System Development

Chess

A2 Computing Coursework

Aman Gill

Contents

Description of the current system & background 5
The Problem 5
Identification of the prospective user(s) 5
Identification of User Needs & Acceptable Limitations 6
Data source(s) and destination(s) 6
Data volumes 6
Analysis data dictionary 7
Data flow diagrams (DFDs) (existing and proposed system) to level 1 7
- Existing System 7
- Proposed System 9

Objectives for the proposed system 10
Realistic appraisal of the feasibility of potential solutions 11
Justification of chosen solution 12
Flow Chart 13
Class Definitions 13
Algorithms Using Pseudocode 14
- Global Variables 14
- The Rules of Chess 14
- The Initial Position of the Pieces 14
- General Piece Rules 15
- Individual Piece Rules 17
- The Game Class 22
- The Board Class 30
- Timers 33
- Saving the Text Log 34

Class Diagrams 35
Hardware Specification 37
Design Data Dictionary 37
Interface Design 38
Preliminary Test Plan 40
System Overview 41
Maintenance 41
- Changing the Piece Images 41

Changing the Background Colour of the PictureBoxes 41
Progression from Design Stage and Problems during Implementation 42
Bugs 42
Outline Test Plan 44
Test Results 44
Fig. 1-1 48
Fig. 1-2 48
Fig. 1-3 49
Fig. 1-4 49
Fig. 1-5 50
Fig. 1-6 50
Fig. 1-7 51
Fig. 1-8 51
Fig. 2-1a 52
Fig. 2-1b 52
Fig. 2-2 53
Fig. 2-3 53
Fig. 2-4 54
Fig. 2-5 54
Fig. 3-1a 55
Fig. 3-1b 55
Fig. 3-2a 56
Fig. 3-2b 56
Fig. 3-3a 57
Fig. 3-3b 57
Fig. 3-4a 58
Fig. 3-4b 58
Fig. 3-5a 59
Fig. 3-5b 59
Fig. 3-6a 60
Fig. 3-6b 60
Fig. 3-7a 61
Fig. 3-7b 61
Fig. 3-7c 62

Fig. 4-1 62
Fig. 4-2 63
Fig. 4-3 63
Fig. 4-6 64
Fig 4-7a 64
Fig. 4-7b 65
Fig. 5-1a 65
Fig. 5-1b 66
Fig. 5-1c 66
Fig. 5-1d 67
Fig. 5-1e 67
Fig. 5-2a 68
Fig. 5-2b 68
Fig. 5-2c 69
Fig. 5-2d 69
Fig. 5-2e 70
Fig. 5-3a 70
Fig. 5-3b 71
Fig. 5-4a 71
Fig. 5-4b 72
System Objectives 73
Analysis of Feedback 75
Potential Improvements 76
Installation 77
Basic Playing 77
  Moving Pieces 77
  Castling 78
  Pawn Promotion 79
  When is it Check or Checkmate? 81
Resetting the Board 81
Using the Clocks 82
  Normal Clock 82
  Speed Chess 82
Saving the Move Log 83

Full Program Listing....84
Main Form....84
Game Class....90
Board Class....112
Piece Class....115
Pawn Class....116
King Class....119
Queen Class....120
Knight Class....121
Rook Class....122
Bishop Class....122

Analysis

Description of the current system & background

The school chess club meets once a week at lunchtimes to improve upon their chess skills and to compete against each other.
The system currently being used by the chess club members is to use a physical chess board, with a manual chess clock that requires the users to press a button once they have made their move. Pencil and paper is generally used to record the moves made in a game.
There are limitations to this system. A button has to be pressed on the clock after each move has been made. This means that, due to the delay from the player making the move and pressing the button, more time will be taken off the clock than was actually used by the player to make the move.
Another limitation is manually recording the moves on paper. Whoever is recording the moves can make a mistake in writing down the moves, and the paper can also easily be misplaced or damaged. If the chess clocks were being used, this would also mean that either a third person would have to be present to record the moves, or the players would have to record their moves themselves, which would use even more of the player's time.

The Problem

The chess club members don't have a reliable way to record their moves in a chess game, or to replay through past games to see where mistakes have been made. Also, they do not have a way to automatically change the turn on the clock once a move has been made.

Identification of the prospective user(s)

My client will be Mr RG Patten, who is the head of the school chess club. I will be consulting him to work out the specifications of my program.
The users of the system will primarily be the members of the chess club. They will primarily be using the chess program as a means to easily record the moves in a game, and to replay their games back to them, with the added functionality of automatically changing the turn on the clock.

Identification of User Needs & Acceptable Limitations

The system must follow the rules of chess, and not allow the player's to deviate from these rules. This is so that the system can be used as a teaching aid to show what is and is not a valid move.

The system must include accurate chess notation for each move. This is to familiarise players with correct notation and also so that Mr Patten can look through the game and show the players any mistakes they are making.

The system should include clocks. This is to get the players to think ahead and make decisions fast.

The system could include a way of replaying through previous games. This is so that games can be easily reviewed for mistakes, with a visual aid of how the current game looks. However, this could be difficult and is not entirely necessary.

Data source(s) and destination(s)

The user will input few forms of data. The main source of data will be the user clicking on pieces on the virtual board, which will result in them moving. The user can also input the amount of time they want the clocks to run for, and the name of the text file the system creates from the Move Log. All of this data can be input using a keyboard and/or a mouse.
There will be an option for the user to save the log of moves to a text file, the name of which will be entered by the user. This text file can then be loaded by entering the name of the text file after clicking the "load" button.
The sole data destination will be the screen, which will display the chess board, the position of pieces on the board, the time remaining for each player, a log of all moves made in the current game, and will indicate whose turn it is.
After every move, the board will be refreshed to show the new positions of the pieces on the board. The turn indicator will change after each move. The log of the moves will be added to after each move. The time remaining on the chess clock will change each second for the player whose turn it is.

Data volumes

The program will be able to store a log of all the moves in a game of chess that has been played in a text file. The size of the text file will depend on the length of the game (usually between 20 and 50 moves); it will be about 8 bytes per turn, as well as about 20 bytes extra to record how much time is left for each player, and whose turn it is (or if checkmate has

been reached). For an average game of about 35 moves, this will produce a text file of about 300 bytes.
There will be 12 PNG images, each around 4.5 kb in size.

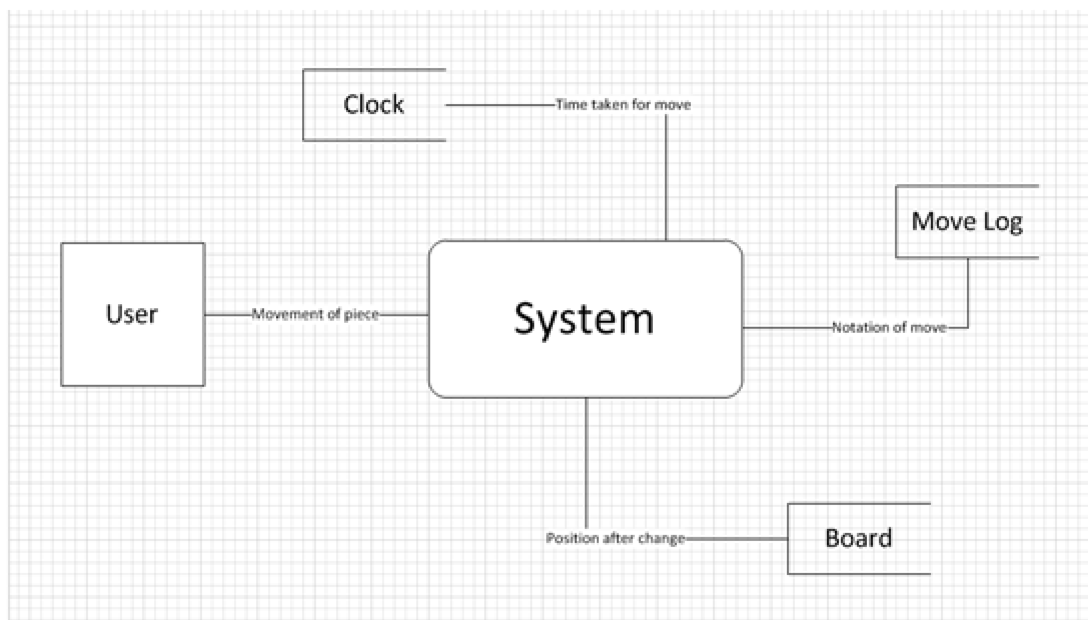
Analysis data dictionary

The program will not use a database, and will not store much data to be used in the program. The data it will be using are 12 PNG images: two for each type of piece (one of each colour).

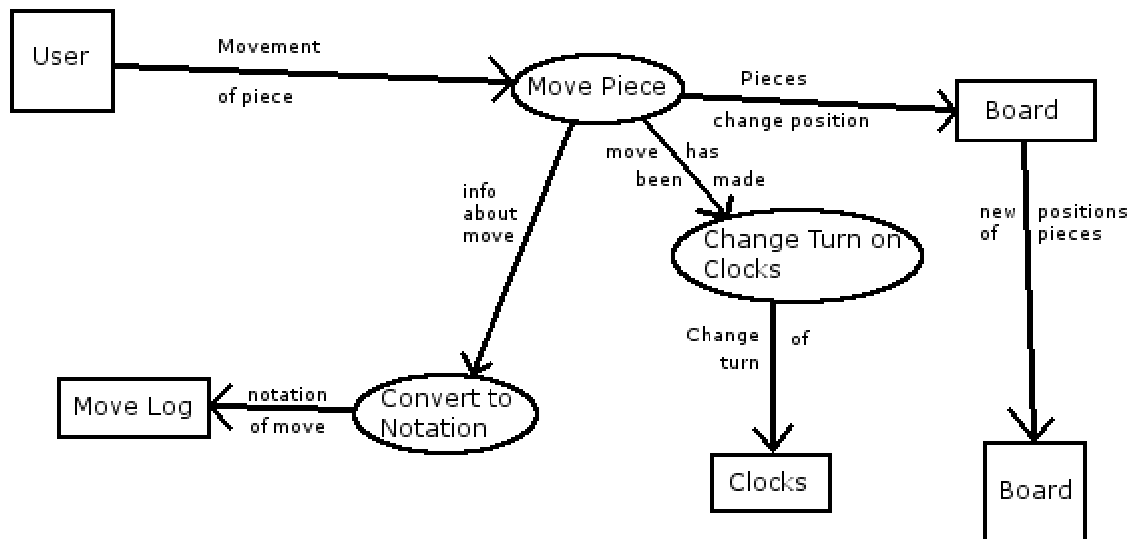
Data flow diagrams (DFDs) (existing and proposed system) to level 1

Existing System

Level 0

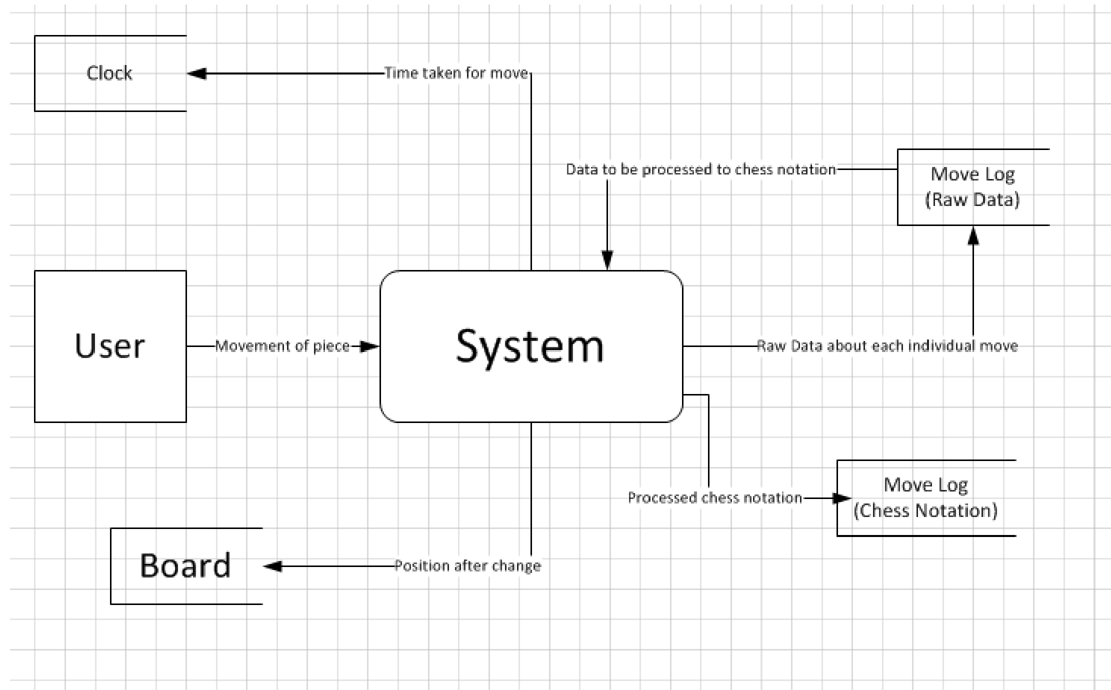


<u>Level 1</u>

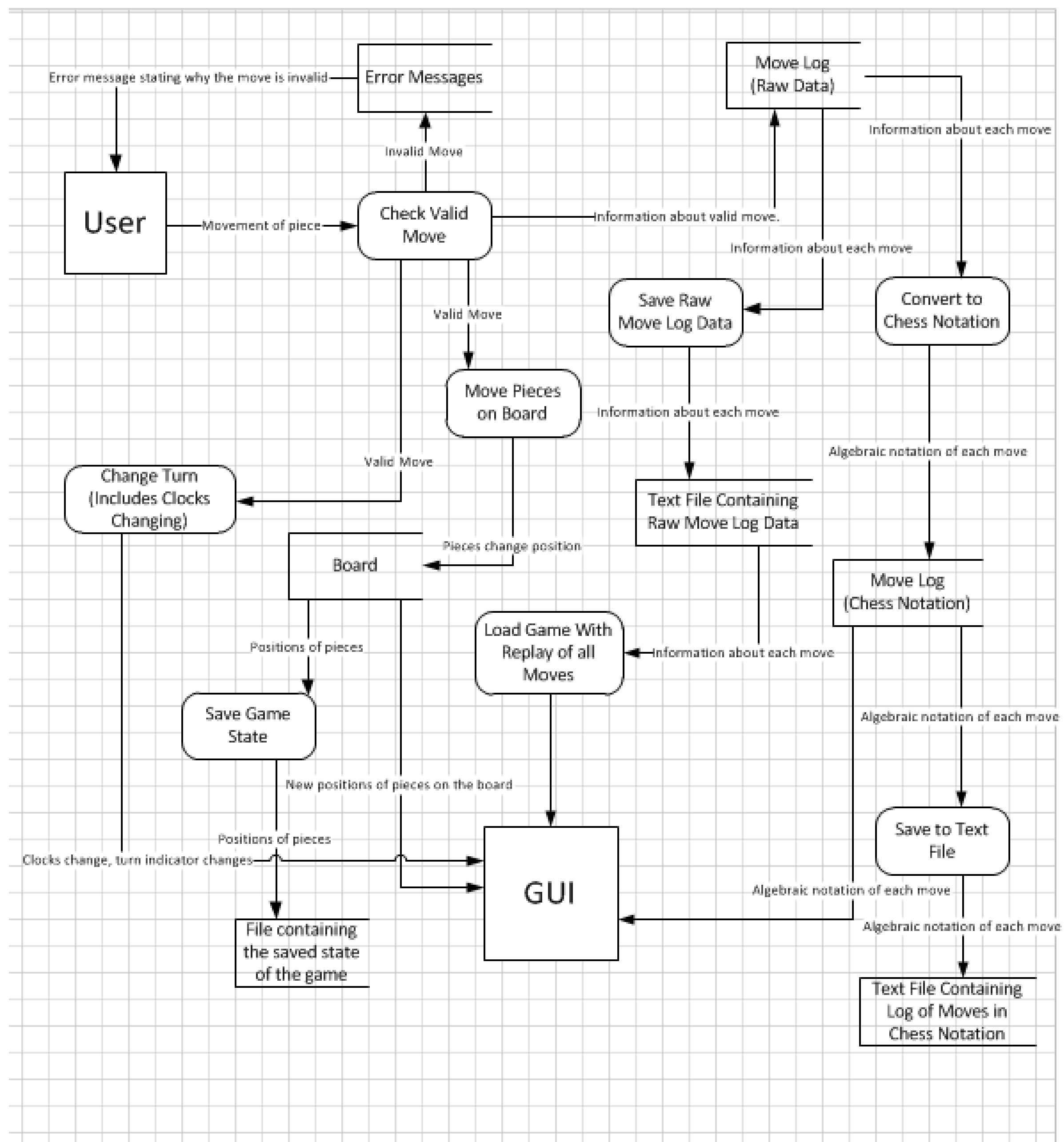


Proposed System

Level 0



Level 1



Objectives for the proposed system

- There must be a visual interactive 2-D grid, which shows the position of the pieces, and allows the player to make moves by clicking on them
- When a piece is clicked, all possible moves should be highlighted on the board

- The program must obey the standard rules of chess, and must not allow players to disobey these rules
- If a player inputs an invalid move, the pieces will stay in the same position on the board
- The program must include the special rules of Castling, En Passant and Pawn Promotion
- The program must include checks for whether a player is in check or checkmate, and notify the user when either of these happen.
- There should be a log of all the moves made in the game, with an option to save the log to a text file
- There must be a reset button to return the board to its starting state
- There should be an indicator to show whose turn it is
- The grid spaces on the chess board should be numbered on the vertical axis, and lettered on the horizontal axis
- The program must incorporate a chess clock, giving each player a certain amount of time to make their moves
- The program should include a pause button, which will stop the clocks to allow the players to have a break
- There must be an option to decide whether the game will be timed, and if so how much time each player will have, as well as how the game will be timed

Realistic appraisal of the feasibility of potential solutions

A manual solution to this problem would be to have a third person, other than the two playing the game, to record the moves in the game. This is impractical, as there has to be a third person, and there might not always be someone available to fulfil this requirement. The third person might also make a mistake with recording the moves in the game. Also there is not a way to automatically move the clock to the next turn with this solution.

There is no clear solution involving generic software to solve the problem. Any attempted solution would still involve manually inputting the moves into a log and manually changing the turn on the clocks.

There are a number of off-the-shelf chess programs available. Of the ones I looked at, some did have the function of having a chess clock, but none of them had the ability to create a record of all the moves played. These programs are quite cheap, but they would not be a full solution to the problem.

Bespoke software is the optimal choice, as it can fulfil the user’s needs entirely. It addresses all of the user’s requests, and creates a system from scratch in order to fulfil them. It is, however, the most expensive solution, as all the costs of development fall on the sole client.

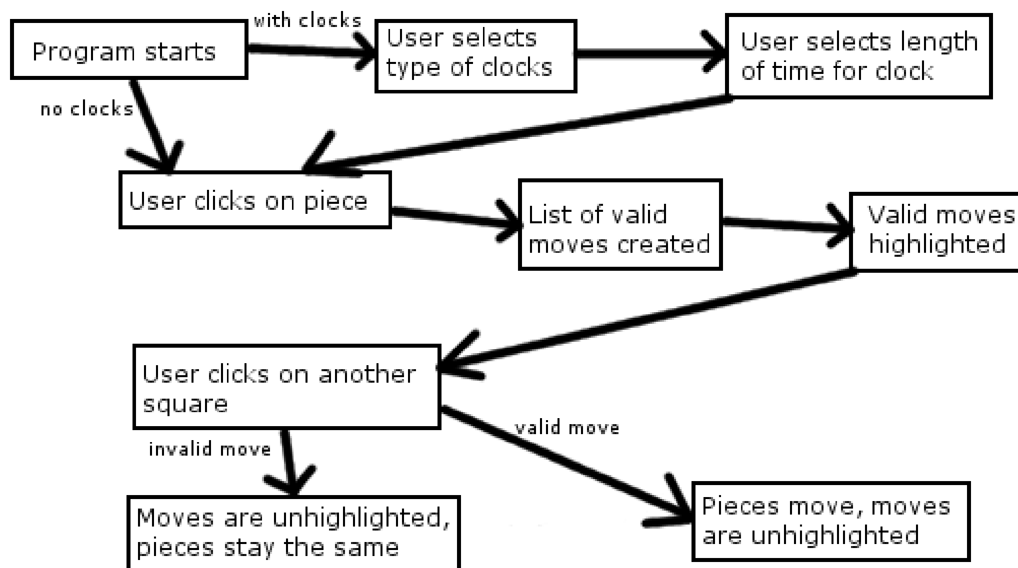
Justification of chosen solution

I have chosen a bespoke software solution to meet to user's needs. The interview I had with my client suggested that the most effective solution would be a software solution, and from looking at the different possible solutions earlier, bespoke is certainly the optimal choice, as it is the only solution to completely fulfil the user's needs.

The solution will be programmed in Visual Basic.NET, in order to take advantage of its Object Oriented capabilities. Visual Studio also allows me to create a good graphical interface very easily, and has pre-built classes for many objects that I will be using for my interface.

Design

Flow Chart



Class Definitions

- Piece
  - This class will contain general information used by any instance of a piece
  - Properties:
    - IsWhite (Boolean) - This will indicate what colour the piece is. A “true” value means the piece is white, and a “false value means the piece is black.
    - PositionX (Integer) - The X coordinate of the position of the piece
    - PositionY (Integer) - The Y coordinate of the position of the piece
    - Active (Boolean) - This indicates whether or not the piece has been captured.
  - This will be the parent class to 6 other classes, one for each type of piece. Each of these classes will inherit the properties of the “Piece” class, and will have their own individual methods to generate a list of possible moves when a piece is clicked.

- Board
  - This will be responsible for drawing the positions of all the pieces on the board.
  - After each turn, it looks at the Move object created for the last turn, and changes the images on the screen based on the initial position of the piece, the end position and

the information of whether or not there was a special move, and which special move if there was one.
  - It will also be responsible for highlighting possible moves on the board.

- Game
  - This class will create all of the Piece objects
  - It will handle all functions not directly related to the on-screen board or checking valid moves for specific piece rules (e.g. moving pieces, initializing the game, etc.)

Algorithms Using Pseudocode

Global Variables

- Grid (8x8 String Array) - In each position of the array, the name of the Piece object that is situated in the corresponding square is stored. If there is no piece, there will be no value stored in that position.
- ValidCheck (8x8 Boolean Array) - Used when checking valid moves; any squares that are the destination of valid moves are marked as true, while any other squares are marked as false
- PieceClick (Boolean) - Used to check whether a square has been clicked
- X1, Y1 (Integer) - Used to indicate the coordinates of the initial square
- CastleWQMoved, CastleWKMoved, CastleBQMoved, CastleBKMoved (Boolean) - Used to indicate whether either of the pieces in a particular castle move have moved.
- CastleWQ, CastleWK, CastleBQ, CastleBK (Boolean) - Used to indicate whether a particular Castling move is valid.
- EnPassant (Boolean) – True when a pawn has moved forward two spaces the previous turn, meaning an En Passant move is possible.
- EPPosX, EPPosY (Boolean) – When a pawn moves forward two spaces, these variables store the destination position of that pawn.
- WhiteTimeStore, BlackTimeStore (Double) – These store how much time each player has left on the clocks.
- WhitePaused (Boolean) – Used when the clocks are paused to store which clock was running at the time.

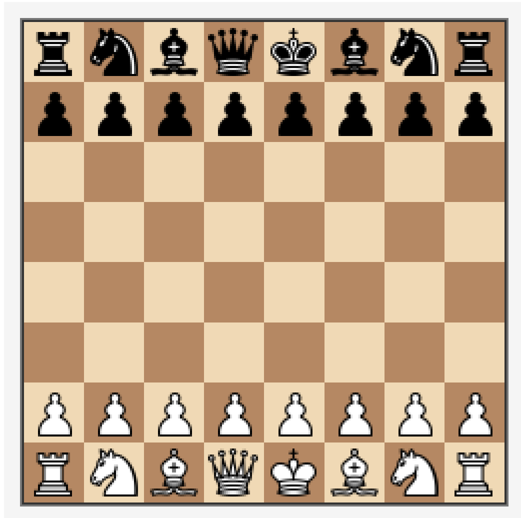
The Rules of Chess

I checked two sources to make sure that I had a reliable set of rules for chess. The sources are as follows:

- http://www.chess.com/learn-how-to-play-chess
- http://en.wikipedia.org/wiki/Rules_of_chess

The Initial Position of the Pieces

The image below shows how the board is initially laid out.



General Piece Rules

- Pieces cannot move through other pieces, with the exception of the Knight, which may 'leap' over other pieces.
- Most pieces can 'capture' an enemy piece (piece of the opposite colour) by landing on the same square as that enemy piece. The exception to this is the pawn, which has separate rules for moving and capturing, detailed below.
- A piece cannot capture a piece of the same colour.

The following algorithms will make sure these rules are followed, and will be implemented in the Piece parent class.

CheckSpaces (X1, Y1, X3, Y3, Valid)

This needs to check whether there are pieces in the spaces between the starting and ending positions. This will check what type of movement it is (diagonal, horizontal, etc.), and the number of spaces moved in that direction, and will use these to determine which spaces need to be checked. If the ending position is one space away from the starting position in the corresponding direction, then this check will be skipped, as there are no spaces between that need to be checked.

```
These values of X3 and Y3 indicate that the piece is only moving one space, and
therefore there will be no spaces between to check
        If X3 = 1 or 0 and Y3 = 1 or 0 then
                Valid = True
        Else
If the absolute values of X3 and Y3 are the same, the movement is diagonal. This
accounts for the four different directions that diagonal movement can take.
                If Abs(X3) = Abs(Y3) then
                        If X3 > 0 and Y3 > 0
                                For i = 1 to X3 - 1
                                        If Grid(X1+i, Y1+i) contains piece
                                                Valid = False
                        ElseIf X3 > 0 and Y3 < 0
```

```
                For i = 1 to X3 - 1
                    If Grid(X1+i, Y1-i) contains piece
                        Valid = False
            ElseIf X3 < 0 and Y3 > 0
                For i = 1 to Y3 - 1
                    If Grid(X1-i, Y1+i) contains piece
                        Valid = False
            ElseIf X3 < 0 and Y3 < 0
                For i = 1 to Abs(X3) - 1
                    If Grid(X1-i, Y1-i) contains piece
                        Valid = False
If X3 = 0, the movement must be vertical, as there is no change in the horizontal
component.
        ElseIf Abs(X3) = 0 then
            If Y3 > 0
                For i = 1 to Y3 - 1
                    If Grid(X1, Y1+i) contains piece
                        Valid = False
            ElseIf Y3 < 0
                For i = 1 to Abs(Y3) - 1
                    If Grid(X1, Y1-i) contains piece
                        Valid = False
If Y3 = 0, the movement must be horizontal, as there is no change in the vertical
component.
        ElseIf Abs(Y3) = 0 then
            If X3 > 0
                For i = 1 to X3 - 1
                    If Grid(X1+i, Y1) contains piece
                        Valid = False
            ElseIf X3 < 0
                For i = 1 to Abs(X3) - 1
                    If Grid(X1-i, Y1) contains piece
                        Valid = False
```

CheckDestination (X2, Y2, Valid)

This subroutine checks the contents of the destination square. If the square contains the same colour piece as the piece that is moving, the move is invalid. If the square is empty or contains an enemy piece, the move is valid. This subroutine is used by the check functions for all the different pieces, except for the

```
If Grid(X2, Y2) contains same colour piece
    Valid = False
ElseIf Grid(X2, Y2) contains opposite colour piece
    Valid = True
Else
    Valid = True
```

CheckValidMoves

This subroutine will go through every square to check if it is a valid move for the piece.

```
For j = 0 to 7 'j' will represent the y coordinate of the piece being checked
       For i = 0 to 7 'i' will represent the x coordinate of the piece being
checked
              Call Rules (i, j) Rules is the subroutine for checking a valid move
for whichever piece has
                            been clicked, 'i' and 'j' are the values being put in
                     for X2 and Y2
              If the move is valid, the position on the ValidCheck array is marked
True.
              If Valid = True
                     ValidCheck(i, j) = True
              Else
                     ValidCheck(i, j) = False
       Next
Next
This loops so that all squares on the board are checked
```

Individual Piece Rules

Each piece has separate rules on how it can move, and so I will be using separate algorithms for each different type of piece in order to check whether a particular move is valid.

The algorithms for checking whether a move is valid will use the following variables:

X1 (Integer) = X coordinate starting position
X2 (Integer) = X coordinate end position
X3 (Integer) = Difference of X coordinates in starting and end position (X2 - X1)
Y1 (Integer) = Y coordinate starting position
Y2 (Integer) = Y coordinate end position
Y3 (Integer) = Difference of Y coordinates in starting and end position (Y2 - Y1)
Valid (Boolean) = Indicates whether the move is valid or not

The following algorithms are used only to *check* whether a given move is valid, and are not responsible for the movement of the pieces. When a piece is clicked, every possible move on the board will be checked using the appropriate algorithm, and a list of valid moves will be generated. These moves will be highlighted on the board for the user.

The algorithms for each different type of piece will be implemented into its own Piece child class (i.e. the rules for the Pawn will be put into the Pawn child class, etc.).

The Pawn

While the Pawn is in its initial position, having not moved so far during the game, it may move 2 spaces forward. Otherwise, the Pawn may move one space. The Pawn may not move backwards. If there is an enemy piece one square diagonally in front of the pawn, it may move into that square and

capture the enemy piece. The pawn may not move in this way otherwise, nor can it capture in any other way.

The Pawn also has two special moves that it can perform; Pawn promotion and En Passant. Pawn promotion happens when a pawn reaches a square on the opposite side of the board to where it started. Once it reaches that square, it is promoted to a Queen, a Rook, a Bishop or a Knight, depending on the choice of the user. En Passant is a special capture that can occur immediately after a pawn has moved two spaces from its starting position, and an enemy pawn is in a position where it could have taken the pawn, had it only moved one space. The enemy pawn can then capture that pawn, with the end positions the same as they would be if the first pawn only moved one space forward.

```
Property Promotion (Char) Indicates whether a piece is promoted, and if so what piece it has been promoted to
```

Rules (X2, Y2)

```
If Promotion = Nothing then
       If X3 = 0 and Y3 = 2 and Y1 = 2 then
              Valid = True 'Sets the Valid variable to True initially
              CALL CheckSpaces 'Since the piece is moving 2 spaces, the square between the starting and destination squares needs to be checked for a piece.
              CALL PawnMove 'Pawns have different rules for moving and attacking, so separate subroutines will be created to check the destination square for different types of movement.
       ElseIf X3 = 0 and Y3 = 1 then
              Valid = True
              CALL PawnMove
       ElseIf Abs(X3) = 1 and Y3 = 1 then 'Abs is the function to provide an absolute value of X3, so that a value of -1 will be given as 1.
              Valid = True
              Call PawnAttack 'As the rules for a pawn are different if they are attacking, a separate subroutine will be created to check the destination square for a pawn moving diagonally.
       Else Valid = False
ElseIf Promotion = "Q" then  A Promotion value of Q,means the Pawn has been promoted to a Queen
       CALL Queen Rules
ElseIf Promotion = "N" then Promoted to Knight
       CALL Knight Rules
ElseIf Promotion = "B" then Promoted to Bishop
       CALL Bishop Rules
ElseIf Promotion = "R" then Promoted to Rook
       CALL Rook Rules
RETURN Valid
```

Promotion (X, Y)

This will be used by the Game class after a pawn has been moved, to check whether that pawn has moved to the other end of the board. If it has, a dialogue box will appear, and the user will input which

piece he or she wants to promote to (Q, N, R or B). That value will then be assigned to the Promotion variable.

```
If (IsWhite = True And Y = 7) Or (IsWhite = False And Y = 0) Then
	Promotion = INPUT
	If Promotion is Not "Q", "N", "R" or "B" then
		OUTPUT "That is not a valid input"
		CALL Promotion If the input is not valid, another input is requested
```

PawnMove (X2, Y2, Valid)

```
If Grid(X2, Y2) contains any piece
	Valid = False
Else
	Valid = True
```

PawnAttack (X2, Y2, Valid)

```
If Grid(X2, Y2) contains opposite colour piece
	Valid = True
Else
	Valid = False
```

EnPassantCheck

If a pawn has moved two spaces forward the previous turn, this will check whether an En Passant move is available for the pawn being moved.

```
If EnPassant = True then
	If IsWhite = True then
		If Abs(EPPosX – PositionX) = 1 AND EPPosY = PositionY then
			ValidCheck(EPPosX, EPPosY + 1) = True
	ElseIf IsWhite = False then
		If Abs(EPPosX – PositionX) = 1 AND EPPosY = PositionY then
			ValidCheck(EPPosX, EPPosY - 1) = True
```

The Rook

The Rook can move any number of spaces horizontally or vertically.

Rules

```
If Abs(X3) > 0 and Y3 = 0
	Valid = True
	CALL CheckSpaces
	CALL CheckDestination
ElseIf X3 = 0 and Abs(Y3) > 0
	Valid = True
	CALL CheckSpaces
	CALL CheckDestination
```

```
Else
      Valid = False
```

The Bishop

The Bishop may move any number of spaces diagonally.

<u>*Rules*</u>

```
If Abs(X3) = Abs(Y3) and X3 Not Equal to 0
      Valid = True
      CALL CheckSpaces
      CALL CheckDestination
Else
      Valid = False
```

The Queen

The Queen may move any number of spaces horizontally, vertically or diagonally.

<u>*Rules*</u>

```
If Abs(X3) = Abs(Y3) and X3 Not Equal to 0
      Valid = True
      CALL CheckSpaces
      CALL CheckDestination
ElseIf Abs(X3) > 0 and Y3 = 0
      Valid = True
      CALL CheckSpaces
      CALL CheckDestination
ElseIf X3 = 0 and Abs(Y3) > 0
      Valid = True
      CALL CheckSpaces
      CALL CheckDestination
Else
      Valid = False
```

The King

The King may move one space in any direction around it. If it has not moved, and the rook closest to it has also not moved, a special move called ‘Castling’ may be performed. In this, the King moves two spaces towards the Rook, and the Rook moves towards the King, landing on the other side of the King. This can be done one Kingside or Queenside, as long as there are no pieces between the King and Rook, and neither the King nor the Rook have moved from their initial position.

<u>*Rules*</u>

```
If (Abs(X3) = 1 or X3 = 0) AND (Abs(Y3) = 1 or Y3 = 0) AND NOT(X3 = 0 and Y3 = 0)
      Valid = True
      CALL CheckDestination
Else
```

```
      Valid = False
```

CastleCheck

```
If IsWhite = True then
      If CastleWKMoved = False AND Grid(5, 0) = Nothing AND Grid(6, 0) = Nothing
then
            CastleWK = True
            ValidCheck(6, 0) = True
      If CastleWQMoved = False AND Grid(1, 0) = Nothing AND Grid(2, 0) = Nothing
AND_
      Grid(3, 0) = Nothing then
            CastleWQ = True
            ValidCheck(2, 0) = True
If IsWhite = False then
      If CastleBKMoved = False AND Grid(5, 7) = Nothing AND Grid(6, 7) = Nothing
then
            CastleBK = True
            ValidCheck(6, 7) = True
      If CastleBQMoved = False AND Grid(1, 7) = Nothing AND Grid(2, 7) = Nothing
AND_
      Grid(3, 7) = Nothing then
            CastleBQ = True
            ValidCheck(2, 7) = True
```

The Knight

The Knight can move in an 'L' shape, one space either horizontal or vertical, and two spaces in the other direction. It is the only piece which can jump over other pieces, and therefore does not need to go through the 'CheckSpaces' subroutine.

Rules

```
If Abs(X3) = 2 and Abs(Y3) = 1
      Valid = True
      CALL CheckDestination
Elseif Abs(X3) = 1 and Abs(Y3) = 2
      Valid = True
      CALL CheckDestination
Else
      Valid = False
```

The Game Class

The variables used specifically in the Game class are as follows:

- WhiteTurn (Boolean) - Indicates whose turn it is; True indicates White Turn, False therefore indicates Black Turn
- SpecialMove (String) - Indicates whether a special move has been made, and if so specifically which one
- TurnCount (Integer) - A counter for how many turns have gone through in the current game

The Game class will also create all Piece objects, as well as the Board object. How the Piece objects are named is as follows:

- “W” or “B” depending on the colour of the piece
- The name of the type of piece (e.g. “Pawn” or “Knight”)
- A number to uniquely identify the piece

For example, “WPawn5” or “BBishop2”.

Initializing the Game

When the game starts, all the piece properties and variables must be set to their initial values, the background colour of the squares must be set to their default values, the images of the pieces must be set to their initial positions and the movelog must be cleared. The following function will do this upon starting the program, and will also be used by the Reset Board button.

