chess report

Description of the game:
The application simulates a Two-Player chess game, where blach and white squares of the board are represented by " . " and " - " respectively, and White pieces are represented by “p”,“r”,“n”,“b”,“q”,“k” for pawns ,rooks, knights, bishops, queen and king respectively.Black pieces are represented by the same letters but capitalized. The game supports " undo & redo " utility, and it can be saved and retrieved at any time in the game.

Design overview:
The overall game is divided mainly into two main parts, firstly, is the essence and core logic of the game which is a permenant WHILE LOOP that alternate the turns between the two players and apply changes to the chessboard, this while loop is broken through one of two ways, either a stalemate or a checkmate.Second, two main functions that support the main while loop in the game, the first function is (CHECKMATE) which is used to check checkmate case every loop repitision. and the other function is (STALEMATE) to check stalemate case.

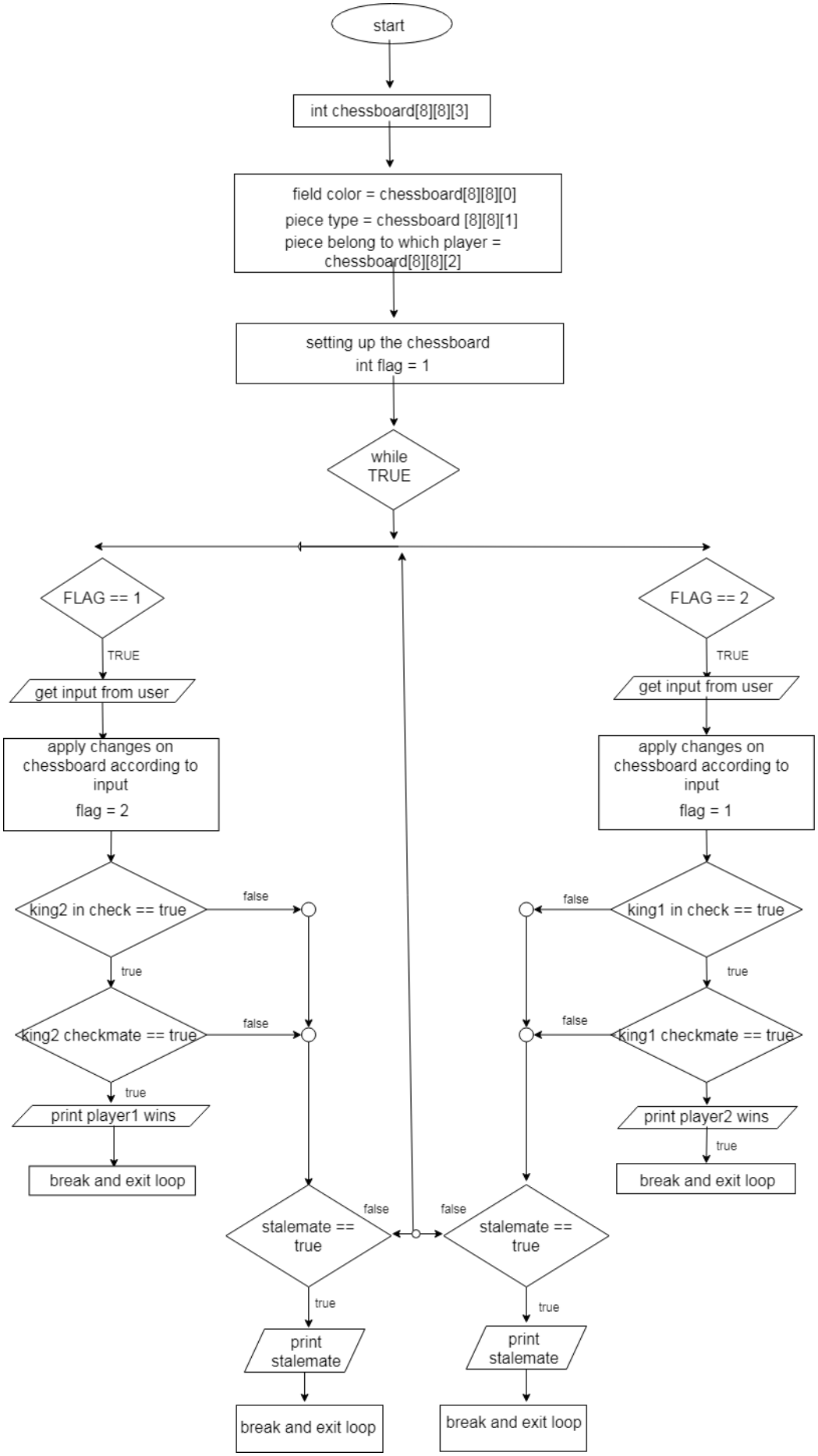
Data structures used:
Integer array: is used to represent the chessboard, where every character is represented using its ASCI code.

