Chess Project Report

Programming 1 project

Name	ID	Class ID
عبدالرحمن السيد احمد علي عمر	19015893	35
عبدالرحمن السيد جاد السيد	19015894	36

Game Description:

Our game is a chess game for 2 players .the game ends with only two ways check mate or draw . You can save in any time during the game but you can only load in the start of the game. You can undo moves to the first move and redo to the last move too .there are many additional features and special cases handled like:

- Many ways to end the game with draw such as stalemate and lack of checkmate material (each player have one of the following : just a king , a king and bishop or a king , a king and a knight or a king and two knights)
- Enpassent
- Castling
- Promotion
- A sound alert in case of check , checkmate and draw
- The ability to continue the game after saving it
- The code can handle any input (upper or lower case)

Overview:

First of all , our code see if the user want to start a new game or load an saved game . in case of loading the game the code take the name of the saved file from the user as input and check if there is a saved file with the same name , if not found the code prints not found and asks for the name again. If found it starts the code from the saved point.

In case the user starts a new game the game starts and it's the first player(white) turn. The program takes the moves as input from the user and check if it is valid or not. if not valid the code prints enter move correctly and takes the move again. If valid the code store the frame in case of needing to undo and move the wanted piece to the wanted position and then print the new board and changes the player turn to the second player (black) and loop again.

Every turn the code check if there is a check. If there isn't, the code check for the draw cases .if there is, the game ends immediately and print draw. If there isn't, the game continue normally. if there is, check the code checks for checkmate .if there is, check mate the game ends immediately and print the winner.if there isn't checkmate the game continues but the next player have to make a move to remove the check.

In case if the user want to undo move the code returns the board to the previous frame and vice versa in case of redo.

In case if the user want to save the code ask him to enter the name of saved file . then the code starts to store everything needed to be stored in the file by a specified order to make it easy to be loaded .

Assumptions:

- In first we thought the numbers started from up to down in the board design then we discover it and handled it in the input to convert the new way the user enter it to the old way the functions can deal with

The old way 

the new way 

The Functions:

- Void movement (int movej= first index of the initial move, int movei= second index of the initial move, int movefj= first index of the final move, int movefi = second index of the final move):
  It makes the move by recording the piece at the initial place and replace its place by – or . according to its place and check the final move if there is a piece ,it is added to died pieces and replaced by the piece

- int CheckRook (int movej= first index of the initial move, int movei= second index of the initial move, int movefj= first index of the final move, int movefi = second index of the final move, char pro = the fifth character in the input)
  return int x which is 0 or 1 and it indicates if it is valid move for rook or not

- int Checkknight (int movej= first index of the initial move, int movei= second index of the initial move, int movefj= first index of the final move, int movefi = second index of the final move, char pro = the fifth character in the input)
  return int x which is 0 or 1 and it indicates if it is valid move for knight or not

- int CheckBishop(int movej= first index of the initial move, int movei= second index of the initial move, int movefj= first index of the final move, int movefi = second index of the final move, char pro = the fifth character in the input)
  return int x which is 0 or 1 and it indicates if it is valid move for bishop or not

- int CheckKing(int movej= first index of the initial move, int movei= second index of the initial move, int movefj= first index of the final move, int movefi = second index of the final move, char pro = the fifth character in the input)
  return int x which is 0 or 1 and it indicates if it is valid move for king or not

- int CheckQueen(int movej= first index of the initial move, int movei= second index of the initial move, int movefj= first index of the final move, int movefi = second index of the final move, char pro = the fifth character in the input)
  return int x which is 0 or 1 and it indicates if it is valid move for queen or not

- int CheckPawnW(int movej= first index of the initial move, int movei= second index of the initial move, int movefj= first index of the final move, int movefi = second index of the final move, char pro = the fifth character in the input)
  return int x which is 0 or 1 and it indicates if it is valid move for white pawn or not

- int CheckPawnB(int movej= first index of the initial move, int movei= second index of the initial move, int movefj= first index of the final move, int movefi = second index of the final move, char pro = the fifth character in the input):
  return int x which is 0 or 1 and it indicates if it is valid move for black pawn or not

- int CheckMovement (int movej= first index of the initial move, int movei= second index of the initial move, int movefj= first index of the final move, int movefi = second index of the final move, char piece= piece color , char pro=fifth character in the input):
  return int x which is 0 or 1 to indicate if it is valid move or not

- void CheckCastling():
  it checks if any rook or king move and record the move in the global array R

- void storepieces() :
  it store the number of pieces and its places and the place of the king in whitePieces and BlackPieces structure.

- Int checked(char p): p is the player that in turn:
  The function returns 1 if there is a check, return 0 if not.

- int tempMoveCheck(char p, int movej, int movei, int movefj, int movefi, char pro): p is the player that in turn, movej= first index of the initial move, movei= second index of the initial move, movefj= first index of the final move, movefi = second index of the final move, pro=fifth character in the input
  The function return 0 if the move is valid and will not cause check on the player, return 1 if the move is not valid or will not cause check on the player.

- void minMax (int arr[]) : arr is array of size 2:
  The function sort array of 2.

- int checkmate(char p): p is the player that in turn:
  The function returns 1 if there is a checkmate , return 0 if not.

- int lackofCheckmate(char p): p is any of the 2 players:
  The function return 1 if there lack of checkmate material with the parameter player, return 0 if not.

NOTE : the lack of checkmate material happened if the player has king and only 1 bishop, king and only 1 knight or king and only 2 knights . if the 2 players have lack of checkmate material the game end with draw

- int stalemate(char p): p is the player that in turn:
  The function returns 1 if there is a stalemate , return 0 if not.

- void storemove(char p, int ifchecked, char startorPlay): p is the, ifchecked=1 if there a check and 0 if not , startorPlay = 's' if the start of new game and = 'p' if it is a play in the game:
  This function declares the linked list node and store the current data in it then add it to the linked list for the undo and redo .

- int undoRedo(char unRedo, char *p, int *ifchecked): unRedo = 'u' if wants to undo, = 'r' if wants to redo and = 'c' if wants to restore the current data , pointer p is pointer in the current player in the game, pointer ifchecked is pointer in the ifchecked variable in the game:
  the return value = 1 if the undo or redo or restoring the data can happen and done, return 0 if it cannot because there is no previous play or next play.
  The function make the undo and the redo and restoring the data after any changes in the code.

- void save(char piece= piece color):
  it opens a file with name taken as input and store the needed data inside it .

- void load(char *piece= piece color):
  it opens a file with name taken as input and load the needed data from it .

- void printBoard():
  it prints the chess board and the died pieces until now.

Description of all used data structures:

1. char board[8][8] : The 2D array for the chess.

2. struct died : The struct members:
   char die[15] : array of died pieces
   int counter : number of died pieces

3. struct pieces : The struct members:
   int k[2] : the place of the king
   int places[16][2] : the places of all the current pieces in board
   int NOp : the number of current pieces in board

4. struct node : The struct members :
   char chess[8][8] : the chess board
   char player
   int R[4] : castling indicators

int pw[8] : white enpassent indicators
int pb[8] : black enpassent indicators
int ifchecked : refer if in this play there are check on the king
int wdie , bdie : counter of the died pieces
struct node *next : pointer on the next node for redo
struct node *prev : pointer on the previous node for undo

Some of Important global variables:

1. For movement file:
   R[4] : castling indicators , pw[8], pb[8] : with and black enpassent indicators
   struct died wdied, bdied : for white and black died pieces
2. For check file :
   struct pieces whitePieces , BlackPieces
   checkby[3] : the first item is number of pieces making check, the second and third is the place of the piece making check if it 1 piece
3. Undo and Redo file :
   struct node *head , *current : pointers on the head of the linked list and pointer on the current node in the game.
4. Main file :
   board[8][8] : game board.

Pseudocode for the main algorithms :

1. Function CheckMovement with parameters (movej= first index of the initial move, movei= second index of the initial move, movefj= first index of the final move, movefi = second index of the final move, piece= piece color , pro=promotion(fifth character)):

```
x=0
If piece =w
        If board[movefj][movefi] is black or '.' or '-' and board[movej][movei] is white
                Check board[movej][movei]
                        If it is 'p' : Check pawn movement with CheckPawnW and record the
                                returned value in x
                        If it is 'r' : Check rook movement with CheckRook and record the returned
                                value in x
                        If it is 'n' : Check knight movement with CheckKhight and record the
                                returned value in x
                        If it is 'b' : Check bishop movement with CheckBishop and record the
                                returned value in x
                        If it is 'k' : Check king movement with CheckKing and record the returned
                                value in x
                        If it is 'q' : Check queen movement with CheckQueen and record the
                        returned value in x
                Return the x value.
If piece =b
        If board[movefj][movefi] is white or '.' or '-' and board[movej][movei] is black
                Check board[movej][movei]
                        If it is 'p' : Check pawn movement with CheckPawnB and record the
                                returned value in x
                        If it is 'r' : Check rook movement with CheckRook and record the returned
                                value in x
                        If it is 'n' : Check knight movement with CheckKhight and record the
                                returned value in x
                        If it is 'b' : Check bishop movement with CheckBishop and record the
                                returned value in x
                        If it is 'k' : Check king movement with CheckKing and record the returned
                                value in x
                        If it is 'q' : Check queen movement with CheckQueen and record the
                                returned value in x
        Return the x value.
Else
        Return 0
```

2. Function movement with parameters (movej= first index of the initial move, movei= second index of the initial move, movefj= first index of the final move, movefi = second index of the final move):

```
Temp=board[movej] [movei]
If board[movefj] [movefi] is upper case letter(black):
        Add board[movefj] [movefi] to black died pieces
        increase the counter of black died pieces by 1

else if board[movefj] [movefi] is lower case letter(white):
        Add board[movefj] [movefi] to white died pieces
        increase the counter of white died pieces 1

if movei+movej is an even number:
        board[movej][movei] = '-'
else:
        board[movej][movei] =  '.'

board[movefj] [movefi]= temp
```

3. Function checked : with parameters (p = player):

```
Store the pieces in the structure
If p = 'b' :
        current =   white pieces places
        (i,j) = black king place
        o = 'w'
else :
        current =   black pieces places
        (i,j) = white king place
        o = 'b'
number of pieces making check = 0
r = 0
for (a,b) in current :
        if piece in (a,b) can go to (i,j) :
                number of pieces making check +1
                place of check by piece = (a,b)
                r=1
Return r
```

4. Function tempMoveCheck : with parameters(p=player , (movej,movei)=start , (movefj,movefi)=end , pro=promotion piece (which is = null if not promoted)):

```
If movej, movei, movefj and moveri not between 0 and 7 :
        Return 1
Store check by piece

if piece in (movej,movei) can go to (movefj,movefi) :
        move piece in (movej,movei) to (movefj,movefi)
        r = checked or not

        restore the data that was before the move using undoRedo function with parameter 'c'
        restore the check by piece

        Return r
Else : return 1
```

