**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

1. 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

1. 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

1. 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

1. 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

1. 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

1. 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

1. 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

1. 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

1. 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

1. 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"

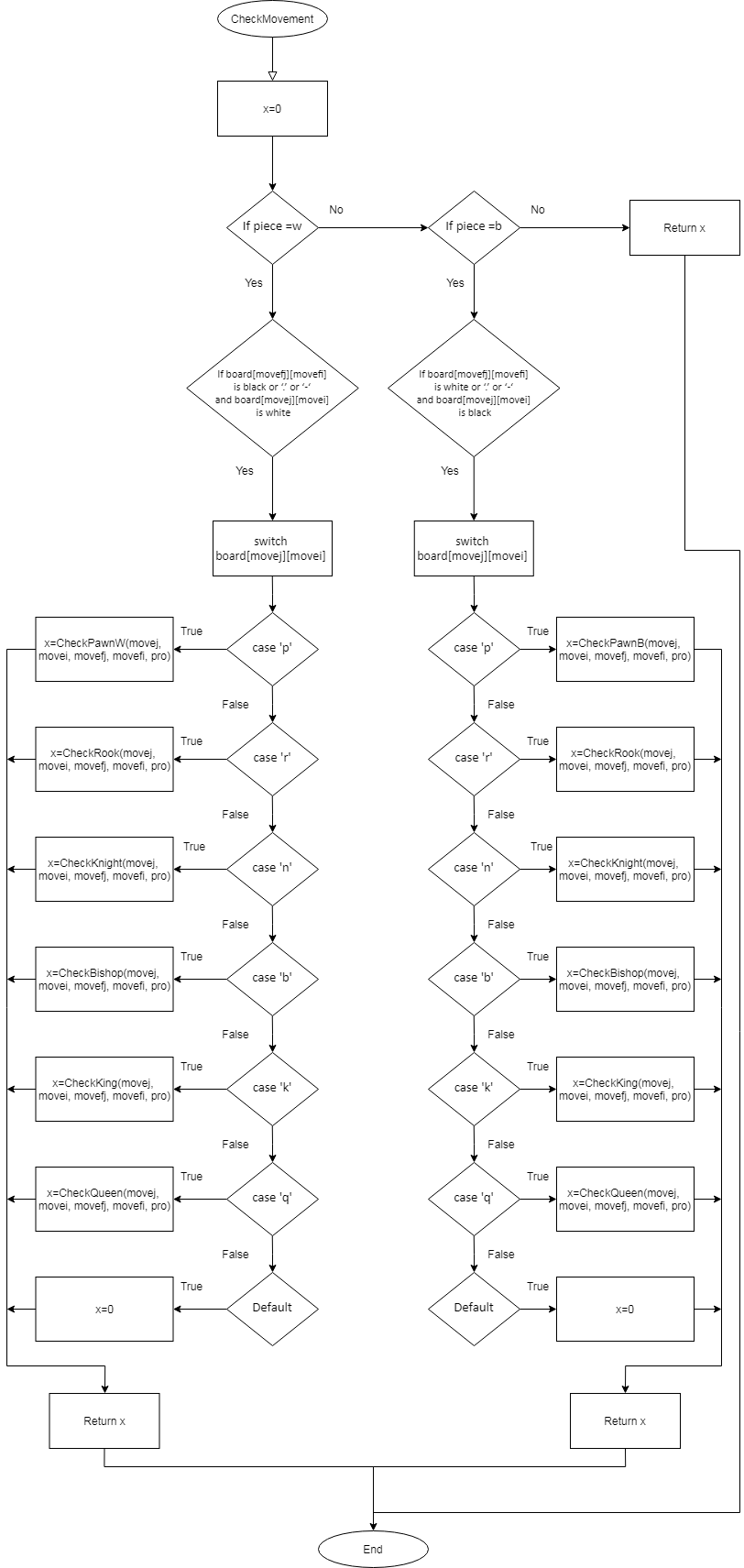
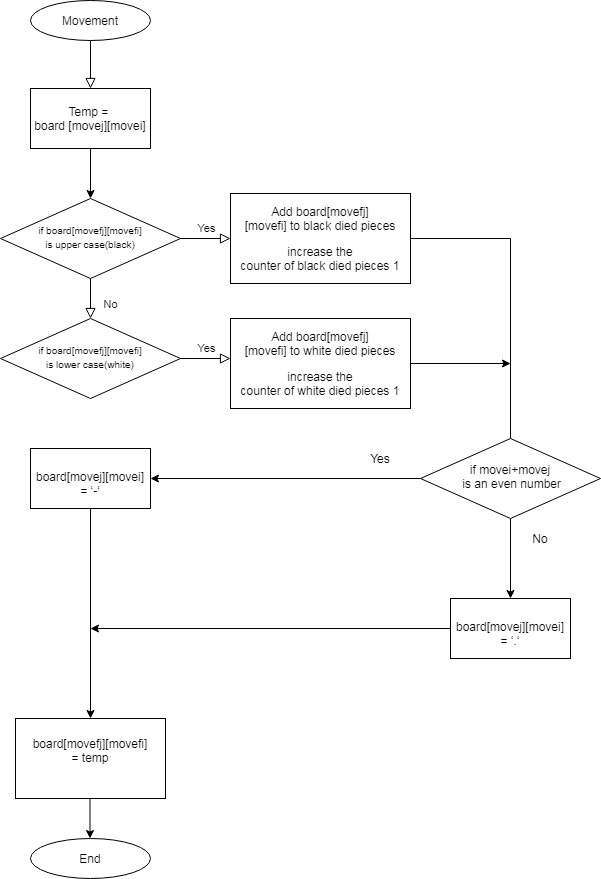
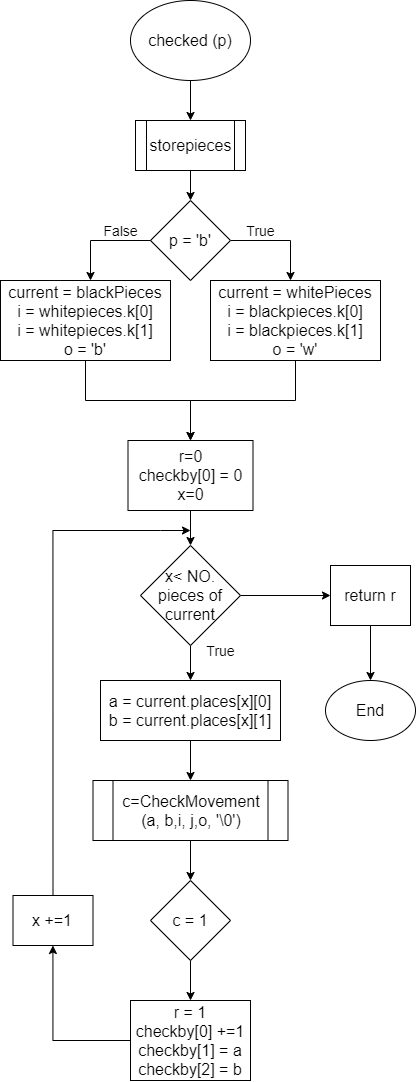
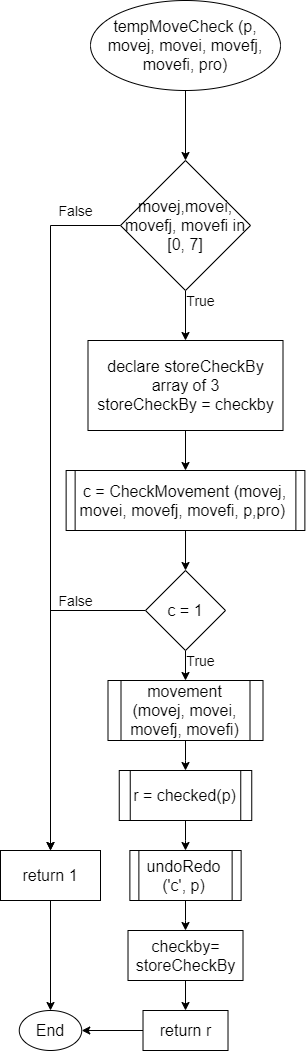
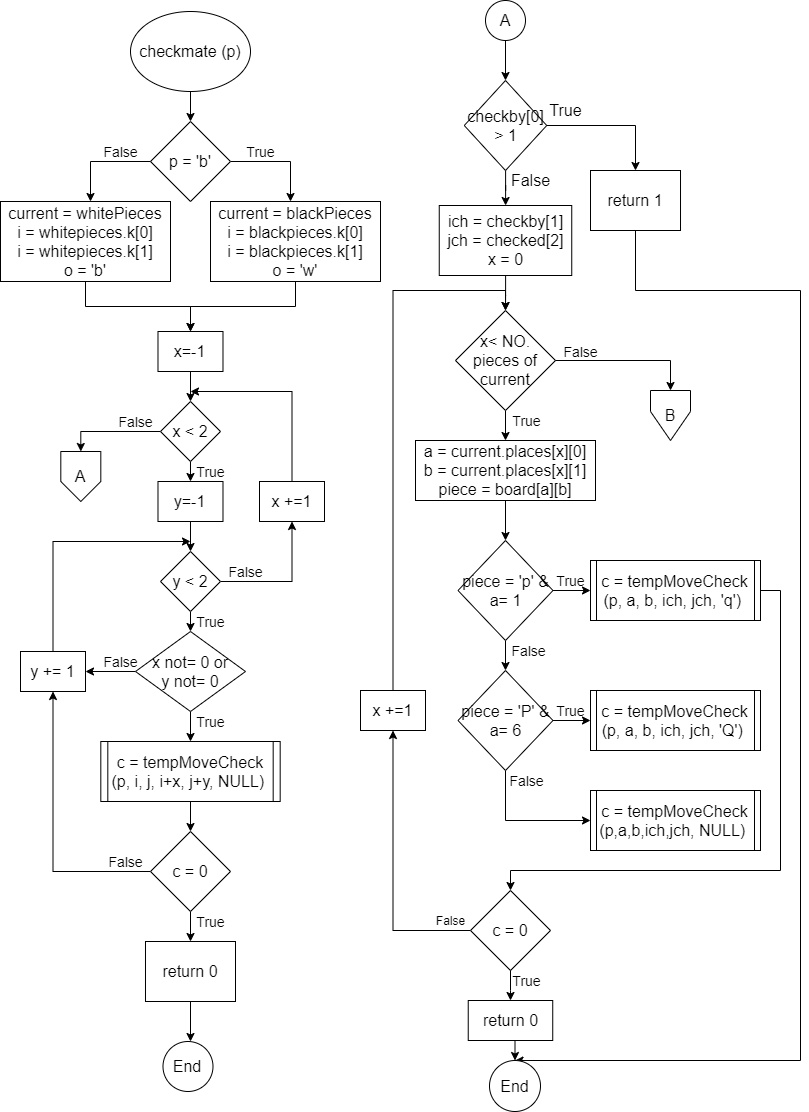
else if there stalemate :