Stack: is used to store the game every play in order to support "undo & redo" utility.

Importat functions used:
check_position: this function is passed a certain location in the chessboard and an integer detector. where it checks if the given position is vulnerable to a certain player determined by the detector – in other word it determines if the given position is reachable by the determined player. if the position is vulnerable it returns 1, if not it returns 0.

checkmate: this function checkes whether there is a checkmate or not. This function is passed as arguments each of the chessboard, position of the checked king, the threatening piece, position of the threatening piece and whom his king is ckecked. If checkmate occures the function returns 1 , and returns 0 otherwise.

stalemate: this function checkes whether there is a stalemate or not. This function is passed as arguments each of the chessboard, position of the king and a detector.It returns 1 in case of stalemate, and returns 0 otherwise.


start
int chessboard[8][8][3]
field color = chessboard[8][8][0]
piece type = chessboard [8][8][1]
piece belong to which player = chessboard[8][8][2]
setting up the chessboard
int flag = 1
while TRUE
FLAG == 1
TRUE
get input from user
apply changes on chessboard according to input
flag = 2
king2 in check == true
false
true
king2 checkmate == true
false
true
print player1 wins
break and exit loop
FLAG == 2
TRUE
get input from user
apply changes on chessboard according to input
flag = 1
king1 in check == true
false
true
king1 checkmate == true
false
true
print player2 wins
break and exit loop
stalemate == true
false
true
print stalemate
break and exit loop
stalemate == true
false
true
print stalemate
break and exit loop

psuedo code:

1. Declaration of 3 dimentional array (chessboard) to carry all information related to the chessboard.
2. Path to this array all required values in order to set it up for the start of the game.
3. Declare an integer variable (flag) that takes the value 1 if it's player's 1 turn, and takes 2 if it's player's 2 turn.
4. We write a while loop that doesn't terminate unless checkmate or stalemate occurs.
5. Get input from user then change the corresponding values in the (chessboard) array in order to meet the new changes. If any pieces are eaten, they are appended in (player's_eaten_pieces) previously declared array.
6. Every play is registered and appended in the stack (undo_stack) previously declared.
7. We then check for checkmate and stalemate cases using the previously defined functions (checkmate & stalemate), if any function of them returns 1, the game terminates.
8. If the user inputs (undo) we overrides (5 & 6 & 7) steps and extract the latest appended play in the (undo_stack) and return it to user, then append this play in the (redo_stack) previously declared stack
9. If the user inputs (redo) we overrides (5 & 6 & 7) steps and extract the latest appended play in the (redo_stack) and return it to user, then append this play in the (undo_stack) .
10. If the user inputs (save) we output all important data of the game to an external file in order to be retrieved the next time we enter the game, then we exit the loop.

