OCR GCE A

COMPUTER SCIENCE PROJECT

H446-03

Name : Alex Dyer

Candidate Number: 8572

Tuxford Academy: 28148

Title of Project: KnightOwl Chess Bot

H446-03 – Project CONTENTS

Table of Contents

[A. Analysis 4](#_Toc183005125)

[Project Identification 4](#_Toc183005126)

[Stakeholders 4](#_Toc183005127)

[Why is this Suited to a Computational Solution? 4](#_Toc183005128)

[Stakeholder Statistics 6](#_Toc183005129)

[My Microsoft Forms Questions 6](#_Toc183005130)

[Final Stakeholder Responses 7](#_Toc183005131)

[Research 10](#_Toc183005132)

[Existing Similar Solutions 10](#_Toc183005133)

[Features of My Proposed Solution 14](#_Toc183005134)

[Requirements 14](#_Toc183005135)

[Software and Hardware Requirements 14](#_Toc183005136)

[Stakeholder Requirements 15](#_Toc183005137)

[Success Criteria 16](#_Toc183005138)

[B. Design 17](#_Toc183005139)

[Systems diagram 17](#_Toc183005140)

[Intro Screen 17](#_Toc183005141)

[Chess Games 18](#_Toc183005142)

[Analysis/Review Screen 18](#_Toc183005143)

[Problem Decomposition – Two Player Chess Game 18](#_Toc183005144)

[All Algorithms for Main.py 20](#_Toc183005145)

[All Algorithms for Moves.py 21](#_Toc183005146)

[All Algorithms for Graphics.py 29](#_Toc183005147)

[Pseudocode for the Project 32](#_Toc183005148)

[Variables 33](#_Toc183005149)

[Developing the coded solution 34](#_Toc183005150)

[D. Evaluation 38](#_Toc183005151)

[Project Appendixes 39](#_Toc183005152)

[References 39](#_Toc183005153)

# A. Analysis

## Project Identification

There have been many different games made over the millennia. Some are older than others, some are newer, and some can be played on a screen with a controller or a keyboard. But one game that has withstood the test of time is Chess. Chess is a strategy game where each player controls sixteen pieces – a king, a queen, two rooks, two knights, two bishops, and eight pawns. They must manoeuvre these pieces around an 8x8 chequered board to checkmate their opponent’s King. This game has been played and studied obsessively for over 1,500 years.

More recently, AI has advanced so that many bots have been programmed to play chess to a higher level than any human. Working off the Elo scale, the greatest player of all time, Magnus Carlsen, has an approximate Elo of 2,900. This is the highest human chess Elo in history. However, chess computers such as Stockfish, Torch, and AlphaZero have been made to beat Magnus Carlsen and the rest of the world’s approximately 1,800 GMs. But one thing that is a lot rarer is a chess bot aimed at being beatable by people of around 1000-1300 Elo. People always go on about these absurdly good chess players and chess bots. How about one that plays similarly to a human at an intermediate to advanced level of chess? This is the absence that I would like to remedy with my project.

I will use Python and one of Python’s graphics libraries Pygame to create KnightOwl. The features of a computer that would be required to use this would be the use of a keyboard and mouse as Python does not work with touchscreens.

### Stakeholders

The clients and general demographics of this bot are chess players from complete beginners to about 1,600 Elo. Because over 70% of chess players fall within this range, the stakeholders will be a representative sample, ranging from people who have just begun playing chess to people who are very comfortable with the game and its various strategies.

Stakeholders for the chess bot represent casual players who play chess occasionally to take a break from work/school to experienced people who want a new challenge or an easy game.

The game, for the most part, will be playable with just a mouse, but there will be a function that takes chess notation as an input too. It will come with an analysis feature that breaks a game down into single moves and explains the pros and cons of each move.

### Why is this Suited to a Computational Solution?

This project lends itself to a computational solution in many ways. The solution will be a chess computer that isn’t so hard to beat that it demoralises new players and angers gifted players, but one that anyone can play against and have a decent chance at winning against.

#### Problem Recognition

The main problem with this is creating a chess bot that plays like a human, as most chess bots have a completely different play style from any human because they’re programmed to make a good move or a worse move randomly. The chances of a good move compared to a bad move differ based on the not’s supposed Elo. My problem is that to make this work, I have to program a specific style into the computer and get it to stick to it. Most human chess players play with their unique style and preferences, and that’s why no two chess games are the same. The challenge is to create a bot that works in the same way.