5. Function checkmate : with parameters (p = player) :

```
If p = 'b' :
        current =   black pieces places
        (i,j) = black king place
        o = 'w'
else :
        current =   white pieces places
        (i,j) = white king place
        o = 'b'

loop on the 9 places around the king:
        if king can move to the place:
                Return 0

if number of pieces making check > 1 :
        Return 1
Else : (one piece making check)
(ich,jch) = place of check by piece
For (a,b) in current :
        If piece in(a,b) can go to (ich,jch) :  (can eat the check by piece)
                Return 0
If check by piece = knight :
        Return 1   (can't interrupt the path)

For place (i,j) in the path between the king and the check by piece :
        For (a,b) in current :
                If piece in (a,b) can go to (i,j) :
                        Return 0  (path can be interrupted)
Return 1
```

6. Function stalemate : with parameters (p = player):

```
If there are lack of checkmate material for white and black :
        Return 1
If p = 'b' :
        current =   black pieces places
        (i,j) = black king place
else :
        current =   white pieces places
        (i,j) = white king place

loop on the 9 places around the king:
        if king can move to the place:
                Return 0
For (i,j) in current :
        If piece in (i,j) can move :
                Return 0
Return 1
```

7. Function storemove : with parameters (p=player , ifchecked , startorPlay):

```
Create the linked list node t
Store the board and data in t
If startorPlay = 's' :  (start new game)
        Previous, next of t = NULL
        head = t
        current = t
else if startorPlay = 'p' :  (play in the game)
        next of current = t
        previous of t = current
        next of t = NULL
        current = t
```

8. Function undoRedo : with parameters (unRedo , pointer p =pointer on player , pointer ifcheck):

```
If unRedo = 'u' :  (making undo)
        If previous of current not NULL :
                current = previous of current
        Else : return 0   (can't do undo)
Else if unRedo = 'r' :  (making redo)
        If next of current not NULL :
                current = next of current
        Else : return 0   (can't do redo)
Else if unredo = 'c' : stay at the current

Board and data = data in current node
Return 1
```

9. Function save with parameter (piece= piece color):

```
Display to the user “Enter the name of the save file”
take the name of the saved file from user
open file by the name of the entered file name in writing mode (create if not found)
loop i from 0 to 8
        loop j from 0 to 8
                store board [i][j] in the file
loop i from 0 to 4
        store R[i] in the file after converting it into charcters
loop i from 0 to 8
        store pw[i] in the file after converting it into charcters
loop i from 0 to 8
        store pb[i] in the file after converting it into charcters
store piece in the file
loop i from 0 to number of white died pieces
        store white died pieces in the file
loop i from 0 to number of black died pieces
        store black died pieces in the file
close the file
```

10. Function load with parameter (piece= piece color):

```
Display to the user “Enter the name of the load file”
loop until the user enter a valid load file name
        take the name of the load file from the user
        open the file with this name in the reading mode (doesn’t open any thing if not found
        if it is found:
                break the loop
        if not found:
                Display to the user “not found”
Loop until the end of file
        If i < 8 and j<8 :
                Board[i][j] = c(character from the file)
                j = j + 1
                If j=8 :
                        Increse i by 1
                        J=0
                        Continue the loop
        If i >= 8 and j <12 :
                R[i-8] = c after convert the characters into numbers
                i = i + 1
                Continue the loop
        If i >= 12 and j <20 :
                pw[i-12] = c after convert the characters into numbers
                i = i + 1
                Continue the loop
```

```
If i >= 20 and j <28 :
        pb[i-20] = c after convert the characters into numbers
        i = i + 1
        Continue the loop
If i=28 :
        Piece =c
        i = i + 1
        Continue the loop
If i =29 && c is lower case character :
        Load the white died pieces
        Increase counter of white dead pieces
        Continue the loop
If i =29 && c is upper case character :
        Load the black died pieces
        Increase counter of black dead pieces
        Continue the loop
Close the opened file
```

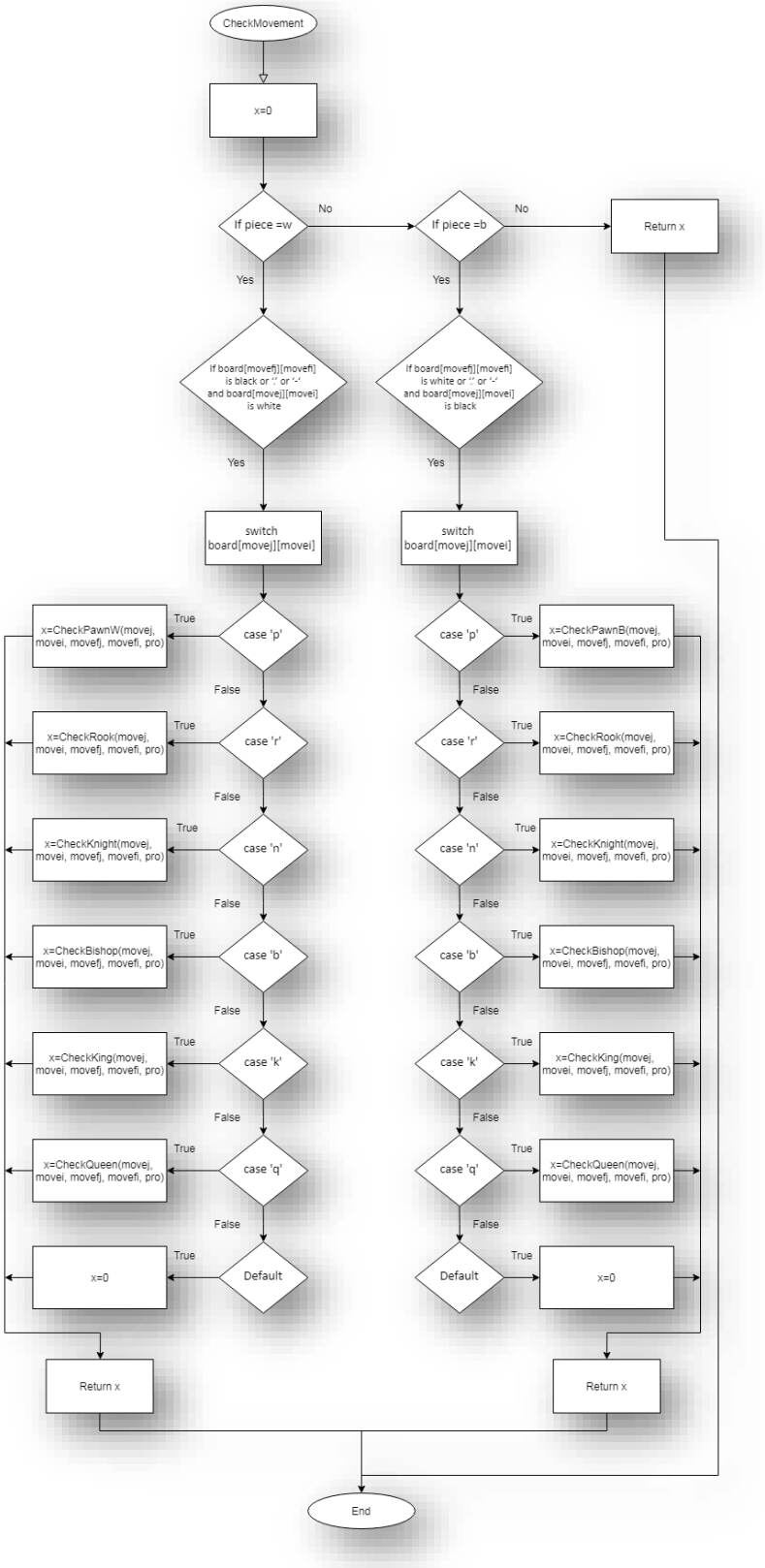
11. Main or the game loop:

```
Exit = 0
X= 0  (refer to if move will happen)
piece = white
white died counter, black died counter = 0
while true :
        scan sl
        if sl = "load":
                call load() function
                break
        else if sl = "start" :
                break
        else: print "Enter correctly"

store the start with storemove() function
while not exit :  loop 1
        call CheckCastling() function
        print the piece
        while true :  loop 2
                scan the move
                (movej,movei) = start position
                (movefj,movefi) = start position
                if move[0] and move[2] between a,h and move[1] and move[3] between 1,8:
                        if not tempMoveCheck() on piece from(movej,movei) to(movefj,movefi):
                                if Checkmovement() from(movej,movei) to(movefj,movefi):
                                        x=1
                                        break
                else if move= "save":
```

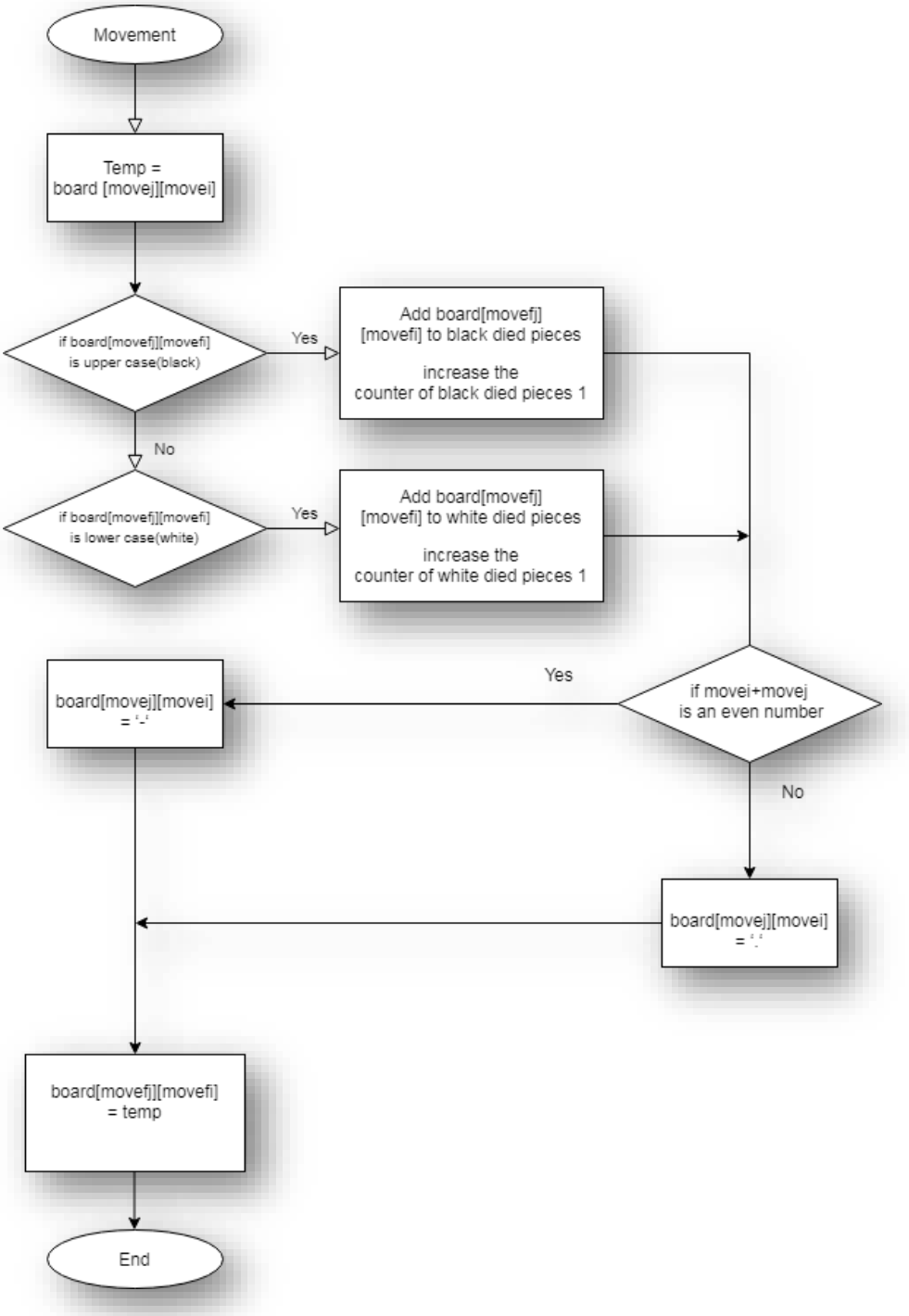
```
                    call save() function
                    print "continue or not"
                    scan t
                    if t= "y" :
                        exit, x = 0
                        call printBoard() function
                        break
                    else if t= "n" :
                        exit =1
                        x=0
                        break
                else if move = "undo" :
                    x=0
                    if undoRedo() with parameter 'u' :  (undo done)
                        call printBoard() function
                        if checked :
                            print "checked"
                    else :  print "can't do undo"
                    break
                else if move = "redo" :
                    x=0
                    if undoRedo() with parameter 'r' :  (redo done)
                        call printBoard() function
                        if checked :
                            print "checked"
                    else :  print "can't do redo"
                    break
        end of loop 2
        if x=1 :  (move will happen)
            call movement function from(movej,movei) to(movefj,movefi)
            call printBoard() function
            switch players
            call storemove() functions with parameter 'p'  (play in game)

            if player is checked :
                if there a checkmate :
                    print "Chackmate"
                    print : player wins
                    exit = 1
                else :
                    print "Check"
            else if there stalemate :
                print : "Draw"
                exit = 1
end of loop 1
```