InitializeGame

```
Call InitializePieces
Call InitializeVariables
MoveLog = Nothing
Call RevertColour These two subroutines are part of the Board class
Call ResetPiecePositions
```

InitializePieces

This will go through every Piece object and set each property to its initial value. For most pieces this will include IsWhite, PositionX, PositionY and Active, but for Pawns the property Promotion is also there.

```
e.g.
WPawn1.IsWhite = True
WPawn1.PositionX = 0
WPawn1.PositionY = 0
WPawn1.Active = True
WPawn1.Promotion = “”
```

This will also initialize the values of all positions in the Grid array.

```
e.g.
Grid(0, 0) = "WRook1"
Grid(1, 0) = "WKnight1"

For j = 2 to 5
       For i = 0 to 7
              Grid(i, j) = Nothing
       Next
Next
```

InitializeVariables

```
If WhiteTurn = False then
       Call ChangeTurn
TurnCount = 0
CastleWKMoved = False
CastleWQMoved = False
CastleBKMoved = False
CastleBQMoved = False
```

Clicking on a Square

When a square is clicked, the handler needs to check the value on the Grid array that corresponds to this square. If that space is empty, nothing will happen. If there is a piece on the square, the object of that particular piece will be located, and the CheckValidMoves subroutine will be called.

SquareClick (X, Y)

```
If PieceClick = False PieceClick is a Boolean variable which indicates whether a
piece has been clicked
       If Grid(x, y) Not empty x and y being the corresponding coordinates of
whichever picture box has been clicked
              CALL CheckTurn
Else
       If ValidCheck(x, y) = True
              CALL MovePiece
              CALL ChangeTurn
       CALL RevertColour
       PieceClick = False
       FalsifySpecialMoves()
```

CheckTurn (X, Y)

Checks whether a piece can be clicked depending on which turn it is.

```
If WhiteTurn = True then
       If Grid(X, Y) starts with "W" then X and Y represent the coordinates of the
square that was clicked
              Call CallRules(X, Y)
              Call DisplayValidMoves
              PieceClick = True
```

```
ElseIf WhiteTurn = False then
	IF Grid(X, Y) starts with “B” then
		Call CallRules(X, Y)
		Call DisplayValidMoves
		PieceClick = True
```

ChangeTurn

Self-explanatory; simply switches the turn.

```
If WhiteTurn = True then
	WhiteTurn = False
	TurnIndicator = “Black Turn”
If WhiteTurn = False then
	WhiteTurn = True
	TurnIndicator = “White Turn”
```

CallRules (X, Y)

This subroutine looks at the corresponding value in the Grid array for the square that has been clicked, and calls the CheckValidMoves routine for whichever Piece object the value in the Grid array refers to.

```
e.g.
If Grid(X, Y) = “BPawn1" then
	Call BPawn1.CheckValidMoves
```

FalsifySpecialMoves

Resets special move checks so that they do not always appear as valid after they have been marked as valid once.

```
CastleWK = False
CastleWQ = False
CastleBK = False
CastleBQ = False
```

Moving the Pieces

Once the user has made a move that has been listed as valid by the above algorithms, the system needs to change values in memory. Certain values in the Grid array have to be changed, the PositionX and PositionY values of the Piece object need to be changed to the X and Y values of the destination square. In the case of castling, this has to be done for two pieces. If a piece has been captured in the move, the object for that piece needs to be destroyed. It must also be slightly altered for En Passant, as a piece is captured that is not in the destination position.

MovePiece (X, Y)

```
If CastleWK = True AND X = 6 AND Y = 0 then
	E1 IMAGE = NOTHING
```

```
        F1 IMAGE = WhiteRook
        G1 IMAGE = WhiteKing
        H1 IMAGE = NOTHING
        Grid(4, 0) = NOTHING
        Grid(5, 0) = “WRook2”
        Grid(6, 0) = “WKing”
        Grid(7, 0) = NOTHING
        ChangeCoordinates(5, 0)
        ChangeCoordinates(6, 0)
        SpecialMove = “CastleWK”
        RecordMove
ElseIf CastleWQ = True AND X = 2 AND Y = 0 then
        A1 IMAGE = NOTHING
        C1 IMAGE = WhiteKing
        D1 IMAGE = WhiteRook
        E1 IMAGE = NOTHING
        Grid(0, 0) = NOTHING
        Grid(2, 0) = “WKing”
        Grid(3, 0) = “WRook1”
        Grid(4, 0) = NOTHING
        ChangeCoordinates(2, 0)
        ChangeCoordinates(3, 0)
        SpecialMove = “CastleWQ”
        RecordMove
ElseIf CastleBK = True AND X = 6 AND Y = 7
        E8 IMAGE = NOTHING
        F8 IMAGE = BlackRook
        G8 IMAGE = BlackKing
        H8 IMAGE = NOTHING
        Grid(4, 7) = NOTHING
        Grid(5, 7) = “BRook2”
        Grid(6, 7) = “BKing”
        Grid(7, 7) = NOTHING
        ChangeCoordinates(5, 7)
        ChangeCoordinates(6, 7)
        SpecialMove = “CastleBK”
        RecordMove
ElseIf CastleBQ = True AND X = 2 AND Y = 7
        A8 IMAGE = NOTHING
        C8 IMAGE = BlackKing
        D8 IMAGE = BlackRook
        E8 IMAGE = NOTHING
        Grid(0, 7) = NOTHING
        Grid(2, 7) = “BKing”
        Grid(3, 7) = “BRook1”
        Grid(4, 7) = NOTHING
        ChangeCoordinates(2, 7)
        ChangeCoordinates(3, 7)
        SpecialMove = “CastleBQ”
        RecordMove
ElseIf EnPassant = True AND X = EPPosX And Abs(EPPosY – Y) = 1 AND X1 NOT X then
```

```
        RecordMove
        ChangeActive(EPPosX, EPPosY)
        EnPassantImageChange
        Grid(X, Y) = Grid(X1, Y1)
        Grid(X1, Y1) = NOTHING
        Grid(EPPosX, EPPosY) = NOTHING
        ChangeCoordinates(X, Y)
Else
        RecordMove
        CastleNull
        If Grid(X, Y) NOT NOTHING then
                ChangeActive(X, Y)
        ImageChange(X, Y)
        Grid(X, Y) = Grid(X1, Y1)
        Grid(X1, Y1) = NOTHING
        ChangeCoordinates(X, Y)
PawnPromotion(X, Y)
EnPassant = False
If Grid(X, Y) FIRST LETTER = "P" then
        If X1 = X AND Abs(Y-Y1) = 2 then
                EnPassant = True
                EPPosX = X
                EPPosY = Y
If the King has been put into check, checkmate is then checked for.
If CheckCheck(NOT WhiteTurn) = True then
        CheckMateCheck
```

ChangeCoordinates (X, Y)

This sub is used to change the position values of the Piece objects.

```
e.g.
If Grid(X, Y) = "BPawn1" then
BPawn1.PositionX = X
BPawn1.PositionY = Y

Etc.
```

CastleNull

This sub marks a castling move as invalid if either of the pieces involved in the castle move.

```
If Grid(X1, Y1) = "WRook1" then
        CastleWQMoved = True
If Grid(X1, Y1) = "WKing" then
        CastleWQMoved = True
        CastleWKMoved = True
If Grid(X1, Y1) = "WRook2"
        CastleWKMoved = True
If Grid(X1, Y1) = "BRook1"
        CastleBQMoved = True
```

```
If Grid(X1, Y1) = "BKing"
       CastleBQMoved = True
       CastleBKMoved = True
If Grid(X1, Y1) = "BRook2"
       CastleBKMoved = True
```

ChangeActive (X, Y)

This sub switches the “Active” Boolean variable from true to false or vice-versa. It is used when a piece is captured, or when checking whether a move puts the player’s own king in check.

```
e.g.
If Grid(X, Y) = “BPawn1” then
       BPawn1.Active = NOT BPawn1.Active
ElseIf Grid(X, Y) = “BPawn2” then
       BPawn2.Active = NOT BPawn1.Active
Etc.
```

PawnPromotion (X, Y)

If the piece being moved is a pawn and that pawn has not already been promoted, the corresponding object's "Promote" subroutine is called, and then if the pawn is promoted, the image is changed

```
If Grid(X, Y) 2nd character = “P” then
       If Grid(X, Y) = “BPawn1” then
              If BPawn1.Promotion = NOTHING then
                     BPawn1.Promote
              PromotedPawnImageChange(X, Y, BPawn1.Promotion, BPawn1.IsWhite)
       Else If Grid(X, Y) = “BPawn2” then
              If BPawn2.Promotion = NOTHING then
                     BPawn2.Promote
              PromotedPawnImageChange(X, Y, BPawn2.Promotion, BPawn2.IsWhite)
Etc.
```

Recording the Moves

RecordMove (X, Y)

This sub creates a string of the chess notation of the move that was just made, and adds it to the Move Log.

```
ChessNotation As String = NOTHING
If SpecialMove = “CastleWK” OR “CastleBK” then
       ChessNotation = “O-O”
ElseIf SpecialMove = “CastleWQ” or “CastleBQ” then
       ChessNotation = “O-O-O”
Else
       If Grid(X1, Y1) 2nd and 3rd characters = “Kn” then
              ChessNotation = “N”
       Else
```

```
            ChessNotation = Grid(X1, Y1) 2nd character
        If Grid(X, Y) NOT NOTHING then
            ChessNotation = ChessNotation + "x"
        ChessNotation = ChessNotation + NumberToLetter(X) + (Y + 1)ToString

If WhiteTurn = True then
        TurnCount += 1
        MoveLogTEXT = MoveLogTEXT & vbNewLine & TurnCount & "."
MoveLogTEXT = MoveLogTEXT & " " & ChessNotation
```

Checking for Check and Checkmate

Check is when a piece has been put into a position where it can capture the enemy King. When this happens, a move must be made to get the King out of check. A player cannot make any move to put their own King in Check.

When a King is put into Check, and the player cannot make any move to get the King out of Check, Checkmate is reached.

CheckCheck (KingColourWhite As Boolean)

This checks whether a move puts a King in check, and returns true or false.

```
X, Y As Integer
Check As Boolean
If KingColourWhite = True then
        X = WKing.PositionX
        Y = WKing.PositionY
        If BPawn1.Rules(X, Y) = True AND BPawn1.Active = True then
            Check = True
        Same for all Black pieces, except for King
Else
        X = BKing.PositionX
        Y = BKing.PositionY
        If WPawn1.Rules(X, Y) = True AND WPawn1.Active = True then
            Check = True
        Same for all White pieces, except for King
RETURN Check
```

PutSelfInCheckCheck (X, Y, IsWhite)

This checks whether a move will put the player's own King in check, deeming it invalid. Returns true or false.

```
IniPos, FinPos As String
Valid As Boolean
IniPos = Grid(X1, Y1)
FinPos = Grid(X, Y)
ChangeActive(X, Y)
Grid(X1, Y1) = NOTHING
Grid(X, Y) = IniPos
```

```
If IniPos = "WKing" then
	WKing.PositionX = X
	WKing.PositionY = Y
ElseIf IniPos = "BKing" Then
	BKing.PositionX = X
	BKing.PositionY = Y
Valid = NOT CheckCheck(IsWhite)
Grid(X1, Y1) = IniPos
Grid(X, Y) = FinPos
ChangeActive(X, Y)
If IniPos = "WKing" then
	WKing.PositionX = X1
	WKing.PositionY = Y1
ElseIf IniPos = "BKing" Then
	BKing.PositionX = X1
	BKing.PositionY = Y1
RETURN Valid
```

CheckMateCheck

This sub is called when a King is put in check. It checks every piece of the opposite colour to see if it can make a move that will leave the player's King out of check. If such a move can be made, CheckMate is marked as false.

```
CheckMate As Boolean
CheckMate = True
If WhiteTurn = True then
	If BPawn1.Active = True then
		CallRules(BPawn1.PositionX, BPawn1.PositionY)
		For j = 0 to 7
			For i = 0 to 7
				If ValidCheck(i, j) = True then
					CheckMate = False
			Next
		Next
	Same for all Black Pieces
ElseIf WhiteTurn = False then
	If WPawn1.Active = True then
		CallRules(WPawn1.PositionX, WPawn1.PositionY)
		For j = 0 to 7
			For i = 0 to 7
				If ValidCheck(i, j) = True then
					CheckMate = False
			Next
		Next
	Same for all White Pieces
```

The Board Class

Displaying the Valid Moves

Once a list of valid moves has been made, they will be highlighted on the screen. This will be done by replacing the background of the picture boxes, which are the basis of the graphical representation of the chess board. A yellow background will indicate any valid move, while any other squares will remain as their default colour (White or Black).

DisplayValidMoves

This sub checks every square to see if it has been marked as a valid move, and if it has, it is highlighted. The square that was clicked is indicated in green. If there is a Castle move valid, it is indicated in red.

```
For j = 0 to 7
	For i = 0 to 7
		If ValidCheck(i, j) = True then
			Highlight(i, j)
	Next
Next
IniPosition As String = NumberToLetter(X1)
IniPoisition = IniPosition + (Y1 + 1).TOSTRING
IniPosition.BackColor = GREEN
If CastleWK = True then
	G1.BackColor = RED
ElseIf CastleWQ = True then
	C1.BackColor = RED
ElseIf CastleBK = True then
	G8.BackColor = RED
ElseIF CastleBQ = True then
	C8.BackColor = RED
```

Highlight (X, Y)

Changes the background colour of a given square to yellow.

```
Str as String
Str = NumberToLetter(X)
Str = Str + (Y + 1).TOSTRING
Str.BackColor = YELLOW
```

NumberToLetter (X)

Converts an X coordinate into the corresponding letter for the name of a PictureBox,

```
Str as String
If X = 0 then
	Str = “A”
```

```
ElseIf X = 1 then
       Str = "B"
ElseIf X = 2 then
       Str = "C"
ElseIf X = 3 then
       Str = "D"
ElseIf X = 4 then
       Str = "E"
ElseIf X = 5 then
       Str = "F"
ElseIf X = 6 then
       Str = "G"
ElseIf X = 7 then
       Str = "H"
RETURN Str
```

Drawing the Pieces on the Board

ImageChange (X, Y)

Sets the images of the destination square to whatever was in the initial square then sets the image of the initial square to nothing.

```
DestSquare, IniSquare As String
DestSquare = NumberToLetter(X) + (Y + 1).TOSTRING
IniSquare = NumberToLetter(X1) + (Y1 + 1).TOSTRING
If Grid(X1, Y1).TRIMLASTCHAR = "WPawn" then
       DestSquare.Image = WhitePawn
ElseIf Grid(X1, Y1).TRIMLASTCHAR = "BPawn" then
       DestSquare.Image = BlackPawn
'Same for all types of pieces of both colours

IniSquare.Image = NOTHING
```

PromotedPawnImageChange (X, Y, Promote, IsWhite)

Used to change a pawn's image to whatever it has been promoted to, if it has been promoted.

```
Square as String
Square = NumberToLetter(X) & (Y + 1).TOSTRING
If IsWhite = True then
       If Promote = "Q" then
              Square.Image = WhiteQueen
       ElseIf Promote = "N" then
              Square.Image = WhiteKnight
       ElseIf Promote = "R" then
              Square.Image = WhiteRook
       ElseIf Promote = "B" then
              Square.Image = WhiteBishop
ElseIf IsWhite = False then
       If Promote = "Q" then
```

```
        Square.Image = BlackQueen
    ElseIf Promote = "N" then
        Square.Image = BlackKnight
    ElseIf Promote = "R" then
        Square.Image = BlackRook
    ElseIf Promote = "B" then
        Square.Image = BlackBishop
```

EnPassantImageChange (X, Y)

Image change when using En Passant. Usual ImageChange is called, and then the image of the pawn that is captured is removed.

```
ImageChange(X, Y)
Square As String = NumberToLetter(EPPosX) + (EPPosY + 1).TOSTRING
Square.Image = NOTHING
```

Un-Highlighting the Valid Moves

RevertColour

Changes the background colour of all squares to their default colours

```
Square as String
For j = 1 to 8
    For i = 0 to 7
        Square = NumberToLetter(i) + j.TOSTRING
        If (i MOD 2 = 1 AND j MOD 2 = 0) OR (i MOD 2 = 0 AND j MOD 2 = 1)
then
            Square.BACKCOLOR = GRAY
        ElseIf (i MOD 2 = 0 AND j MOD 2 = 0) OR (i MOD 2 = 1 AND j MOD 2 = 1)
then
```

Setting the images to their original positions

ResetPiecePositions

Sets the images of all PictureBoxes to their initial states.

```
Square as String
A1.Image = WhiteRook
B1.Image = WhiteKnight
Etc. for all piece's original positions

For j = 3 to 6
    For i = 0 to 7
        Square = NumberToLetter & j.TOSTRING
        Square.Image = NOTHING
    Next
```

```
Next
```

Timers

When the clocks are activated, they will count down for a given player on that player's turn. When the player makes a move, their timer will stop and the other player's will start. For the NormalClock setting, there will be an overall time for each player to make all of their moves. For the SpeedChess setting, the player's will have a fixed time in which to make each of their moves, and is reset upon the start of their turns. The interval between timer ticks will be 0.1 seconds.

Starting the clocks

StartClocksClick

If the NormalClock checkbox is checked, this sub takes the input from the textbox under said checkbox, and multiplies it by 60 to get a value in seconds, from the input being in minutes, and then starts the timer. If the SpeedChess checkbox is checked, it takes the input from the textbox under that checkbox, which will be the value in seconds, and then starts the timer.

```
If NormalClock is CHECKED then
	WhiteTimeStore = NormalClockInput * 60
	BlackTimeStore = NormalClockInput * 60
	WhiteTimer START
ElseIf SpeedChess is CHECKED then
	WhiteTimeStore = NormalClockInput
	BlackTimeStore = NormalClockInput
	WhiteTimer START
```

Handling Clock Ticks

WhiteTimerTick

```
WhiteTimeStore = WhiteTimeStore – 0.1
WhiteTime.Text = WhiteTimeStore / 60 & ":" & WhiteTimeStore MOD 60
If WhiteTimeStore < 0 then
	WhiteTimer STOP
	OUTPUT "Player White has run out of time!"
```

This will be exactly the same for the Black timer; all the "White"s will simply be replaced with "Black".

Pause

This will stop whichever clock is running, and store which clock that was. Then, when it is clicked again, that clock will resume

PauseClick

```
If ClockOff CHECKED = False Then
	If WhiteTimer ENABLED = True Then
		WhiteTimer STOP
		WhitePaused = True
```

```
        ElseIf BlackTimer ENABLED = True Then
                BlackTimer STOP
                WhitePaused = False
        Else
                If WhitePaused = True Then
                        WhiteTimer START
                Else
                        BlackTimer START
```

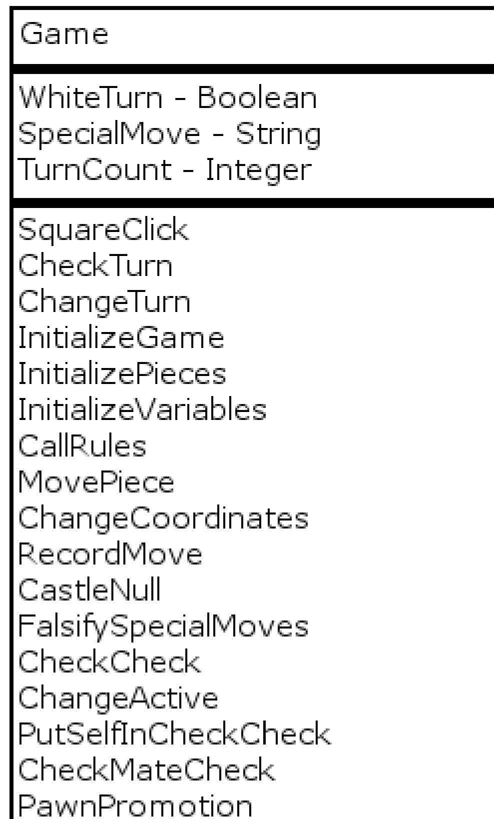
Saving the Text Log

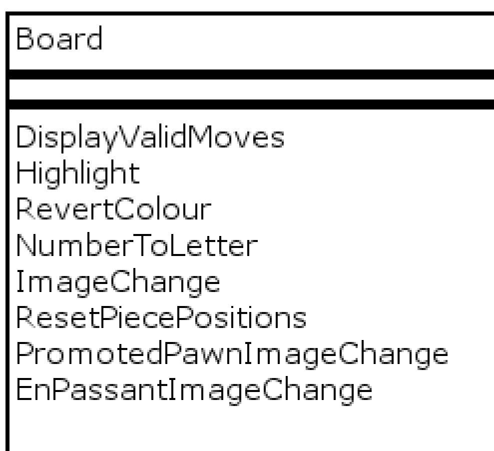
This will write to the C drive root directory, as that is a common directory that all PCs will have. The user will input a name for the file, and the method will write the file to the directory with that file name.

SaveLogClick

```
FileName as String
FileName = INPUT
FilePath as String
FilePath = "C:\" + FileName + ".txt"
CREATE FilePath
WRITE MoveLog.Text to FilePath
```

Class Diagrams





Piece
IsWhite - Boolean PositionX - Integer PositionY - Integer Active - Boolean
CheckSpaces CheckDestination CheckValidMoves

Pawn - Inherits Piece
Promotion - Char
Rules PawnMove PawnAttack Promote EnPassantCheck

King - Inherits Piece
Rules CastleCheck

Queen - Inherits Piece
Rules

Rook - Inherits Piece
Rules

Bishop - Inherits Piece
Rules

Knight - Inherits Piece
Rules

Hardware Specification

The program does not need a particularly fast processing speed to run, but it does need to be fast enough that it can run without any delays. The program installation will also occupy a very small amount of disc space, and the text logs that can be saved require very little space also. The recommended minimum specifications for my program are as follows:

- 5 MB disc space (for the program installations and any move logs that will be saved)
- A 2.5 GHz processor (this will be sufficient to ensure that the program will run with no delays)
- At least 1 GB of main memory (this will help ensure that the program runs smoothly)
- A Mouse (An input device used to play the game, click the buttons, etc.)
- A Keyboard (An input device used to input how much time the user wants to have on their clock, or which piece they want their Pawn to be promoted to, etc.)
- A Display Monitor

The desktop computer that resides in the room in which chess club occurs meets all of these specifications and therefore will be optimal for running the program.

Design Data Dictionary

As previously stated in the Analysis Data Dictionary, the program will not utilise much stored data. The data that will be stored in the installation of the program is only the images that will be displayed on the interface.

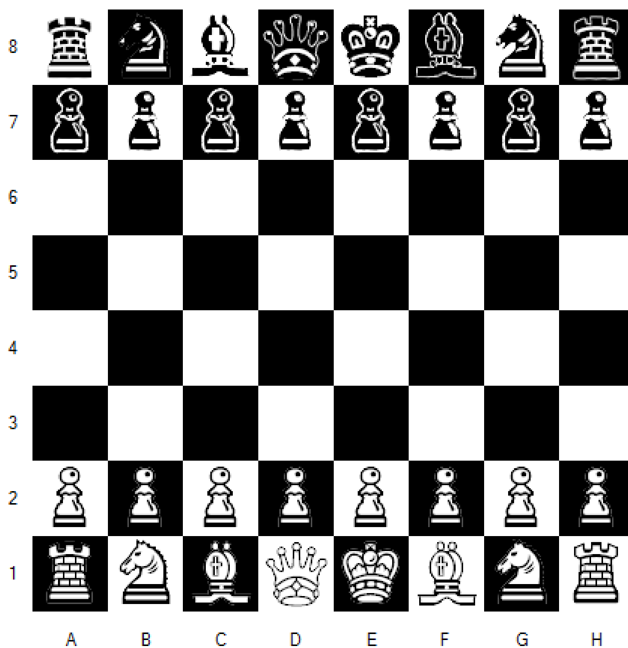
The program will be using 12 PNG images: two for each type of piece (one of each colour). These have transparent backgrounds so that the background colour of the PictureBoxes can be seen behind them, and are 50x50 pixels in size. They will be stored in the Resources folder.

Interface Design

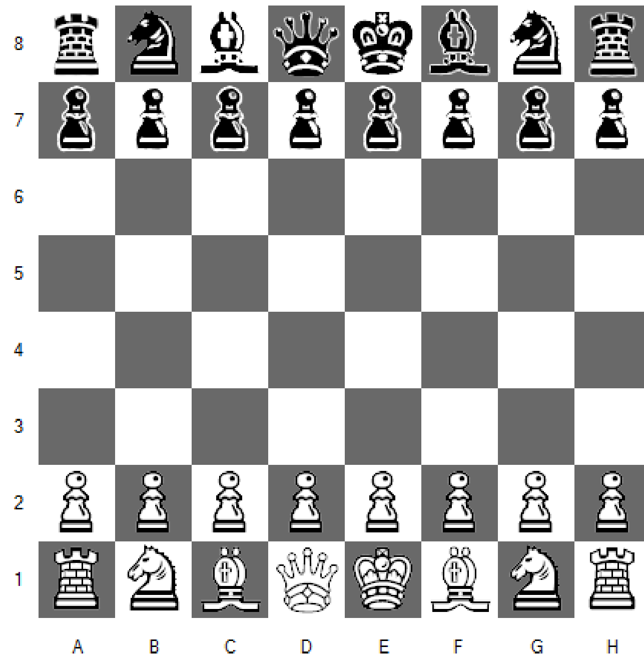
The interface for my chess program must have some key features:

- A 2-D grid which will act as the visual display for the main game. The pieces will be moved by interacting with this grid. The squares on this grid should be numbered on the vertical axis, and lettered on the horizontal axis.
- A button that resets the board to its initial state.
- An indicator to show which player's turn it currently is.
- A text log that records each move that each player makes.
- A button that allows the user to save the text log to a text file.
- Two clock indicators, one for each player, that show the remaining time for each player.
- An option to decide whether the game will be timed, and if so how it will be timed, and how much time each player gets.
- A button to pause and start the clocks in case the users must take a break.

Originally, the PictureBoxes had alternating black and white backgrounds.

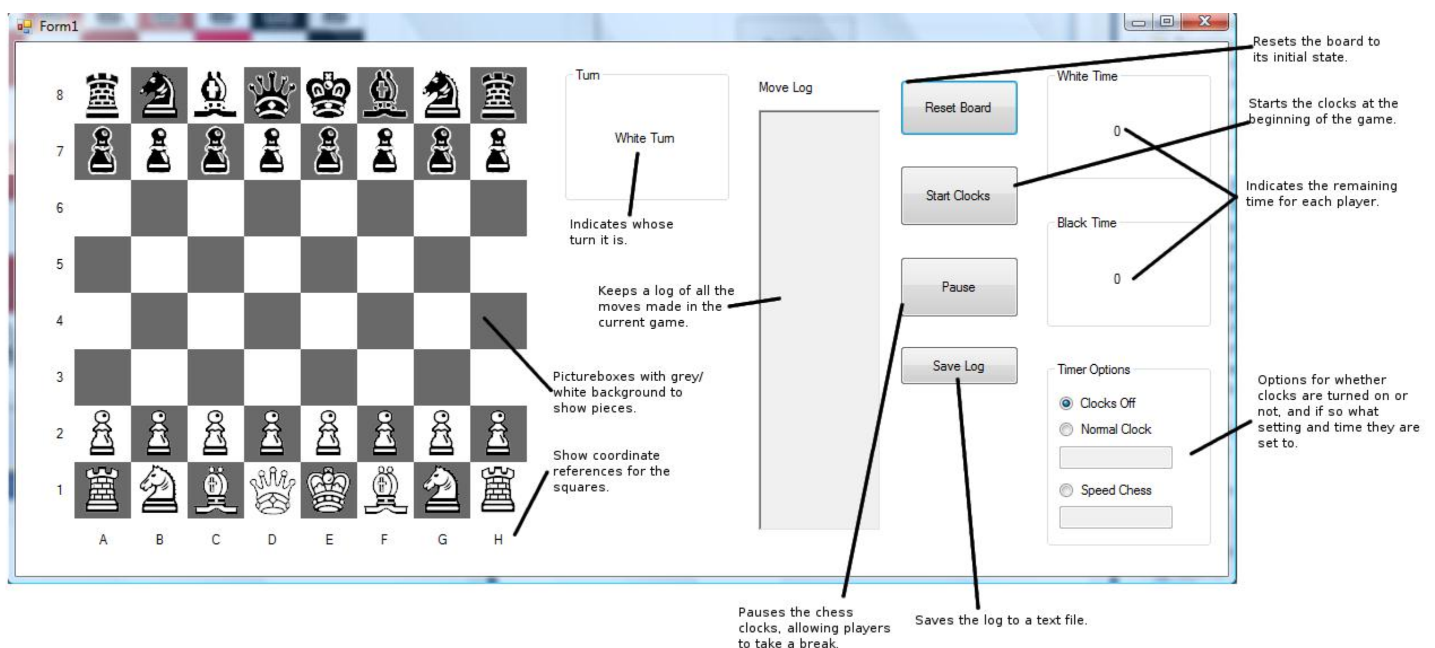


However, when I showed this to my end user, he requested that the black coloured squares be changed, as the black pieces do not show up very well on them. I changed the black colour to a light grey colour, and this is the result.



The black pieces are now distinct from the background colour of the PictureBoxes. I showed this to my end user, and he was satisfied with the result.

This is my final interface design.



Preliminary Test Plan

Test Area	Explanation
1	Check each piece's individual rules to see if they are followed correctly.
2	Check that each special move is functional.
3	Check that Check and Checkmate are working properly.
4	Check that the movelog is recording moves accurately, and that the save log function works.
5	Check that the timers are fully functional.

System Implementation and Maintenance

Most of the information necessary for system maintenance can be found in the annotations for the program code, included in the appendix. All variables, functions and sub-routines are explained in these.

System Overview

This system is used for two players to play chess against each other. All valid moves a piece can make are highlighted on-screen when a piece is clicked. The piece can then be moved by clicking on one of these highlighted squares. If a non-highlighted square is clicked, the pieces do not move, and the squares revert back to their previous colour. All the rules of chess are followed by the system. The system includes a move log, which contains the chess notation of all the moves made in the current game. This move log can be saved to a text file with a function included in the system. The system also includes chess clocks, which can be set to two different settings, and the amount of time given to each player is decided by the user.

Maintenance

Changing the Piece Images

All the images for the pieces have been placed within the Resources folder. These can be altered to better suit the tastes of the user, or replaced altogether. However, they must retain the filenames of the original images. They must also be .PNG images, with a transparent background, and a size of 50x50 pixels.

Changing the Background Colour of the PictureBoxes

In the RevertColour subroutine, which resides within the Board class, there are two lines which can be altered to change the background colour of the PictureBoxes. They are as follows:

```
Form1.Controls(Square).BackColor = Color.DimGray
Form1.Controls(Square).BackColor = Color.White
```

The "Color.DimGray" and "Color.White" can be changed to any of the wide range of colours that Visual Basic provides.

The colour that the program uses to highlight valid moves can also be changed in a similar fashion. The Highlight subroutine, which also resides in the Board class, contains the line:

```
Form1.Controls(str).BackColor = Color.Yellow
```

The "Color.Yellow" can also be changed to any colour that VB provides.

Progression from Design Stage and Problems during Implementation

The CheckValidMoves subroutine included in the Piece class was moved to each individual child class. This was because the method could not access the Rules method included in the child classes, as it was not a part of the parent class.
Many of the If statements used were replaced by Case statements where it was possible. This reduces the amount of processing used, and helps to tidy the code and make it easy to read.
Some Try and Catch statements were added for validation. These were added into the StartClocks_Click and Promote subs so that an invalid conversion would not cause an error. When this happens, the user is informed of that it is an invalid input and another input is requested.
In the Timer_Tick handlers, the amount that the TimeStore variables are decreased by was changed from 0.1 to WhiteTimer.Interval / 1000. This means that the interval can be changed without affected the timing of the clocks; they will still go down in seconds.

Bugs

During implementation, I was constantly running the program to check if what I had done so far worked. During the course of this, I encountered some bugs which I managed to fix by looking through my code for the reason the bug occurred. I have documented these bugs below

Problem: If the user selects their King when a Castle move is valid, the appropriate square is highlighted in red. However, after this, whenever a piece is clicked, the same square is highlighted in red again.

Reason: The variables used to mark a Castle move as valid remained true after the move has been made.

Solution: Add a subroutine to mark all the Castle moves back to false after each move has been made successfully.

Problem: Highlighted moves were appearing one space below where they should have.

Reason: Values in the array for any given square are one less than the values for the Pictureboxes. This had been compensated for with the X values, when the NumberToLetter function was used, but the Y values had not been corrected.

Solution: Incrementing the Y value from the array by one before using it to change the background colour for the on-screen squares.

Problem: Pieces that could move multiple spaces diagonally (i.e. Bishop and Queen) could move through other pieces.

Reason: The section in the CheckSpaces subroutine that validated diagonal moves was missing a "Valid = False" statement

Solution: The "Valid = False" statement was added.

Problem: When the King was selected, and a Castle move was marked as valid, if that King was deselected and another Piece was selected, the square for the valid Castle move was marked in red again. Then, if that piece can move into the Castle square, and it is done so, the Castle move is made instead of the intended one.

Reason: The Castle was only set to be false after a move had been made, and if a move has not been made, it stayed valid.

Solution: FalsifySpecialMoves was moved to the "If PieceClick = True" section of SquareClick.

Problem: When a pawn moved to the other side, the dialogue box for Pawn Promotion was not appearing.

Reason: The 'Promotion = ""' statements were coming out as false, despite the variable being set to "" earlier.

Solution: The statements were changed to 'Promotion = Nothing', which comes out as true.

Problem: After a pawn moved two spaces forward, if another piece moves to the space in front of it or behind it, the pawn is captured.

Reason: The program believes it to be an En Passant capture.

Solution: a Chess.Grid(Form1.X1, Form1.Y1).Substring(1, 1) = "P" requirement statement is added to En Passant captures, so that only pawns can perform this special move.

Testing

Outline Test Plan

Test Area	Explanation
1	Check each piece's individual rules to see if they are followed correctly.
2	Check that general rules are followed correctly.
3	Check that each special move is functional.
4	Check that the movelog is recording moves accurately, and the save log function works.
5	Check that the timers, buttons (apart from the Save Log button), labels and the timer options are fully functional.

