OCR

GCE A Computer Science

Component 03: Programming Project

Name: Edvinas Sadaunikas

Candidate No: 0458

Title of project: Chess game - SchChess

Centre No:

Contents

Analyse the problem

The problem

Chess is a very old game with a lot of history behind it. It has always found a way to keep up with the times whether it be by adding new rules or being adapted to fit new requirements by players. While

Chess has been computerised extensively many of the solutions are online or require powerful computers to be able to run extensive AI that looks many move in depth.

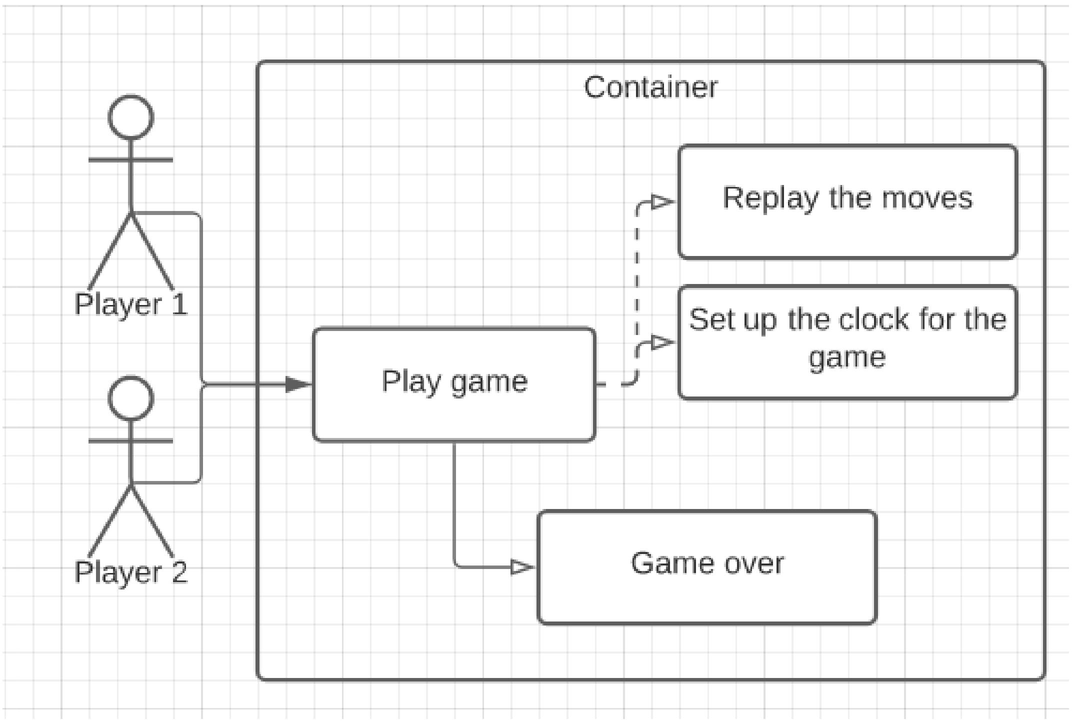
My solution will attempt to combat this by making a chess game that is offline and played between two players. The game will be simple and will only have the required basics this means that most computers will be able to run the program and enjoy chess to their liking.

Stakeholders

The targeted audience of the game will be already existing chess players and the new chess players that may be coming into the game. Since chess is a popular board game it has many already existing players and many people trying it out at least once in their lifetime. My solution will target those players who would want to play Chess on their computer and those who preferably would want to play it as a Desktop application and therefore have the ability to play it offline without the internet to connect to a game server. Most of the existing players that already use a online solution of the game will be able to enjoy my solution as well as its going to be about the same to the end user.

My solution – Computational approach

As my solution is based on an old game and one which has had many iterations of computerisation most of the abstraction is already done. From researching other solutions I understand that I must make my own solution as simple as possible to attract the target audience and to make the game feel the same as playing on an actual board.



My solution will at the core be simple. Its aim it to allow users to play chess, anything else is a bonus. The user will be able to navigate the simple UI using their mouse and will be able to move the piecve by clicking on them and where they want to move them. There will also be move highlighting that will show which places the current piece can move to. This means that the game will be more accessible to players who have never played the game before and will be able to attract more newer players and will get more people into chess.

Research

Currently there are 2 main competitors in the online-based chess games. Chess.com and Lichess.org, while chess.com is bigger and has more features many people still use the open-source and ad-free Lichess instead. The main competition will have clear benefits such as puzzles to improve, other game modes and a news feed that will show players the latest chess news.

Solution 1: Chess.com (https://www.chess.com/)

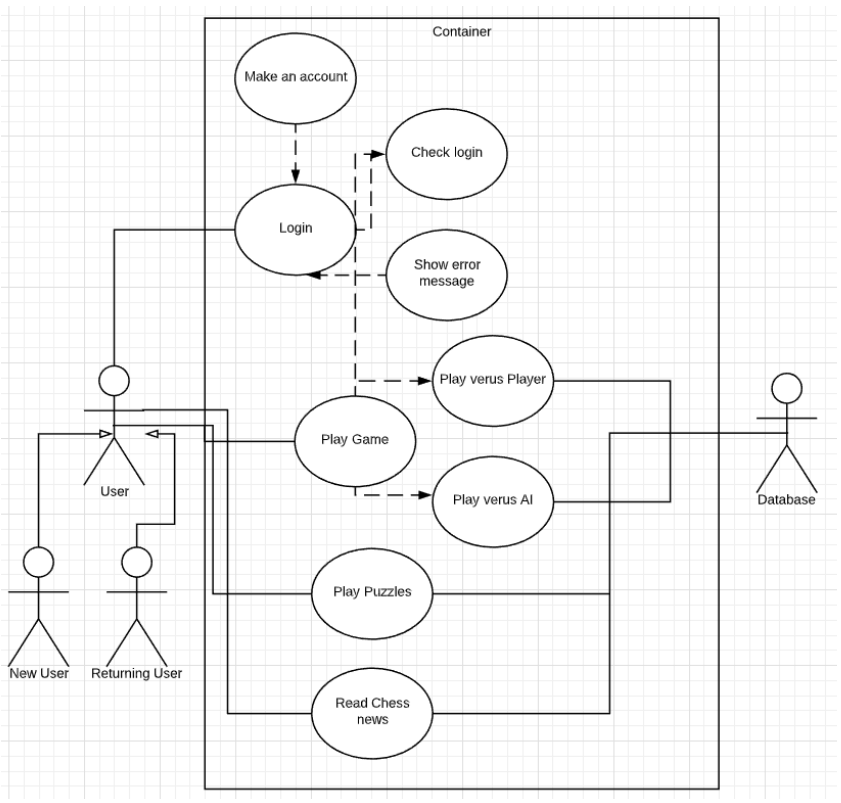
Chess.com is an online chess site which allows anyone to play chess, with a stable internet connection. Chess.com is widely used by many people ranging from chess grandmasters to child prodigies and just causal players. Chess.com managed to solve the problem of sometimes being unable to play chess due to being unable to put out a chess board by making it online. Allowing anyone to go on and play vs another person or a computer. For the Ai games played on the site and the analysis after games users have a choice between the Stockfish Ai also used by Lichess and the Komodo Ai both of these engines rank highly in the world tournaments for chess Ai.

Target audience

Chess.com attracts many used from different nation by giving them many language choices and it attracts players of all games as chess isn't an age dependent game, any age can play the game. It helps attracts the older generations with the news articles on the site, about chess, since they usually enjoy reading news articles and such. The younger generation is pulled in by the chess.com tournaments which happen often and are streamed to a top game livestream site, Twitch.com, this attracts many new players and even professionals of other videogames.