Flowchart for the main algorithms :

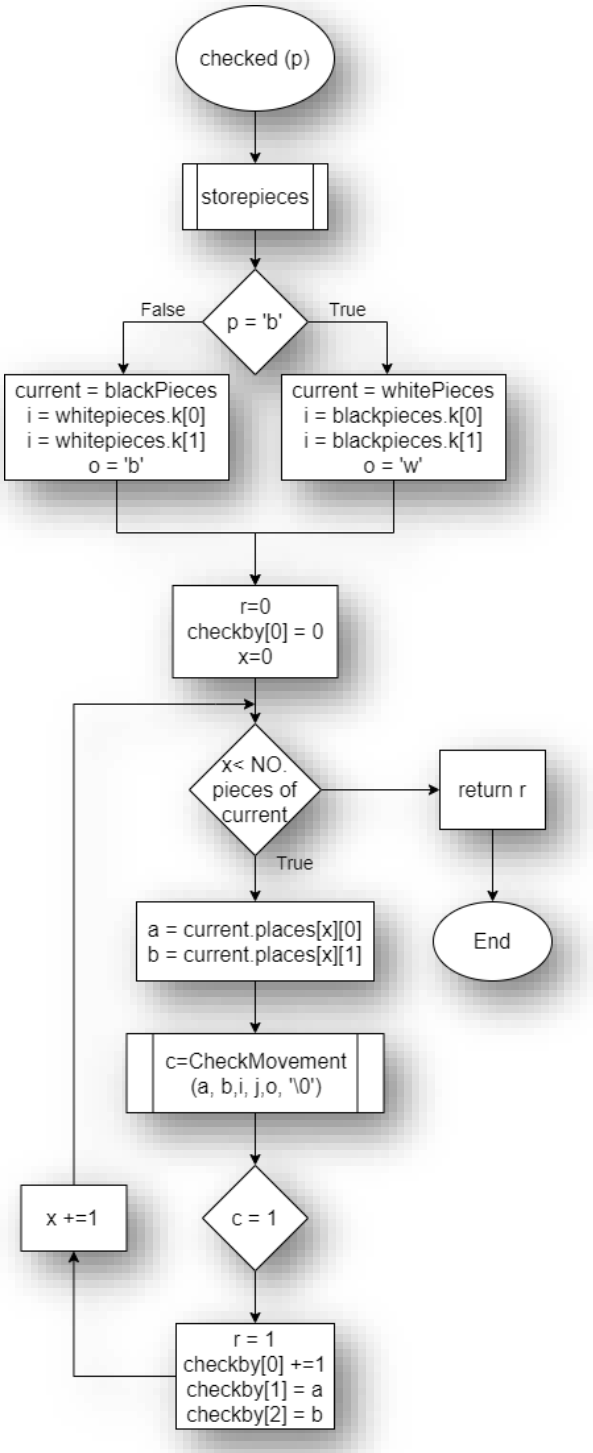
1. Checkmovement :



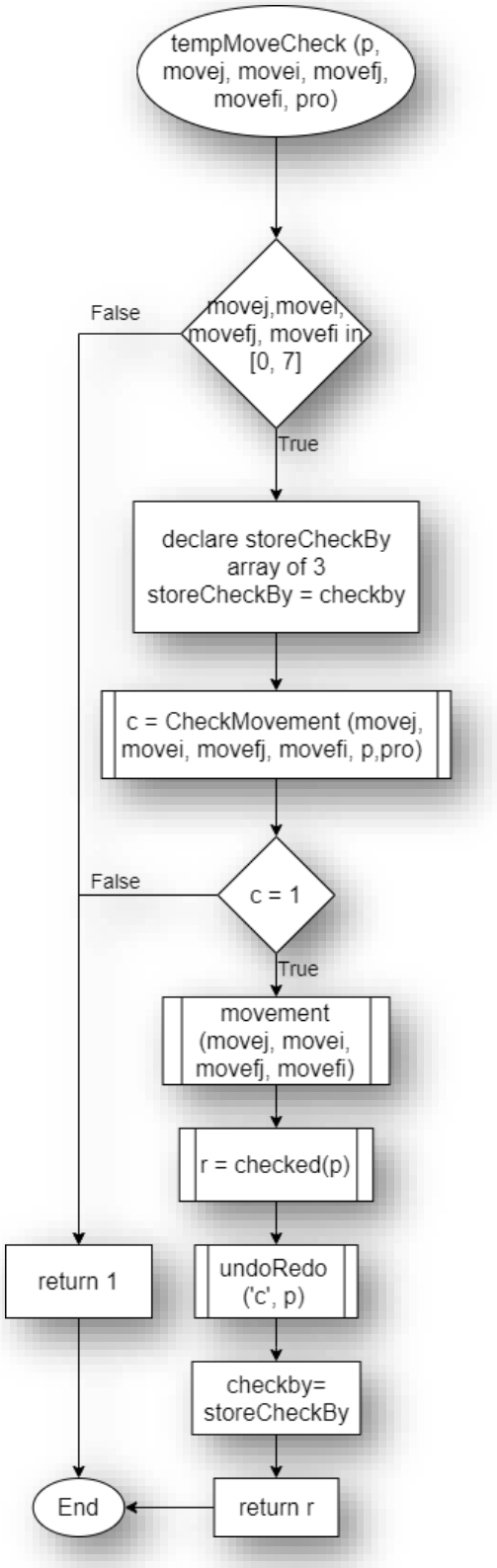
2. Movement:


Movement
Temp =
board [movej][movei]
if board[movefj][movefi]
is upper case(black)
Yes
Add board[movefj]
[movefi] to black died pieces
increase the
counter of black died pieces 1
No
if board[movefj][movefi]
is lower case(white)
Yes
Add board[movefj]
[movefi] to white died pieces
increase the
counter of white died pieces 1
board[movej][movei]
= '-'
Yes
if movei+movej
is an even number
No
board[movej][movei]
= '.'
board[movefj][movefi]
= temp
End

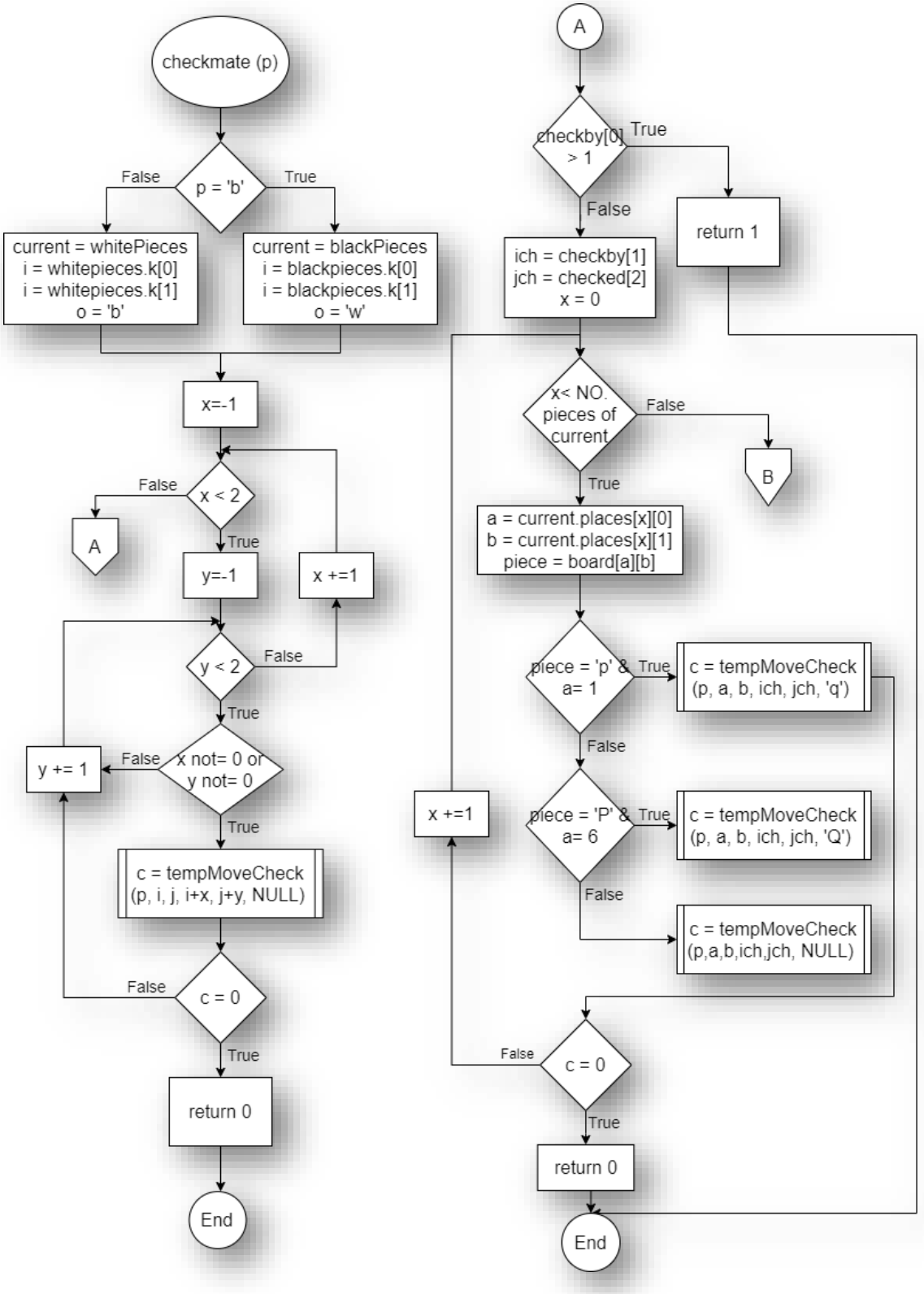
3. Check :