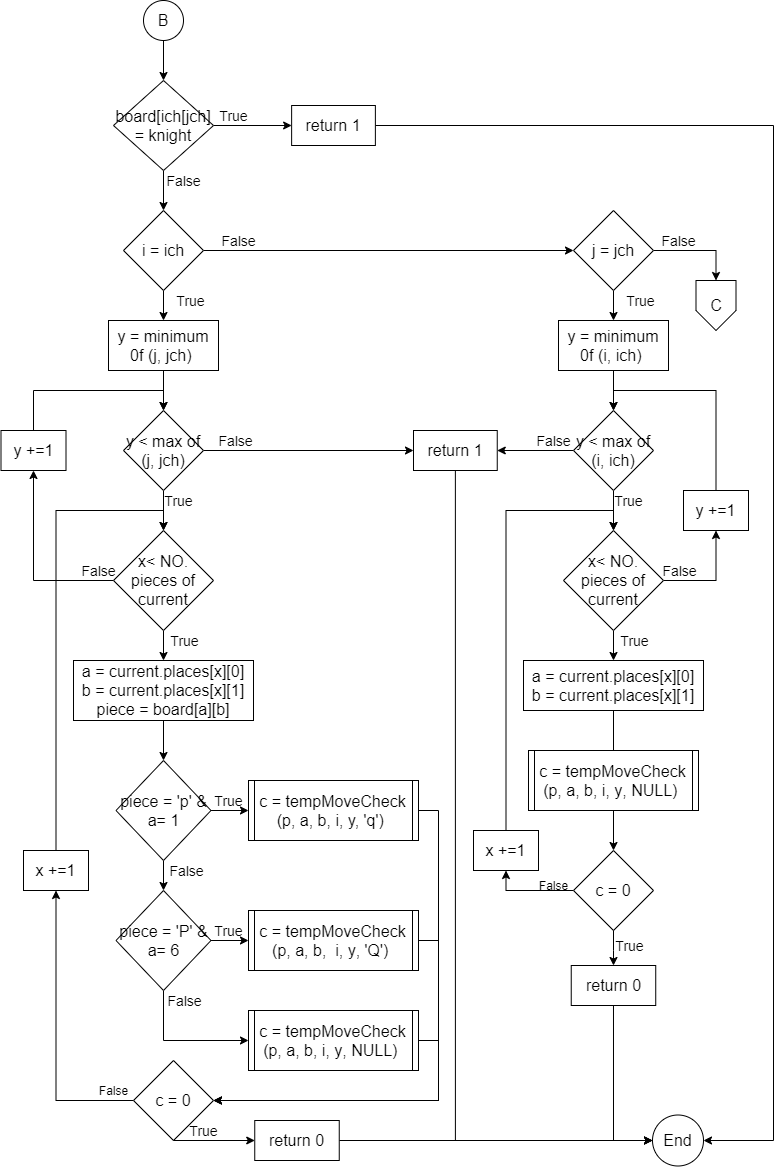
print : "Draw"