Chess.com - Use case diagram

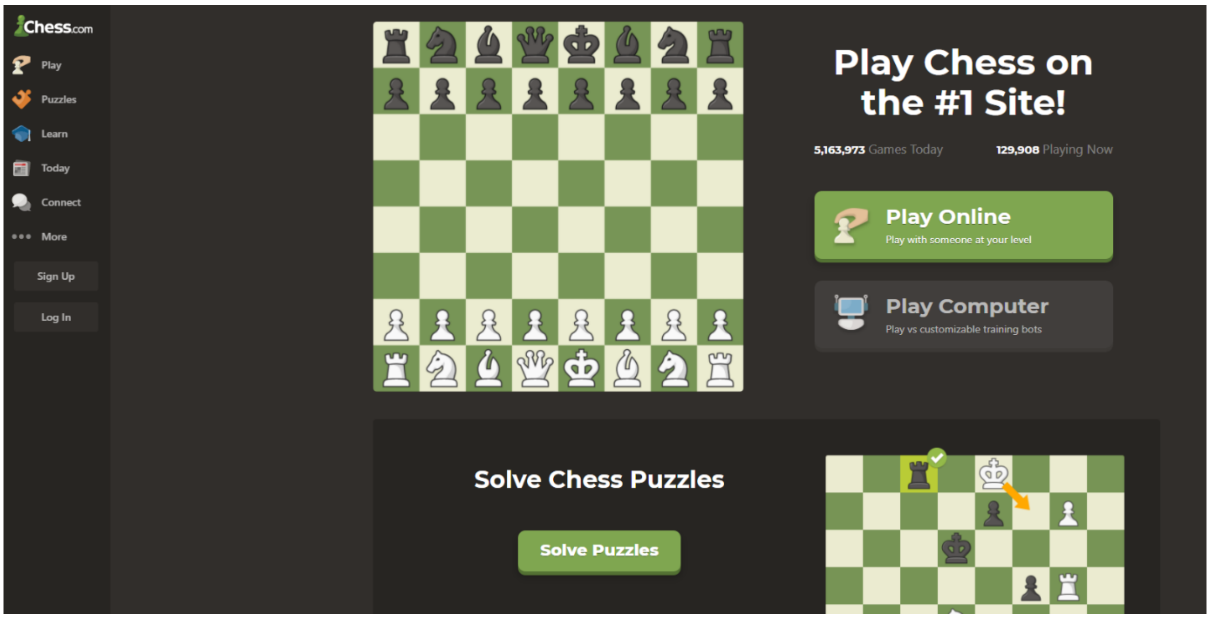
On the Chess.com website when your first visit, you have a choice of to browse their many options such as puzzles to improve and the options of reading chess news or playing a game. Its seen on the diagram as when a user clicks onto the site there are many options for them to proceed down, making an account, playing chess games against players or ai, playing puzzles to help them get better at different aspects of the game or they can read new chess news that tell them what's going on in the world of chess (tournaments, events, etc)



This approach has attracted many new users to try the site out and many returning users to continue using it to learn and play the game.

However, this approach can be seen as too complicated by some people when they visit the site as it may look cluttered to them or the UI may just have too many options.

Chess.com - homepage



On the left-hand side, it gives you the main features of the site such as playing the game, the puzzles, news etc. However, some users just may find this UI unobtrusive to use and therefore not enjoy using the site.

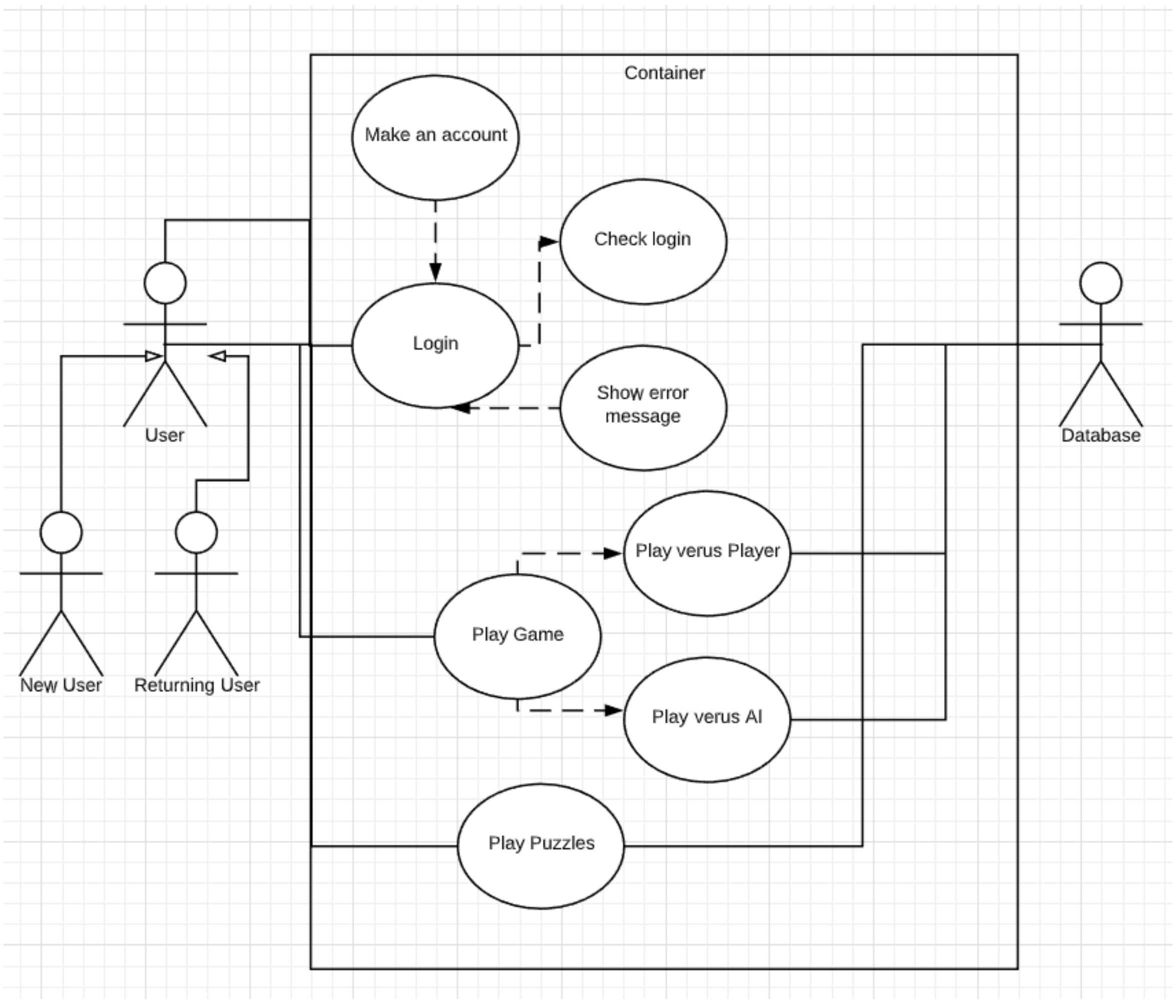
Solution 2: lichess.org (*https://lichess.org/*)

Lichess.org is the next biggest competitor, Lichess operates on an open source server that is a non for profit. Since the website is hosted by a non for profit it means there is no ads and the whole service is free. This free approach has attracted many people who cannot afford to pay for the chess.com premium features. Also, the open source nature of Lichess has allowed its community to help improve the service that they use. For AI games and for game analysis the chess engine used is Stockfish it is an open-source engine which has been developed alongside Lichess since the site uses it so often. It also ranks near the top in the world for best chess engines.

Target audience

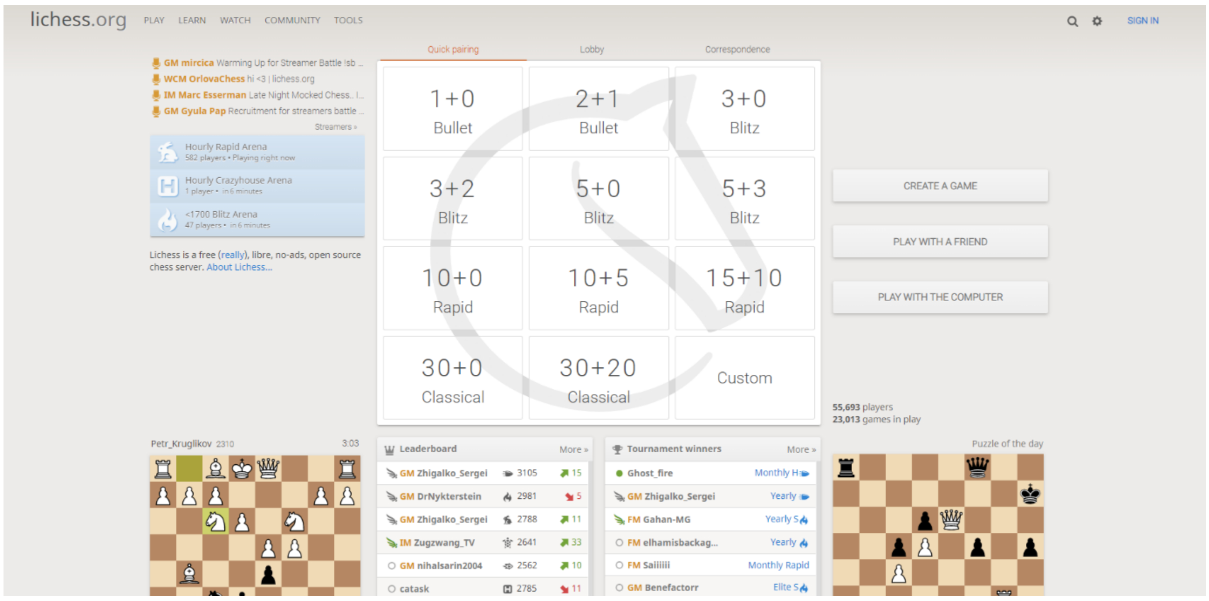
Lichess targets the same chess playing audience as all other competition however since they provide a much simpler layout and therefore their more targeted audience is more niche and smaller.