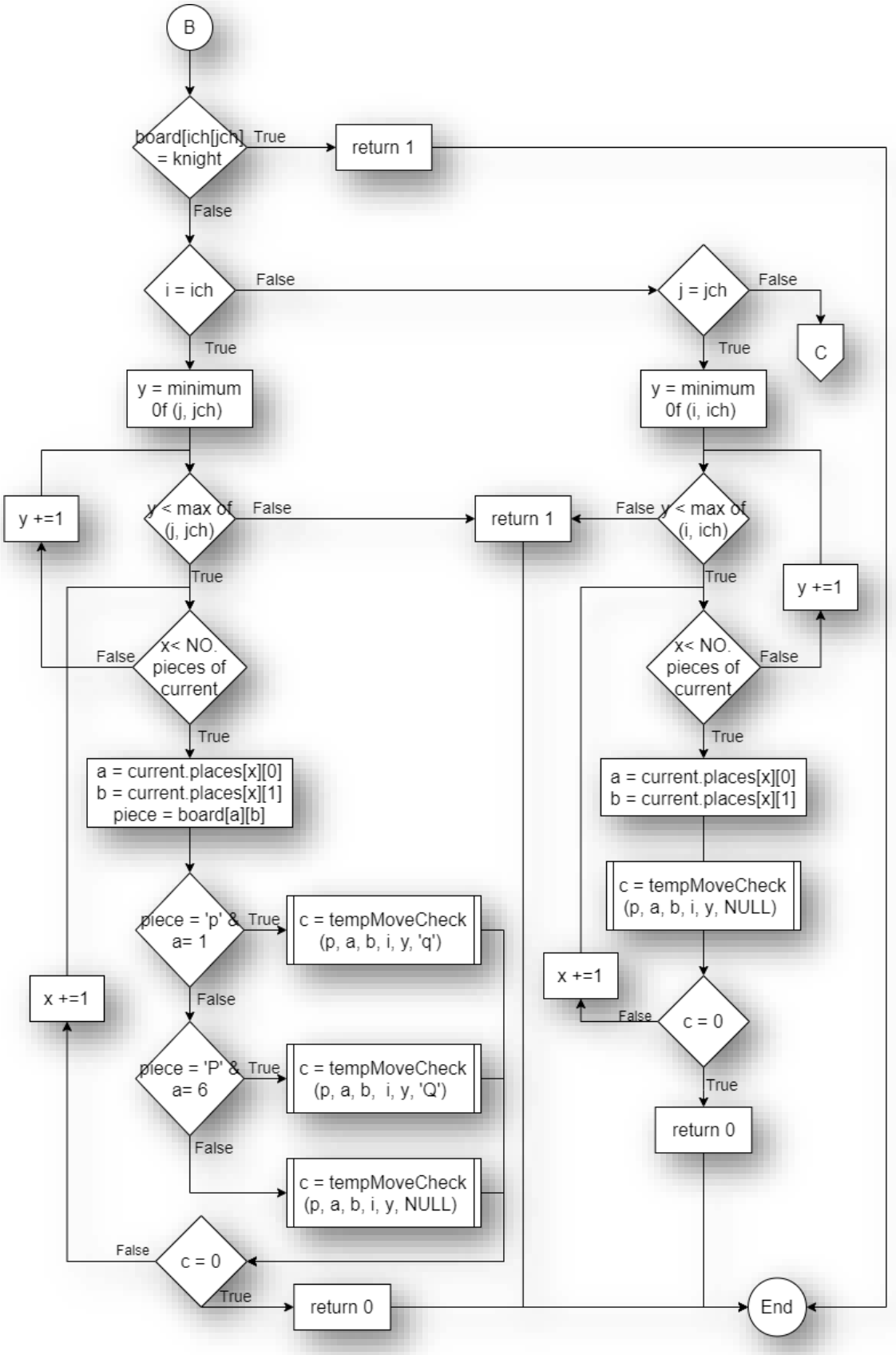


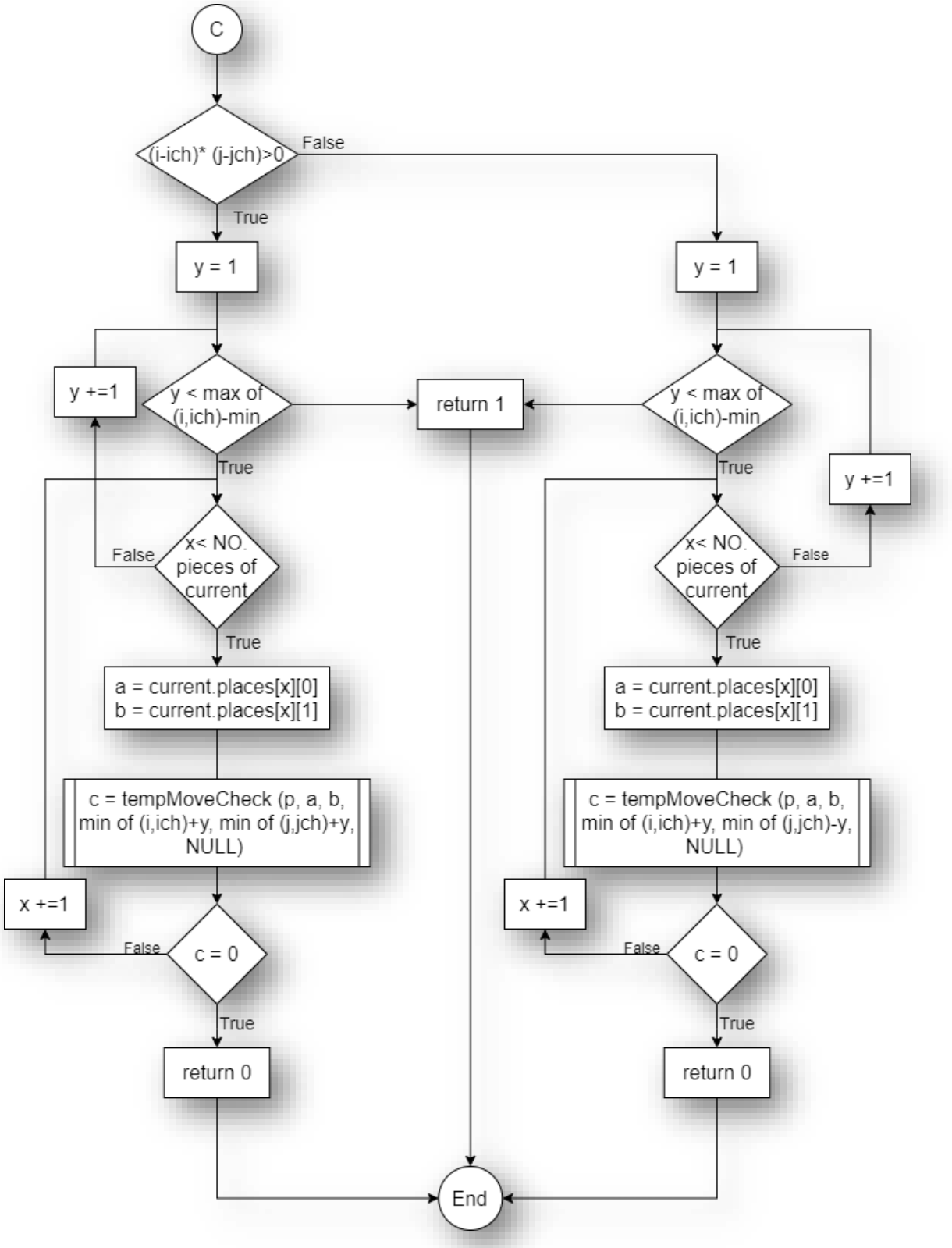
4. tempMoveCheck :



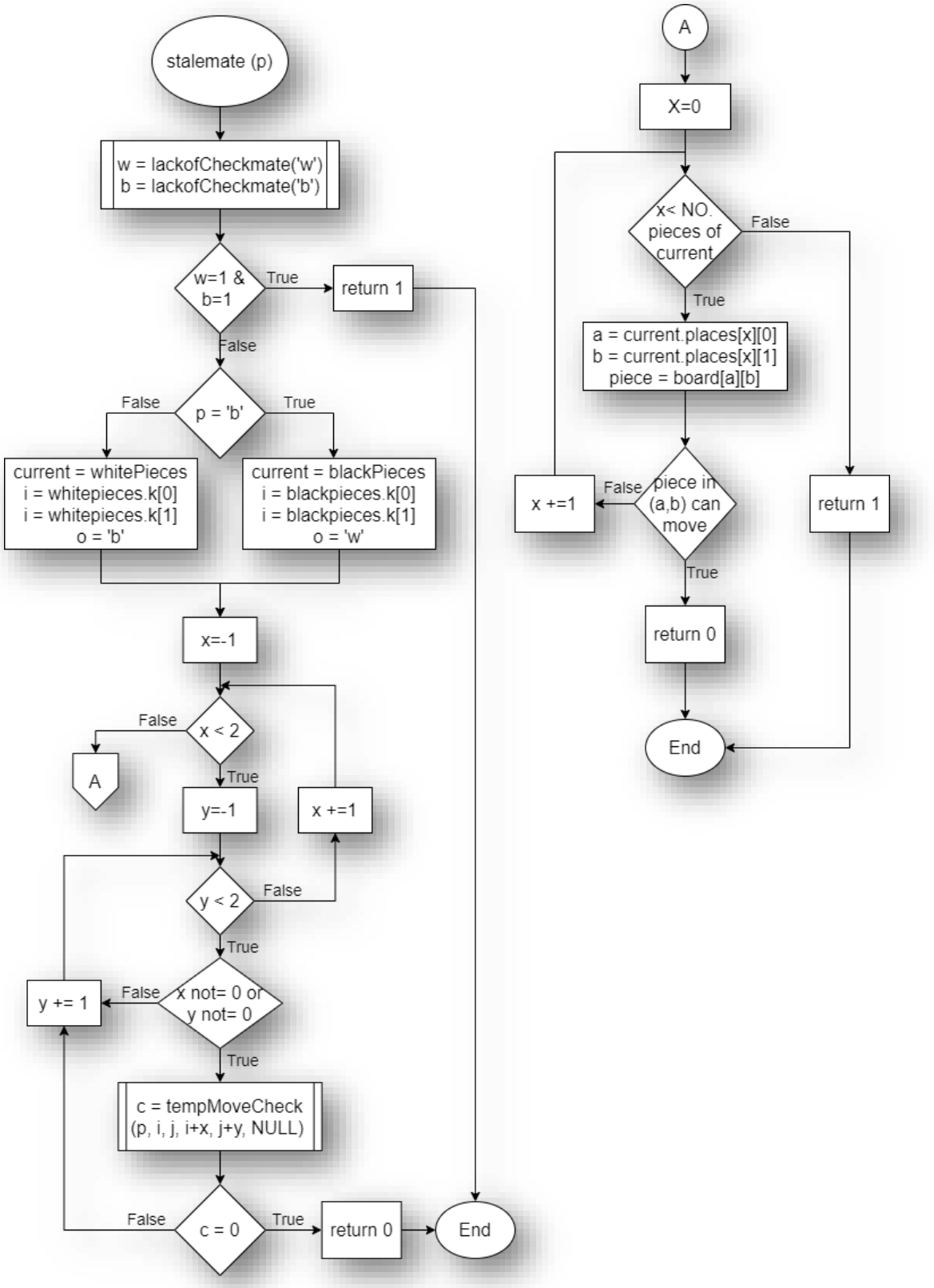
5. Checkmate :