Test Results

Test Area	Test No.	Description	Expected Result	Actual Result	Reference
1	1	Check Pawn movement from initial square.	Both squares in front are highlighted as valid, if unoccupied.	As expected.	Fig. 1-1
1	2	Check Pawn movement from non-initial square.	Square in front is highlighted as valid, if unoccupied	As expected.	Fig. 1-2
1	3	Check Pawn attack.	If a piece is directly in front of the pawn, it is not a valid move. If an enemy piece is diagonally in front, it is a valid move.	As expected	Fig. 1-3

1	4	Check Knight movement.	Unoccupied squares that are one square in the horizontal direction and two squares in the vertical direction away, and vice-versa, are valid if unoccupied. Unoccupied squares that are blocked by other pieces are valid also. Enemy pieces are capturable, friendly pieces are not.	As expected	Fig. 1-4
1	5	Check Bishop movement.	Unoccupied squares in any diagonal direction are valid. Enemy pieces are capturable, friendly pieces are not.	As expected	Fig. 1-5
1	6	Check Rook movement.	Unoccupied squares in any horizontal or vertical direction are valid. Enemy pieces are capturable, friendly pieces are not.	As expected	Fig. 1-6
1	7	Check Queen movement.	Unoccupied squares in any horizontal, vertical or diagonal direction are valid. Enemy pieces are capturable, friendly pieces are not.	As expected	Fig. 1-7
1	8	Check King movement.	All unoccupied adjacent squares are valid. Enemy pieces are capturable, friendly pieces are not.	As expected	Fig. 1-8
2	1	Check that a piece will not move if an invalid move is selected.	Pieces remain in their original position, move indicator does not change.	As expected	Fig. 2-1a Fig. 2-1b
2	2	Check that the turn indicator changes.	When a valid move is made, the turn indicator will change from white turn to black turn or vice-versa	As expected	Fig. 2-2
2	3	Check that a player cannot make a move on the other player's turn.	When a white piece is selected on the black player's turn, or vice-versa, there will be no change.	As expected	Fig. 2-3
2	4	Check that the check for Check is functioning correctly.	When a move is made that puts the opponent's King in check, a "+" is added to the move notation in the move log.	As expected	Fig. 2-4
2	5	Check that the check for checkmate is functioning correctly.	When a move is made that results in checkmate, a dialogue box appears with the text "Checkmate".	As expected	Fig. 2-5
3	1	Check that the White King-side Castle is functional.	If there are no pieces between the King and Rook, and neither the King nor the Rook have moved, a red square will appear two spaces towards the Rook from the King. When this is clicked, the King will move to this square, and the rook will move to the square on the other side of the King.	As expected	Fig. 3-1a Fig. 3-1b
3	2	Check that the White	If there are no pieces between	As expected	Fig. 3-2a

		Queen-side Castle is functional.	the King and Rook, and neither the King nor the Rook have moved, a red square will appear two spaces towards the Rook from the King. When this is clicked, the King will move to this square, and the rook will move to the square on the other side of the King.		Fig. 3-2b
3	3	Check that the Black King-side Castle is functional.	If there are no pieces between the King and Rook, and neither the King nor the Rook have moved, a red square will appear two spaces towards the Rook from the King. When this is clicked, the King will move to this square, and the rook will move to the square on the other side of the King.	As expected	Fig. 3-3a Fig. 3-3b
3	4	Check that the Black Queen-side Castle is functional.	If there are no pieces between the King and Rook, and neither the King nor the Rook have moved, a red square will appear two spaces towards the Rook from the King. When this is clicked, the King will move to this square, and the rook will move to the square on the other side of the King.	As expected	Fig. 3-4a Fig. 3-4b
3	5	Check that Pawn Promotion is functional.	When a pawn is moved to a square in the last row on the opposite side of the board, an input box appears requesting an input for what type of piece to promote the pawn to. Once this has been done successfully, the image of the pawn should change to the type of piece that the user selected. The rules of that piece also become that of the type of piece the user has selected.	As expected	Fig. 3-5a Fig. 3-5b
3	6	Check that the input box for Pawn Promotion functions correctly.	The user should be able to input “q”, “r”, “n” or “b” to pick which piece they would like the pawn to be promoted to. If the user inputs anything other than one of these letters, the program will request a valid input.	As expected	Fig. 3-6a Fig. 3-6b Fig. 3-5a Fig. 3-5b
3	7	Check that En Passant is functional.	When a pawn moves forward two spaces from its starting position, and an enemy pawn is at a position where it could capture that piece if it had moved a single square forward, that enemy pawn should be able to capture the pawn that moved.	As expected	Fig. 3-7a Fig. 3-7b Fig. 3-7c
4	1	Check normal movement without capture writes correctly.	The notation should be the a character denoting which type of piece moved, and then the coordinate position of the destination square.	As expected	Fig. 4-1
4	2	Check capture	The notation should be the same as 4.1, with an added “x”	As expected	Fig. 4-2

		movement writes correctly.	after the letter denoting the piece.		
4	3	Check castle movement writes correctly.	For king-side castles, the notation should be “O-O”, and for queen-side castles it should be “O-O-O”	As expected	Fig. 4-3
4	4	Check Pawn Promotion writes correctly.	The notation should be the same as 4.1, but with an added “=” as well as a character denoting the type of piece the pawn promoted to.	As expected	Fig. 3-5b
4	5	Check Check writes correctly.	For a move that puts the enemy king in check (but not checkmate), a “+” is added to the end to whatever move was made.	As expected	Fig. 2-4
4	6	Check Checkmate writes correctly.	A “#” should be added to the end of whatever move was made.	As expected	Fig. 4-6
4	7	Check that the Save Log function works correctly.	A filename will be requested from the user after the Save Log button is pressed. Once a filename has been given, a text file containing the log will be created in the C Drive root directory.	As expected.	Fig. 4-7a Fig. 4-7b
5	1	Check “Normal Clock” setting of timers functions correctly.	The number in the textbox, entered by the user, is the amount of minutes each player has. The timer for a given player will be active when it is that players turn, and will stop when it is the other player’s turn. When a timer runs out, a message will appear saying “Player X has run out of time!”. If an invalid input is put into the textbox, a message will appear to inform the user of this.	As expected	Fig. 5-1a Fig. 5-1b Fig. 5-1c Fig. 5-1d Fig. 5-1e
5	2	Check “Speed Chess” setting of timers functions correctly.	The number in the textbox, entered by the user, is the number of seconds each player has to make each move. When a move is made, the timer resets to whatever the input is. When a timer runs out, a message will appear saying “Player X has run out of time!”. If an invalid input is put into the textbox, a message will appear to inform the user of this.	As expected	Fig. 5-2a Fig. 5-2b Fig. 5-2c Fig. 5-2d Fig. 5-2e
5	3	Check “Pause” button functions correctly.	When the button is pressed, whichever timer is running will stop, and when the button is pressed again, that timer will start.	As expected	Fig. 5-3a Fig. 5-3b
5	4	Check “Reset Board” button functions correctly.	All pieces return to their original position, the Move Log is cleared, the turn indicator switches to White Turn if it was Black Turn, the timer options switch to Clocks Off.	As expected	Fig. 5-4a Fig. 5-4b

Test Appendix

Fig. 1-1

Pawn movement from the initial square.

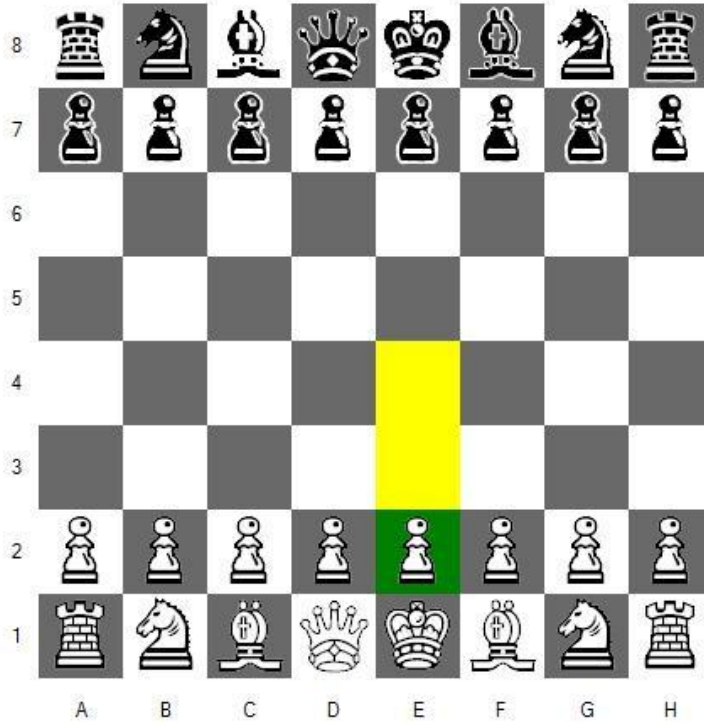


Fig. 1-2

Pawn movement from non-initial square

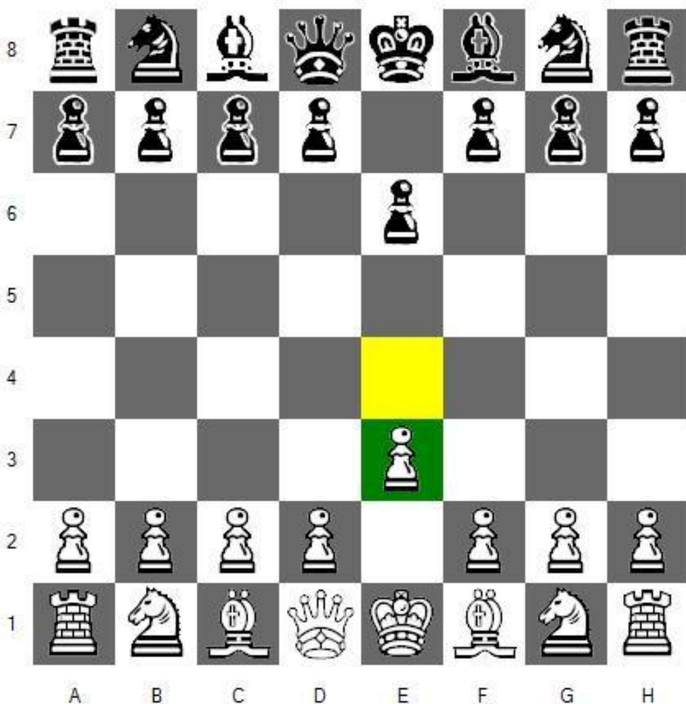


Fig. 1-3

Pawn Attack

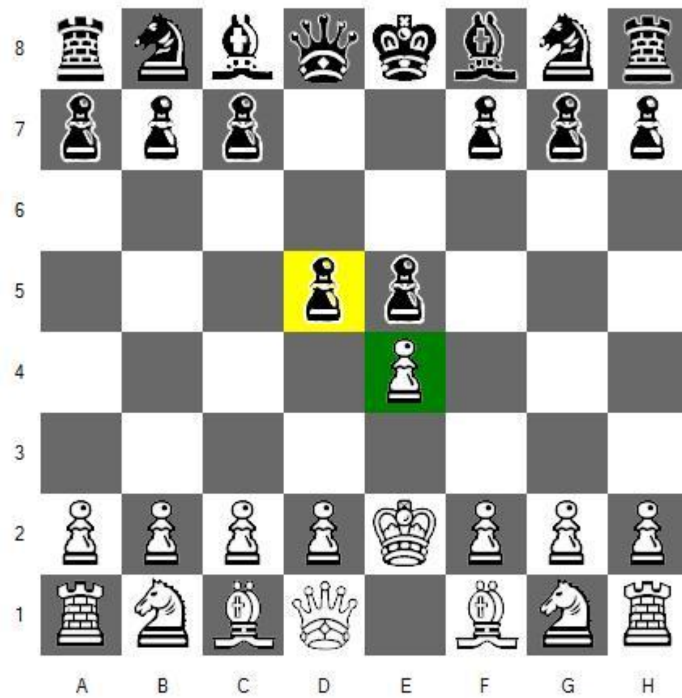


Fig. 1-4

Knight movement

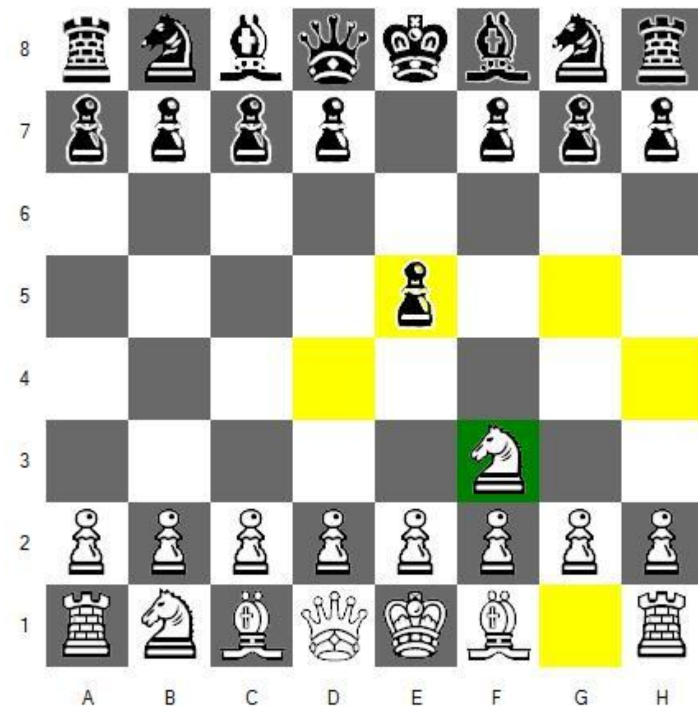


Fig. 1-5

Bishop movement

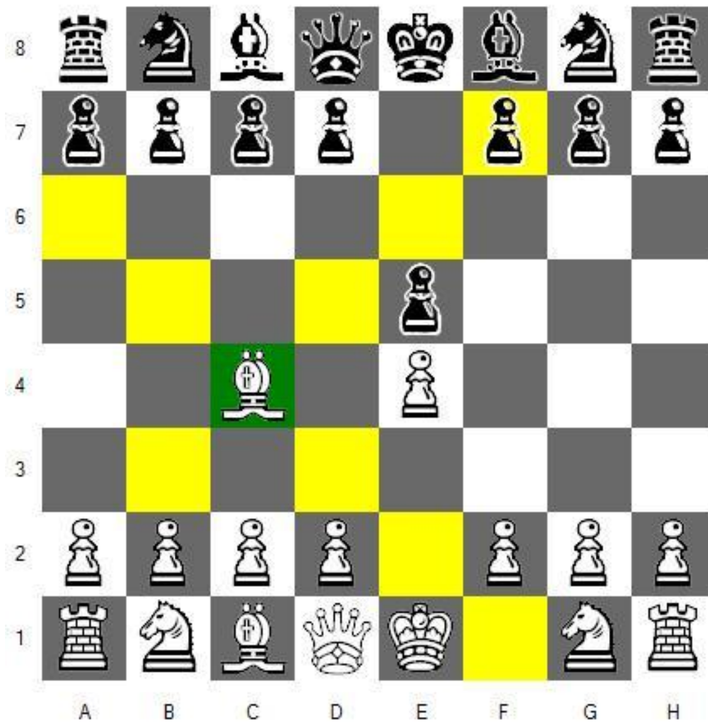


Fig. 1-6

Rook movement

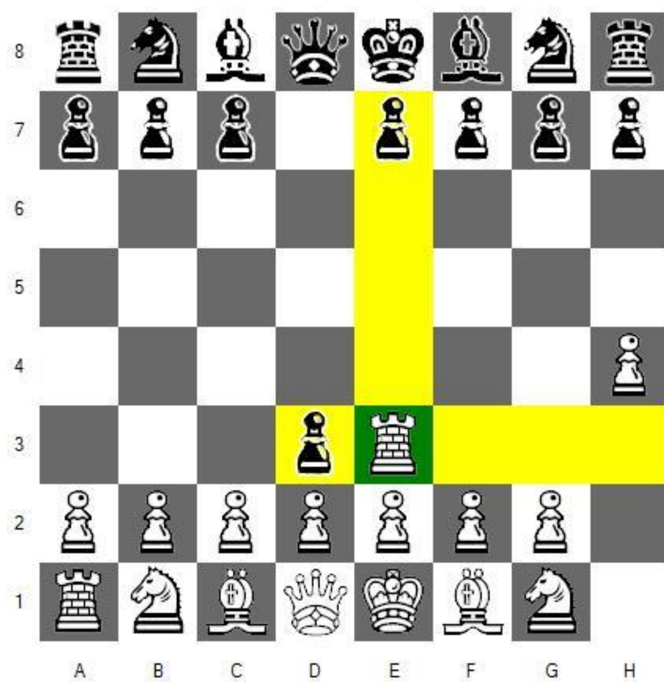


Fig. 1-7
Queen movement

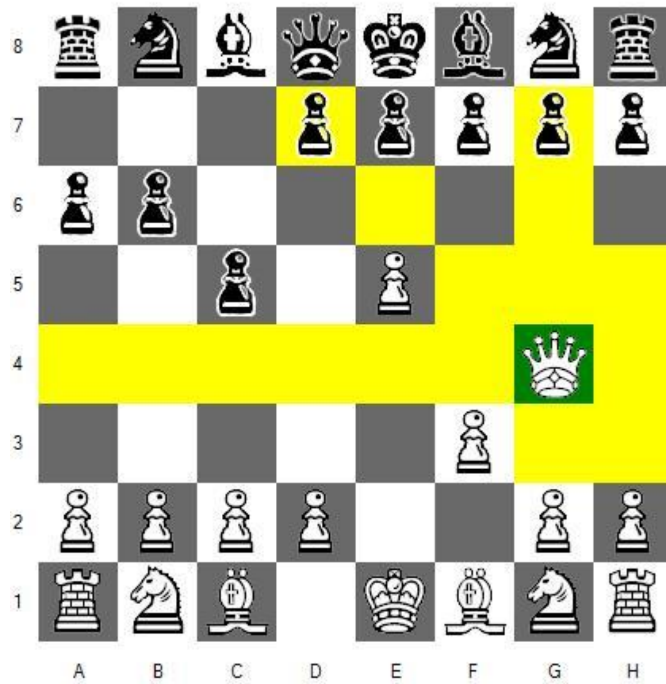


Fig. 1-8
King movement

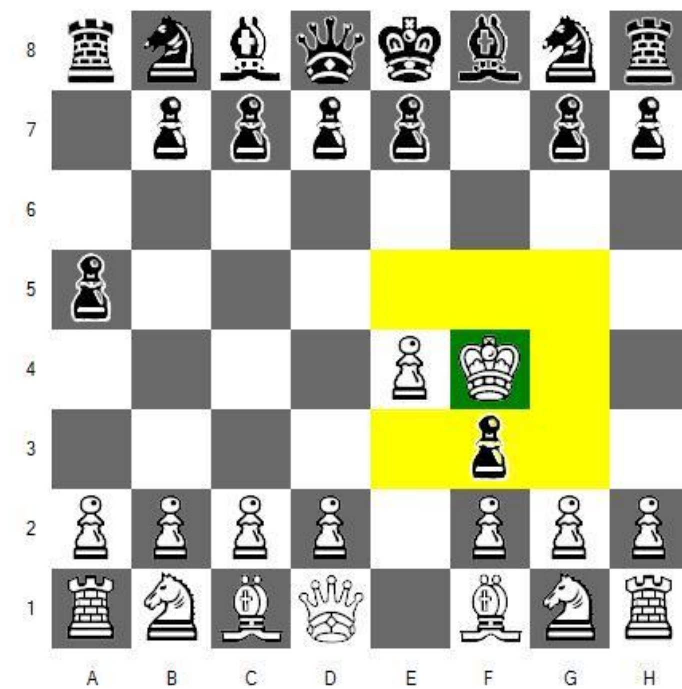


Fig. 2-1a

Before selecting an invalid move.

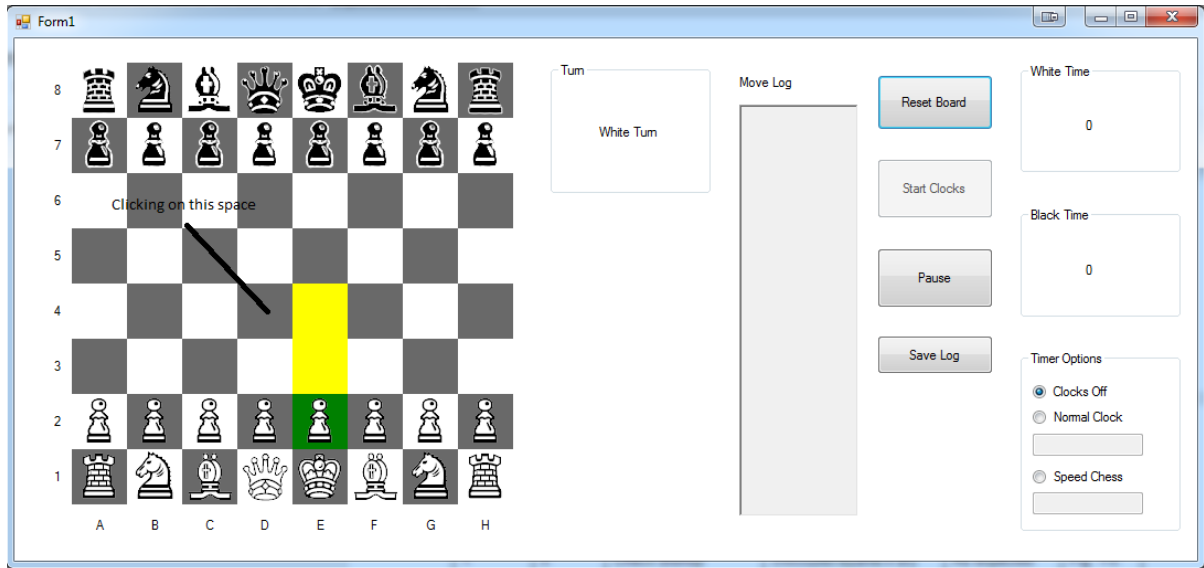


Fig. 2-1b

After selecting an invalid move.

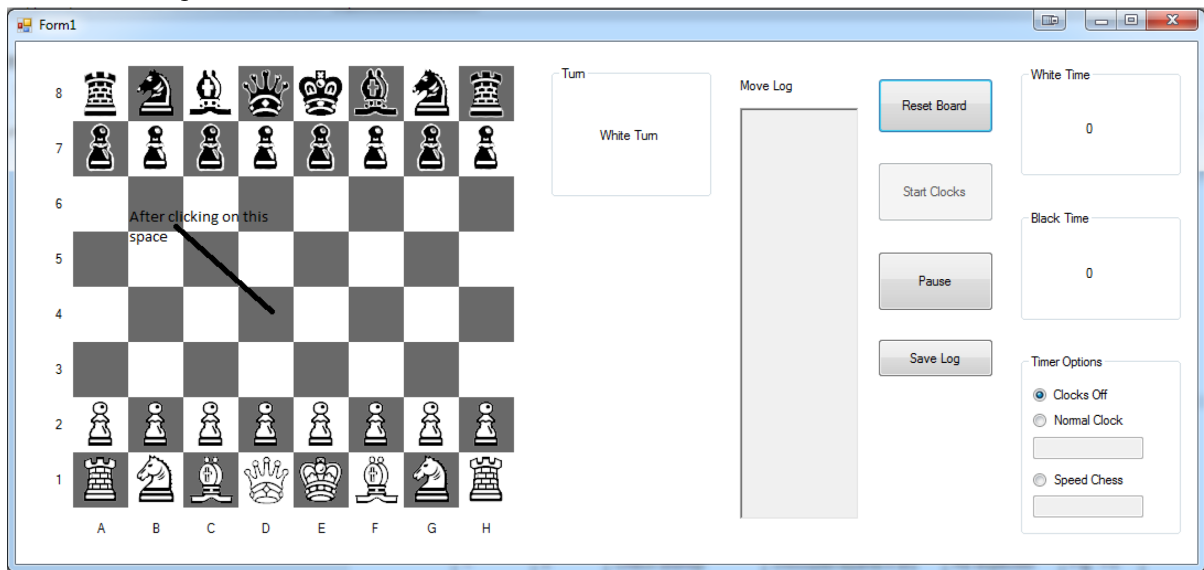


Fig. 2-2

Turn indicator changes to black turn after white player has made a move.

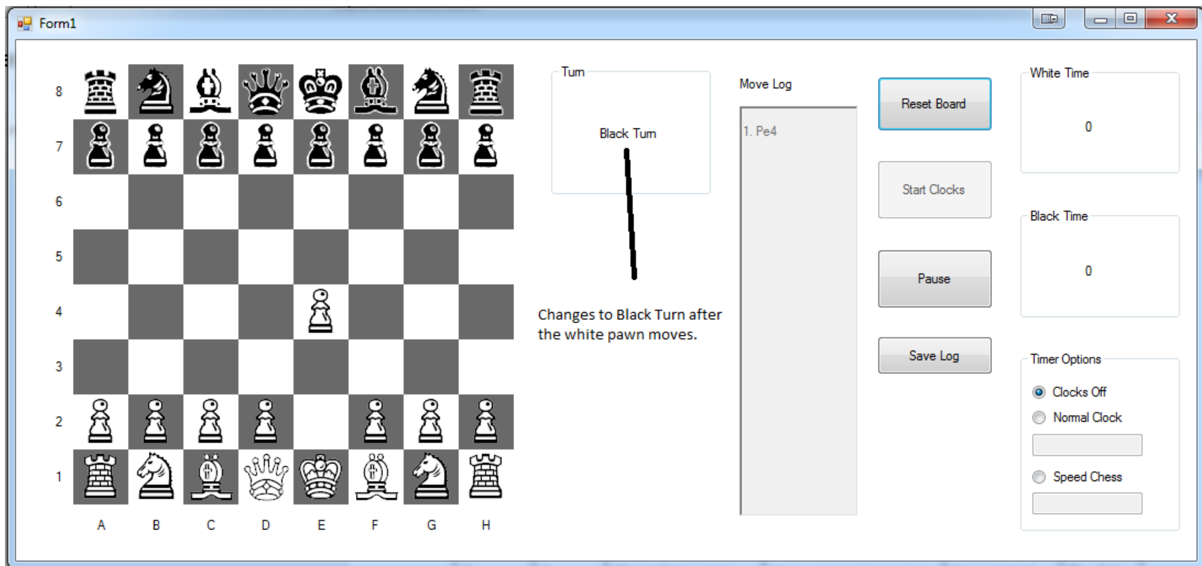
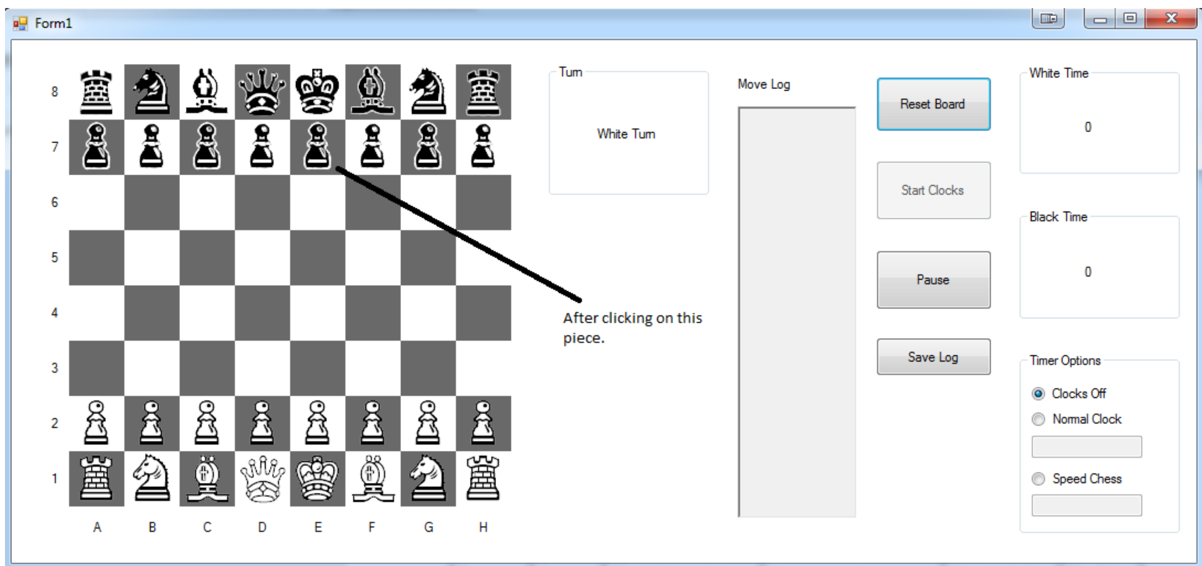


Fig. 2-3



After clicking on a Black piece on White player's turn.

Fig. 2-4

King has been put into check, the move that did so has a “+” added to the end in the Move Log.

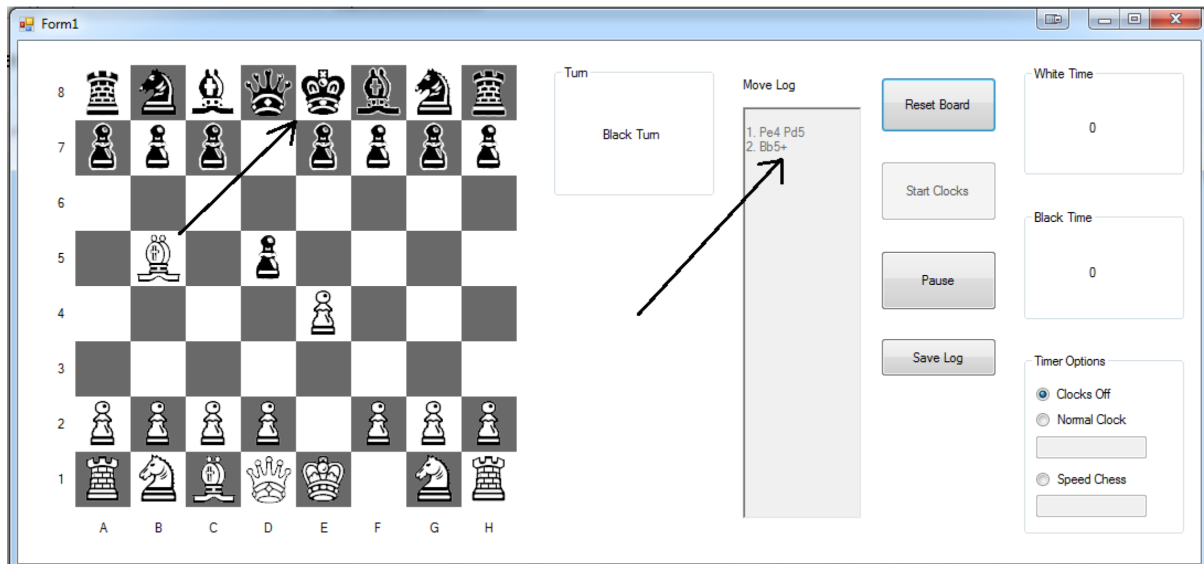


Fig. 2-5

Black player has been put into checkmate; a message box appears informing the players of this.

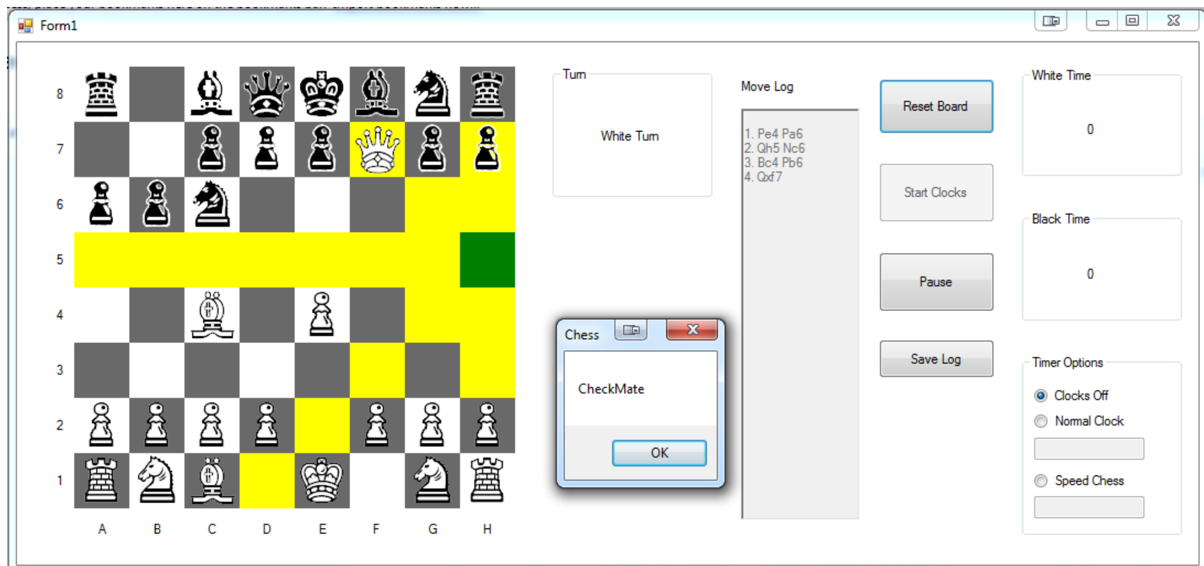


Fig. 3-1a

King has been clicked while castle is valid, shows the castle move as a red square.

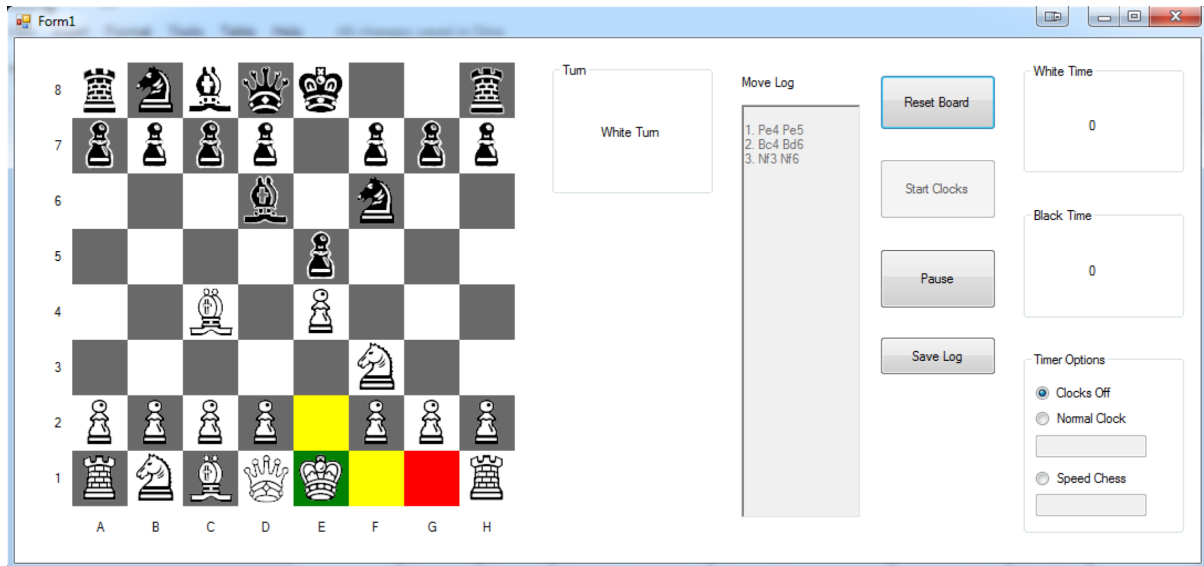


Fig. 3-1b

Castle move has been selected, King moves to the square that was highlighted in red, the castle moves to the other side of the King.

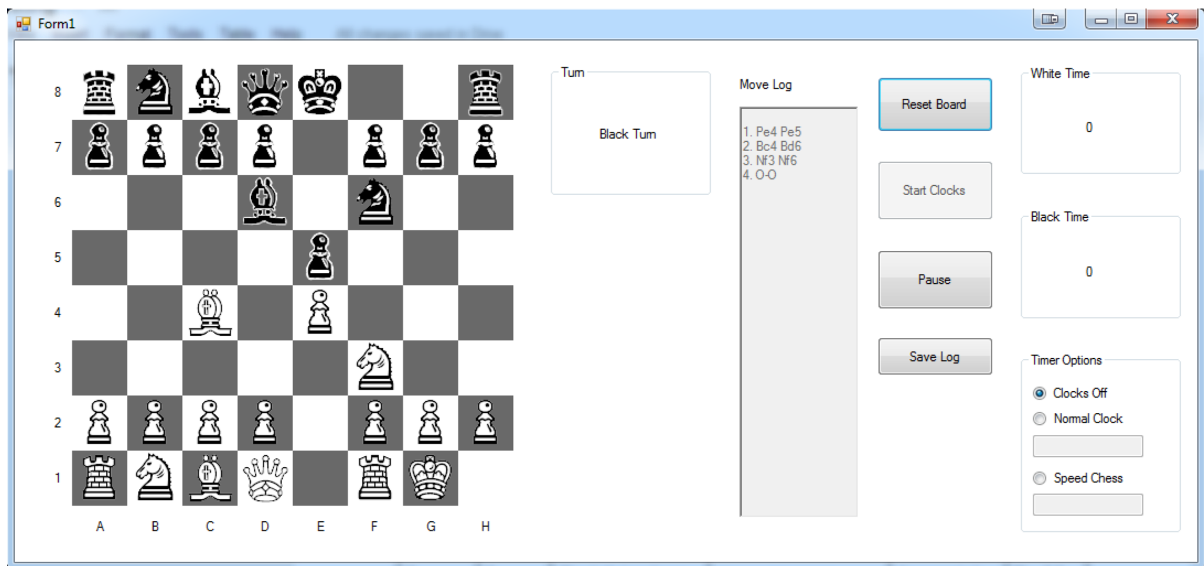


Fig. 3-2a

King has been clicked while castle is valid, shows the castle move as a red square.

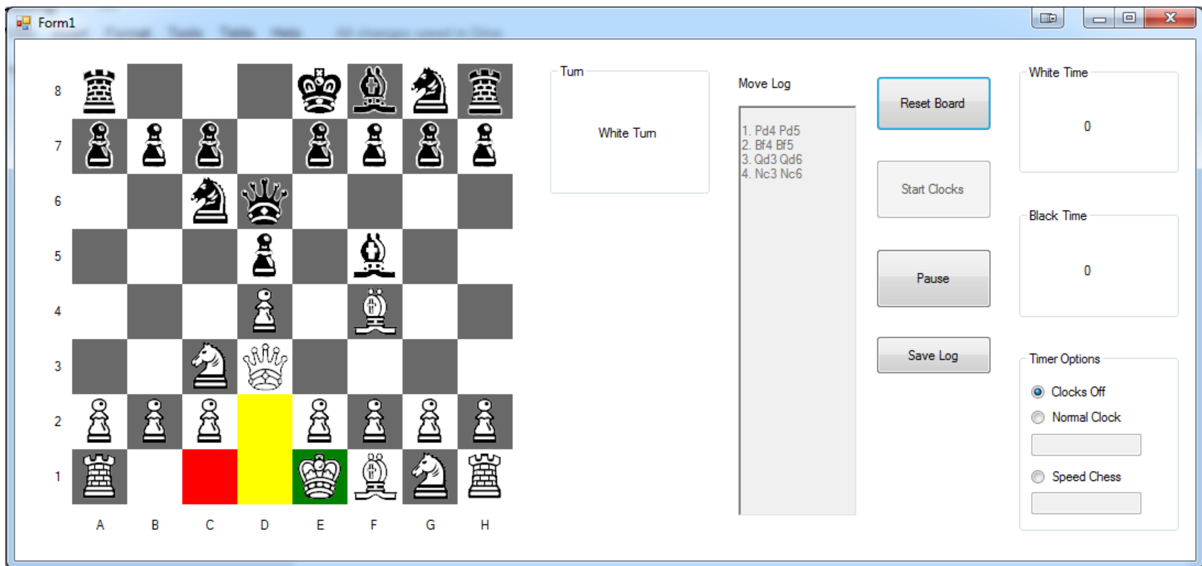


Fig. 3-2b

Castle move has been selected, King moves to the square that was highlighted in red, the castle moves to the other side of the King.