Lichess.org - use case diagram



This simplicity can be seen in the use case diagram, while not offering as many options to users as Chess.com it provides a much more simplistic layout and offering. This attracts users that previously stayed away from chess.com because they enjoy using the UI more.

Lichess.org - Homepage



GM mircica Warming Up for Streamer Battle !sb ...
WCM OrlovaChess hi <3 | lichess.org
IM Marc Esserman Late Night Mocked Chess.. I...
GM Gyula Pap Recruitment for streamers battle ...

Hourly Rapid Arena
582 players • Playing right now

Hourly Crazyhouse Arena
1 player • in 6 minutes

<1700 Blitz Arena
47 players • in 6 minutes

Lichess is a free (really), libre, no-ads, open source chess server. About Lichess...

55,693 players
23,013 games in play

Leaderboard		More »
GM Zhigalko_Sergei	3105	15
GM DrNykterstein	2981	5
GM Zhigalko_Sergei	2788	11
IM Zugzwang_TV	2641	33
GM nihalsarin2004	2562	10
catask	2785	11

Tournament winners	More »
Ghost_fire	Monthly H
GM Zhigalko_Sergei	Yearly
FM Gahan-MG	Yearly S
FM elhamisbackag...	Yearly
FM Saiiiiii	Monthly Rapid
GM Benefactorr	Elite S

The simplicity can be seen on their homepage, it has the different type of game modes all ready for you to hop into and the other minor features like the leader boards and high rank games all below neatly out of the main view of the user.

Overview:

UI

Taking from both Lichess.org and Chess.com means to make the solution simple but able to complete its main purpose, which is playing chess correctly. Also, it can be added that any bonus features seem to be well apricated by the userbase, which would make the solution more appreciated.

Accessibility

While both lichess.org and chess.com are online based, it can be said that this is liked due to its simplicity, no need to download anything just go onto the sites. My solution can possibly be adapted to fit onto a website and be played online, however it is currently planned to be a downloadable executable.

Chess AI engine

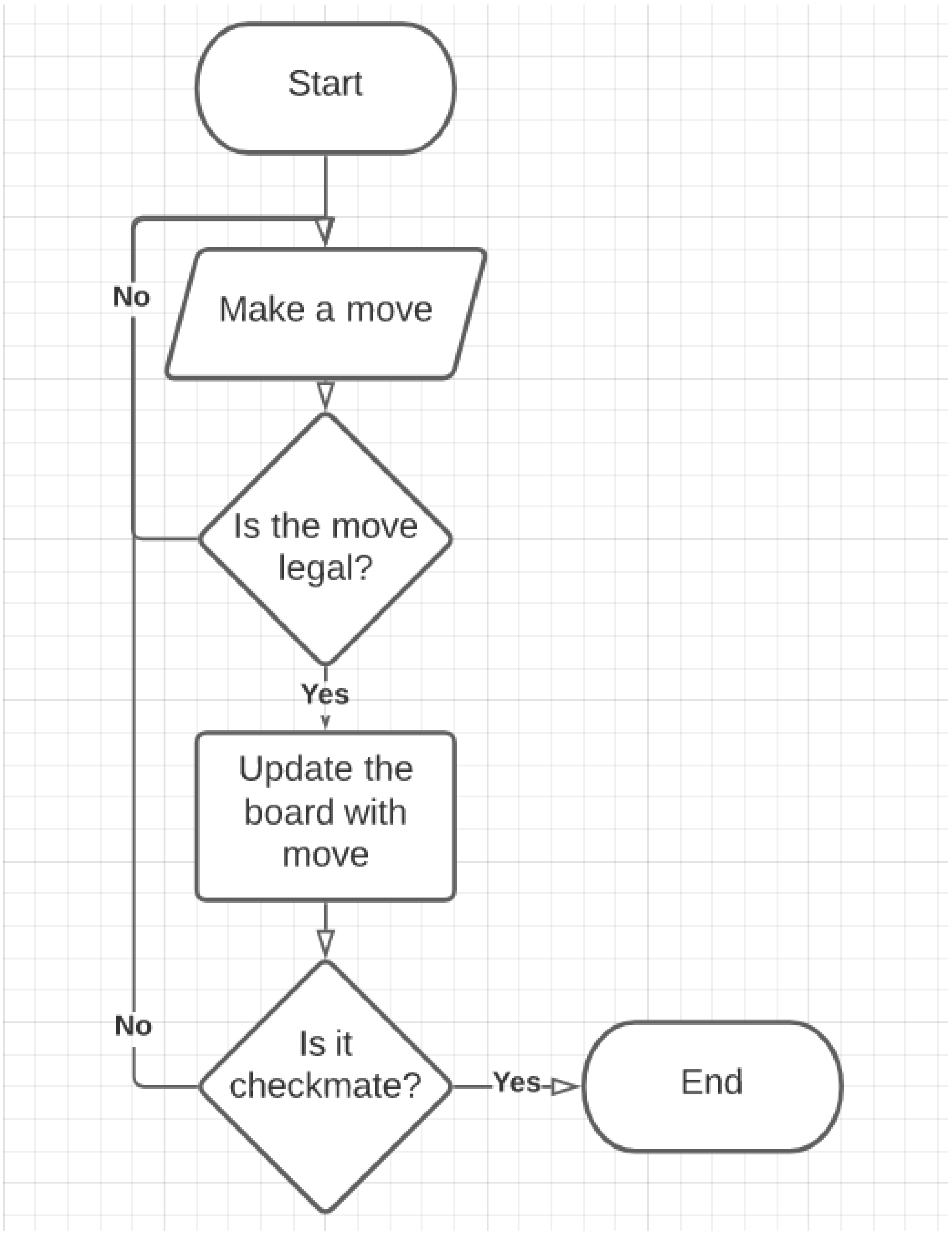
Since my solution will be coded in python it will be hard to integrate an engine that was built on C++ or Js, meaning that I will have to use a python-based chess engine. I have found one called sunfish which seems to be a lightweight and easy chess Ai however it'll need some testing and some finetuning to make it work for the game. However, the main objective is to make it a 2 player game as integrating a engine would add to the games complexity.

Game features

- Move highlighting
  - Legal moves can be highlighted to show what moves the player can play. This will allow for players to have a easier time to pick their move, since they will have to focus less on figuring out where a piece can move.
  - Pieces could also be highlighted to show things such as the King being in check. Tells the player when their King is in check and can help them figure out how to move their King to safety.
- Timers
  - Normal chess is played with the game usually lasting 5 to 10 minutes with a timer counting down the amount of time a player has before the game ending. This provides a more realistic feel to chess as its typically played with a timer and also means that no player can take too long to figure out his move.
  - If a user runs out of time before they move, they will automatically lose by timeout.
- Board
  - The game will involve a realistic board that will have images representing the pieces. This will allow players to play chess as it would be out on a board in front of them

Graphical features

- Image of the board and pieces
- Simple UI to navigate
  - With easy to see and use buttons to navigate and play the game.



This flowchart displays the simplicity of my solution. The core of my solution is intended to be very basic and to only include what is required. The program will ask a player to make their move, then will make sure its legal and will update the game at the end of each move the program can check if its checkmate or not and can end the game or can carry on. However, it can be built up with extra elements such as more UI, AI or puzzles

Since Chess by nature is a simple game it doesn't require much input from the user. Also, since it's an offline solution it doesn't require an account to play. Meaning that it doesn't take any personal data as an input for the solution to work.

Input

- Player board moves
  - Players will be able to move pieces on the board by clicking on the piece they want to move and will be able to click where they want to move it.

Output

- Match result
  - (Victory/Defeat)
- Board state
  - The current board with all the pieces in their places will be output to the users

Processing

- Calculation of Legal moves. Making sure players are able to perform pawn promotion, castling and en passant.
- Keeping track of time for both players. Meaning that players know how much time they have remaining on the clock.
- Keeping track of the board and enforcing the rules This will make sure that a player is unable to play moves that a illegal.

Software/Hardware requirements

Since the game will run on python it'll require the user to be able to download and install python and then be able to run it. The program will not be too heavy, so python minimum requirements are enough.

Hardware

Processor	x86 64-bit CPU (Intel / AMD)
RAM	4GB
Storage	5GB minimum for python
Operating System	Windows 7 or up/ MacOS equivalent

Software

Browser of choice (Chrome/Firefox, etc)	Required to browse internet to download the Python 3 for the game to work
Python 3	Game made on python and therefore requires it for it run.

Features/Limitations

Feature	Reason/ Limitation
Two player game	The game being two player means that two friends can play against each other on a single computer, or a player can play by themselves against themselves. However, it may mean that after a while a player may get bored with only being able to play with a small selection of people.
Simple graphics / UI	Simple graphics mean that there is non much potential for a player to be overwhelmed by many features or complex features. This means that its simple for the player to sit down

	and play chess. However, it also means that many features that are present in competitors may be absent in my solution.
Desktop application	Being a desktop solution, it means it runs a program instead of running on a browser. Running a desktop program means that there are less points of failure and it is much simpler for a player to player, as there are less steps to get the game working. However, it means that the player must first download the game or be able to access the program from their computer.
Mouse controlled	The game being mouse-controlled means that the player is easily able to move the pieces. This also makes the game simpler because it does not require that they know chess notation and it is more realistic to the game of chess since it feels like playing on a real chess board. However, some users may have troubles using their hands correctly and so alternative controlling methods should be considered.
Offline	The game being offline means that a player is not dependant on an internet connection to be able to play a game. This is a benefit above many of the competitors because they require a constant high-speed connection to be able to continue playing. However, this means that the player cannot connect to a server and will be unable to find other players to be able to play against.

Success requirements

At the end of the development process there must be requirements the game must meet to be considered successful and for the development to be considered complete

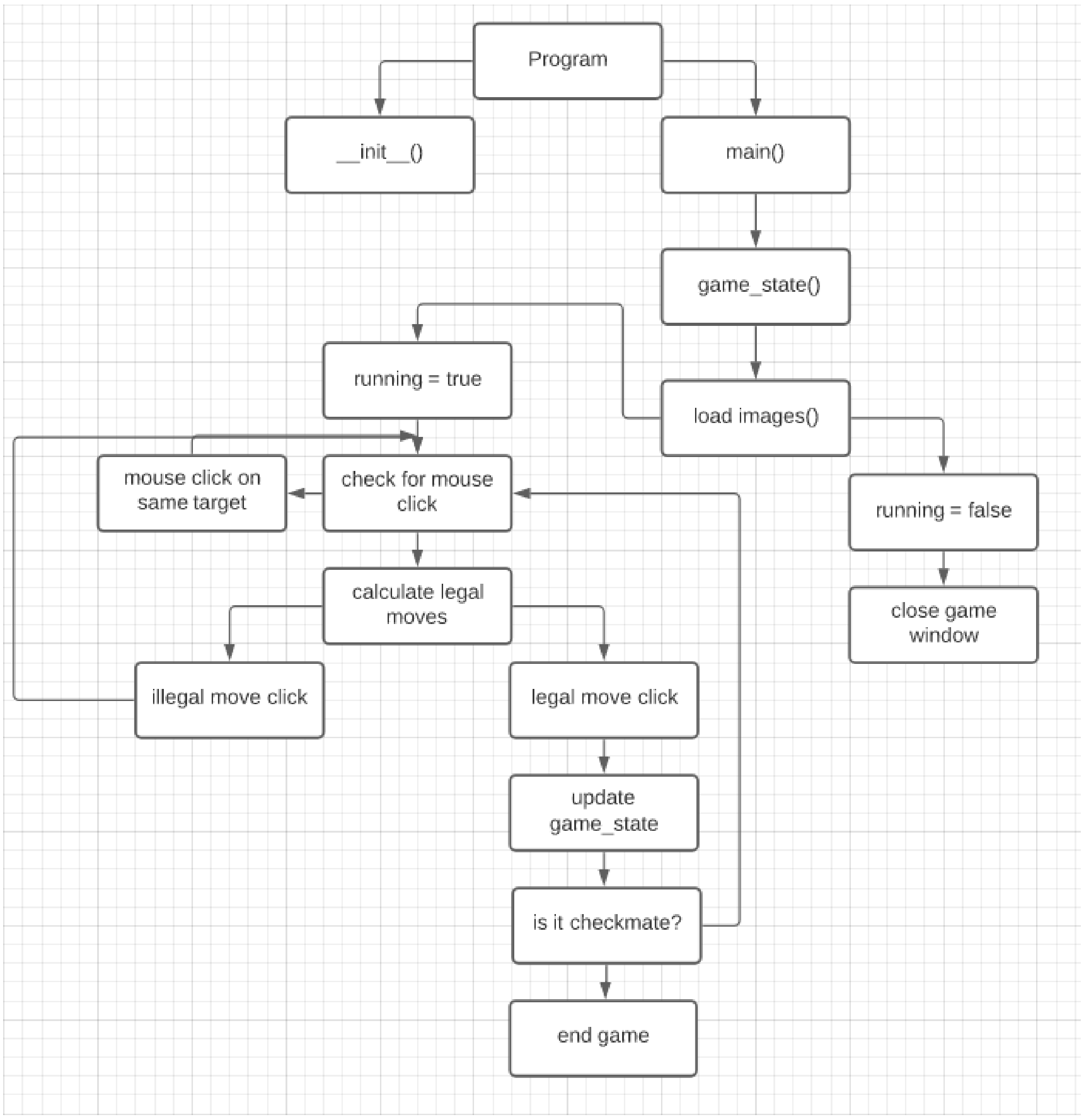
This includes:

- Being able to play a whole game of chess with another player, with timers and rules.
- Having a chess engine to calculate the legal moves and to make sure that a player can checkmate the other player.
- Having meaningful quality of life improvements such as highlighting moves, move suggestion, ect.

If this is met it means that the program can be considered a success and can be used to play a quality game of chess on.

Design specification

Structure diagram



This structure chart shows the basic structure and workings of the game. The game runs in the main loop and calls upon game_state class for the engine. Load images is used to load in the sprites used in the game and can be recalled on for the images. Running, is the main event loop of the game and it will be looking out for when a user's clicks on a piece and for when the user wants to close the game.

Whist running is true the game will continually look out for mouse clicks and make sure that they are not the same square. If the move is not the same square, then it will generate all the legal moves possible and will only allow the user to move the piece to its legal place. After the moved piece is updated to the game_state the engine will then be able to check whether its check mate. If it is checkmate the game can end there and the board can be reset.

If the user wants to close the game, they can do so by clicking the close button on the window and that will set the running event loop to false and will close the game down.

Key Variables

Variable Name	**Data type**	**Purpose**
running	Boolean	Main event loop of the game, watches out for inputs and keeps the game running or closes it down when the user wants to.
Load images	Array	A dictionary that stores all the images used in the game and will be recalled loading them when needed.
Mouse clicks	Integer	Stores the number of total mouse clicks. Used for when the program needs to check whether the user has clicks 2 times and will then be able to move the pieces to the right place.
Mouse click pos (X, Y)	Integer	Stores the position of the X and Y mouse click. This is used to determine whether the click was on the same square and to also determine the row and rank the user is clicking on. Allows for easier movement of pieces.
Legal moves	Array	An array containing all the legal moves for the piece the player clicks on. This means that the player is unable to perform plays that are considered illegal under normal chess rules.
Board	Array	A 2D array that stores the board in its current state, could also be used for first initialising the game. It will be called on to save the board after each move and will therefore be able to keep track of the pieces throughout the game.
Move log	Array	An array holding all the moves played before by the players. This will be used later in the development when implementing takebacks, castling and pawn promotion.
Check mate	Boolean	Stores a Boolean value that tells the player if checkmate has occurred and will end the game.
WhiteToMove	Boolean	Stores a Boolean value that determines which player must make a move.
Colours	Integer	Stores the hex values for the colours used in the game

Validation / Input / Output

The nature of chess and computerised chess is that inputs are very minimal. The user will only be able to use the mouse to input their moves into the game. Therefore, there is very little need for validation of inputs because the only check will be for whether they clicked and where they clicked. Another input the game will be looking for will be the user clicking on the exit button on the window.

Outputs will be limited as well as there will be little to output to the user. The board will be constantly output to the screen. This also includes the board pieces which would be output on the board to the screen for the user to be able to click on the easily and move them. Once either player is in checkmate their king can be highlighted in red to show the player that they are in checkmate and that the game is over.

Algorithms/ Functions/ Classes

<u>Clicking on pieces</u>

This algorithm checks for the number of clicks and the location of the clicks on the board. Starts off by looking for 2 clicks and getting their x and y position. Then compares them to check if they are the same if not, then it gets the rank and file the user has clicked on and passes it on to a function to move the pieces.

```
If mouse click down:
    If len(mouse click down) == 2:
        game_state(make move(mouse_location x,y)
    elif mouse_location x = mouse_location y
        pass
```

<u>Loading images</u>

This algorithm will load in all the images in the images folder, and these will be stored while the program is running so that the game can call upon the dictionary this stores the images on later on. Uses a for loop to make better use of the computers resources as it needs to be looped over for each piece needed to be loaded in.

```
            pieces = 
['bR','bN','bB','bQ','bK','bP','wP','wR','wN','wB', 'wQ', 'wK']
            for x in pieces
                load pieces "images/" + x
```

<u>Drawing the board</u>

This algorithm will draw the board that is stored in the game_state and will be constantly called upon to keep drawing the board after each move. The algorithm uses for loops to go through the eight rows and columns of the chess board. Then it colours odd squares white and even ones blue, the colours on the board can be changed in the code.

```
For row in range dimension
    For column range dimension
        Total = (row+column) mod 2
        If total = even
            Draw white rectangle
        If total = odd
            Draw blue rectangle
```

<u>Game state class</u>

A game state class will be used to keep track of the current board state and will keep track of most elements of the game such as which player has to move and the log of moves.

```
class game_state():
```

```
def __init__(self):
        self.board = []
        self.WhiteToMove = True
        self.MoveLog = ()
```

The class will hold information about the board and will contain most of the main functions to keep track of different elements to later enable castling, promotion and en passant. The class will be able to call upon other functions and classes to move the pieces or receive extra information.

```
def makeMove(self, move):
        self.board = [move start row, move start
        column] = empty
        self.board = [move end row, move end
        column = piece moved
        swaps self.WhiteToMove
        self.MoveLog.append move
```

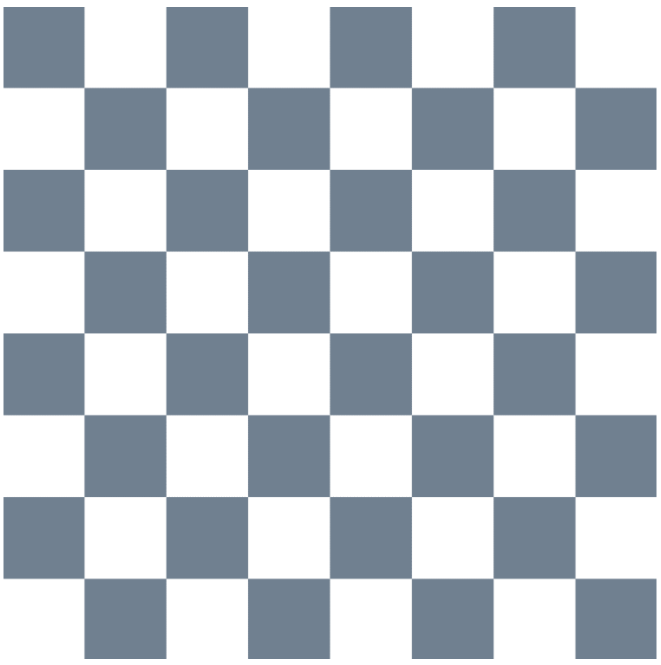
When calling upon the function to make a move it take information about the pieces starting row and column and sets it to empty then it makes the board record the move to the end row and column. Afterwards it swaps WhiteToMove to be false meaning that black now has to make a move and appends the move to the MoveLog list.

Interface design

The design of the game will be kept simple and will just include the chess board printed on the screen.

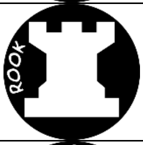
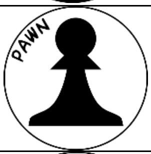
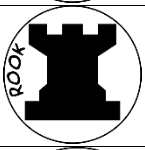
Board on screen

The board will be displayed in a 800x800 window on the players screen. Extra options could be added to show the user the time or the move logs on the side of the UI. This could be achieved by using pygame and drawing rectangles on the screen layer.



Chess pieces

These images will be used in the game to clearly display the pieces on the board. These images can easily be swapped out for other images if the players wish to do so. This could be done by replacing the images with the same name in the images folder. This will ensure that the right images are loaded into the right places as the game begins.

Piece	Image
White Pawn	
White Knight	
White Bishop	
White Rook	
White Queen	
White King	
Black Pawn	
Black Knight	
Black Bishop	
Black Rook	
Black Queen	
Black King	

Usability features

The window for the game is large and the icons for the pieces are 100x100, this size was picked because the bigger pieces means that its easier for players to see and 800 is a number that can easily be divisible into images big enough for the game. The players can change the size of the window and the pieces in the code, however it should be noted that the number must easily be divided by eight otherwise the will be issues with the UI.

The simple design of the UI means that players will have a easy time understanding what to do and will not be overwhelmed by many elements that are present in other solutions.

Testing strategy

The program will be developed using iterative development, meaning that it will be split into smaller parts. Each stage of the program will be tested, and it will also be tested once its fully complete. Discovered bugs will be fixed in the next cycle as to not delay the competition of each cycle as that may delay the completion of the whole project.

When testing I will record the input used, the expected result and whether the program preformed as expected, if not then I will record the bug and any possible errors.

Testing checklist

Test number	Input	Expected outcome	Preformed as expected?
1		Chess pieces are printed on screen as required.	
2	Clicking on piece, then moving it to another square	For the piece to move to another square from its starting one	
3		Updated chess board which shows the moved piece	
4	Clicking on other player's piece to try to capture it	One player's piece captures the other player's piece.	
5		Chess rules work as intended, piece move restrictions, pawn promotion, castling, en passant.	
6	Moving a pawn to the last rank to earn promotion	Moving the pawn to the end of the board should allow the pawn to be promoted into whatever piece a player wants, except King.	
7	Moving the King over the Rook	Should allow the player to Castle if the conditions to do so are met.	
8	Moving enemy King into checkmate	Should end the game and provide a message and visual clue to the players that the King is in checkmate.	
9	Clicking the close button on the window	Should close the game	

Development plan

A development plan is used to more clearly establish the aims and goals of each cycle of development and the testing that will be done to it at the end of the cycle. Also provides a clear timeframe that the development of the program should stick to. Thus, ensuring that the program is completed by the final deadline.

Cycle	Aims/ Goals of cycle	Testing	Completion date
1	• Output of chess board on screen and the pieces. • Movement of pieces by using left click.	• Loading the chess board and checking that it is output in the right layout. • Loading all the pieces onto the board in the right order • Making sure that the pieces are able to be moved when the player clicks on them and a square to move them.	9/03/21
2	• Bug fixing from cycle 1 • Implementation of chess engine. • Only allowing legal moves to be played. • Including pawn promotion, castling, en passant. • Calculating checkmate.	• Making sure chess engine only calculates the correct and legal moves. • Pawn promotion working as intended when conditions met. • Castling working only when King and Rook are unmoved. • En passant working according to rules.	23/03/21
3	• Bug fixing from cycle 2 • Implementing quality of life changes such as legal move highlighting and showing suggested moves. • Expanding UI to include move log and a timer for the players.	• Making sure move highlighting highlights the correct moves. • UI elements working correctly, clock cannot be going down too quickly and must change sides when a player makes their move. • Move log should show all the moves played by the players in the correct chess notation.	6/04/21

Development Cycle 1 – Basic aspects of Chess

Development plan – Chess Game

Date	Task	Specification link
07/03/21	Coding a game state object that would keep track of most the data for the program and getting a chess board printed on screen	Immersion, allows the players to more easily visualise the board and to think about the moves they want to do.
08/03/21	Coding chess pieces into the board and getting high quality images for the pieces.	Further improves Immersion as the players will be able to see the

		pieces and will be more able to focus on game strategy.
10/03/21	Coding to allow the pieces to move and take other pieces, development of the rules for the pieces will be done in the next phase.	The game will become playable, and players will be able to begin to play against each other.
12/03/21	Optimization and bug fixing.	Optimizing the game provides a better player experience and will make better use of the computer's resources.
14/03/21	Final testing of cycle 1 and review of possible improvements that may be needed when proceeding.	Testing will make sure that all pieces are able to move as they should be able to and that the right legal moves are being highlighted.

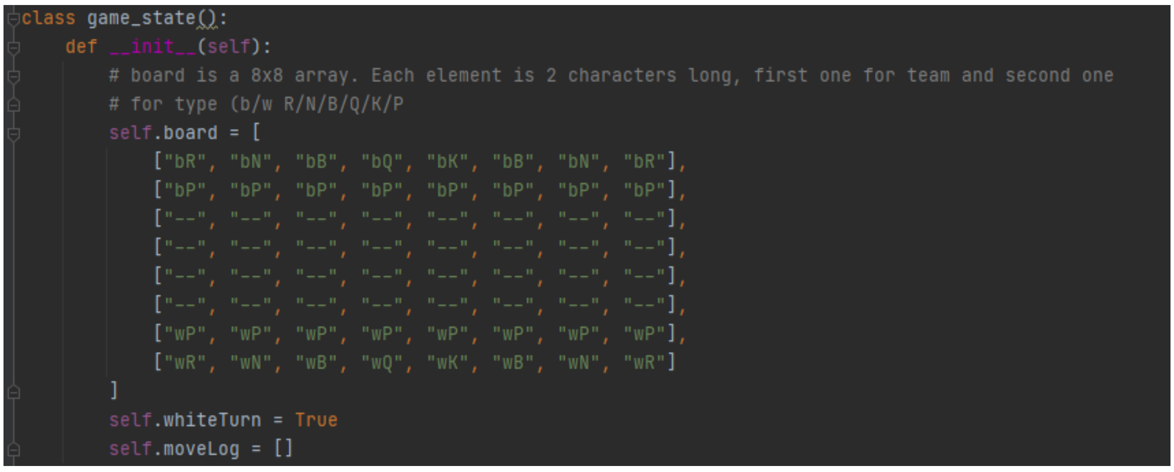
The development plan displays a rough timeline that the basic aspects of the game will be coded and provides a clear deadline for when first iteration should be completed. Each task must be completed before going on to the next as this will be the foundation of the game therefore skipping basic features will impact the future playability of the game. The complete time for cycle 1 is 2 weeks, and it should not be delayed because it will bog down the rest of the development cycle.

Cycle one – Documentation

At each point of cycle one the program must be tested, and it should be working as needed. When coding in pycharm it has a running tab, which allows easy debugging and if errors in the code occur it quickly shows which line has the issue.

Game state object

The game state stores most of the data that will be used and processed by the program. It stores the starting order of the board and can easily be updated when players make a move, it also stores a move log and keeps track of which team has to move.

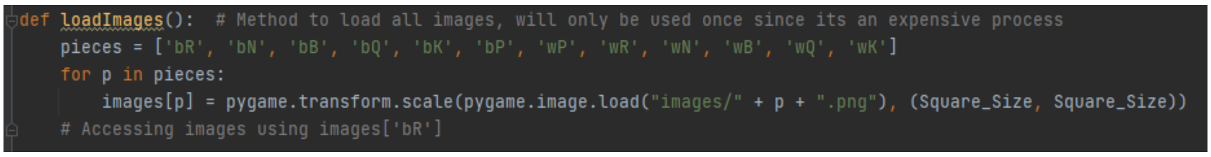


```
class game_state():
    def __init__(self):
        # board is a 8x8 array. Each element is 2 characters long, first one for team and second one
        # for type (b/w R/N/B/Q/K/P
        self.board = [
            ["bR", "bN", "bB", "bQ", "bK", "bB", "bN", "bR"],
            ["bP", "bP", "bP", "bP", "bP", "bP", "bP", "bP"],
            ["--", "--", "--", "--", "--", "--", "--", "--"],
            ["--", "--", "--", "--", "--", "--", "--", "--"],
            ["--", "--", "--", "--", "--", "--", "--", "--"],
            ["--", "--", "--", "--", "--", "--", "--", "--"],
            ["wP", "wP", "wP", "wP", "wP", "wP", "wP", "wP"],
            ["wR", "wN", "wB", "wQ", "wK", "wB", "wN", "wR"]
        ]
        self.whiteTurn = True
        self.moveLog = []
```

Board holds information at 8x8 2D array that keeps track of the board and helps first initialise it when a new game begins. WhiteTurn holds a Boolean value that allows the class to keep track of which side has to make a move and can easily be changed when a player makes a move. MoveLog holds a list of all the moves made by players, this can be used later to display it on the UI or to replay certain moves from a game.

Chess pieces and sprites

The chess pieces are loaded into the game into a dictionary this allows the to be easily accessed and they are only loaded once at the start because loading images in pygame is an expensive process.

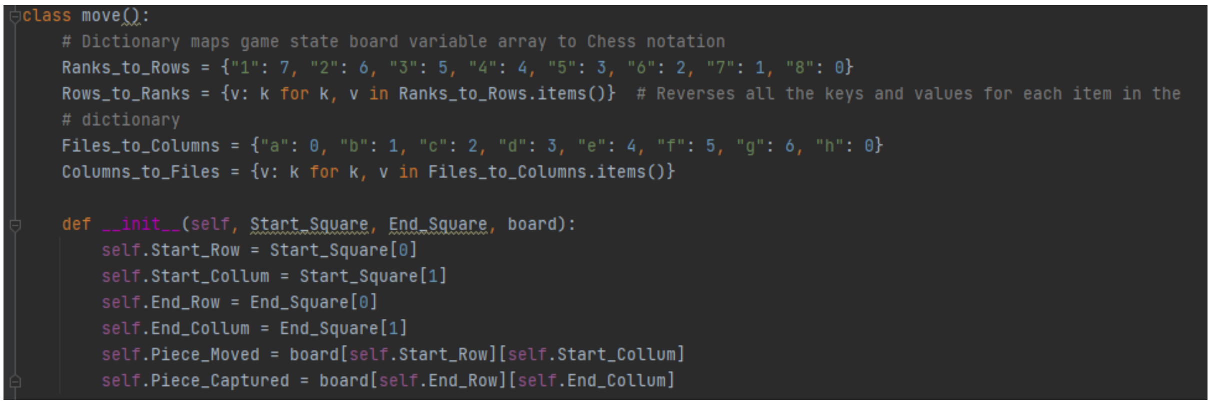


```
def loadImages():  # Method to load all images, will only be used once since its an expensive process
    pieces = ['bR', 'bN', 'bB', 'bQ', 'bK', 'bP', 'wP', 'wR', 'wN', 'wB', 'wQ', 'wK']
    for p in pieces:
        images[p] = pygame.transform.scale(pygame.image.load("images/" + p + ".png"), (Square_Size, Square_Size))
    # Accessing images using images['bR']
```

All the required pieces are loaded into a dictionary that can easily be accessed later. A for loop is used to make loading the images more efficient and to save on resources, this has the added benefit of then being able to run on less powerful computers.