#### Decomposition

This issue can be broken down into x smaller steps:

1. Work out the graphics for the program (e.g. the board, pieces, movement animation, etc)
2. Code the game itself
3. Build a basic bot
4. Improve the bot
5. Work out the finer details, such as a playstyle and aggressive/defensive tactical/positional preferences

When these steps are complete, the program doesn’t lag, and the bot can beat me (13-1400 Elo) about half the time, it’s done. With machine learning and reinforcement learning, I should be able to get the bot to play against itself a sufficient amount of times so that it can train itself to get to a high enough level, putting its findings into an SQL database. I can hard-code any chess theory needed into the database, to make it seem more like a human.

#### Divide and Conquer

Although these steps seem challenging, they are perfectly feasible. Solving each of these independently and combining them into a complete, modular program uses the divide-and-conquer way of problem-solving.

## Stakeholder Statistics

### My Microsoft Forms Questions

I will summarise the main questions that I have asked my stakeholders about their general chess-playing, and whether they’d be interested in my proposed project.

#### Current Chess Info for each Stakeholder

Questions 1-3 summarise how often the stakeholder plays chess and how good they are at it

1. Do they play chess?
2. Current chess Elo
3. How often do they play chess?

#### Views on Current Chess Bots

Questions 4-6 are on the stakeholder’s typical opponents and their views on the existing bots

1. How often do they play against bots?
2. Would they consider playing a new chess bot?

#### Would they Play my Chess Bot?

Questions 7-9 are views regarding my chess bot

1. Would they be interested in seeing a chess bot that plays like a human would up to an Elo of 1600?
2. What name would be suggested for the chess bot?

### Final Stakeholder Responses

A white background with black text

Description automatically generated

A graph with different colored bars

Description automatically generated

A white background with black text

Description automatically generated

The first three questions show what quantity of stakeholders play chess and their strengths/regularity with chess. The statistics show that most of the stakeholders play chess and, of the people who do play chess, the majority are between 400 and 1100 Elo. This would mean that a 1600 Elo bot would be a difficult, but reachable challenge for them. Also, a lot of them either play rarely or weekly, meaning that, hopefully, my bot would get the people who play rarely to play more often and therefore increase the popularity of chess.

A white background with black text

Description automatically generated

A white background with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

According to the statistics from the survey, most people usually or only play other people, however, they would like to see and, for the most part, would definitely or possibly consider playing a bot at around 1600 Elo. As shown above, all stakeholders asked would like to see a bot that plays like a human to an Elo of 1600. Therefore, I think that the chess bot will be popular among chess players of all skill levels.

A screenshot of a phone

Description automatically generated

I will decide on a name nearer the time

## Research

### Existing Similar Solutions

#### chess.com

##### Overview

A screenshot of a game

Description automatically generated

Chess.com contains over fifty chess bots of various skill levels, ranging from absolute beginner levels to being better than any Grandmaster. This provides a wide range of fun bots who each go for their own various opening selection, tactics, fortes, and catchphrases. Complete with an easy-to-use user interface, social/friends database to play chess with friends, online capabilities, puzzles, and lessons, this is a user-friendly and fun way to play chess against anyone, regardless of whether or not you know them.

The user interface is immensely simple and easy to follow as the menu is always displayed and there are large buttons on-screen labelled with anything that can be done on the site. The home screen contains the menu, a suggestion of a player on your friends list to play against, puzzles and a game review suggestion, as shown below:

A screenshot of a game

Description automatically generated

A screenshot of a game

Description automatically generated

The ‘Game Review’ function allows a player to get the chess engine to analyse one game a day on the free version, or unlimited depending on what subscription plan they have. It contains game accuracy, the number of different types of moves made, the best moves for each position, why they were the best moves and an Elo prediction.