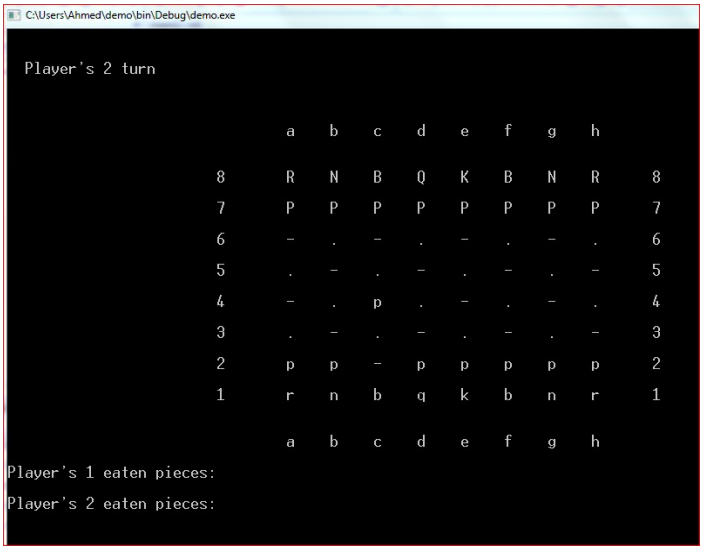
User manual:

- The user specifies his next move by entering the index of the piece he wishes to move, followed by the index of the square he wishes to move it to. (e.g. A3B4 will move the piece at A3 to the square B4). The only exception is in the case of promotion, where the user must specify an additional character indicating the desired piece. (e.g.H7H8B will promote pawn to a bishop).

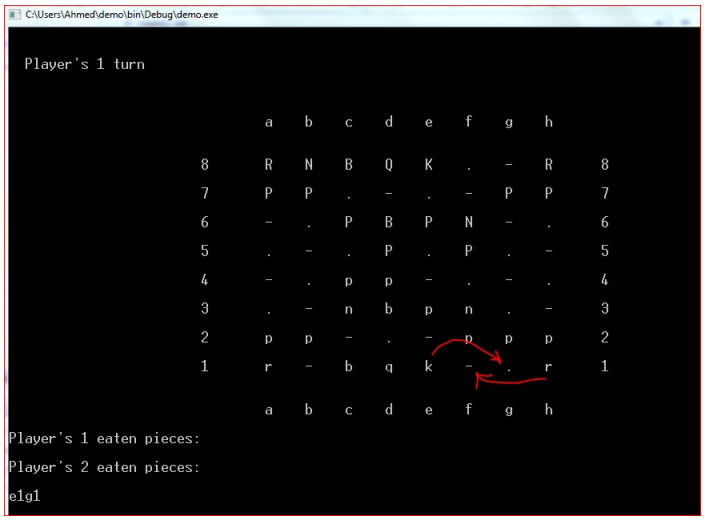
- To undo or redo any movement in the game, the user inputs (undo\ redo) then presses enter.
- To save the game at any time the user inputs (save) then presses enter.

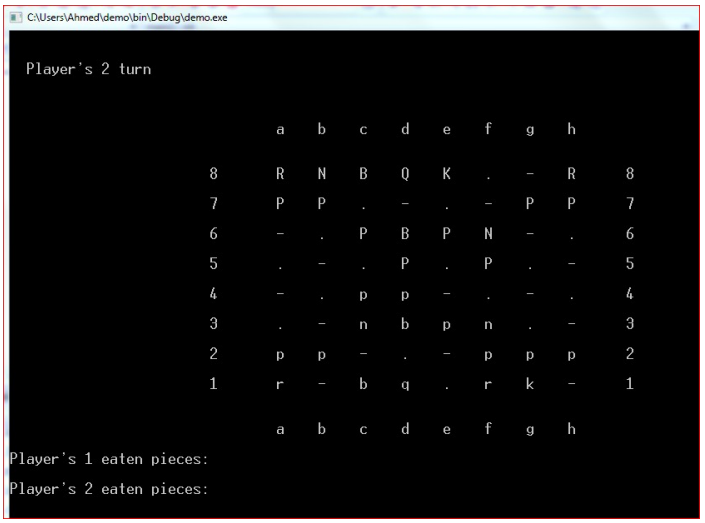
Test cases:

