game feedback whatsoever. It is apparent from both of these studies that in-game real time chess game feedback is an area that is severely lacking in the current scene. An in game feedback akin to Civilization V's, one that gives clear colour coded and simple advice could be implemented as an improvement over the existing solutions.

2.4 Social & Ethical issues

A social implication of this work could be that, if too successful, it could be too addictive for a player, potentially hampering performance in other aspects of their lives. Chess is often compared to pitting two military commanders against each other. Instead of row and column, rank and file are used to describe the board, which is military vocabulary. The suggestion of a militarised conflict could potentially be an ethical issue with the chess game project.

3. Design Statement

Chapter Summary: This chapter displays what the end product should resemble, the decisions made during the conception of the product and a brief look into how some of the more technical aspects of the program will be implemented.

3.1 Approaches to In-Game Teaching

3.1.1 Novice to Grandmaster

The chess game must aid the user in becoming more proficient at chess, to understand how to go about converting a complete novice chess player to an expert player; an itinerary can be produced. The guideline set out by Emanuel Lasker is useful as it can be used to discern the weighting of importance on each aspect of chess (Lasker, 2009).

Rules of Play and Exercises: 5 hours
Elementary Endings: 5 hours
Some Openings: 10 hours
Combination: 20 hours
Position Play: 40 hours
Play and Analysis: 120 hours

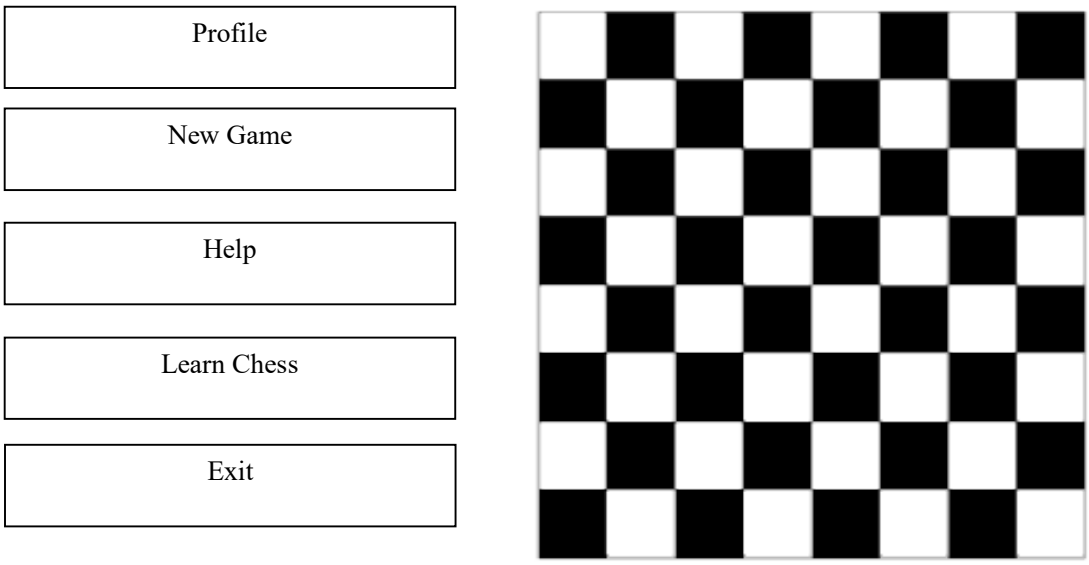
As shown, the mainstay of chess learning and improvement is done during play and analysis, however, learning the rules, endings and openings are important preliminary steps which a novice must undertake. The chess game project will follow this philosophy and provide the users with all means necessary to improve in each of these aspects. The menu screen design reflects this. The “learn chess” option is provided first, this option will fulfil the “Rules of Play and Exercises” in the users learning agenda. The Following learning sections will be fulfilled in the “play versus AI” option, as the user will learn as they play.

3.1.2 Maximisation of User Retention

To further invest the player in completing the 120 hours necessary of play and analysis to have mastery over chess, an importance on user retention must be placed (Chisholm, 1911). An ‘XP’ or experience system can be implemented to give the user a sense of progression throughout the game, increasing the chances of the user returning to the game (App Annie, 2015). Each completed lesson, each win and loss versus the AI will garner the user in-game experience. This experience could be used in-game to unlock aesthetic differences; however, even if the experience fulfils no purpose, the sense of progression alone can inspire more commitment to playing the game (Nunes & Dréze, 2006).

3.2 Program Design

3.2.1 The Menu Screen



The “Learn Chess” and “Help” options will take the user to other forms.

The “Learn Chess” option will start the novice user’s chess experience by giving them information about each piece, its legal moves, its strength and weaknesses etc.

The “New Game” option will take the user into a chess game against the computer; the user will select the difficultly of the AI opponent once they have entered this screen. This option is greatly advised for the beginner chess player to choose after they have explored the “Learn Chess” option.

Once inside the game, the user can press the Help button to gain advice from the program’s advisor.

3.2.2 Advisor Overview

The Advisor will provide the user with the chess engine heuristics during the game. The heuristics will be short-term goals during each sequence of the game, which, if achieved, will improve the user's performance during that section. With this in mind, we can split the game into three sections: Early Game, Midgame and Endgame. The information should be presented to the user in the most clear and understandable way possible. Positive heuristics will be coloured green and negative coloured red to aid in this aspect (Meier, 2010).

3.2.3 Early game/Chess opening

The first few moves in a chess game are called the 'opening'. An effective opening aims to give the King piece adequate protection, take control of the centreboard, place minor pieces in important positions and give mobility to minor pieces (Keene & Levy, 1993).

This important information will be relayed to the user during this phase. It could be useful for the user if the squares that the user should take control of were highlighted as shown in figure 8.

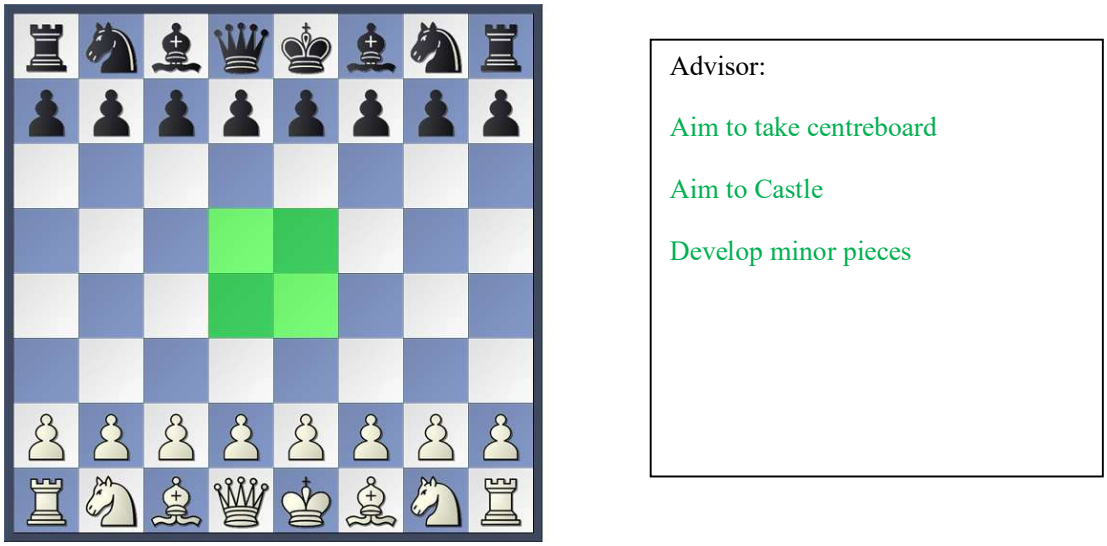


Figure 8 Advisor in use during the early game (Chess Central, 2018)

There is a finite amount of feasible chess openings, which theoretical variations could potentially be memorised. These openings are named, certain chess openings are considered more effective for beginner chess players (CHESScom, 2017). These openings are:

Italian game: 1.e4 e5 2.Nf3 Nc6 3.Bc4. Aims to control the centreboard quickly with the pawn and knight, followed by placing the bishop in its most effective position and being ready to castle.

Sicilian Defence: 1. e4 c5. Performed by the Black player, this is an aggressive opening for Black, which aims to exchange a central pawn for a bishop's pawn.

In a similar fashion to highlighting the important squares during the early game, the Advisor could dictate these moves to beginner players to help them to give them a start in their chess game, as, during the opening the user can easily be overwhelmed by the choices given to them.

The chess opening stage will end once both players have either castled their kings or developed their minor pieces, the knights and bishops (Gaspard, 2009). Once this stage is complete, the midgame will begin.

3.2.4 Midgame

The chess midgame is the most complex phase of the chess game as there are still many pieces on the board. It is the phase of the game in which chess engines are most likely to overtake human opponents, as they are much better at analyzing trades and positional play than humans. The Advisor can provide the following advice to the user; however, it is the phase of the game in which the assistant can provide the least amount of aid to the user. It would be advantageous for user learning to give them the option to undo a move if it proves to be disastrous, as it keeps them in the game and teaches them what not to do the next time.

Advisor:

Pin important enemy pieces

Set up outposts

Attack enemy pieces as white

Defend as black

Take control of important diagonals

3.2.5 Endgame

The chess endgame is somewhat similar to chess openings in the sense that there are much fewer feasible moves, having much less complexity and clearer goals (Fine, 2003). The aims of the endgame are very different from the midgame. The Advisor can give the following advice to the user during the game: Aim to promote a pawn by reaching the eighth rank. Use the King piece aggressively as it can be a useful attacking piece. If the user has more material, aim to exchange minor and major pieces, otherwise avoid exchanging minor and major pieces and aim to exchange pawns.

The information will be presented in as clear a way as possible. As shown in figure 9.

Figure 9 Advisor in use during the endgame

3.2.6 Chess Basics: Notation & Material Definition

The Learn chess option will provide the novice user with the necessary information to begin playing chess. The basics of the chess piece, the notation, the value and how the piece moves will be taught to the user.

Inside the Learn Chess option, the user will be able to iterate over all of the chess pieces and view the information on them accompanied by an image to show what the piece looks like.

Pawn

Value: 1

Notation: N/A

Movement:

Pawns cannot move backwards
Pawns advance a single square forward, except for the first move, where it has the option of advancing two squares forward
Any piece in front of a pawn blocks the pawns advance
The pawn captures diagonally of its position

Strengths:

Strong defensive lines
Can be promoted to any minor or major piece
Control of important squares

Weaknesses:

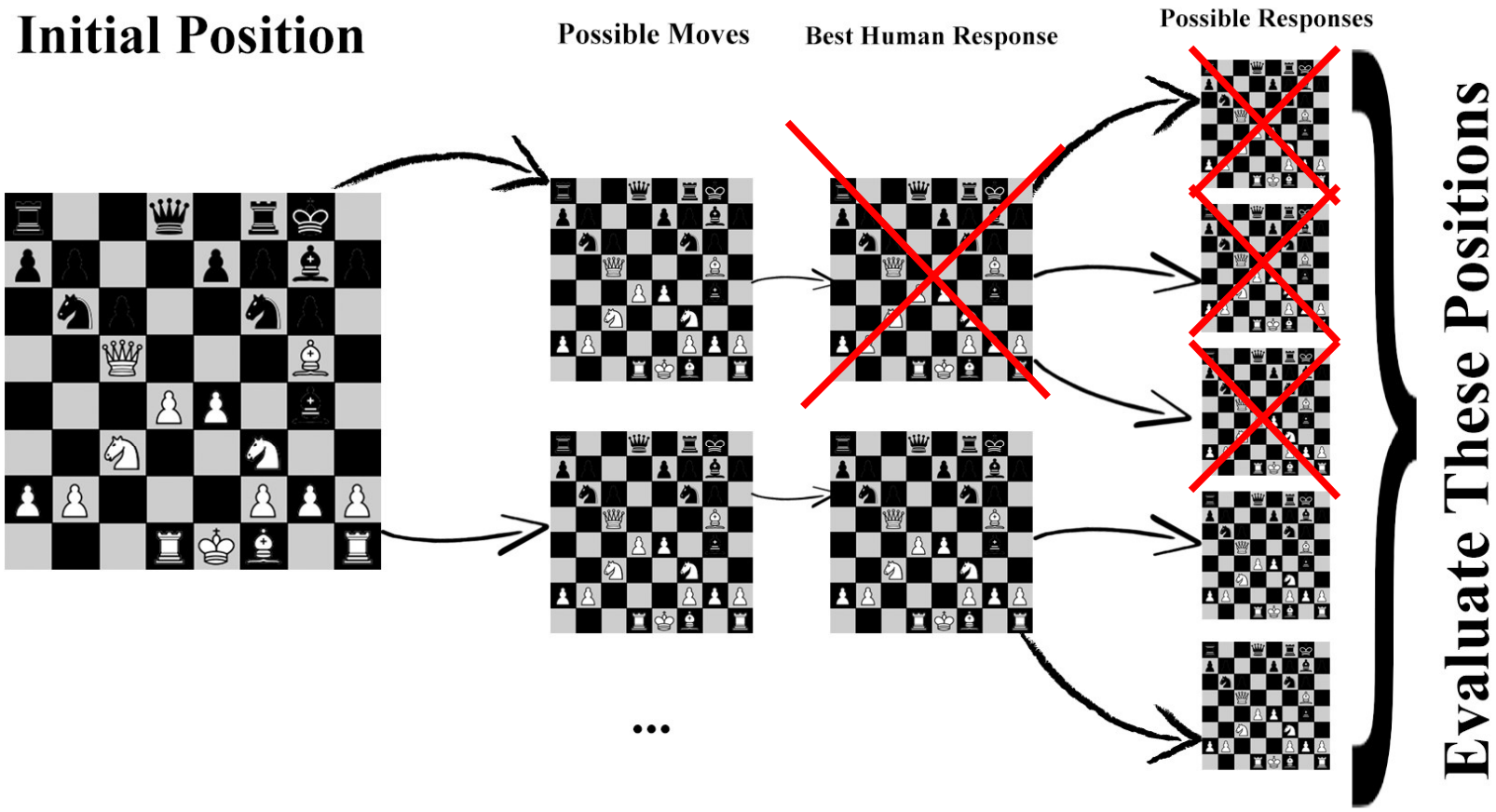
Stacked pawns are ineffective
Pawn movement is irreversible

Above is an example of a piece's definition (Max, 1997) (Waffllemaster, 2012). The fact file will include the value of the piece, its notation, the legal movement, its strengths and weaknesses.