A screenshot of a game

Description automatically generated

A screenshot of a game

Description automatically generated

Higher ratings mean better gameplay, as do higher accuracies. The Game Review feature is very easy to understand and very informative of good and bad moves and sums them up into two numerical values (Elo and accuracy)

##### What can I apply To My Solution?

In my solution, I would like to build an easy-to-follow user interface, similar to chess.com’s, as well as an analysis/game review function to show players where they can improve. The main difference between the bots on chess.com and the bot I plan to code is that the bots on chess.com play completely differently from humans, in so far as they are designed to have a randomised chance to play a move, changed based on the supposed rating of each bot. Mine is supposed to play more similarly to a human and play more human moves that, although they may not be the best moves of an engine, are more likely to feel natural to a beginner/intermediate chess player.

#### Lichess.org

A screenshot of a computer

Description automatically generated

This is another chess site that is one of chess.com’s main competitors. Although it isn’t as good as chess.com, it has a massive opening database, showing the most common and the best human moves for hundreds of assorted chess openings. It also has a FEN analysis function, meaning you can get the engine to analyse any given position. Similar to chess.com, it also has a game analysis function, however, this one isn’t as good as it only shows the bad moves and doesn’t rate the good moves made by the player. As shown in the chess.com diagram above, chess.com shows all moves ranging through brilliant, great, best…mistake, miss, blunder. Lichess only focuses on inaccuracies, mistakes, and blunders. It is a very good, albeit immensely critical chess engine.

Another feature that it has in common with chess.com is the fact that the player can also play the computer. However, the computers here aren’t as immersive as the ones on chess.com, as they don’t possess the same skill range, catchphrases, or range of bots.

##### What Can I Apply to My Solution?

The only feature that I would probably use for my chess bot would be the opening database because it would help me differentiate ‘Book’ moves and theory from any middlegame/endgame moves. It would also vastly help see what rating the user plays like, based on how common the opening is and whether they do something classic for many people of a low Elo, which is to attempt to invent and utilise their openings, instead of learning any theory.

### Features of My Proposed Solution

#### My Initial Solution Proposition

My solution will be a chess bot that plays like a 1,600 Elo human because, although both lichess and chess.com both contain their own bots, neither of them contains bots that play like a human. In this case, I think that a combination of the lichess opening database, put into my own SQL database, and a similar range to the chess.com bots, in that, I could attempt to do different difficulties, ie: 400, 700, 1000, 1300, 1600 Elos. These bots will have a visual move-log and game review/self-analysis functions similar to the ones offered by chess.com, however, unlike chess.com, there is unlimited access to these features for free. There will also be a function to save played games so that they can be looked at again later down the line for improvements and future analysis.

#### Potential Limitations

The main limitation of my solution is that, although it incorporates a lot of functions based around the bot, it won’t include any online functionality, puzzles, or lessons, as the main point of this solution is to code the bot(s).

Another is that, because I’m using Pygame instead of a more advanced GUI, the UI is likely to be fairly straightforward. It will include some buttons for which bot you want to play against, and a list of recent saved games, with analysis if analysis is requested for the game in question.

## Requirements

### Software and Hardware Requirements

#### Hardware Requirements

**A computer capable of running an up-to-date version of Pygame** –The chess bot will need to be run on a computer capable of running Python 3.10, MySQL and Pygame. A laptop with a processor capable of running at 30+ fps is recommended, but not required

**A mouse and keyboard** – Pygame isn’t compatible with touchscreens, so mobile devices/iPads won’t be compatible with the software

**6GB RAM** – 4GB is the minimum RAM requirement to run Pygame, and 2GB is my current estimate of how much RAM will be required to access the MySQL database. Anything below this could potentially cause the chess bot to crash

**3GB HDD/SSD space** – Pygame will need 3GB of HDD or SSD space to run, MySQL needs another 2GB and the image files need about 100KB, so at least 7-8GB will be needed to record

#### Software Requirements

**Windows, Mac, or Linux OS** – These are the Operating Systems that support Python and SQL

**480p** **Resolution** – This will be the minimum monitor resolution required to play the chess game as it needs to be a resolution where the user does not need to strain their eyes to make out what pieces are what, so they don’t damage their eyes