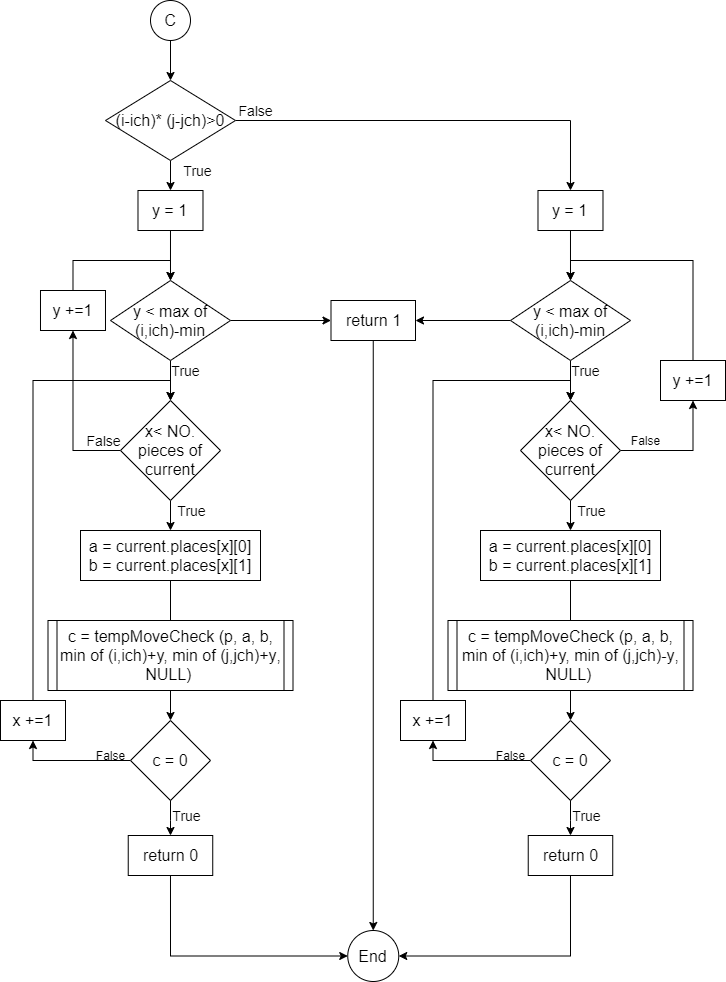
exit = 1

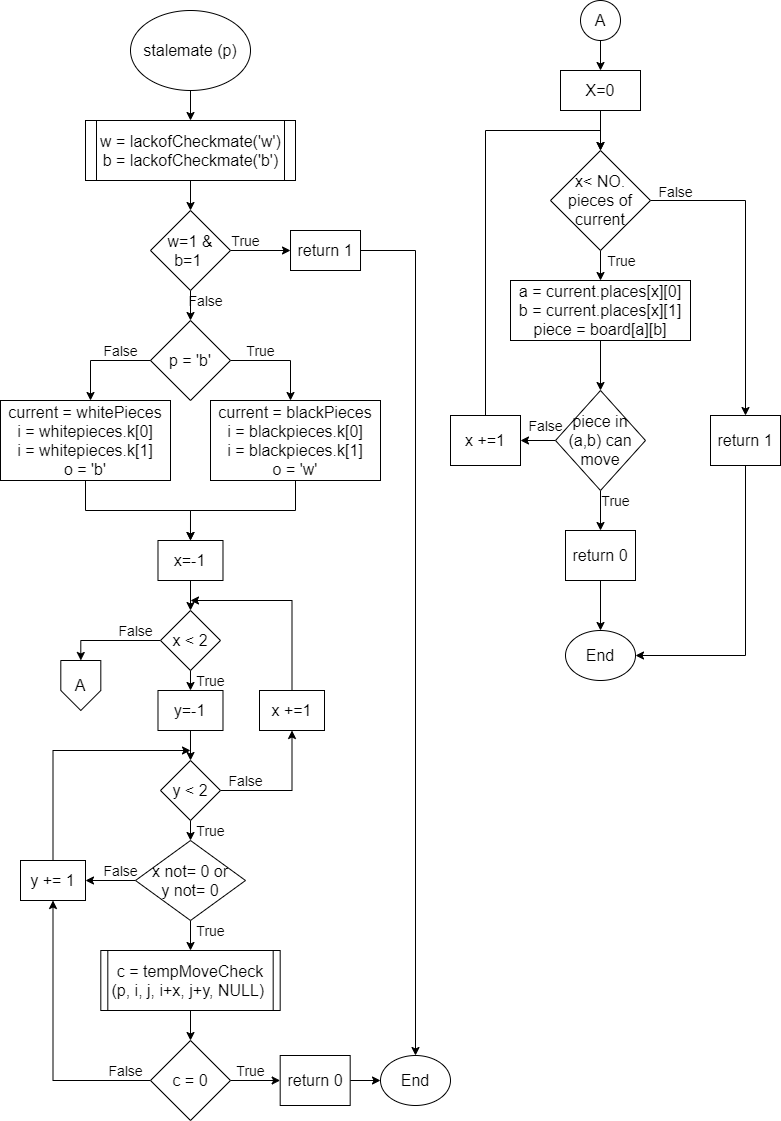
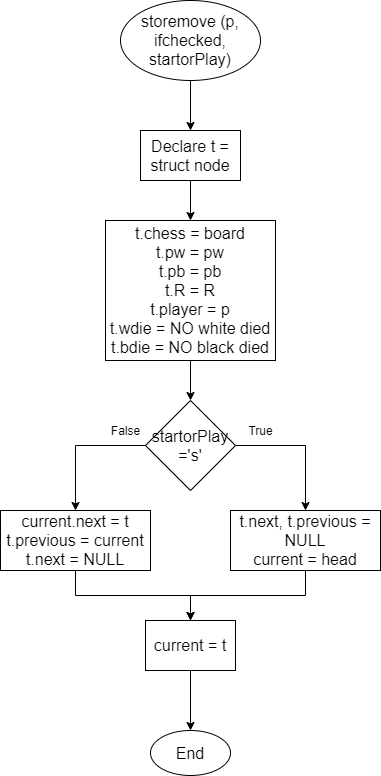
end of loop 1

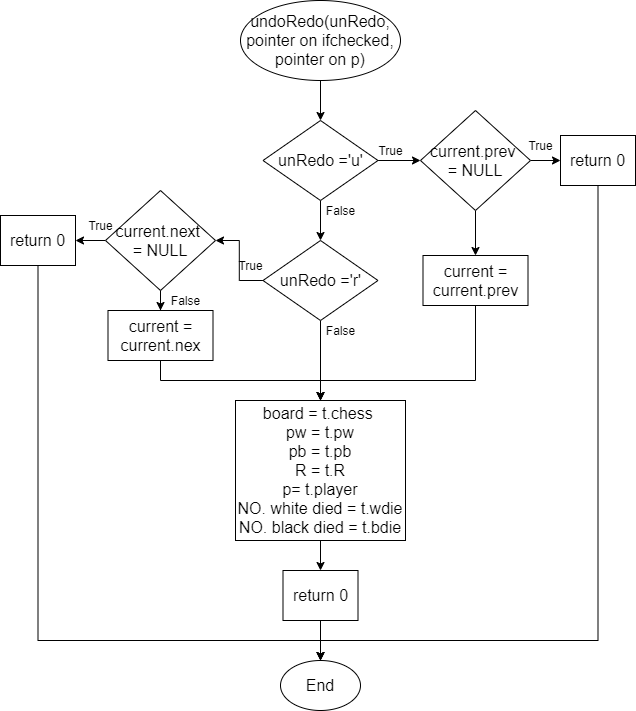
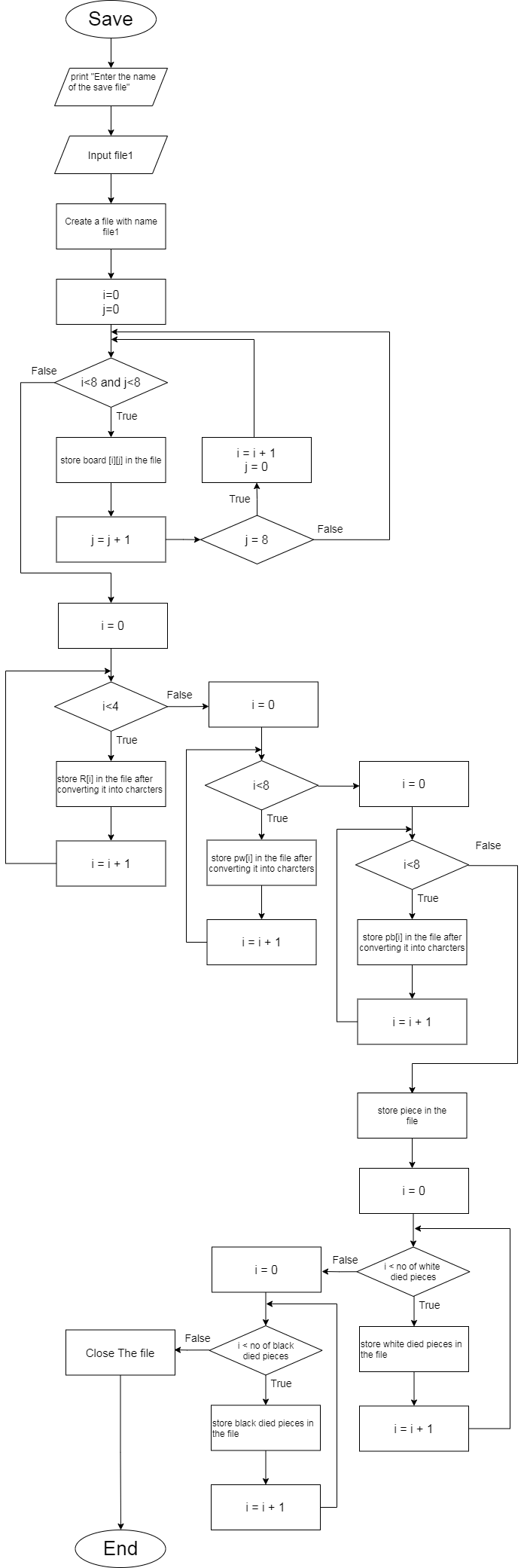
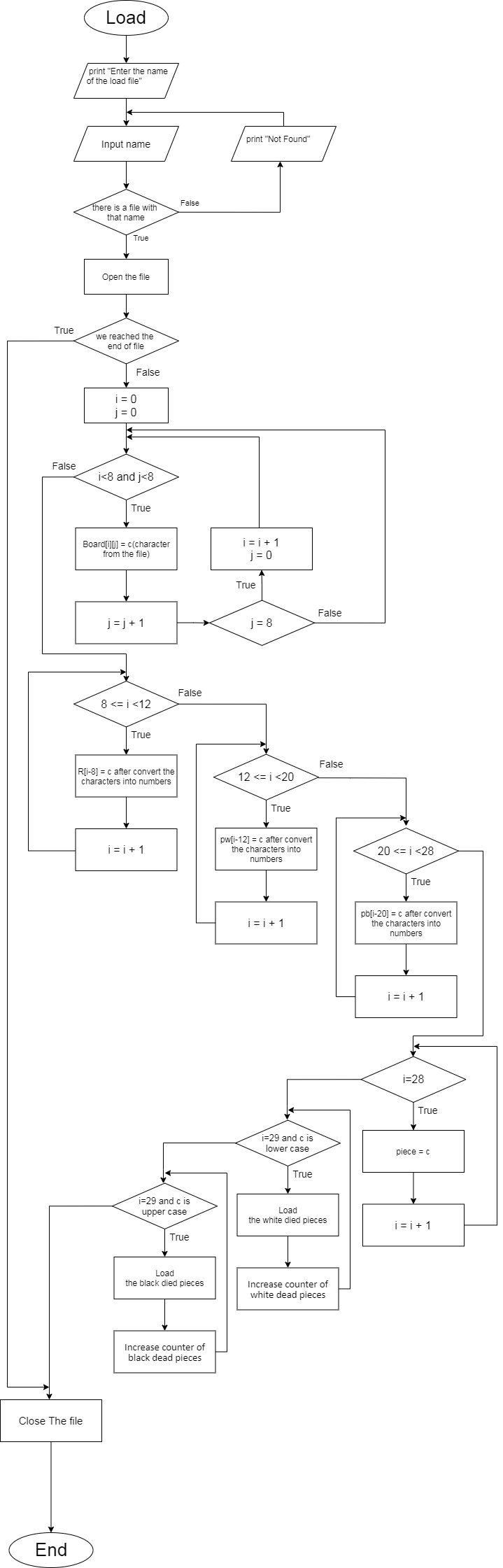
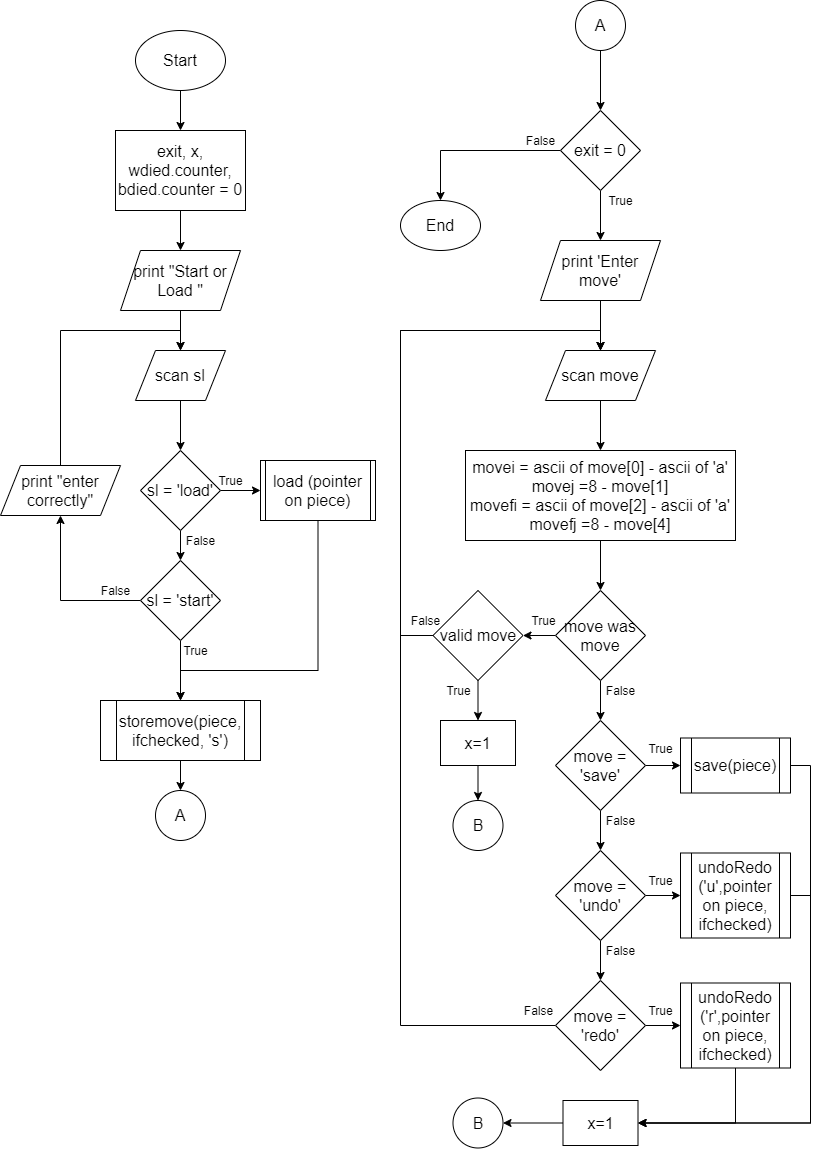
**Flowchart​ for the main algorithms :**

1. Checkmovement :
2. Movement:
3. Check : 4. tempMoveCheck :
4. Checkmate :





1. Stalemate :
2. Storemove :

1. undoRedo :
2. Save :
3. Load :
4. Game loop or the flow :