3.2.7 Chess Engine Algorithm Design

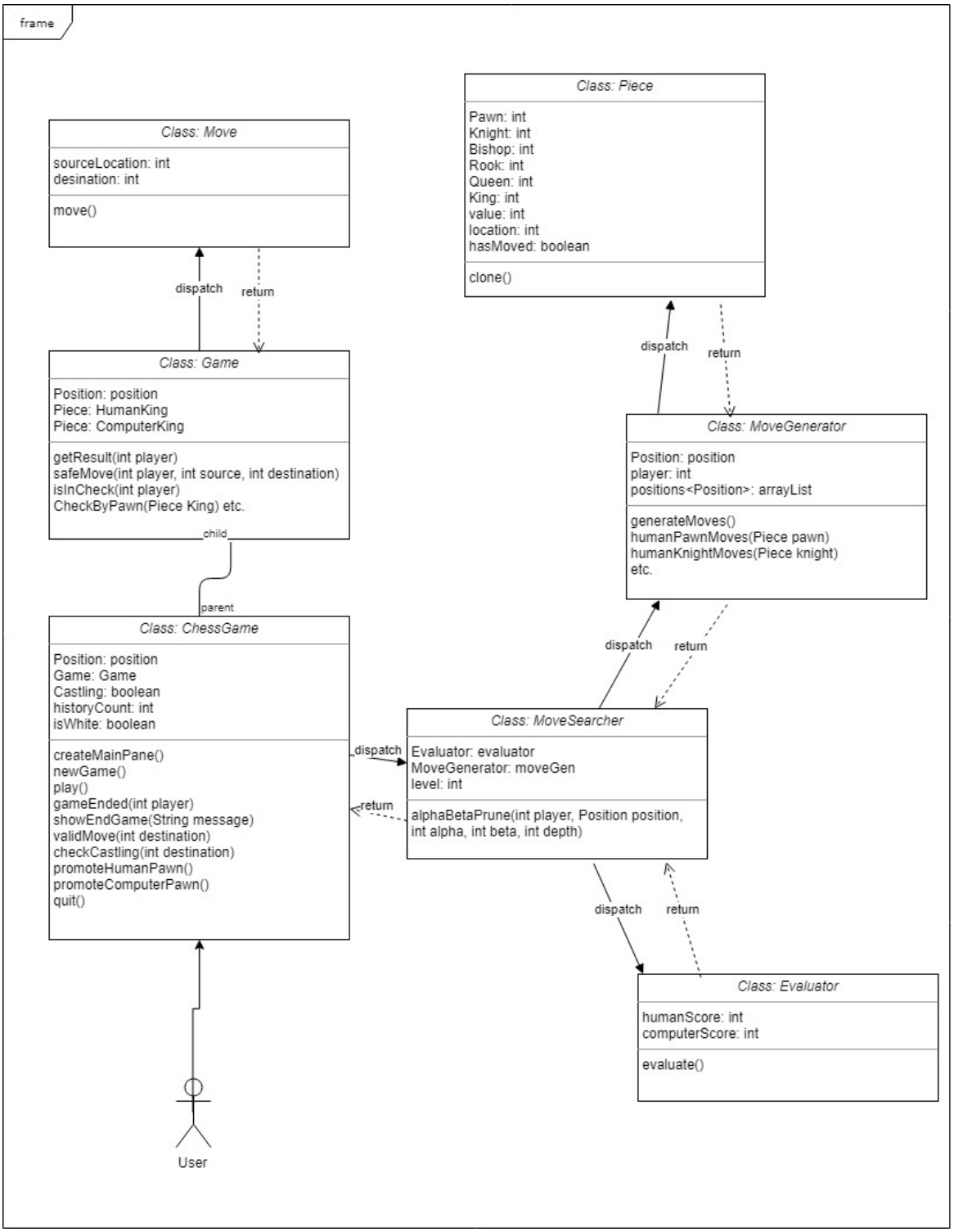
The Chess Engine is to be modelled on the already existing stockfish engine. A full technical breakdown of the stockfish engine can be found in Appendix D. The engine should use alpha-beta pruning of a search tree akin to stockfish's implementation. The nodes of the search tree will be chess positions. To create the most optimal search tree, the tree should aim to be balanced, as memory usage is proportional to the depth of the tree rather than the number of nodes. Whilst the tree can be searched via a recursive and iterative algorithm, and an iterative being cheaper performance-wise, a recursive alpha-beta pruning algorithm will be implemented. This was chosen as it requires much less code to operate and it is less prone to errors. There is also no guarantee that the iterative equivalent of the recursive function wouldn't just be simulating the recursive function anyway. The rewriting of explicit stack operations via an iterative algorithm could prove useful if the recursive method is deemed too costly on resources. The Alpha-Beta search algorithm is the algorithm that should be used as it is a more time and resource effective implementation of the basic minimax algorithm and is the standard for machine opponents in two player games like chess (Russell & Norvig, 2010).

The diagram below displays what the alpha-beta pruning algorithm would be implemented as in the chess game. It is an improvement in search time over the minimax algorithm as the nodes that could not possibly have influence over the final node choice are removed once identified (signalled by a red cross).


Initial Position
Possible Moves
Best Human Response
Possible Responses
...
Evaluate These Positions

3.2.8 UML Design

This section displays the class diagrams of the program to be constructed. This section can be used for reference when creating the program. The UML diagrams ensure appropriate methods belong to the appropriate classes.



3.3 User Experience & Levelling

3.3.1 Allocation of Experience Points & Level Threshold

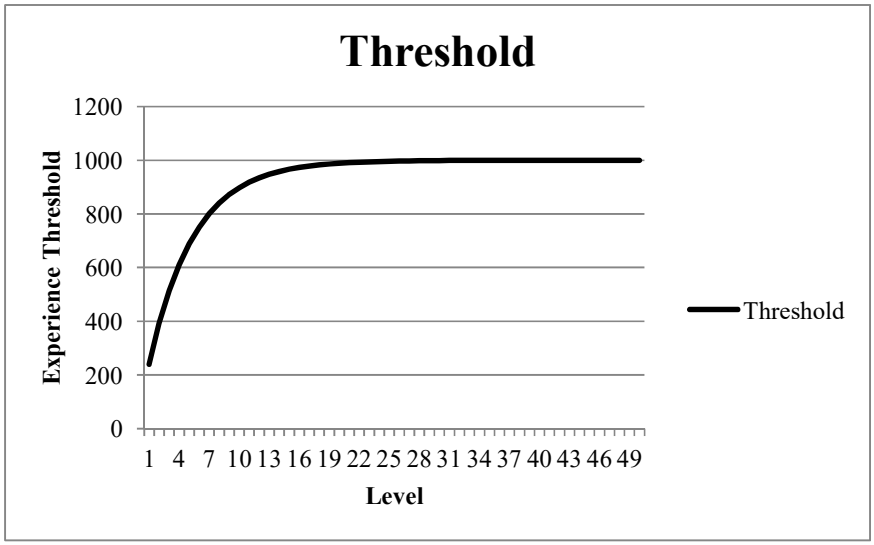
The User will be granted Experience Points when they complete certain tasks. If a user reads the piece information for the first time, they will be rewarded with Experience; this encourages them to continue to read the pre-game information as they get a sense of reward for doing so. Upon defeating the AI opponent the user will also gain some experience, the higher the level of the AI, the more experience the user will gain. This encourages the user to undertake the harder challenge of defeating the higher level AI opponent, as, without this incentive, the user may be inclined to continuously play against the AI opponent difficulty level that is beneath their own. The user will have two attributes, Level and Experience. To Level up the user will have to reach a threshold determined by the following equation.

$$\theta = 1000 - ((0.8^{x})*950)$$

Equation 1 Original equation designed to calculate level threshold

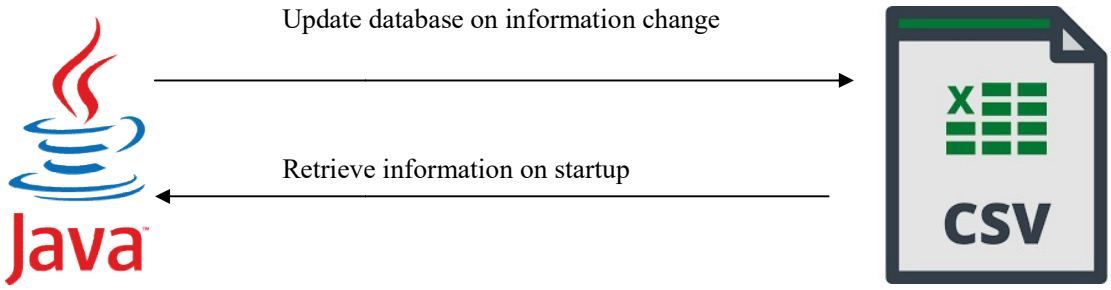
θ = Threshold
x = Level

This equation yields the following graph for the first 50 values of x. This shows clearly a logarithmic curve. The first level corresponds to a Threshold of 240 which quickly tends towards the absolute value of 1000. The equation has been designed this way to give the user a particularly easy time levelling up to begin with, keeping in line with the philosophy of rewarding the player early on to incentivise more play (Meier, 2010). The equation also puts a cap at 1000 for the experience Threshold, this type of curve was chosen over an indefinitely rising linear equation as it would be too time consuming for the user after reaching a certain level to continue attempting to level up if the threshold increased indefinitely (Voigt, 2017).



3.3.2 Permanence of User Information

For the user information to be stored permanently, a CSV data will be used. Upon starting the program, the user level and experience will be loaded from the file so they therefore can continue with the progress they have already made in the game. The CSV file will be updated every time the user exit's the program as to not lose any progress.



It is important that the user's profile information is stored, as without it, all sense of progression will be lost upon exiting the program. Discouraging the user from playing the Chess Game again as it would require a completely fresh restart.

4. Implementation

Chapter Summary: This Chapter explains the process that was taken in the creation of the product, including the setup of the IDE to begin the project and an explanation of each of the classes. The appropriate technology used in the product is described in the corresponding class. The classes are separated into User Interface and Core Chess Game classes; this separation is so that it is easy to find the more technical and algorithm oriented code in the core chess game classes and the more objective visual code pieces in the user interface classes.

4.1 Project Setup

To begin the project, a new Java Project was set up in Eclipse; the project contains the following classes, sorted by their section of the product.

4.1.1 Core Chess Game Classes - ChessGame.Java

This class is the main controller class of the program, it contains the Main method and creates the initial interface which has the main menu and the game itself. Upon startup, this class retrieves user information (called user.csv) from a Comma Separated Values file and holds them in corresponding global variables. These variables are declared as static as they should be able to be accessed without creating a new instance of the main ChessGame.Java class. Upon program exit, the user.csv file is update to reflect the experience gathered during this instance of the program. This lets the user continually advance in experience levels in the game - encouraging them to return.

The Menu is populated with the following items:

Profile
New Game
History
Learn
Help
Quit

The buttons: Profile, Learn and Help create instances of other classes which will be discussed later. New Game creates a new instance of the chess game on the main board. The Pieces are returned to their starting positions, the turn timer is set back to 0 and the user goes through the new game prompt, in which the user sets the difficulty of the opposing AI player and whether they are playing as white or black. The History button extends the main pane to include the east pane, which contains another chess board displaying the previous moves of the current game. This lets the user scroll through each turn so they can analyse their play afterwards, a valuable action if the user would like to improve. Quit calls the quit method which saves the user's progress and gives the user a prompt to ensure that exiting the program is what they intended.

To aid with the User Experience, when a button is hovered over the colours are inverted. This provides the user with instant feedback of their mouse movement which is an integral part of a good User Interface.
The ChessGame.Java class contains the chess game itself. For programmatic purposes, the chess board was visualised as an array of an absolute position values. The position of each square have been visualised with a chess board to show how each position maps to the chessboard itself.

```
//21 22 23 24 25 26 27 28
//31 32 33 34 35 36 37 38
//41 42 43 44 45 46 47 48
//51 52 53 54 55 56 57 58
//61 62 63 64 65 66 67 68
//71 72 73 74 75 76 77 78
//81 82 83 84 85 86 87 88
//91 92 93 94 95 96 97 98
```

With each square having its own unique ID number, it is possible to check which piece lies on which square. As each piece moves in a predictable manner, possible moves can be calculated with simple math equations. To further aid with the User Interface, when a user selects a piece, it's possible moves are visualised by a glow on the squares that the user can move that piece to.