**Python Interpreter** – The code will be written in Python, with MySQL and Pygame libraries

**MySQL for Python** – The opening database will require an SQL database, and therefore a Python plugin for SQL

**Pygame for Python** – The GUI will require a Python GUI library to run the chess bot graphics

### Stakeholder Requirements

#### Design

|  |  |
| --- | --- |
| Requirement | Explanation |
| Basic user interface with the ability to change the difficulties of the bots | So the user can adjust and access the features of the program easily |
| Easy-to-understand, informative analysis programs for any game played | Allows the user to rate their performance on each game based on the engine’s analysis of their game, and their analysis |
| An instructions page | Shows the user any key binds associated with the program, how they are used and how to navigate the program if they don’t understand it or aren’t good with technology in general |

#### Functionality

|  |  |
| --- | --- |
| Requirement | Explanation |
| Use of mouse to control moves | The mouse’s position is recorded by Pygame and is used to control which piece was clicked and where it was moved to |
| Use of keyboard to undo moves and input chess notation | The backspace key is going to be the keybind for undoing moves, and the main keyboard will be used to input chess notation as another way to control piece movement |
| Instructions on how to edit the theme of the board and the style of the pieces | If the user dislikes the look of the chess board and pieces, there will be instructions on how to edit the themes within the game |

## Success Criteria

|  |  |
| --- | --- |
| Criterion | How to show it’s been met |
| The main window showing a working chess engine | Screenshot of working graphics/video of a working engine |
| The AI makes non-randomised moves | Run a game through Stockfish to show a decent amount of logic |
| Simple, navigable design | Screenshot of the GUI, with a menu, chess engine and instructions |
| A working Exit button | The Exit button (namely the big X in the top right) is clicked and shows that it closes the window |
| Instructions | A screenshot of a page that’s accessible through the main menu, walking the user through the keybinds and any other features that they need to know and that points them to a site where they can learn to play chess if they don’t already know |
| The chess AI plays at the desired level (1,600 Elo) | The AI can beat me over 50% of the time that it plays me, as shown through the PGN files of the games |

# B. Design

## Systems diagram

There will be many elements to my project. The main one is the chess engine that will be originally made for two players on the same device. The following design is for that and for the prospective main menu, and other elements will be shown later depending on my timescales.

### Intro Screen

The Main Menu screen will be fairly simple, with options to play the bot, play two-player, enter settings or access the openings database. The option to play KnightOwl will take the user straight to a game and will randomly assign a side (white or black) to each player.

Settings, as shown in the diagram above, will allow the user to alter the designs and colour schemes of the board and pieces so that they can use their preferred look for the board. These preferences will be saved to a text file, so they don’t need to be configured every time the game is loaded. It will also allow the user to adjust the difficulty level (Elo) of KnightOwl so they can play at a level that challenges them.

Lastly, the openings database will contain a board and FEN input bar, so that the most common opening moves can be found in a position (if no games have been played in a position, it will say so). This information will be pulled from lichess.org’s API whenever a new position is given to the system. The API will show what win/draw/loss rates, the average ratings of the players who use that specific move and the most common moves afterwards.

### Chess Games

The chess games, although the basic GUI makes them look pretty straightforward, are in fact, extremely complex. These include a series of interlinked algorithms which narrow down the moves available by considering checks, pins and of course the general movement rules of the game, algorithms to search for pins and checks and a constantly changing 2-dimensional board array.

After seeing the difficulty ranges of the bots available on chess.com, I’ve decided to make KnightOwl a bot playable by anyone and everyone by attempting to include difficulty levels up to (hopefully) 3,000 Elo, however I don’t know how manageable that will be. The pseudocode for the game as a two-player game and with the computer will be shown below separately.

### Analysis/Review Screen

These screens will be available after any game played against a bot or 2-player on the same device. This will give a rundown of a player’s performance in a game by showing an evaluation bar of who was winning and by how much after each move, as well as a system used by chess.com to class moves based on how good they were (Best, Excellent, Good, Inaccuracy, Mistake, Blunder) with Brilliant and Great moves being shown as well as Misses and Book (Opening Theory) moves. Essentially, I want to make this similar to the one offered by chess.com without having to pay an extortionate amount of money to get more than one a day. It will also give an estimate of a player’s rating based on the single game, as well as a rating based on which bots they beat and how consistently.

## Problem Decomposition – Two Player Chess Game

There are various sub-problems that my chess bot needs to be broken down into. Each of these will require its own separate algorithm (flow charts and pseudocode later). I’ll start with the chess game itself, separately to the bot (minus the analysis and review screen for now too).