1) pawn movement

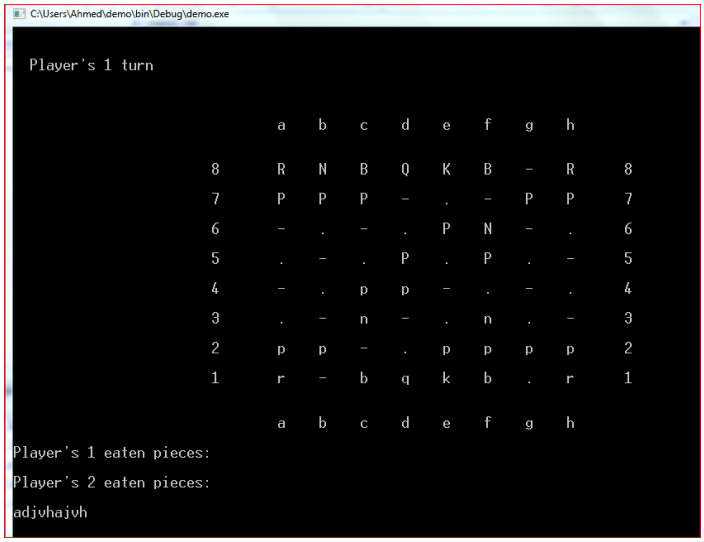


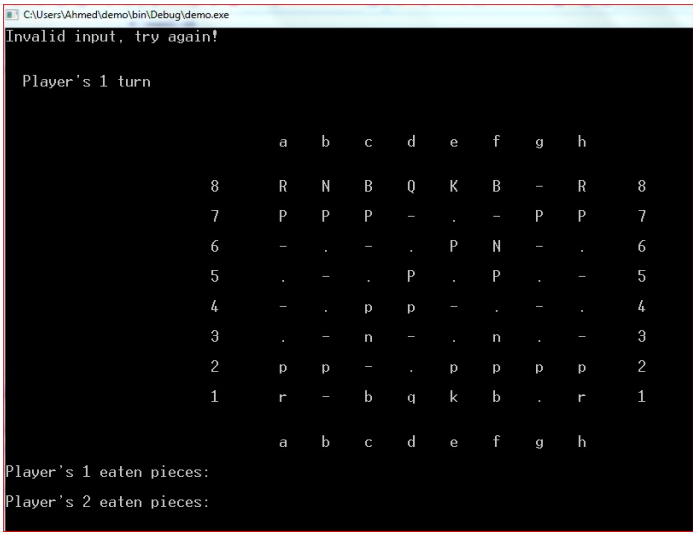
2) castling

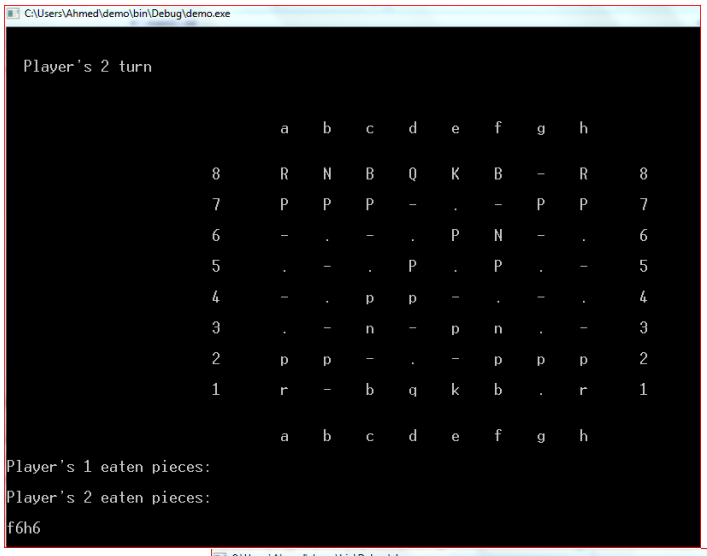


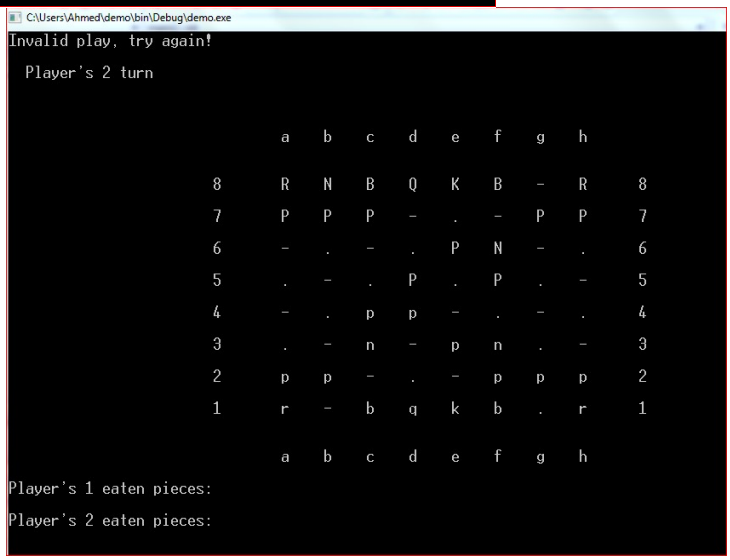


3) checking for error

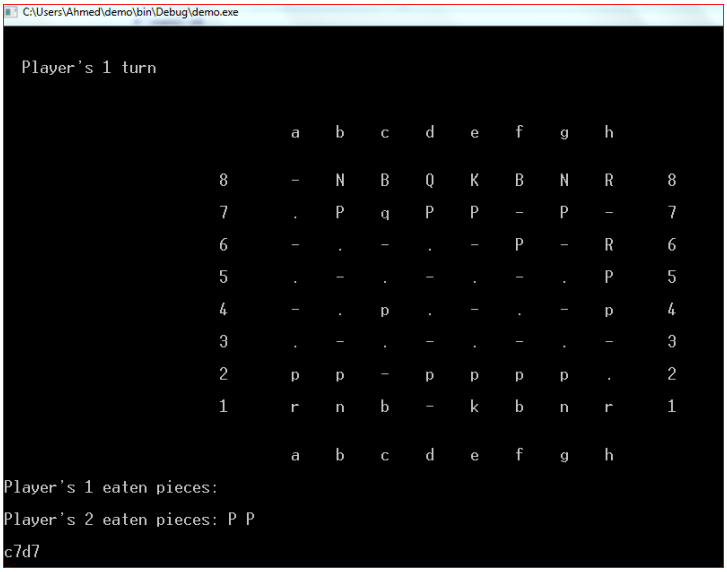


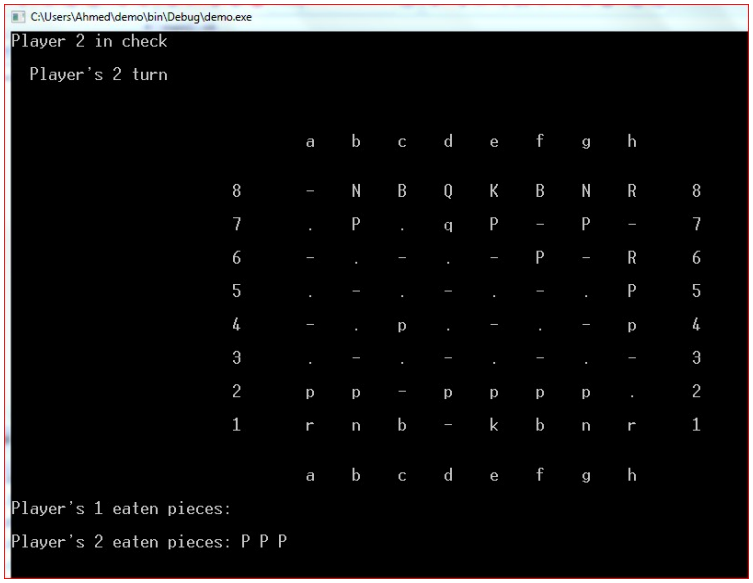




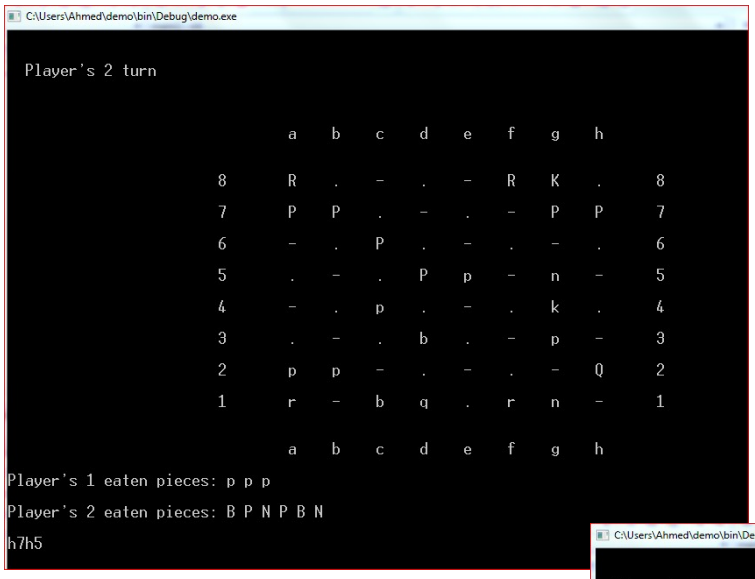


4) player in ckeck