checkmate (p)
p = 'b'
False
True
current = whitePieces
i = whitepieces.k[0]
i = whitepieces.k[1]
o = 'b'
current = blackPieces
i = blackpieces.k[0]
i = blackpieces.k[1]
o = 'w'
x=-1
x < 2
False
True
A
y=-1
x +=1
y < 2
False
True
y += 1
False
x not= 0 or
y not= 0
True
c = tempMoveCheck
(p, i, j, i+x, j+y, NULL)
False
c = 0
True
return 0
End
A
checkby[0]
> 1
True
False
return 1
ich = checkby[1]
jch = checked[2]
x = 0
x< NO.
pieces of
current
False
True
B
a = current.places[x][0]
b = current.places[x][1]
piece = board[a][b]
piece = 'p' &
a= 1
True
c = tempMoveCheck
(p, a, b, ich, jch, 'q')
False
piece = 'P' &
a= 6
True
c = tempMoveCheck
(p, a, b, ich, jch, 'Q')
False
c = tempMoveCheck
(p,a,b,ich,jch, NULL)
x +=1
False
c = 0
True
return 0
End

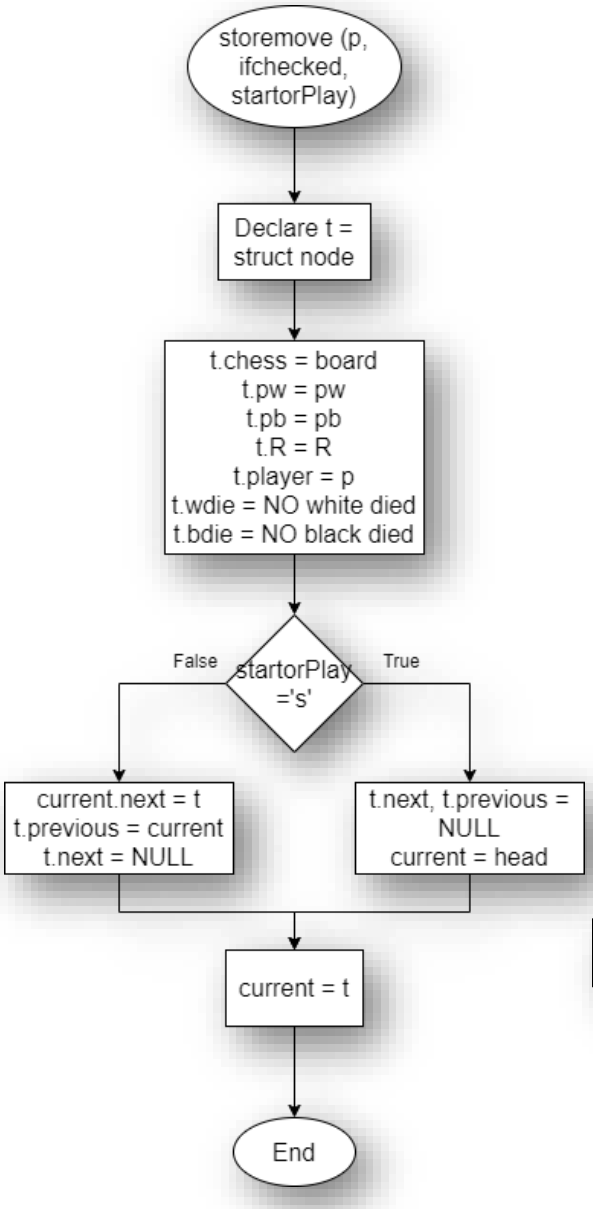

B
board[ich[jch] = knight
True
return 1
False
i = ich
False
j = jch
False
C
True
True
y = minimum 0f (j, jch)
y = minimum 0f (i, ich)
y +=1
y < max of (j, jch)
False
return 1
False
y < max of (i, ich)
True
True
y +=1
False
x< NO. pieces of current
x< NO. pieces of current
False
True
True
a = current.places[x][0]
b = current.places[x][1]
piece = board[a][b]
a = current.places[x][0]
b = current.places[x][1]
piece = 'p' & a= 1
True
c = tempMoveCheck (p, a, b, i, y, 'q')
c = tempMoveCheck (p, a, b, i, y, NULL)
x +=1
x +=1
False
False
c = 0
piece = 'P' & a= 6
True
c = tempMoveCheck (p, a, b, i, y, 'Q')
True
return 0
False
c = tempMoveCheck (p, a, b, i, y, NULL)
False
c = 0
True
return 0
End


C
(i-ich)* (j-jch)>0
False
True
y = 1
y = 1
y +=1
y < max of (i,ich)-min
return 1
y < max of (i,ich)-min
True
True
y +=1
False
x< NO. pieces of current
x< NO. pieces of current
False
True
True
a = current.places[x][0]
b = current.places[x][1]
a = current.places[x][0]
b = current.places[x][1]
c = tempMoveCheck (p, a, b, min of (i,ich)+y, min of (j,jch)+y, NULL)
c = tempMoveCheck (p, a, b, min of (i,ich)+y, min of (j,jch)-y, NULL)
x +=1
x +=1
False
c = 0
False
c = 0
True
True
return 0
return 0
End

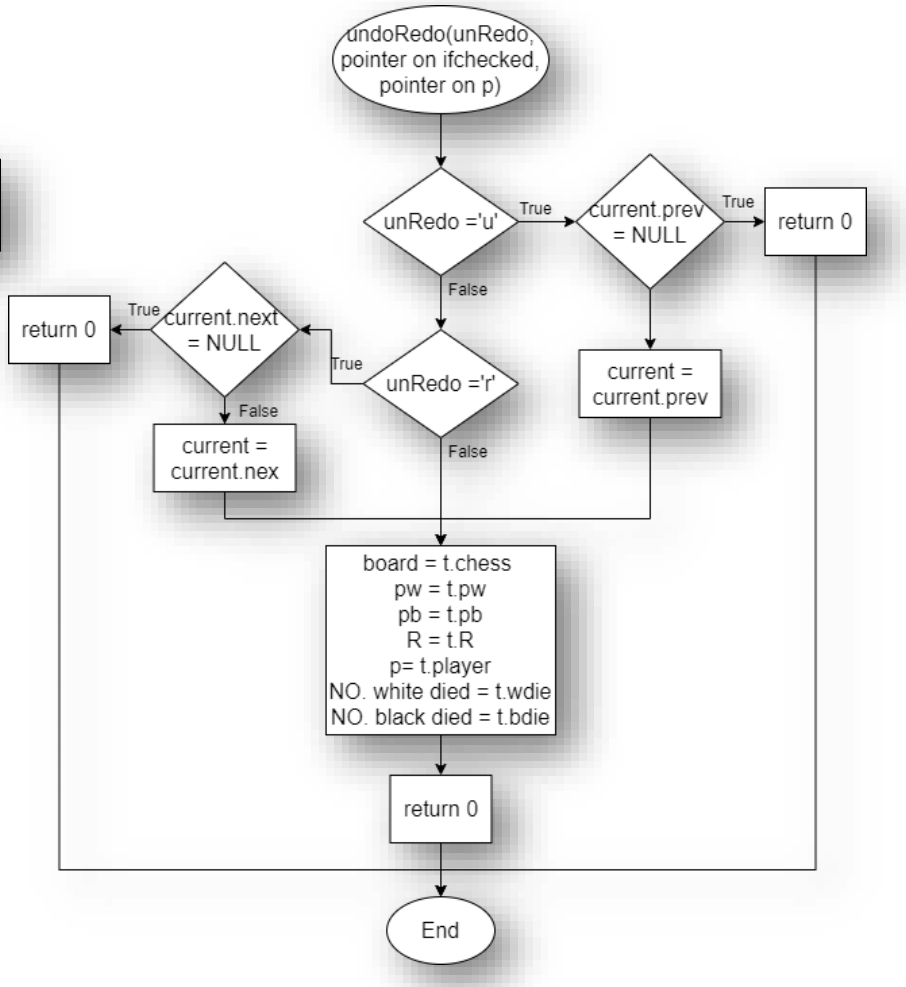
6. Stalemate :


stalemate (p)
w = lackofCheckmate('w')
b = lackofCheckmate('b')
w=1 & b=1
True
return 1
False
p = 'b'
False
True
current = whitePieces
i = whitepieces.k[0]
i = whitepieces.k[1]
o = 'b'
current = blackPieces
i = blackpieces.k[0]
i = blackpieces.k[1]
o = 'w'
x=-1
x < 2
False
A
True
y=-1
x +=1
y < 2
False
True
y += 1
False
x not= 0 or y not= 0
True
c = tempMoveCheck (p, i, j, i+x, j+y, NULL)
c = 0
False
True
return 0
End
A
X=0
x< NO. pieces of current
False
True
a = current.places[x][0]
b = current.places[x][1]
piece = board[a][b]
piece in (a,b) can move
False
x +=1
True
return 0
return 1
End

