**Background pattern

Description automatically generated**

**A chess project**

**Puzzle Chess**

peter mcguinness

**Table of contents**

Table of Contents

**1.Introduction 2**

1.1 Problem Definition2

1.2 Overview 3

1.3 Aims 3

**2.Research4**

2.1 Initial Research5

2.2 Python and Pygame5

3. Requirements for the system3

3.1 Requirements for the system3

3.2 Key Requirements3

3.3 Additional Requirements3

4.System Architecture and Design3

4.1 Using Pygame3

**4.2 How the System Works**3

5. UML Diagrams and Implementation3

5.1 System Functions Flowchart3

**5.2 Mouse Movement Diagrams**3

6.Testing and Evaluation3

6.1 Developing code using an iterative cycle3

7.Disscussion3

8.Conclusion 3

9.Appendix3

10.Bibliography 3

1. **Introduction**

In this paper I will discuss the development cycle of the game puzzle chess. As an student of Information technology, I have always had interested in the development of software and technologies. This project is a step to what will become my career and my life. I decided to use chess as my project as I thought it would be a challenge to my programming skills. The game is design to be a simple puzzle displayed to the user using the pygame library to display the board to the user’s screen. There will be two files, the main.py file and the logic.py file. These files allow the game to function as it does. The main file imports all the necessary parts such as the piece images, sounds, makes some basis checks and initiates some functions and the pygame library and starts the game running. The logic file is where most of the engine programming happens. It is not a full chess game engine. This logic file checks some move legality. This will be explained in more detail. After finishing a puzzle, the user can then review the puzzle on an online platform in a form which was written in html and css.

The outcome of this project will be chess game with a puzzle displayed to the user. The user can that rate the puzzle online. I hope that you will find the following paper informational to the extend that is intended.

* 1. **Problem definition**

The objective for this project was to create simple computerized chess game which would allow the user to play a chess puzzle, playable either online or on the system is it being used on. Chess is a strategy game which has been played for centuries. That is one of the reasons for taking on this challenge. This was to allow myself to understand this game more clearly. The goal for this project was to create this feeling of a chess game in a computerized manner. This project will be programmed in python using the Pygame framework along with implementing a function file for the logic programming also programmed in python. The aim for this game was to allow the user to get a checkmate in the current puzzle and move on to the next puzzle after the user confirming what they would like to do. After completing a puzzle, while playing the online version the user would be offered to rate the puzzle. This aim and intention were developed and altered as the development life cycle went on to accommodate the requirements of the software.

* 1. **Overview**

This project has two main components in python. The main.py file which is where the game is initially loaded and initialized. This is where the while loop run as is necessary in pygame along with the variables for the screen size. The main function which is where the while loop lands and other functions such as def loadImages(), def convertFenToImg(), def drawGamestate, def drawhighlight, def drawboard and def draw pieces. These functions are the building blocks of the chess game. For the imports, we import pygame to allow the python script to read the pygame functions and other necessary associated values. We import logic file as this is the pygame logic I created which allows the connection between the main.py and locgic.py files. And we also must import the asyncio library as it will allow the program to be used on a website. This is also the reason for the main file being named async def main() rather than just main. The logic.py file is where the brunt of the logic is commanded. There are two classes. The gamestate class which contains the if, elif and else statements surrounding movements on the board and the move class which is a blueprint for the functions using move as a parameter.

* 1. **Aims**

My aim was to create a fun easy to understand chess puzzle game using python and pygame as a framework for building this game. I want to create this game, so it is usable as a tool for learning chess and having fun in free time. The game was meant to be a simple pygame built web application. It turned out to be far more complex than I initially though. As I had to program the chess logic file, this was far more of a task than my initial expectation. I was informed by my supervisor to program the visual aspects of the game initially, so displaying the pieces to the board and drawing the board to the create. This was not a difficult task, I had this completed and implemented in December of 2022. This was just a simple main file and the logic file. Moves and logic was to be implemented later. Programming the movement in pygame would prove to be the most difficult take of this project. The logic file was by far the most difficult and most time-consuming task I had to undertake.

The graphics will be implemented by the pygame library. Pygame is a library produced to the creation of video games and visual data representations. To give a better overview of this programming I will display a UML class diagram and an activity diagram.

1. **Research**

**2.1 Initial Research**

For the research area of this project, I researched online chess games such as lichess.org and chess.com to understand the look and feel of what a chess game is supposed to feel like. Lichess.org is where I learned the rules and regulations of chess. It developed my understand and allowed me to program this project with more of an understand of what I was doing.

I found what Lichess and chess.com lack is the fast paces chess puzzle aspect theory which my game would hopefully bring to light. My game is a simple lightweight puzzle game. I believe this is something that isn’t being done to in this manner. I think of games more like wordle or solitaire when I try to visualize this puzzle game.

**Lichess.org Puzzle Page (figure 2)**

The above figure is a screengrab of the Lichess puzzle webpage as can be seen. This is what in an ideal sense my chess project would end up looking like. The features around the sides are what I don’t want my project to end up like. The analyses aspects of chess were not what I intended to do. The puzzle on the Lichess website is pulled for a PGN database and works through it using the stockfish engine. Using the stockfish engine for move analysis would be a nice feature to include but would have been extremely time and cost consuming on the rest of the project. I just created the visual aspect on this project. What I like about what Lichess does is the movement highlighting and the random puzzle being used. These are features I incorporated into my project.



**Chess.com Puzzle Webpage (figure 2.1)**

The above is a screengrab from the popular chess website [www.chess.com](http://www.chess.com). It is very similar to Lichess in the puzzle page. But there is too much going on here. I understand there is a Zen mode, but this has to be checked to confirm this. I would just like to create a puzzle game. This gives you the option to analyse your game after and there are more features such as “Social”, “News” and the “login” and “Sign up”. The most valuable thing a person holds in their data and these websites push the idea of signing in and interacting. This is not what chess is about and what my project will be doing differently.

**2.2 Python and Pygame**

I researched pygame and other similar libraries. Pygame is by far the easiest library I have found to work with visuals in python and I have some previous experience with using the software. There is a lot of information on the internet surrounding pygame and python websites such as pygame.org, stackoverflow.com, w3schools.com and geeksforgeeks.com. These were used for trouble shooting and structuring.

For the website hosting I began with researching a platform called Hostinger. This platform allows the user to create as many html(hypertext markup language), CSS(cascading style sheets) and JavaScript files as they would like. This was what I had planned to use for hosting the python file. Although, I ran into complications while attempting to implement the python file. As I was using pygame within the python file. The software did not allow the pygame file to run as it requires specific system requirements. I had to find another method to run the pygame file without using Hostinger.

I came across a python file with the capacity to run pygame files on an online platform. Although this software was only in a very early build, I decided it would be appropriate for this project as this is not being developed on a professional scale. If I were to return to this project in the future, I would find a better method of hosting the file on a website but for now I found this method sufficient. The python file is called Pygbag. This is the reasoning for the main file being named async main as it allows pygbag to read this as the main.py file. Pygbag is a web assembly program written for producing pygame files on a website. The program does not always work. It can be very glitchy. This was an oversight on my end but seeing the requirements of what I needed to create I found it appropriate.

1. **System Requirements**

**3.1 Requirements of the system**

These are the requirements of the system to function as normal. These include key and additional requirements. Data requirements and constraints. Data requirements would be things such as the loadimages() function being able to find and use the images from the file in which they are located or being able to find the FEN.txt files to display the FENS to the board. If these requirements are not met this will cause the game not to function as normal and there will be an error given. Constraints exist on the systems side on this. What can the system handle. The use of if and elif statements in my code is a constraint as it tells the program while running if a condition is not met don’t run the code, run the else statement.

**3.2 Key Requirements**

Key requirements refer to the order of task that must be performed in this software to allow it to function as intended. Key requirements are the essentials of how the code will be structured. These requirements will dictate the behavioural and performance requirements of this system. Key requirements also allow the system to respond and react to the user inputs. As an example of this we can view getting a checkmate in the project. A checkmate happens in this game upon a white piece taking the black king. This calls on the mate function which reviews the board to see if the king (‘k’) is still on the board. If this condition is met, the mate function calls the FEN function if the requirements are met,

the FEN function allows a new FEN to be displayed. This program will let the user to get a checkmate in a puzzle and move on to a new puzzle straight away.

The requirements I was focused on were getting the game to a state where it could be run and understood in the simplest format and also the user is allowed to rate the puzzle based on there experience. The list of the key and additional requirements for this system are as follows.

|  |  |
| --- | --- |
| Key Requirements | Additional Requirements |
| Displaying the pieces according to the FEN in the database | Have a sound play upon a piece being moved on the board |
| Allowing the user to move the pieces in the puzzles according to a PieceMoves dictionary | Drawing a highlight square(s) after a piece has been selected to display where the piece can move to |
| Allowing the user to review the puzzle |  |
|  |  |

**Table of Requirements (figure 3.1)**

**3.2 Additional Requirements**

Additional requirements detail how the system performs, these are related to the stability of the system and the performance of the system. The requirements will display how efficient the system will be and the reliability of the system. For the system to function properly. It is required that we use the main.py and logic.py class. The pygbag module is an additional requirement, although it allows the program to be displayed on localhost:8000 it does impacts the quality and efficient of the pygame files. It allows the system to run on local host but does not do it well. This software is still in a very early stage of development. It lacks the quality and performance that the game requires. Often times while using the Pygbag software it will often just not allow the game to run. This is why I would prefer not to use it for this project, it is the only method I came across that allows a pygame file to be hosted on a website.

1. **System Architecture and Design**

**4.1 Using Pygame**

My reasoning for using python was I had found it to be the best in terms of logicality as python is used for calculations and algorithms and chess is just calculations of movement and algorithms to make checks on pieces. I found it appropriate. The use of pygame in this program was an oversight on my end. I found based on my initial research that python chess and chess using python is all over the internet in terms of prepared tutorials and demos. I found many resources on this topic. That is another reasoning for my using python. Pygame has proven to be very finicky the software needs to be developed more and be readable to the user when this are not working with it. I believe this is the main reasoning for the development being a slow process.

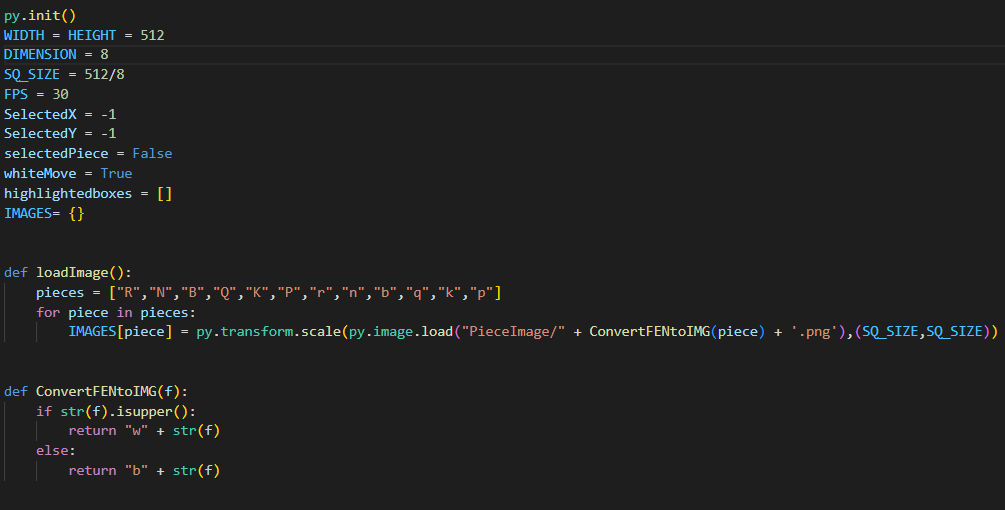
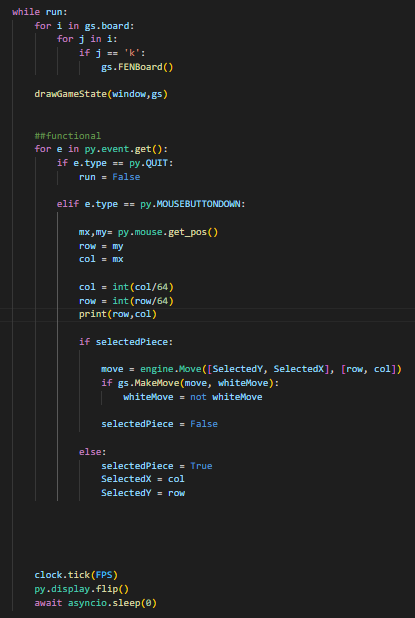
**SDLC(figure 4.1)[3]**

I followed the SDLC for this project. The SDLC is the software development lifecycle. It details the actions one must perform at each stage of the create of a piece of software and the order in which these are carried out. The program I have created is roughly 550 lines of code. I tried to optimize the efficient of this project to allow it to run as fast as possible. The pygbag program adds a large amount of code. It transfers the pygame file into web assembly format so that it can be interpreted by the webserver, which it is being hosted upon. In terms of project planning the first step in the SDLC, I researched lichess.org and chess.com to understand the general grasp of how this game is meant to function.

The program which I have created uses a built-in list of Forsyth-Edwards Notation(FEN). This is standard notation for displaying a single game state to the board. With the function built into the game, it holds a list of .txt files which correspond to a FEN in a folder in the “chessgame” folder. This is the method I used to program the game. Given a more professional capacity the game would be developed with a database with would hold the FEN files. Understandable it would be ideal if I were able to save a Portable Game Notation or PGN. This would allow the user to rewind a move and potentially move forward a move. This is not possible however currently.

**4.2 How the System Works**

The system uses python 3.11.2 to run with pygame as a library for the graphical interface. It also uses mixer from the pygame library this allows music and sound to function when these functions are called. Although these are not used to much throughout the program. This pygame file uses Pygbag which allows the system to respond with localhost:8000 and display the main.py file. The main.py coordinates the asyncio main which is what connects it to the localhost:8000. “Requirements are a declaration for an encouraged method of expression” [1](Karen Young, 2022). The system displays a screen which is 256x256 pixels. This fits perfectly with the pieces as the pieces at 64x64. This just makes the maths easier to understand.



**Initial Pygame Requirements (figure 4.1)**

These two sections of code (the main function and the py.init() ) are the piece of code which allow pygame to function. These are a requirement of this game and the game would not working without them. The py.init runs the init() function within the pygame library, checks if everything is working and allows the rest of the code to be run. “Draw grid draws the boundaries to the grid(so the black horizontal and vertical lines that separate the tiles) and draw grid and make grid is creating the 2d list **The Run loop (figure 4.2)** which are going to use to access all the nodes later ”

[2] (PasiduPerera, 2022).

The loop function(figure 1.2) is the code which will run at every frame of the game. The first four lines of code must be constantly running to ensure if there is a ‘k’ on the board and if there is it will move to the next puzzle this is what the gs.FENBoard() function does. The drawGameState function is what calls the rest of the game to load. This function calls the drawboard(), drawHighlight() and the drawpiece(). The for e in py.event.get() is just a requirement pygame statement to close the game if it detects that it is being closed. The mouse.get.pos is a critical line of code. It allows pygame to view the mouse position and the next part places them in variables divided them by 64 to make them each a single point of on the eight-by-eight board. It then passes the code through to the main class as can be seen in figure1.2’s second half. The else statement just sets the variables to a state which they can be reached if they need to be used. The logic.Move just calls the logic class in a different file in the same directory.

The logic file class is where most of the logical code is programmed. It holds two classes which respond to the calls of the main.py file. The class move responds with the previously mentioned code. It creates a blueprint for the rest of the functions in the file. The GamesState class holds the functions to check if the user is making a valid move or not, if the user is attempted an “En passant” or castle manoeuvre this is also read by this class. The class also holds functions for a mate check as mentioned in the main class, the FENBoard function and the getPieceAttributes() function for piece highlighting.

In the logic file, there two classes and four functions. At the beginning of the python file, a dictionary called piece Moves is placed. This is referenced later and is “double” dictionary for all of the piece movements in the game. The code is easy to understand. It just allows the files to view and see if the move attempting to be made is first of all a valid piece and in this dictionary. The dictionary holds between two and four moves depending on the piece. As an example of this. The pawn can only move forwards, it can move two squares forward initially and this is categorized as a special move, while it also holds the “Enpassant” attack which is a special attack that allows the pawn to move two squares forward and one square in the row direction. It takes the piece it has cross. I implemented this and castling as I thought these would be a good feature and not to difficult or time consuming on the processing power and performance of the game.

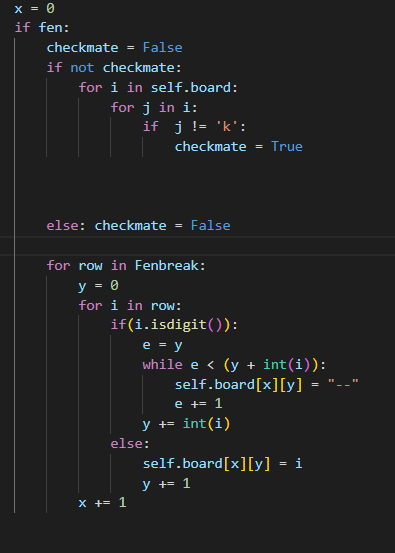
Text

Description automatically generated

**The FEN Board Function (figure 4.3)**

The FenBoard is subject to be changed slightly and improved if needs be. There are more than one methods to replace the board when a checkmate is achieved. Ideally I would like this to be on the click of a button to proceed to the next puzzle with another button to review the completed puzzle. On checkmate the FenBoard function is called. This function is another check to see if checkmate is achieved. I found this necessary for the game as checking checkmate twice was proven to work better under testing. (figure1.3)

As can be seen above the function checks for a checkmate this code will be followed in figure 1.4. The self.board denoted the board as it should start when the program starts. The “--” are all positions on the board which the pieces will be plotted to. The Fenlist is in a beta testing format in this image. It is essentially the database in which the FENs for the board are stored in a folder on the machine or webserver. In the following, it can be seen how the file is opened and read. The Fenbreak variable is the most important part of this code. It divides the FEN up into it’s respective lines.