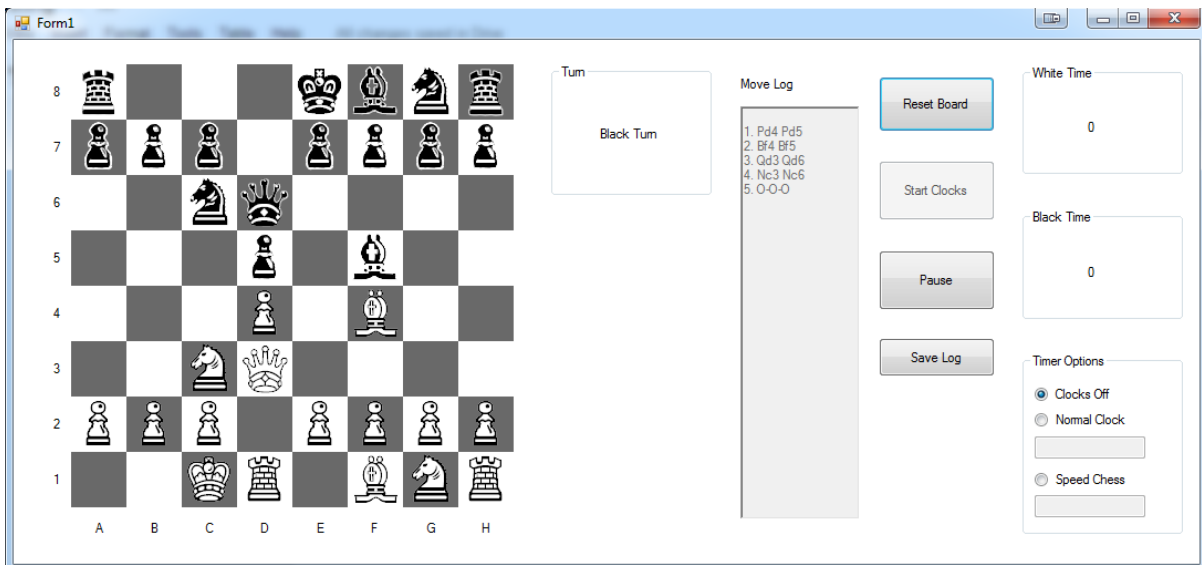


Fig. 3-3a

King has been clicked while castle is valid, shows the castle move as a red square.

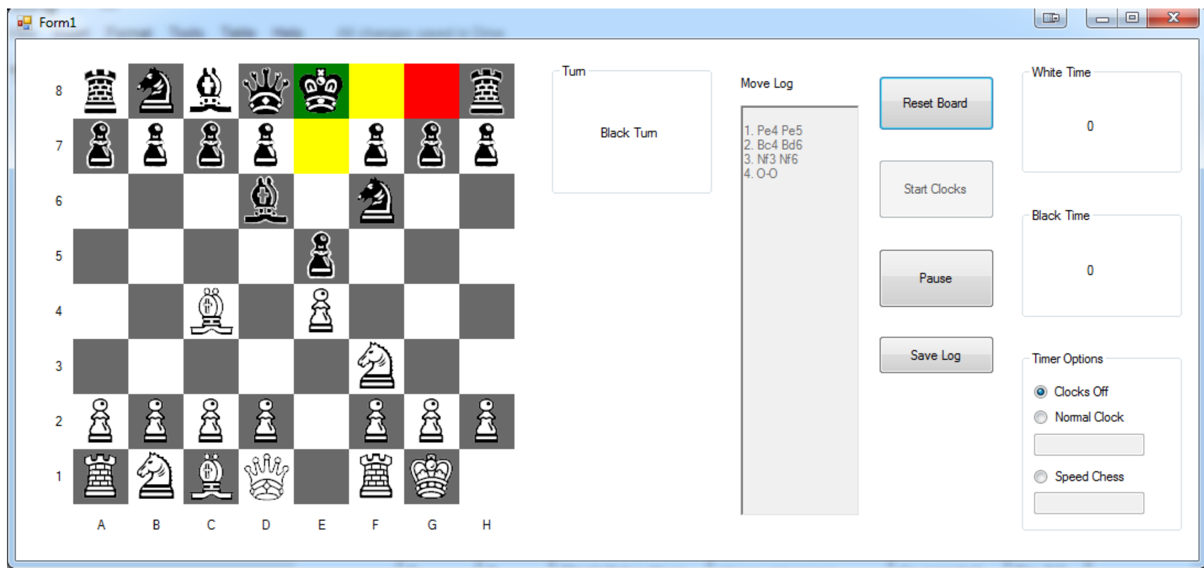
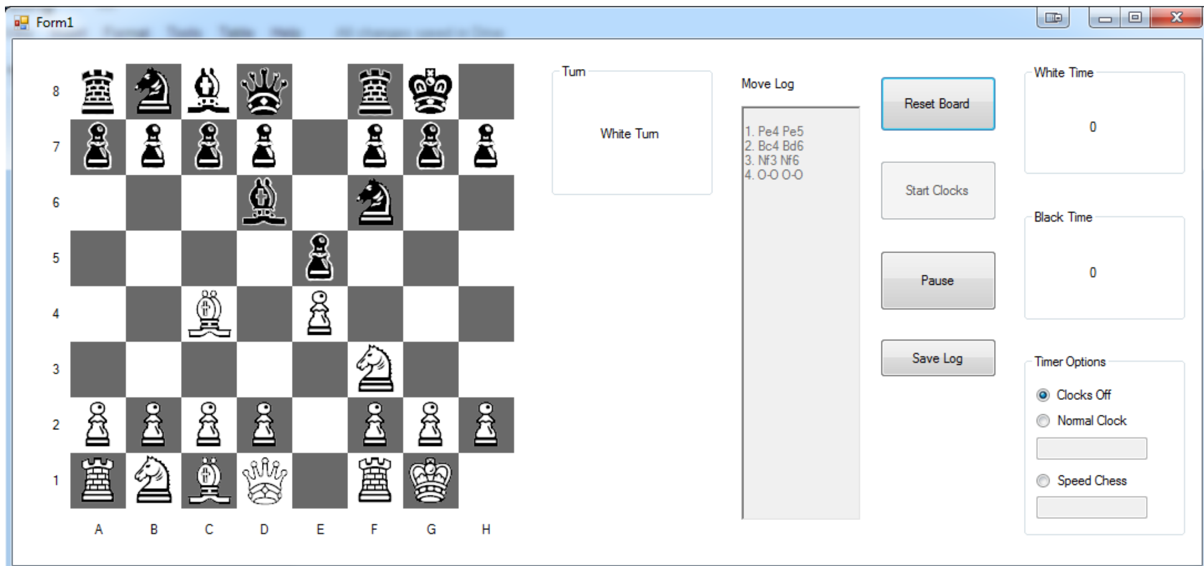


Fig. 3-3b

Castle move has been selected, King moves to the square that was highlighted in red, the castle



moves to the other side of the King.

Fig. 3-4a

King has been clicked while castle is valid, shows the castle move as a red square.

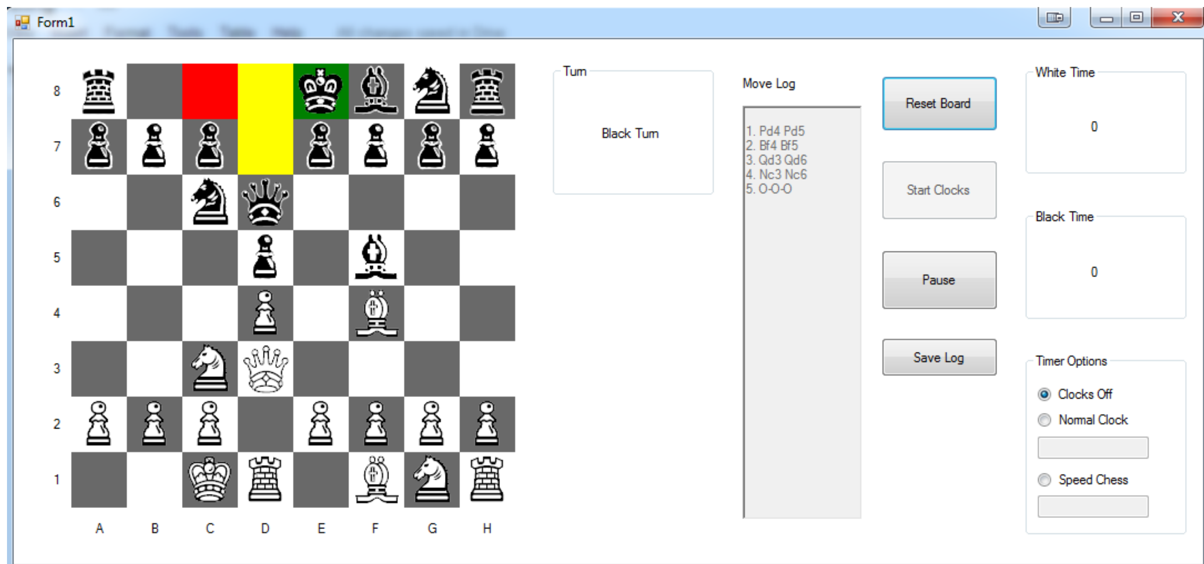


Fig. 3-4b

Castle move has been selected, King moves to the square that was highlighted in red, the castle moves to the other side of the King.

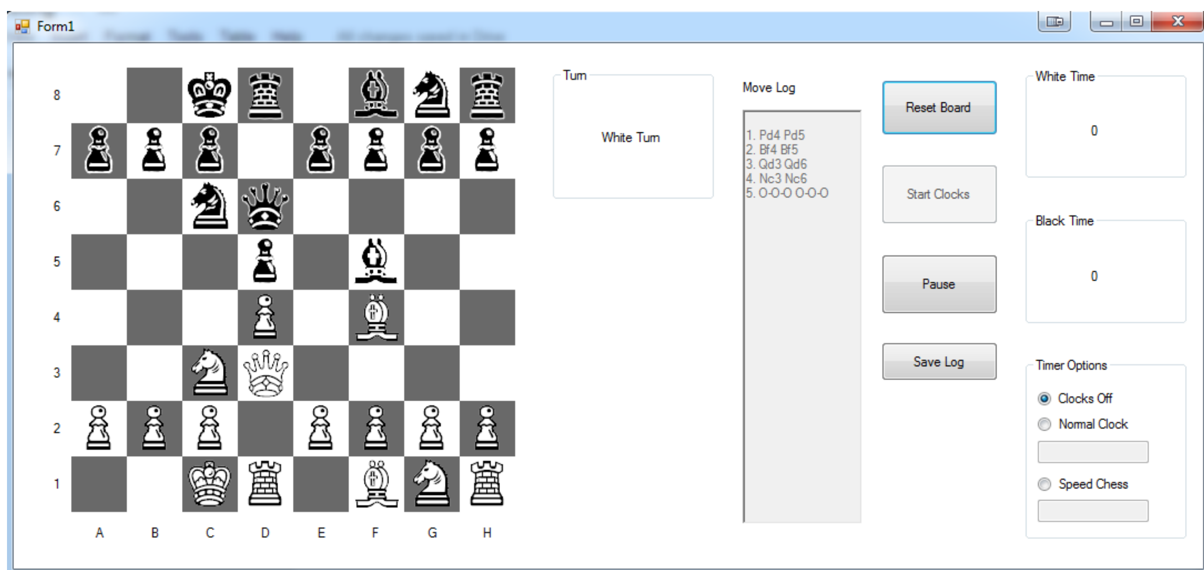


Fig. 3-5a

When a pawn reaches the other side of the board, this dialogue box appears. The user inputs which piece they want to promote to. Here a ‘Q’ is input.

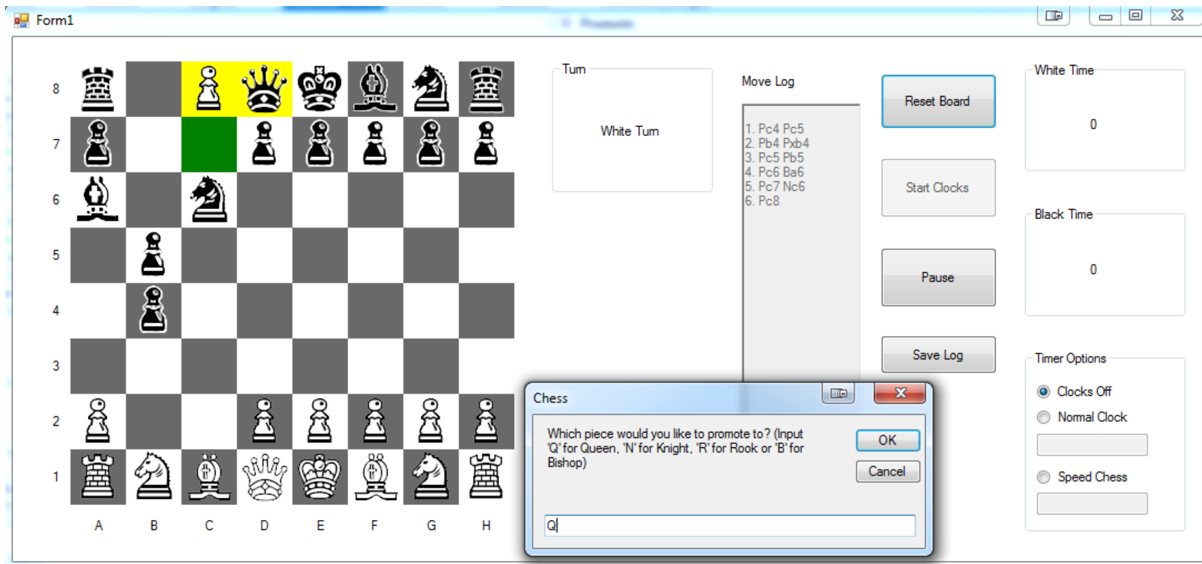


Fig. 3-5b

The pawn has been promoted to a Queen.

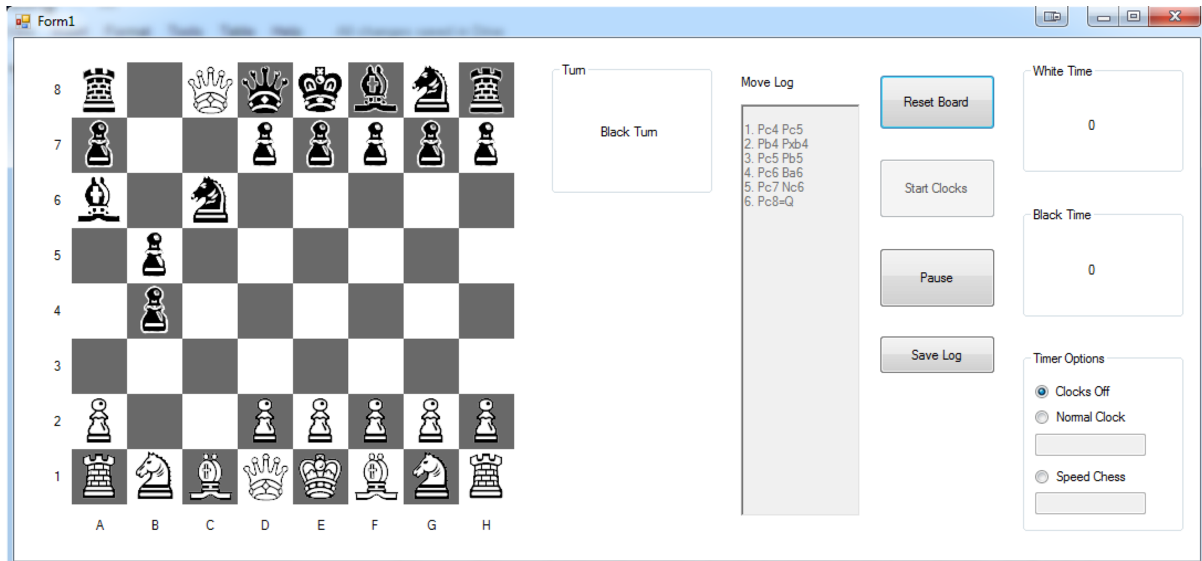


Fig. 3-6a

The user is attempting to input an invalid input into the inputbox.

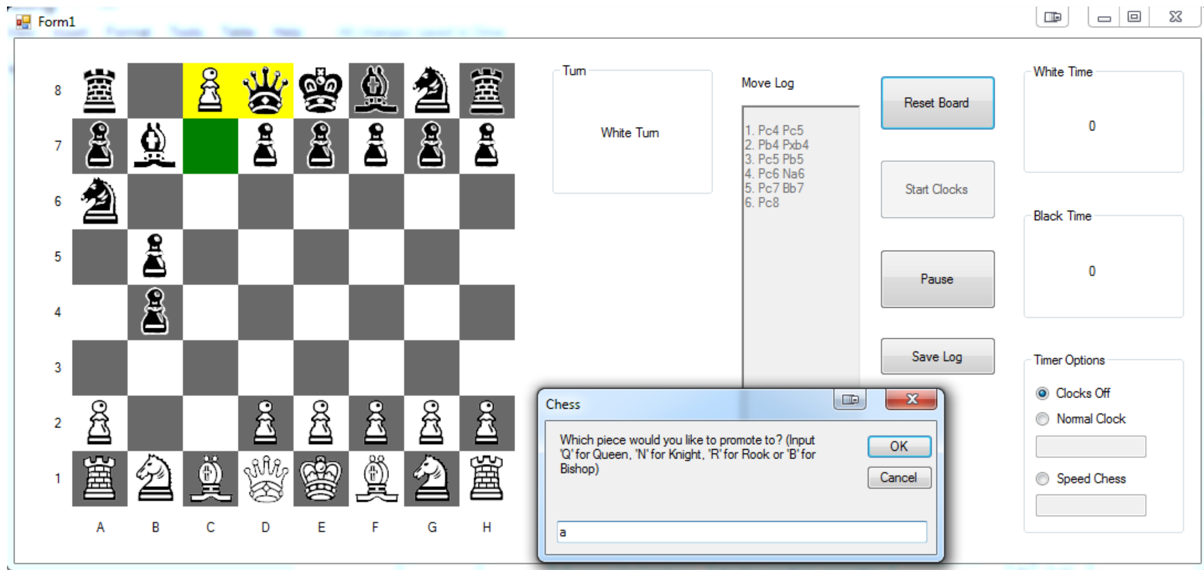
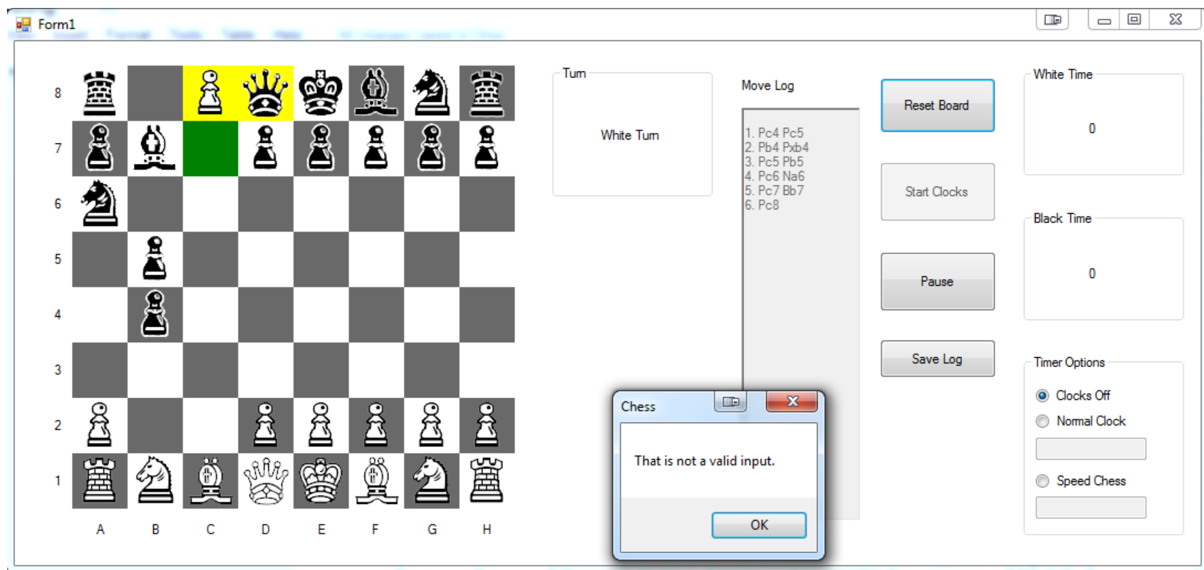


Fig. 3-6b



The program rejects the invalid input and requests a valid one.

Fig. 3-7a

Shows a pawn moving two spaces forward from its starting position.

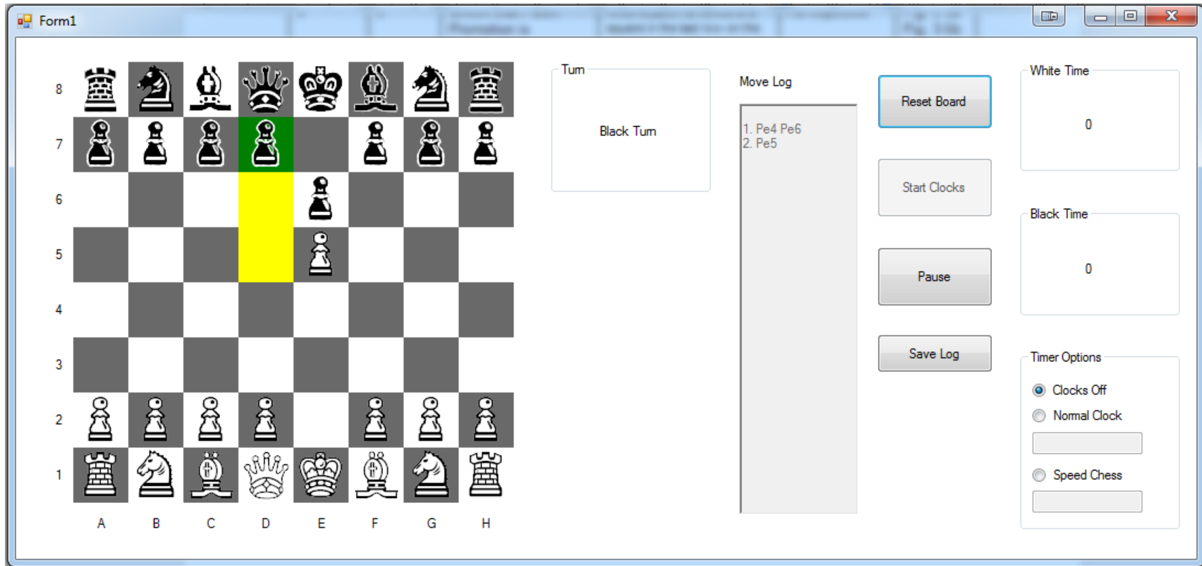


Fig. 3-7b

The white pawn can now take the piece that just moved forward two spaces with an En Passant capture.

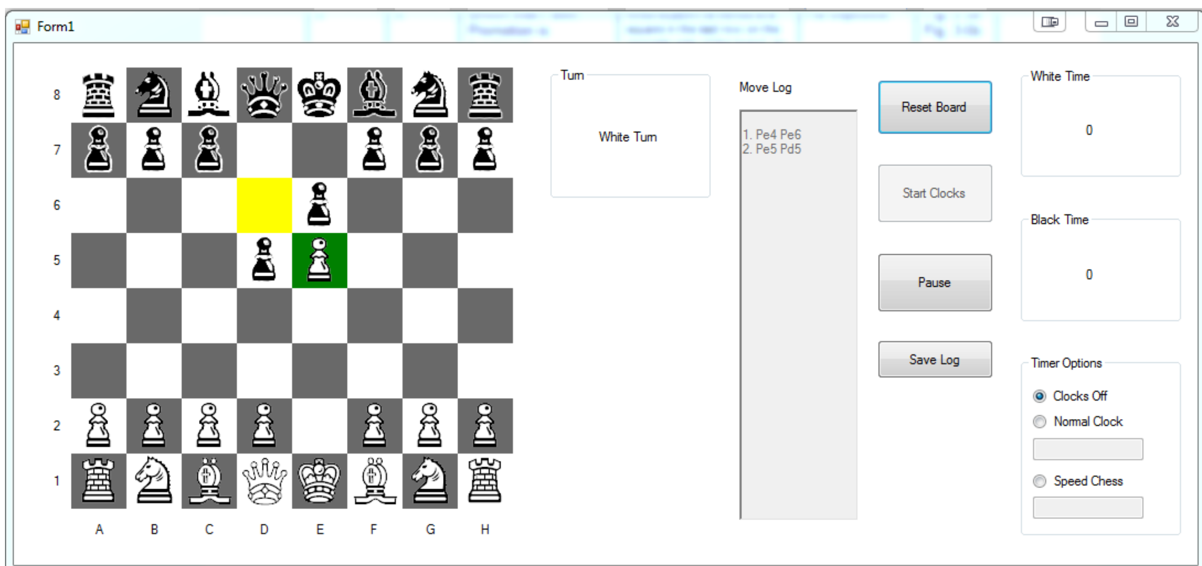


Fig. 3-7c

The black pawn has been taken by the white one by En Passant.

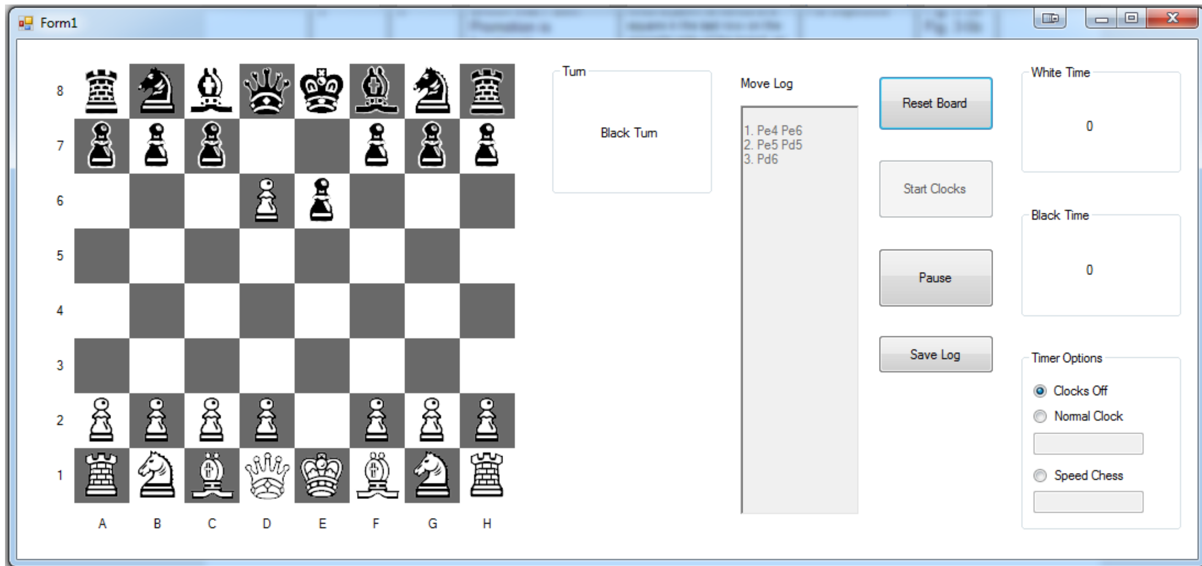


Fig. 4-1

Normal movement of a piece. The correct chess notation has been written in the Move Log.

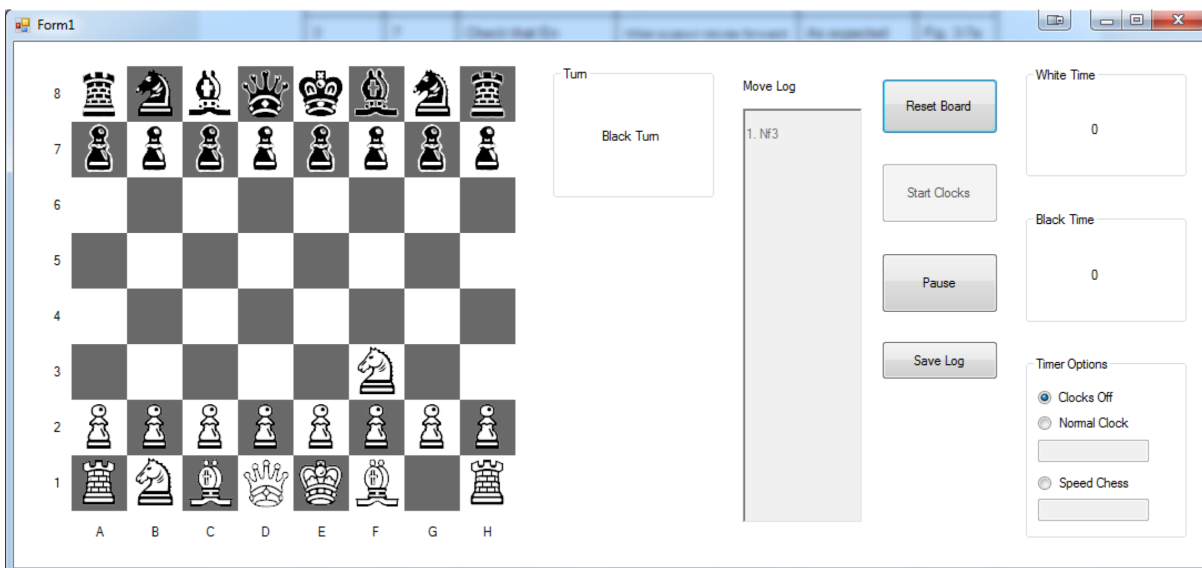


Fig. 4-2

Capture move has just been made. The correct chess notation has been written in the Move Log. An "x" is shown after the "P", which denotes a capture.

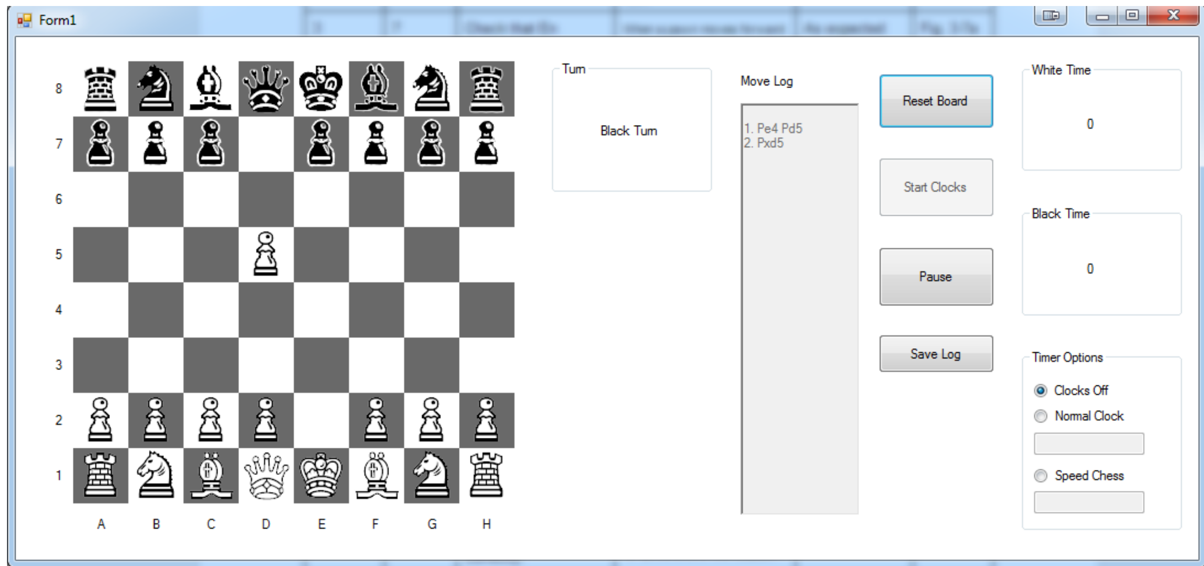


Fig. 4-3

A castle move has been made. The correct chess notation has been written in the Move Log.

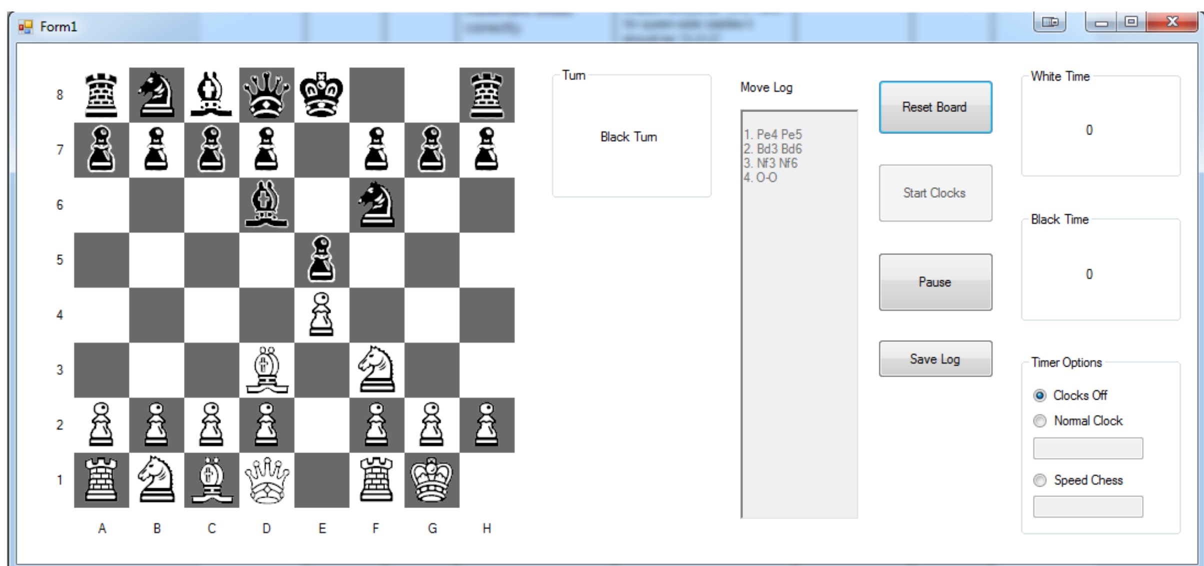


Fig. 4-6

A checkmate move has been made. A "#" has been added to the end of the move notation.

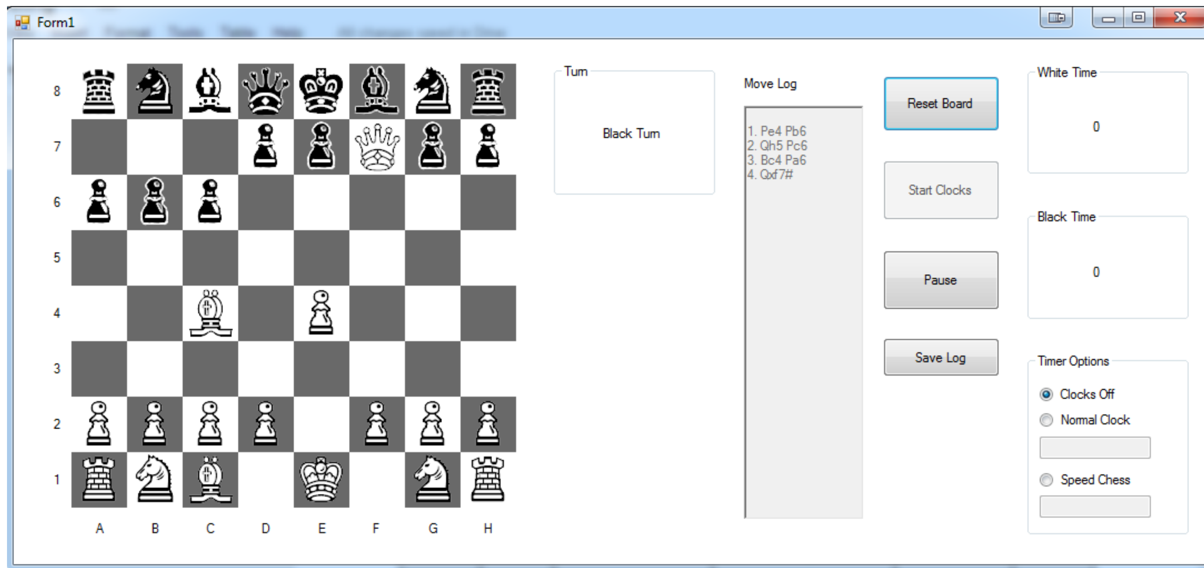


Fig 4-7a

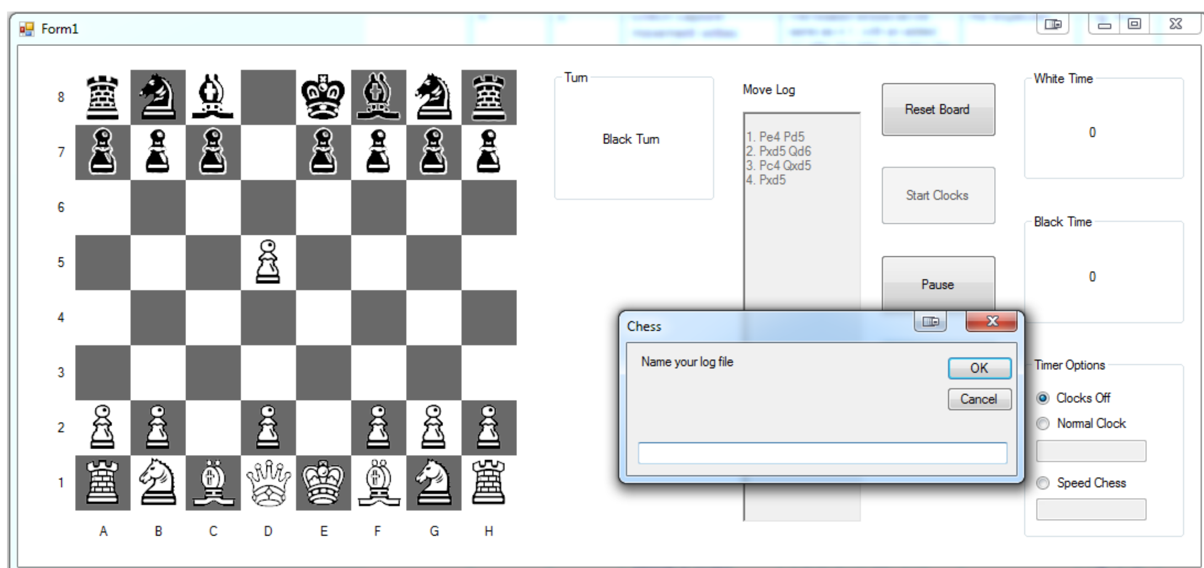


Fig. 4-7b

Text file containing the log saved from the program.