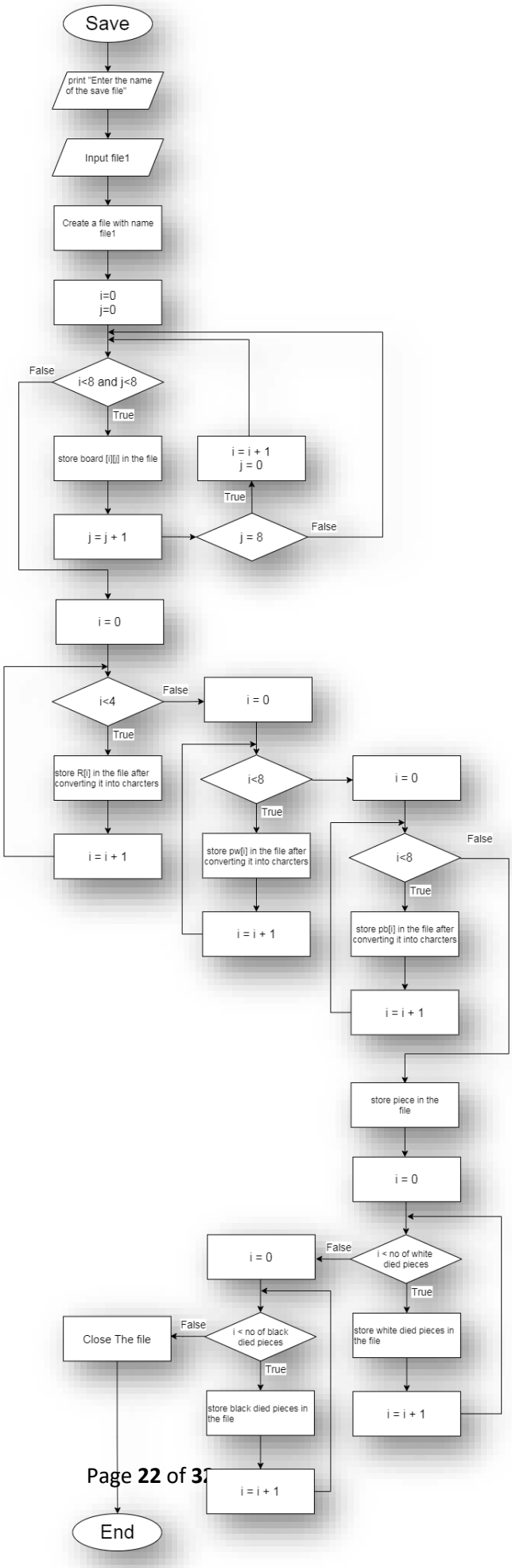
7. Storemove :



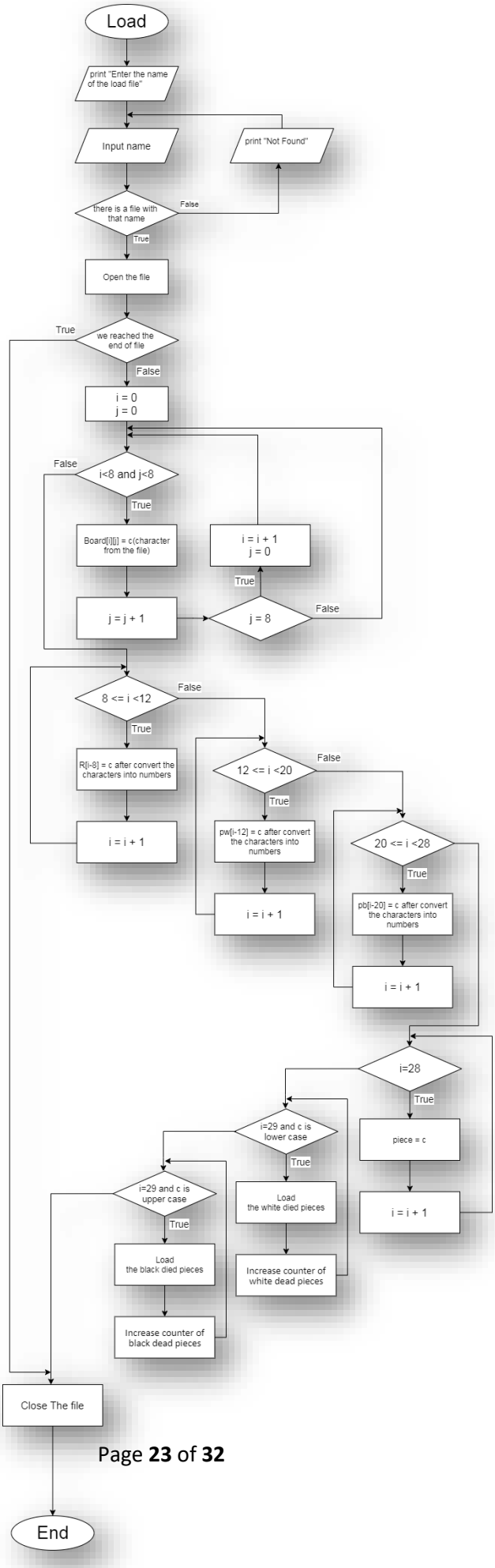
8. undoRedo :



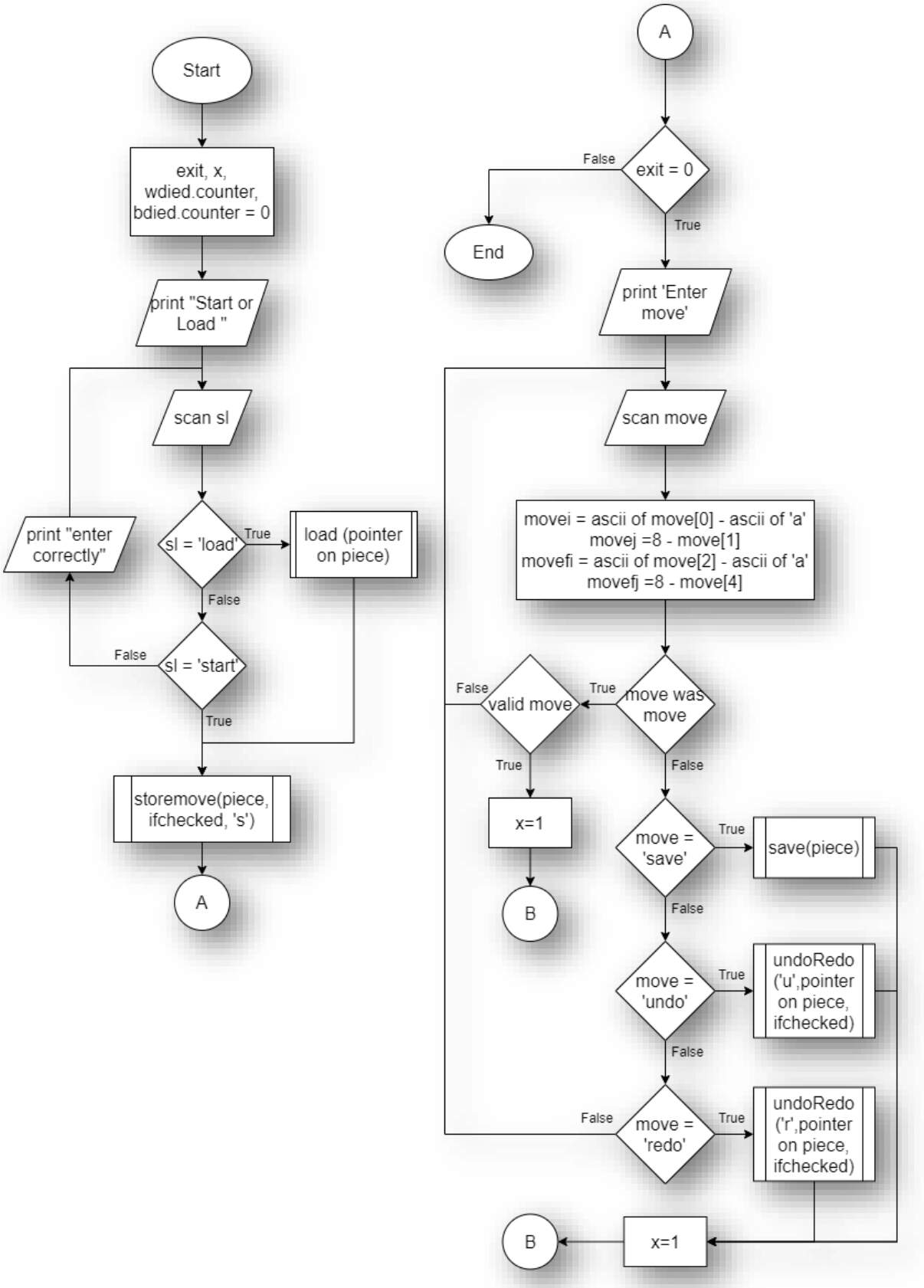
9. Save :

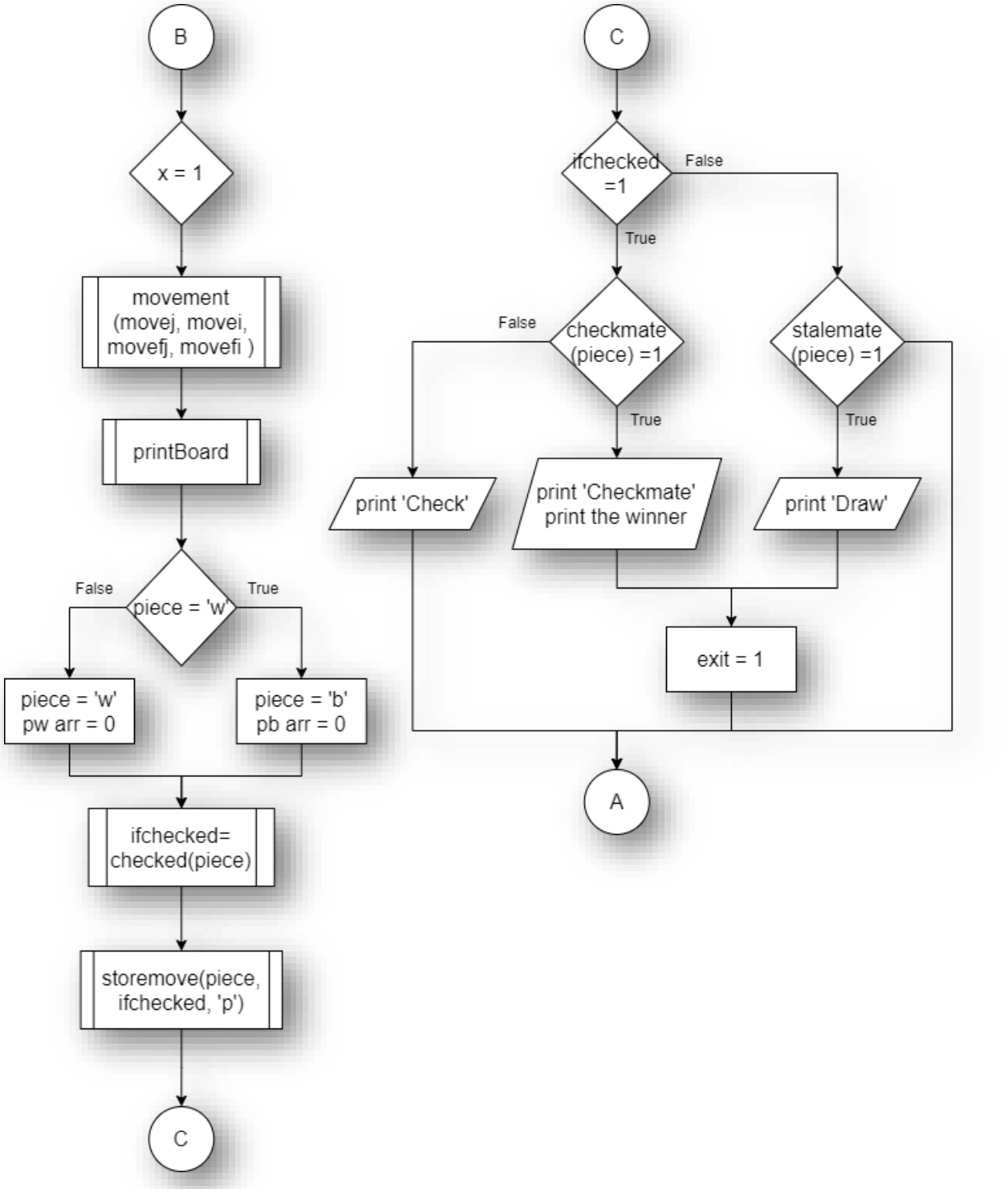

Save
print "Enter the name of the save file"
Input file1
Create a file with name file1
i=0
j=0
i<8 and j<8
False
True
store board [i][j] in the file
j = j + 1
j = 8
True
False
i = i + 1
j = 0
i = 0
i<4
False
True
store R[i] in the file after converting it into charcters
i = i + 1
i = 0
i<8
True
store pw[i] in the file after converting it into charcters
i = i + 1
i = 0
i<8
False
True
store pb[i] in the file after converting it into charcters
i = i + 1
store piece in the file
i = 0
i < no of white died pieces
False
True
store white died pieces in the file
i = i + 1
i = 0
i < no of black died pieces
False
True
store black died pieces in the file
i = i + 1
Close The file
End