Pawn

The following product code is ran when the user selects a piece, the first if statement checks what piece the user has selected. Each piece has a corresponding numerical value, these numerical values reflect the value of the piece to the computer (note, the numbers are on a scale ten times larger than the piece value systems commonly used by humans. This is to allow for greater specificity in the piece values without having to have the pieces declared as doubles). To check the square directly above the selected piece, a value of 10 must be subtracted from the starting square's value. it is possible to see from the square value matrix that a square's location - 9 corresponds to the square which is one square up and one to the right and a square's location - 11 corresponds to the square which is one square up and one to the left. These are the basic pawn moves, -10 being a standard move upwards and -9 & and -11 being taking moves. The -20 (two moves forward) however, must also be checked as at the first time a pawn moves, it has the option to move two squares forwards. The validMoveCheck function returns whether the proposed position is a valid one.

```
if(piece == 100) { // if the piece is a pawn
  if(validMoveCheck(lastClickLocation-10)) {
     g.drawImage(images.get(GameData.GLOW), xPos, yPos -
squareSize,this); // show where the pawn can move
  }
  if(validMoveCheck(lastClickLocation-20)) { // up two
squares if not moved
     g.drawImage(images.get(GameData.GLOW), xPos, yPos -
squareSize - squareSize,this); // show where the pawn can
move
  }
  if(validMoveCheck(lastClickLocation-9)) { // up and to the
right
      g.drawImage(images.get(GameData.GLOW), xPos +
squareSize, yPos - squareSize,this); // show where the pawn
can move
  }
  if(validMoveCheck(lastClickLocation-11)) { // up and to
the left
    g.drawImage(images.get(GameData.GLOW), xPos - squareSize,
yPos - squareSize,this); // show where the pawn can move
  }
```

Knight

The Knights movements is visualised by the code comment below. The knight can move to 8 possible places maximum. All of these positions will be checked, as if the position is simply off the board and invalid the validMoveCheck function will return false. The Knight develops in 'L' shape movements; these correspond to absolute square value changes of -21, -19, -12, -8, 8, 12, 19 and 21.

```
//   x1   x2
// x3       x4
//      Y
// x5       x6
//   x7   x8
// 8 potential places to move to
```

Bishop

For the following positions the description of absolute square value changes takes a different approach. As the bishop's movement options aren't a fixed amount (the bishop can move the most amount of squares when situated on a board corner where it can move a maximum of 7 squares), a for loop was set up which checks the maximum possibility of square value changes in all directions and if those positions correspond to positions off the board, the validMoveCheck returns false and the proposed position is not shown.

```
if (piece == 325) { // if the piece is a bishop, -11 x1, -9
x2, +9 x3, + 11 x4
  for(int j = 1; j <= 8; j = j + 1) { // x1
      int source = move.source_location;
      int topLeft = source-(j*11);
        if(topLeft < 21) {}
    else {
    int destinationSquareTopLeft = position.board[topLeft];
    if(destinationSquareTopLeft == GameData.EMPTY &&
validMoveCheck(topLeft) == true) {
        g.drawImage(images.get(GameData.GLOW), xPos - (j *
squareSize), yPos - (j * squareSize), this);
    }
  }
}
```

This code checks the Northwesterly movement of the Bishop. A check is done to ensure that the bishop is not trying to move to a position outside the array of position so avoid an 'out of bounds array exception'.

```
// x       x
//  x     x
//    x1  x2
//      y
//    x3  x4
//  x     x
// x       x
 // varying places to move to
// 7 max in any diagonal
```

Castle

The castle's movement is very similar to the bishop, except whilst the bishop moves diagonally in the northwest, northeast, southwest and southeast directions, the castle moves either vertically or horizontally. Also similarly to the bishop, the castle's potential novelist is dynamic - the amount the castle can move in one direction depends on where on the board the castle is. For the castle a for loop is also set up to check each direction. An if check is also done to ensure the castle isn't trying to access positions outside the bounds of the absolute square values array.

```
//        x
//        x
//        x3
// x x x2 y x1 x x
//        x4
//        x
//        x
```

Queen

The Queen is an amalgamation of the Castle and Bishop movements combined as the queen can move in any direction for any amount of squares. The code used for checking the bishop and castle positions can be reused in conjunction with each other, as both are independent and do not alter the other's checks.

```
//  x    x    x
//    x  x  x
//     x x x
// x x x y x x x
//     x x x
//    x  x  x
//  x    x    x
```

King

The King can move in any direction but only by one square. Because the King has a small finite amount of possible moves, a selection of if statements can be used similarly to the pawn and knight pieces.

```
//
//    x4x3x2
//    x5y x1
//    x6x7x8
//
```

History

The History pane shows, the current games previous moves. To store this information an arrayList of type 'Position' (a class in the program) named 'previousPositions' was created. Once a move has been performed during the animating pieces stage, the positions of the move are added into the arrayList through the method 'newHistoryPosition()'. The variable 'historyCount' is used to access the positions in this array. When the next, previous, last and first buttons are pressed the historyCount variable is updated and the corresponding index in the arrayList is accessed.

4.1.2 Core Chess Game Classes - Evaluator.Java

The Evaluator class can be seen as the 'brain' of the AI. The evaluator class takes a position and returns via a numerical value how advantageous it views that position. The numerical value is calculated using real chess heuristics. For example, a piece of advice given to the user by the program is that they should aim to take the centreboard with their pawns. This is enforced in the evaluation through the following code.

```
if(row<4) human_score+=(8-row)
if(col>4){
  human_score -= ((col-4));
} else if(col<3){
  human_score -= ((3-col));
}
```

This lowers the human_score value the further away from the centreboard their pawns are. The columns start at 0 and end at 7, therefore, the two centreboard squares are situated at columns 3 and 4. With the code shown above, the user is penalised to a greater degree the further from the centre their pieces are, this in addition to penalising the user the further they are from the centreboard in terms of rows, it gives an accurate evaluation of the user's capitalisation of centreboard dominance. A similar process is carried out for each of the pieces. The evaluator gives a higher evaluation if the chess heuristics are followed.

4.1.3 Core Chess Game Classes - EvaluatorForFeedback.Java

This class is specifically for the user's game evaluations. The Evaluator is used by the chess Artificial Intelligence to make effective moves, whilst this evaluator uses the exact same heuristics to judge the user's position. These values are used to update the evaluator display.

4.1.4 Core Chess Game Classes - Game.Java

This class contains important game related methods. The mainstay of this class determines whether a player is in check or not. This is done by having a master method named 'isChecked' which calls each piece's check status, if the king is in check by any piece then he is in check. The code below shows the implementation of this.

```
public boolean isChecked(int player){ // function to check if the player is in check
    boolean checked = false; // create the checked boolean which is instantiated as false
    Piece king = (player == GameData.HUMAN)?human_king:computer_king;
    if(king == null) return false; // check the kingpiece isn't a null value, avoids errors in execution
    checked = checkedByPawn(king); // change the checked boolean to true if the king is in check by a pawn
    if(!checked) checked = checkedByKnight(king); //do for each of the pieces if not already place in check
    if(!checked) checked = checkedByBishop(king);
    if(!checked) checked = checkedByRook(king);
    if(!checked) checked = checkedByQueen(king);
    if(!checked) checked = desSquareAttackedByKing(king);
    return checked; // return the final state true or false of whether the king is in check or not
}
private boolean checkedByPawn(Piece king){ // function to determine if a king is placed in check by a pawn
    boolean checked = false;
    int location = king.location; // get the location of the King Piece that has been passed into the function
    if(king == human_king){ // if the king is the user's king
        int right_square = position.board[location-9]; // the right square is the position on the board of the user's king - 9,
        int left_square = position.board[location-11]; // the left square is the position on the board of the user's king - 11
        if(right_square == GameData.ILLEGAL || left_square == GameData.ILLEGAL) return false; // if the above squares are in ill
        if(right_square<0 && position.computerPieces[-right_square].value == Piece.PAWN) // if the right square to the king is o
            checked = true;
        if(left_square<0 && position.computerPieces[-left_square].value == Piece.PAWN) // likewise if the left square to the kin
            checked = true;
    }else{ // for the Computer's King piece
        int right_square = position.board[location+11];  // right square is the position of the king +11
        int left_square = position.board[location+9]; // left square is the position of the king +9
        if(right_square != GameData.ILLEGAL){ // if the right square isn't in an illegal position
            if(right_square>0 && right_square != GameData.EMPTY && position.humanPieces[right_square].value == Piece.PAWN) // if
                checked = true; // the computer's king is in check
        }
        if(left_square != GameData.ILLEGAL){ // likewise if the left square is occupied by a human pawn
            if(left_square>0 && left_square != GameData.EMPTY &&
                    position.humanPieces[left_square].value == Piece.PAWN)
                checked = true;
        }
    }
    return checked; // return the value true or false of whether the king passed in is in check or not by a pawn
```

Figure 10 Code that checks whether a player is in check

4.1.5 Core Chess Game Classes - MoveGenerator.Java

This class is another fundamental class to the Chess Artificial Intelligence. To generate all the possible moves, a for loop is used to iterate over all of a player's pieces and a corresponding method is called to see where that piece can move to.

```
public void generateMoves(){
    if(player == GameData.HUMAN){
        for(int i=1; i<position.humanPieces.length; i++){
            Piece piece = position.humanPieces[i];
            if(piece == null) continue;
            switch(piece.value){
                case Piece.PAWN:
                    humanPawnMoves(piece);
                    break;
                case Piece.KNIGHT:
                    humanKnightMoves(piece);
                    break;
                case Piece.KING:
                    humanKingMoves(piece);
                    break;
                case Piece.BISHOP:
                    humanBishopMoves(piece);
                    break;
                case Piece.ROOK:
                    humanRookMoves(piece);
                    break;
                case Piece.QUEEN:
                    humanQueenMoves(piece);
            }
        }
```

Figure 11 The Move generation master method

As shown by Figure 11, the sub-method appropriate to the current piece in the for loop is called to generate the moves. The moves are stored in an array list populated by instances of the Position class. An instance of the 'Game' class is also present in this class, this is so the important move checking methods in the Game class can be accessed when generating a move – there is no point in storing a move that would put the player in check.

```
public void humanPawnMoves(Piece pawn){
    int location = pawn.location;
    int forward_piece_index = position.board[location-10];
    if(forward_piece_index != GameData.ILLEGAL){
        if(forward_piece_index == GameData.EMPTY && gs.safeMove(GameData.HUMAN,location,location-10)) {
            positions.add(new Position(position,new Move(location,location-10)));
        }
    }

    if(location > 80 && forward_piece_index == GameData.EMPTY &&
            position.board[location-20] == GameData.EMPTY && gs.safeMove(GameData.HUMAN,location,location-20)) {
            positions.add(new Position(position,new Move(location,location-20)));
    }

    int right_piece_index = position.board[location-9];
    if(right_piece_index != GameData.ILLEGAL) {
        if(right_piece_index<0 && gs.safeMove(GameData.HUMAN,location,location-9))
            positions.add(new Position(position,new Move(location,location-9)));
    }
    int left_piece_index = position.board[location-11];
    if(left_piece_index != GameData.ILLEGAL) {
        if(left_piece_index<0 && gs.safeMove(GameData.HUMAN,location,location-11))
            positions.add(new Position(position,new Move(location,location-11)));
    }
}
```

Figure 12 Move Generation sub-method

Figure 12 shows one of the sub methods used for generating the pawn movements. This is similar to showing the pawn's potential moves in the 'ChessGame' class as the absolute square value differences are the same when moving a piece.

4.1.6 Core Chess Game Classes – MoveSearcher.Java

The MoveSearcher class is also a class that is a main part of the chess Artificial Intelligence; it is in this class that the difficulty of the Artificial Intelligence opponent is implemented. To search for the best move, the program uses Alpha-Beta pruning as its search algorithm. This algorithm decreases the amount of nodes that are evaluated in the search tree as it stops evaluating a move when one it is found that there can be no improvement on the currently optimal move. It is important that Alpha-Beta pruning is used over naïve minimax as it limits the search time of the algorithm, making the Artificial Intelligence in game make moves quicker which improves the User Experience.

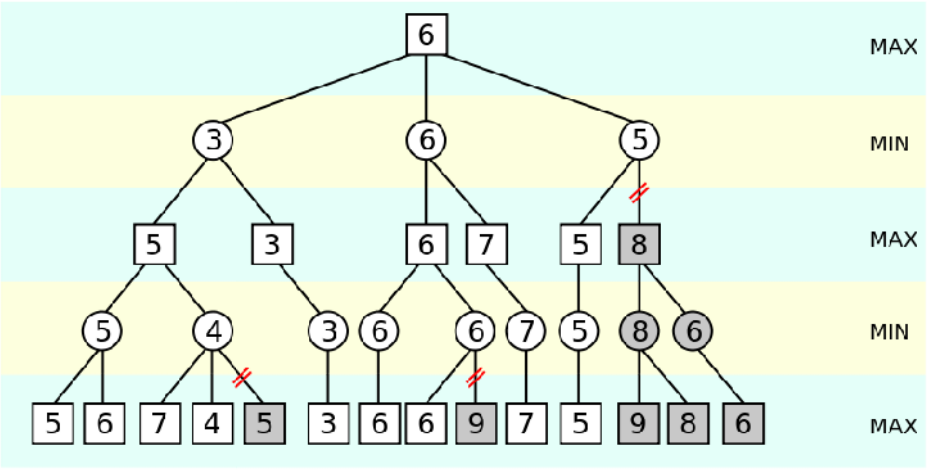


Figure 13 Alpha-Beta pruning tree example

Figure 13 shows an example of Alpha-Beta pruning. In naïve minimax pruning all of the nodes would have been examined, however certain nodes yield a position that is less

effective than the nodes already visited. The less effective nodes will have no influence on the final result of the search result, they therefore can be eliminated (greyed out in the illustration) improving search time. In the illustration the Max and Min correspond to each player's turn, from the Artificial Intelligence's perspective, it would like to maximize its position evaluation whilst also minimizing the player's. Succinctly this means that the Artificial Intelligence makes sure it makes the best moves for itself whilst forcing the worst moves for the user (Russell & Norvig, 2010).