The problems that I will be breaking down are as follows:

1. Adding the board and pieces to the screen – this will be done using the blit operation in Pygame to add 32 darker squares to the screen over a white background, then I can download and once again use blit to add

these to my screen. This is fairly self-explanatory: if I don’t do this, then the user will have no usable interface with which to interact and play chess on

1. Moving the Pieces Randomly – I’ll use a two-dimensional array called ‘board’ to store the current positions of all the pieces on the board currently. When a piece is moved, the array can be altered appropriately to portray the new layout. Without this ability, I can’t start to code and test move generation for the different pieces. This is immensely important as the users need to be able to see the current board state to make it easier to visualise follow-up moves and to calculate
2. Move Generation – If the pieces could move anywhere randomly, the King would be taken on the first move and it would be game over. This makes the game playable (mostly – en passant, pawn promotion and castling aren’t added yet) and uses a Boolean flag to show whose turn it is. This will be done using Object Oriented Programming in a separate file to create all the different algorithms and functions needed to find all the possible moves and narrow them down to the valid moves – the flow charts later lay out how I plan to cover all the necessary functions for this.
3. Check, checkmate and stalemate – These are some of the most important rules of chess. It will determine whether the King is being attacked and whether any pieces are pinned. This will be done through more functions within the moves.py file and a long algorithm to remove any ‘illegal’ moves, including linked algorithms to search for pins and checks.
4. Pawn Promotion – This is one of three more complex rules in chess. This will check if a pawn is on the ‘back rank’ and if it is will promote it to a Queen, Rook, Knight or Bishop, depending on the user’s choice.

I can do this using some graphics to create a square on the board containing a rook, queen, knight and bishop where the user clicks one of them and the pawn promotes to one or other of them

1. En Passant – The second more complex rule in chess. If one pawn moves two squares forward and is horizontally adjacent to an enemy pawn, that enemy pawn can take it as if it had only moved one square. I can make a subroutine that is called from the pawn move function that checks if a pawn moved two squares on the previous go and if so, add any moves for pawns that could play en passant
2. Castling – The third more complex rule in chess. If there are no pieces between the King and Rook and neither has moved, the king can move two spaces to the side and the rook stands on the adjacent square, closer to the centre. This will need to be a separate function in the moves class which detects if the king and/or rooks have moved

### All Algorithms for Main.py

Pseudocode can be found in the Pseudocode Folder within write-up

#### Main Loop

[main.py flow chart view link](https://miro.com/welcomeonboard/YlFpaEFWZ1lrY3oxM3dtc2cwTEVRMHJ1VnJ1c1BnYmJmd2JMREdCUWhRSUY5N2tkd25EOFVtSFB0RlF0bUdHd3wzNDU4NzY0NTk3NzA1ODM5NDE2fDI=?share_link_id=241004072459)