10. Load :


Load
print "Enter the name of the load file"
Input name
print "Not Found"
there is a file with that name
False
True
Open the file
we reached the end of file
True
False
i = 0
j = 0
i<8 and j<8
False
True
Board[i][j] = c(character from the file)
i = i + 1
j = 0
j = j + 1
j = 8
True
False
8 <= i <12
False
True
R[i-8] = c after convert the characters into numbers
i = i + 1
12 <= i <20
False
True
pw[i-12] = c after convert the characters into numbers
i = i + 1
20 <= i <28
True
pb[i-20] = c after convert the characters into numbers
i = i + 1
i=28
True
piece = c
i = i + 1
i=29 and c is lower case
True
Load the white died pieces
Increase counter of white dead pieces
i=29 and c is upper case
True
Load the black died pieces
Increase counter of black dead pieces
Close The file
End

11. Game loop or the flow :




B
x = 1
movement
(movej, movei,
movefj, movefi)
printBoard
False
piece = 'w'
True
piece = 'w'
pw arr = 0
piece = 'b'
pb arr = 0
ifchecked=
checked(piece)
storemove(piece,
ifchecked, 'p')
C
C
ifchecked
=1
False
True
False
checkmate
(piece) =1
stalemate
(piece) =1
True
True
print 'Check'
print 'Checkmate'
print the winner
print 'Draw'
exit = 1
A

User Manual:

- First you can choose to start a new game or to load a saved game

1- Load:-
   Type load
   Then type the name of the saved file
2- Save:-
   Type start

- During the game :-

1- Move:-
   Type the move in form of b1a3
   b1 refers to the piece wanted to move indexes (b refers to the vertical index and 1 refers to the horizontal index)
   a3 refers to the place the wanted piece will move to(a refers to the vertical index and 3 refers to the horizontal index)
   the program will accept the move if its valid.
   each piece has has its own valid moves
   Rook : moves vertically or horizantallly only ,can move any number of squares but cannot leap over other piece
   Bishop : moves diagonally only ,can move any number of squares but cannot leap over other piece
   Queen: it can moves vertically , horizontally or diagonally only ,can move any number of squares but cannot leap over other piece
   Knight; it can moves only in L shape (two squares vertically and one square horizontally or two squares horizontally and one square vertically) .The knight is the only piece that can leap over other pieces
   King : can move to any one from the 8 adjacent squares
   It has a special move called castling and it happens when the king and the rook didn't move before and there isn't any piece between them and the king and the two adjacent squares in the direction of the rook are not in check . in this case the king can move two squares in the direction of the rook and the rook moves to the square at the other direction of the king.
   Pawn:the pawn cannot move backward and has three kinds of moves
   1-one square to the forward and this is the normal move
   2-two squares to the forward and this can happen only if this the first time it moves
   3-one square to the diagonal in case there is a piece from the other color there
   From the special things in the pawn is the promotion which happens in case the pawn reaches the last row of the board, it is the only case that you can add fifth character in the input .the fifth character can be (b-q-r-n).
   The pawn can do special way to eat the other colors pawn that is the enpassent it can happens when the pawn is two squares away from the adjacent other color pawn ,so the other color pawn moves two squares to be a side of the pawn to escape from eating .But the pawn still can eat It by the enpassent which means eating it without staying in its square.
   If the move is valid according to its type its still not valid until checking for the check as you can't move any move you want in case of check you just can move any move can remove the check.

2-Save:-

Type save
Then type the name of the saved file
Then you have two choices to continue the game or not
1-to continue : type y
2-to end the game : type n

3-Undo:-

Type undo

4-Redo:-

Type redo

- The game ends in 2 ways

1- Check mate

It happens when there is a check but there isn't any allowed move to the king to escape and there isn't any piece can block the bath to the king or eat the piece that causes the check.

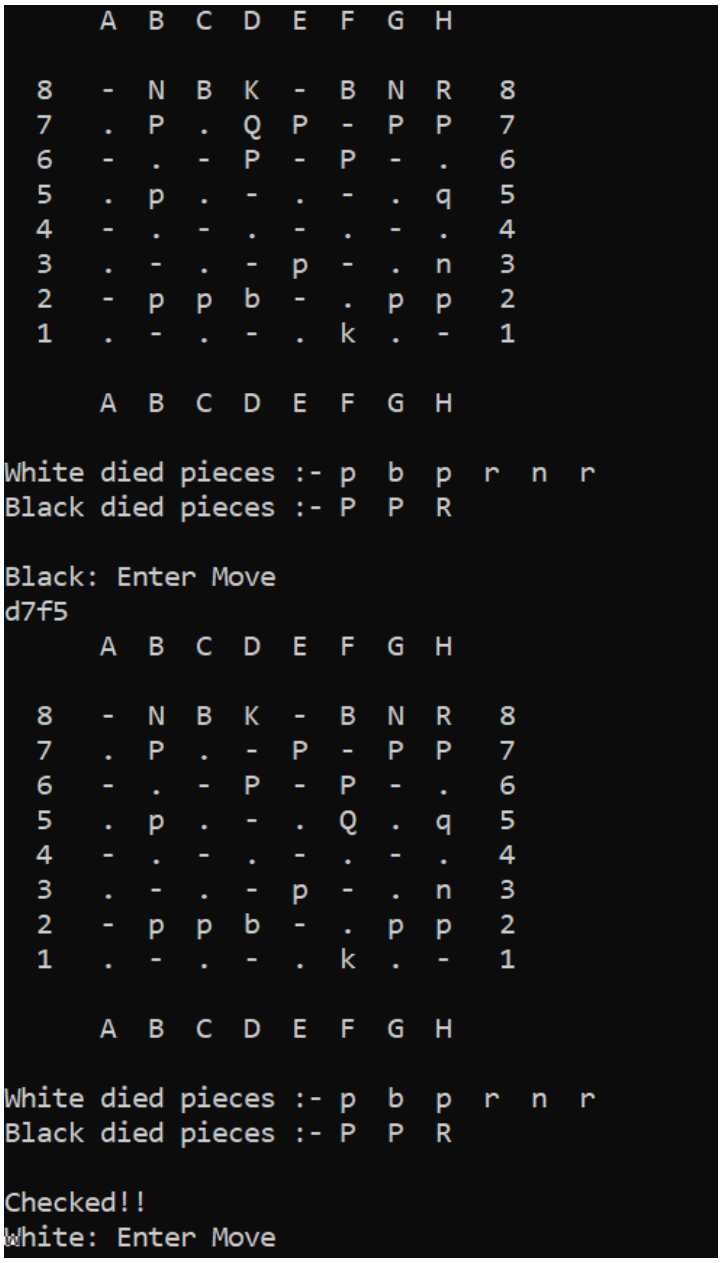
2- Draw

It happens when there is a stalemate (which is the player that is about to play is not in check but has not any legal move) or in case of lack of checkmate material (each player have one of the following : just a king , a king and bishop or a king , a king and a knight or a king and two knights)

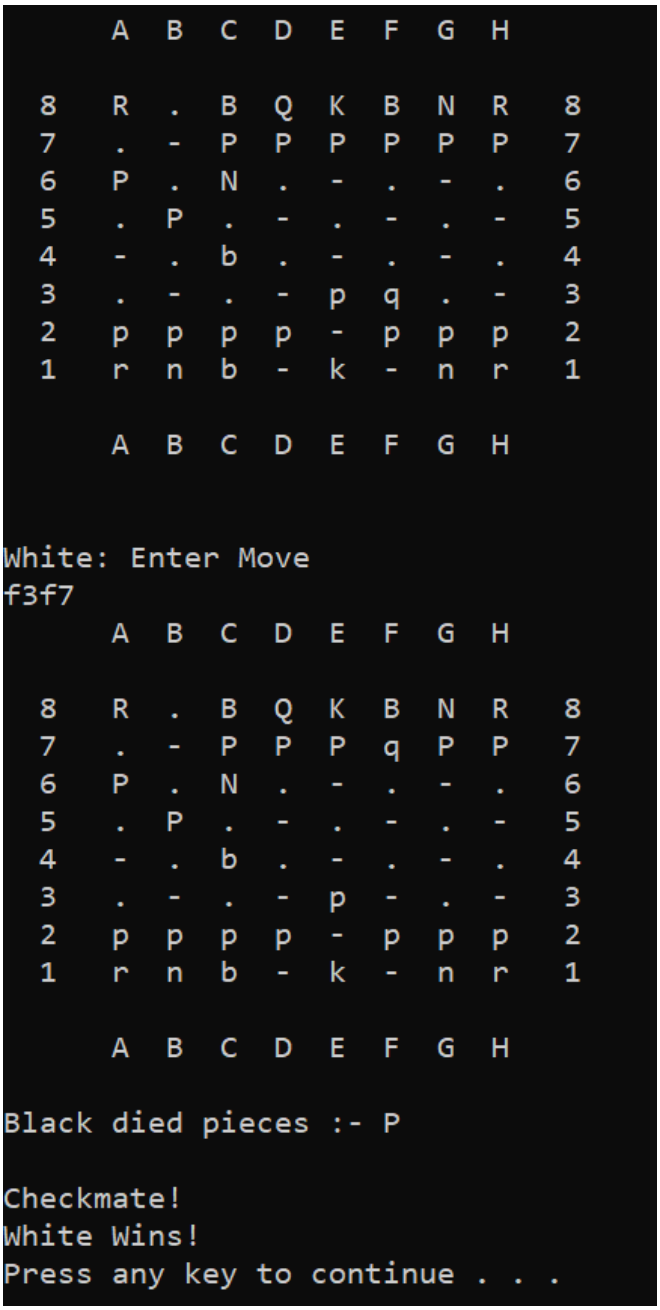
- All the inputs can be lower cases or upper cases letters.