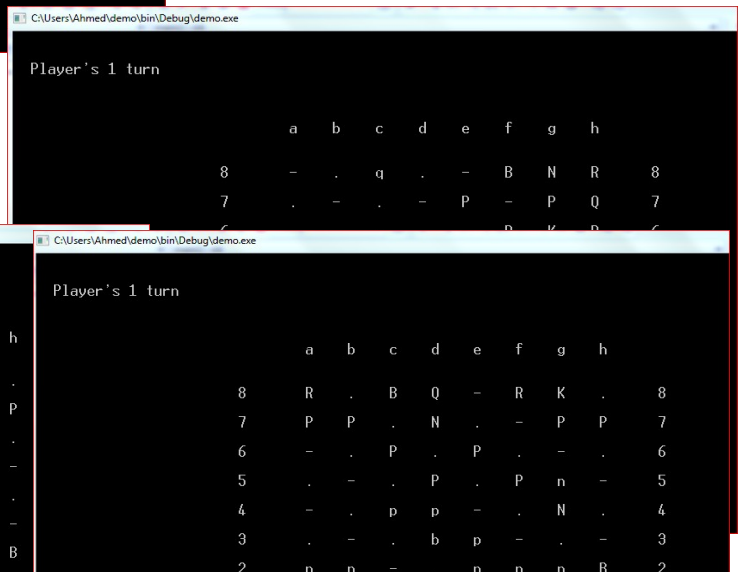




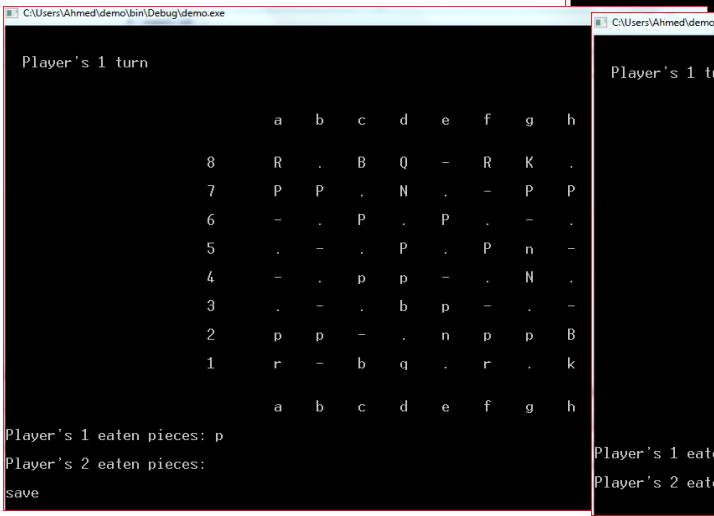
5) checkmate



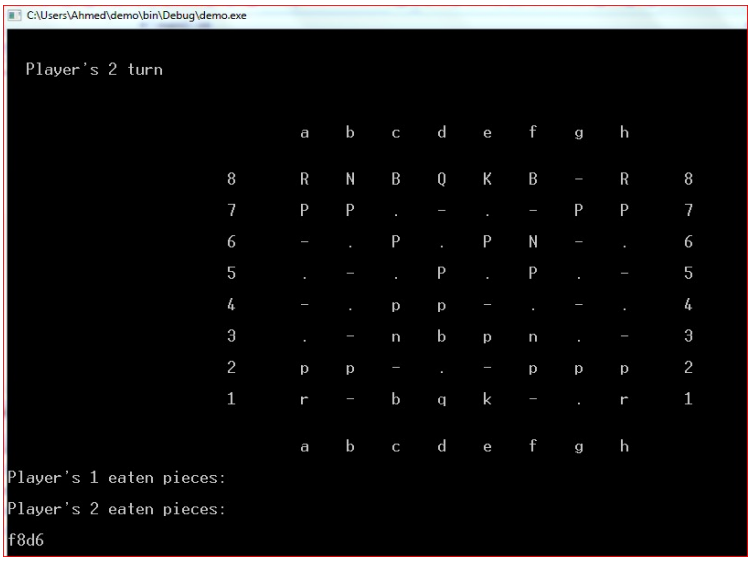
6) stalemate

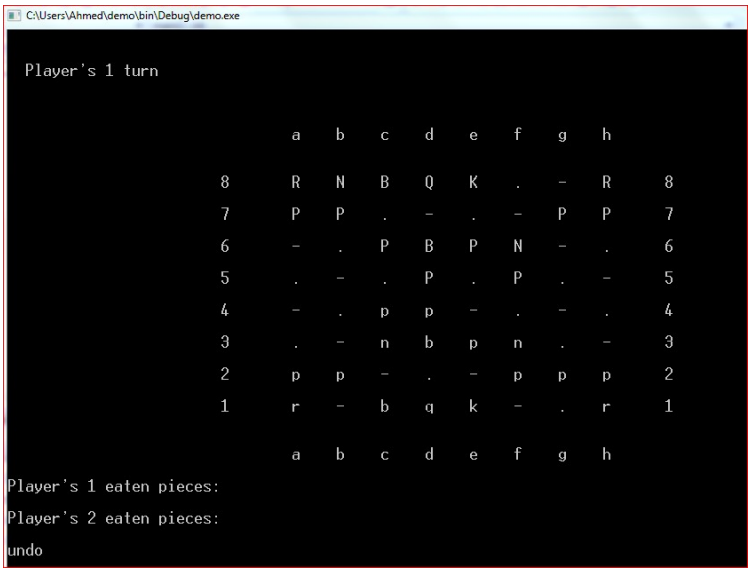


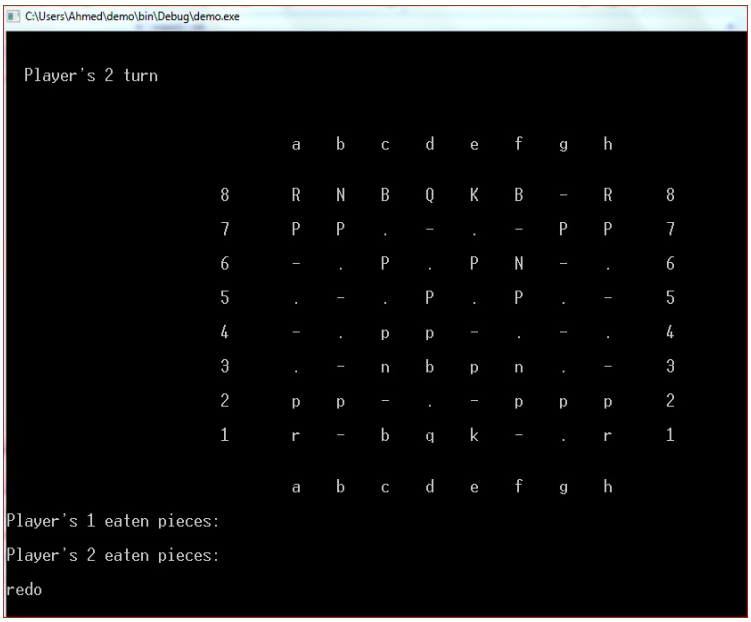
7) saving the game

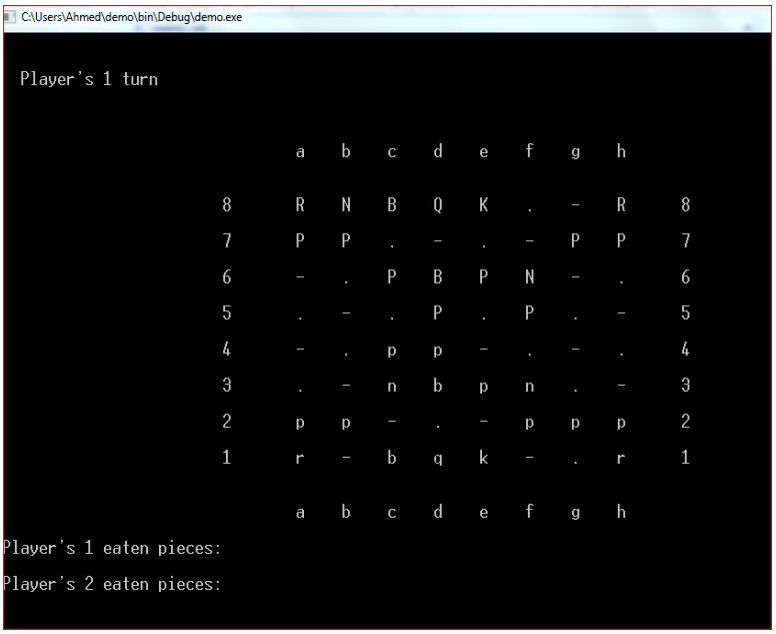


8) undo – redo


C:\Users\Ahmed\demo\bin\Debug\demo.exe
Player's 2 turn
a b c d e f g h