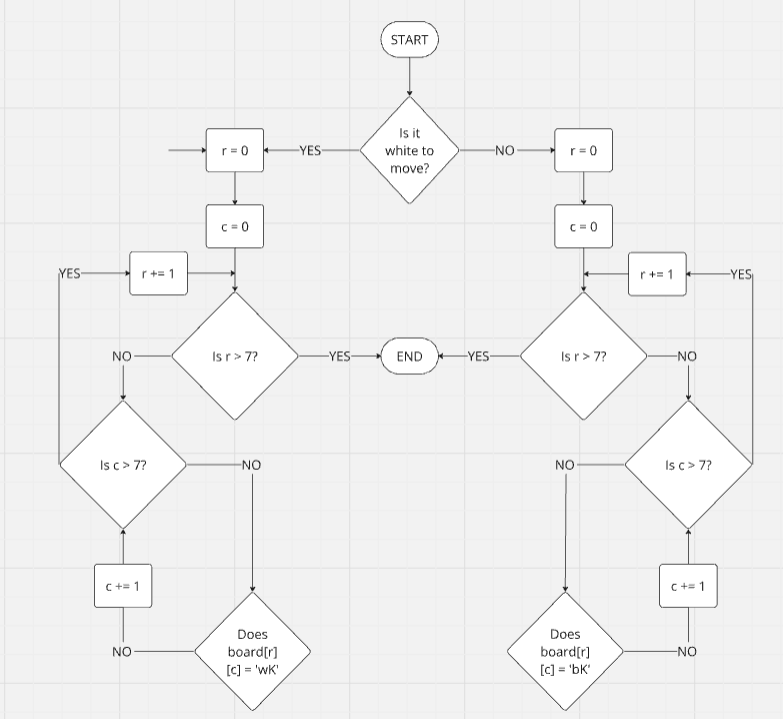
A diagram of a flowchart

Description automatically generated

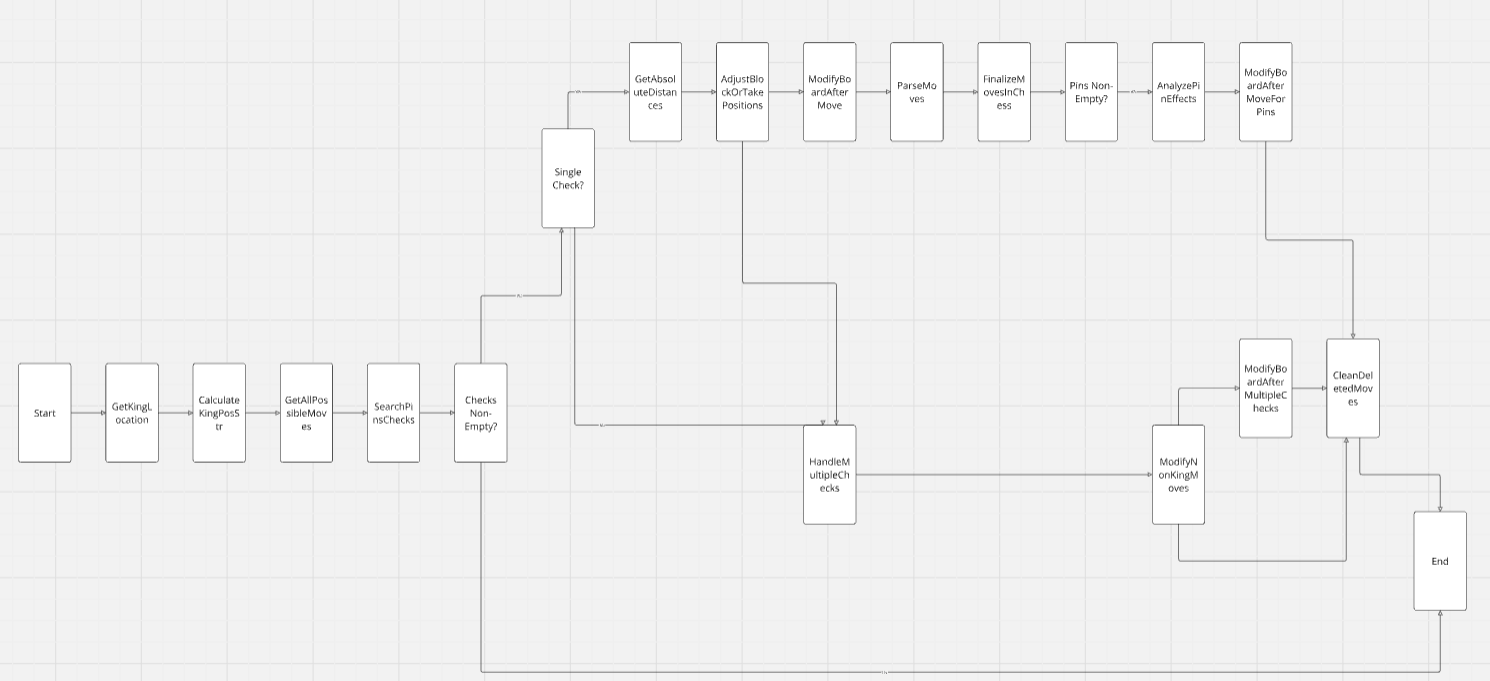
### All Algorithms for Moves.py

Pseudocode can be found in the Pseudocode Folder within write-up

#### Finding the King’s Location



#### Getting All Valid Moves



#### Getting All Possible Moves (Excluding Checks and Pins)

A diagram of a flowchart

Description automatically generated

#### Checking If Castling Is Possible

A diagram of a black king

Description automatically generated

#### All Algorithms for PIece Movement

<https://miro.com/app/board/uXjVKi6uJQE=/>

#### Algorithm for Searching for Pins and Checks

A diagram of a check

Description automatically generated

#### Algorithm for Knight CHecks

A diagram of a algorithm

Description automatically generated

#### Algorithm for Pawn Checks

A screenshot of a computer screen

Description automatically generated

#### Checking for Rook/Queen Checks

Queen checks in this flow chart are horizontal or vertical only

<https://miro.com/app/board/uXjVKidCQyE=/>

A diagram of a flowchart

Description automatically generated

#### Algorithm for Bishop/Queen Checks

Queen checks in this algorithm are diagonal only

A white square with black text

Description automatically generated

### All Algorithms for Graphics.py­­­­­­­­

Pseudocode can be found in the Pseudocode Folder within write-up

#### Drawing the 8x8 Board

A diagram of a flowchart

Description automatically generated

#### Move HIghlighting

A diagram of a flowchart

Description automatically generated

#### Creating the Pawn Promotion Options Square

A diagram of a flowchart

Description automatically generated

#### Drawing the Pieces

A diagram of a flowchart

Description automatically generated

## Pseudocode for the Project

See flowcharts above for the full breakdown of what each function does

Also, see the Pseudocode folder in the zip file for the actual pseudocode

## Variables

Here is a list of the key variables and what they are used for:

board – 2D array holding the current position on the board. I used a 2D array here because it allows me to use the first dimension for the columns and the second for rows

moves – Holds all the valid moves for that turn in the same format as in squares to make it easy to see if they match

squares – Holds the coordinates of the squares clicked. If the entire array doesn’t match up to a value in moves, it gets emptied

clicks – Works closely with the squares variable. This holds the number of clicks the player has done. If it reaches two, that’s how the computer knows to check the squares variable

w/bKingLocation – Holds the location of each King on the board to help search for checks and pins

# Developing the coded solution

A green rectangular object with a white border

Description automatically generated

This was my first step. I created the Main class to hold the main sequencing for the program and to create the order for the events to happen in. I decided to take an Object-Oriented approach to the program so that there is a definite structure throughout, I can access my various modules quickly, and I can access variables in one file from another easily.

A green and white square with dots

Description automatically generated

The next step was to start creating some of the main variables throughout the program. I called this file ‘Vars’, though it would later be changed to ‘chess\_engine’. I made the self.board

variable as a 2D list so that it can be altered after each move, and so that it has a row/column structure to it to make it easy to access each square on the board. self.whiteToMove determines whose move it is. If it is true, it is white’s turn to move, otherwise it is black’s.

main.py

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

vars.py



I made the new vars file, after moving the previous contents to chess\_engine.py. Although you can’t make a constant in Python, I moved the general constants to a separate file where I can manually change them.

In the main.py file, I made the screen and a dictionary containing the image files (PNGs) for each piece. I then filled the screen in white and made a draw\_squares method to draw the board. This would later be moved to graphics.py and the colour of the dark squares would be cycled through by pressing the left control key.

main.py

A screenshot of a computer program

Description automatically generated

chess\_engine.py (formerly vars.py)

A screenshot of a computer program

Description automatically generated

I had a bug with the board array (as seen above) where I made it a 1 dimensional list by accident, so I changed that to 2 dimensional here.

I also coded the algorithm to draw the pieces on the screen (they are not currently moveable)

main.py

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

I added piece movement by assigning the mouse position tuple to a variable and using the coordinates to work out what, if anything, was clicked. Later, I’ll extend this to check if the desired move is a legal move (involving castling, en passant and pawn promotion)

moves.py

A computer screen shot of a program code

Description automatically generated

A screenshot of a computer program

Description automatically generated

I created a number of dictionaries to convert row-column notation to rank-file notation, as well as creating an algorithm to collect all the possible moves (discounting checks and pins).

# D. Evaluation

<See H446-03 Project Advice Booklet for help and guidance of what must go here.>

# Project Appendixes

Insert as many project appendixes as you need for your project.

These might include, but are not limited to:

* Complete Code Listing (ESSENTIAL)
* Interview Transcripts
* Meeting notes
* Observation notes or questionnaires

# References