As shown by this implementation of the algorithm, Alpha-Beta pruning is recursive, each switch between MAX and MIN in figure 13 represents another self-call of the method and a switch between each player's turn

```
public Position alphaBeta(int player, Position position, int alpha, int beta, int depth){ //alpha beta search f
    if(depth == 0) return position;
    Position optimalPosition = null;
    MoveGenerator moveGenerator = new MoveGenerator(position,player);
    moveGenerator.generateMoves();
    Position[] positions = moveGenerator.getPositions();
    if(positions.length == 0) return position;

    Evaluator evaluator = new Evaluator();
    for(Position _position:positions){
        if(optimalPosition == null) optimalPosition = _position; // the first time running through set the firs
        if(player == GameData.HUMAN){
            Position opponentPosition = alphaBeta(GameData.COMPUTER,_position,alpha,beta,depth-1);
            int score = evaluator.evaluate(opponentPosition);
            if(score>alpha){
                optimalPosition = _position;
                alpha = score;
            }
        }else{
            Position opponentPosition = alphaBeta(GameData.HUMAN,_position,alpha,beta,depth-1);
            if(new Game(opponentPosition).isChecked(GameData.HUMAN)){
                return _position;
            }
            int score = evaluator.evaluate(opponentPosition);
            if(score<=alpha && level > 4) return _position;
            if(score<beta){
                optimalPosition = _position;
                beta = score;
            }
        }
    }
    return optimalPosition;
}
```

Figure 14 Implentation of Alpha-Beta Pruning

Figure 14 shows the implementation in the product of the Alpha-Beta Pruning algorithm. The value 'depth' is passed in as a parameter; this determines the difficulty of the Artificial Intelligence as if the tree is allowed to be greater in depth a more foresighted move can made by the AI. This means for the implementation, level one especially is very easily taken advantage of. At level one, the Artificial Intelligence does not have the foresight to know that it will lose a piece in return for taking a piece. It can only see that it can take a piece and therefore will always do so as it is unaware of the consequences. This isn't necessarily a problem however, as level one is supposed to represent the easiest difficulty and an incompetent opponent which is an appropriate opponent for a beginner to challenge.

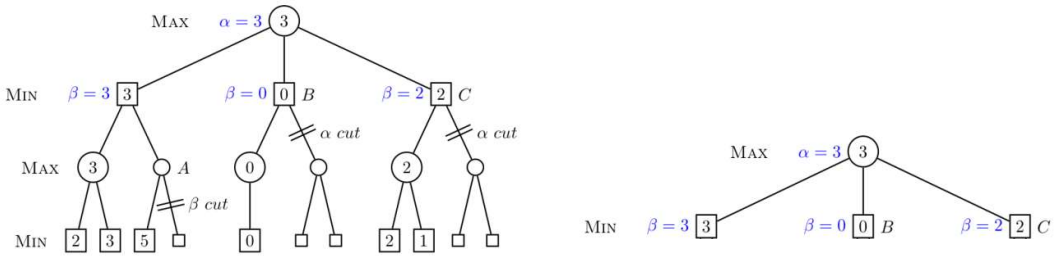


Figure 15 An Alpha-Beta pruning example of Depth/Difficulty 3 against a Depth/Difficulty of 1

4.1.7 Core Chess Game Classes - Piece.Java

This class simply contains the piece values for the Artificial Intelligence. The pieces follow a similar number scheme to the ones used by players; however the scale that the in game uses is greater to avoid using doubles to represent them.

```
public class Piece {
    public final static int PAWN = 100; // numerical values to refer to the pieces of chess
    public final static int KNIGHT = 320;
    public final static int BISHOP = 325;
    public final static int ROOK = 500;
    public final static int QUEEN = 900;
    public final static int KING = 1000000;
```

Figure 16 Piece values in the implementation

It is worth noting that the King piece is given a value of 1,000,000. Whilst the King piece's value is indefinite, a value of one million is enough in relation to all other pieces to replicate that effect.

4.1.8 User Interface - ProfilePane.Java

The profile pane in the game displays the player's experience, level and rank. This is accompanied by a rank image to give the user a sense of accomplishment and to encourage them to keep playing. It is worth noting the change in experience needed to level up, this is determined by the equation declared in section 3.3.1.

Figure 17 Profile at different levels with their corresponding emblems

4.1.9 User Interface - LearnPane.Java

The ‘LearnPane’ displays information about the chess pieces. The user can scroll through all the chess pieces and see information about them.

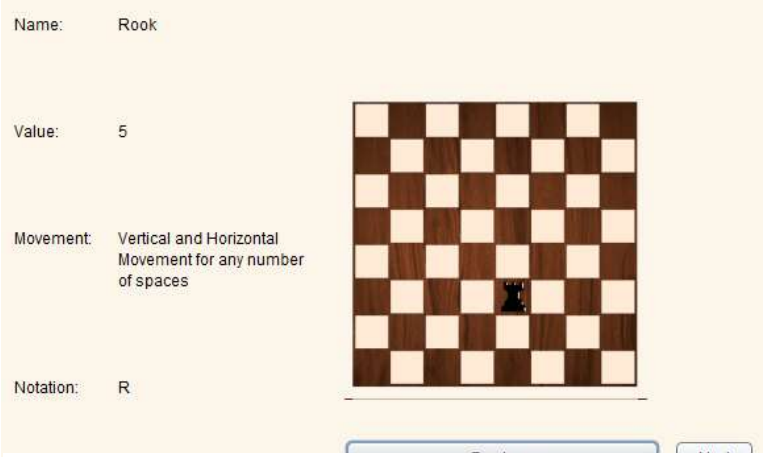


Figure 18 Rook information in the learn pane

Figure 18 shows an instance of the Learn Pane, the currently selected piece is the Rook. The value of the piece, a description of how it moves and its notation is displayed to the user. This information is accompanied by a moving gif file which shows the piece in action. The pieces are iterated through in order of value starting at pawn and finishing with the kingpiece. The next and previous buttons scroll through an array of PiecesDataClass.Java’ objects.

4.1.10 User Interface - EvaluationPane.Java

The evaluation pane gives the user real time in game feedback on their current position. The evaluation categories are split into pawn, bishop, knight and overall. This is due to how the 'EvaluatorForFeedback.java' (see 4.1.3) evaluates a user's position. Negatively evaluated pieces are colour coded red and likewise positively evaluated pieces are coloured green. This helps the user at a glance see which pieces they need to develop more.

Figure 19 Evaluator in effect

4.1.11 User Interface – HelpPane.Java

The help pane is the main focus of the chess game project. An instance of the help pane is generated when the user presses the 'help' button. This provides the user with situation specific aid. The program takes into account the level of difficulty the user selected, the phase of the game (early game, midgame or endgame) and if a game has been started or not. The advice given to the user is constructed used the 'Advice.Java' class. Different pieces of advice are appended together at call-time to suit the situation at hand.

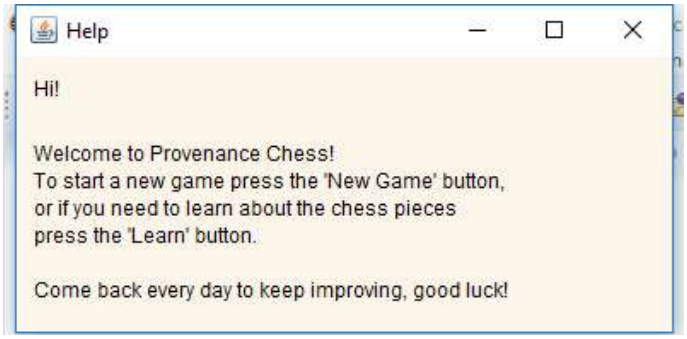


Figure 20 Help given before a game has been started

Figure 20 displays the help given to the user before they have started the game. It lets them know how to get started with the program. If the user creates a new game and selects either difficulty 1 or 2, a help pane is automatically created as displayed in figure 21.

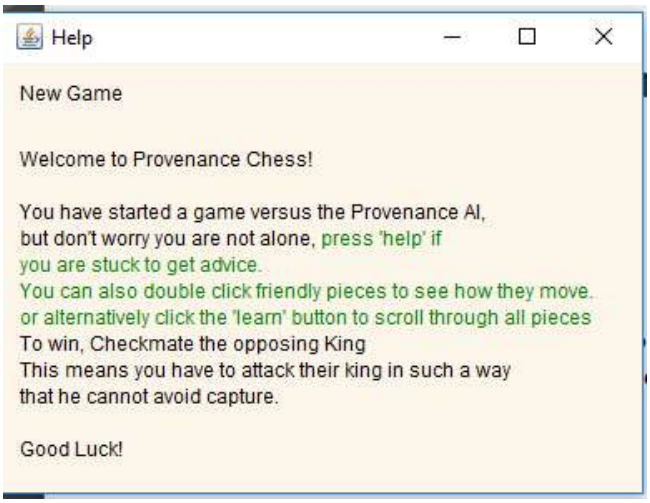


Figure 21 automatically created help pane for beginners

It is assumed that user's selecting a difficulty of 3 or above do not require the beginners advice so it is not shown. Figure 22 displays the help pane if accessed during an early game on a low difficulty. The chess manoeuvres are given a more extensive description as the user most likely does not know their definition. It also provides the user with specific advice on how to beat the low level Artificial Intelligence (as explained in 4.1.6). At difficulties of 3 or above the advice is much more minimal as it doesn't give a description of what the manoeuvres are, rather just which manoeuvres to carry out.

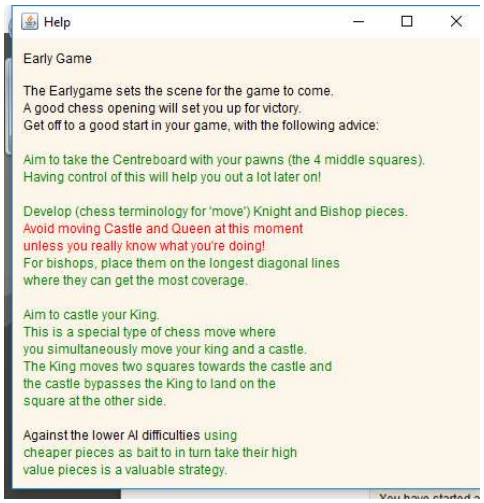


Figure 22 help pane for early game beginners

Similarly to the evaluation pane the advice is colour coded. Things that the user should try to accomplish are coloured green whilst things that must be avoided are coloured red. The amount of red text is limited, this bring attention to what little red text there is present as the mistakes the beginner should avoid making are more important.

4.1.12 User Interface - PreferencesPane.Java

The Preferences pane is shown to the user when they start a new game; this simply allows the user to select their colour out of black or white and the difficulty of their Artificial Intelligence opponent.

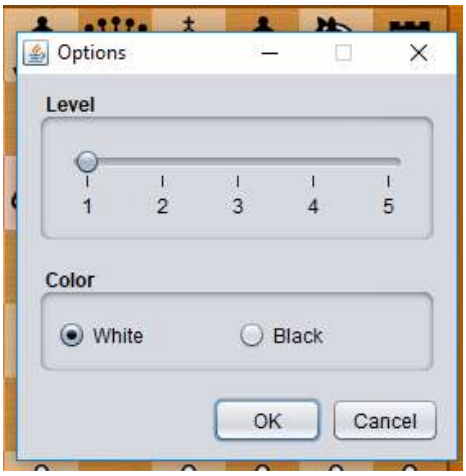


Figure 23 preferences pane instance

4.1.12 User Interface - PromotionPane.Java

The promotion pane is displayed to the user once a pawn reaches the opponents edge of the board, the user can choose to upgrade their pawn to a major or minor piece.

Figure 24 pawn promotion pane

4.2 Testing

It is worth noting that an evaluator experienced a crash with the system during their evaluation, a “java.lang.arrayindexoutofboundsexception” was thrown whilst moving a piece causing the program to become unresponsive. The crash has been not been replicated since however, it is still an issue that needs solving.

4.2.1 Black Box Testing

Black box testing was chosen as the method for testing the program as the user at the end will be using the program in this way. Black box testing assumes no knowledge of the internals of the program, only the functionality is tested. With this in mind an array of tests were created to make sure all the buttons work and everything works as intended.

Test Name	Result
Open Profile Pane	Pass
Open Learn Pane	Pass
Open Help Pane before game start	Pass
Iterate through Learn Pane to end	Pass
Iterate back through Learn Pane to Beginning	Pass
Start new game, White, 1	Pass
Start new game, White, 2	Pass
Start new game, White, 3	Pass
Start new game, White, 4	Pass
Start new game, White, 5	Pass
Start new game, Black, 1	Pass
Start new game, Black, 2	Pass
Start new game, Black, 3	Pass
Start new game, Black, 4	Pass
Start new game, Black, 5	Pass
Open Help Pane as game start	Pass
Pawn move options	Fail (it overwrites the pieces)
Knight move options	Pass
Bishop move options	Fail (it doesn’t show all)
Rook move options	Fail(it doesn’t show all)
Queen move options	Fail(it doesn’t show all)
King move options	Pass
Open Help pane at midgame	Pass
Close history pane midgame	Pass
Double click piece to show Information	Pass
Promote pawn	Pass
Check opponent	Pass
Be checked by opponent	Pass

Castle King	Pass
Checkmate opponent	Pass
Start new game immediately	Pass
Level up	Pass
Quit midgame	Pass
Quit post-game	Pass
Open History pane pre-game	Pass
Attempt to start a new game mid-AI move	Pass

Table 1 black box tests

The program passed most of the tests ran on it. From the testing it is apparent that there is a problem with the code in displaying the possible move options of certain pieces, they either don't display all of the possible moves or they place the glow over the top of the piece they can take. An example of this is shown in the figure below. The queen can take the knight piece but that square is not highlighted.

Figure 25 potential move highlighting not working

Whilst it is important that the whole game runs as intended, this particular section isn't too influential in the overall game experience. This bug will have to be fixed for a more stable release.

4.2.2 Resource Usage

The chess game was tested on both Windows and Macintosh. The Windows computer used for testing is a desktop computer that has a 3.5 GHz i5 3570k CPU with 8GB of 1366 MHz Random Access Memory. The Macintosh computer being used for testing is a laptop that has a 1.6 GHz i5 CPU with 4GB of 1600 MHz Random Access Memory. The Windows platform being used has a better performing CPU, it is expected that the program runs more effectively on that platform. Two tests were run on both platforms with the task manager (Windows) and activity monitor (Macintosh) running simultaneously during both. The Artificial Intelligence Chess Engine opponent was set to a difficulty of 1 for the first test (meaning the Alpha-Beta pruning search had a depth of 1) and set to a difficulty of 5 for the second. The tests were also timed to gauge the average waiting time for a move.

Platform	Test	Time Taken	CPU Usage	Memory Usage
Windows	Difficulty 1	Instantaneous	2.6%	692.4MB
Windows	Difficulty 5	~4 Seconds	33.9%	688.1MB
Macintosh	Difficulty 1	Instantaneous	13.8%	760.2MB
Macintosh	Difficulty 5	~7 Seconds	100%	780.1MB

Table 2 performance test results

The level 5 difficulty as expected took much longer than the level 1 difficulty to make a move. Most problematically the Alpha-Beta pruning with a depth of 5 completely maxed out the Macintosh laptop's CPU. This is due to the fact that the Macintosh laptop's Central Processing Unit is only a duo core unit that has a relatively low clock rate compared to the windows desktop. Due to this strain on system resources it is not recommended that lower end system's play above a difficulty of 4. Another solution to this could be a refinement and optimisation of the code used in the program to be less straining on CPU resources.

5. Evaluation

Chapter Summary: This chapter explains whether the implementation was of sufficient quality to meet the aims and objectives outlined in the introduction. Evaluators were used to test the product. A self-evaluation was also carried out in relation to the literature survey and design for effective feedback.

5.1 User Evaluation

This sub-chapter explores the results of the user evaluation and shows the exact evaluation form used by the evaluators.

5.1.1 Evaluation Form

The Evaluation form comprised of 4 initial questions followed by a System Usability Scale evaluation. The evaluation form used can be found in the appendices as appendix C. The questions were designed to gather information about the core aspects of the chess game's purpose. A final question, "What improvements could be made to enhance the Chess Game Experience?" was used, as this is an effective indirect way of getting evaluators to describe what they thought was ineffective in the chess game program.

5.1.2 Evaluator distribution

A total of 7 evaluators participated in the evaluation, a unique ID and a corresponding chess proficiency level was given to the user before they began the evaluation to allow for an explanation of the evaluators themselves. An ordinal scale was created for the evaluator labels, the scale is as follows:

- Absolute Beginner – someone that has never played chess before, they do not understand how the pieces move or how to play.
- Beginner – someone that has very limited experience with chess, they have a loose understanding of the pieces and how to play.
- Intermediate – someone that has experience with chess, they understand how the pieces move and how to win but are not proficient at doing so.
- Advanced – someone that has a decent level of experience with chess, they completely understand how the pieces move, can utilise more advanced chess manoeuvres and can win in an effective manner.
- Grandmaster – someone that has a master level of experience with chess, they can be considered flawless chess players with the most optimal chess move being made almost every turn.

The distribution of the evaluators can be visualised in the following table.

Experience	Amount
Absolute Beginner	1
Beginner	1
Intermediate	4
Advanced	1
Grandmaster	0

Table 3 evaluator distribution

As shown in table 1, the evaluator distribution is very uneven. The mode experience level was intermediate, with more people being from this category than all other categories combined. For a more effective evaluation more people from the other categories should be included. Having a grandmaster player evaluate the game could also be useful; however this would understandably be difficult to carry out as there are not a great amount of them.

5.1.3 Evaluator's Feedback

The Evaluator's feedback is visualised in the graphs below. 'T' in the graph represents an answer 'True' in response to the question and likewise 'F' represents 'False'.

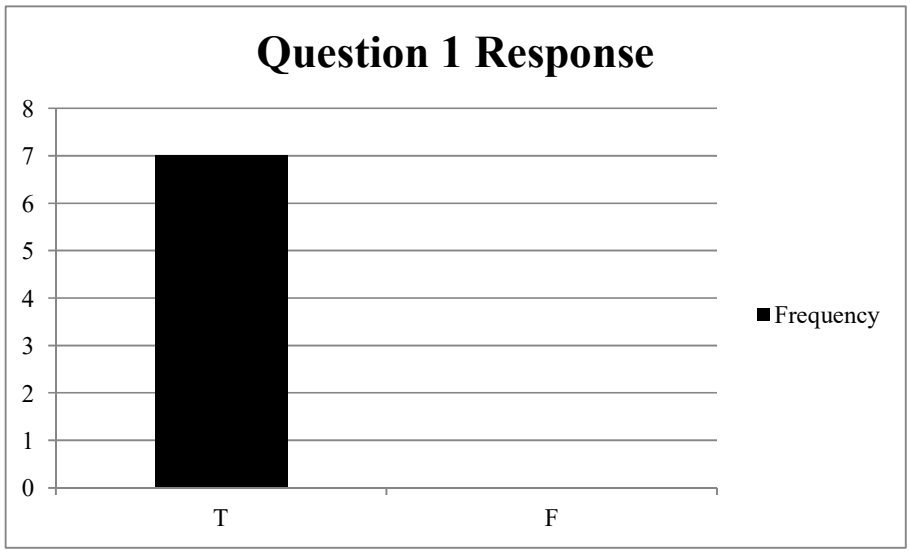


Figure 26 I found all necessary information in the place I expected it to be.

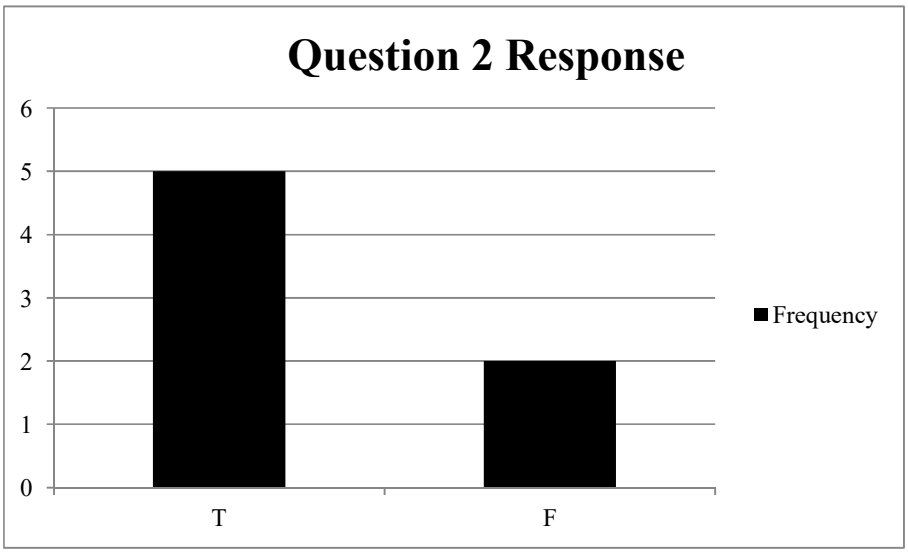


Figure 27 I found the difficulty level of the chess AI matched the difficulty level that I expected to play against.

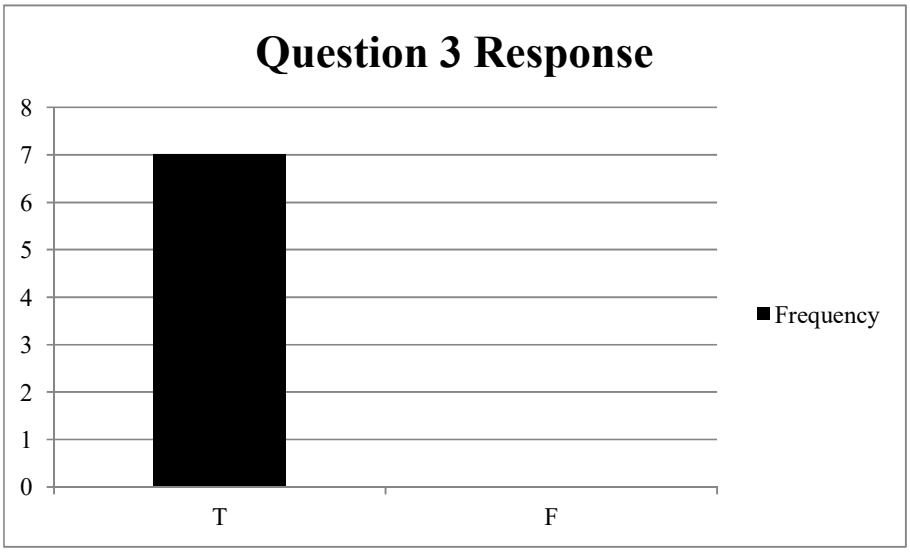


Figure 28 I found the in-game assistant to be helpful

Based on this evaluator feedback, the most problematic part of the chess game was the Artificial Intelligence difficulty level. Two evaluators answered false when asked if they found the difficulty to be what they expected. When pressed further for why, both answered that they found the Artificial Intelligence to be too difficult to defeat. It is also worth noting that these two answers were given by the Beginner and Absolute Beginner players. Both players were playing at a difficulty of '1', whilst the Artificial Intelligence is incredibly flawed at this level especially, the hyper-aggressive play style of the level one chess AI shocked beginner players and they struggled to handle a quick loss of material.

The System Usability Scale results are displayed in the table below.

As this scale is numeric, a mean average can be calculated. The mean of these results is 90. A System Usability Scale average value above 68 can be considered above average and a score below 68 likewise below average (usability.gov, 2017). This places the evaluation of this system far above the average. The system can therefore be considered to have a sufficiently effective User Interface.

Evaluator	SUS Result
1	85
2	85
3	87.5
4	97.5
5	100
6	90
7	85

Table 4 system usability scale results

5.1.4 Feedback Correlations

To try and find a correlation from the evaluator's feedback, the results were ran through WEKA with a J48 decision tree classifier used for its ability to visualise the classification method. The classifier was trying to predict the answer to question 2, the problematic question at hand. The classifier found that if the user gave the system a relatively low SUS score, they were more likely to find the chess AI too difficult.

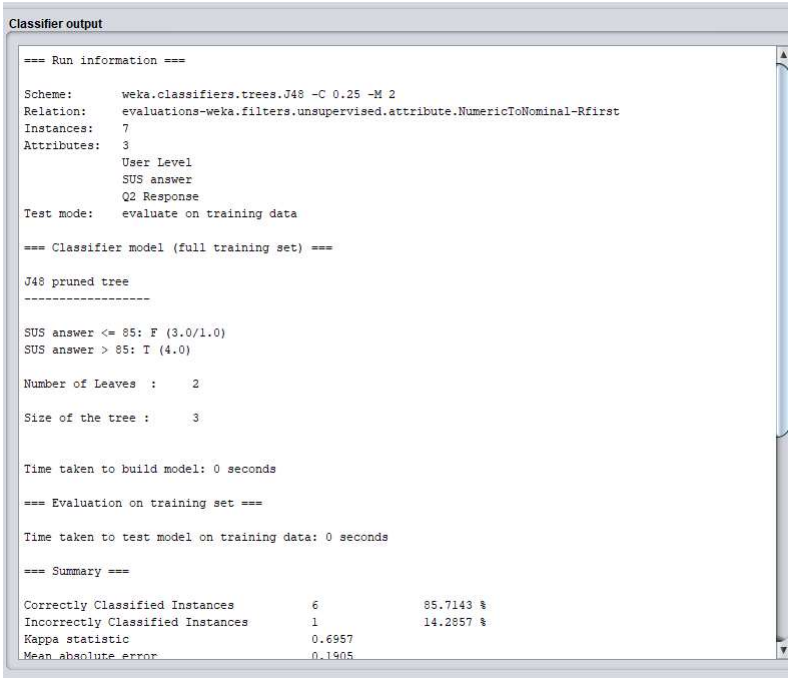


Figure 29 decision tree classification of evaluator results

This correlation suggests that an improved User Interface would make more users believe that the chess AI opponent was of an appropriate difficulty.

To solve the potential problem of the chess AI being too difficult at lower levels for beginners, 2 solutions are proposed. Firstly, change how the AI is constructed at lower levels to make the AI itself make even less effective moves, or improve the User Interface of the program itself to aid the user which in turns makes the difficulty of the AI less daunting. These potential solutions could be explored as further work.

5.2 Self-Evaluation of the Product

5.2.1 Literature Survey to Implementation Relation

In relation to the Literature Survey, the implementation of the chess game can be considered a success. The literature survey pointed out the effective user feedback method in the game Civilization V. The implementation of the chess game evaluator is clearly inspired by this design and brings about the clear user feedback system which was direly missing in all chess implementations so far.

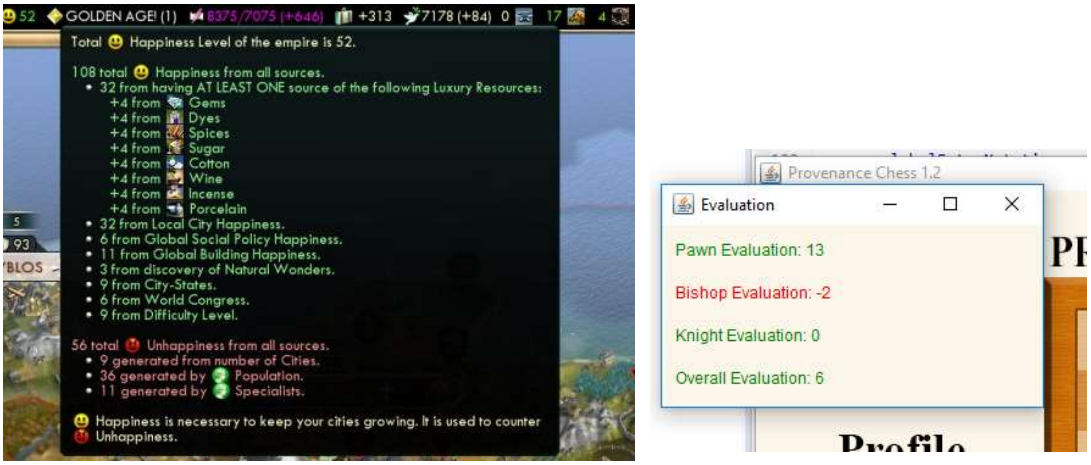


Figure 30 Civilization V's user feedback in contrast to the chess game's user feedback

The ‘help pane’ implementation also was modelled on Civilization V’s ‘advisor’ feature. There are very visible similarities between the two in practice and both have their success in their implementation.

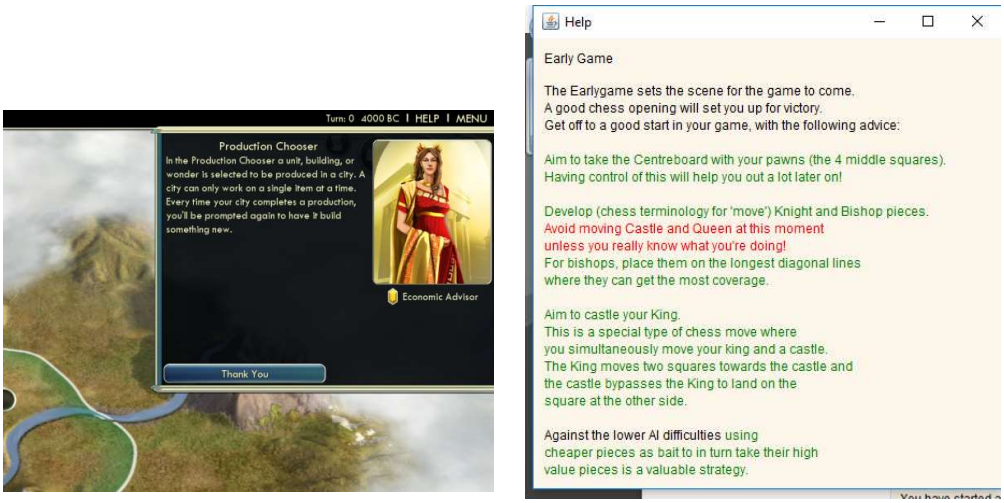


Figure 31 Civilization V's advisor in contrast to the chess game's help pane

When contrasted to the related works, Chesscademy and Chess.com, a great improvement was made through this chess game. The chess game gives the user some valuable in game feedback whilst the related previous works essentially provided none – an easily visible improvement. This improvement coupled with the fact that all evaluators thought the help pane was useful provides confidence in the improvement of this chess game over all other previous implementations.

The Chess Game Artificial Intelligence opponent was also modelled after an implementation discussed in the literature survey. The Stockfish Open Source Artificial Intelligence Chess Engine uses alpha beta pruning as its tree searching algorithm as does the chess game created in this project. The main differences between the Stockfish chess engine implementation and this project’s chess engine is that Stockfish is written in C++, a powerful and more importantly quicker programming language, and Stockfish’s position evaluator is much more advanced. As discussed in the literature survey however, an improvement on Stockfish’s Chess Engine was not the focus of this project as it was deemed unfeasible (Stockfish won the Top Chess Engine Championship in 2016 & 2017) (TCEC, 2018).

5.2.2 Design to Implementation Relation

There are a few differences between the designed program and the final implementation. Some improvements were made during the implementation stage; however, some parts of the design weren't implemented as designed. One such section was the 'Learn' menu option. The fact files presented were not as thorough as initially intended but the addition of a moving gif improved the presentation of the form greatly. The 'advisor' feature as described in the design chapter of the dissertation was eventually named 'evaluator' in the implementation. Apart from these two features, the implementation followed the design completely and therefore can be considered satisfactory in relation to the design.

5.3 Self Performance Reflection

During this project, the author's performance could be rated fairly highly. The fundamental coding for the product began relatively early, during the 3rd term of the 2nd year of University and during the summer between the 3rd and 2nd year. This proved to be very advantageous as it allowed for a focus on the User Interface and the research that clearly defined the contribution of the project. The schedule in appendix A was followed to an appropriate degree, ensuring that progress was consistently being made on the project throughout the duration. Overall, in each aspect of the project an adequate amount of effort was put in to achieve the results necessary.

6. Conclusion

Chapter Summary: This chapter recaps the project overall, discusses the key points and the overall outcome of the project. A discussion on what future work could be carried out is also present.

6.1 Success and Failures of the Project

This project aimed to develop a java application for use by beginner chess players that aided users during their chess matches to help them make better decisions and to improve as chess players. The implementation of the designed program achieved this aim via the implementation of the in game advisor and the evaluator which gave the user real time feedback on their chess play. The application was tested by multiple evaluators, two of which were beginner chess players, with overall positive results achieved. The evaluation in general found the project to be successful in achieving the aims. The program achieved an average of 90 on the System Usability Scale, placing it above the minimum acceptable score of 68. With the evaluation completed, there can be confidence placed in the program and its ability to fulfil the objectives outlined in the beginning chapter of this report.

6.2 Overcoming Difficulties in the Project

There were multiple aspects of the project that were critical and had difficulties associated with their implementation. One such aspect was how to begin creation of the chess AI engine. This difficulty was overcome by extensive research into the field. Multiple resources were found that helped to conceptualise and implement the chess engine. One such resource was the Chess Wiki (Chess Programming WIKI, 2018) which is due to close September 30th 2018. This provided a good reference point for the creation of the engine throughout the entire project. Many other resources were found online to aid in this aspect (Laramée, 2000), especially with the explanation of the Alpha-Beta Pruning algorithm (Frayn, 2005).

Another difficulty in the project was creating the Human Computer Interface, for this aspect research was done in the field. Another method that was used to overcome this issue was self referencing the previous year of work as creating Java Swing environments was part of the 1st term of the 2nd year of the Computer Science course. The lab work outlined by the tutors was used but re-adapted to fit the chess game.

6.3 Further Work

Whilst the program performed fairly well, some optimisation should be carried out to improve performance on less powerful Central Processing Units (see 5.3.2). This could be done by re-implementing the MoveSearcher class (see 4.1.6 and 3.2.7) from the recursive method to an iterative method. The visuals of the program could also be improved. The Java Swing toolkit was used to create the User Interface in this instance, however, JavaFX could be used instead as it is a more modern and more effective method of creating user interfaces (Terpilowski, 2013). This could improve the design and overall layout of the program, further increasing the visual appeal of the program and heightening user retention when using it. In terms of evaluation, an improve evaluator distribution could also be sought after. Having an increased amount of absolute beginner players (the target audience) would be welcomed as it would vastly improve evaluator's feedback.

References

Adams, E., 2013. *Fundamentals of Game Design.* s.l.:New Riders.

Allebest, E. & Severson, J., 2007. *Chess.com,* s.l.: s.n.

Anon., 2014. *Laws of Chess.* [Online].

App Annie, 2015. *5 Proven Ways You Can Increase Retention for Your Games.* [Online]
Available at: https://www.appannie.com/en/insights/mobile-gaming/increase-user-retention-for-games/

Bernstein, S., 2012. Valedictorians of the Soviet School: Professionalization and the Impact of War in Soviet Chess. *Kritika: Explorations in Russian and Eurasian History Volume 13, Number 2 ,* pp. 395-418.

Chess Central, 2018. *Chess Strategy For Chess Openings and Chess Principles.* [Online]
Available at: https://www.chesscentral.com/pages/learn-chess-play-better/chess-strategy-for-chess-openings-and-chess-principles.html

Chess Programming WIKI, 2018. *Home.* [Online]
Available at: http://chessprogramming.wikispaces.com/
[Accessed 26 April 2018].

Chess Programming, 2017. *Horizon Effect.* [Online]
Available at: https://chessprogramming.wikispaces.com/Horizon+Effect
[Accessed 12 December 2017].

CHESScom, 2017. *The Best Chess Openings For Beginners.* [Online]
Available at: https://www.chess.com/article/view/the-best-chess-openings-for-beginners

Chisholm, H., 1911. Chess. In: *Encyclopaedia Britannica.* Cambridge: Cambridge University Press, pp. 93-106.

Costalba, M., Kiiski, J., Linscott, G. & Romstad, T., 2017. *GitHub - official-stockfish.* [Online]
Available at: https://github.com/official-stockfish/Stockfish
[Accessed 16 11 2017].

Dehghani, H. & Morteza, S., 2017. *A GA based method for search-space reduction of chess game tree.* Kashan: University of Kashan.

Fine, R., 2003. *Basic Chess Endgames.* s.l.:Random House Puzzles & Games, Revised Edition.

Frayn, C., 2005. *Computer Chess Programming Theory.* [Online]
Available at: http://www.frayn.net/beowulf/theory.html#abpruning
[Accessed 6 October 2017].

Gaspard, F., 2009. *How to Play Chess: Chess Middle Game Tips* [Interview] (6 July 2009).

Hudson, M. A., 2013. *Storming Fortresses: A political History of chess in the Soviet Union, 1917-1948,* Santa Cruz: University of California.

Keene, R. & Levy, D., 1993. *How to Play the Opening in Chess.* Batsford: Henry Holt & Co.

Kolb, D. A., 2014. *Experiental Learning: Experience as the Source of Learning and Development.* 2nd ed. s.l.:FT Press.

Laramée, F. D., 2000. *Chess Programming Part I: Getting Started.* [Online]
Available at: https://www.gamedev.net/articles/programming/artificial-intelligence/chess-programming-part-i-getting-started-r1014
[Accessed 4 September 2017].

Lasker, E., 2009. *Lasker's Manual of Chess, New Edition.* s.l.:Russell Enterprises.

Max, E., 1997. *The Development of Chess Style.* s.l.:Intl Chess Enterprises.

Meier, S., 2010. *Civilization V.* s.l.:Firaxis.

Meier, S., 2010. *The Psychology of Game Design (Everything You Know Is Wrong).* San Francisco(CA): Game Developers Conference.

Newborn, M., 1997. *Kasparov verses Deep Blue: Computer Chess Comes of Age.* New York: Springer-Verlag.

Ng, A., Hinson, F. & Dasgupta, S., 2015. *Chesscademy,* s.l.: s.n.

Nunes, J. C. & Dréze, X., 2006. *The Wndowed Progress Effect: How Artificial Advancement Increases Effort.* Los Angeles: University of Southern California.

Phillips, K. R. & Bruce, C. S., 1990. *KC Chess.* Fredericton: University of New Brunswick.

Russell, S. J. & Norvig, P., 2010. *Artificial Intelligence: A Modern Approach.* 3rd ed. New Jersey: Pearson.

Sheridan, H. & Reingold, M. E., 2014. Expert vs Novice differences in the detection of relevant information during a chess game: evidence from eye movements. *Frontiers in Psychology.*

Solomon, J., 2016. *Firaxis' Jake Solomon on what went wrong with XCOM2* [Interview] (25th February 2016).

Tate, P. J. & Keeton, M. T., 1978. *Learning by Experience - What, Why, How.* 1st ed. s.l.:Jossey-Bass.

TCEC, 2018. *Top Chess Engine Championship.* [Online]
Available at: http://tcec.chessdom.com/archive.php?ma=3&ga=1&ag
[Accessed 14 April 2018].

Terpilowski, R., 2013. *Pro, cons of moving from Swing to JavaFX: UI tools a plus* [Interview] (October 2013).

usability.gov, 2017. *System Usability Scale (SUS).* [Online]
Available at: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html
[Accessed 14 April 2018].

usability.gov, 2017. *User Interface Design Basics.* [Online]
Available at: https://www.usability.gov/what-and-why/user-interface-design.html
[Accessed 20 April 2018].

Voigt, B., 2017. *stackexchange.* [Online]
Available at: https://math.stackexchange.com/questions/2323103/can-anyone-help-me-with-a-graph-exponential-but-never-reaches-1
[Accessed 4 January 2018].

Waffllemaster, 2012. *Minor and Major pieces.* [Online]
Available at: https://www.chess.com/forum/view/general/minor-and-major-pieces

World Chess Federation, 2014. *Laws of Chess.* [Online]
Available at: https://www.fide.com/fide/handbook.html?id=171&view=article
[Accessed 4 November 2017].

Zaifrun, 2010. *Creating a chess engine from scratch (part 1).* [Online]
Available at: https://www.chess.com/blog/zaifrun/creating-a-chess-engine-from-scratch-part-1
[Accessed 26 11 2017].

Appendices

A. Terms of Reference

Course Specific Learning Outcomes

A student successfully completing this project will:

- Use knowledge, abilities and skills for further study and for a range of employment in areas related to scientific and technical computing.
- Interpret legislation appropriate to computer professionals and also be aware of relevant ethical issues and the role of professional bodies.
- Analyse, design, and implement algorithms using a range of appropriate languages and/or methodologies.
- Design, implement and interrogate database systems.
- Demonstrate effective communication, decision making and creative problem solving skills, and identify appropriate practices within a professional, legal and ethical framework.
- Critically appraise and apply suitable artificial intelligence techniques for a variety of software systems.
- Identify and use a range of approaches and techniques for the design and development of digital games.
- Recognise and critically asses the underlying principles guiding the design of relevant visual, audio, interactive, and narrative aesthetics for digital games.

Project Background

A Strategy game is a game in which the players must use intelligent decision making and situational awareness to defeat their opponent. Chess is one of the most popular and frequently played strategy games. The game is a two player board game in which the players must use their 16 pieces to force their opponent into a checkmate scenario, at which point the opposing king piece cannot avoid capture.

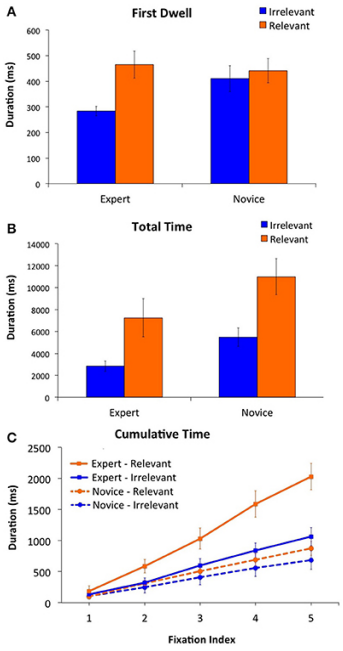
Players can use their pieces, which range in power from pawn to queen, on the 8 by 8 chequered game board to capture opposing pieces whilst protecting their own. Each type of piece having its own specific move set which allows for complex strategies to be developed to force a checkmate.

Beginning in the 20th century, chess engines have been in development - computers, programmed to analyse the board and make a decision on the best chess move. A historic moment for chess engine development was when IBM's Deep Blue machine defeated the World Chess Champion Garry Kasparov in 1997 (Newborn, 1997). In more recent times, online chess sites such as Chess.com which allow for player versus player and player versus AI have become popular.

For a beginner chess player these sites may not be optimal, as whilst they do often have a learning section, they overload the user with information and attempt to teach outside of the game environment through videos and isolated scenarios. Aiding the user in learning to play chess is not the sole focus of these sites.

A more dedicated chess learning site, ‘Chesscademy’ also employs the use of video but includes quizzes to ensure the user is aware of the value of each piece. A criticism of this teaching style is that, whilst the videos help the user to learn the fundamentals of the game, it does not necessarily help the user when in game to make better chess moves. The problem derived from this, is that currently, there is no specifically beginner user oriented chess software.

A 2014 study carried out by the Psychology departments of Toronto and Southampton Universities discovered a great difference in the amount of time novices spent dwelling on irrelevant chess information when compared to expert players. As shown in part B of the charts, novice players spent almost double the amount of time considering irrelevant information when compared to experts, furthermore, “The experts were faster at detecting relevant information than the novices, as shown by the finding that experts (but not novices) were able to distinguish between relevant and irrelevant information during the early part of the trial.”. (Sheridan & Reingold, 2014) Novices also spent a much greater amount of time before making a move.



The purpose of this project is to create an interactive chess simulation which will aid the user in identifying the most relevant pieces on the chessboard on any given turn. This can be done through the development of a computer algorithm which can detect which pieces that are the most relevant and display this information to the user.

During its time the Soviet Union dominated the chess world, excluding the years of Bobby Fischer. This was mainly due to the reason that “chess became a significant cultural component in the lives of many Soviet citizens.” (Hudson, 2013) The Soviet population engaged massively with their national pastime, which they considered a sport rather than an art or science. They came to dominate international play, largely due to the “founding of the state-run chess section in 1924” (Bernstein, 2012). Engagement played a massive part in the proliferation of skilled Soviet chess players; therefore engagement is a fundamental key in creating expert chess competitors.

To be an engaging simulation, the game should follow Vandenberghe’s five domains of play and provide for the user: Novelty, Challenge, Stimulation, Harmony and Threat. (Adams, 2013). The game must give challenge and feedback to the user for them to keep engaging with the program and therefore improving.

Aim

The Chess Game project aims to create an offline Java chess application which provides support during a match to better the user's decision making skills, an accessible Graphical User Interface to provide a comfortable and intuitive environment, and the design and implementation of an effective chess engine.

Objectives

To achieve the aim specified above, certain objectives must be met. These objectives are:

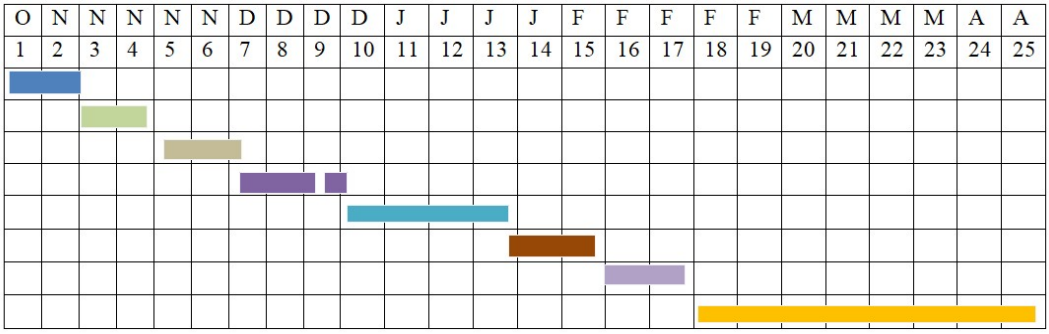
- Research existing chess simulation solutions and determine the positive and negative aspects of each.
- Research existing chess engines algorithms & criticise.
- Research game design fundamentals & how to keep user engagement high within a game environment.
- Design basic chess game interface & menu screens.
- Implement the basic chess environment in java IDE.
- Design Chess engine algorithm with the intention of varying difficulties.
- Design user friendly 'Chess Aid', using research of important chess strategies.
- Implement designed AI opponent, with multiple stages of difficulty.
- Implement chess aid program.
- Present program to participants of the course and receive feedback in the form of pre-made questionnaire.
- Evaluate the chess algorithm and the chess aid's effectiveness in providing an engaging and improving experience for user.
- Implement Amendments based on participant feedback & finalise chess program.
- Produce final report & documentation.

Critical stages of work

Within the project lie a few critical stages of work. One of these stages is the implementation and merging of the chess piece classes, movement conditions and endgame conditions onto the graphical user interface, each piece's movement must reflect graphically the movement conditions of that class, this is the first critical stage of work as it forms the basis for the chess game. The second critical section is the creation of the chess engine and implementation into the existing chess environment. Finally, the creation of the user aid is a critical section of work, as it forms the basis of the evolutionary step of the chess program. This section must be created to be as user and beginner friendly as possible.

Schedule & Deliverables

Task name	Duration
Literature review	2wk
Algorithm & GUI research	2wk
Program planning	2wk
Basic Chess game production	3wk
Chess Aid & algorithm creation	3.5wk
Participant testing & feedback	2wk
Feedback implementation	2wk
Final report & finalization	8wk



A git repository can be used to archive the iterations of the chess game once created. This is recommended.

Required Resources

To complete this project, a java IDE is required: either Eclipse or IntelliJ with the minimum hardware specification required to run said programs.

B. Product Code

The code for the product can be found at: https://stummuac-my.sharepoint.com/:u:/g/personal/15072935_stu_mmu_ac_uk/EaRv3QeN38VFrtxSj6yHOBoBj08POfqnnNPTLklMr2qlQQ

C. Evaluation Form

Evaluation – Chess Game

User ID:

Question 1: I found all necessary information in the place I expected it to be

True False

If false, why?

Question 2: I found the difficulty level of the chess AI matched the difficulty level that I expected to play against

True False

If false, why?

Question 3: I found the in-game assistant to be helpful

True False

If false, why?

Question 4: What improvements could be made to enhance the Chess Game Experience?

System Usability Scale

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

D.Stockfish Technical Evaluation

Stockfish is an open Source chess engine written in C++ (Costalba, et al., 2017). Stockfish regularly is ranked in the top 2 chess engine rating lists, with currently only one chess engine, Komodo, having a higher rating. After analysis of the code used in stockfish, it is apparent that stockfish takes into account many factors when evaluating a position. Similarly to how humans do, with each factor having its own specific weight in the outcome of the turn. To evaluate a chess position, Stockfish accounts for the following things:

- Material – the amount of pieces on the board. Having more pieces is more advantageous than having less.
    - Each piece has a unique value which changes as the game progresses:
        - Pawn Midgame = 198, Pawn Endgame = 258
        - Knight Midgame = 817, Knight Endgame = 846
        - Bishop Midgame = 836, Bishop Endgame = 857
        - Rook Midgame = 1270, Rook Endgame = 1281
        - Queen Midgame = 2521, Queen Endgame = 2558
    - Human chess players, use the value system:
        - Pawn = 1
        - Knight & Bishop = 3
        - Rook = 5
        - Queen = 10
    - The values used by the stockfish engine have a similar ratio between pieces to what human players use.
- General Piece placement – controlling certain squares is more advantageous.
    - Attacking or defending the centre squares on the board is better than controlling unoccupied squares.
    - Defending important friendly pieces is advantageous.
    - Attacking important pieces of the opponent is advantageous.
    - A Knight being in an outpost is advantageous.
    - Bishops controlling the major diagonal is advantageous.
    - Having both bishops is massively advantageous.
    - Having bishops in open positions is advantageous.
    - A King protected by friendly pieces is advantageous.
    - Having multiple pawns in the same file is disadvantageous, as all but the highest pawn have no mobility and cannot defend each other.
    - Isolated pawns are disadvantageous as they cannot defend other pieces.
    - Mobility – the summation of legal moves with bonus mobility for specific pieces. Rook vertical mobility is considered more valuable than horizontal.

Each one of these bullet points has its own unique weight that has been fine-tuned over the years. The evaluation does not account for everywhere the pieces could move in the future, and a positive evaluation does not ensure checkmate is imminent. Rather, evaluation gives a general sense of where a position is headed in the future, it can be thought of as an approximate derivative.

After evaluating positions, stockfish examines the resulting positions from all the legal moves available and chooses the most advantageous one, essentially searching for the best move. The problem with this is that a move may be beneficial in the short term, but a counter-move by the opponent could lead to the engine losing the game. The primitive response to this problem is attempting to consider all possible moves, all the way until checkmate, however there are far too many possible moves until checkmate is reached. Currently, it is not possible to solve every game of chess from the starting position all the way until checkmate with always having an optimal move.

After the first two turns, 400 possible board setups exist. After the second pair of turns there are 197,742. After three moves, 121 million possibilities exist. Approximately there are, 3.35 x10^123 potential moves in a typical chess game tree. (Dehghani & Morteza, 2017) A computer could not possibly examine all of these positions.

Instead of evaluating all the way until checkmate, Stockfish simply looks around 30+ nodes, and uses heuristics based on the stage of the game to assign a value to how advantageous the move really is based off of the resulting position. To do this, Stockfish constructs a tree full of increasing depth, consisting of moves followed by more moves, along with their values. While modern hardware can analyze hundreds of thousands of positions in seconds, multiple steps are still taken to prune and recognize patterns inside the tree.

The techniques used are as follows:

- Alpha-beta pruning is used to eliminate disadvantageous moves, or moves that after counter-moves result in extremely disadvantageous positions.
    - A transposition-table is created to store frequently visited positions, so no identical nodes are analyzed more than once.
- A forward pruning is used where rather than skipping an entire sub tree, the engine searches down it to a reduced depth. The advantage of this is that you get most of the saving but with much lower risk than pruning entire sub trees like with alpha-beta pruning. This comes from the idea that from any given position the opponent will be able to find at least one move that improves their position.
- Opening heuristics
    - Developing minor pieces is advantageous.
    - Moving the queen piece out early is disadvantageous.
    - Castling as soon as possible is advantageous.
    - Controlling the centre pieces as soon as possible is advantageous.
    - A computer could also possibly use the first few moves from the chess standards as they are often the exact same each game.
- Mid-game heuristics
    - Setting the opponent up to make a disadvantageous move is advantageous.
    - Pinning important opposing pieces, setting up outposts is advantageous.
    - Moving pieces to optimal positions as soon as possible is advantageous.
    - Having the initiative by attacking the enemy pieces as white is advantageous.

- Defending as black until white makes a punishable move is advantageous.

The mid game is generally where chess engines are most likely to overtake human opponents, as they are much better at analyzing trades and positional play than humans.

Lastly, the end game is pre-solved. The engine simply compares its current position against table of potential positions called 6-man Syzygy Tablebases. It performs each turn following which next position in the table more likely results in a victory.

Stockfish is similar to most computer chess programs. It is a very efficient search that wastes as little time as possible looking into fruitless lines, while focusing on comparing those that give it the most advantageous position. These positions are analysed based on the above bullet points. Fundamentally, it has the ability to consistently evaluate positions and work that in conjunction with the search. An important part of Stockfish compared to other engines is that is it open source, anyone can contribute to its improvement. The minor tweaks and improvements have made it a very competitive chess engine.

Stockfish also provides the option to have a selected difficult out of 20. This is an aspect of stockfish that provides a lot of utility, as, at its best, Stockfish is above all human grandmasters. The difficulty levels are required so that a human could feasibly defeat the engine. Stockfish scales its difficulty by changing the fixed depth, the further it moves down the tree the more effective it is at choosing advantageous moves. It is very important that a chess engine provides a sufficient but not overbearing level of difficult for its players. This aspect of stockfish should be implemented in the chess program. Overall the Stockfish technique of having weighted factors contribute to the evaluation of a position, followed by a traversal of a tree can be applied to the solution.

E. Ethics Documents

Ethics forms can be found on the following pages.

ETHICS CHECKLIST

Manchester Metropolitan University

This checklist must be completed **before** commencement of **any** research project. This includes projects undertaken by **staff and by students as part of a UG, PGT or PGR programme**. Please attach a Risk Assessment.

Please also refer to the University's Academic Ethics Procedures; Standard Operating Procedures and the University's Guidelines on Good Research Practice

Full name and title of applicant:	Mr Vincenzo Scialpi-Sullivan
University Telephone Number:	07498621206
University Email address:	15072935@stu.mmu.ac.uk
Status: All staff and students involved in research are strongly encouraged to complete the Research Integrity Training which is available via the Staff and Research Student Moodle areas	Undergraduate Student ☑ Postgraduate Student: Taught ☐ Postgraduate Student: Research ☐ Staff ☐
Department/School/Other Unit:	Computing, Mathematics and Digital Technology
Programme of study (if applicable):	
Name of DoS/Supervisor/Line manager:	Amna Eleyan
Project Title:	Chess
Start & End date (cannot be retrospective):	03/07/2017 - 28/04/2018
Number of participants (if applicable):	15
Funding Source:	N/A

Brief description of research project activities (300 words max):

Development of Java chess game. Participant required to give feedback on game UI and appeal to beginner chess players.

	YES	NO
Does the project involve NHS patients or resources? If 'yes' please note that your project may need NHS National Research Ethics Service (NRES) approval. Be aware that research carried out in a NHS trust also requires governance approval. Click here to find out if your research requires NRES approval Click here to visit the National Research Ethics Service website To find out more about Governance Approval in the NHS click here	☐	☑
Does the project require NRES approval?	☐	☑
If yes, has approval been granted by NRES? Attach copy of letter of approval. Approval cannot be granted without a copy of the letter.	☐	☐

NB Question 2 should only be answered if you have answered YES to Question 1. All other questions are mandatory.	YES	NO
1. Are you are gathering data from people?	☑	☐
For information on why you need informed consent from your participants please click here		
2. If you are gathering data from people, have you:	☐	☐
a. attached a participant information sheet explaining your approach to their involvement in your research and maintaining confidentiality of their data?	☑	☐
b. attached a consent form? (not required for questionnaires)	☑	☐
Click here to see an example of a participant information sheet and consent form		
3. Are you gathering data from secondary sources such as websites, archive material, and research datasets?	☐	☑
Click here to find out what ethical issues may exist with secondary data		
4. Have you read the guidance on data protection issues?	☑	☐
a. Have you considered and addressed data protection issues – relating to storing and disposing of data?	☑	☐
b. Is this in an auditable form? (can you trace use of the data from collection to disposal)	☑	☐
5. Have you read the guidance on appropriate research and consent procedures for participants who may be perceived to be vulnerable?	☑	☐
a. Does your study involve participants who are particularly vulnerable or unable to give informed consent (e.g. children, people with learning disabilities, your own students)?	☐	☑
6. Will the study require the co-operation of a gatekeeper for initial access to the groups or individuals to be recruited (e.g. students at school, members of self-help group, nursing home residents)?	☐	☑
Click for an example of a PIS and information about gatekeepers		
7. Will the study involve the use of participants' images or sensitive data (e.g. participants personal details stored electronically, image capture techniques)?	☐	☑
Click here for guidance on images and sensitive data		
8. Will the study involve discussion of sensitive topics (e.g. sexual activity, drug use)?	☐	☑
Click here for an advisory distress protocol		
9. Could the study induce psychological stress or anxiety in participants or those associated with the research, however unlikely you think that risk is?	☐	☑
Click here to read about how to deal with stress and anxiety caused by research procedures		
10. Will blood or tissue samples be obtained from participants?	☐	☑
Click here to read how the Human Tissue Act might affect your work		
11. Is your research governed by the Ionising Radiation (Medical Exposure) Regulations (IRMER) 2000?	☐	☑
Click here to learn more about IRMER		
12. Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants or will the study involve invasive, intrusive or potentially harmful procedures of any kind?	☐	☑
Click here to read about how participants need to be warned of potential risks in this kind of research		
13. Is pain or more than mild discomfort likely to result from the study? Please attach the pain assessment tool you will be using.	☐	☑

Click here to read how participants need to be warned of pain or mild discomfort resulting from the study and what do about it.		
14. Will the study involve prolonged or repetitive testing or does it include a physical intervention?	☐	☑
Click here to discover what constitutes a physical intervention and here to read how any prolonged or repetitive testing needs to managed for participant wellbeing and safety		
15. Will participants to take part in the study without their knowledge and informed consent? If yes, please include a justification.	☐	☑
Click here to read about situations where research may be carried out without informed consent		
16. Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?	☐	☑
Click here to read guidance on payment for participants		
17. Is there an existing relationship between the researcher(s) and the participant(s) that needs to be considered? For instance, a lecturer researching his/her students, or a manager interviewing her/his staff?	☐	☑
Click here to read guidance on how existing power relationships need to be dealt with in research procedures		
18. Have you undertaken Risk Assessments for each of the procedures that you are undertaking?	☑	☐
19. Is any of the research activity taking place outside of the UK?	☐	☑
20. Does your research fit into any of the following security sensitive categories: • commissioned by the military • commissioned under an EU security call • involve the acquisition of security clearances • concerns terrorist or extreme groups If Yes, please complete a Security Sensitive Information Form	☐	☑

I understand that if granted, this approval will apply to the current project protocol and timeframe stated. If there are any changes I will be required to review the ethical consideration(s) and this will include completion of a 'Request for Amendment' form.

☑ have attached a Risk Assessment
☑ have attached an Insurance Checklist

If the applicant has answered **YES** to **ANY** of the questions 5a – 17 then they must complete the MMU Application for Ethical Approval.

Signature of Applicant: Vincenzo Scialpi-Sullivan Digitally signed by Vincenzo Scialpi-Sullivan Date: 2018.01.20 09:01:18 Z Date: 20/01/18 (DD/MM/YY)

Independent Approval *for the above project is (please check the appropriate box):*

Granted

☑ I confirm that there are no ethical issues requiring further consideration and the project can commence.

Not Granted

☐ I confirm that there are ethical issues requiring further consideration and will refer the project protocol to the Faculty Research Group Officer.

Signature: Stephen Gordon Digitally signed by Stephen Gordon Date: 2017-11-15 12:39:12 Date: 15/11/2017 (DD/MM/YY)

Print Name: Stephen Gordon Position: Senior Lecturer

Approver: Independent Scrutiniser for UG and PG Taught/ PGRs RD1 Scrutiniser/ Faculty Head of Ethics for staff.

F. Presentation Slides

Content Outline

- Project's Purpose
- Aim
- Objectives
- Related Work
- Product's Design, Interface and AI
- Product Implementation
- Testing
- Evaluation Findings
- Conclusion & Future work

What is the purpose of the Project

What additions could there be in a chess environment to aid beginner chess players in improving proficiency in their chess game

My aim was to implement an in-game system which would help the user to make superior chess moves

Why?

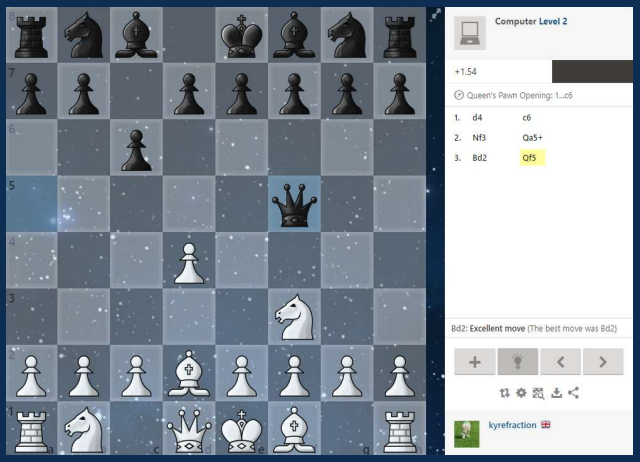
- Current popular chess solutions don't provide any useful in-game support
- Provides a means of Experimental Learning instead of academic
- Experimental Learning cements knowledge through concrete experience

General Objectives

- User Friendly HCI
- Support System
- Varying AI Opponent Difficulties
- Program Must Hold User's Attention

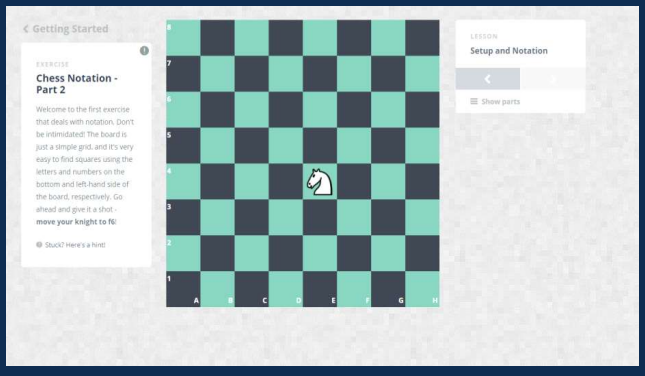
Effective in-game support

Poor in-game support



Existing Solution to Aiding beginner players

- Chesscademy provides beginner chess aid through academic learning



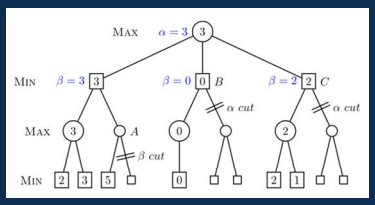
My Derived Solution Given the Existing Work

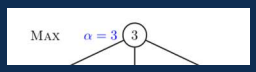
- Take inspiration from popular strategy games with effective in-game user feedback and User Interface Design
- Implement a similar User Feedback system
- Implement a Chess AI with scalable difficulty to provide appropriate challenge for all chess players

Design

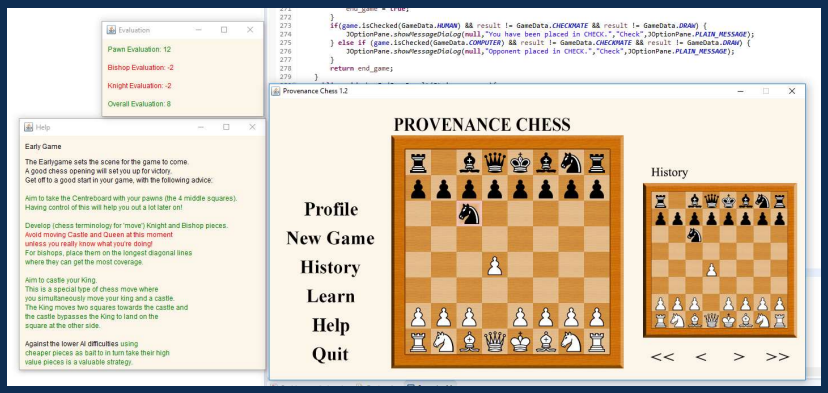
- AI should use 'Alpha-beta' pruning, which is an improved minimax searching algorithm
- Difficulty level will be determined by the Depth of the search algorithm
- The game should give the user feedback in a method similar to Civilization V

Differing AI Difficulty Example

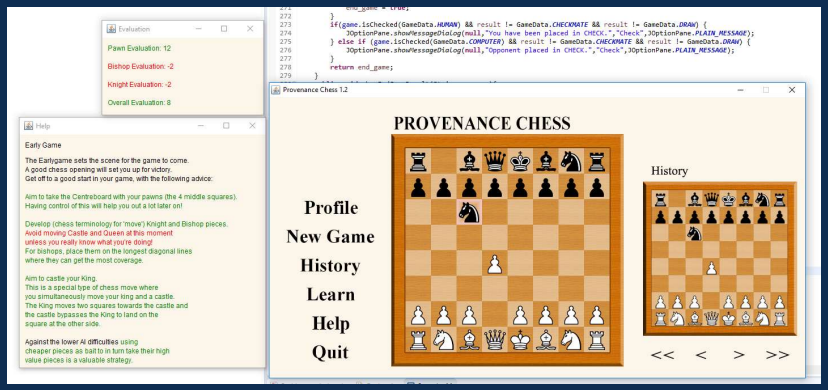




Implementation



Implementation

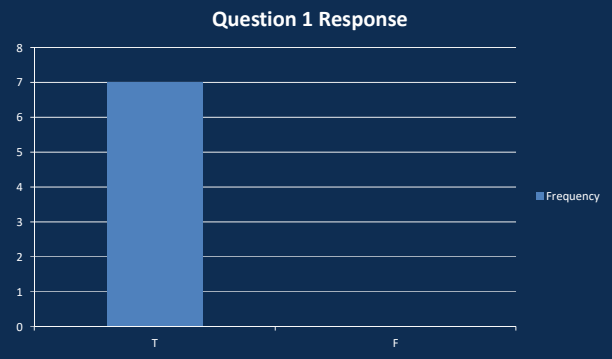


Testing

- In testing the program passed everything apart from certain pieces showing the incorrect move highlighting
- Setting the AI Opponent to Maximum difficulty causes high CPU usage

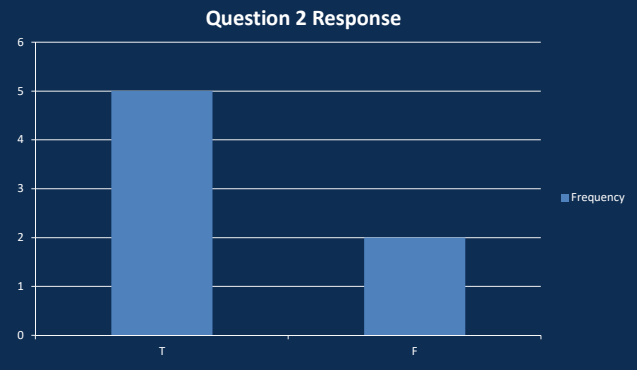
Question 1

- I found all necessary information in the place I expected it to be



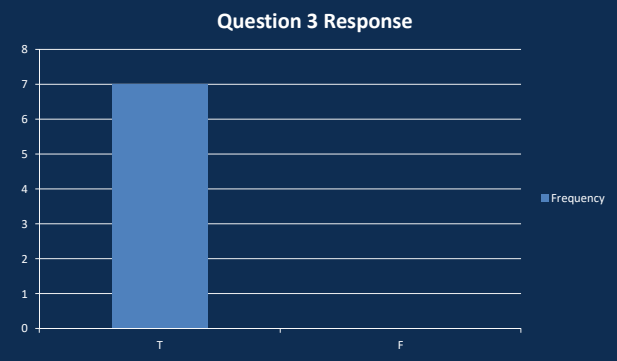
Question 2

- I found the difficulty of the AI matched the difficulty I expected to play against

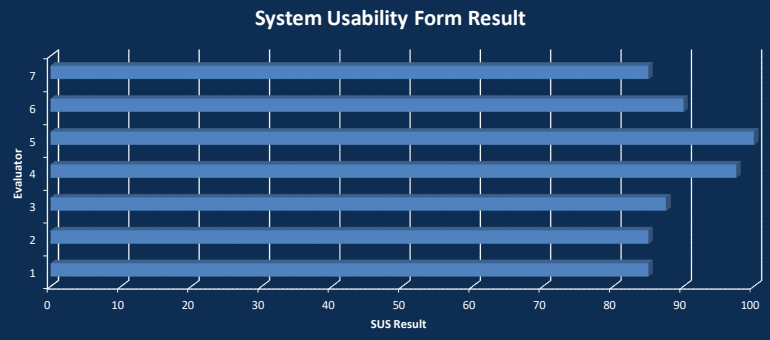


Question 3

- I found the in-game assistant to be helpful



System Usability Results



Future Work

- JavaFX recreation of the User Interface
- Syzygy Tablebases for endgame

Conclusions & Implications

- The in-game support was effective in providing support
- The AI opponent was effective, yet at lower levels was deemed as too aggressive for a beginner player to handle

Demonstration

Questions