 **The Fen Display Code(figure 4.4)**

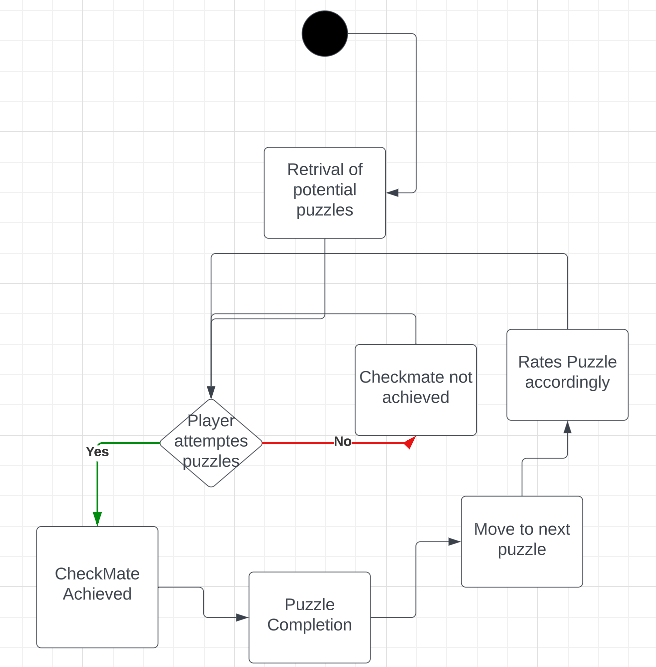
This section of code checks to see if a checkmate has been achieved. This will past back to the FENboard function to display a new FEN on checkmate. The second section of this code places the pieces on the board. The double for loop dictates for each row and then for each column. The if(i.isdigit) uncovers whether or not the character in the list is a digit or a letter. If it is a letter it displays it to the board and adds one value to the y variable to continue the iteration.

The embedded while loop checks that while e is less then y + i, it adds to the self.board and iterates plus one. This is the basic format of how the FEN works for the game and without this the code simply would not function the way it is intended.

1. **UML Diagrams and Implementation**

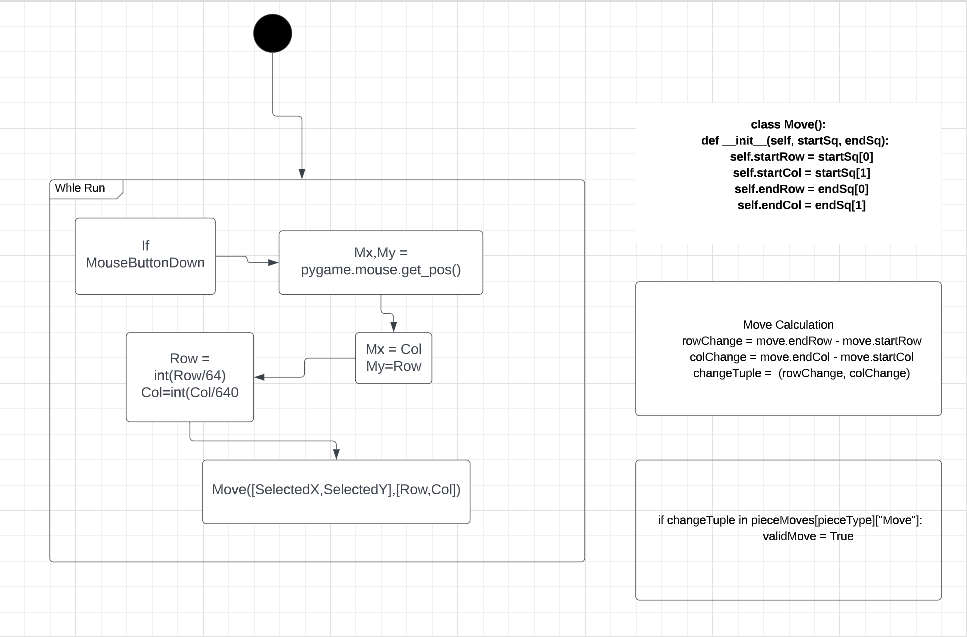
This section will detail the use of unified modelling language diagrams to aid the understanding of this project. I chose to do an activity diagram for the mouse movement tracking as I found it needed to be explained in a diagram. The movement of the mouse is the most important factor in this project. Without it the game would not be playable. The first diagram is just the basic requirements of the system I have designed in an easy-to-understand format.

But first, I would like to display a flow chart of how the system is meant to function in a simple format.

**5.1 System Functions FlowChart (figure 5.1)**

The above diagram displays a broken-down version of the system designed. The “Retrieval of potential puzzles would be searching the FEN list for an appropriate puzzle to display. After which the user will attempt the puzzle, on a checkmate being achieved the puzzle is complete and the user can then rate the puzzle, before moving on to the next puzzle. Note that the user does not have to rate the puzzle to play another puzzle. The user can play as many puzzles as they would like.