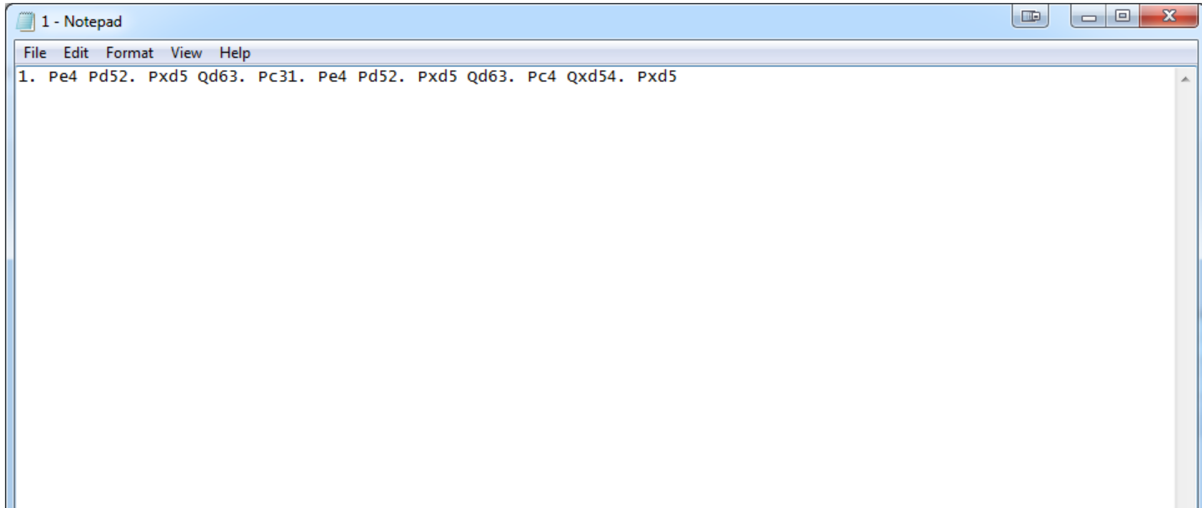


Fig. 5-1a

Normal clock has been selected, a time of 2 minutes has been selected. Start Clocks has been clicked, White Time is now counting down from 2:00.

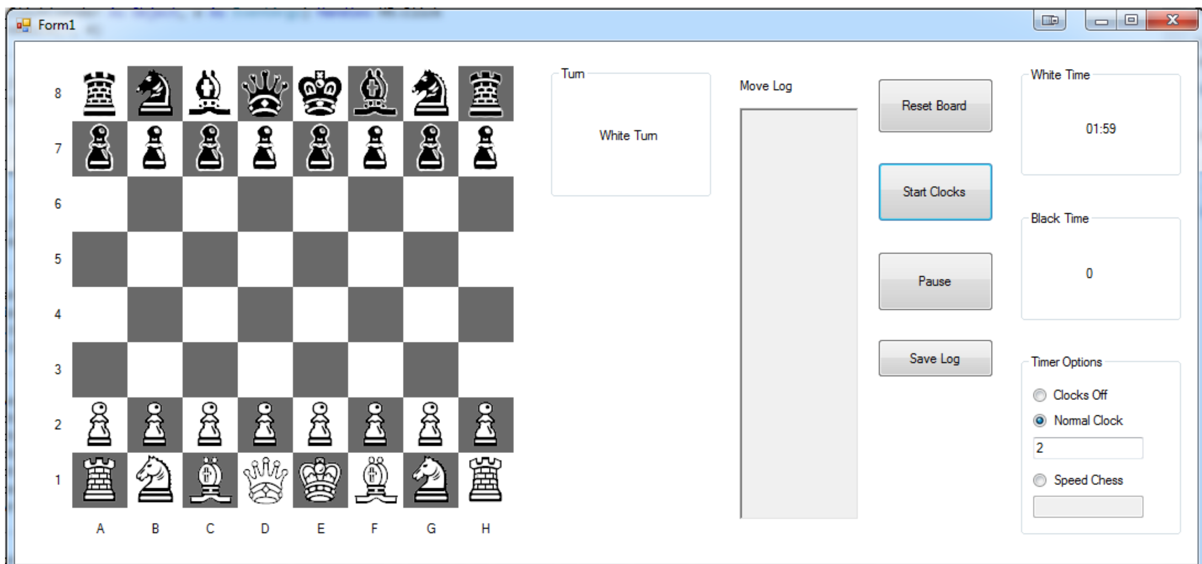


Fig. 5-1b

White player has made a move. The White timer has been paused, the Black timer is now running.

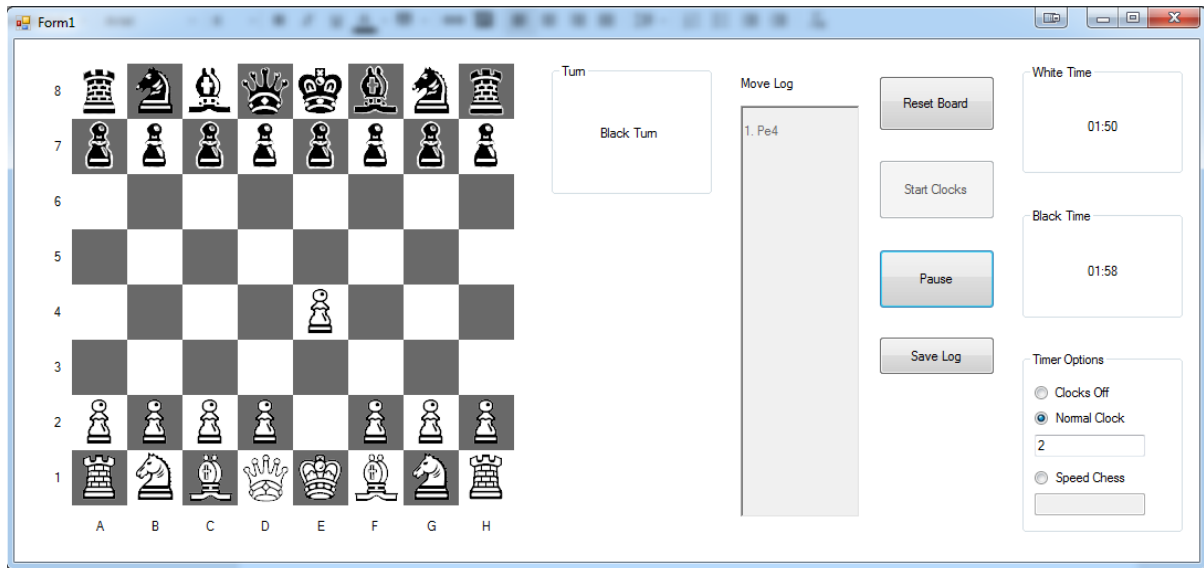


Fig. 5-1c

Black player has made a move; black timer is paused and white timer is running again, resuming from the time it was left with from last turn.

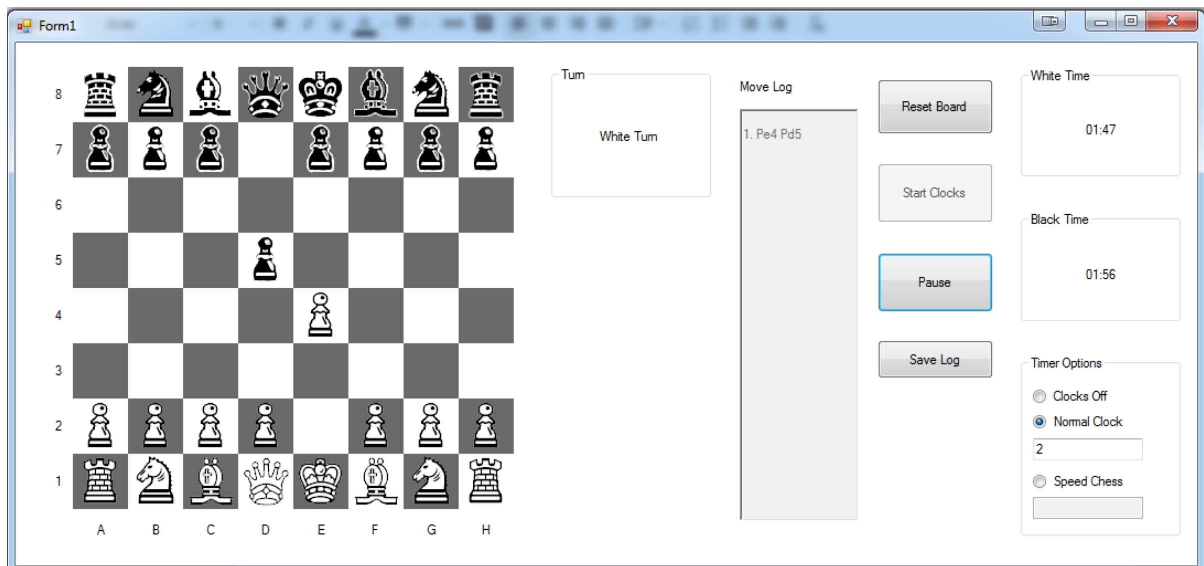


Fig. 5-1d

White player has run out of time, a message displays informing the users of this.

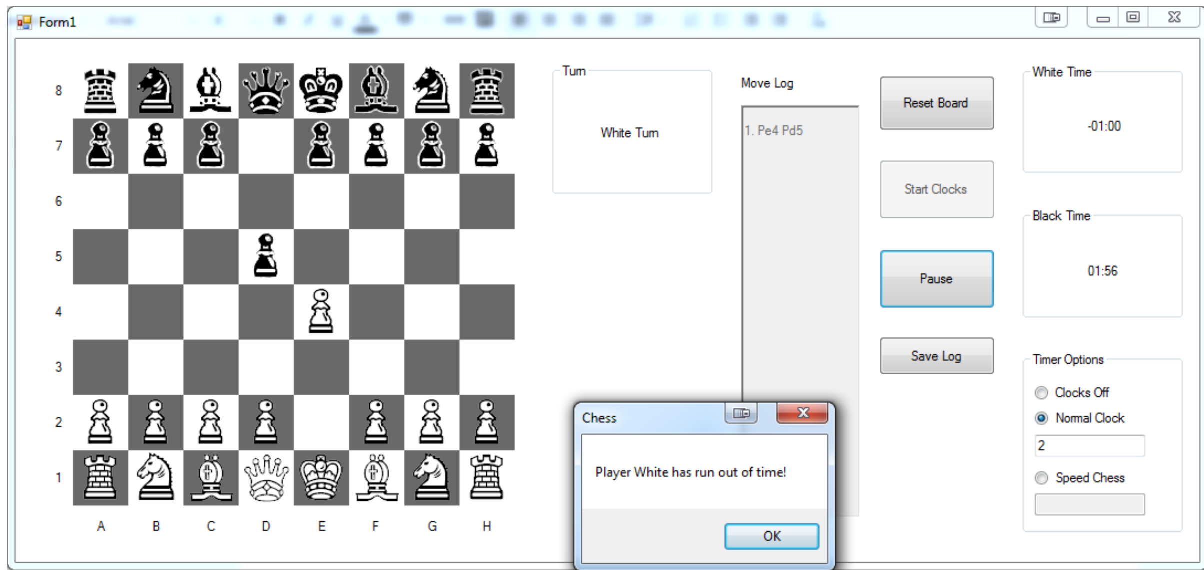


Fig. 5-1e

The user has attempted to input an invalid input into the textbox. When the Start Clocks button is pressed, the program rejects the input and informs the user that it is invalid.

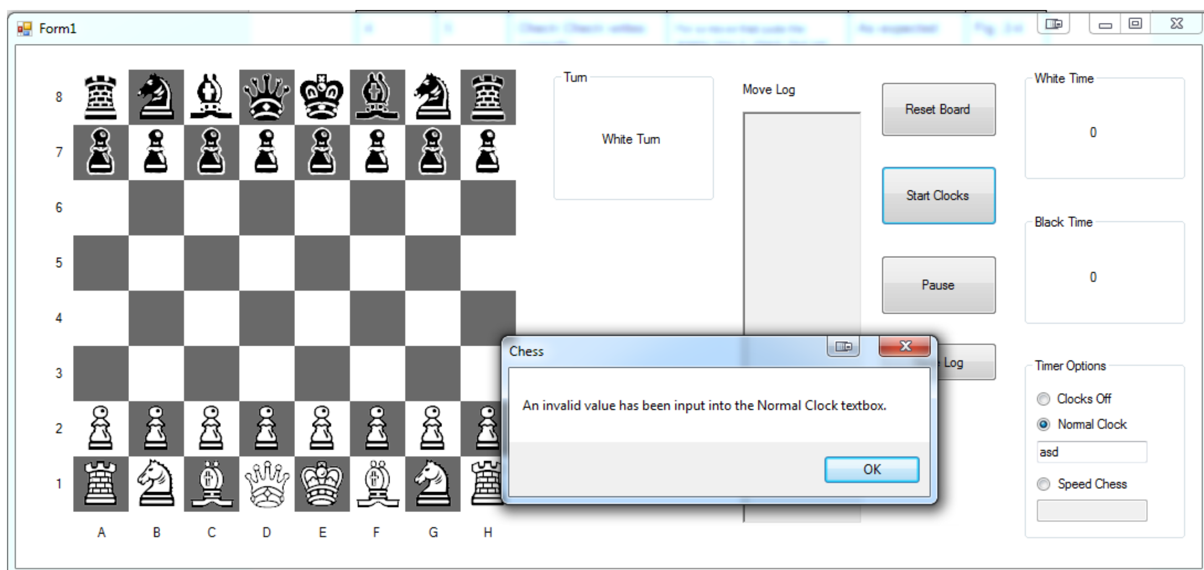


Fig. 5-2a

Speed Chess setting has been selected. A time of 10 seconds has been input. Start Clocks has been pressed. White timer is now counting down from 10 seconds.

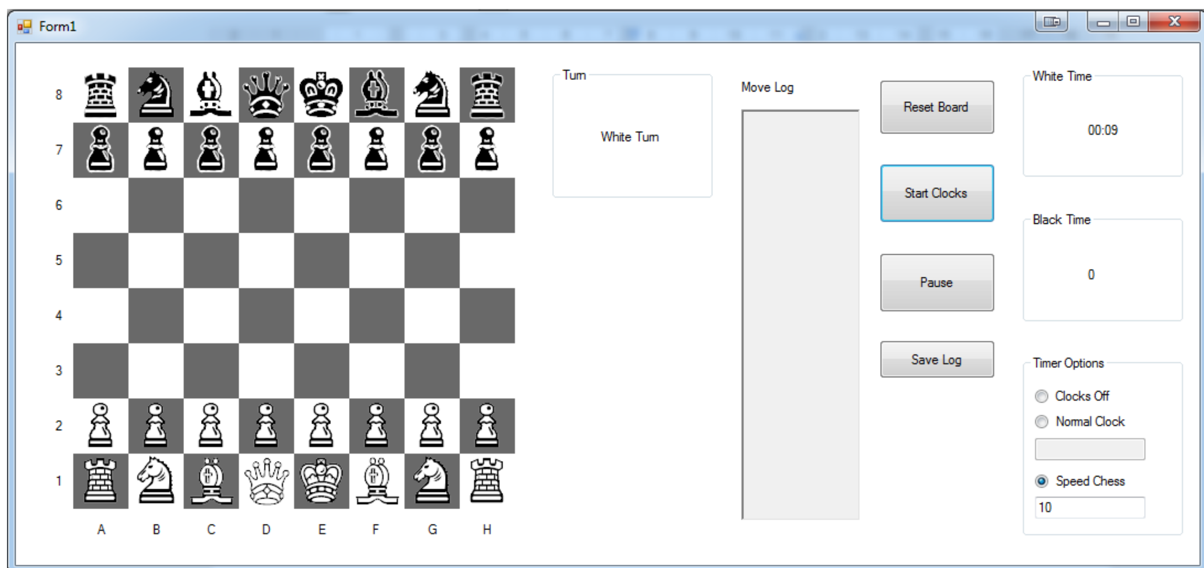


Fig. 5-2b

White player has made a move; white timer has stopped and black timer is now counting down from 10 seconds.

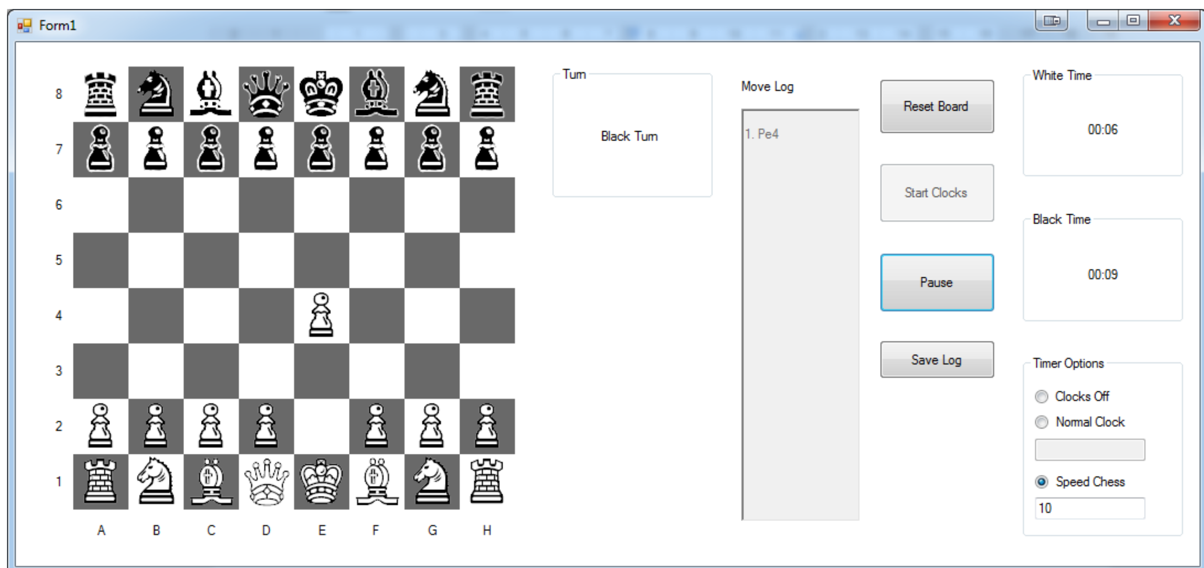


Fig. 5-2c

Black player has made a move; white timer is reset to 10 seconds again, and is now counting down.

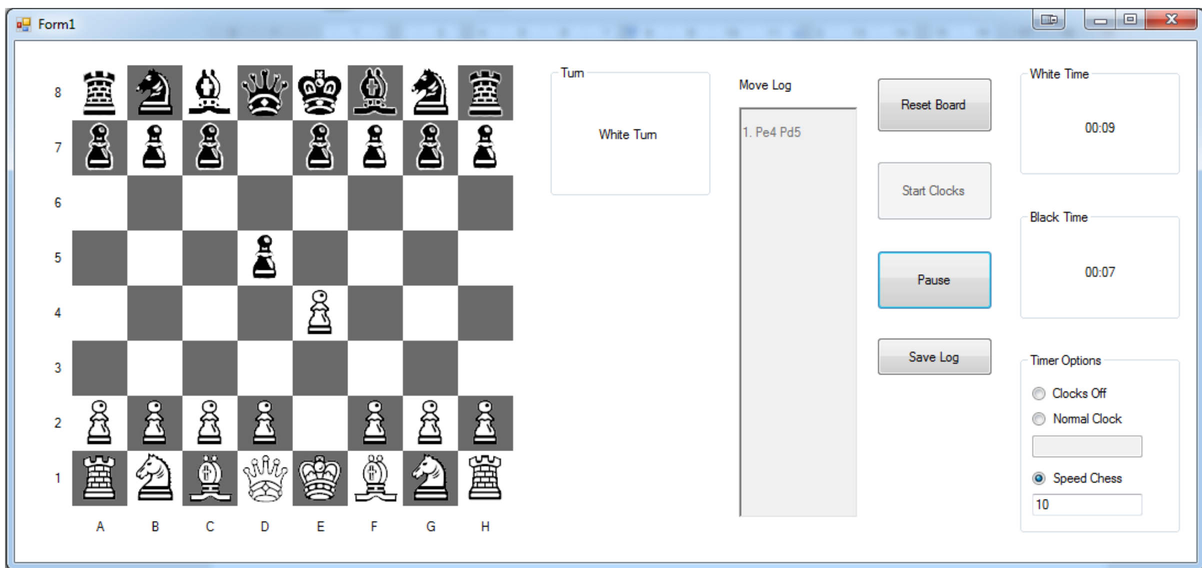


Fig. 5-2d

White player has run out of time, and a message box appears to inform the user of this.

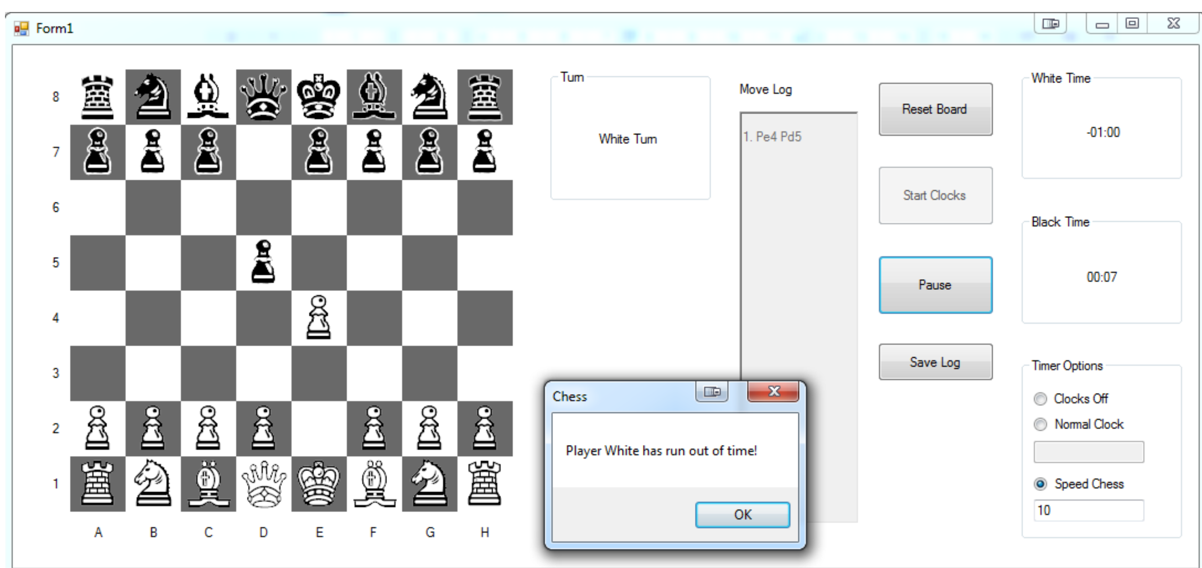


Fig. 5-2e

The user has attempted to input an invalid input into the textbox. When the Start Clocks button is pressed, the program rejects the input and informs the user that it is invalid.

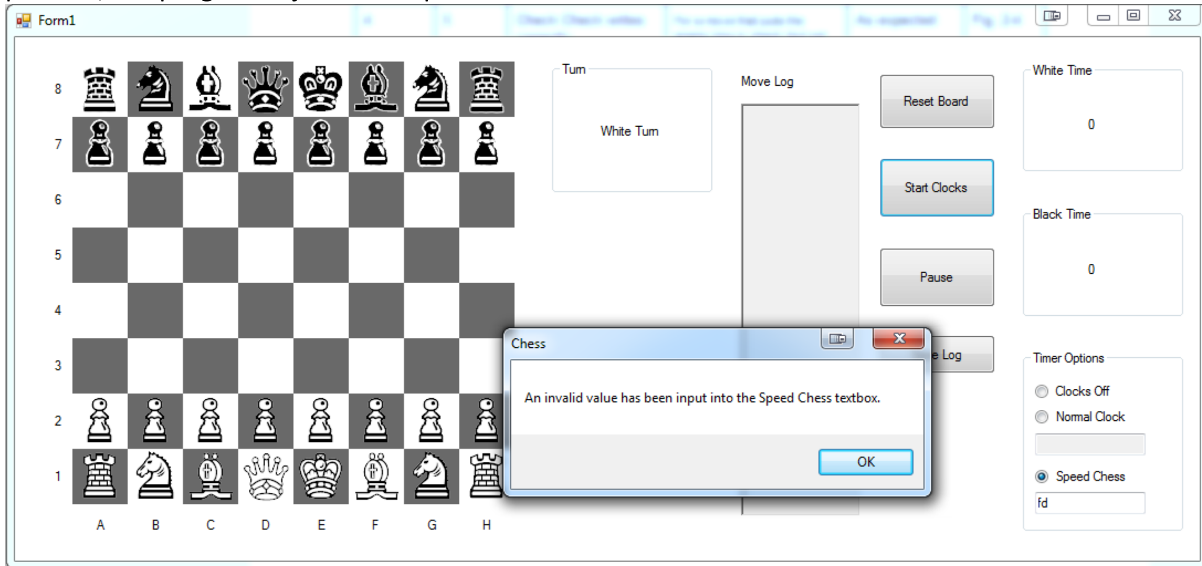


Fig. 5-3a

The user has selected 2 minutes for the input. Start Clocks has been clicked, and then Pause was clicked 1 second later, and the program was left for 10 seconds before a screenshot was taken. The time here is still paused at 1:59.

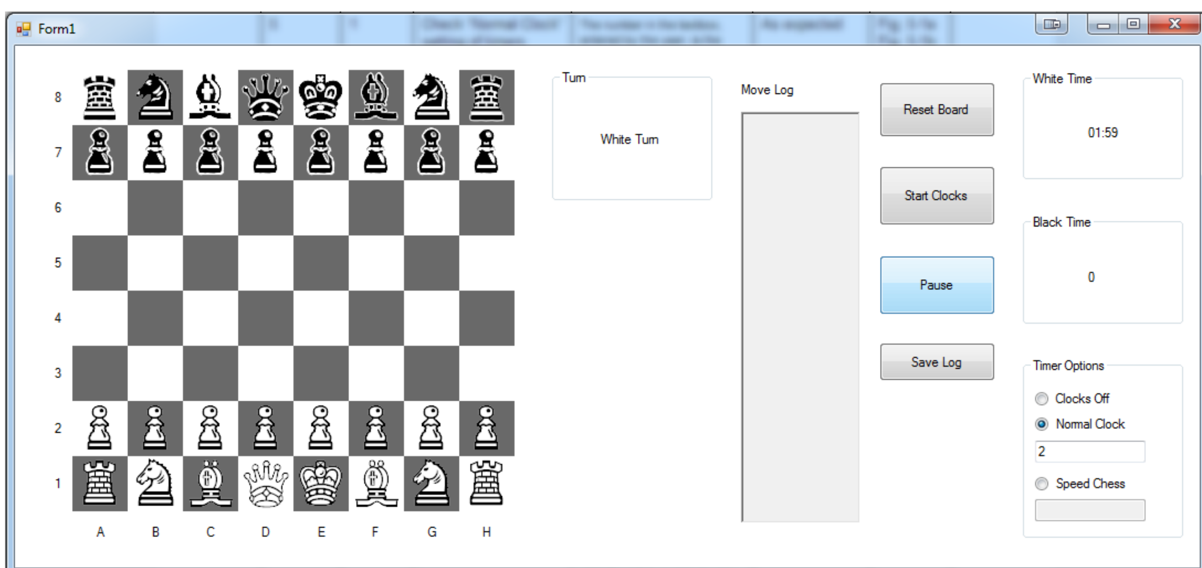


Fig. 5-3b

The Pause button is clicked once again, and the program is left for 5 seconds before a screenshot is taken. As can be seen, the timer has started, and has counted down another 5 seconds from when it was paused.

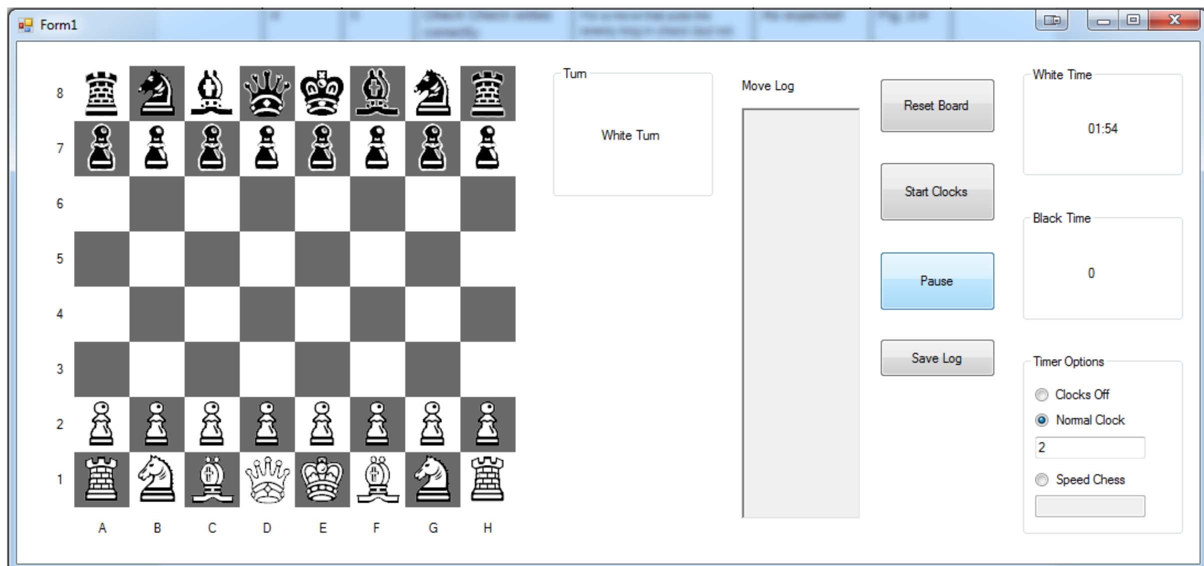


Fig. 5-4a

Showing a game in progress.

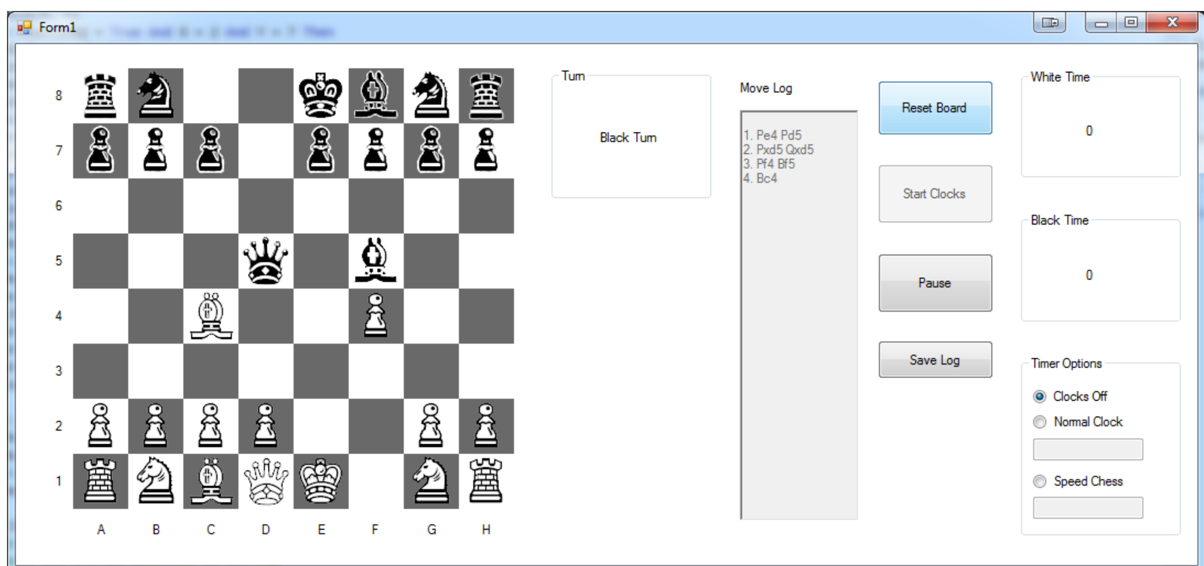
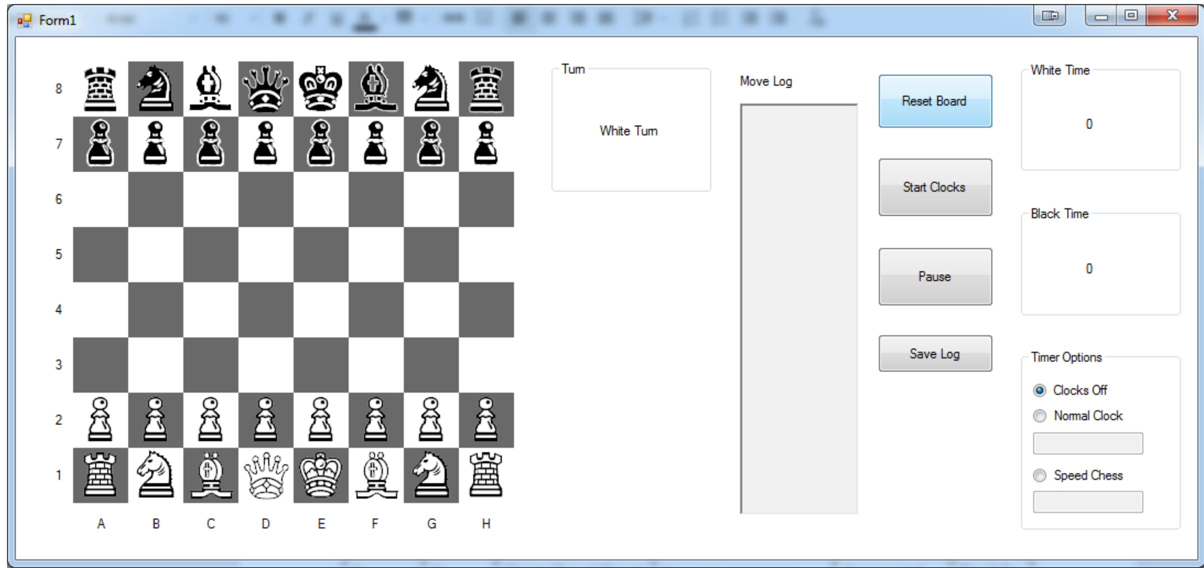


Fig. 5-4b

After the Reset Board button has been pressed.



Appraisal

System Objectives

There must be a visual interactive 2-D grid, which shows the position of the pieces, and allows the player to make moves by clicking on them.

Objective met.

As can be seen from the Interface Design section, the chosen design includes such a grid. The screenshots taken for test areas 1, 2 and 3, which are included in the appendix, show that the grid displays the positions of the pieces at any given point in the game, and they also show that the result of clicking on a square is a set of possible moves; one of these can be clicked to make a move.

When a piece is clicked, all possible moves should be highlighted on the board.

Objective met.

As can also be seen by the screenshots taken for test areas 1, 2 and 3, which are included in the appendix, the click of a square which contains a piece results in all possible moves being highlighted on the board.

The program must obey the standard rules of chess, and must not allow players to disobey these rules.

Objective met.

In the Testing section, test areas 1, 2 and 3 test whether the program follows all the rules of chess correctly. The results of these tests were all as expected, so the program does not allow players to disobey the rules of chess.

If a player inputs an invalid move, the pieces will stay in the same position on the board.

Objective met.

As shown by test 2.1 in the testing section, if an invalid move in input, the pieces do stay in the same position.

The program must include the special rules of Castling, En Passant and Pawn Promotion.

Objective met.

As shown by test area 3 in the testing section, these special moves are included and they all function properly.

There should be a log of all the moves made in the game, with an option to save the log to a text file.

Objective met.

As can be seen by the final interface design, there is a textbox that is used as a move log, and a button labelled “Save Log” which is used to save the log to a text file. Test area 6 in the testing section shows that these are fully functional.

The program must include checks for whether a player is in check or checkmate, and notify the user when either of these happen.

Objective met.

As shown by test 2.4, when a player is in check the move that puts that player in check has an exclamation mark following it, indicating a check move. As shown by test 2.5, when a move is played that puts a player into checkmate, a dialogue box appears notifying the players that checkmate has been made.

There must be a reset button to return the board to its starting state.

Objective met.

As can be seen by the final interface design, there is a button labelled “Reset Board”. As shown by test 5.4, this button will return the board to its starting state.

There should be an indicator to show whose turn it is.

Objective met.

The final interface design shows that there is a turn indicator and test 2.2 shows this indicator to be fully functional.

The grid spaces on the chess board should be numbered on the vertical axis, and lettered on the horizontal axis.

Objective met.

As can be seen from the final interface design, the grid spaces are labelled exactly as the objective states.

The program must incorporate a chess clock, giving each player a certain amount of time to make their moves.

Objective met.

As can be seen by test area 5, the chess clocks are implemented and fully functional.

> **The program should include a pause button, which will stop the clocks to allow the players to have a break.**

Objective met.

As can be seen by the final interface design, there is a Pause button included, and as shown by test 5.3, this button is fully functional.

> **There must be an option to decide whether the game will be timed, and if so how much time each player will have, as well as how the game will be timed.**

Objective met.

As can be seen by the final interface design there are timer options included, and as shown by tests 5.1 and 5.2, these options are fully functional.

Analysis of Feedback

The following is a report on my project by my end user, Mr RG Patten.

> Aman has been constructing a chess program: not the Herculean task of making the computer actually play the game, but a teaching aid to display and record the moves. He has made the graphics work very successfully, with a clear board diagram in standard chess figurines, all possible moves being highlighted when a piece is clicked, and move entry by drag and drop working very conveniently. This will make the program particularly amenable to SmartBoard use. The moves also appear as a scrollable move list on the right of the chessboard diagram, which is valuable both for working back and forward through a game, and for familiarising pupils with the correct chess notation.
>
> The project was very near to completion when I last saw it, and looked set to be a very worthwhile piece of software. The task of encoding the rules of chess, to teach the machine to distinguish between legal and illegal moves, is quite complicated but he has accomplished it nicely. I am looking forward to using this with junior members of the Chess Club.

R.G. Patten,
Deputy Director of Studies (Timetable),
King Edward VI School,
Kellett Road,
Southampton
SO15 5UQ

Overall, the feedback from Mr Patten is very positive. He noted that the graphics work very well, and everything is clearly presented. He also notes that the way in which pieces are moved will work very well with the SmartBoard. The SmartBoard is an interactive whiteboard

that effectively replicates a touchscreen control. This will mean that users can tap the pieces that they want to move on-screen, instead of having to share a mouse in order to play.

Mr Patten also comments on the usefulness of the Move Log, as it allows the players to review the game, and for teaching pupils chess notation. He then writes that he is very pleased with the program.

Potential Improvements

A function could be added to replay through previous games. The program could load the text files with chess notation that have been saved by the user, and the user will be able to review the entire game easily with a graphical representation. This would be great for educational purposes, to show the members of the chess club where they have gone wrong in a game, and how they can improve. Several buttons will have the be added; one to load a Move Log text file, one to move on to the next move in the replay, and possibly also one to go backwards.

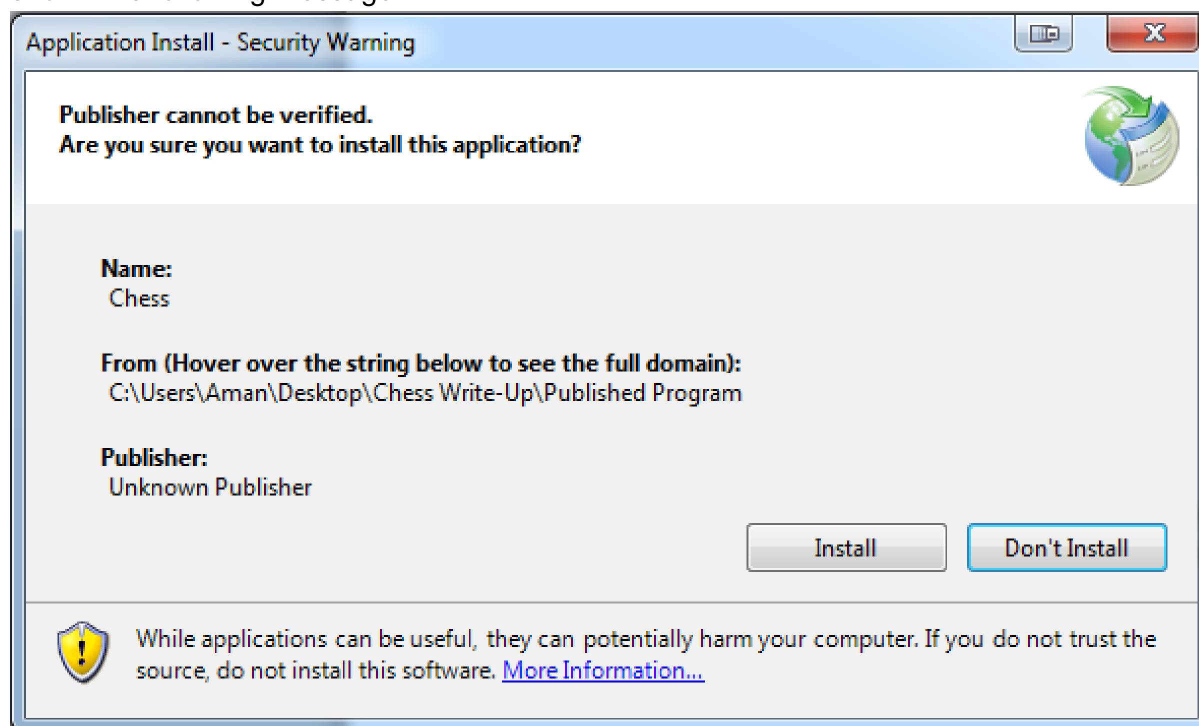
Another useful addition to the program would be a function to detect Stalemate. Stalemate is when a player is not in Checkmate, but cannot make any valid moves. Currently, the system will not tell the user if Stalemate has been reached; the users will have to figure this out by themselves by clicking on their available pieces.

A very useful addition would be to add an artificial intelligence to play against a single user. This would allow the user to practice or to play for fun without having to find a human opponent to play with. This would involve a large amount of knowledge about playing patterns and techniques when playing chess, and would be a very difficult task to complete. The intended users of this program, however, are the members of the chess club, and at chess club there will be no shortage of human players to play against. As such, this feature of the program might not be taken advantage of by the users.

User Manual

Installation

Insert the flash drive or other device with the published software on it. Navigate to the folder where the software has been placed, and double-click on the “setup” file. You will then be shown the following message:



Click Install, and the software will install. The application should automatically open at this point, and can now be accessed at any time via:

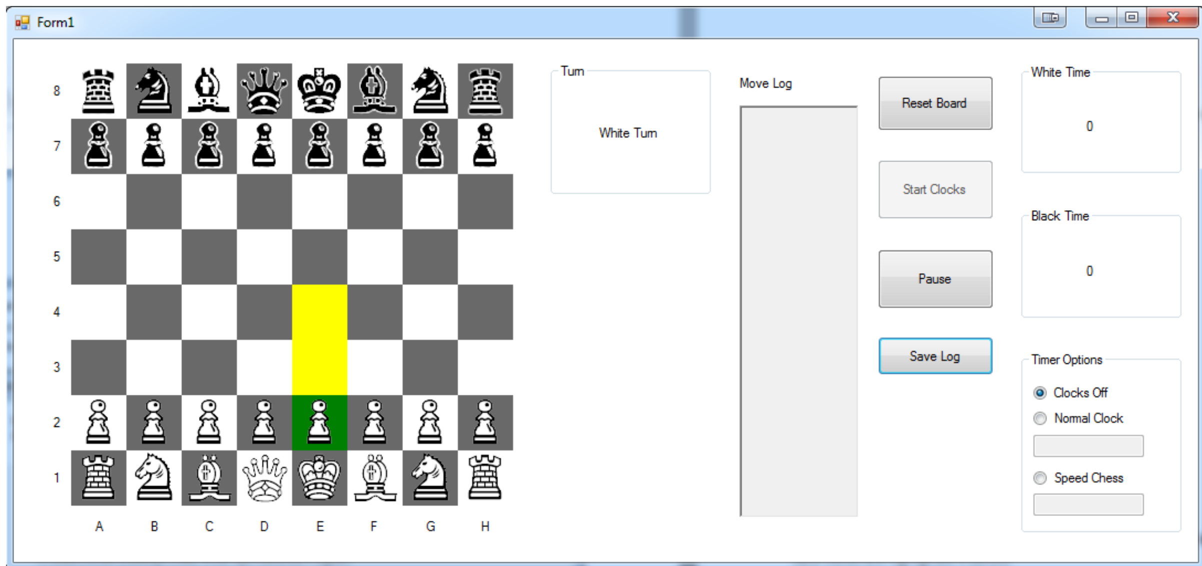
Start > All Programs > Chess > Chess.exe

Basic Playing

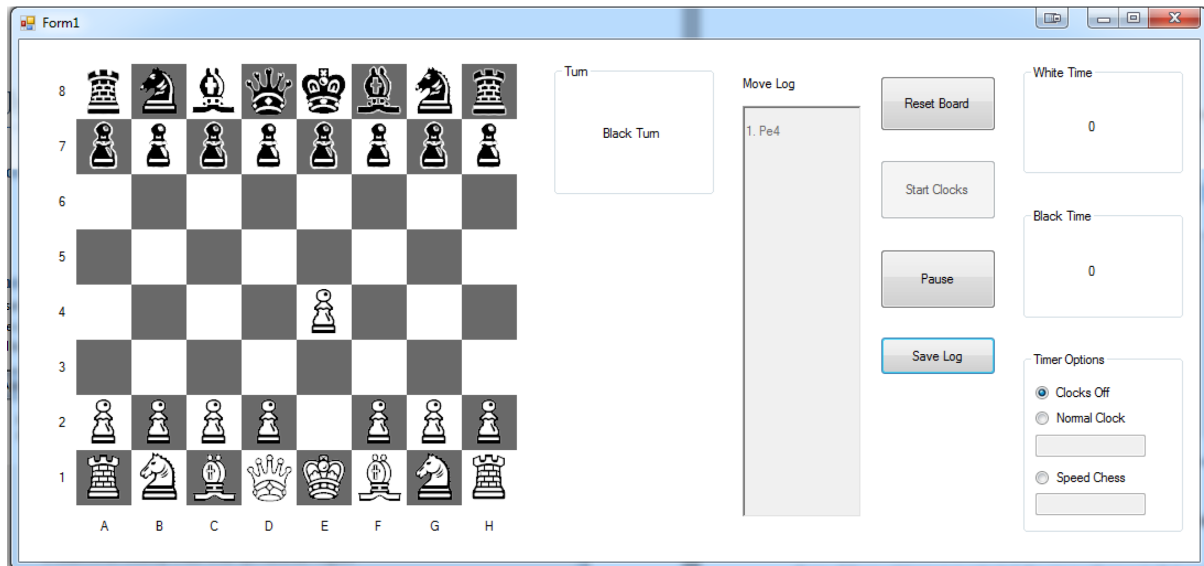
This manual assumes the user has at least a basic grasp on how to play the game of Chess.

Moving Pieces

If you click any piece (assuming it is the correct colour of piece for which turn it is), all valid moves for that piece will be highlighted on-screen, like so:

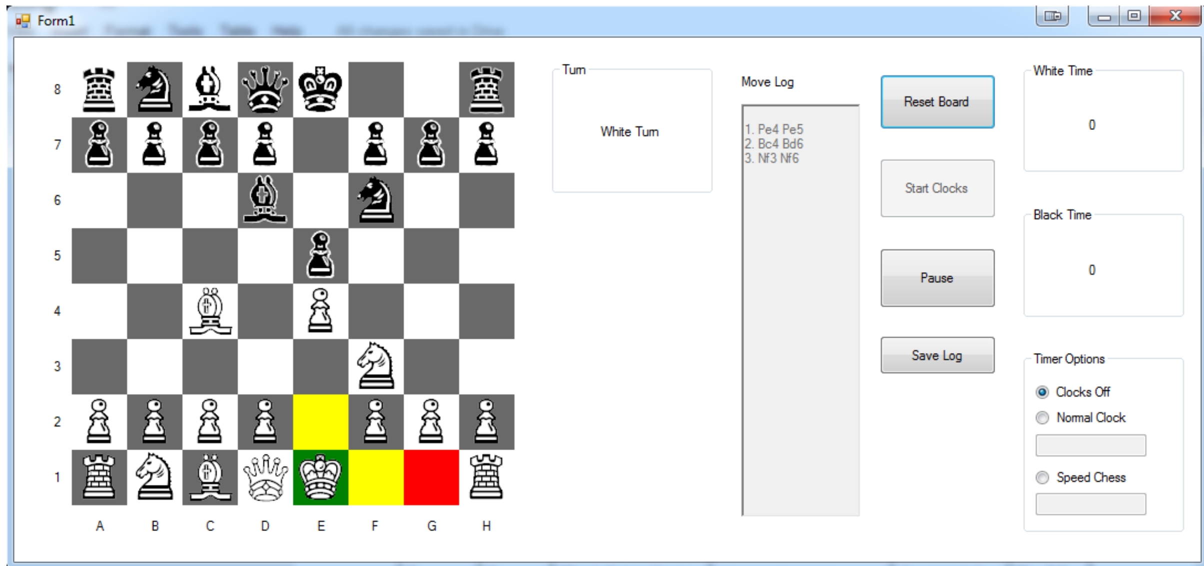


As can be seen from the picture above, the piece that is selected is highlighted in green, while any valid moves are highlighted in yellow.
If one of these valid moves is selected, the piece will move to that position, and the turn indicator will change.

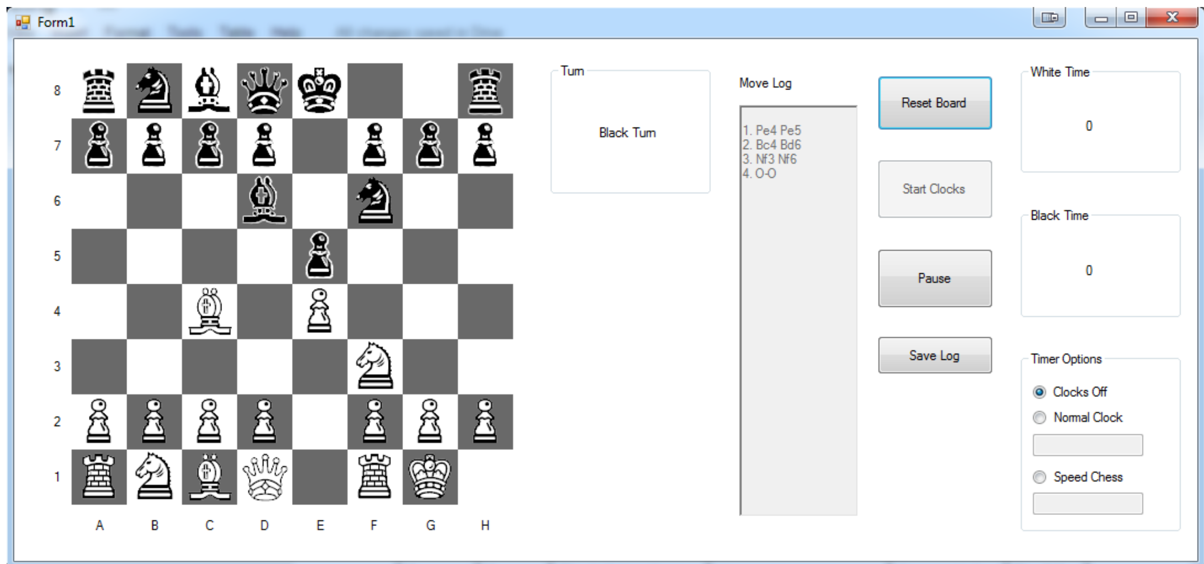


Castling

Castling works very similarly to normal moves. In order to perform a Castle move, you must select the King while a Castle is available.

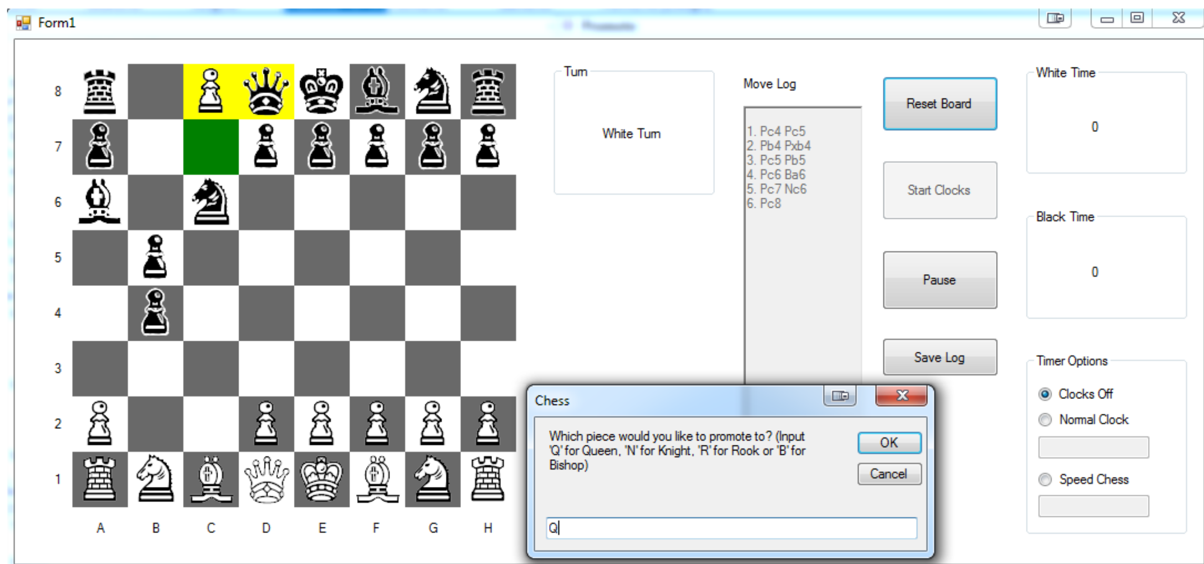


You can see here that a square is highlighted in red. If you click this square, the Castle will be performed.

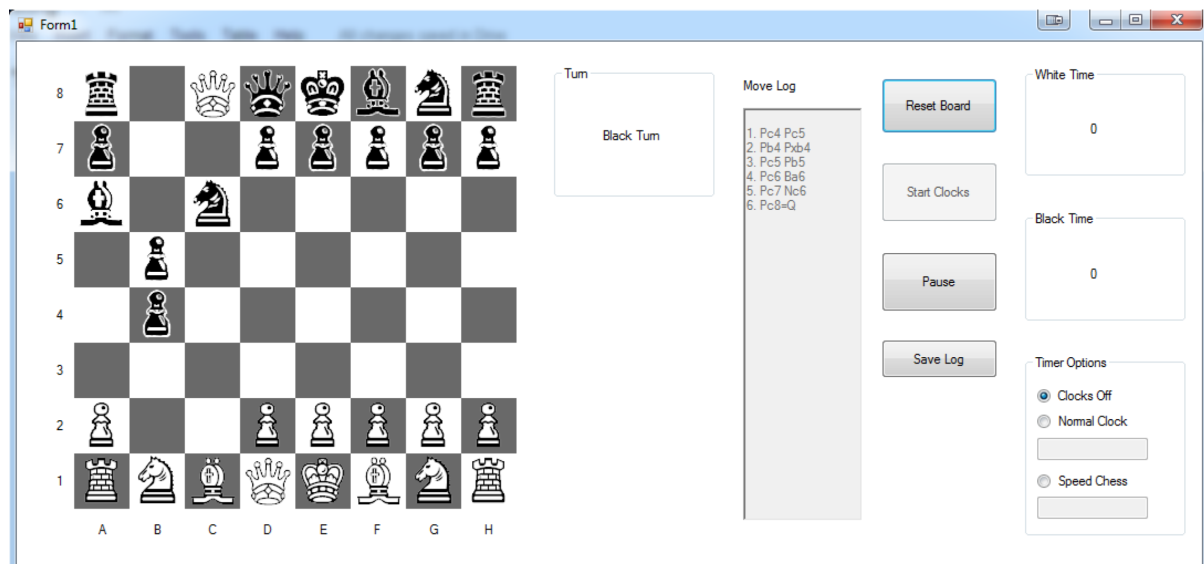


Pawn Promotion

When a pawn is moved to the other side of the board, an dialogue box will appear requesting an input.

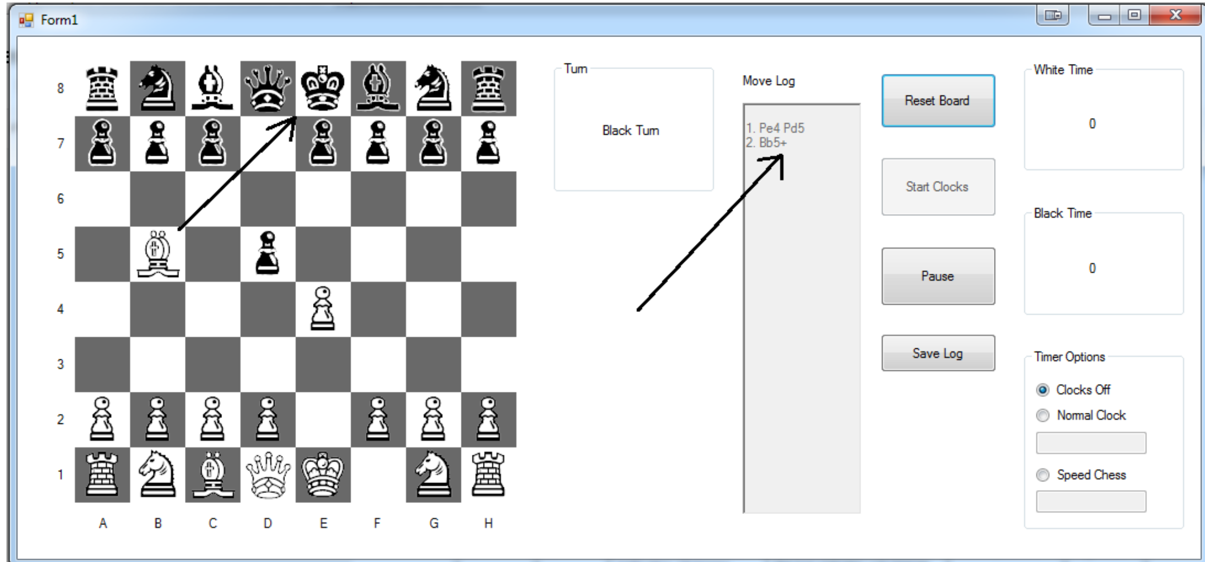


The input it requests is for which piece you want the pawn to be promoted to. The options are “Q” for a Queen, “N” for a Knight, “R” for a Rook or “B” for a Bishop. Lowercase letters are accepted. If an invalid input is put into the textbox, the application will request a valid input.
Once a valid input has been given, the pawn will be replaced with the piece that was requested, like so:

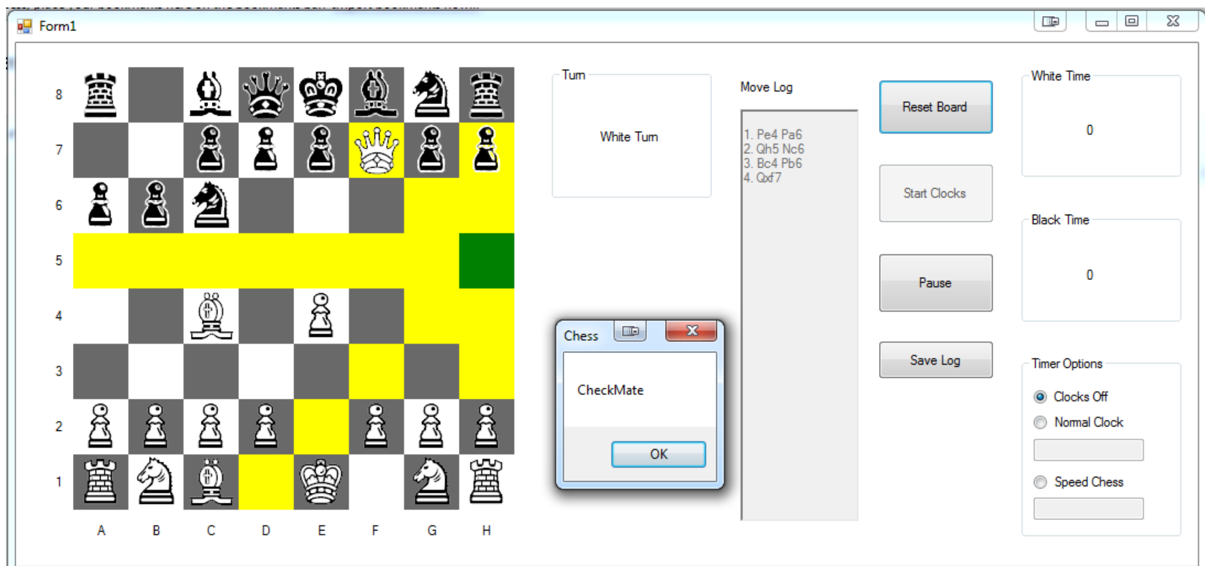


When is it Check or Checkmate?

When a move puts a King into Check, the only indication is an added "+" to the end of the move in the Move Log.



However, when Checkmate has been reached, a message box appears on-screen, like so:



Resetting the Board

To reset the board at any point in the game, simply click the "Reset Board" button, and the board will return to its starting state. All the pieces will be in their starting positions, and the Move Log will be cleared.

Using the Clocks

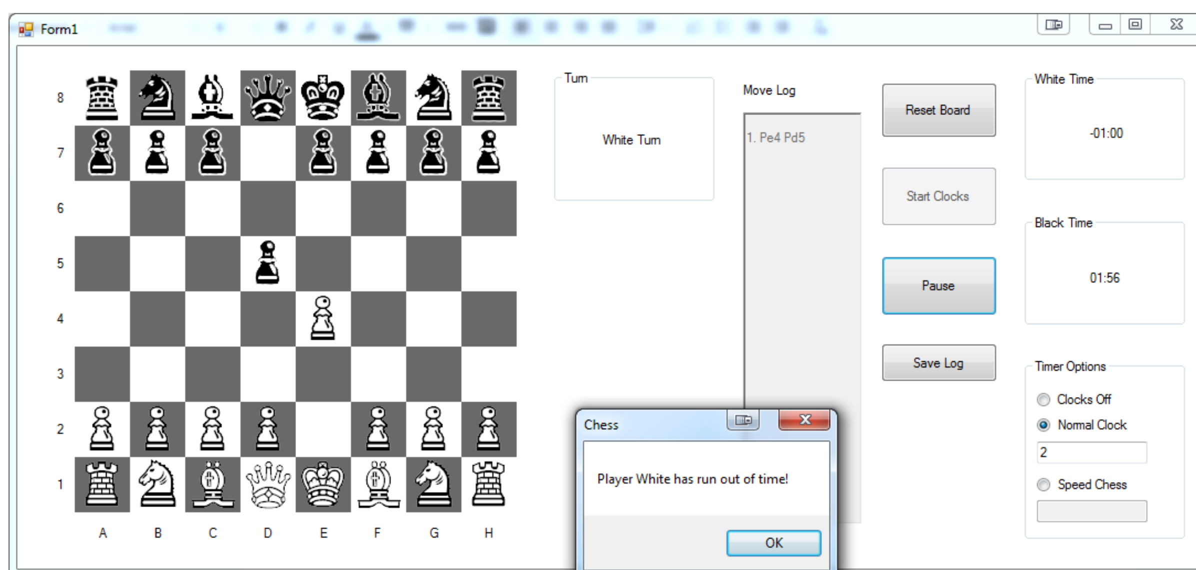
As default, the clocks are off. This can be seen in the Timer Options section, at the bottom-right of the application:

Timer Options
Clocks Off
Normal Clock
Speed Chess

As can be seen here, there are two options for the clocks; Normal Clock and Speed Chess.

Normal Clock

When the Normal Clock option is selected, the textbox underneath it becomes available for input. The amount of time in minutes that each player will have should be input into this box. After that has been done, the “Start Clocks” button should be pressed, which will start the timer under “White Time”. The program will start counting down from however many minutes were input into the textbox. When a player makes a move, their timer will pause, and the opponent’s timer will start. When a player runs out of time on their clock, a message will appear, like so:



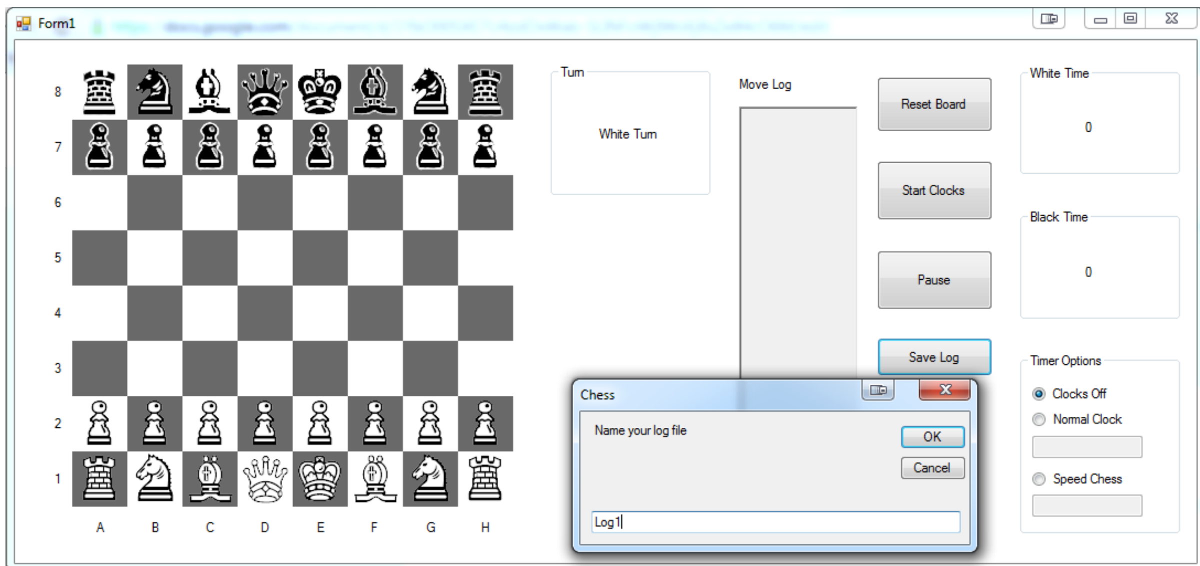
Speed Chess

When the Speed Chess option is selected, the textbox underneath it becomes available for input. In this case, the number input into the textbox will correspond to an amount of time in seconds for each player. When the Start Clocks button is clicked, the timer under “White Time” will start counting down from however many seconds were input into the textbox. When a player makes a move, their timer will stop, and the next player’s will start from however many seconds were input into the textbox. At the start of each player’s turn, their

timer will be reset to the amount of seconds input into the Speed Chess textbox. When a player runs out of time, a message will appear informing the players of this, the same as with the Normal Clock setting.

Saving the Move Log

Saving the Move Log to a text file is a very simple process. Simply click the “Save Log” button, and then type in a name for your log in the inputbox that appears, like so:



Click “OK”, then your log will be saved to the root C Drive directory, under C:*Filename*.txt, with the *Filename* being replaced by whatever was input in the textbox. In the instance shown above, it will be C:\Log1.txt.

Appendix

Full Program Listing

Main Form

```
Public Class Chess
    Public Grid(7, 7) As String 'Keeps a store of which piece is in which square
    Public ValidCheck(7, 7) As Boolean 'Used when checking valid moves; any squares
that are the destination of valid moves are marked as true, while any other squares
are marked as false
    Public PieceClick As Boolean = False 'Used to check whether a square has been
clicked
    Public X1, Y1 As Integer 'Used to indicate the coordinates of the initial square
    Public Game1 As Game = New Game 'Creates Game object
    Public CastleWQMoved, CastleWKMoved, CastleBQMoved, CastleBKMoved As Boolean 'Used
to indicate whether either of the pieces in a particular castle move have moved.
    Public CastleWQ, CastleWK, CastleBQ, CastleBK As Boolean 'Used to indicate whether
a particular Castling move is valid
    Public WhiteTimeStore, BlackTimeStore As Double 'Stores for how much time each
player has left
    Public EnPassant As Boolean 'Whether a pawn has moved forward two spaces last turn
    Public EPPosX, EPPosY As Integer 'The destination position of the pawn to move
forward two spaces
    Dim WhitePaused As Boolean 'Used when the clocks are paused to store which clock
was running at the time
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        Game1.InitializeGame()
    End Sub

    Private Sub A1_Click(sender As Object, e As EventArgs) Handles A1.Click
        Game1.SquareClick(0, 0)
    End Sub

    Private Sub A2_Click(sender As Object, e As EventArgs) Handles A2.Click
        Game1.SquareClick(0, 1)
    End Sub

    Private Sub A3_Click(sender As Object, e As EventArgs) Handles A3.Click
        Game1.SquareClick(0, 2)
    End Sub

    Private Sub A4_Click(sender As Object, e As EventArgs) Handles A4.Click
        Game1.SquareClick(0, 3)
    End Sub

    Private Sub A5_Click(sender As Object, e As EventArgs) Handles A5.Click
        Game1.SquareClick(0, 4)
    End Sub

    Private Sub A6_Click(sender As Object, e As EventArgs) Handles A6.Click
        Game1.SquareClick(0, 5)
    End Sub

    Private Sub A7_Click(sender As Object, e As EventArgs) Handles A7.Click
        Game1.SquareClick(0, 6)
    End Sub
```

```
Private Sub A8_Click(sender As Object, e As EventArgs) Handles A8.Click
    Game1.SquareClick(0, 7)
End Sub

Private Sub B1_Click(sender As Object, e As EventArgs) Handles B1.Click
    Game1.SquareClick(1, 0)
End Sub

Private Sub B2_Click(sender As Object, e As EventArgs) Handles B2.Click
    Game1.SquareClick(1, 1)
End Sub

Private Sub B3_Click(sender As Object, e As EventArgs) Handles B3.Click
    Game1.SquareClick(1, 2)
End Sub

Private Sub B4_Click(sender As Object, e As EventArgs) Handles B4.Click
    Game1.SquareClick(1, 3)
End Sub

Private Sub B5_Click(sender As Object, e As EventArgs) Handles B5.Click
    Game1.SquareClick(1, 4)
End Sub

Private Sub B6_Click(sender As Object, e As EventArgs) Handles B6.Click
    Game1.SquareClick(1, 5)
End Sub

Private Sub B7_Click(sender As Object, e As EventArgs) Handles B7.Click
    Game1.SquareClick(1, 6)
End Sub

Private Sub B8_Click(sender As Object, e As EventArgs) Handles B8.Click
    Game1.SquareClick(1, 7)
End Sub

Private Sub C1_Click(sender As Object, e As EventArgs) Handles C1.Click
    Game1.SquareClick(2, 0)
End Sub

Private Sub C2_Click(sender As Object, e As EventArgs) Handles C2.Click
    Game1.SquareClick(2, 1)
End Sub

Private Sub C3_Click(sender As Object, e As EventArgs) Handles C3.Click
    Game1.SquareClick(2, 2)
End Sub

Private Sub C4_Click(sender As Object, e As EventArgs) Handles C4.Click
    Game1.SquareClick(2, 3)
End Sub

Private Sub C5_Click(sender As Object, e As EventArgs) Handles C5.Click
    Game1.SquareClick(2, 4)
End Sub

Private Sub C6_Click(sender As Object, e As EventArgs) Handles C6.Click
    Game1.SquareClick(2, 5)
End Sub

Private Sub C7_Click(sender As Object, e As EventArgs) Handles C7.Click
    Game1.SquareClick(2, 6)
```

```
    End Sub

    Private Sub C8_Click(sender As Object, e As EventArgs) Handles C8.Click
        Game1.SquareClick(2, 7)
    End Sub

    Private Sub D1_Click(sender As Object, e As EventArgs) Handles D1.Click
        Game1.SquareClick(3, 0)
    End Sub

    Private Sub D2_Click(sender As Object, e As EventArgs) Handles D2.Click
        Game1.SquareClick(3, 1)
    End Sub

    Private Sub D3_Click(sender As Object, e As EventArgs) Handles D3.Click
        Game1.SquareClick(3, 2)
    End Sub

    Private Sub D4_Click(sender As Object, e As EventArgs) Handles D4.Click
        Game1.SquareClick(3, 3)
    End Sub

    Private Sub D5_Click(sender As Object, e As EventArgs) Handles D5.Click
        Game1.SquareClick(3, 4)
    End Sub

    Private Sub D6_Click(sender As Object, e As EventArgs) Handles D6.Click
        Game1.SquareClick(3, 5)
    End Sub

    Private Sub D7_Click(sender As Object, e As EventArgs) Handles D7.Click
        Game1.SquareClick(3, 6)
    End Sub

    Private Sub D8_Click(sender As Object, e As EventArgs) Handles D8.Click
        Game1.SquareClick(3, 7)
    End Sub

    Private Sub E1_Click(sender As Object, e As EventArgs) Handles E1.Click
        Game1.SquareClick(4, 0)
    End Sub

    Private Sub E2_Click(sender As Object, e As EventArgs) Handles E2.Click
        Game1.SquareClick(4, 1)
    End Sub

    Private Sub E3_Click(sender As Object, e As EventArgs) Handles E3.Click
        Game1.SquareClick(4, 2)
    End Sub

    Private Sub E4_Click(sender As Object, e As EventArgs) Handles E4.Click
        Game1.SquareClick(4, 3)
    End Sub

    Private Sub E5_Click(sender As Object, e As EventArgs) Handles E5.Click
        Game1.SquareClick(4, 4)
    End Sub

    Private Sub E6_Click(sender As Object, e As EventArgs) Handles E6.Click
        Game1.SquareClick(4, 5)
    End Sub
```

```
    Private Sub E7_Click(sender As Object, e As EventArgs) Handles E7.Click
        Game1.SquareClick(4, 6)
    End Sub

    Private Sub E8_Click(sender As Object, e As EventArgs) Handles E8.Click
        Game1.SquareClick(4, 7)
    End Sub

    Private Sub F1_Click(sender As Object, e As EventArgs) Handles F1.Click
        Game1.SquareClick(5, 0)
    End Sub

    Private Sub F2_Click(sender As Object, e As EventArgs) Handles F2.Click
        Game1.SquareClick(5, 1)
    End Sub

    Private Sub F3_Click(sender As Object, e As EventArgs) Handles F3.Click
        Game1.SquareClick(5, 2)
    End Sub

    Private Sub F4_Click(sender As Object, e As EventArgs) Handles F4.Click
        Game1.SquareClick(5, 3)
    End Sub

    Private Sub F5_Click(sender As Object, e As EventArgs) Handles F5.Click
        Game1.SquareClick(5, 4)
    End Sub

    Private Sub F6_Click(sender As Object, e As EventArgs) Handles F6.Click
        Game1.SquareClick(5, 5)
    End Sub

    Private Sub F7_Click(sender As Object, e As EventArgs) Handles F7.Click
        Game1.SquareClick(5, 6)
    End Sub

    Private Sub F8_Click(sender As Object, e As EventArgs) Handles F8.Click
        Game1.SquareClick(5, 7)
    End Sub

    Private Sub G1_Click(sender As Object, e As EventArgs) Handles G1.Click
        Game1.SquareClick(6, 0)
    End Sub

    Private Sub G2_Click(sender As Object, e As EventArgs) Handles G2.Click
        Game1.SquareClick(6, 1)
    End Sub

    Private Sub G3_Click(sender As Object, e As EventArgs) Handles G3.Click
        Game1.SquareClick(6, 2)
    End Sub

    Private Sub G4_Click(sender As Object, e As EventArgs) Handles G4.Click
        Game1.SquareClick(6, 3)
    End Sub

    Private Sub G5_Click(sender As Object, e As EventArgs) Handles G5.Click
        Game1.SquareClick(6, 4)
    End Sub

    Private Sub G6_Click(sender As Object, e As EventArgs) Handles G6.Click
        Game1.SquareClick(6, 5)
```

```
    End Sub

    Private Sub G7_Click(sender As Object, e As EventArgs) Handles G7.Click
        Game1.SquareClick(6, 6)
    End Sub

    Private Sub G8_Click(sender As Object, e As EventArgs) Handles G8.Click
        Game1.SquareClick(6, 7)
    End Sub

    Private Sub H1_Click(sender As Object, e As EventArgs) Handles H1.Click
        Game1.SquareClick(7, 0)
    End Sub

    Private Sub H2_Click(sender As Object, e As EventArgs) Handles H2.Click
        Game1.SquareClick(7, 1)
    End Sub

    Private Sub H3_Click(sender As Object, e As EventArgs) Handles H3.Click
        Game1.SquareClick(7, 2)
    End Sub

    Private Sub H4_Click(sender As Object, e As EventArgs) Handles H4.Click
        Game1.SquareClick(7, 3)
    End Sub

    Private Sub H5_Click(sender As Object, e As EventArgs) Handles H5.Click
        Game1.SquareClick(7, 4)
    End Sub

    Private Sub H6_Click(sender As Object, e As EventArgs) Handles H6.Click
        Game1.SquareClick(7, 5)
    End Sub

    Private Sub H7_Click(sender As Object, e As EventArgs) Handles H7.Click
        Game1.SquareClick(7, 6)
    End Sub

    Private Sub H8_Click(sender As Object, e As EventArgs) Handles H8.Click
        Game1.SquareClick(7, 7)
    End Sub

    Private Sub ResetButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ResetButton.Click
        Game1.InitializeGame()
        WhiteTimer.Stop()
        BlackTimer.Stop()
    End Sub

    Private Sub SaveLog_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SaveLog.Click
        'Takes input from user and uses that as a filename
        'then writes the contents of the Move Log into a file with that name on the C
root directory
        Dim FilePath1 As String = "C:\"
        Dim FilePath2 As String = InputBox("Name your log file")
        Dim FilePath3 As String = ".txt"
        Dim FilePath As String = FilePath1 + FilePath2 + FilePath3
        Dim Objwriter As New System.IO.StreamWriter(FilePath, True)
        Objwriter.Write(MoveLog.Text)
        Objwriter.Close()
    End Sub
```

```
    Private Sub ClockOff_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ClockOff.CheckedChanged
        'The input textboxes for a given setting are only set to True if the
corresponding checkbox is checked
        NormalClockInput.Enabled = False
        SpeedChessInput.Enabled = False
    End Sub

    Private Sub NormalClock_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles NormalClock.CheckedChanged
        NormalClockInput.Enabled = True
        SpeedChessInput.Enabled = False
    End Sub

    Private Sub SpeedChess_CheckedChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SpeedChess.CheckedChanged
        NormalClockInput.Enabled = False
        SpeedChessInput.Enabled = True
    End Sub

    Private Sub StartClocks_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles StartClocks.Click
        If NormalClock.Checked = True Then
            'Takes the NormalClockInput and converts it into the number of minutes for
each player
            'then starts the white timer
            Try
                WhiteTimeStore = NormalClockInput.Text * 60
                BlackTimeStore = NormalClockInput.Text * 60
                WhiteTimer.Start()
            Catch
                MsgBox("An invalid value has been input into the Normal Clock
textbox.")
            End Try
        ElseIf SpeedChess.Checked = True Then
            'Takes the input as the number of seconds for each player each turn
            Try
                WhiteTimeStore = SpeedChessInput.Text
                BlackTimeStore = SpeedChessInput.Text
                WhiteTimer.Start()
            Catch
                MsgBox("An invalid value has been input into the Speed Chess
textbox.")
            End Try
        End If
    End Sub

    Private Sub WhiteTimer_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles WhiteTimer.Tick
        'Every second, a value of 1 is taken from the WhiteTimeStore
        WhiteTimeStore = WhiteTimeStore - (WhiteTimer.Interval / 1000)
        'This formats the value of WhiteTimeStore into Minutes:Seconds and writes that
to WhiteTime
        WhiteTime.Text = Format(Math.Floor(WhiteTimeStore / 60), "00") & ":" &
Format(WhiteTimeStore Mod 60, "00")
        'If the timer goes below zero, the players are informed that one of them has
run out of time
        If WhiteTimeStore < 0 Then
            WhiteTimer.Stop()
            MsgBox("Player White has run out of time!")
        End If
```

```
    End Sub

    Private Sub BlackTimer_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BlackTimer.Tick
        'Same here as WhiteTimer_Tick
        BlackTimeStore = BlackTimeStore - (BlackTimer.Interval / 1000)
        BlackTime.Text = Format(Math.Floor(BlackTimeStore / 60), "00") & ":" &
Format(BlackTimeStore Mod 60, "00")
        If BlackTimeStore < 0 Then
            BlackTimer.Stop()
            MsgBox("Player Black has run out of time!")
        End If
    End Sub

    Private Sub Pause_Click(sender As Object, e As EventArgs) Handles Pause.Click
        'Stops whichever clock is running, and stores which clock was running
        If ClockOff.Checked = False Then
            If WhiteTimer.Enabled = True Then
                WhiteTimer.Stop()
                WhitePaused = True
            ElseIf BlackTimer.Enabled = True Then
                BlackTimer.Stop()
                WhitePaused = False
            Else
                'When the button is clicked again, the timer that was running before
starts
                If WhitePaused = True Then
                    WhiteTimer.Start()
                Else
                    BlackTimer.Start()
                End If
            End If
        End If
    End Sub
End Class
```

Game Class

```
Public Class Game
    Public Board1 As Board = New Board
    Public WPawn1 As Pawn = New Pawn
    Public WPawn2 As Pawn = New Pawn
    Public WPawn3 As Pawn = New Pawn
    Public WPawn4 As Pawn = New Pawn
    Public WPawn5 As Pawn = New Pawn
    Public WPawn6 As Pawn = New Pawn
    Public WPawn7 As Pawn = New Pawn
    Public WPawn8 As Pawn = New Pawn
    Public BPawn1 As Pawn = New Pawn
    Public BPawn2 As Pawn = New Pawn
    Public BPawn3 As Pawn = New Pawn
    Public BPawn4 As Pawn = New Pawn
    Public BPawn5 As Pawn = New Pawn
    Public BPawn6 As Pawn = New Pawn
    Public BPawn7 As Pawn = New Pawn
    Public BPawn8 As Pawn = New Pawn
    Public WBishop1 As Bishop = New Bishop
    Public WBishop2 As Bishop = New Bishop
    Public BBishop1 As Bishop = New Bishop
    Public BBishop2 As Bishop = New Bishop
    Public WKnight1 As Knight = New Knight
```

```
    Public WKnight2 As Knight = New Knight
    Public BKnight1 As Knight = New Knight
    Public BKnight2 As Knight = New Knight
    Public WRook1 As Rook = New Rook
    Public WRook2 As Rook = New Rook
    Public BRook1 As Rook = New Rook
    Public BRook2 As Rook = New Rook
    Public WKing As King = New King
    Public BKing As King = New King
    Public WQueen As Queen = New Queen
    Public BQueen As Queen = New Queen
    Dim WhiteTurn As Boolean = True 'Indicates whose turn it is; True indicates White
Turn, False therefore indicates Black Turn
    Dim SpecialMove As String 'Indicates whether a special move has been made, and if
so specifically which one
    Dim TurnCount As Integer = 0 'A counter for how many turns have gone through in
the current game
    Public Sub SquareClick(ByVal X As Integer, ByVal Y As Integer) 'This subroutine
handles the pictureboxes being clicked.
        'All event handlers call this sub, with their unique coordinates as the
parameters.
        If Chess.PieceClick = False Then 'If a piece has not already been selected
            If Chess.Grid(X, Y) <> "" Then 'the square is checked to see if there is a
piece in it or not.
                CheckTurn(X, Y) 'If there is, this subroutine is called.
            End If
        Else
            If Chess.ValidCheck(X, Y) = True Then 'If a piece has already been clicked,
and the square now being clicked
                MovePiece(X, Y) 'has been listed as a valid move, the piece will be
moved,
                ChangeTurn() 'and the turn will be changed.
            End If
            Board1.RevertColour() 'Whether a move is made or not, the colour of the
squares are returned to their
            Chess.PieceClick = False 'original colours, PieceClick is set to false,
allowing a new move to be made.
            FalsifySpecialMoves() 'Any special moves that have been marked as true are
set to false.
        End If
        Chess.StartClocks.Enabled = False
    End Sub
    Private Sub CheckTurn(ByVal X As Integer, ByVal Y As Integer) 'A list of valid
moves will only be made if the piece on
        'the square being clicked matches the current turn. This subroutines checks
that.
        Select Case WhiteTurn
            Case True
                If Chess.Grid(X, Y).StartsWith("W") = True Then 'The colour of the
piece is checked by looking at the
                    CallRules(X, Y) 'first letter of the Grid value in that square. If
it matches, the list of valid moves
                    Call Board1.DisplayValidMoves() 'will be compiled, and then
displayed on the screen
                    Chess.PieceClick = True
                End If
            Case False
                If Chess.Grid(X, Y).StartsWith("B") = True Then
                    CallRules(X, Y)
                    Call Board1.DisplayValidMoves()
                    Chess.PieceClick = True
                End If
```

```
        End Select
    End Sub
    Private Sub ChangeTurn() 'Simply changes the turn from White to Black or vice-
versa
        Select Case WhiteTurn
            Case True
                WhiteTurn = False 'Switches the indicator based on the current turn
                Chess.TurnIndicator.Text = "Black Turn" 'This is to indicate on the
board whose turn it is
                If Chess.NormalClock.Checked = True Then
                    Chess.WhiteTimer.Stop()
                    Chess.BlackTimer.Start()
                ElseIf Chess.SpeedChess.Checked = True Then
                    Chess.BlackTimeStore = Chess.SpeedChessInput.Text
                    Chess.WhiteTimer.Stop()
                    Chess.BlackTimer.Start()
                End If
            Case False
                WhiteTurn = True
                Chess.TurnIndicator.Text = "White Turn"
                If Chess.NormalClock.Checked = True Then
                    Chess.BlackTimer.Stop()
                    Chess.WhiteTimer.Start()
                ElseIf Chess.SpeedChess.Checked = True Then
                    Chess.WhiteTimeStore = Chess.SpeedChessInput.Text
                    Chess.BlackTimer.Stop()
                    Chess.WhiteTimer.Start()
                End If
        End Select
    End Sub
    Public Sub InitializeGame() 'Used to bring the game's state to its initial state
        Board1.RevertColour()
        InitializePieces()
        InitializeVariables()
        Chess.MoveLog.Text = "" 'Clears the Movelog
        Board1.ResetPiecePositions()
        Chess.ClockOff.Checked = True
        Chess.StartClocks.Enabled = True
    End Sub
    Private Sub InitializePieces() 'This sub sets all the properties of each Piece
object to their initial values
        'Setting initial properties of White Pawns
        WPawn1.IsWhite = True
        WPawn1.PositionX = 0
        WPawn1.PositionY = 1
        WPawn1.Active = True
        WPawn1.Promotion = ""
        WPawn2.IsWhite = True
        WPawn2.PositionX = 1
        WPawn2.PositionY = 1
        WPawn2.Active = True
        WPawn2.Promotion = ""
        WPawn3.IsWhite = True
        WPawn3.PositionX = 2
        WPawn3.PositionY = 1
        WPawn3.Active = True
        WPawn3.Promotion = ""
        WPawn4.IsWhite = True
        WPawn4.PositionX = 3
        WPawn4.PositionY = 1
        WPawn4.Active = True
        WPawn4.Promotion = ""
```

```
WPawn5.IsWhite = True
WPawn5.PositionX = 4
WPawn5.PositionY = 1
WPawn5.Active = True
WPawn5.Promotion = ""
WPawn6.IsWhite = True
WPawn6.PositionX = 5
WPawn6.PositionY = 1
WPawn6.Active = True
WPawn6.Promotion = ""
WPawn7.IsWhite = True
WPawn7.PositionX = 6
WPawn7.PositionY = 1
WPawn7.Active = True
WPawn7.Promotion = ""
WPawn8.IsWhite = True
WPawn8.PositionX = 7
WPawn8.PositionY = 1
WPawn8.Active = True
WPawn8.Promotion = ""
'White Bishops
WBishop1.IsWhite = True
WBishop1.PositionX = 2
WBishop1.PositionY = 0
WBishop1.Active = True
WBishop2.IsWhite = True
WBishop2.PositionX = 5
WBishop2.PositionY = 0
WBishop2.Active = True
'White Knights
WKnight1.IsWhite = True
WKnight1.PositionX = 1
WKnight1.PositionY = 0
WKnight1.Active = True
WKnight2.IsWhite = True
WKnight2.PositionX = 6
WKnight2.PositionY = 0
WKnight2.Active = True
'White Rooks
WRook1.IsWhite = True
WRook1.PositionX = 0
WRook1.PositionY = 0
WRook1.Active = True
WRook2.IsWhite = True
WRook2.PositionX = 7
WRook2.PositionY = 0
WRook2.Active = True
'White Queen
WQueen.IsWhite = True
WQueen.PositionX = 3
WQueen.PositionY = 0
WQueen.Active = True
'White King
WKing.IsWhite = True
WKing.PositionX = 4
WKing.PositionY = 0
WKing.Active = True
'Black Pawns
BPawn1.IsWhite = False
BPawn1.PositionX = 0
BPawn1.PositionY = 6
BPawn1.Active = True
```

```
BPawn1.Promotion = ""
BPawn2.IsWhite = False
BPawn2.PositionX = 1
BPawn2.PositionY = 6
BPawn2.Active = True
BPawn2.Promotion = ""
BPawn3.IsWhite = False
BPawn3.PositionX = 2
BPawn3.PositionY = 6
BPawn3.Active = True
BPawn3.Promotion = ""
BPawn4.IsWhite = False
BPawn4.PositionX = 3
BPawn4.PositionY = 6
BPawn4.Active = True
BPawn4.Promotion = ""
BPawn5.IsWhite = False
BPawn5.PositionX = 4
BPawn5.PositionY = 6
BPawn5.Active = True
BPawn5.Promotion = ""
BPawn6.IsWhite = False
BPawn6.PositionX = 5
BPawn6.PositionY = 6
BPawn6.Active = True
BPawn6.Promotion = ""
BPawn7.IsWhite = False
BPawn7.PositionX = 6
BPawn7.PositionY = 6
BPawn7.Active = True
BPawn7.Promotion = ""
BPawn8.IsWhite = False
BPawn8.PositionX = 7
BPawn8.PositionY = 6
BPawn8.Active = True
BPawn8.Promotion = ""
'Black Bishops
BBishop1.IsWhite = False
BBishop1.PositionX = 2
BBishop1.PositionY = 7
BBishop1.Active = True
BBishop2.IsWhite = False
BBishop2.PositionX = 5
BBishop2.PositionY = 7
BBishop2.Active = True
'Black Knights
BKnight1.IsWhite = False
BKnight1.PositionX = 1
BKnight1.PositionY = 7
BKnight1.Active = True
BKnight2.IsWhite = False
BKnight2.PositionX = 6
BKnight2.PositionY = 7
BKnight2.Active = True
'Black Rooks
BRook1.IsWhite = False
BRook1.PositionX = 0
BRook1.PositionY = 7
BRook1.Active = True
BRook2.IsWhite = False
BRook2.PositionX = 7
BRook2.PositionY = 7
```

```
        BRook2.Active = True
        'Black Queen
        BQueen.IsWhite = False
        BQueen.PositionX = 3
        BQueen.PositionY = 7
        BQueen.Active = True
        'Black King
        BKing.IsWhite = False
        BKing.PositionX = 4
        BKing.PositionY = 7
        BKing.Active = True
    End Sub
    Private Sub InitializeVariables() 'Sets the values of the variables in the game to
their initial values
        'Sets the turn to White for the start of the game
        If WhiteTurn = False Then
            ChangeTurn()
        End If
        'Reset TurnCount
        TurnCount = 0
        'Resetting these values to indicate that the kings and rooks have not yet
moved in the current game
        Chess.CastleWKMoved = False
        Chess.CastleBKMoved = False
        Chess.CastleWQMoved = False
        Chess.CastleBQMoved = False
        'Form1.Grid array values
        Chess.Grid(0, 0) = "WRook1"
        Chess.Grid(1, 0) = "WKnight1"
        Chess.Grid(2, 0) = "WBishop1"
        Chess.Grid(3, 0) = "WQueen"
        Chess.Grid(4, 0) = "WKing"
        Chess.Grid(5, 0) = "WBishop2"
        Chess.Grid(6, 0) = "WKnight2"
        Chess.Grid(7, 0) = "WRook2"
        Chess.Grid(0, 1) = "WPawn1"
        Chess.Grid(1, 1) = "WPawn2"
        Chess.Grid(2, 1) = "WPawn3"
        Chess.Grid(3, 1) = "WPawn4"
        Chess.Grid(4, 1) = "WPawn5"
        Chess.Grid(5, 1) = "WPawn6"
        Chess.Grid(6, 1) = "WPawn7"
        Chess.Grid(7, 1) = "WPawn8"
        Chess.Grid(0, 7) = "BRook1"
        Chess.Grid(1, 7) = "BKnight1"
        Chess.Grid(2, 7) = "BBishop1"
        Chess.Grid(3, 7) = "BQueen"
        Chess.Grid(4, 7) = "BKing"
        Chess.Grid(5, 7) = "BBishop2"
        Chess.Grid(6, 7) = "BKnight2"
        Chess.Grid(7, 7) = "BRook2"
        Chess.Grid(0, 6) = "BPawn1"
        Chess.Grid(1, 6) = "BPawn2"
        Chess.Grid(2, 6) = "BPawn3"
        Chess.Grid(3, 6) = "BPawn4"
        Chess.Grid(4, 6) = "BPawn5"
        Chess.Grid(5, 6) = "BPawn6"
        Chess.Grid(6, 6) = "BPawn7"
        Chess.Grid(7, 6) = "BPawn8"
        'Sets the Grid values of all the empty squares in the middle of the board to
nothing
        For j = 2 To 5
```

```
                For i = 0 To 7
                    Chess.Grid(i, j) = ""
                Next
            Next
            Chess.EPPosX = 0
            Chess.EPPosY = 0
            'Sets initial values of these variables to remove bugs associated with En
Passant
        End Sub
        Private Sub CallRules(ByVal X As Integer, ByVal Y As Integer)
            'Looks at the value in the Grid for the coordinates given, and calls the
CheckValidMoves sub for the corresponding object
            'then checks through the moves listed as valid to see if they put their King
in check, then list those that do as invalid
            Select Case Chess.Grid(X, Y)
                Case "BPawn1"
                    BPawn1.CheckValidMoves()
                Case "BPawn2"
                    BPawn2.CheckValidMoves()
                Case "BPawn3"
                    BPawn3.CheckValidMoves()
                Case "BPawn4"
                    BPawn4.CheckValidMoves()
                Case "BPawn5"
                    BPawn5.CheckValidMoves()
                Case "BPawn6"
                    BPawn6.CheckValidMoves()
                Case "BPawn7"
                    BPawn7.CheckValidMoves()
                Case "BPawn8"
                    BPawn8.CheckValidMoves()
                Case "BBishop1"
                    BBishop1.CheckValidMoves()
                Case "BBishop2"
                    BBishop2.CheckValidMoves()
                Case "BKnight1"
                    BKnight1.CheckValidMoves()
                Case "BKnight2"
                    BKnight2.CheckValidMoves()
                Case "BRook1"
                    BRook1.CheckValidMoves()
                Case "BRook2"
                    BRook2.CheckValidMoves()
                Case "BQueen"
                    BQueen.CheckValidMoves()
                Case "BKing"
                    BKing.CheckValidMoves()
                Case "WPawn1"
                    WPawn1.CheckValidMoves()
                Case "WPawn2"
                    WPawn2.CheckValidMoves()
                Case "WPawn3"
                    WPawn3.CheckValidMoves()
                Case "WPawn4"
                    WPawn4.CheckValidMoves()
                Case "WPawn5"
                    WPawn5.CheckValidMoves()
                Case "WPawn6"
                    WPawn6.CheckValidMoves()
                Case "WPawn7"
                    WPawn7.CheckValidMoves()
                Case "WPawn8"
```

```
                WPawn8.CheckValidMoves()
            Case "WBishop1"
                WBishop1.CheckValidMoves()
            Case "WBishop2"
                WBishop2.CheckValidMoves()
            Case "WKnight1"
                WKnight1.CheckValidMoves()
            Case "WKnight2"
                WKnight2.CheckValidMoves()
            Case "WRook1"
                WRook1.CheckValidMoves()
            Case "WRook2"
                WRook2.CheckValidMoves()
            Case "WQueen"
                WQueen.CheckValidMoves()
            Case "WKing"
                WKing.CheckValidMoves()
        End Select
        'Checks the colour of the piece to use as a parameter for the
PutSelfInCheckCheck sub
        Dim IsWhite As Boolean
        If Chess.Grid(X, Y).StartsWith("W") = True Then
            IsWhite = True
        Else
            IsWhite = False
        End If
        'Checks for squares that have been listed as valid moves, then check them
further with PutSelfInCheckCheck
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    Chess.ValidCheck(i, j) = PutSelfInCheckCheck(i, j, IsWhite)
                End If
            Next
        Next
    End Sub
    Private Sub MovePiece(ByVal X As Integer, ByVal Y As Integer)
        'This sub moves the pieces on the board (Grid values, Piece properties and
images on the board are changed)
        'It first checks to see whether a special move is being performed.
        'If so, the values are specifically changed based on which move is made.
        If Chess.CastleWK = True And X = 6 And Y = 0 Then
            Chess.E1.Image = Nothing
            Chess.F1.Image = My.Resources.White_Rook
            Chess.G1.Image = My.Resources.White_King
            Chess.H1.Image = Nothing
            Chess.Grid(4, 0) = ""
            Chess.Grid(5, 0) = "WRook2"
            Chess.Grid(6, 0) = "WKing"
            Chess.Grid(7, 0) = ""
            ChangeCoordinates(5, 0)
            ChangeCoordinates(6, 0)
            SpecialMove = "CastleWK"
            RecordMove(X, Y)
        ElseIf Chess.CastleWQ = True And X = 2 And Y = 0 Then
            Chess.A1.Image = Nothing
            Chess.C1.Image = My.Resources.White_King
            Chess.D1.Image = My.Resources.White_Rook
            Chess.E1.Image = Nothing
            Chess.Grid(0, 0) = ""
            Chess.Grid(2, 0) = "WKing"
            Chess.Grid(3, 0) = "WRook1"
```

```
            Chess.Grid(4, 0) = ""
            ChangeCoordinates(2, 0)
            ChangeCoordinates(3, 0)
            SpecialMove = "CastleWQ"
            RecordMove(X, Y)
        ElseIf Chess.CastleBK = True And X = 6 And Y = 7 Then
            Chess.E8.Image = Nothing
            Chess.F8.Image = My.Resources.Black_Rook1
            Chess.G8.Image = My.Resources.Black_King
            Chess.H8.Image = Nothing
            Chess.Grid(4, 7) = ""
            Chess.Grid(5, 7) = "BRook2"
            Chess.Grid(6, 7) = "BKing"
            Chess.Grid(7, 7) = ""
            ChangeCoordinates(5, 7)
            ChangeCoordinates(6, 7)
            SpecialMove = "CastleBK"
            RecordMove(X, Y)
        ElseIf Chess.CastleBQ = True And X = 2 And Y = 7 Then
            Chess.A8.Image = Nothing
            Chess.C8.Image = My.Resources.Black_King
            Chess.D8.Image = My.Resources.Black_Rook1
            Chess.E8.Image = Nothing
            Chess.Grid(0, 7) = ""
            Chess.Grid(2, 7) = "BKing"
            Chess.Grid(3, 7) = "BRook1"
            Chess.Grid(4, 7) = ""
            ChangeCoordinates(2, 7)
            ChangeCoordinates(3, 7)
            SpecialMove = "CastleBQ"
            RecordMove(X, Y)
        ElseIf Chess.EnPassant = True And X = Chess.EPPosX And Math.Abs(Chess.EPPosY -
Y) = 1 And Chess.X1 <> X And Chess.Grid(Chess.X1, Chess.Y1).Substring(1, 1) = "P" Then
            RecordMove(X, Y)
            ChangeActive(Chess.EPPosX, Chess.EPPosY)
            Board1.EnPassantImageChange(X, Y)
            Chess.Grid(X, Y) = Chess.Grid(Chess.X1, Chess.Y1)
            Chess.Grid(Chess.X1, Chess.Y1) = ""
            Chess.Grid(Chess.EPPosX, Chess.EPPosY) = ""
            ChangeCoordinates(X, Y)
        Else
            'If the move is not a special move, the following functions will change
the values using the following subs
            RecordMove(X, Y)
            CastleNull()
            If Chess.Grid(X, Y) <> "" Then
                ChangeActive(X, Y)
            End If
            Board1.ImageChange(X, Y)
            'Changes the grid value of the destination position to that of the initial
position,
            'then sets the initial position to nothing
            Chess.Grid(X, Y) = Chess.Grid(Chess.X1, Chess.Y1)
            Chess.Grid(Chess.X1, Chess.Y1) = ""
            ChangeCoordinates(X, Y)
        End If
        PawnPromotion(X, Y)
        Chess.EnPassant = False
        If Chess.Grid(X, Y).Substring(1, 1) = "P" Then
            If Chess.X1 = X And Math.Abs(Y - Chess.Y1) = 2 Then
                Chess.EnPassant = True
                Chess.EPPosX = X
```

```
                Chess.EPPosY = Y
            End If
        End If
        'If the King has been put into check, checkmate is then checked for.
        If CheckCheck(Not WhiteTurn) = True Then
            CheckMateCheck()
        End If
    End Sub
    Private Sub ChangeCoordinates(ByVal X As Integer, ByVal Y As Integer)
        'After the Grid value has been changed, it is then used to change the values
of the coordinates of the Piece objects
        Select Case Chess.Grid(X, Y)
            Case "BPawn1"
                BPawn1.PositionX = X
                BPawn1.PositionY = Y
            Case "BPawn2"
                BPawn2.PositionX = X
                BPawn2.PositionY = Y
            Case "BPawn3"
                BPawn3.PositionX = X
                BPawn3.PositionY = Y
            Case "BPawn4"
                BPawn4.PositionX = X
                BPawn4.PositionY = Y
            Case "BPawn5"
                BPawn5.PositionX = X
                BPawn5.PositionY = Y
            Case "BPawn6"
                BPawn6.PositionX = X
                BPawn6.PositionY = Y
            Case "BPawn7"
                BPawn7.PositionX = X
                BPawn7.PositionY = Y
            Case "BPawn8"
                BPawn8.PositionX = X
                BPawn8.PositionY = Y
            Case "BBishop1"
                BBishop1.PositionX = X
                BBishop1.PositionY = Y
            Case "BBishop2"
                BBishop2.PositionX = X
                BBishop2.PositionY = Y
            Case "BKnight1"
                BKnight1.PositionX = X
                BKnight1.PositionY = Y
            Case "BKnight2"
                BKnight2.PositionX = X
                BKnight2.PositionY = Y
            Case "BRook1"
                BRook1.PositionX = X
                BRook1.PositionY = Y
            Case "BRook2"
                BRook2.PositionX = X
                BRook2.PositionY = Y
            Case "BQueen"
                BQueen.PositionX = X
                BQueen.PositionY = Y
            Case "BKing"
                BKing.PositionX = X
                BKing.PositionY = Y
            Case "WPawn1"
                WPawn1.PositionX = X
```

```
                WPawn1.PositionY = Y
            Case "WPawn2"
                WPawn2.PositionX = X
                WPawn2.PositionY = Y
            Case "WPawn3"
                WPawn3.PositionX = X
                WPawn3.PositionY = Y
            Case "WPawn4"
                WPawn4.PositionX = X
                WPawn4.PositionY = Y
            Case "WPawn5"
                WPawn5.PositionX = X
                WPawn5.PositionY = Y
            Case "WPawn6"
                WPawn6.PositionX = X
                WPawn6.PositionY = Y
            Case "WPawn7"
                WPawn7.PositionX = X
                WPawn7.PositionY = Y
            Case "WPawn8"
                WPawn8.PositionX = X
                WPawn8.PositionY = Y
            Case "WBishop1"
                WBishop1.PositionX = X
                WBishop1.PositionY = Y
            Case "WBishop2"
                WBishop2.PositionX = X
                WBishop2.PositionY = Y
            Case "WKnight1"
                WKnight1.PositionX = X
                WKnight1.PositionY = Y
            Case "WKnight2"
                WKnight2.PositionX = X
                WKnight2.PositionY = Y
            Case "WRook1"
                WRook1.PositionX = X
                WRook1.PositionY = Y
            Case "WRook2"
                WRook2.PositionX = X
                WRook2.PositionY = Y
            Case "WQueen"
                WQueen.PositionX = X
                WQueen.PositionY = Y
            Case "WKing"
                WKing.PositionX = X
                WKing.PositionY = Y
        End Select
    End Sub
    Private Sub RecordMove(ByVal X As Integer, ByVal Y As Integer)
        'This sub creates a string, which is the chess notation of the move that was
just made,
        'and adds it to the Move Log
        Dim ChessNotation As String = ""
        'Special moves have specific chess notation
        Select Case SpecialMove
            Case "CastleWK"
                ChessNotation = "O-O"
            Case "CastleWQ"
                ChessNotation = "O-O-O"
            Case "CastleBK"
                ChessNotation = "O-O"
            Case "CastleBQ"
```

```
                ChessNotation = "O-O-O"
            Case ""
                'Adds the letter denoting which piece is being moved
                If Chess.Grid(Chess.X1, Chess.Y1).Substring(1, 2) = "Kn" Then
                    ChessNotation = "N"
                Else
                    ChessNotation = Chess.Grid(Chess.X1, Chess.Y1).Substring(1, 1)
                End If
                'If the move involves a capture, an "x" is added at this point
                If Chess.Grid(X, Y) <> "" Then
                    ChessNotation = ChessNotation + "x"
                End If
                'This adds the position being moved to on to the end of the string
                ChessNotation = ChessNotation +
Board1.NumberToLetter(X).ToString.ToLower + (Y + 1).ToString
        End Select
        'After each player has made a move, the turncount goes up by one

        If WhiteTurn = True Then
            TurnCount += 1
            Chess.MoveLog.Text = Chess.MoveLog.Text & vbNewLine & TurnCount & "."
        End If
        Chess.MoveLog.Text = Chess.MoveLog.Text & " " & ChessNotation
        SpecialMove = ""
    End Sub
    Private Sub CastleNull() 'Marks a castling move as invalid if either of the pieces
involved in the castle moves
        Select Case Chess.Grid(Chess.X1, Chess.Y1)
            Case "WRook1"
                Chess.CastleWQMoved = True
            Case "WKing"
                Chess.CastleWQMoved = True
                Chess.CastleWKMoved = True
            Case "WRook2"
                Chess.CastleWKMoved = True
            Case "BRook1"
                Chess.CastleBQMoved = True
            Case "BKing"
                Chess.CastleBQMoved = True
                Chess.CastleBKMoved = True
            Case "BRook2"
                Chess.CastleBKMoved = True
        End Select
    End Sub
    Private Sub FalsifySpecialMoves() 'Resets special move checks so that they do not
always appear as valid after they have been marked as valid once.
        Chess.CastleWK = False
        Chess.CastleWQ = False
        Chess.CastleBK = False
        Chess.CastleBQ = False
    End Sub
    Private Function CheckCheck(ByVal KingColourWhite As Boolean)
        'Checks whether a move puts a King in check, returns true or false
        Dim X As Integer
        Dim Y As Integer
        Dim Check As Boolean = False
        Select Case KingColourWhite
            Case True
                'Checking if the White King is in check, checking whether any black
pieces are in a position to capture it
                X = WKing.PositionX
                Y = WKing.PositionY
```

```
            If BKnight1.Rules(X, Y) = True And BKnight1.Active = True Then
                Check = True
            ElseIf BKnight2.Rules(X, Y) = True And BKnight2.Active = True Then
                Check = True
            ElseIf BBishop1.Rules(X, Y) = True And BBishop1.Active = True Then
                Check = True
            ElseIf BBishop2.Rules(X, Y) = True And BBishop2.Active = True Then
                Check = True
            ElseIf BRook1.Rules(X, Y) = True And BRook1.Active = True Then
                Check = True
            ElseIf BRook2.Rules(X, Y) = True And BRook2.Active = True Then
                Check = True
            ElseIf BQueen.Rules(X, Y) = True And BQueen.Active = True Then
                Check = True
            ElseIf BPawn1.Rules(X, Y) = True And BPawn1.Active = True Then
                Check = True
            ElseIf BPawn2.Rules(X, Y) = True And BPawn2.Active = True Then
                Check = True
            ElseIf BPawn3.Rules(X, Y) = True And BPawn3.Active = True Then
                Check = True
            ElseIf BPawn4.Rules(X, Y) = True And BPawn4.Active = True Then
                Check = True
            ElseIf BPawn5.Rules(X, Y) = True And BPawn5.Active = True Then
                Check = True
            ElseIf BPawn6.Rules(X, Y) = True And BPawn6.Active = True Then
                Check = True
            ElseIf BPawn7.Rules(X, Y) = True And BPawn7.Active = True Then
                Check = True
            ElseIf BPawn8.Rules(X, Y) = True And BPawn8.Active = True Then
                Check = True
            End If
        Case False
            'Same here fore the Black King, checking all White Pieces.
            X = BKing.PositionX
            Y = BKing.PositionY
            If WKnight1.Rules(X, Y) = True And WKnight1.Active = True Then
                Check = True
            ElseIf WKnight2.Rules(X, Y) = True And WKnight2.Active = True Then
                Check = True
            ElseIf WBishop1.Rules(X, Y) = True And WBishop1.Active = True Then
                Check = True
            ElseIf WBishop2.Rules(X, Y) = True And WBishop2.Active = True Then
                Check = True
            ElseIf WRook1.Rules(X, Y) = True And WRook1.Active = True Then
                Check = True
            ElseIf WRook2.Rules(X, Y) = True And WRook2.Active = True Then
                Check = True
            ElseIf WQueen.Rules(X, Y) = True And WQueen.Active = True Then
                Check = True
            ElseIf WPawn1.Rules(X, Y) = True And WPawn1.Active = True Then
                Check = True
            ElseIf WPawn2.Rules(X, Y) = True And WPawn2.Active = True Then
                Check = True
            ElseIf WPawn3.Rules(X, Y) = True And WPawn3.Active = True Then
                Check = True
            ElseIf WPawn4.Rules(X, Y) = True And WPawn4.Active = True Then
                Check = True
            ElseIf WPawn5.Rules(X, Y) = True And WPawn5.Active = True Then
                Check = True
            ElseIf WPawn6.Rules(X, Y) = True And WPawn6.Active = True Then
                Check = True
            ElseIf WPawn7.Rules(X, Y) = True And WPawn7.Active = True Then
```

```
                    Check = True
                ElseIf WPawn8.Rules(X, Y) = True And WPawn8.Active = True Then
                    Check = True
                End If
        End Select
        Return Check
    End Function
    Private Sub ChangeActive(ByVal X As Integer, ByVal Y As Integer)
        'Simply switches a piece's active state from true to false or vice-versa
        'A non-active piece is one that has been captured. This sub is used when a
piece has been captured, or for
        'simulating a capture when checking whether a move puts the player's own King
in check
        Select Case Chess.Grid(X, Y)
            Case "BPawn1"
                BPawn1.Active = Not BPawn1.Active
            Case "BPawn2"
                BPawn2.Active = Not BPawn2.Active
            Case "BPawn3"
                BPawn3.Active = Not BPawn3.Active
            Case "BPawn4"
                BPawn4.Active = Not BPawn4.Active
            Case "BPawn5"
                BPawn5.Active = Not BPawn5.Active
            Case "BPawn6"
                BPawn6.Active = Not BPawn6.Active
            Case "BPawn7"
                BPawn7.Active = Not BPawn7.Active
            Case "BPawn8"
                BPawn8.Active = Not BPawn8.Active
            Case "BBishop1"
                BBishop1.Active = Not BBishop1.Active
            Case "BBishop2"
                BBishop2.Active = Not BBishop2.Active
            Case "BKnight1"
                BKnight1.Active = Not BKnight1.Active
            Case "BKnight2"
                BKnight2.Active = Not BKnight2.Active
            Case "BRook1"
                BRook1.Active = Not BRook1.Active
            Case "BRook2"
                BRook2.Active = Not BRook2.Active
            Case "BQueen"
                BQueen.Active = Not BQueen.Active
            Case "WPawn1"
                WPawn1.Active = Not WPawn1.Active
            Case "WPawn2"
                WPawn2.Active = Not WPawn2.Active
            Case "WPawn3"
                WPawn3.Active = Not WPawn3.Active
            Case "WPawn4"
                WPawn4.Active = Not WPawn4.Active
            Case "WPawn5"
                WPawn5.Active = Not WPawn5.Active
            Case "WPawn6"
                WPawn6.Active = Not WPawn6.Active
            Case "WPawn7"
                WPawn7.Active = Not WPawn7.Active
            Case "WPawn8"
                WPawn8.Active = Not WPawn8.Active
            Case "WBishop1"
                WBishop1.Active = Not WBishop1.Active
```

```
            Case "WBishop2"
                WBishop2.Active = Not WBishop2.Active
            Case "WKnight1"
                WKnight1.Active = Not WKnight1.Active
            Case "WKnight2"
                WKnight2.Active = Not WKnight2.Active
            Case "WRook1"
                WRook1.Active = Not WRook1.Active
            Case "WRook2"
                WRook2.Active = Not WRook2.Active
            Case "WQueen"
                WQueen.Active = Not WQueen.Active
        End Select
    End Sub
    Private Function PutSelfInCheckCheck(ByVal X As Integer, ByVal Y As Integer, ByVal
IsWhite As Boolean)
        'Checks whether a move puts the player making the move in check, returns true
or false.
        Dim IniPos, FinPos As String
        Dim Valid As Boolean
        'Stores what is in these spaces to restore later
        IniPos = Chess.Grid(Chess.X1, Chess.Y1)
        FinPos = Chess.Grid(X, Y)
        'If there is a piece that is capturable in the destination position, it is
changed to inactive for
        'checking what the state of the board would be after this move is made.
        ChangeActive(X, Y)
        Chess.Grid(Chess.X1, Chess.Y1) = ""
        Chess.Grid(X, Y) = IniPos
        If IniPos = "WKing" Then
            WKing.PositionX = X
            WKing.PositionY = Y
        ElseIf IniPos = "BKing" Then
            BKing.PositionX = X
            BKing.PositionY = Y
        End If
        'Checks whether the player's own King is in check after the move being checked
is made
        Valid = Not CheckCheck(IsWhite)
        'Returns board to previous state
        Chess.Grid(Chess.X1, Chess.Y1) = IniPos
        Chess.Grid(X, Y) = FinPos
        ChangeActive(X, Y)
        If IniPos = "WKing" Then
            WKing.PositionX = Chess.X1
            WKing.PositionY = Chess.Y1
        ElseIf IniPos = "BKing" Then
            BKing.PositionX = Chess.X1
            BKing.PositionY = Chess.Y1
        End If
        Return Valid
    End Function
    Private Sub CheckMateCheck()
        'This sub is called when a King is put in check
        'This checks every piece to see if it can make a move that will leave the
player's King out of check
        'If such a move can be made, CheckMate is marked as false
        Dim CheckMate As Boolean
        CheckMate = True
        Select Case WhiteTurn
            Case True
                If BPawn1.Active = True Then
```

```
        CallRules(BPawn1.PositionX, BPawn1.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If BPawn2.Active = True Then
        CallRules(BPawn2.PositionX, BPawn2.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If BPawn3.Active = True Then
        CallRules(BPawn3.PositionX, BPawn3.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If BPawn4.Active = True Then
        CallRules(BPawn4.PositionX, BPawn4.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If BPawn5.Active = True Then
        CallRules(BPawn5.PositionX, BPawn5.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If BPawn6.Active = True Then
        CallRules(BPawn6.PositionX, BPawn6.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If BPawn7.Active = True Then
        CallRules(BPawn7.PositionX, BPawn7.PositionY)
        For j = 0 To 7
```

```
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If BPawn8.Active = True Then
            CallRules(BPawn8.PositionX, BPawn8.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If BBishop1.Active = True Then
            CallRules(BBishop1.PositionX, BBishop1.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If BBishop2.Active = True Then
            CallRules(BBishop2.PositionX, BBishop2.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If BKnight1.Active = True Then
            CallRules(BKnight1.PositionX, BKnight1.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If BKnight2.Active = True Then
            CallRules(BKnight2.PositionX, BKnight2.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If BRook1.Active = True Then
            CallRules(BRook1.PositionX, BRook1.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
```

```
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If BRook2.Active = True Then
            CallRules(BRook2.PositionX, BRook2.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If BQueen.Active = True Then
            CallRules(BQueen.PositionX, BQueen.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        CallRules(BKing.PositionX, BKing.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    Case False
        If WPawn1.Active = True Then
            CallRules(WPawn1.PositionX, WPawn1.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If WPawn2.Active = True Then
            CallRules(WPawn2.PositionX, WPawn2.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
            Next
        End If
        If WPawn3.Active = True Then
            CallRules(WPawn3.PositionX, WPawn3.PositionY)
            For j = 0 To 7
                For i = 0 To 7
                    If Chess.ValidCheck(i, j) = True Then
                        CheckMate = False
                    End If
                Next
```

```
        Next
    End If
    If WPawn4.Active = True Then
        CallRules(WPawn4.PositionX, WPawn4.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If WPawn5.Active = True Then
        CallRules(WPawn5.PositionX, WPawn5.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If WPawn6.Active = True Then
        CallRules(WPawn6.PositionX, WPawn6.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If WPawn7.Active = True Then
        CallRules(WPawn7.PositionX, WPawn7.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If WPawn8.Active = True Then
        CallRules(WPawn8.PositionX, WPawn8.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
    If WBishop1.Active = True Then
        CallRules(WBishop1.PositionX, WBishop1.PositionY)
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    CheckMate = False
                End If
            Next
        Next
    End If
```

```
If WBishop2.Active = True Then
    CallRules(WBishop2.PositionX, WBishop2.PositionY)
    For j = 0 To 7
        For i = 0 To 7
            If Chess.ValidCheck(i, j) = True Then
                CheckMate = False
            End If
        Next
    Next
End If
If WKnight1.Active = True Then
    CallRules(WKnight1.PositionX, WKnight1.PositionY)
    For j = 0 To 7
        For i = 0 To 7
            If Chess.ValidCheck(i, j) = True Then
                CheckMate = False
            End If
        Next
    Next
End If
If WKnight2.Active = True Then
    CallRules(WKnight2.PositionX, WKnight2.PositionY)
    For j = 0 To 7
        For i = 0 To 7
            If Chess.ValidCheck(i, j) = True Then
                CheckMate = False
            End If
        Next
    Next
End If
If WRook1.Active = True Then
    CallRules(WRook1.PositionX, WRook1.PositionY)
    For j = 0 To 7
        For i = 0 To 7
            If Chess.ValidCheck(i, j) = True Then
                CheckMate = False
            End If
        Next
    Next
End If
If WRook2.Active = True Then
    CallRules(WRook2.PositionX, WRook2.PositionY)
    For j = 0 To 7
        For i = 0 To 7
            If Chess.ValidCheck(i, j) = True Then
                CheckMate = False
            End If
        Next
    Next
End If
If WQueen.Active = True Then
    CallRules(WQueen.PositionX, WQueen.PositionY)
    For j = 0 To 7
        For i = 0 To 7
            If Chess.ValidCheck(i, j) = True Then
                CheckMate = False
            End If
        Next
    Next
End If
CallRules(WKing.PositionX, WKing.PositionY)
For j = 0 To 7
```

```
                    For i = 0 To 7
                        If Chess.ValidCheck(i, j) = True Then
                            CheckMate = False
                        End If
                    Next
                Next
        End Select
        'If it is Checkmate, the players are informed, and a # is appended to the end
of the notation
        'If it is not, a + is appended to indicate check
        If CheckMate = True Then
            Chess.WhiteTimer.Stop()
            Chess.BlackTimer.Stop()
            MsgBox("CheckMate")
            Chess.MoveLog.Text = Chess.MoveLog.Text + "#"
        Else
            Chess.MoveLog.Text = Chess.MoveLog.Text + "+"
        End If
    End Sub
    Private Sub PawnPromotion(X, Y)
        'If the piece being moved is a pawn and that pawn has not already been
promoted,
        'the corresponding object's "Promote" subroutine is called, and then if the
pawn is promoted, the image is changed
        If Chess.Grid(X, Y).Substring(1, 1) = "P" Then 'This checks if the piece is a
pawn
            'Checks which pawn it is, and calls the appropriate Promotion sub and then
the image change
            Select Case Chess.Grid(X, Y)
                Case "BPawn1"
                    If BPawn1.Promotion = Nothing Then
                        BPawn1.Promote(X, Y)
                    End If
                    Board1.PromotedPawnImageChange(X, Y, BPawn1.Promotion,
BPawn1.IsWhite)
                Case "BPawn2"
                    If BPawn2.Promotion = Nothing Then
                        BPawn2.Promote(X, Y)
                    End If
                    Board1.PromotedPawnImageChange(X, Y, BPawn2.Promotion,
BPawn2.IsWhite)
                Case "BPawn3"
                    If BPawn3.Promotion = Nothing Then
                        BPawn3.Promote(X, Y)
                    End If
                    Board1.PromotedPawnImageChange(X, Y, BPawn3.Promotion,
BPawn3.IsWhite)
                Case "BPawn4"
                    If BPawn4.Promotion = Nothing Then
                        BPawn4.Promote(X, Y)
                    End If
                    Board1.PromotedPawnImageChange(X, Y, BPawn4.Promotion,
BPawn4.IsWhite)
                Case "BPawn5"
                    If BPawn5.Promotion = Nothing Then
                        BPawn5.Promote(X, Y)
                    End If
                    Board1.PromotedPawnImageChange(X, Y, BPawn5.Promotion,
BPawn5.IsWhite)
                Case "BPawn6"
                    If BPawn6.Promotion = Nothing Then
                        BPawn6.Promote(X, Y)
```

```
                End If
                Board1.PromotedPawnImageChange(X, Y, BPawn6.Promotion,
BPawn6.IsWhite)
            Case "BPawn7"
                If BPawn7.Promotion = Nothing Then
                    BPawn7.Promote(X, Y)
                End If
                Board1.PromotedPawnImageChange(X, Y, BPawn7.Promotion,
BPawn7.IsWhite)
            Case "BPawn8"
                If BPawn8.Promotion = Nothing Then
                    BPawn8.Promote(X, Y)
                End If
                Board1.PromotedPawnImageChange(X, Y, BPawn8.Promotion,
BPawn8.IsWhite)
            Case "WPawn1"
                If WPawn1.Promotion = Nothing Then
                    MsgBox("B")
                    WPawn1.Promote(X, Y)
                End If
                Board1.PromotedPawnImageChange(X, Y, WPawn1.Promotion,
WPawn1.IsWhite)
            Case "WPawn2"
                If WPawn2.Promotion = Nothing Then
                    WPawn2.Promote(X, Y)
                End If
                Board1.PromotedPawnImageChange(X, Y, WPawn2.Promotion,
WPawn2.IsWhite)
            Case "WPawn3"
                If WPawn3.Promotion = Nothing Then
                    WPawn3.Promote(X, Y)
                End If
                Board1.PromotedPawnImageChange(X, Y, WPawn3.Promotion,
WPawn3.IsWhite)
            Case "WPawn4"
                If WPawn4.Promotion = Nothing Then
                    WPawn4.Promote(X, Y)
                End If
                Board1.PromotedPawnImageChange(X, Y, WPawn4.Promotion,
WPawn4.IsWhite)
            Case "WPawn5"
                If WPawn5.Promotion = Nothing Then
                    WPawn5.Promote(X, Y)
                End If
                Board1.PromotedPawnImageChange(X, Y, WPawn5.Promotion,
WPawn5.IsWhite)
            Case "WPawn6"
                If WPawn6.Promotion = Nothing Then
                    WPawn6.Promote(X, Y)
                End If
                Board1.PromotedPawnImageChange(X, Y, WPawn6.Promotion,
WPawn6.IsWhite)
            Case "WPawn7"
                If WPawn7.Promotion = Nothing Then
                    WPawn7.Promote(X, Y)
                End If
                Board1.PromotedPawnImageChange(X, Y, WPawn7.Promotion,
WPawn7.IsWhite)
            Case "WPawn8"
                If WPawn8.Promotion = Nothing Then
                    WPawn8.Promote(X, Y)
                End If
```

```
                    Board1.PromotedPawnImageChange(X, Y, WPawn8.Promotion,
WPawn8.IsWhite)
                End Select
            End If
        End Sub
End Class
```