Moving the pieces

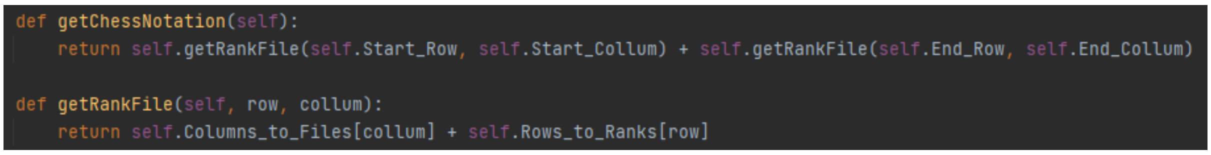
Making the pieces move required a move class and a addition to the while loop that will look out for mouse clicks. This required coding in logic for when a user may click the same square twice and to look out for two clicks.



```
class move():
    # Dictionary maps game state board variable array to Chess notation
    Ranks_to_Rows = {"1": 7, "2": 6, "3": 5, "4": 4, "5": 3, "6": 2, "7": 1, "8": 0}
    Rows_to_Ranks = {v: k for k, v in Ranks_to_Rows.items()}  # Reverses all the keys and values for each item in the
    # dictionary
    Files_to_Columns = {"a": 0, "b": 1, "c": 2, "d": 3, "e": 4, "f": 5, "g": 6, "h": 0}
    Columns_to_Files = {v: k for k, v in Files_to_Columns.items()}

    def __init__(self, Start_Square, End_Square, board):
        self.Start_Row = Start_Square[0]
        self.Start_Collum = Start_Square[1]
        self.End_Row = End_Square[0]
        self.End_Collum = End_Square[1]
        self.Piece_Moved = board[self.Start_Row][self.Start_Collum]
        self.Piece_Captured = board[self.End_Row][self.End_Collum]
```

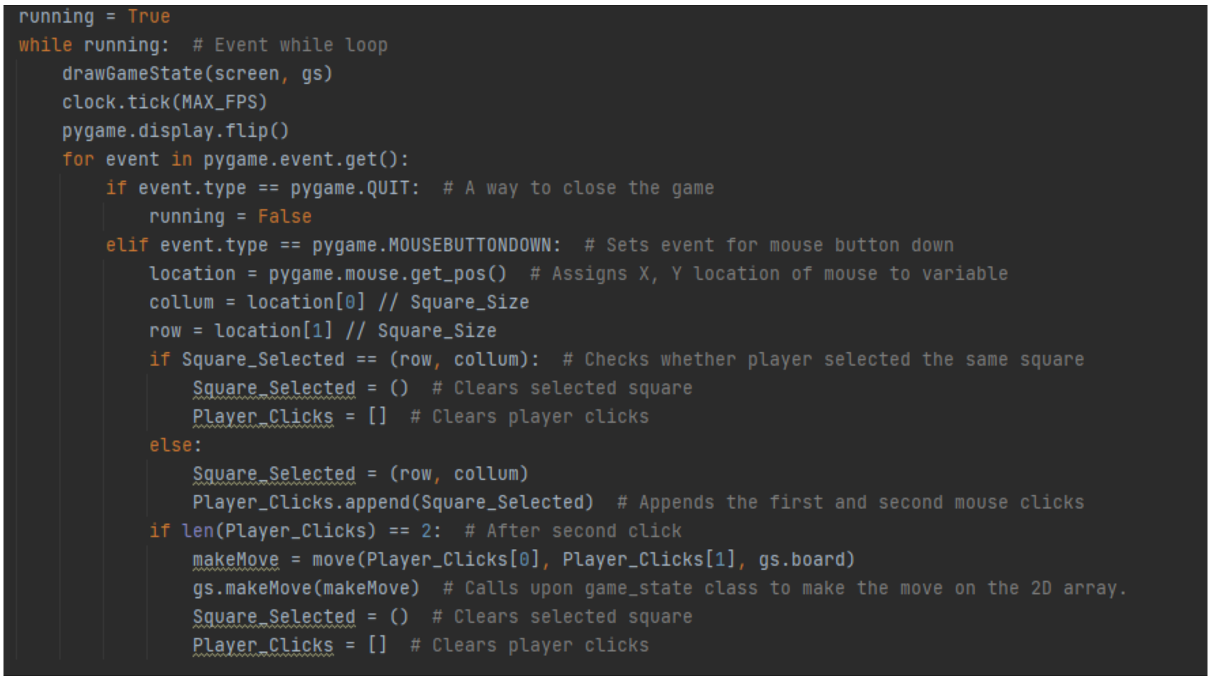
A move class was made to keep track of which rank and file the piece was moved from and to, and to keep track of whether a piece was moved and whether it was captured. A dictionary was defined at the start to allow for methods to find chess notation on the board which will help later in debugging or when needing to print out moves. This is because it is a lot easier to read chess notation than the index of the array, E.g. [4, 4 -> E,4].



```
def getChessNotation(self):
    return self.getRankFile(self.Start_Row, self.Start_Collum) + self.getRankFile(self.End_Row, self.End_Collum)

def getRankFile(self, row, collum):
    return self.Columns_to_Files[collum] + self.Rows_to_Ranks[row]
```

These two functions in the class can be called upon to get the values of the rank and file and then to piece them together to get the complete chess notation. The getChessNotation function can be added to by making it more close to chess. Such as which piece moved, and a ‘x’ can be added to show when a piece was captured.



```python
running = True
while running:  # Event while loop
    drawGameState(screen, gs)
    clock.tick(MAX_FPS)
    pygame.display.flip()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:  # A way to close the game
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:  # Sets event for mouse button down
            location = pygame.mouse.get_pos()  # Assigns X, Y location of mouse to variable
            collum = location[0] // Square_Size
            row = location[1] // Square_Size
            if Square_Selected == (row, collum):  # Checks whether player selected the same square
                Square_Selected = ()  # Clears selected square
                Player_Clicks = []  # Clears player clicks
            else:
                Square_Selected = (row, collum)
                Player_Clicks.append(Square_Selected)  # Appends the first and second mouse clicks
            if len(Player_Clicks) == 2:  # After second click
                makeMove = move(Player_Clicks[0], Player_Clicks[1], gs.board)
                gs.makeMove(makeMove)  # Calls upon game_state class to make the move on the 2D array.
                Square_Selected = ()  # Clears selected square
                Player_Clicks = []  # Clears player clicks
```

The event loop was added to look out for a mouse button down press, this then gets the presses location and waits for another button press. There is logic coded into it to make sure that a player does not click on the same rank and file again and after the move is done it clears the Square_Selected and Player_Clicks variables.

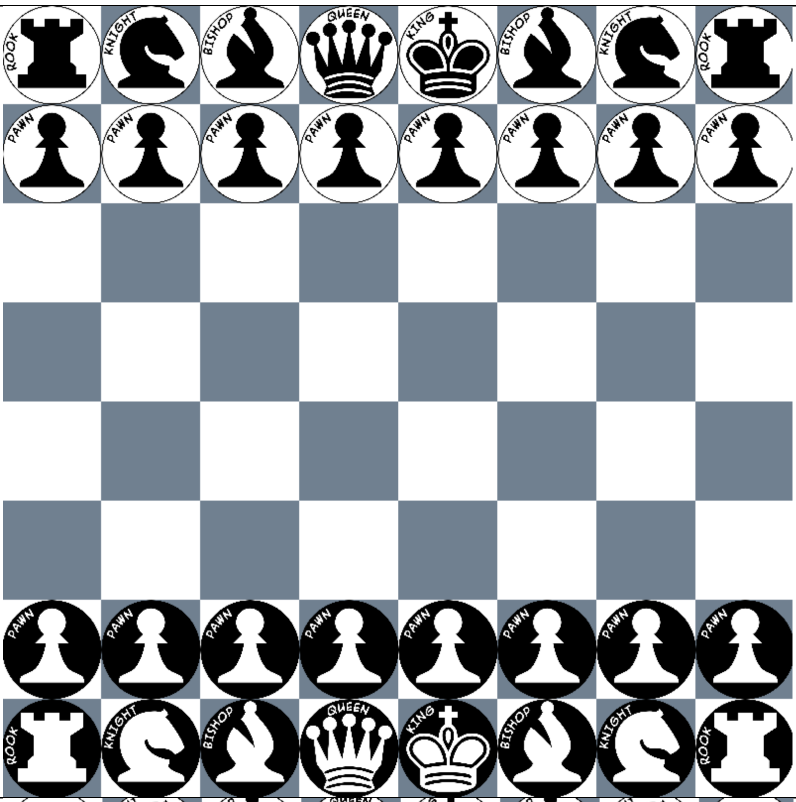
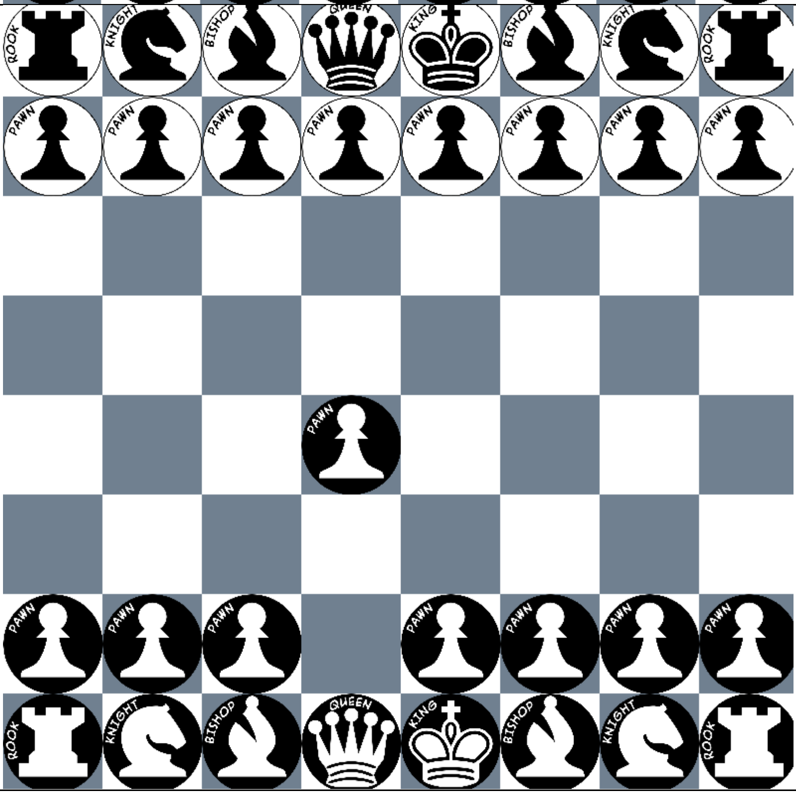
Cycle one – Testing

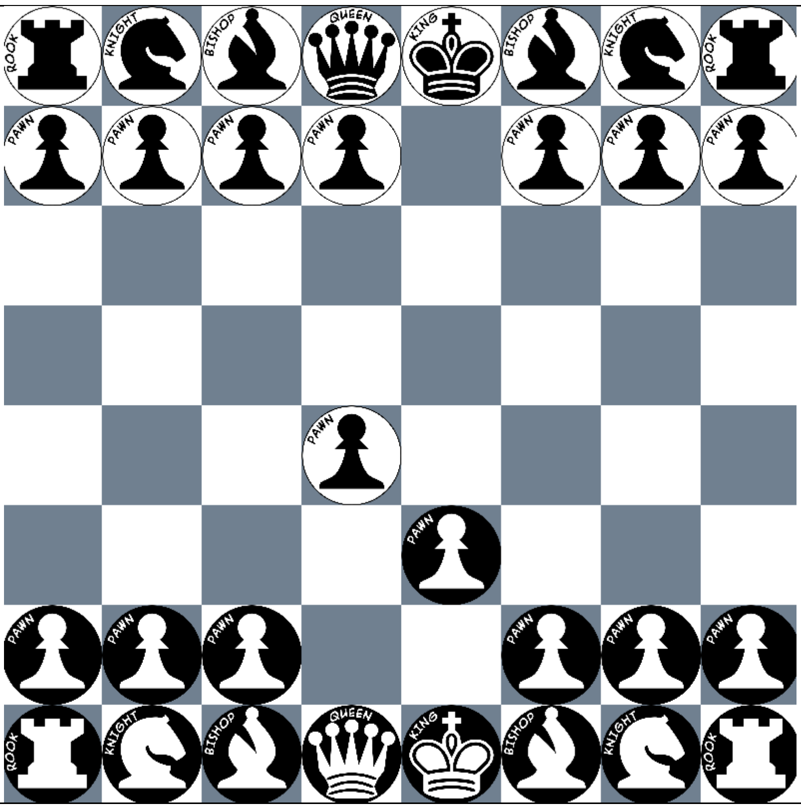
The testing for cycle one involves checking that all the requirements for cycle one are met and finding any bugs that may be present in the code.

Test number	Input	Expected outcome	Preformed as expected?	Screenshot reference
1		Chess pieces are printed on screen as required.	Yes	1
2	Clicking on piece, then moving it to another square	For the piece to move to another square from its starting one	Yes	2
3		Updated chess board which shows the moved piece	Yes	2
4	Attempting to take another player's piece.	Player one takes player two piece.	Yes	3

Screenshot reference

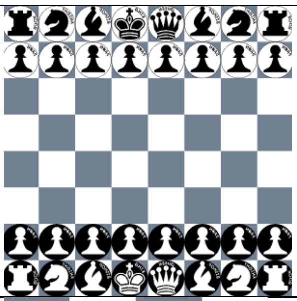
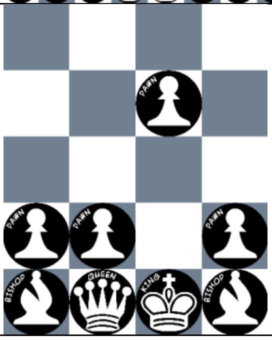
Screenshot reference	Screenshot

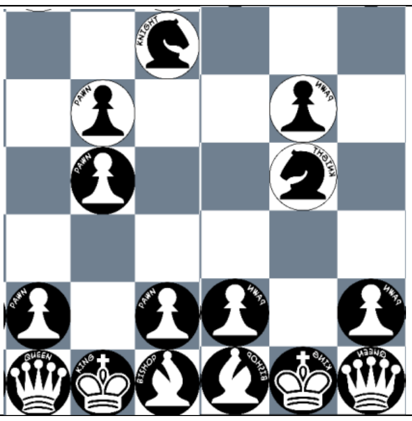
1	ROOK
KNIGHT
BISHOP
QUEEN
KING
BISHOP
KNIGHT
ROOK
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
ROOK
KNIGHT
BISHOP
QUEEN
KING
BISHOP
KNIGHT
ROOK
2	ROOK
KNIGHT
BISHOP
QUEEN
KING
BISHOP
KNIGHT
ROOK
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
PAWN
ROOK
KNIGHT
BISHOP
QUEEN
KING
BISHOP
KNIGHT
ROOK

3	

Testing reveals that the game so far works as intended by the end of cycle 1. Pieces are displayed on an 8x8 chess board and the player is able to move them as they like. Only issue is that so far there are no rules for how a player may move a piece, this can result in players who don't know the rules of chess making illegal moves. Cycle two would involve adding rules to the chess engine to calculate all legal moves and only allow players to make a legal move. It would also keep track of Kings, Rooks and Pawns positions to check for Castling and for En passant.

Cycle one – Evaluation

User requirement	**Test evidence**
Print out current board. **Evaluation** – The current chess board that is stored in game_state object is printed out with imagines to display the different pieces. This meets the user requirement set out for development cycle one and will be used later on to expand on to the game to make it more playable.	
Ability to move chess pieces. **Evaluation** – The players are able to move their pieces as they want to. This allows them to begin playing the game and play against another player. Meets the end requirement for development cycle one and allows users to play the game, however moving pieces lacks any rules and illegal moves are able to be played.	

Taking over player's pieces **Evaluation** – Players are able to take other players pieces. This allows them to play the game as intended.	

Overall, cycle one completes it aims of making the basic foundation of the chess game. It prints out the board with high quality sprites for the pieces and allows for pieces to be moved as the players like. The current state of the game means that there is no promotion, castling or engine to check the legal moves. The limitations are planned to be fixed in cycle two with added quality of life fixes such as introducing move highlighting to show which moves can be played, adding this feature will help people who don't know much about chess to begin playing it for the first time.