**5.2 Mouse Movement Diagram**

In this section the detail of how the system deals with the use of mouse clicks. This is important as it is the main component of the systems python files functionality. 

**Mouse Movement Code (figure 5.2)**

1. **Testing and Evaluation**

**6.1 Developing my Code Through an Iterative System**

This section will detail the testing and evaluation of this project. This was much of a trial, error, evaluate and retry. This is the section of the project which took the longest to figure out. I had been backing up my code once every two weeks, so if my laptop was to be broken or damaged or lost, I would still have my project. This idea proved to be useful when my laptop stopped working. My previous code can be seen below.

Text

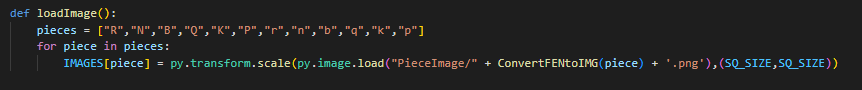
Description automatically generated

This was how I had originally written the code to import and set the pieces to scale. As can be seen and proven through testing. This is not the most efficient method of writing code. I could import and set the pieces to scale using a loop.

**First build piece displaying. (Figure 6.1)**

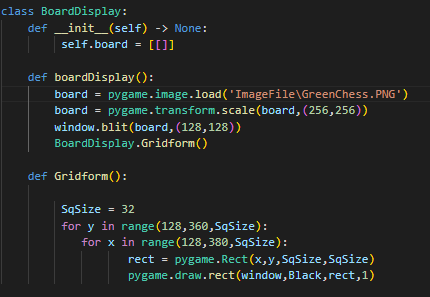
“wb” = white bishop

“.scale(wb,(length,heigth))”

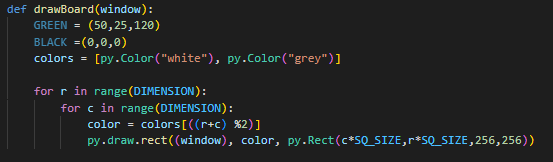


**Final build LoadImages Function (Figure6.2)**

The above shows the final build for the image loading. It creates an list called “pieces”, which stores the pieces according to the letters they appear. It then uses a for loop to add the pieces to list and sets them to scale. It is far more efficient than the previous. It also stores the images into a list and sets them to scale. There were a lot of bugs which were discovered in testing.

This is the original class for creating and displaying the board. This is an early mock-up of the class. When we compare it to the file currently being used to run the board it can be seen that this code is just to simple.

**Draft Build Board Display (Figure 6.3)**

****As can be seen the code is much more efficient. The building blocks are the same and it can be seen that the features will remain the same. The code in (left) figure is written better as it uses variables. So, at any time the “DIMENSION” variable is changed it will be updated for the rest of the code where it is used, which will make the code break less in theory.

**Final Board Display Function (Figure 6.4)**

1. **Discussion**

In this section I will discuss my thoughts, hopes and downfalls of this project. I would like to exercise the point that this is one of my favourite projects I have ever worked on. I enjoyed relearning and understanding python and pygame. This project further developed my understanding of programming, teamwork and how things don’t often work out the way as intended.

I thought that initially given more of an understanding of pygame, the production of this game would have been much more fluid and quicker so to speak. My comprehension of python wasn’t the strongest in the beginning as I had been learning and developing my programming knowledge using Java, html, css and js and hadn’t looked at any python in three years. This was a challenge, but I overcame it and now understand more python and pygame. Using pygame with this project is something I would not like to do again. Given the opportunity to do this project or a similar project again I would not use this method of execution. I would perhaps use Java or Unity2D (c#) as I think it would allow more functionality and there would be more resources online for this topic. These languages allow the creator more functionally and freedom. I would enjoy reusing this code on my own and creating a new game. A quick-fire lightweight puzzle game called “mate in one” similar to wordle where the user can play once a day. There would be a streak leader board, which would count the number of days the user has been playing the game and update according. Ideally, I would like to continue the development process of this game to allow it to grow. I would expand it so it could be used on mobile and tablet devices.

1. **Conclusion**

To conclude this project, I will first be discussing what I thought of this project and then I will be discussing my findings and learning throughout this module. I found this to be a good challenge and test of the programming skills. This has been an excellent opportunity to deeply my knowledge of my own programming skills. I underestimated the research section as I found more research into python chess may have helped. If I had followed the “import chess”, I would say this project would have been much less demanding. I did enjoy the challenge however of figuring out pygame. I think the game really shows the effort I put in.

There are a lot of resources online to build a python chess game and using pygame with it. But these resources were challenging to understand. I watched a coupe tutorials on YouTube which I viewed to help me develop and understand python and pygame in a more comprehensive fashion. There is a chess library for python which employs methods of legality checks and other things such as move generation and move validation. Although by the time I had come across these methods, I had already begun my software development phases and chose not to use them.

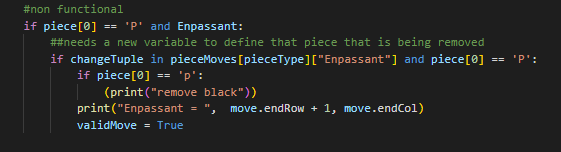
I would like to express my sincere gratitude to my supervisor, Professor Colm O’Riordan in the Information Technology Department of University of Galway for believing in me, supporting me and to everyone else who supported my learning in the last three years of my studies.

**9.Appendix**

In this section, I will discuss the unused code I created when creating this project. There was a not a huge amount of unused code. It was my intention to implement a system of checking the movements of piece to track if they were moving through pieces. This proven to be an issue. Not a “game breaking bug but something that would have been nice to have ironed out. It is checked in the final code in the make move section. There was also the idea of allowing the user to use the castling feature as is standard in chess as well as “Enpassant” another similar feature in chess. I wrote this code but as it was not a key requirement to make this game. It was at the end of the scale of things that needed to be completed. Given more time and energy, I would be able to figure this out.

The “Enpassant” feature was something I was developing for a few days as well as “Castling”. This were features which I really wanted in this project. But they took too much time away from the main focus which was the project.

**Enpassant Code (Figure 9.1)**

The piece variable is set to the position on the board which the user clicked. It first checks if the user has clicked the write piece and then check if the “Enpassant” Boolean is True. After this it checks if the move is in the “Enpassant” section of the dictionary. After this part the code checks again if the piece is correct. The print “remove black” is there a placeholder which is where the code for removing the black piece would have gone. And the print “Enpassant” statement just shows where the piece is on the board.

The castling feature is very similar. Although I altered this code before this screenshot was taken. It was more along the lines of the “Enpassant”. It is undeveloped currently. This is something I look forward to fixing in the future development of this project as I will continue my development of this game in a professional manner for my portfolio.

Text

Description automatically generated This code is similar enough to the previous, this code would slightly more complicated than the “Enpassant” feature. As it requires two moving parts. Therefore, it would need to move two different pieces to two different locations.

**Shortened Castling Code(Figure 9.2)**

**Bibliography**

[1]

Karen Young (2022) “Requirements: Elicitation,” *Systems Analysis and Design*. *CT2103 Systems Analysis and Design, Karen Young, School of Computer Science, NUI, Galway 2022 Requirements: Elicitation 1-February-2022*, Galway, 1 February.

[2]

PasiduPerera (2022) *Chess-python*, *Medium*. Level Up Coding. Available at: https://levelup.gitconnected.com/chess-python-ca4532c7f5a4 (Accessed: March 28, 2023).

[3]

Mohamed Sami (2021) *So*

*ftware development life cycle models and methodologies*, *Mohamed Sami*. Publisher Name https://melsatar.blog Publisher Logo. Available at: https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/ (Accessed: March 28, 2023).