Sample runs:-

1. Check :



2. Checkmate :



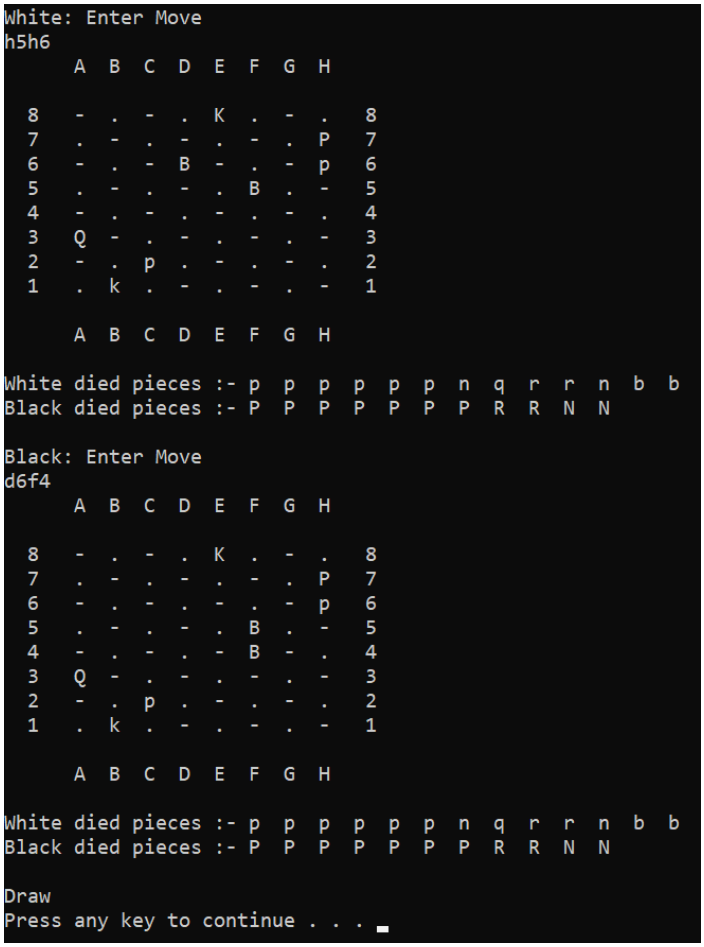
3. Castling :

```
f3g4
     A  B  C  D  E  F  G  H

  8  R  .  -  .  K  B  N  R  8
  7  P  -  P  N  P  P  P  P  7
  6  -  P  -  Q  -  .  -  .  6
  5  .  -  .  -  .  -  .  -  5
  4  -  .  -  p  -  .  p  .  4
  3  .  -  .  -  .  -  .  -  3
  2  p  p  p  .  -  .  p  p  2
  1  r  n  b  q  k  b  n  r  1

     A  B  C  D  E  F  G  H

White died pieces :- p
Black died pieces :- P  B

Black: Enter Move
e8c8
     A  B  C  D  E  F  G  H

  8  -  .  K  R  -  B  N  R  8
  7  P  -  P  N  P  P  P  P  7
  6  -  P  -  Q  -  .  -  .  6
  5  .  -  .  -  .  -  .  -  5
  4  -  .  -  p  -  .  p  .  4
  3  .  -  .  -  .  -  .  -  3
  2  p  p  p  .  -  .  p  p  2
  1  r  n  b  q  k  b  n  r  1

     A  B  C  D  E  F  G  H

White died pieces :- p
Black died pieces :- P  B
```

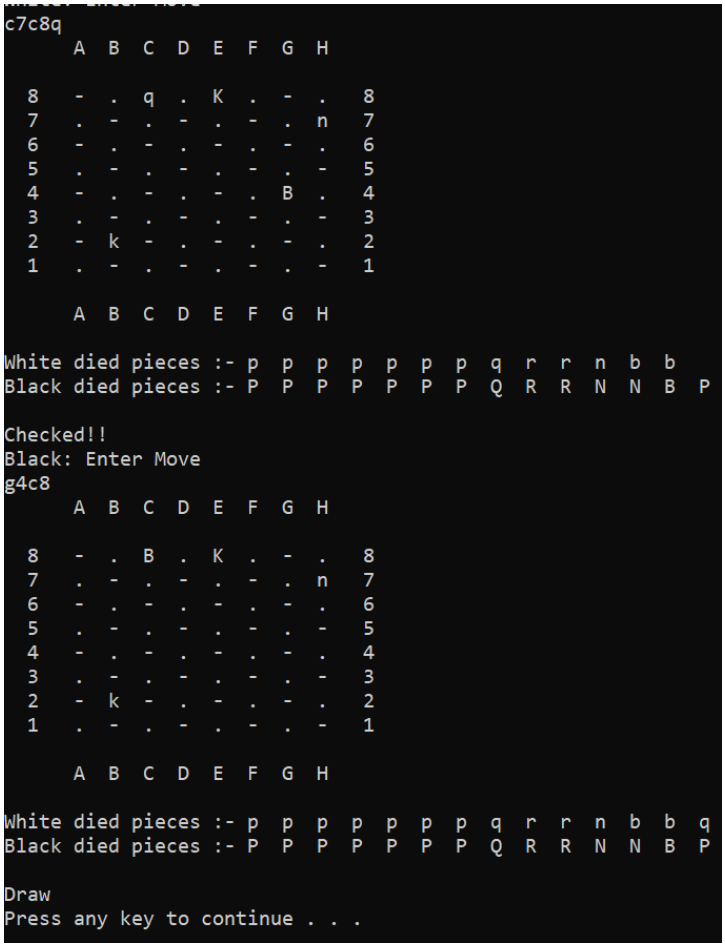
4. Enpassent

```
Black: Enter Move
d7d5
     A  B  C  D  E  F  G  H

  8  R  N  B  Q  K  B  N  R  8
  7  P  -  P  -  P  P  P  P  7
  6  -  P  -  .  -  .  -  .  6
  5  .  -  .  P  p  -  .  -  5
  4  -  .  -  .  -  .  -  .  4
  3  .  -  .  -  .  -  .  -  3
  2  p  p  p  p  -  p  p  p  2
  1  r  n  b  q  k  b  n  r  1

     A  B  C  D  E  F  G  H


White: Enter Move
e5d6
     A  B  C  D  E  F  G  H

  8  R  N  B  Q  K  B  N  R  8
  7  P  -  P  -  P  P  P  P  7
  6  -  P  -  p  -  .  -  .  6
  5  .  -  .  -  .  -  .  -  5
  4  -  .  -  .  -  .  -  .  4
  3  .  -  .  -  .  -  .  -  3
  2  p  p  p  p  -  p  p  p  2
  1  r  n  b  q  k  b  n  r  1

     A  B  C  D  E  F  G  H

Black died pieces :- P
```

5. Draw (with stalemate and lack of checkmate material)



```
White: Enter Move
h5h6
     A  B  C  D  E  F  G  H

  8   -  .  -  .  K  .  -  .   8
  7   .  -  .  -  .  -  .  P   7
  6   -  .  -  B  -  .  -  p   6
  5   .  -  .  -  .  B  .  -   5
  4   -  .  -  .  -  .  -  .   4
  3   Q  -  .  -  .  -  .  -   3
  2   -  .  p  .  -  .  -  .   2
  1   .  k  .  -  .  -  .  -   1

     A  B  C  D  E  F  G  H

White died pieces :- p  p  p  p  p  p  n  q  r  r  n  b  b
Black died pieces :- P  P  P  P  P  P  P  R  R  N  N

Black: Enter Move
d6f4
     A  B  C  D  E  F  G  H

  8   -  .  -  .  K  .  -  .   8
  7   .  -  .  -  .  -  .  P   7
  6   -  .  -  .  -  .  -  p   6
  5   .  -  .  -  .  B  .  -   5
  4   -  .  -  .  -  B  -  .   4
  3   Q  -  .  -  .  -  .  -   3
  2   -  .  p  .  -  .  -  .   2
  1   .  k  .  -  .  -  .  -   1

     A  B  C  D  E  F  G  H

White died pieces :- p  p  p  p  p  p  n  q  r  r  n  b  b
Black died pieces :- P  P  P  P  P  P  P  R  R  N  N

Draw
Press any key to continue . . .
```



```
c7c8q
     A  B  C  D  E  F  G  H

  8   -  .  q  .  K  .  -  .   8
  7   .  -  .  -  .  -  .  n   7
  6   -  .  -  .  -  .  -  .   6
  5   .  -  .  -  .  -  .  -   5
  4   -  .  -  .  -  .  B  .   4
  3   .  -  .  -  .  -  .  -   3
  2   -  k  -  .  -  .  -  .   2
  1   .  -  .  -  .  -  .  -   1

     A  B  C  D  E  F  G  H

White died pieces :- p  p  p  p  p  p  p  q  r  r  n  b  b
Black died pieces :- P  P  P  P  P  P  P  Q  R  R  N  N  B  P

Checked!!
Black: Enter Move
g4c8
     A  B  C  D  E  F  G  H

  8   -  .  B  .  K  .  -  .   8
  7   .  -  .  -  .  -  .  n   7
  6   -  .  -  .  -  .  -  .   6
  5   .  -  .  -  .  -  .  -   5
  4   -  .  -  .  -  .  -  .   4
  3   .  -  .  -  .  -  .  -   3
  2   -  k  -  .  -  .  -  .   2
  1   .  -  .  -  .  -  .  -   1

     A  B  C  D  E  F  G  H

White died pieces :- p  p  p  p  p  p  p  q  r  r  n  b  b  q
Black died pieces :- P  P  P  P  P  P  P  Q  R  R  N  N  B  P

Draw
Press any key to continue . . .
```

6. Save :

```
     A  B  C  D  E  F  G  H

  8  -  .  K  R  -  B  N  R  8
  7  P  -  P  N  P  P  P  P  7
  6  -  P  -  Q  -  .  -  .  6
  5  .  -  .  -  .  -  .  -  5
  4  -  .  -  p  -  .  p  .  4
  3  .  -  .  -  .  -  .  -  3
  2  p  p  p  .  -  .  p  p  2
  1  r  n  b  q  k  b  n  r  1

     A  B  C  D  E  F  G  H

White died pieces :- p
Black died pieces :- P  B

White: Enter Move
save
Enter the name of the save file
testsave
Continue(Y/N):y
     A  B  C  D  E  F  G  H

  8  -  .  K  R  -  B  N  R  8
  7  P  -  P  N  P  P  P  P  7
  6  -  P  -  Q  -  .  -  .  6
  5  .  -  .  -  .  -  .  -  5
  4  -  .  -  p  -  .  p  .  4
  3  .  -  .  -  .  -  .  -  3
  2  p  p  p  .  -  .  p  p  2
  1  r  n  b  q  k  b  n  r  1

     A  B  C  D  E  F  G  H

White died pieces :- p
Black died pieces :- P  B
```

7. load :

```
(start/load)
Make Your Choice
load
Enter the name of the load file
testsave
     A  B  C  D  E  F  G  H

  8  -  .  K  R  -  B  N  R  8
  7  P  -  P  N  P  P  P  P  7
  6  -  P  -  Q  -  .  -  .  6
  5  .  -  .  -  .  -  .  -  5
  4  -  .  -  p  -  .  p  .  4
  3  .  -  .  -  .  -  .  -  3
  2  p  p  p  .  -  .  p  p  2
  1  r  n  b  q  k  b  n  r  1

     A  B  C  D  E  F  G  H

White died pieces :- p
Black died pieces :- P  B

White: Enter Move
```

References:

https://en.wikipedia.org/wiki/Chess