Board Class

```
Public Class Board
    Public Sub DisplayValidMoves()
        'Looks for all valid moves, then highlights those
        'The square that was clicked is indicated in green
        'If there is a Castle move valid, that is indicated in red
        For j = 0 To 7
            For i = 0 To 7
                If Chess.ValidCheck(i, j) = True Then
                    Highlight(i, j)
                End If
            Next
        Next
        Dim IniPosition As String = NumberToLetter(Chess.X1)
        IniPosition = IniPosition + (Chess.Y1 + 1).ToString
        Chess.Controls(IniPosition).BackColor = Color.Green
        If Chess.CastleWK = True Then
            Chess.G1.BackColor = Color.Red
        ElseIf Chess.CastleWQ = True Then
            Chess.C1.BackColor = Color.Red
        ElseIf Chess.CastleBK = True Then
            Chess.G8.BackColor = Color.Red
        ElseIf Chess.CastleBQ = True Then
            Chess.C8.BackColor = Color.Red
        End If
    End Sub
    Private Sub Highlight(ByVal X As Integer, ByVal Y As Integer)
        'Changes the background colour of a given square to yellow
        Dim str As String = NumberToLetter(X)
        str = str + (Y + 1).ToString
        Chess.Controls(str).BackColor = Color.Yellow
    End Sub
    Public Sub RevertColour()
        'Changes the background colour of all squares to their default colours.
        Dim Square As String
        For j = 1 To 8
            For i = 0 To 7
                Square = NumberToLetter(i)
                Square = Square + j.ToString
                If (i Mod 2 = 1 And j Mod 2 = 0) Or (i Mod 2 = 0 And j Mod 2 = 1) Then
                    Chess.Controls(Square).BackColor = Color.DimGray
                ElseIf (i Mod 2 = 0 And j Mod 2 = 0) Or (i Mod 2 = 1 And j Mod 2 = 1)
Then
                    Chess.Controls(Square).BackColor = Color.White
                End If
            Next
        Next
    End Sub
    Public Function NumberToLetter(ByVal X As Integer)
        'Converts an X coordinate into the corresponding letter for the name of a
PictureBox
        Dim Str As String = ""
        Select Case X
            Case 0
```

```
                Str = "A"
            Case 1
                Str = "B"
            Case 2
                Str = "C"
            Case 3
                Str = "D"
            Case 4
                Str = "E"
            Case 5
                Str = "F"
            Case 6
                Str = "G"
            Case 7
                Str = "H"
        End Select
        Return Str
    End Function
    Public Sub ImageChange(ByVal X As Integer, ByVal Y As Integer)
        'Sets the image of the destination square to whatever was in the initial
square
        'then sets the image of the initial square to nothing
        Dim DestSquare As String
        Dim IniSquare As String
        DestSquare = NumberToLetter(X) + (Y + 1).ToString
        IniSquare = NumberToLetter(Chess.X1) + (Chess.Y1 + 1).ToString
        Select Case Chess.Grid(Chess.X1, Chess.Y1).TrimEnd("1", "2", "3", "4", "5",
"6", "7", "8")
            Case "WPawn"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.White_Pawn
            Case "BPawn"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.Black_Pawn
            Case "WBishop"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.White_Bishop
            Case "BBishop"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.Black_Bishop
            Case "WKnight"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.White_Knight
            Case "BKnight"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.Black_Knight1
            Case "WRook"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.White_Rook
            Case "BRook"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.Black_Rook1
            Case "WQueen"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.White_Queen
            Case "BQueen"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.Black_Queen
            Case "WKing"
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.White_King
            Case "BKing"
```

```
                DirectCast(Chess.Controls(DestSquare), PictureBox).Image =
My.Resources.Black_King
        End Select
        DirectCast(Chess.Controls(IniSquare), PictureBox).Image = Nothing
    End Sub
    Public Sub ResetPiecePositions()
        'Sets the images of all PictureBoxes to their initial states
        Dim Square As String
        Chess.A1.Image = My.Resources.White_Rook
        Chess.B1.Image = My.Resources.White_Knight
        Chess.C1.Image = My.Resources.White_Bishop
        Chess.D1.Image = My.Resources.White_Queen
        Chess.E1.Image = My.Resources.White_King
        Chess.F1.Image = My.Resources.White_Bishop
        Chess.G1.Image = My.Resources.White_Knight
        Chess.H1.Image = My.Resources.White_Rook
        For i = 0 To 7
            Square = NumberToLetter(i) & "2"
            DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.White_Pawn
        Next
        Chess.A8.Image = My.Resources.Black_Rook1
        Chess.B8.Image = My.Resources.Black_Knight1
        Chess.C8.Image = My.Resources.Black_Bishop
        Chess.D8.Image = My.Resources.Black_Queen
        Chess.E8.Image = My.Resources.Black_King
        Chess.F8.Image = My.Resources.Black_Bishop
        Chess.G8.Image = My.Resources.Black_Knight1
        Chess.H8.Image = My.Resources.Black_Rook1
        For i = 0 To 7
            Square = NumberToLetter(i) & "7"
            DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.Black_Pawn
        Next
        For j = 3 To 6
            For i = 0 To 7
                Square = NumberToLetter(i) & j
                DirectCast(Chess.Controls(Square), PictureBox).Image = Nothing
            Next
        Next
    End Sub
    Public Sub PromotedPawnImageChange(ByVal X As Integer, ByVal Y As Integer, ByVal
Promote As Char, ByVal IsWhite As Boolean)
        'Used to change a Pawn's image to whatever it has been promoted to, if it has
been promoted
        Dim Square As String
        Square = NumberToLetter(X) & (Y + 1).ToString
        Select Case IsWhite
            Case True
                Select Case Promote
                    Case "Q"
                        DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.White_Queen
                    Case "N"
                        DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.White_Knight
                    Case "R"
                        DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.White_Rook
                    Case "B"
                        DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.White_Bishop
```

```
                End Select
            Case False
                Select Case Promote
                    Case "Q"
                        DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.Black_Queen
                    Case "N"
                        DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.Black_Knight1
                    Case "R"
                        DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.Black_Rook1
                    Case "B"
                        DirectCast(Chess.Controls(Square), PictureBox).Image =
My.Resources.Black_Bishop
                End Select
        End Select
    End Sub
    Public Sub EnPassantImageChange(X, Y)
        'Image change when using En Passant
        'Usual ImageChange is called, then the image of the pawn being captured is
removed
        ImageChange(X, Y)
        Dim Square As String = NumberToLetter(Chess.EPPosX) + (Chess.EPPosY +
1).ToString
        DirectCast(Chess.Controls(Square), PictureBox).Image = Nothing
    End Sub
End Class
```

Piece Class

```
Public Class Piece
    Property IsWhite As Boolean
    Property PositionX As Integer
    Property PositionY As Integer
    Property Active As Boolean
    Sub CheckSpaces(ByVal PositionX As Integer, ByVal PositionY As Integer, ByVal X3
As Integer, ByVal Y3 As Integer, ByRef Valid As Boolean)
        'Used to check if there are pieces in the spaces between the starting and
destination positions
        If (X3 = 1 Or X3 = 0) And (Y3 = 1 Or Y3 = 0) Then
            Valid = True
        Else
            If Math.Abs(X3) = Math.Abs(Y3) Then
                If X3 > 0 And Y3 > 0 Then
                    For i = 1 To X3 - 1
                        If Chess.Grid(PositionX + i, PositionY + i) <> "" Then
                            Valid = False
                        End If
                    Next
                ElseIf X3 > 0 And Y3 < 0 Then
                    For i = 1 To X3 - 1
                        If Chess.Grid(PositionX + i, PositionY - i) <> "" Then
                            Valid = False
                        End If
                    Next
                ElseIf X3 < 0 And Y3 > 0 Then
                    For i = 1 To Y3 - 1
                        If Chess.Grid(PositionX - i, PositionY + i) <> "" Then
                            Valid = False
                        End If
```

```
                    Next
                ElseIf X3 < 0 And Y3 < 0 Then
                    For i = 1 To Math.Abs(X3) - 1
                        If Chess.Grid(PositionX - i, PositionY - i) <> "" Then
                            Valid = False
                        End If
                    Next
                End If
            ElseIf Math.Abs(X3) = 0 Then
                If Y3 > 0 Then
                    For i = 1 To Y3 - 1
                        If Chess.Grid(PositionX, PositionY + i) <> "" Then
                            Valid = False
                        End If
                    Next
                ElseIf Y3 < 0 Then
                    For i = 1 To Math.Abs(Y3) - 1
                        If Chess.Grid(PositionX, PositionY - i) <> "" Then
                            Valid = False
                        End If
                    Next
                End If
            ElseIf Math.Abs(Y3) = 0 Then
                If X3 > 0 Then
                    For i = 1 To X3 - 1
                        If Chess.Grid(PositionX + i, PositionY) <> "" Then
                            Valid = False
                        End If
                    Next
                ElseIf X3 < 0 Then
                    For i = 1 To Math.Abs(X3) - 1
                        If Chess.Grid(PositionX - i, PositionY) <> "" Then
                            Valid = False
                        End If
                    Next
                End If
            End If
        End If
    End Sub
    Sub CheckDestination(ByVal X2 As Integer, ByVal Y2 As Integer, ByRef Valid As
Boolean)
        'Used to check whether there is a piece at the destination position, and if
there is, whether it is capturable
        Select Case IsWhite
            Case True
                If Chess.Grid(X2, Y2).StartsWith("W") = True Then
                    Valid = False
                End If
            Case False
                If Chess.Grid(X2, Y2).StartsWith("B") = True Then
                    Valid = False
                End If
        End Select
    End Sub
End Class
```

Pawn Class

```
Public Class Pawn : Inherits Piece
    Property Promotion As Char = ""
```

```
    Public Sub CheckValidMoves()
        'Checks all spaces on the board for valid moves
        Dim Valid As Boolean
        Chess.X1 = PositionX
        Chess.Y1 = PositionY
        For j = 0 To 7
            For i = 0 To 7
                Valid = Rules(i, j)
                If Valid = True Then
                    Chess.ValidCheck(i, j) = True
                Else
                    Chess.ValidCheck(i, j) = False
                End If
            Next
        Next
        EnPassantCheck()
    End Sub
    Public Function Rules(ByVal X2 As Integer, ByVal Y2 As Integer)
        'Checks whether a given move can be performed for a given pawn
        Dim X3 As Integer = X2 - PositionX
        Dim Y3 As Integer = Y2 - PositionY
        Dim Valid As Boolean
        Select Case Promotion
            Case ""
                Select Case IsWhite
                    Case True
                        If X3 = 0 And Y3 = 2 And PositionY = 1 Then
                            Valid = True 'Sets the Valid variable to True initially
                            Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
'Since the piece is moving 2 spaces, the square between the starting and destination
squares needs to be checked for a piece.
                            Call PawnMove(X2, Y2, Valid) 'Pawns have different rules
for moving and attacking, so separate subroutines will be created to check the
destination square for different types of movement.
                        ElseIf X3 = 0 And Y3 = 1 Then
                            Valid = True
                            Call PawnMove(X2, Y2, Valid)
                        ElseIf Math.Abs(X3) = 1 And Y3 = 1 Then 'Abs is the function
to provide an absolute value of X3, so that a value of -1 will be given as 1.
                            Valid = True
                            Call PawnAttack(X2, Y2, Valid) 'As the rules for a pawn
are different if they are attacking, a separate subroutine will be created to check
the destination square for a pawn moving diagonally.
                        Else
                            Valid = False
                        End If
                    Case False
                        If X3 = 0 And Y3 = -2 And PositionY = 6 Then
                            Valid = True
                            Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
                            Call PawnMove(X2, Y2, Valid)
                        ElseIf X3 = 0 And Y3 = -1 Then
                            Valid = True
                            Call PawnMove(X2, Y2, Valid)
                        ElseIf Math.Abs(X3) = 1 And Y3 = -1 Then
                            Valid = True
                            Call PawnAttack(X2, Y2, Valid)
                        Else
                            Valid = False
                        End If
                End Select
            Case "Q"
```

```
                'Queen rules if the pawn has been promoted to a Queen
                If Math.Abs(X3) = Math.Abs(Y3) And X3 <> 0 Then
                    Valid = True
                    Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
                    Call CheckDestination(X2, Y2, Valid)
                ElseIf X3 <> 0 And Y3 = 0 Then
                    Valid = True
                    Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
                    Call CheckDestination(X2, Y2, Valid)
                ElseIf X3 = 0 And Y3 <> 0 Then
                    Valid = True
                    Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
                    Call CheckDestination(X2, Y2, Valid)
                Else
                    Valid = False
                End If
            Case "N"
                'Knight rules if the pawn has been promoted to a Knight
                If (Math.Abs(X3) = 2 And Math.Abs(Y3) = 1) Or (Math.Abs(X3) = 1 And
Math.Abs(Y3) = 2) Then
                    Valid = True
                    Call CheckDestination(X2, Y2, Valid)
                End If
            Case "R"
                'Rook rules if the pawn has been promoted to a Rook
                If X3 <> 0 And Y3 = 0 Then
                    Valid = True
                    Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
                    Call CheckDestination(X2, Y2, Valid)
                ElseIf X3 = 0 And Y3 <> 0 Then
                    Valid = True
                    Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
                    Call CheckDestination(X2, Y2, Valid)
                Else
                    Valid = False
                End If
            Case "B"
                'Bishop rules if the pawn has been promoted to a Bishop
                If Math.Abs(X3) = Math.Abs(Y3) And X3 <> 0 Then
                    Valid = True
                    Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
                    Call CheckDestination(X2, Y2, Valid)
                Else
                    Valid = False
                End If
        End Select
        Return Valid
    End Function
    Private Sub PawnMove(ByVal X2 As Integer, ByVal Y2 As Integer, ByRef Valid As
Boolean)
        'Checks whether a pawn can move forward normally
        'Only if there is no piece in front
        If Chess.Grid(X2, Y2) <> "" Then
            Valid = False
        End If
    End Sub
    Private Sub PawnAttack(ByVal X2 As Integer, ByVal Y2 As Integer, ByRef Valid As
Boolean)
        'Checks whether a pawn can capture a piece
        If Chess.Grid(X2, Y2) = "" Then
            Valid = False
        End If
```

```
        Select Case IsWhite
            Case True
                If Chess.Grid(X2, Y2).StartsWith("W") = True Then
                    Valid = False
                End If
            Case False
                If Chess.Grid(X2, Y2).StartsWith("B") = True Then
                    Valid = False
                End If
        End Select
    End Sub
    Public Sub Promote(ByVal X As Integer, ByVal Y As Integer)
        'Requests user input for which piece the pawn should be promoted to
        If (IsWhite = True And Y = 7) Or (IsWhite = False And Y = 0) Then
            Try
                Promotion = InputBox("Which piece would you like to promote to? (Input
'Q' for Queen, 'N' for Knight, 'R' for Rook or 'B' for Bishop)").ToUpper
            Catch
                MsgBox("That is not a valid input.")
            End Try
            If Promotion <> "Q" And Promotion <> "N" And Promotion <> "R" And
Promotion <> "B" Then
                MsgBox("That is not a valid input.")
                Call Promote(X, Y)
            End If
            Chess.MoveLog.Text = Chess.MoveLog.Text + "=" & Promotion
        End If
    End Sub
    Private Sub EnPassantCheck()
        'Checks whether an En Passant move is possible
        If Chess.EnPassant = True Then
            Select Case IsWhite
                Case True
                    If Math.Abs(Chess.EPPosX - PositionX) = 1 And Chess.EPPosY =
PositionY Then
                        Chess.ValidCheck(Chess.EPPosX, Chess.EPPosY + 1) = True
                    End If
                Case False
                    If Math.Abs(Chess.EPPosX - PositionX) = 1 And Chess.EPPosY =
PositionY Then
                        Chess.ValidCheck(Chess.EPPosX, Chess.EPPosY - 1) = True
                    End If
            End Select
        End If
    End Sub
End Class
```

King Class

```
Public Class King : Inherits Piece
    Public Sub CheckValidMoves()
        'Checks all spaces on the board for valid moves
        Dim Valid As Boolean
        Chess.X1 = PositionX
        Chess.Y1 = PositionY
        For j = 0 To 7
            For i = 0 To 7
                Valid = Rules(i, j)
                If Valid = True Then
                    Chess.ValidCheck(i, j) = True
```

```
                Else
                    Chess.ValidCheck(i, j) = False
                End If
            Next
        Next
        CastleCheck()
    End Sub
    Private Function Rules(ByVal X2 As Integer, ByVal Y2 As Integer)
        'Checks whether a move can be made for a King
        Dim X3 As Integer = X2 - PositionX
        Dim Y3 As Integer = Y2 - PositionY
        Dim Valid As Boolean
        If (Math.Abs(X3) = 1 Or X3 = 0) And (Math.Abs(Y3) = 1 Or Y3 = 0) And Not (X3 =
0 And Y3 = 0) Then
            Valid = True
            Call CheckDestination(X2, Y2, Valid)
        Else
            Valid = False
        End If
        Return Valid
    End Function
    Private Sub CastleCheck()
        'Checks whether a Castle move can be made
        Select Case Chess.Grid(Chess.X1, Chess.Y1)
            Case "WKing"
                If Chess.CastleWKMoved = False And Chess.Grid(5, 0) = "" And
Chess.Grid(6, 0) = "" Then
                    Chess.CastleWK = True
                    Chess.ValidCheck(6, 0) = True
                End If
                If Chess.CastleWQMoved = False And Chess.Grid(1, 0) = "" And
Chess.Grid(2, 0) = "" And Chess.Grid(3, 0) = "" Then
                    Chess.CastleWQ = True
                    Chess.ValidCheck(2, 0) = True
                End If
            Case "BKing"
                If Chess.CastleBKMoved = False And Chess.Grid(5, 7) = "" And
Chess.Grid(6, 7) = "" Then
                    Chess.CastleBK = True
                    Chess.ValidCheck(6, 7) = True
                End If
                If Chess.CastleBQMoved = False And Chess.Grid(1, 7) = "" And
Chess.Grid(2, 7) = "" And Chess.Grid(3, 7) = "" Then
                    Chess.CastleBQ = True
                    Chess.ValidCheck(2, 7) = True
                End If
        End Select
    End Sub
End Class
```

Queen Class

```
Public Class Queen : Inherits Piece
    Public Sub CheckValidMoves()
        'Checks all spaces on the board for valid moves
        Chess.X1 = PositionX
        Chess.Y1 = PositionY
        Dim Valid As Boolean
        For j = 0 To 7
            For i = 0 To 7
```

```
                Valid = Rules(i, j)
                If Valid = True Then
                    Chess.ValidCheck(i, j) = True
                Else
                    Chess.ValidCheck(i, j) = False
                End If
            Next
        Next
    End Sub
    Public Function Rules(ByVal X2 As Integer, ByVal Y2 As Integer)
        'Checks whether a move is valid for a Queen
        Dim X3 As Integer = X2 - PositionX
        Dim Y3 As Integer = Y2 - PositionY
        Dim Valid As Boolean
        If Math.Abs(X3) = Math.Abs(Y3) And X3 <> 0 Then
            Valid = True
            Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
            Call CheckDestination(X2, Y2, Valid)
        ElseIf X3 <> 0 And Y3 = 0 Then
            Valid = True
            Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
            Call CheckDestination(X2, Y2, Valid)
        ElseIf X3 = 0 And Y3 <> 0 Then
            Valid = True
            Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
            Call CheckDestination(X2, Y2, Valid)
        Else
            Valid = False
        End If
        Return Valid
    End Function
End Class
```

Knight Class

```
Public Class Knight : Inherits Piece
    Public Sub CheckValidMoves()
        'Checks all spaces on the board for valid moves
        Dim Valid As Boolean
        Chess.X1 = PositionX
        Chess.Y1 = PositionY
        For j = 0 To 7
            For i = 0 To 7
                Valid = Rules(i, j)
                If Valid = True Then
                    Chess.ValidCheck(i, j) = True
                Else
                    Chess.ValidCheck(i, j) = False
                End If
            Next
        Next
    End Sub
    Public Function Rules(ByVal X2 As Integer, ByVal Y2 As Integer)
        'Checks whether a move is valid for a Knight
        Dim X3 As Integer = X2 - PositionX
        Dim Y3 As Integer = Y2 - PositionY
        Dim Valid As Boolean
        If (Math.Abs(X3) = 2 And Math.Abs(Y3) = 1) Or (Math.Abs(X3) = 1 And
Math.Abs(Y3) = 2) Then
            Valid = True
```

```
            Call CheckDestination(X2, Y2, Valid)
        End If
        Return Valid
    End Function
End Class
```

Rook Class

```
Public Class Rook : Inherits Piece
    Public Sub CheckValidMoves()
        'Checks all spaces on the board for valid moves
        Dim Valid As Boolean
        Chess.X1 = PositionX
        Chess.Y1 = PositionY
        For j = 0 To 7
            For i = 0 To 7
                Valid = Rules(i, j)
                If Valid = True Then
                    Chess.ValidCheck(i, j) = True
                Else
                    Chess.ValidCheck(i, j) = False
                End If
            Next
        Next
    End Sub
    Public Function Rules(ByVal X2 As Integer, ByVal Y2 As Integer)
        'Checks whether a move is valid for a Rook
        Dim X3 As Integer = X2 - PositionX
        Dim Y3 As Integer = Y2 - PositionY
        Dim Valid As Boolean
        If X3 <> 0 And Y3 = 0 Then
            Valid = True
            Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
            Call CheckDestination(X2, Y2, Valid)
        ElseIf X3 = 0 And Y3 <> 0 Then
            Valid = True
            Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
            Call CheckDestination(X2, Y2, Valid)
        Else
            Valid = False
        End If
        Return Valid
    End Function
End Class
```

Bishop Class

```
Public Class Bishop : Inherits Piece
    Public Sub CheckValidMoves()
        'Checks all spaces on the board for valid moves
        Dim Valid As Boolean
        Chess.X1 = PositionX
        Chess.Y1 = PositionY
        For j = 0 To 7
            For i = 0 To 7
                Valid = Rules(i, j)
                If Valid = True Then
                    Chess.ValidCheck(i, j) = True
```

```
                Else
                    Chess.ValidCheck(i, j) = False
                End If
            Next
        Next
    End Sub
    Public Function Rules(ByVal X2 As Integer, ByVal Y2 As Integer)
        'Checks whether a move is valid for a Bishop
        Dim X3 As Integer = X2 - PositionX
        Dim Y3 As Integer = Y2 - PositionY
        Dim Valid As Boolean
        If Math.Abs(X3) = Math.Abs(Y3) And X3 <> 0 Then
            Valid = True
            Call CheckSpaces(PositionX, PositionY, X3, Y3, Valid)
            Call CheckDestination(X2, Y2, Valid)
        Else
            Valid = False
        End If
        Return Valid
    End Function
End Class
```