8 R N B Q K B - R 8
7 P P . - . - P P 7
6 - . P . P N - . 6
5 . - . P . P . - 5
4 - . p p - . - . 4
3 . - n b p n . - 3
2 p p - . - p p p 2
1 r - b q k - . r 1
a b c d e f g h
Player's 1 eaten pieces:
Player's 2 eaten pieces:
f8d6


C:\Users\Ahmed\demo\bin\Debug\demo.exe
Player's 1 turn
a b c d e f g h
8 R N B Q K . - R 8
7 P P . - . - P P 7
6 - . P B P N - . 6
5 . - . P . P . - 5
4 - . p p - . - . 4
3 . - n b p n . - 3
2 p p - . - p p p 2
1 r - b q k - . r 1
a b c d e f g h
Player's 1 eaten pieces:
Player's 2 eaten pieces:
undo


C:\Users\Ahmed\demo\bin\Debug\demo.exe
Player's 2 turn
a b c d e f g h
8 R N B Q K B - R 8
7 P P . - . - P P 7
6 - . P . P N - . 6
5 . - . P . P . - 5
4 - . p p - . - . 4
3 . - n b p n . - 3
2 p p - . - p p p 2
1 r - b q k - . r 1
a b c d e f g h
Player's 1 eaten pieces:
Player's 2 eaten pieces:
redo


C:\Users\Ahmed\demo\bin\Debug\demo.exe
Player's 1 turn
a b c d e f g h
8 R N B Q K . - R 8
7 P P . - . - P P 7
6 - . P B P N - . 6
5 . - . P . P . - 5
4 - . p p - . - . 4
3 . - n b p n . - 3
2 p p - . - p p p 2
1 r - b q k - . r 1
a b c d e f g h
Player's 1 eaten pieces:
Player's 2 eaten pieces: