Name	ID	Number in Department
Youssef Ahmed Saeed Zaki	19016903	90 → 93
Abdelmeniem Hany Abdelmeniem Mohamed	19017359	98 →101

Programming1

Final Project Report

Report containments:

Description of Application:

Our application is a multiplayer chess game application coded by C programming language and designed to be an online multiplayer game, where each player makes a move in the chess board alternatively till one of them wins by checkmating the other. The game can end in draw in some cases too.

Features provided:

- On each move, we give each player the opportunity to make some non-move options:
    - Resign the game.
    - Submit a draw request.
        - If other player agrees, game ends in draw.
    - Save the game.
    - Submit a load request.
        - If other player agrees, the last game saved in the application will be loaded.
    - Submit an undo request.
        - If there has been at least one move since the beginning of the game.
            - If the other player agrees, the last move will be undo-ed.
            - NB: one move = both players entered one valid input each.
    - Submit a redo request.
        - If there has been at least one undo since the last move,
            - If the other player agrees, the last move will be redo-ed.
    - End the game in draw when threefold repetition takes place.
    - Submit a restart request.
        - If the other player agrees, the game will be restarted (feature added after video and some sample runs)..
- We ask for the 2 players' names at the beginning of the game so we can call them by it.
    - We give then the option to go ahead as player 1 and player 2.
    - We ask them to either:
        - Declare who will be player 1.
        - Let the game choose player 1 randomly.

- Game can be undo-ed to the beginning of the game even if it was loaded (‘even if it was loaded’ → feature added after video and some sample runs).
- Game can be redo-ed to the last move executed.
- All special moves are applied such as:
    - Castling
    - En passant
    - Promotion
- Game ends in draw in the following cases:
    - 2 players agreed to end in draw.
    - 1 player decided to end in draw when threefold repetition took place.
    - Fivefold repetition.
    - Dead position.
    - Stalemate.
- Game ends by a player who won in the following cases:
    - If this player checkmated the other player.
    - If the other player resigned.
- After the game ends, players have the ability to play again if there’s a mutual agreement to that (with no need od restarting application).
    - They are called by their names without asking in the second to last games (feature added after video and some sample runs).

Overview of Design:

- It is preferrable to play the game with a full screen of the console application.
- The game starts by asking for the names of 2 players, and ask them to indicate who is player 1.
- Then, it shows the user manual on the screen followed by the initial board which has white pieces in the bottom and black on top.
- Right after one player enters a valid input, a new board appears on screen having all the moves since the beginning of the game.
- After each move, a message is printed indicating which player's turn it is then.
- The lowercase letters belong to player 1 and the uppercase letters belong to player 2.
- After each move, the board reflects about the x-axis at its centre to give better visualization for each player on his turn (feature added after video and some sample runs).
- The block of '#' is a black block and the block of '-' and '|' is a white block.
- When a white piece is captured, it shows up in the next move under the "White pieces Captured" label on the left of the board.
- When a black piece is captured, it shows up in the next move under the "Black pieces Captured" label on the right of the board.
- In each of the following conditions, an indicator message shows up:
    - Promotion.
    - Check.
    - Checkmate.
    - Stalemate.
    - Threefold right to end the game in draw.
    - Fivefold.
    - A player won.
    - Draw.
    - Saving game.
    - Dead Position.

- A message is printed indicating the non-move options for each player on every turn.
- All invalid inputs are covered and a message indicating that shows up.
    - Ex:
        - User wants to move a piece that is not his.
        - User wants to move his piece in a wrong way.
        - User wants to make a move that will result in his king. being checked.
        - User enters any wrong input by mistake.
- After the game ends, the 2 players have the option to play again.
    - If not program terminates in 5 seconds (feature added after video and some sample runs).

Design/implementation assumptions made:

- Game will not be more than 300 moves.
    - As the longest chess game ever played, with all rules applied was 277 moves.
- Player 2 cannot undo just his move as it will be player 1's turn after that. He can only undo the last move for the both of them when it is his turn and when player 1 agrees and vice versa.
- Player 2 cannot play when the program says it is player 1's turn and vice versa.
- Player can save game, end in draw by threefold repetition and resign without taking permission of the other player.
- If one player closes application without saving, we assume he took the permission of the other player in real life and they decided that they do not want to complete the game. So, their progress is not saved.
- Both players actually know about all the valid moves of chess as our application alerts the user when an invalid input is entered, but it does not provide him with the valid inputs to choose from them.
- When game is saved, the saved game before it is overwritten as there is only one file for saving and loading.
- User shouldn't change anything in the file where we save the board upon request **"SaveMESaveME".**

Further Necessary Details:

Any player who approaches the game should know the following:

- All possible moves for all pieces and when these moves become invalid.
- Check: When current king is threatened by an enemy piece.
- Checkmate: When there is no valid more for current player that makes his king not checked.
- Stalemate: When current player has no valid moves at all including the fact that his king is not checked.
- Dead position: When no sequence of valid moves can get any player to win, so game ends with draw right away.
- Threefold Repetition: When the same board gets repeated 3 times with same conditions including player's turn, current player has ability to declare a draw.
- Fivefold Repetition: When the same board gets repeated 5 times with same conditions including player's turn, game ends with draw as it is not going anywhere and there is no progress being made.

Description of all used data structures:

- C[8][8] → 2D array that contains current board (global)..
- D[600][8][8] → 3D array that is where we save all our C's (global).
- A[8] → contains the main pieces ordered from left to right (pawns excluded) , and we used it to create C[8][8] initial and in promotion (global).
- pieceIndex[2] → stores the row and column respectively of a certain piece whenever we need it to (global).
- whiteTaken[16] → contains all the white pieces captured since the beginning of the game (global).
- saveWhiteTaken[600][16] → 2D array where we save our whiteTaken's (global).
- blackTaken[16] → contains all the white pieces captured since the beginning of the game (global).
- saveBlackTaken[600][16] → 2D array where we save our blackTaken's (global).
- A[] in validate_inp → represents columns and we use them to check if input is valid (local).
- B[] in validate_inp → represents rows and we use them to check if input is valid (local).
- Input[6] → takes the input of the user (global).
- s[20] → takes the inputs per line from the saved file (local).
- Player1[50] → stores name of player 1 (global).
- Player2[50] → stores name of player 2 (global).

Description of the important functions:

- saveBoard → saves the current data (C, whiteTaken, blackTaken) to the databases (D, saveWhiteTaken, saveBlackTaken)..
- deadPosition → checks if deadposition takes place at current C and there are 4 cases:
    - king VS king.
    - king VS king + knight.
    - king VS king + bishop.
    - king + bishop VS king + bishop.
        - where bishops stand on likely colored blocks.
- Fold → counts how many times the current board having current player's turn took place before and returns this value.
- checkPromotion → validates all '5 characters inputs', which should take place only when promotion. So, it checks whether selected block has a pawn, then checks whether movement of pawn is valid, then checks that the fifth character is valid for promotion. It prints messages in all these cases.
- shiftArray → takes 4 parameters representing previous and current indices in C and its responsibility is to make the move and add the captured piece (if there's any) to the corresponding array.
- check → checks if the current king is in check in current C
- possibleCheck → takes 4 parameters representing previous and current indices in C and pass them to shiftArray, then it returns check, and puts everything back to what it was before the function was called.
- Check_movement → takes 4 parameters representing previous and current indices in C, figures out the piece in the previous block, and returns 1 if its move is valid. Also uses check to make sure king is not checked in castling.
- validate_inp →
    - checks if input is written correctly, then it converts the input into previous and current indices in C.
    - uses checkPromotion to validate 5 charactors input.

    - Checks if previous block contains a piece that belongs to current player.
    - Checks if current block contains an enemy piece or nothing.
    - Uses possibleCheck to check if this move will result current king being in check.
    - Uses check_movement to check if this is a valid move.
    - When it gets an invalid input,
        - Prints an alert.
        - Calls itself till it gets a valid one.
- Stalemate → loops over all the current player's pieces and uses possibleCheck and check_movement to check if any piece has any valid move. If not, returns 1.
- move →
    - print previous board before each move as well as white and black pieces captured.
    - Saves by saveBoard.
    - prints who's turn it is to make the move.
    - Calls stalemate, check, deadPosition, fold (to detect threefold and fivefold).
    - If stalemate and check return 1, then it's a checkmate.
    - If game should end, it ends it, after displaying a message accordingly, by breaking out of game loop.

Flowchart of Main Algorithm:

There is a file in the project zipped file called
Youssef_Ahmed_19016903_Abdelmoniem_Hany_19017359_flowchart.pdf

You can also access our flowchart from **CHESS FLOWCHART.drawio** in the zipped file.

Psuedocode of Main Algorithm:

Global Variables and Arrays:

char A[8] = {'R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'};

char c[8][8]; (chessboard board array)

char D[600][8][8]; (database for each move players make which saves the chess boards)

int player = 2;

int sm, dp, chck;

int prev_c = -1, curr_c = -1, prev_r = -1, curr_r = -1;

int endGame = 0, draw = 0, resign = 0, checkmate = 0,undo = 0;

int moveIndex = 0; (the number of moves made and chessboards printed with 0 being the initial board)

int brk = 0;

int invalid = 0;

int promotion = 0;

int pieceIndex[2];

char input[7] = {' '}; (Character Array that holds the user's input)

int threeFold = 0;

int fiveFold = 0;

char whiteTaken[16]={0}; (contains the white chess pieces that got eaten)

char saveWhiteTaken[600][16]; (database for the white chess pieces eaten each move)

char blackTaken[16]={0}; (contains the white chess pieces that got eaten)

char saveBlackTaken[600][16]; (database for the black chess pieces eaten each move)

```
main Function :
        INITIALIZE char pointer p, array s[20], i, j
        FOR ( every row in the chess board )
                FOR ( every column in the chess board )
                        PUT initial Chess pieces in c[8][8]
                END FOR
        END FOR
        PRINT Welcome message and some instructions
        CALL for move
        WHILE ( endGame = 0 )
                GET input from user
                CREATE a text file
                IF ( a user inputed 'S' ) THEN
                        SAVE c[8][8], whiteTaken[], blackTaken[] in text file
                        PRINT ("Game is saved")
                        PRINT the players' turns
                        CONTINUE
                END IF
                IF ( a user inputed 'L' ) THEN
                        PRINT a question that asks the other player if he would
like to load a game as well
                        IF ( the other player agreed and entered 'L' ) THEN
                                LOAD the game from the text file
                                CALL for move
```

```
                    CONTINUE
                ELSE
                    PRINT the the other player didn't accept his request for loading a saved game
                    PRINT the players' turns
                    CONTINUE
                END IF
            END IF
            IF ( a user inputed 'D' and threeFold = 1 ) THEN
                PRINT ("GAME ENDED IN DRAW BY THREEFOLD REPETITION")
                SET endGame to 1
                BREAK
            ELSE IF ( a user inputed 'D' ) THEN
                PRINT a question whether the other player agrees to end the game with a draw
                GET input from the other user
                IF ( the other user inputed 'D' ) THEN
                    SET endGame = 1
                    Print("GAME ENDED WITH A DRAW")
                    BREAK
                ELSE
                    PRINT a message that says that the other player didn't accept his request for a draw
                    PRINT the players' turns
                    CONTINUE
                END IF
```

```
            END IF
            IF ( a user inputed 'R' ) THEN
                SET endGame = 1
                PRINT a message that congratulats the other user for
winning because the user that inputed 'R' resigned
                BREAK
            END IF
            IF ( a user inputed 'U' ) THEN
                IF ( 2 moves were played ) THEN
                    PRINT a question whether the other player agrees
to undo the last pair of moves.
                    GET input from the other player
                    IF ( the other user inputed 'U' ) THEN
                        FOR ( every row in the chess board )
                            FOR ( every column in the chess board
)
                                c[i][j] = D[moveIndex-3][i][j]
(Overwrite c with the board saved in D two moves before)
                            END FOR
                        END FOR
                        FOR ( i=0, i<16, INCREMENT i )
                            whiteTaken[i] =
saveWhiteTaken[moveIndex-3][i]
                        blackTaken[i] = saveBlackTaken[moveIndex-3][i]
                    END FOR
                    SET moveIndex to moveIndex-3
                    SET undo = 1
```

CALL for move

ELSE

PRINT a message that says that the other user didn't accept the request for undo

PRINT the players' turns

ELSE

PRINT a message that says that you can't undo because you haven't made any moves left

PRINT the players' turns

END IF

CONTINUE

IF (a user inputed 'RE') THEN

IF (undo = 1) THEN

PRINT a question for the other user whether he agrees to redo the last pair of moves

GET input from the other

IF (the other user inputed 'RE') THEN

FOR (every row in the chess board)

FOR (every column in the chess board)

c[i][j] = D[moveIndex+1][i][j] (Overwrite c with the board saved in D two moves after)

END FOR

END FOR

FOR (i=0, i<16, INCREMENT i)

whiteTaken[i] = saveWhiteTaken[moveIndex+1][i]

```
                    blackTaken[i] = saveBlackTaken[moveIndex+1][i]
                END FOR
                SET moveIndex = moveIndex + 1
                CALL for move
                    ELSE
                PRINT a message that says that the other user didn't
accept the request for redo
                PRINT the players' turns
                ELSE
                PRINT a message that says that you can't redo because
you haven't undone any moves
                PRINT the players' turns
            END IF
            CONTINUE
              END IF
              ELSE
                CALL for validate_inp
                IF ( brk = 1 ) THEN
                    PRINT ("Please enter your input again")
                    CONTINUE
                END IF
                IF ( invalid = 0) THEN
                    SET undo = 0
                END IF
              END IF
              CALL for shiftArray
```

```
        Call for move
    END WHILE
    RETURN 0
END FUNCTION
```

Important Function:

SaveBoard FUNCTION :

```
    INITIALIZE i and j as rows(i) and columns(j)
    FOR every row
        FOR every column
            SAVE c[row][column] to D[moveIndex][i][j]


        END FOR
    END FOR
    FOR i = 0, i < 16, INCREMENT i ( i is the possible pieces captured)
        SAVE blackTaken[i] to saveBlackTaken[moveIndex][i]
        SAVE whiteTaken[i] to saveWhiteTaken[moveIndex][i]
    END FOR
    Increment moveIndex
    END FUNCTION
```

deadPosition FUNCTION :

INITIALIZE colourb, colourB, i, j, nb=0, NB=0, b=0, B=0, specialCase = 0

FOR every row

FOR every column

IF c[row][column] contains anything other than k, K, b, B, n, N, or space THEN

RETURN 0

END IF

ELSE IF c[row][column] contains b, or n THEN

INCREMENT nb

IF c[row][column] contains b Then

COMPUTE colourb as row+column

INCREMENT b

END IF

ELSE IF c[row][column] contains B, or N THEN

INCREMENT NB

IF c[row][column] contains B Then

COMPUTE colourB as row+column

INCREMENT B

END IF

END IF

END FOR

END FOR

IF (b=1 and B=1 and (colourB mod 2 = colourb mod 2)) THEN

SET specialCase as 1

END IF

IF ((nb=1 and NB=0) or (nb=0 and N==1) or specialCase = 1) THEN

RETURN 1

ELSE

RETURN 0

END FUNCTION

move FUNCTION :

INITIALIZE i, j, k, count = 0

PRINT ("White Pieces Captured", "Black Pieces Captured", and Column Names (A, B, C, D, E, F, G, H)

FOR LOOPS to print the row numbers and the white pieces taken and black pieces taken.

For Loops to print Black Squares (#) and White Square (-) and inside it c[row][column] which is the chess piece in it's corresponding position

CALL saveBoard()

CASE player's turn IS

1 : player's turn is 2

2 : player's turn is 1

END CASE

INITIALIZE sm = CALL for stalemate(), dp = CALL for deadPosition(), chck = Call for check()

IF (CALL for fold() = 5) THEN

SET fiveFold = 1

SET threeFold =1

```
        ELSE IF (CALL for fold() = 3) THEN
            SET threeFold = 1
        ELSE
            SET fiveFold = threeFold = 0
        END IF
        IF (promotion = 1)
            PRINT (" Player (1-2) promoted his pawn to
(c[current_row][current_column])")
        IF (sm = 0 and dp = 0 and fiveFold = 0) Then
            SHIFT TURN
            IF (chck = 1)
                PRINT ("YOUR KING IS IN CHECK")
            ELSE IF (threeFold = 1)
                PRINT("You have the right to end the game in draw by
threefold repetition")
            END IF
        ELSE IF (sm = 1 and dp = 0 and fiveFold = 0 and chck = 0) THEN
            PRINT("THE GAME ENDED IN A DRAW BY STALEMATE")
            SET endGame = 1
        ELSE IF (dp = 1) THEN
            PRINT( "THE GAME ENDED IN A DRAW BY DEAD
POSITION")
            SET endGame = 1
        ELSE IF (fiveFold = 1) THEN
            PRINT("THE GAME ENDED IN A DRAW BY FIVEFOLD
REPETITION")
            SET endGame = 1
```

```
        ELSE IF (chck = 1 and sm =1 ) THEN
                PRINT("CHECKMATE !!! CONGRATULATIONS PLAYER %d.
YOU WON.")
                SET endGame = 1
        ENDIF
END FUNCTION



fold FUNCTION :
        INITIALIZE i, j, k, count, foldCount = 0
        IF (player =1) THEN
                FOR (k=moveIndex-1, k>-1, decrement k by 2)
                        SET count = 0
                        FOR each row
                                For each column
                                        IF (c[i][j] == D[k][i][j]) THEN
                                                INCREMENT count
                                        END IF
                                END FOR
                        END FOR
                        IF count == 64 THEN INCREMENT foldCount, ELSE do
nothing
                END FOR
        ELSE
                FOR (k=moveIndex-1, k>-1, decrement k by 2)
                        SET count = 0
```

```
            FOR each row
                    For each column
                        IF (c[i][j] == D[k][i][j]) THEN
                            INCREMENT count
                        END IF
                    END FOR
                END FOR
                IF count == 64 THEN INCREMENT foldCount, ELSE do
nothing
            END FOR
        END IF
        RETURN fold Count
END FUNCTION


possibleCheck FUNCTION :
        INITIALIZE a, i, j
        CALL for shiftArray(p_r, p_c, c_r, c_c)
        IF (Call for check() = 1) THEN
            SET a = 1
        ELSE
            SET a = 0
        END IF
        FOR each row
            FOR each column
                SAVE c[i][j] to D[moveIndex-1][i][j]
```

```
            END FOR
        END FOR
        FOR (i=0, i<16, INCREMENT i)
            SAVE whiteTaken[i] to saveWhiteTaken[moveIndex-1][i]
            SAVE blackTaken[i] to saveBlackTaken[moveIndex-1][i]
        END FOR
        IF (a = 1) THEN
            RETURN 1
        ELSE
            RETURN 0
        END IF
END FUNCTION


validate_inp FUNCTION :
        INITIALIZE a = 0, b = 0, j2
        INITIALIZE x as length of the input string
        INITIALIZE A[]={'A','B','C','D','E','F','G','H'}
        INITIALIZE B[]={'1','2','3','4','5','6','7','8'}
        INITIALIZE Char Pointer *p
        SET p = input
        IF the input is just ( U, L, S, RE, R, D) THEN
            SET brk = 1
        ELSE
            SET brk = 0
```

```
FOR (j2=0, j2<8, INCREMENT j2)
    IF (first character from input string is in A[j2]) THEN
        INCREMENT a
    END IF
END FOR
FOR (j2=0, j2<8, INCREMENT j2)
    IF (second character from input string is in B[j2]) THEN
        INCREMENT a
    END IF
END FOR
FOR (j2=0, j2<8, INCREMENT j2)
    IF (third character from input string is in A[j2]) THEN
        INCREMENT a
    END IF
END FOR
FOR (j2=0, j2<8, INCREMENT j2)
    IF (fourth character from input string is in B[j2]) THEN
        INCREMENT a
    END IF
END FOR
IF ( (first character != third character) or (second character != fourth character) ) THEN
    SET b = 1
END IF
IF ((x!=4 and x!=5) or a!=4 or b = 0)
    PRINT("Please Enter valid input")
```

```
        SET invalid = 1
        READ new input from user and store in input[]
        RECURSIVE CALL FOR validate_inp()
ELSE
        CASE(first character) IS
                'A' : previous_column = 0
                'B' : previous_column = 1
                'C' : previous_column = 2
                'D' : previous_column = 3
                'E' : previous_column = 4
                'F' : previous_column = 5
                'G' : previous_column = 6
                'H' : previous_column = 7
        END CASE
        CASE(second character) IS
                '1' : previous_row = 7
                '2' : previous_row = 6
                '3' : previous_row = 5
                '4' : previous_row = 4
                '5' : previous_row = 3
                '6' : previous_row = 2
                '7' : previous_row = 1
                '8' : previous_row = 0
        END CASE
        CASE(third character) IS
```

```
                'A' : current_column = 0
                'B' : current_column = 1
                'C' : current_column = 2
                'D' : current_column = 3
                'E' : current_column = 4
                'F' : current_column = 5
                'G' : current_column = 6
                'H' : current_column = 7
            END CASE
            CASE (fourth character) IS
                '1' : current_row = 7
                '2' : current_row = 6
                '3' : current_row = 5
                '4' : current_row = 4
                '5' : current_row = 3
                '6' : current_row = 2
                '7' : current_row = 1
                '8' : current_row = 0
            END CASE
            SET promotion = Call for checkPromotion()
            IF (Block that sends out =  ' ') THEN
                PRINT("Duh. You haven't chosen any piece. Please
enter another input.")
                SET invalid = 1
                READ new input from user and store in input[]
                RECURSIVE CALL FOR validate_inp()
```

END IF

IF (it's player 1's turn) THEN

IF (the block that sends out contains a capital letter) THEN

PRINT("(c[previous_row][previous_column]) is not your piece. Please enter another input.")

SET invalid = 1

READ new input from user and store in input[]

RECURSIVE CALL FOR validate_inp()

ELSE IF (the block that recieves contains a small letter) THEN

PRINT("Please Enter valid input.")

SET invalid = 1

READ new input from user and store in input[]

RECURSIVE CALL FOR validate_inp()

END IF

END IF

IF (it's player's 2's turn) THEN

IF (the block that sends out contains a small letter) THEN

PRINT("(c[previous_row][previous_column]) is not your piece. Please enter another input.")

SET invalid = 1

READ new input from user and store in input[]

RECURSIVE CALL FOR validate_inp()

ELSE IF (the block that recieves contains a capital letter) THEN

```
                                PRINT("Please Enter valid input.")
                                SET invalid = 1
                                READ new input from user and store in input[]
                                RECURSIVE CALL FOR validate_inp()
                            END IF
                        END IF
                        IF ( CALL for possibleCheck(p_r, p_c, c_r, c_c) = 1 )
THEN
                            PRINT("(c[previous_row][previous_column]) cannot
move here. Enter another input.")
                            SET invalid = 1
                            READ new input from user and store in input[]
                            RECURSIVE CALL FOR validate_inp()
                        ELSE IF ( CALL for check_movement(p_r, p_c, c_r, c_c)
= 0 ) THEN
                            PRINT("(c[previous_row][previous_column]) cannot
move here. Enter another input.")
                            SET invalid = 1
                            READ new input from user and store in input[]
                            RECURSIVE CALL FOR validate_inp()
                        END IF
                END IF
        END IF
END FUNCTION
```

check Promotion FUNCTION :

```
INITIALIZE i = 0, x = length of input string
IF ( x = 5) THEN
    IF( block that sends out contains 'p') THEN
        IF ( (it moves in the same column and it moves forward 1 block and the block that recieves is empty) or (it moves forward horizontaly (left or right) by 1 block and the block that recieves is not empty)) THEN
            IF ( if it's in row 7 and going to the last row ) THEN
                FOR ( i=0, i<4; INCREMENT I )
                    IF ( the fifth character is a small character in the character Array A[] ) THEN
                        RETURN 1
                    END IF
                END FOR
                PRINT ("Please Enter a valid input for promotion")
                SET invalid = 1
                READ new input from user and store in input[]
                RECURSIVE CALL FOR validate_inp()
            ELSE
                PRINT ("Please Enter valid input")
                SET invalid = 1
                READ new input from user and store in input[]
                RECURSIVE CALL FOR validate_inp()
            END IF
        ELSE
            PRINT ("(c[prev_r][prev_c]) cannot move here. Enter another input.")
```

```
                    SET invalid = 1
                    READ new input from user and store in input[]
                    RECURSIVE CALL FOR validate_inp()
                END IF
            ELSE IF ( block that sends out contains 'P') THEN
                IF ( (it moves in the same column and it moves backward
1 block and the block that recieves is empty) or (it moves backward
horizontaly (left or right) by 1 block and the block that recieves is not
empty)) THEN
                    IF ( if it's in row 2 and going to the first row ) THEN
                        FOR ( i=0, i<4; INCREMENT I )
                            IF ( the fifth character is a Capital
character in the character Array A[] ) THEN
                                RETURN 1
                            END IF
                        END FOR
                        PRINT ("Please Enter a valid input for
promotion")
                        SET invalid = 1
                        READ new input from user and store in input[]
                        RECURSIVE CALL FOR validate_inp()
                    ELSE
                        PRINT ("Please Enter valid input")
                        SET invalid = 1
                        READ new input from user and store in input[]
                        RECURSIVE CALL FOR validate_inp()
                    END IF
```

```
                ELSE
                        PRINT ("(c[prev_r][prev_c]) cannot move here.
Enter another input.")
                        SET invalid = 1
                        READ new input from user and store in input[]
                        RECURSIVE CALL FOR validate_inp()
                END IF
        ELSE
                PRINT ("Please Enter valid input")
                SET invalid = 1
                READ new input from user and store in input[]
                RECURSIVE CALL FOR validate_inp()
        END IF
    END IF
    RETURN 0
END FUNCTION


check_movement FUNCTION :
    INITIALIZE i = 0, j = 0 ,k = 0
    IF ( the block that sends out contains 'n' or contains 'N' ) THEN
        IF ( it moves right or left by 1 block ) THEN
            IF ( it moves forward or backward by 2 blocks ) THEN
                RETURN 1
            ELSE
                RETURN 0
```

END IF

ELSE IF (it moves right or left by 2 blocks) THEN

IF (it moves forward or backward by 1 block) THEN

RETURN 1

ELSE

RETURN 0

END IF

ELSE

RETURN 0

END IF

ELSE IF (the block that sends out contains 'k' or contains 'K') THEN

IF (it moves right or left by 1 block) THEN

IF (it moves upward or downward by 1 block or stays in the same row) THEN

RETURN 1

END IF

ELSE IF (it stays in the same column) THEN

IF (it moves foward or backward by 1 block) THEN

RETURN 1

END IF

ELSE IF (it moves right by 2 blocks and in the same row and CALL for check(c) = 0) THEN

IF (c[curr_r][5] contains space and c[curr_r][6] contains space)

FOR (i=0, i<moveIndex, INCREMENT i)

IF (D[i][prev_r][4] doesn't contain space) THEN INCREMENT j ELSE do nothing (king initial position whether black or white)

IF (D[i][prev_r][7] doesn't contain space) THEN INCREMENT j ELSE do nothing (right rook initial position whether black or white)

END FOR

IF (j == moveIndex and k == moveIndex) THEN (king and rook haven't changed position throughout the whole game and stayed in their initial positons for castling to happen)

PUT c[prev_r][7] IN c[prev_r][curr_c-1] (rook moves to the left of the king)

PUT ' ' IN c[prev_r][7]

RETURN 1

END IF

ELSE

RETURN 0

END IF

ELSE IF (it moves left by 2 blocks and in the same row and CALL for check(c) = 0) THEN

IF (c[curr_r][1] contains space and c[curr_r][2] contains space and c[curr_r][3] contains space) THEN

FOR (i=0, i<moveIndex, INCREMENT i)

IF (D[i][prev_r][4] doesn't contain space) THEN INCREMENT j ELSE do nothing (king initial position whether black or white)

IF (D[i][prev_r][0] doesn't contain space) THEN INCREMENT j ELSE do nothing (left rook initial position whether black or white)

END FOR

```
                    IF ( j == moveIndex and k == moveIndex) THEN
(king and rook haven't changed position throughout the whole game and
stayed in their initial positons for castling to happen)
                        PUT c[prev_r][0] IN c[prev_r][curr_c+1]  (rook
moves to the right of the king)
                        PUT ' ' IN c[prev_r][7]
                        RETURN 1
                    END IF
                ELSE
                    RETURN 0
                END IF
            END IF
        ELSE IF ( the block that sends out contains 'r' or 'R' or 'q' or 'Q' )
THEN
            IF ( it moves in the same column) THEN
                IF( it moves downward) THEN
                    FOR (i=prev_r+1, i<curr_r, INCREMENT i)
                        IF (c[i][curr_c] doesn't contain space) THEN
                            RETURN 0
                        END IF
                    END FOR
                    RETURN 1
                ELSE (it moves upward)
                    FOR (i=prev_r-1, i>curr_r, DECREMENT i)
                        IF (c[i][curr_c] doesn't contain space) THEN
                            RETURN 0
                        END IF
```

```
                END FOR
                RETURN 1
            END IF
        ELSE IF ( it moves in the same row) THEN
            IF ( it moves right) THEN
                FOR (i=prev_c+1, i<curr_c, INCREMENT I)
                    IF (c[curr_r][i] doesn't contain space) THEN
                        RETURN 0
                    END IF
                END FOR
                RETURN 1
            ELSE (it moves left)
                FOR (i=prev_c+1, i<curr_c, INCREMENT I)
                    IF (c[curr_r][i] doesn't contain space) THEN
                        RETURN 0
                    END IF
                END FOR
                RETURN 1
            END IF
        END IF
END IF
IF ( the block that sends out contains 'b' or 'B' or 'q' or 'Q') THEN
        IF ( it moves diagonally) THEN
            IF ( it moves right and downward) THEN
                SET i = prev_r + 1
```

```
                SET j = prev_c + 1
                WHILE ( (i doesn't equal curr_r) and (j doesn't equal
curr_c) )
                    IF ( c[i][j] doesn't contain space) THEN
                        RETURN 0
                    END IF
                    INCREMENT i and j
                END WHILE
                RETURN 1
            ELSE IF ( it moves left and downard) THEN
                SET i = prev_r + 1
                SET j = prev_c - 1
                WHILE ( (i doesn't equal curr_r) and (j doesn't equal
curr_c) )
                    IF ( c[i][j] doesn't contain space) THEN
                        RETURN 0
                    END IF
                    INCREMENT i
                    DECREMENT j
                END WHILE
                RETURN 1
            ELSE IF ( it moves right and upward) THEN
                SET i = prev_r - 1
                SET j = prev_c + 1
                WHILE ( (i doesn't equal curr_r) and (j doesn't equal
curr_c) )
```

```
                    IF ( c[i][j] doesn't contain space) THEN
                        RETURN 0
                    END IF
                    DECREMENT i
                    INCREMENT j
                END WHILE
                RETURN 1
            ELSE ( it moves left and upward) THEN
                SET i = prev_r - 1
                SET j = prev_c - 1
                WHILE ( (i doesn't equal curr_r) and (j doesn't equal
curr_c) )
                    IF ( c[i][j] doesn't contain space) THEN
                        RETURN 0
                    END IF
                    DECREMENT i
                    DECREMENT j
                END WHILE
                RETURN 1
            END IF
        END IF
    ELSE IF ( the block that send out contains 'p' ) THEN
        IF ( promotion = 1) THEN
            SET c[prev_r][prev_c] = input [4]
            RETURN 1
```

ELSE IF (it moves forward once in the same column and where it's going is space) THEN

RETURN 1

ELSE IF (it moves forward two blocks in the same column and is in the initial position (prev_r = 6) and the block where it goes and the block before are spaces) THEN

RETURN 1

ELSE IF (it moves forward one block diagonally (left or right)) THEN

IF (the block that receives is not space) THEN

RETURN 1

ELSE IF ('P' is in c[1][curr_c] for the last two moves and 'p' is in prev_r = 3 and 'p' is going to a block next to 'P' (c[prev_r][curr_c] = 'P')) THEN (EN Passant move for white pawn)

Initialize Char Pointer *p

for (p=blackTaken, p<blackTaken+16, INCREMENT p)

IF (value in pointer p equals 0) THEN

SET value of p = c[prev_r][curr_c] (eaten black Pawn)

BREAK

END IF

END FOR

PUT ' ' IN c[prev_r][curr_c]

RETURN 1

END IF

END IF

ELSE IF (the block that send out contains 'P') THEN

IF (promotion = 1) THEN

SET c[prev_r][prev_c] = input [4]

RETURN 1

ELSE IF (it moves backward once in the same column and where it's going is space) THEN

RETURN 1

ELSE IF (it moves backward two blocks in the same column and is in the initial position (prev_r = 1) and the block where it goes and the block before are spaces) THEN

RETURN 1

ELSE IF (it moves backwards one block diagonally (left or right)) THEN

IF (the block that receives is not space) THEN

RETURN 1

ELSE IF ('p' is in c[6][curr_c] for the last two moves and 'P' is in prev_r = 4 and 'P' is going to a block next to 'p' (c[prev_r][curr_c] = 'p')) THEN (EN Passant move for black pawn)

Initialize Char Pointer *p

for (p=whiteTaken, p<whiteTaken+16, INCREMENT p)

IF (value in pointer p equals 0) THEN

SET value of p = c[prev_r][curr_c] (eaten white Pawn)

BREAK

END IF

END FOR

PUT ' ' IN c[prev_r][curr_c]

RETURN 1

```
                END IF
            END IF
        END IF
        RETURN 0
END FUNCTION


shiftArray FUNCTION :
        INITIALIZE temp, i, j
        IF ( the block that recieves contains space ) THEN
            PUT c[prev_r][prev_c] IN c[curr_r][curr_c]
            PUT space IN  c[prev_r][prev_c]
        ELSE
            SET temp = c[curr_r][curr_c]
            PUT c[prev_r][prev_c] IN c[curr_r][curr_c]
            PUT space IN  c[prev_r][prev_c]
        END IF
        INITIALIZE Char Pointer p
        SET p = blackTaken[]
        INITIALIZE Char Pointer q
        SET q = whiteTaken[]
        IF ( it's player 1's turn ) THEN
            FOR (p=blackTaken, p<blackTaken+16, INCREMENT p)
                IF ( *p = 0 ) ( loops over array blackTaken till it finds an
empty spot)
                    SET value of pointer p to temp
```

```
                    BREAK
                END IF
            END FOR
        END IF
        IF ( it's player 2's turn ) THEN
            FOR (q=whiteTaken, q<whiteTaken+16, INCREMENT q)
                IF ( *q = 0 ) ( loops over array whiteTaken till it finds an
empty spot)
                    SET value of pointer q to temp
                    BREAK
                END IF
            END FOR
        END IF
END FUNCTION
```

check FUNCTION :

CALL for getPieceIndex('k' (if it's player 1's turn), 'K' (if it's player 2's turn)) (Get's king's row and column in the chessboard each move)

INITIALIZE i, j and SET kingRow = pieceIndex[0], KingCol = pieceIndex[1]

INITIALIZE shifter, piece and SET piece 2 = 'q'

IF (it's player 1's turn) THEN

SET shifter to 'k'

ELSE

SET shifter t 'K'

```
        END IF
        CALL for stringInverter(shifter, address of piece2)
        IF (it's player 1's turn) THEN
            IF ( there is a black pawn one block diagonally away from white
king) THEN
                RETURN 1
            END IF
        ELSE
            IF ( there is a white pawn one block diagonally away from a
black king) THEN
                RETURN 1
            END IF
        END IF
        SET piece to 'n'
        Call for stringInverter(shifter, address of piece)
        FOR ( i=-2, i<3, INCREMENT i )  (where i represents the number of
rows away from the king's occupied row)
            IF ( a knight is found one row front or back of  a king ) THEN
                IF ( a knight is found two columns right or left of a king )
THEN
                    RETURN 1
                END IF
            ELSE IF ( a knight is found two rows front or back of a king )
THEN
                IF ( a knight is found pne column right or left of a king )
THEN
                    RETURN 1
```

```
            END IF
        END IF
END FOR
SET piece to 'b'
Call for stringInverter(shifter, address of piece)
SET j = king's column + 1
FOR ( i = king's row + 1, i<8 and j<8, INCREMENT i)
        IF ( c[i][j] contains bishop or queen ) THEN
            RETURN 1
        ELSE IF ( c[i][j] contains space ) THEN
            INCREMENT j
            CONTINUE
        ELSE
            BREAK
        END IF
END FOR
SET j = king's column - 1
FOR ( i = king's row + 1, i<8 and j>=0, INCREMENT i)
        IF ( c[i][j] contains bishop or queen ) THEN
            RETURN 1
        ELSE IF ( c[i][j] contains space ) THEN
            DECREMENT j
            CONTINUE
        ELSE
            BREAK
```

```
        END IF
END FOR
SET j = king's column + 1
FOR ( i = king's row - 1, i>=0 and j<8, DECREMENT i)
        IF ( c[i][j] contains bishop or queen ) THEN
                RETURN 1
        ELSE IF ( c[i][j] contains space ) THEN
                INCREMENT j
                CONTINUE
        ELSE
                BREAK
        END IF
END FOR
SET j = king's column - 1
FOR ( i = king's row - 1, i>=0 and j>=0, DECREMENT i)
        IF ( c[i][j] contains bishop or queen ) THEN
                RETURN 1
        ELSE IF ( c[i][j] contains space ) THEN
                DECREMENT j
                CONTINUE
        ELSE
                BREAK
        END IF
END FOR
SET piece to 'r'
```

```
Call for stringInverter(shifter, address of piece)
FOR ( i = king's row + 1, i<8, INCREMENT i)
        IF ( c[i][king's column] contains rook or queen ) THEN
                RETURN 1
        ELSE IF ( c[i][king's column] contains space ) THEN
                CONTINUE
        ELSE
                BREAK
        END IF
END FOR
FOR ( i = king's row - 1, i>=0, DECREMENT i)
        IF ( c[i][king's column] contains rook or queen ) THEN
                RETURN 1
        ELSE IF ( c[i][king's column] contains space ) THEN
                CONTINUE
        ELSE
                BREAK
        END IF
END FOR
FOR ( i = king's column + 1, i<8, INCREMENT i)
        IF ( c[king's row][i] contains rook or queen ) THEN
                RETURN 1
        ELSE IF ( c[king's row][i] contains space ) THEN
                CONTINUE
        ELSE
```

```
                BREAK
            END IF
        END FOR
            FOR ( i = king's column- 1, i>=0, DECREMENT i)
            IF ( c[king's row][i] contains rook or queen ) THEN
                RETURN 1
            ELSE IF ( c[king's row][i] contains space ) THEN
                CONTINUE
            ELSE
                BREAK
            END IF
        END FOR
        SET piece to 'k'
        Call for stringInverter(shifter, address of piece)
        FOR ( i = king's row - 1, i < king's row + 2, INCREMENT i)
            FOR ( j = king's column - 1, j < king's column + 2, INCREMENT
j)
                IF ( c[i][j] contains other king)
                    RETURN 1
                END IF
            END FOR
        END FOR
        RETURN 0
END FUNCTION
```

checkVMPawn FUNCTION :

IF (c[i][j] contains 'p') THEN

IF ((block above contains space and Call for possibleCheck = 0) or (one block up diagonally is a capital letter and Call for possibleCheck = 0)) THEN

RETURN 0

ELSE IF (En passant move is made and Call for possibleCheck = 0) THEN

RETURN 0

END IF

ELSE IF (c[i][j] contains 'P') THEN

IF ((block below contains space and Call for possibleCheck = 0) or (one block down diagonally is a small letter and Call for possibleCheck = 0)) THEN

RETURN 0

ELSE IF (En passant move is made and Call for possibleCheck = 0) THEN

RETURN 0

END IF

END IF

RETURN 1

END FUNCTION

checkVMKnight FUNCTION :

INITIALIZE k

IF (c[i][j] contains 'n' or 'N') THEN

FOR (all row changes a knight can make (from -2 to 2 rows))

IF (it moved two rows up or down)

If (it's player 1's turn) THEN

IF (it moves 1 block right or left and the block that recieves doesn't contain a small letter and CALL for possible Check = 0) THEN

RETURN 0

END IF

ELSE

IF (it moves 1 block right or left and the block that recieves doesn't contain a capital letter and CALL for possible Check = 0) THEN

RETURN 0

END IF

END IF

ELSE IF (it moved one row up or down)

If (it's player 1's turn) THEN

IF (it moves 2 blocks right or left and the block that recieves doesn't contain a small letter and CALL for possible Check = 0) THEN

RETURN 0

END IF

ELSE

IF (it moves 2 blocks right or left and the block that recieves doesn't contain a capital letter and CALL for possible Check = 0) THEN

RETURN 0

END IF

```
                    END IF
                END IF
            END FOR
        END IF
        RETURN 1
END FUNCTION
```

checkVMRookQ1 FUNCTION :

```
        INTIALIZE k
        IF ( c[i][j] contains 'r' or 'R' or 'q' or 'Q' ) THEN
            FOR ( every row in the chess board )
                IF ( user doesn't input the same block as a move and Call for possibleCheck = 0 for any block in the same row ) THEN
                    IF ( call for check_movement = 1 for any block in the same row ) THEN
                        RETURN 0
                    END IF
                END IF
            END FOR
            FOR ( every column in the chess board )
                IF ( user doesn't input the same block as a move and Call for possibleCheck = 0 for any block in the same column ) THEN
                    IF ( call for check_movement = 1 for any block in the same column ) THEN
                        RETURN 0
                    END IF
```

```
                END IF
            END FOR
        END IF
        RETURN 1
END FUNCTION
```

checkVMBishopQ2 FUNCTION :

```
        INITIALIZE k, u
        IF ( c[i][j] contains 'b' or 'B' or 'q' or 'Q' ) THEN
            FOR ( every row in the chess board )
                FOR ( every column in the chess board )
                    IF ( user doesn't input the same block as a move
and Call for possibleCheck = 0 for any block in chess board ) THEN
                        IF ( call for check_movement = 1 for any block
in the chessboard ) THEN
                            RETURN 0
                        END IF
                    END IF
                END FOR
            END FOR
        END IF
        RETURN 1
END FUNCTION
```

```
check VMKing FUNCTION :
        INITIALIZE k, u
        IF ( c[i][j] contains 'k' or 'K' ) THEN
                FOR ( the rows surronding the king's block (from king' row -1 to king's row +1) )
                        FOR ( the columns surronding the king's block (from king' column -1 to king's column +1) )
                                IF ( it's player 1's turn ) THEN
                                        IF ( the block that recieves doesn't contain a small letter ) THEN
                                                RETURN 0
                                        END IF
                                ELSE
                                        IF ( the block that recieves doesn't contain a captial letter ) THEN
                                                RETURN 0
                                        END IF
                                END IF
                        END FOR
                END FOR
        END IF
        RETURN 1
END FUNCTION


stalemate FUNCTION :
        INITIALIZE i, j
```

```
    FOR ( every row in the chess board )
        FOR ( every column in the chess board )
            IF ( when it's player 1's turn and c[i][j] contains space or a capital letter ) THEN
                CONTINUE
            ELSE IF ( when it's player 2's turn and c[i][j] contains space or a small letter ) THEN
                CONTINUE
            ELSE
                IF ( CALL for checkVMPawn = 0 ) THEN
                    RETURN 0
                ELSE IF ( CALL for checkVMKnight = 0 ) THEN
                    RETURN 0
                ELSE IF ( CALL for checkVMKING = 0 ) THEN
                    RETURN 0
                ELSE IF ( CALL for checkVMRookQ1 = 0 ) THEN
                    RETURN 0
                ELSE IF ( CALL for checkVMBishopQ2 = 0 ) THEN
                    RETURN 0
                END IF
            END IF
        END FOR
    END FOR
    RETURN 1
END FUNCTION
```

Complementary unimportant functions :

strLength Function gets the length of a character array

getPieceIndex Function gets the row and column of the character that gets entered as a paramter to the function anywhere this piece is in the chessboard

stringInverter Function converts small letters to capital according to a shifter

lsp Function returns 1 when the character entered as a paramter is a space or a small letter, otherwise it returns 0

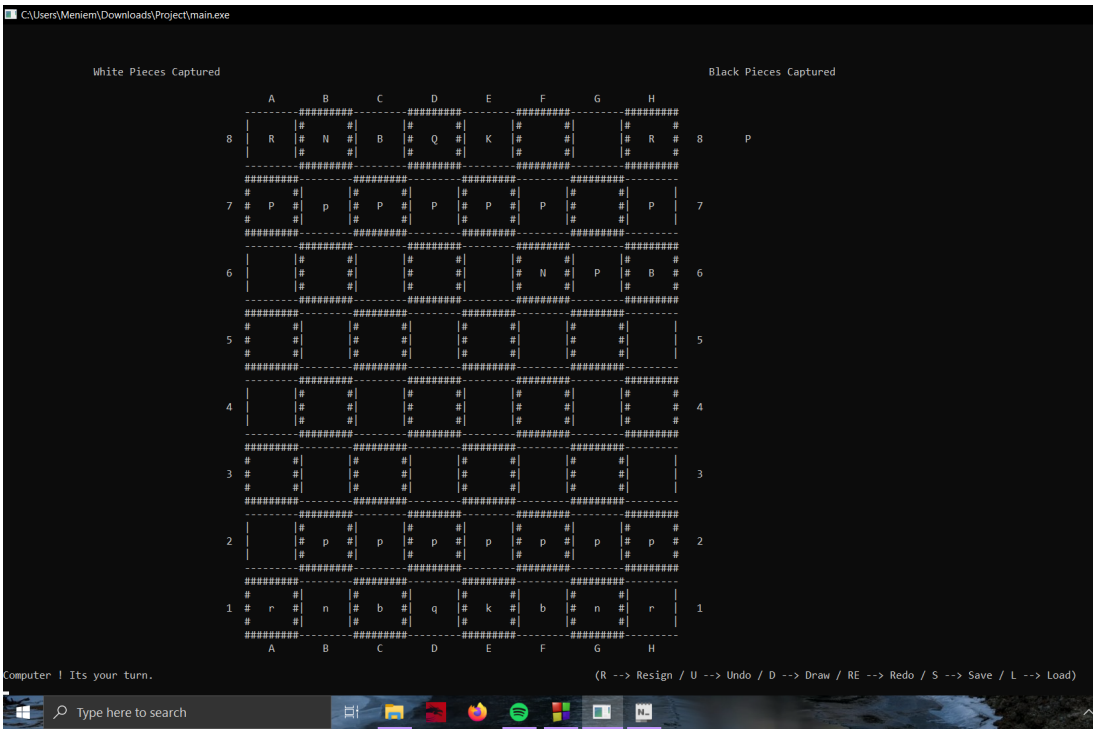
usp Function returns 1 when the character entered as a paramter is a space or a capital letter, otherwise it returns 0

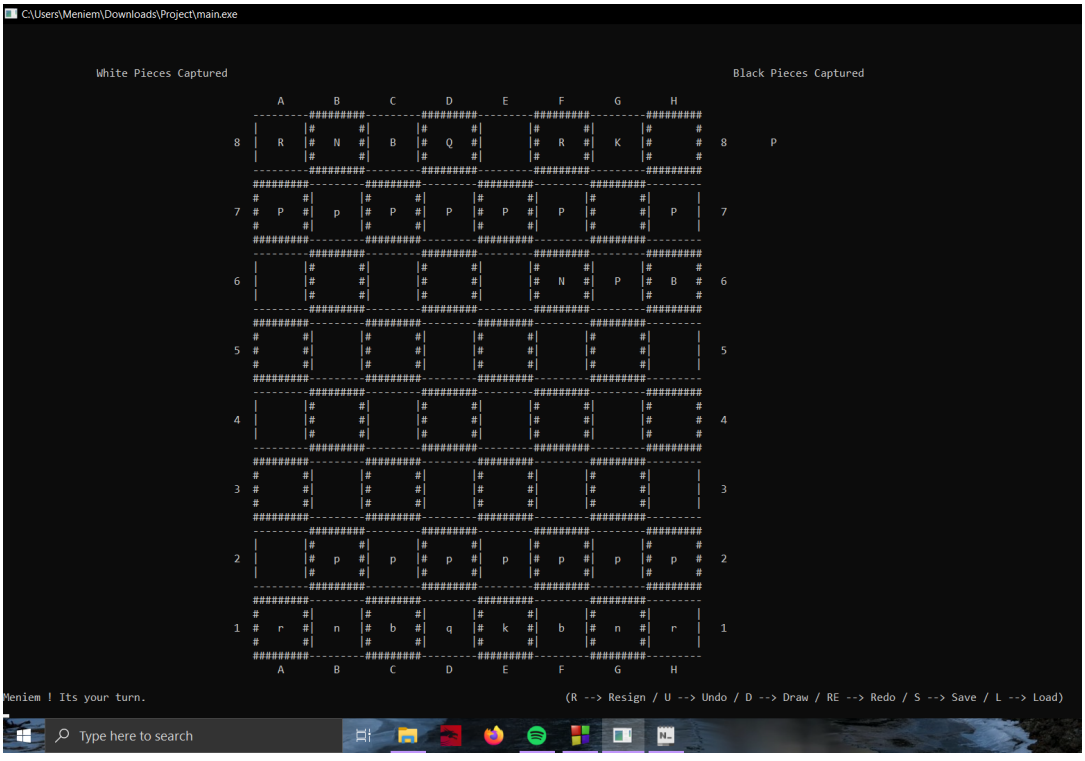
User Manual:

1. Enter both your names having a new line between them (*To go ahead with player 1 and player 2, type 0 --> Preferred if you have a saved game).
2. If not *, type 1 to take ‘entered input’ as player 1, 2 to take ‘entered input after new line’ as player 1 and type anything else for choosing player 1 randomly.
3. Enter the column name of the first block followed by its row name.
4. Enter the column name of the second block followed by its row name.
5. Ex: A7A5
6. (R --> Resign / U --> Undo / D --> Draw / RE --> Redo / S --> Save / L --> Load)
7. When promotion, follow your input by the piece you want to promote your pawn to (uppercase or lowercase according to which player you are 1 or 2).
8. Ex: A7A8q
9. NB: No spaces are allowed in input.

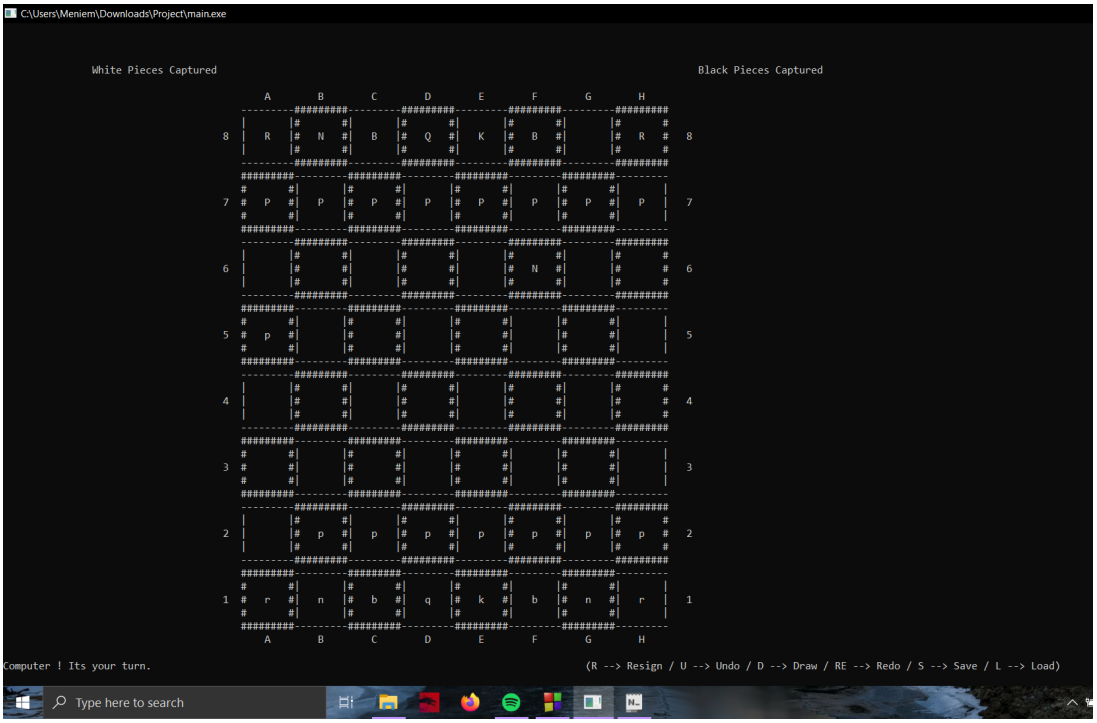
Sample Runs:

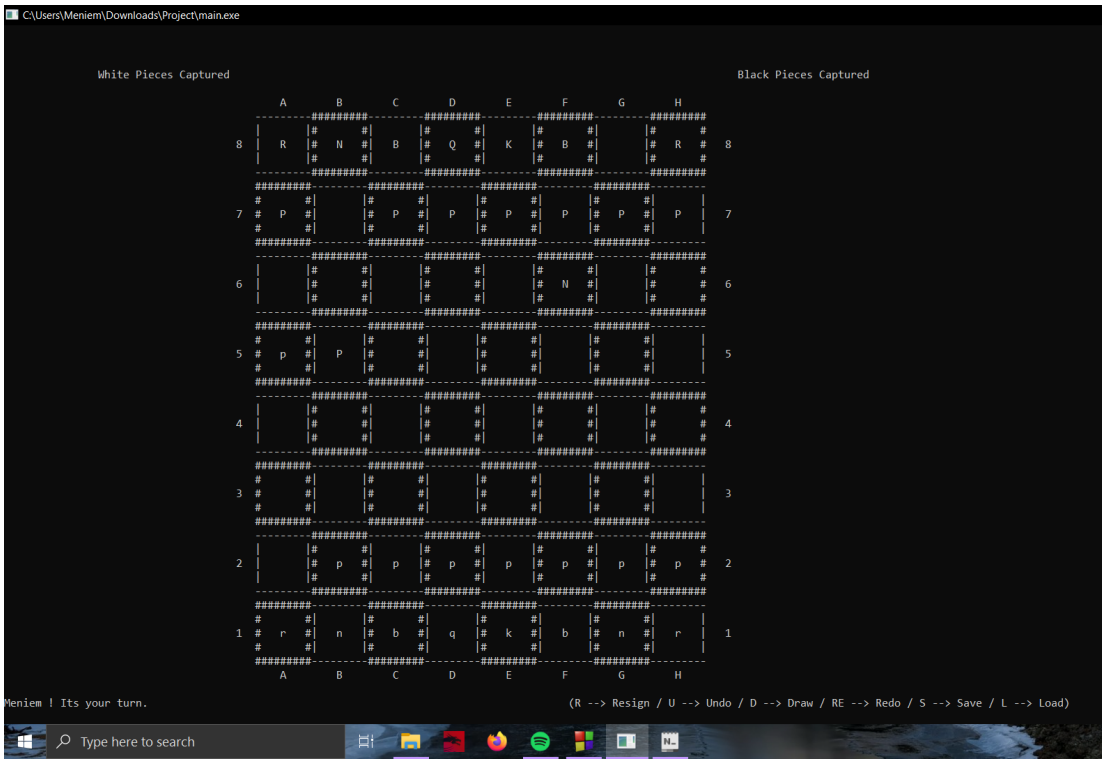
Brief sample run for castling move:

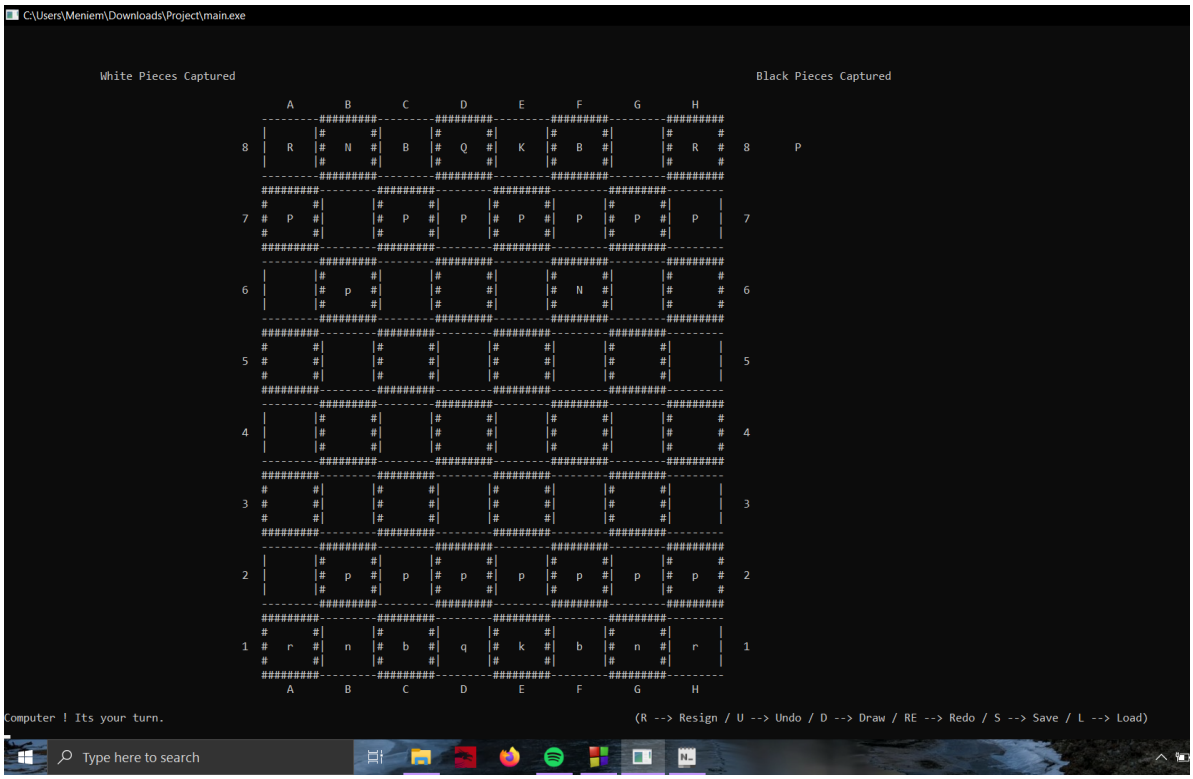




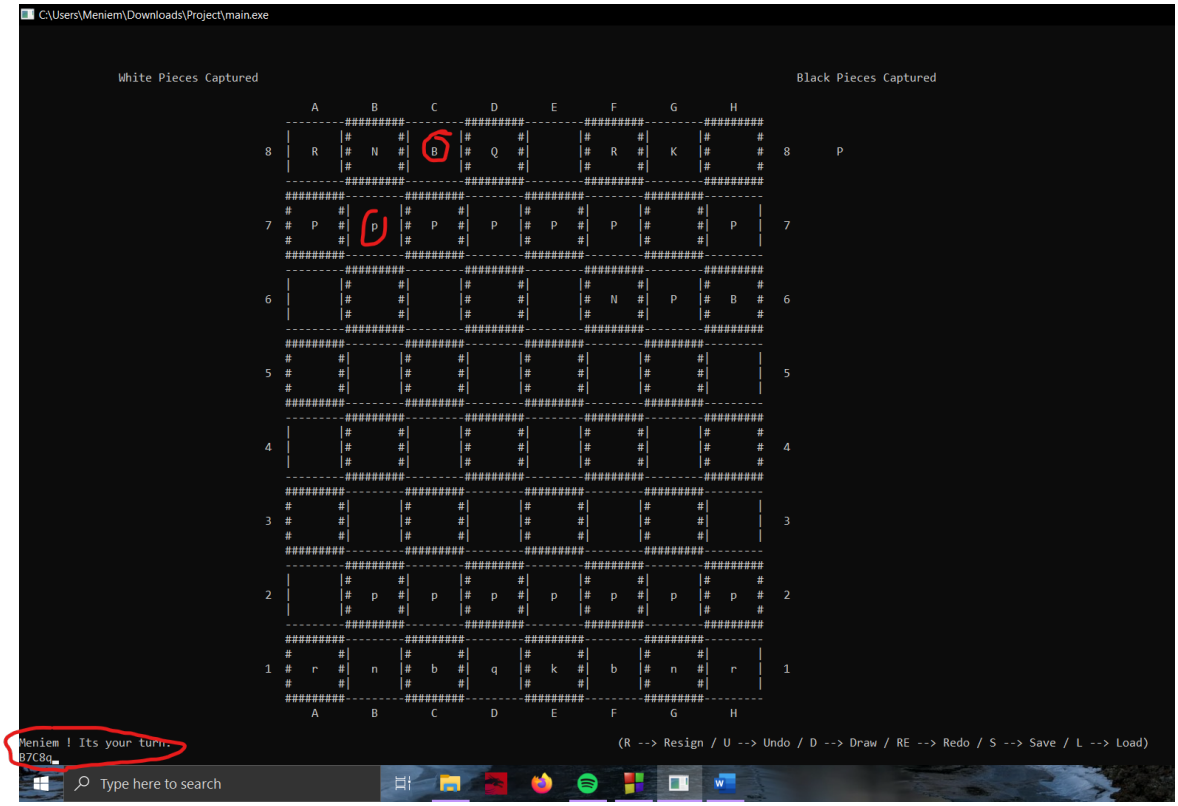
Brief sample run for En passant move:

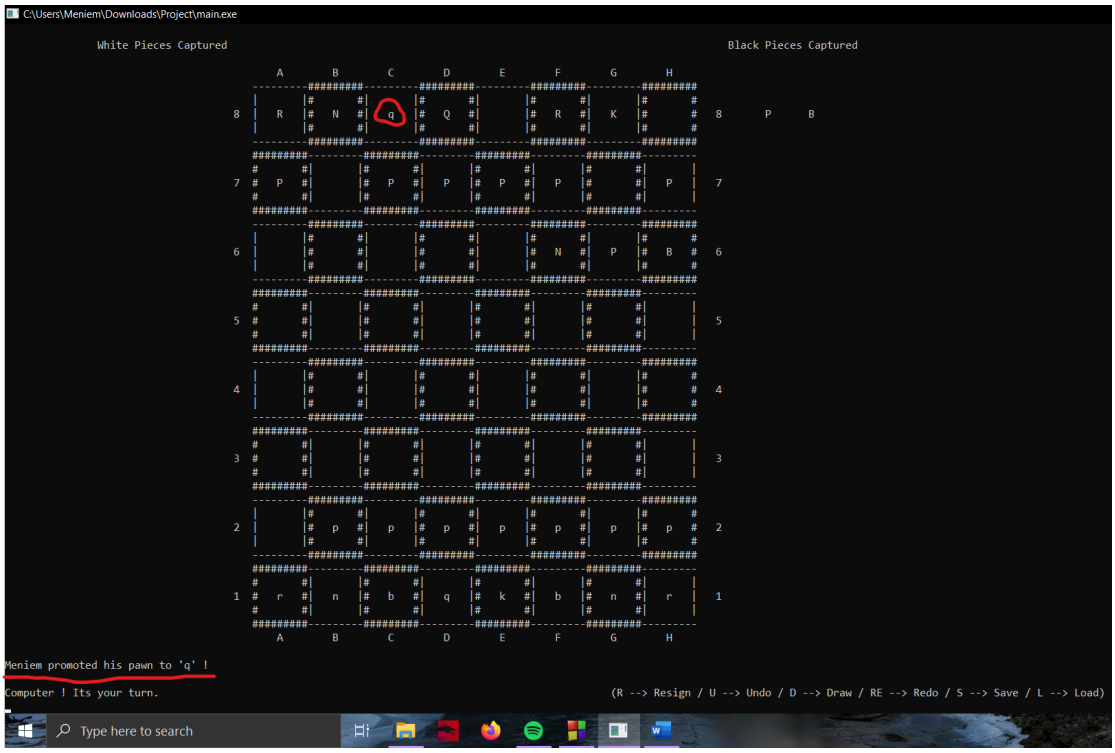




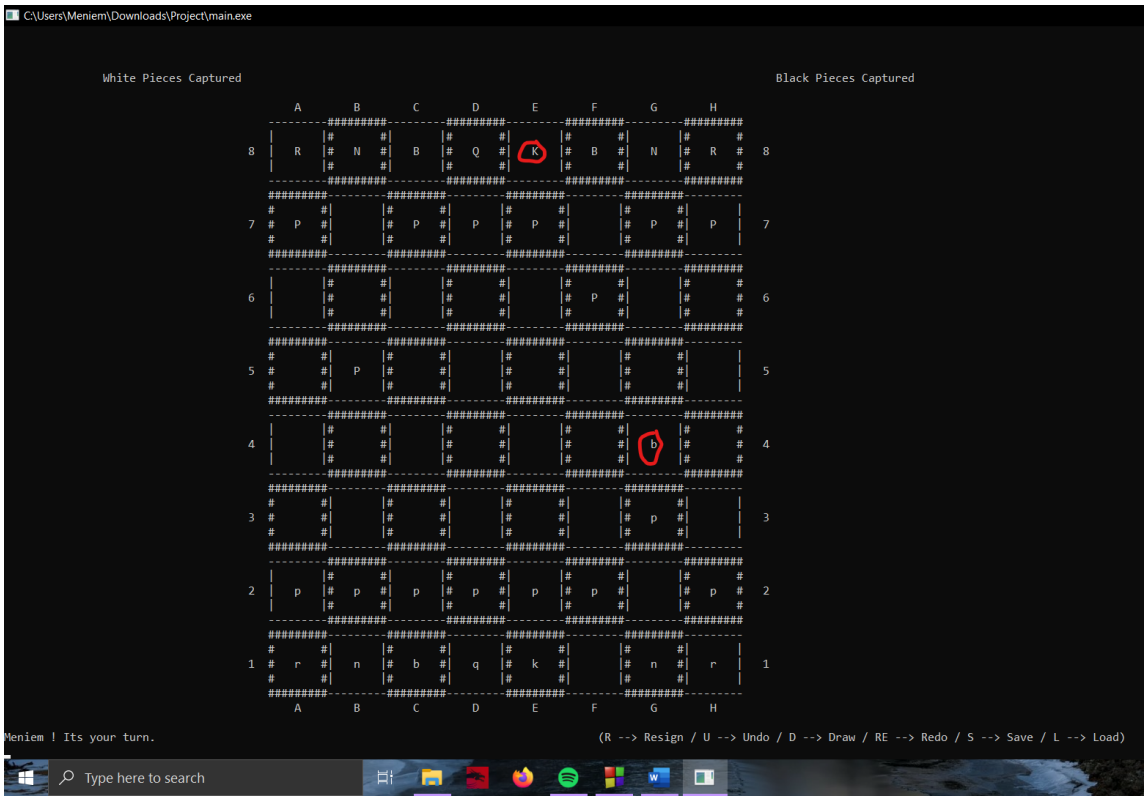


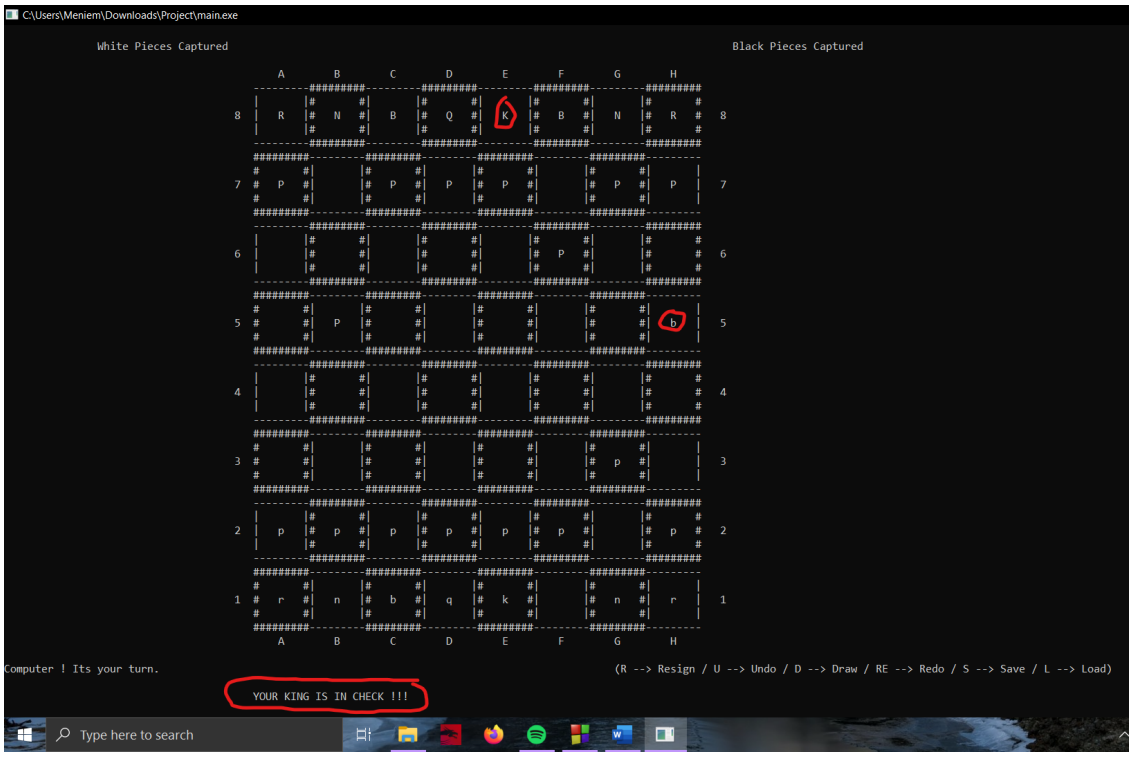
Brief sample run for Promotion:



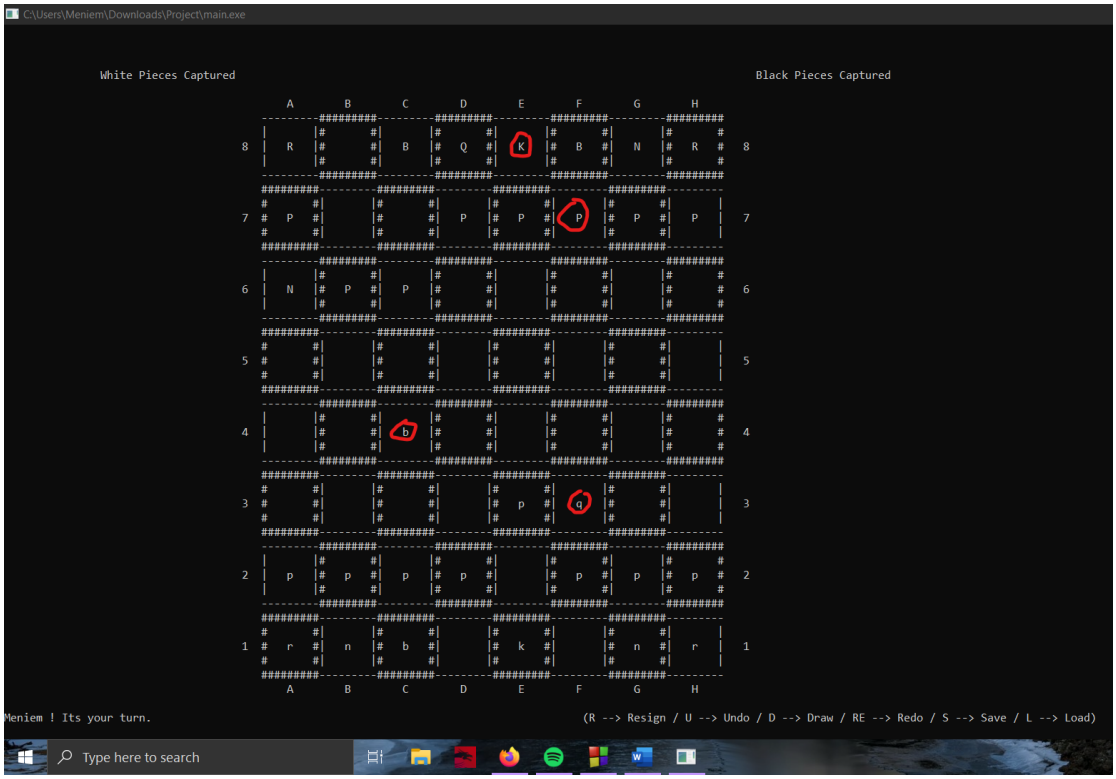


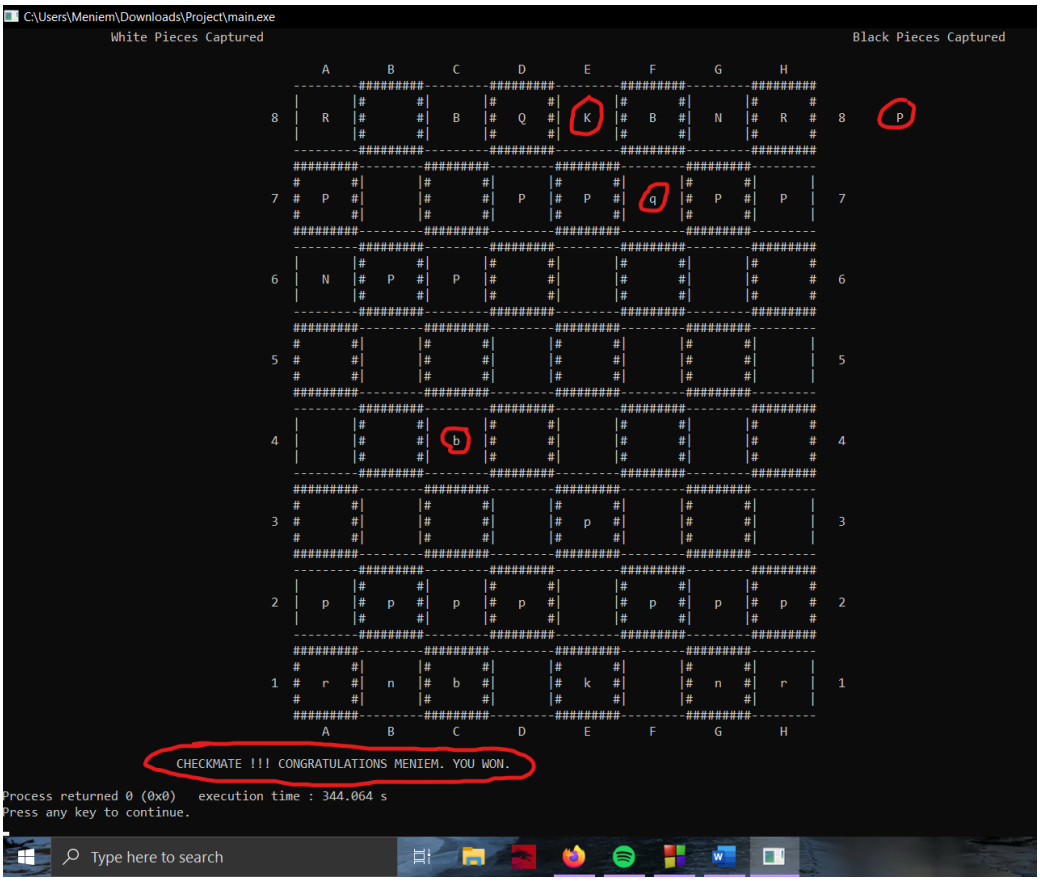
Brief sample run for check:



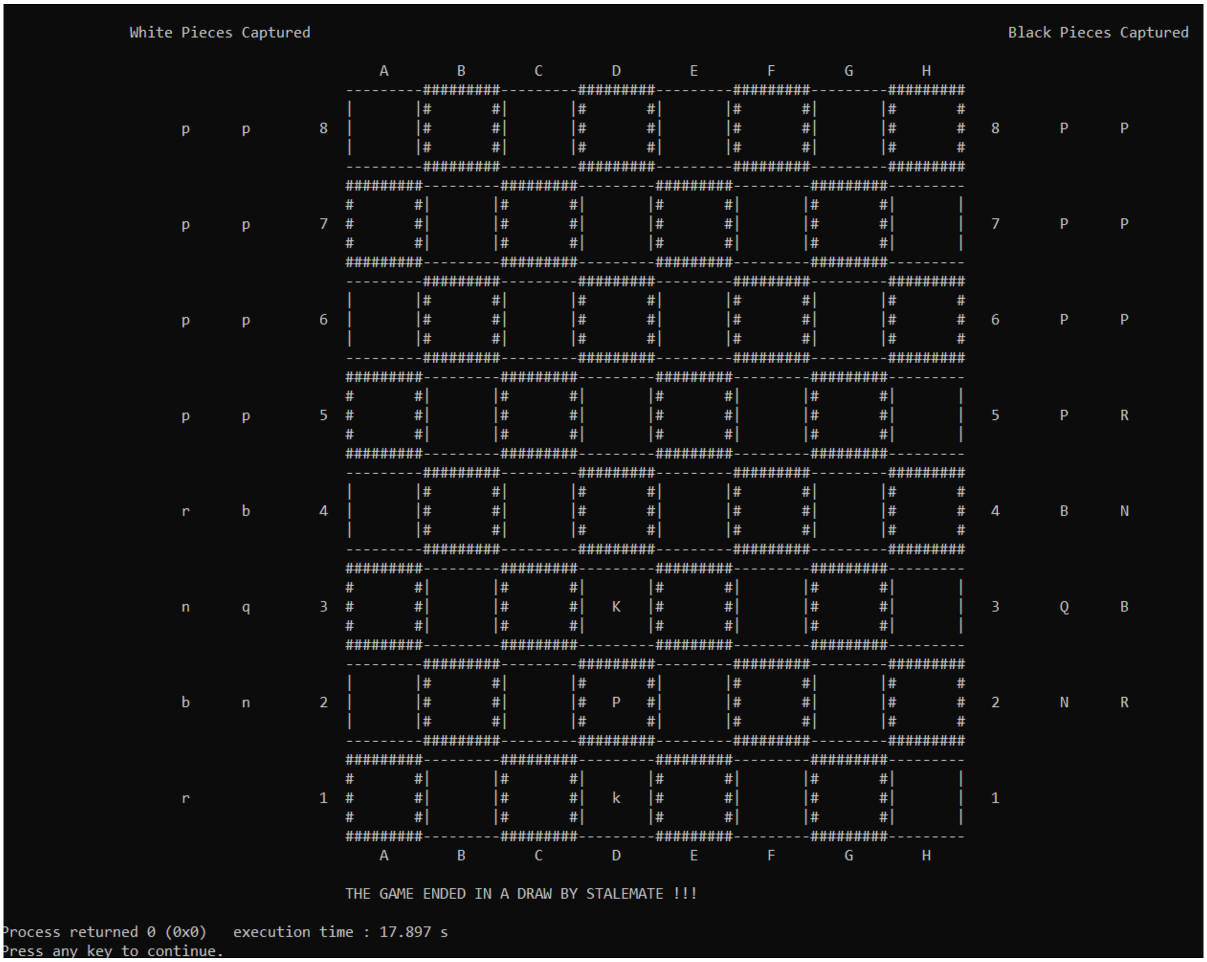


Brief sample run for Napoleon Checkmate:

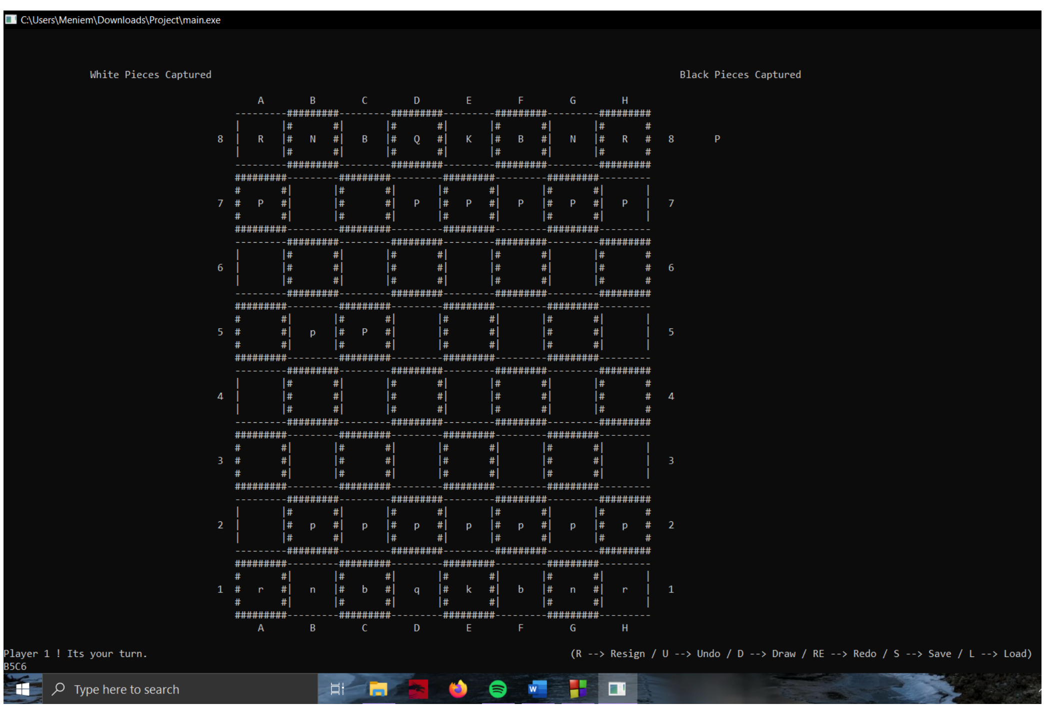


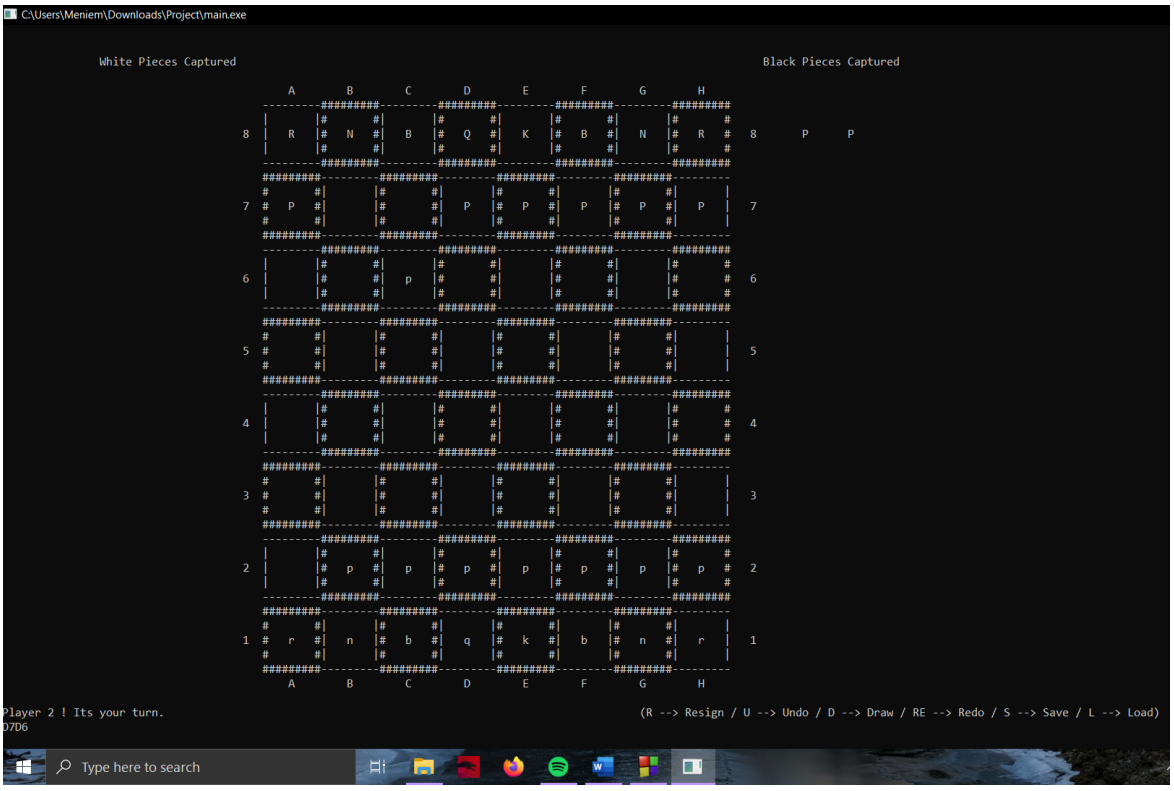


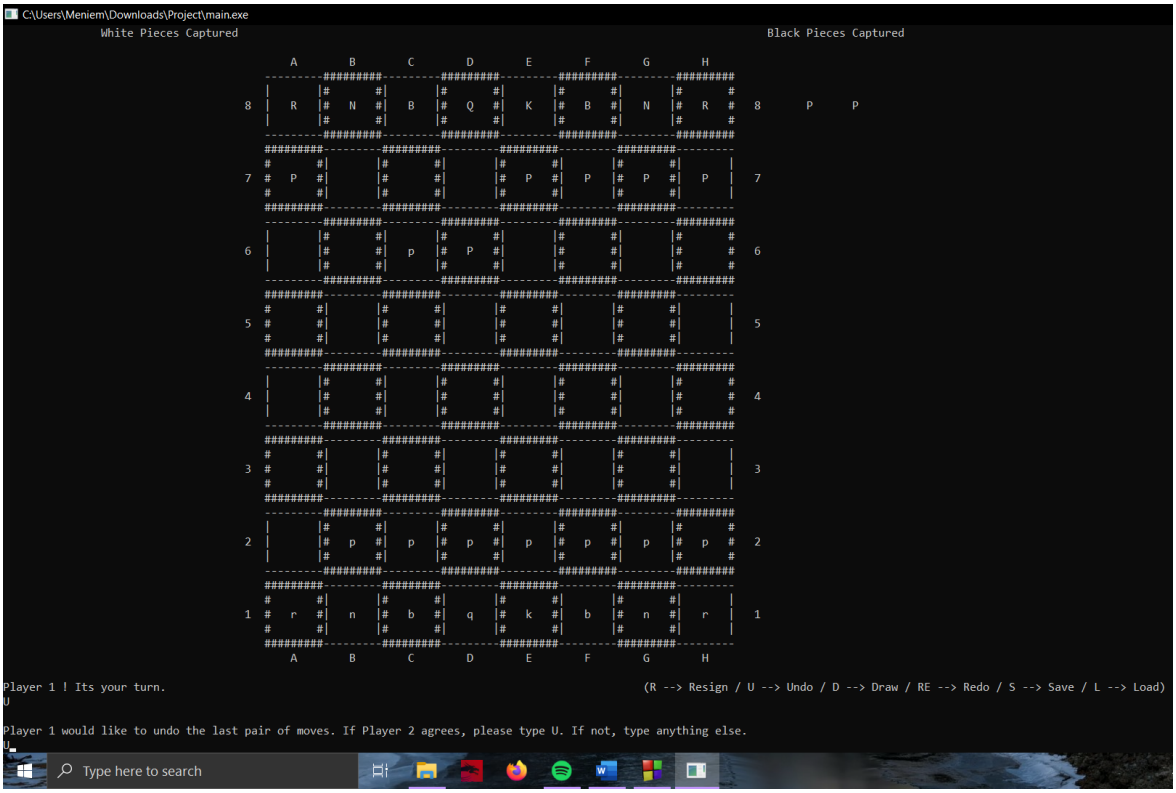
Brief sample run for stalemate:

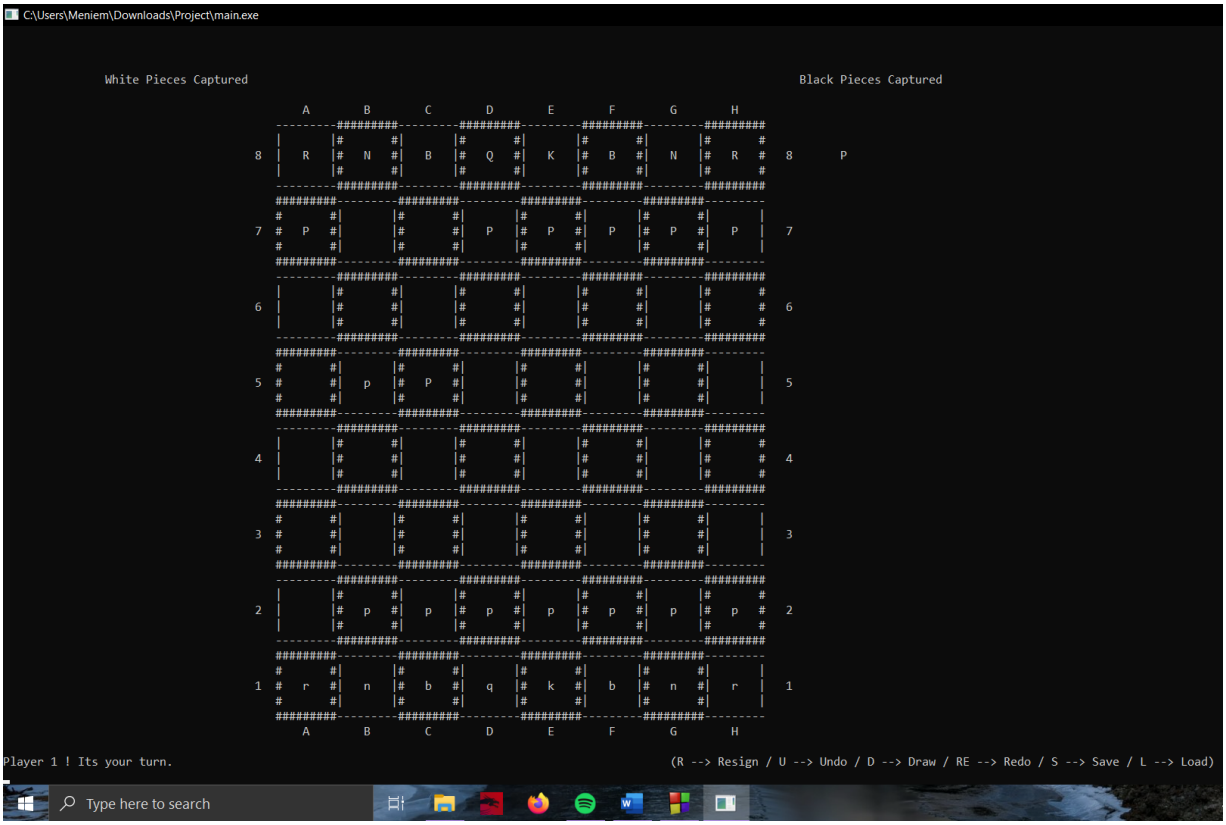


Brief sample run for Undo and Redo:

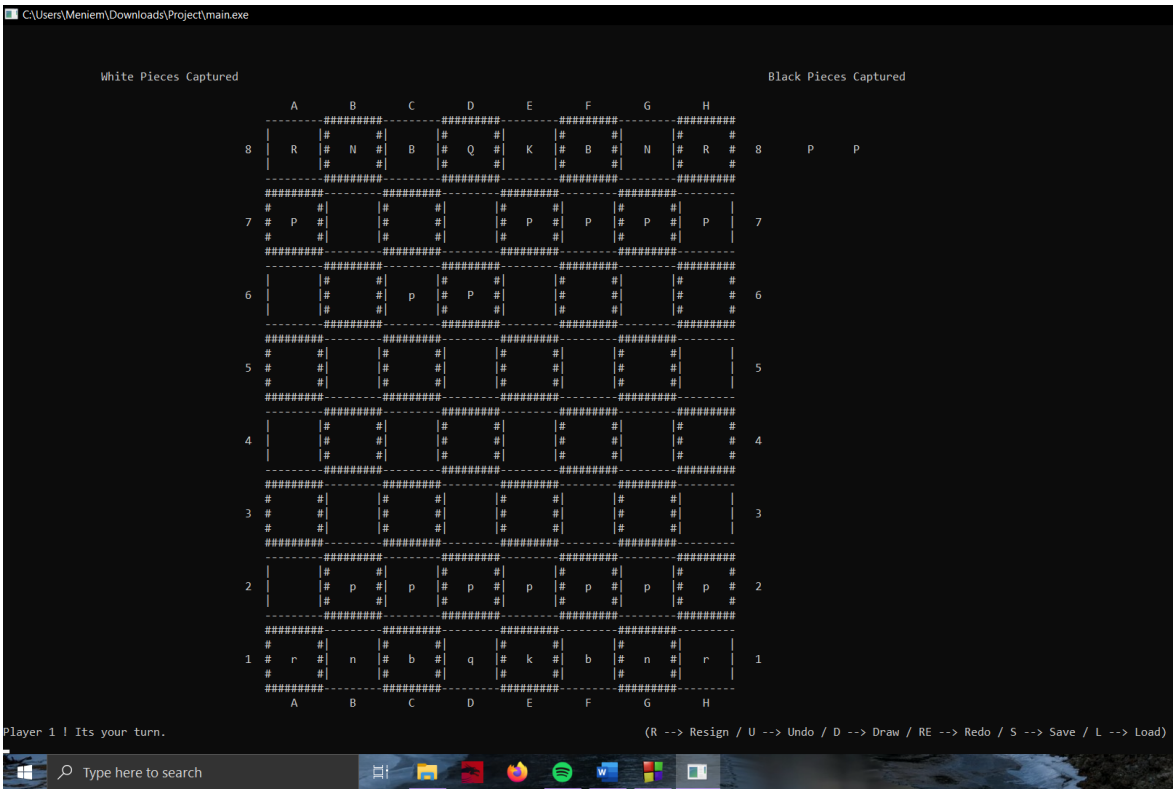



C:\Users\Meniem\Downloads\Project\main.exe
White Pieces Captured
Black Pieces Captured
P P
Player 2 ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / RE --> Redo / S --> Save / L --> Load)
D7D6
Type here to search

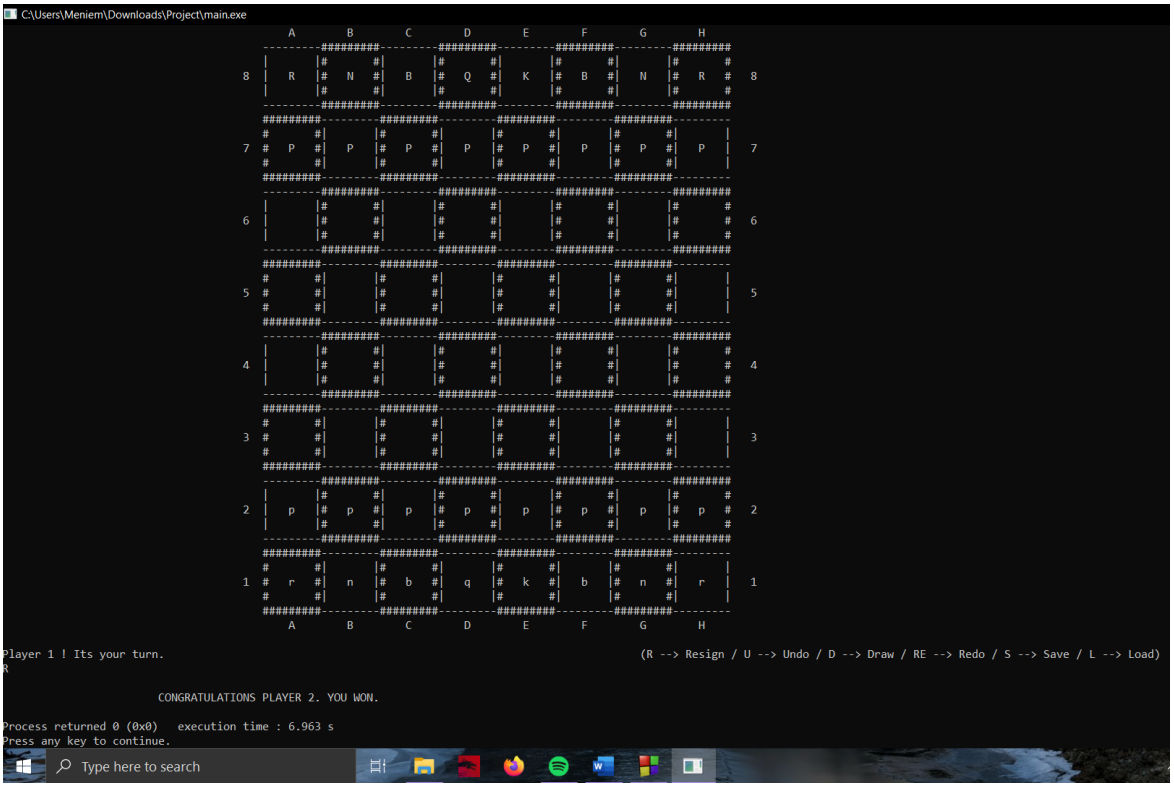

C:\Users\Meniem\Downloads\Project\main.exe
White Pieces Captured
Black Pieces Captured
P P
Player 1 ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / RE --> Redo / S --> Save / L --> Load)
U
Player 1 would like to undo the last pair of moves. If Player 2 agrees, please type U. If not, type anything else.
U
Type here to search

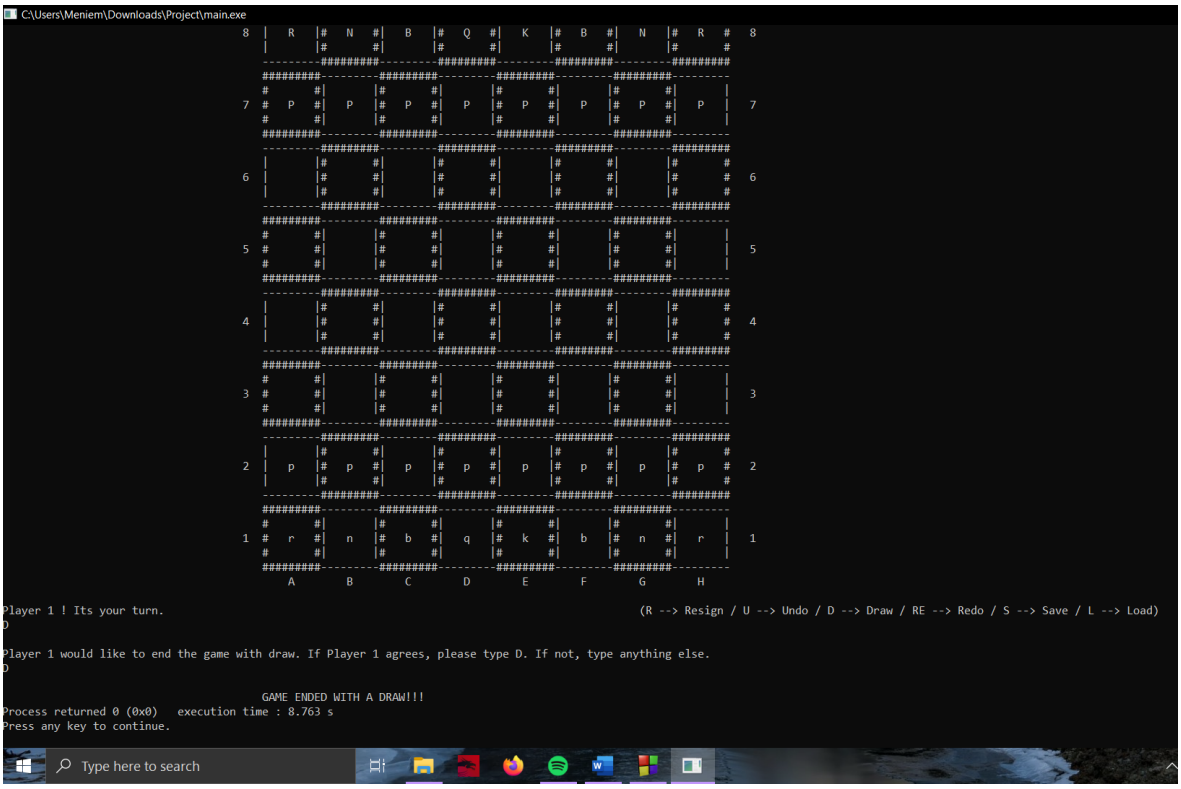

C:\Users\Meniem\Downloads\Project\main.exe
White Pieces Captured
Black Pieces Captured
A B C D E F G H
8 R N B Q K B N R 8
P
7 P P P P P P P P 7
6 6
5 p P 5
4 4
3 3
2 p p p p p p p p 2
1 r n b q k b n r 1
A B C D E F G H
Player 1 ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / RE --> Redo / S --> Save / L --> Load)
Type here to search


C:\Users\Meniem\Downloads\Project\main.exe
A B C D E F G H
8 R N B Q K B N R 8
P
7 P P P P P P P P 7
6 6
5 p P 5
4 4
3 3
2 p 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
Player 1 ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / RE --> Redo / S --> Save / L --> Load)
RE
Player 1 would like to redo the last pair of moves. If Player 2 agrees, please type RE. If not, type anything else.
RE
Type here to search



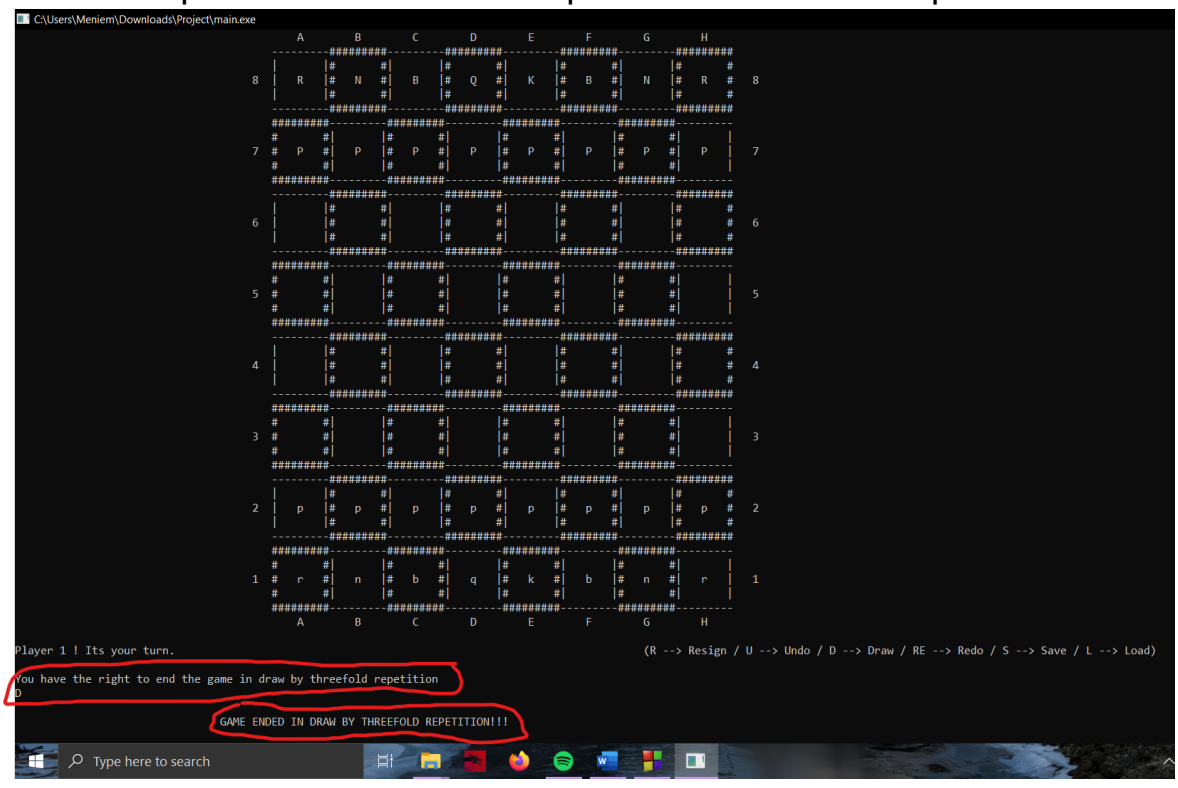
Brief sample run for Resign and Draw.

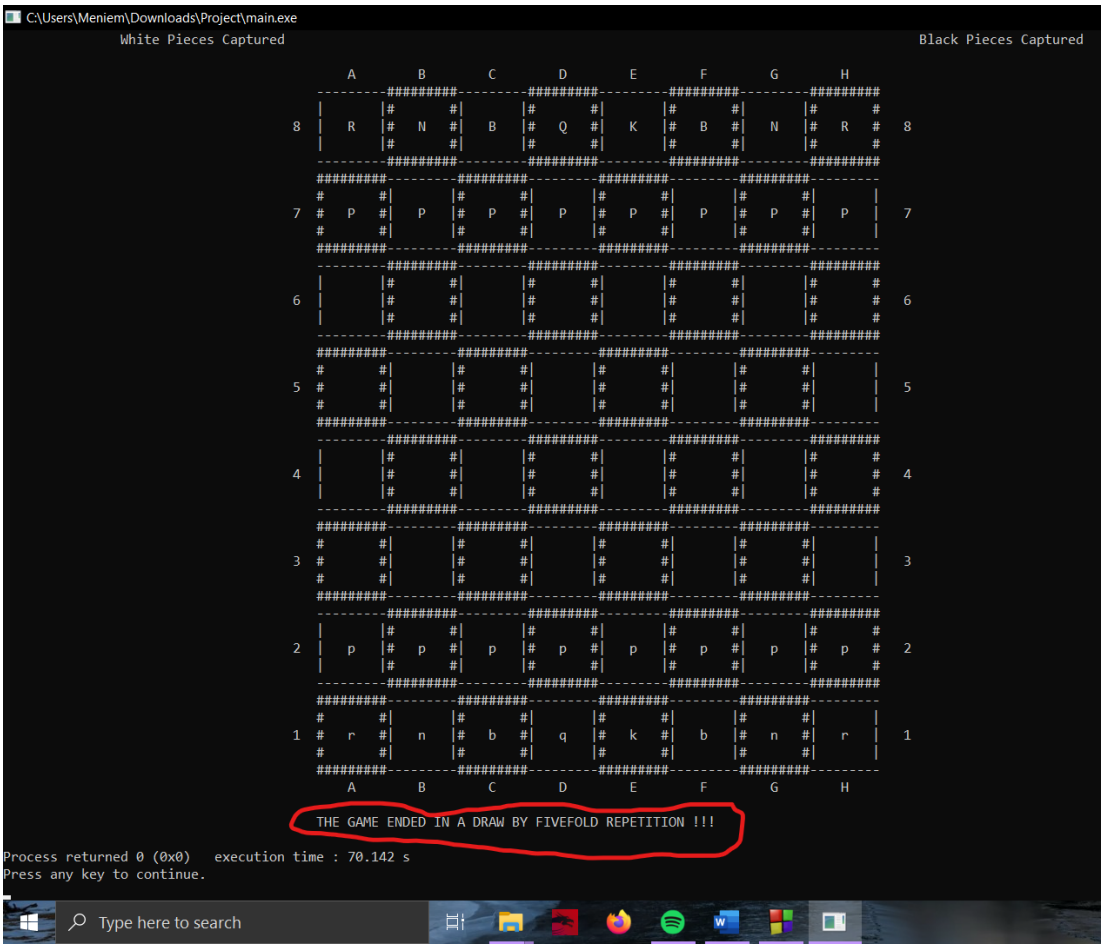




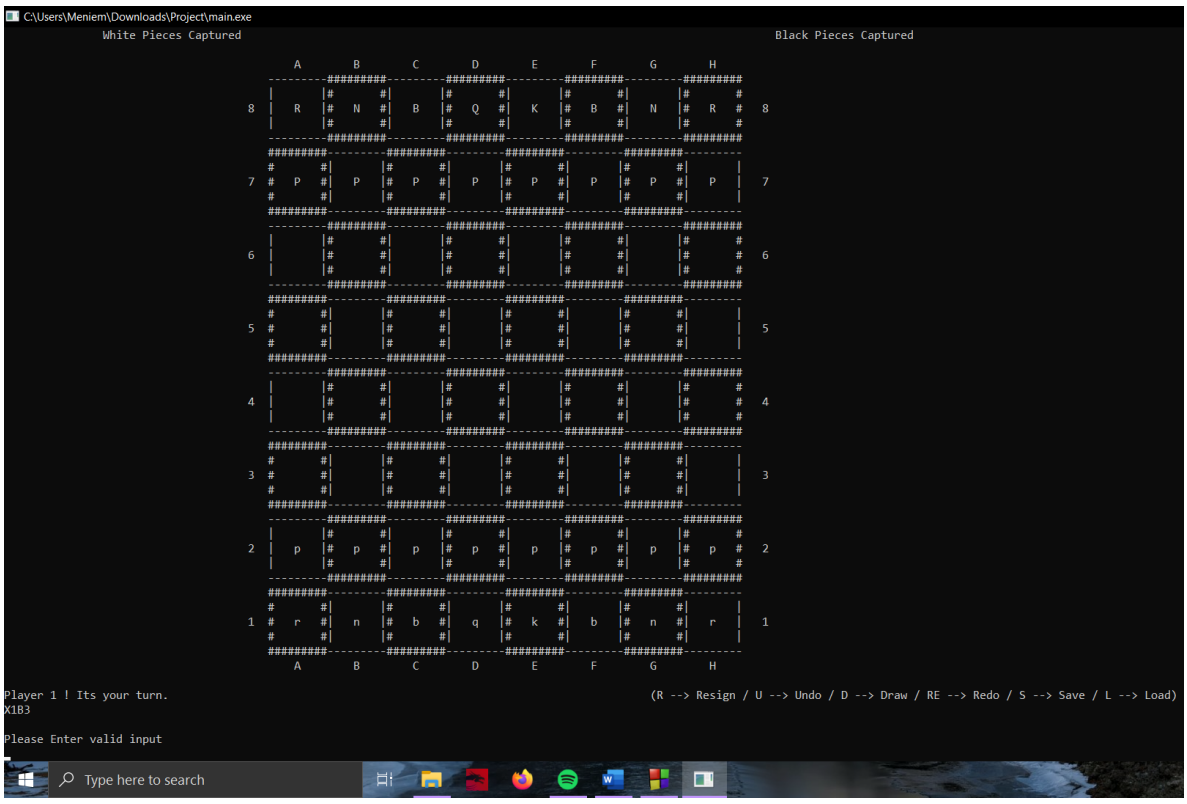
Brief sample run for Dead Position:

Brief sample runs for threefold repetition and fivefold repetition:

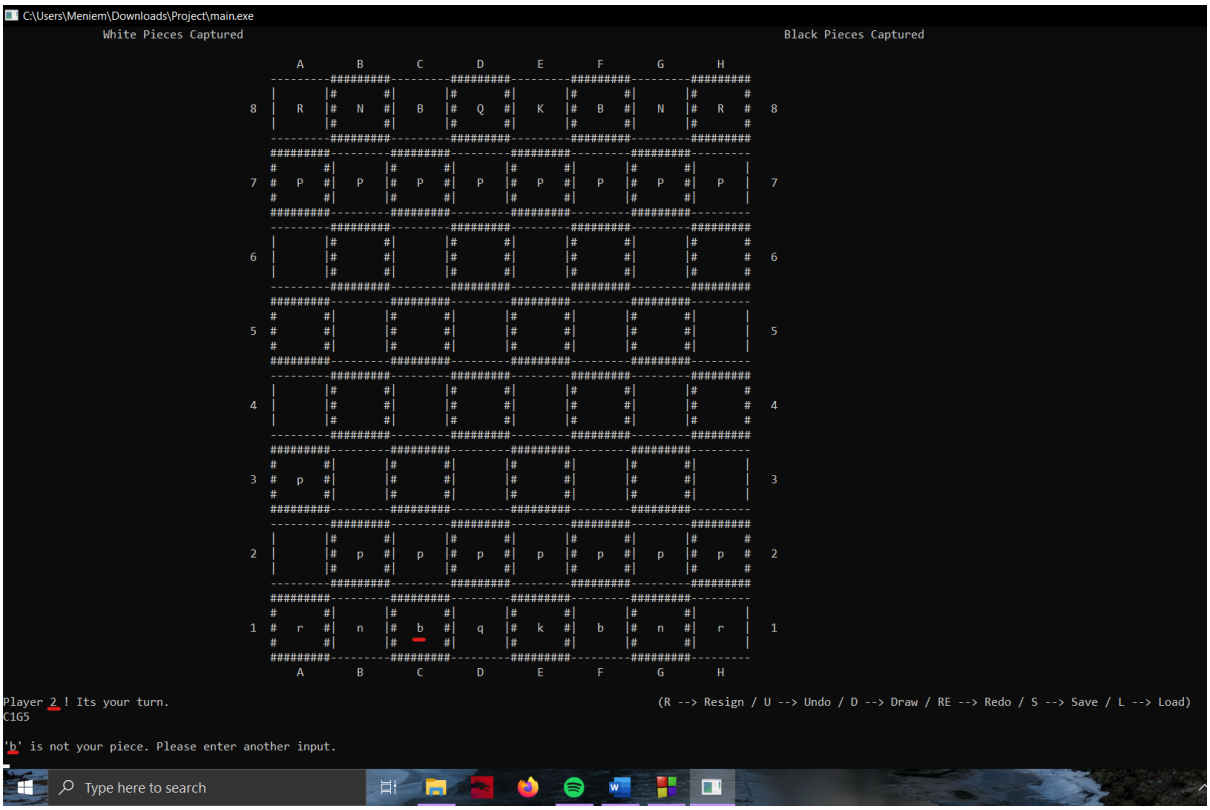




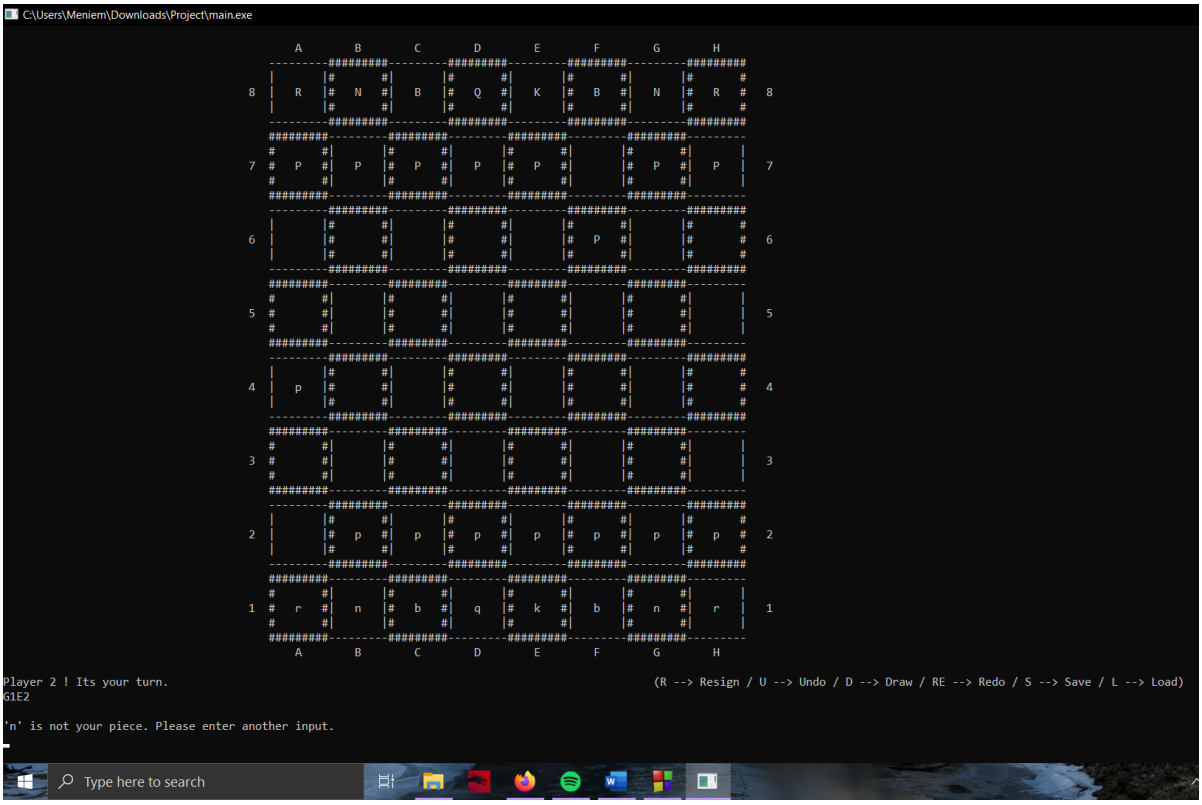
Sample Runs for some Invalid Inputs and wrong chess piece movements:




C:\Users\Meniem\Downloads\Project\main.exe
Player 1 ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / RE --> Redo / S --> Save / L --> Load)
X1B3
Please Enter valid input
A1A5
'r' cannot move here. Enter another input.
Type here to search

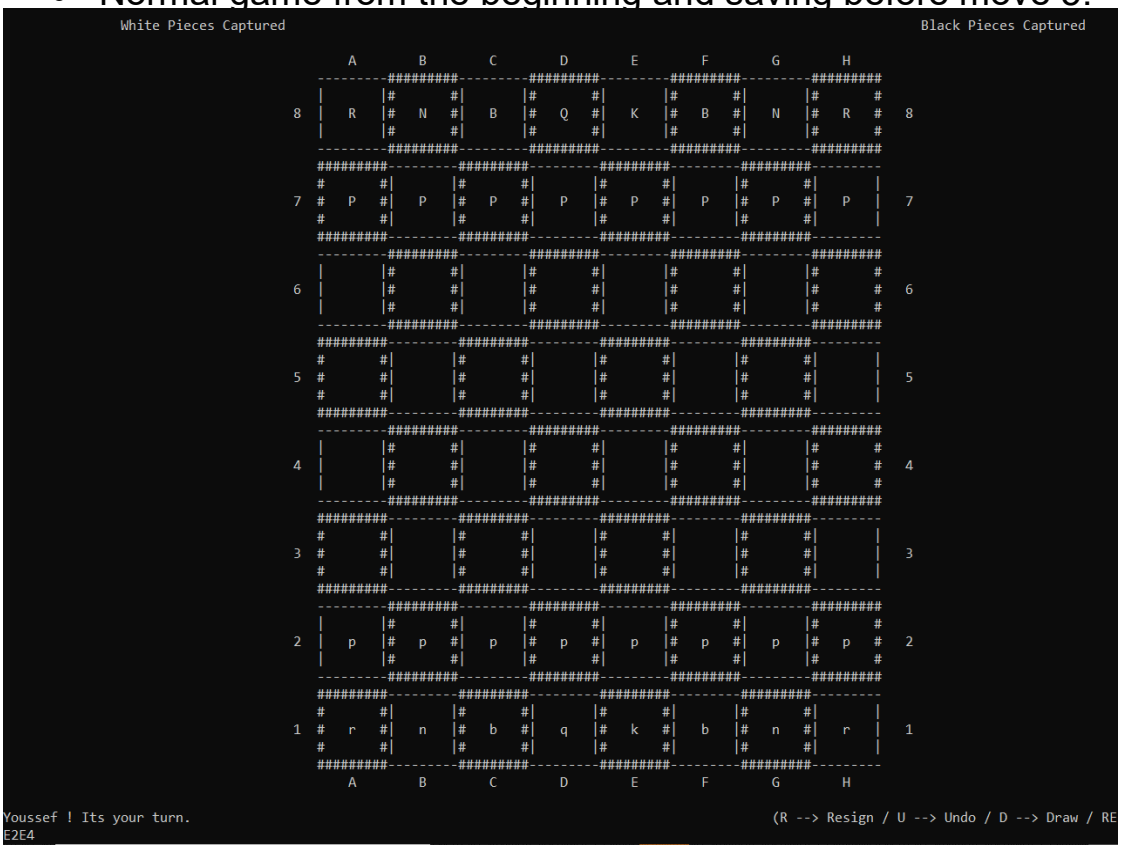

C:\Users\Meniem\Downloads\Project\main.exe
White Pieces Captured
Black Pieces Captured
Player 2 ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / RE --> Redo / S --> Save / L --> Load)
C1G5
'b' is not your piece. Please enter another input.
Type here to search

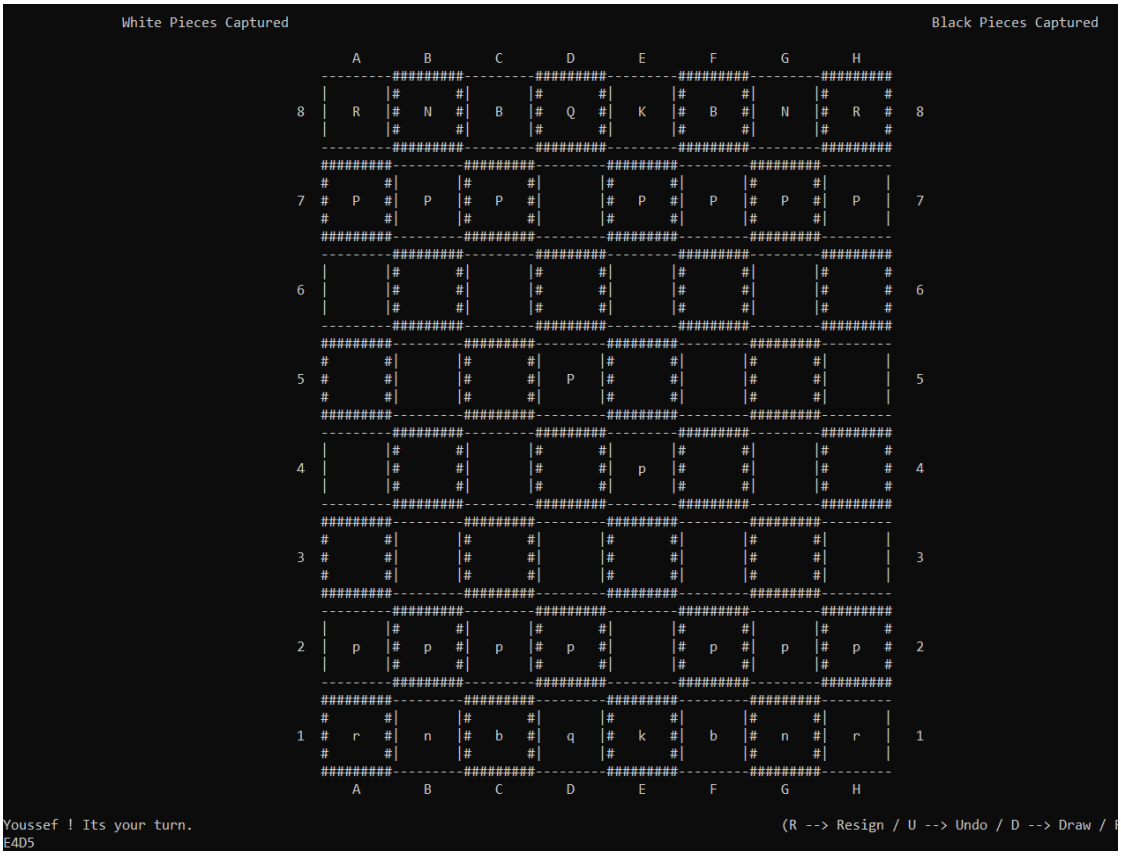

C:\Users\Meniem\Downloads\Project\main.exe
A B C D E F G H
Player 2 ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / RE --> Redo / S --> Save / L --> Load)
C1G5
'b' is not your piece. Please enter another input.
C8H3
'B' cannot move here. Enter another input.
Type here to search

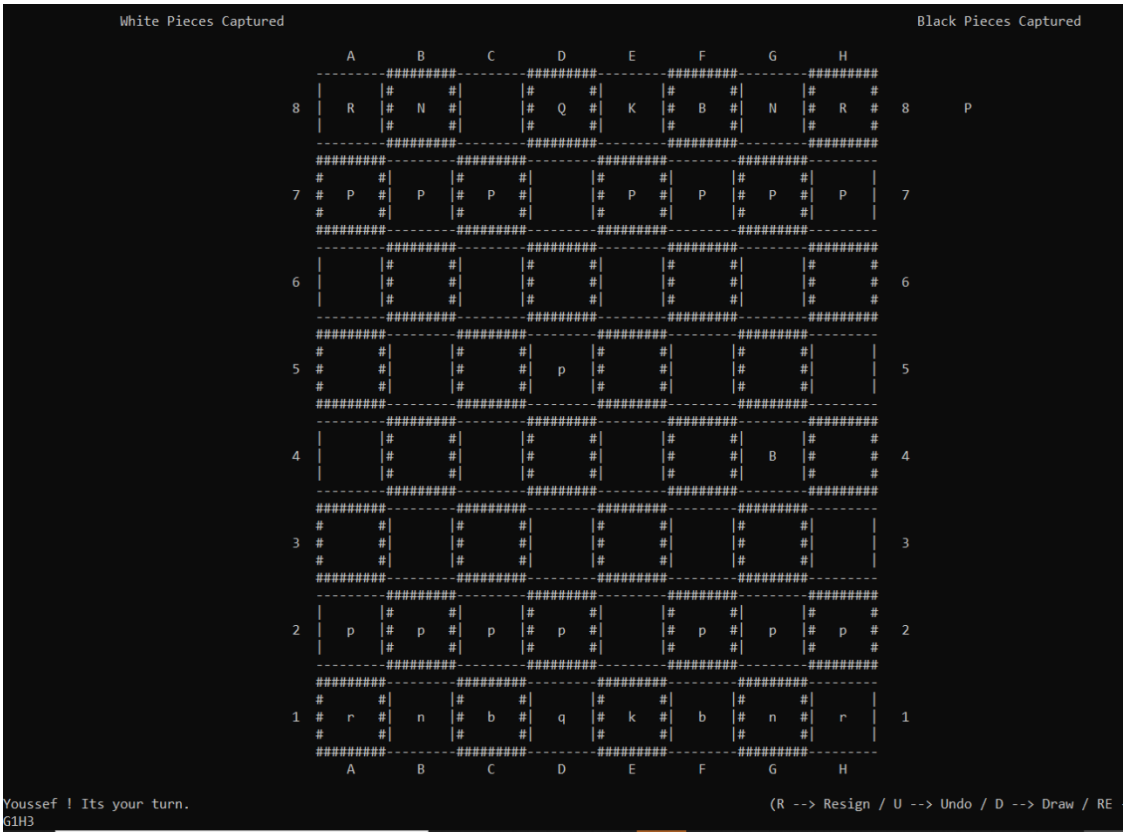

C:\Users\Meniem\Downloads\Project\main.exe
A B C D E F G H
Player 2 ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / RE --> Redo / S --> Save / L --> Load)
G1E2
'n' is not your piece. Please enter another input.
Type here to search

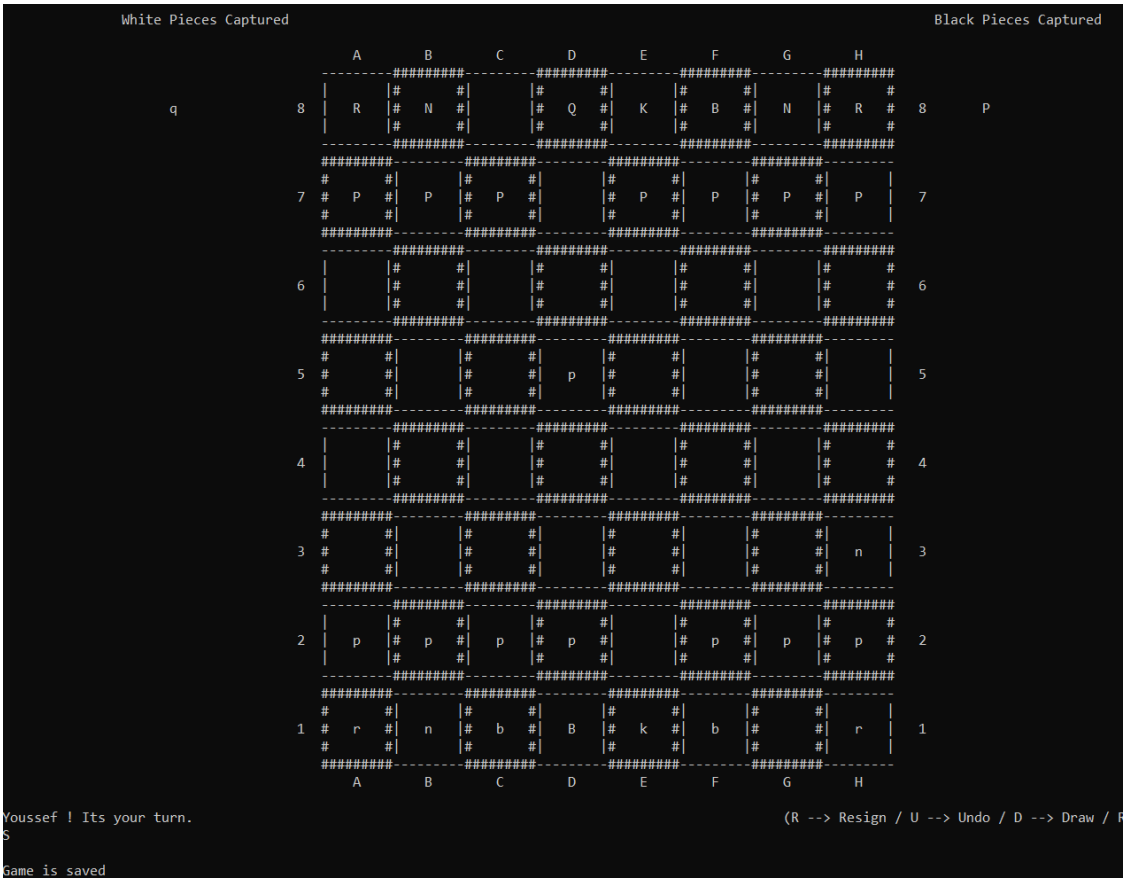
Brief sample run for save and load:

- Normal game from the beginning and saving before move 5:

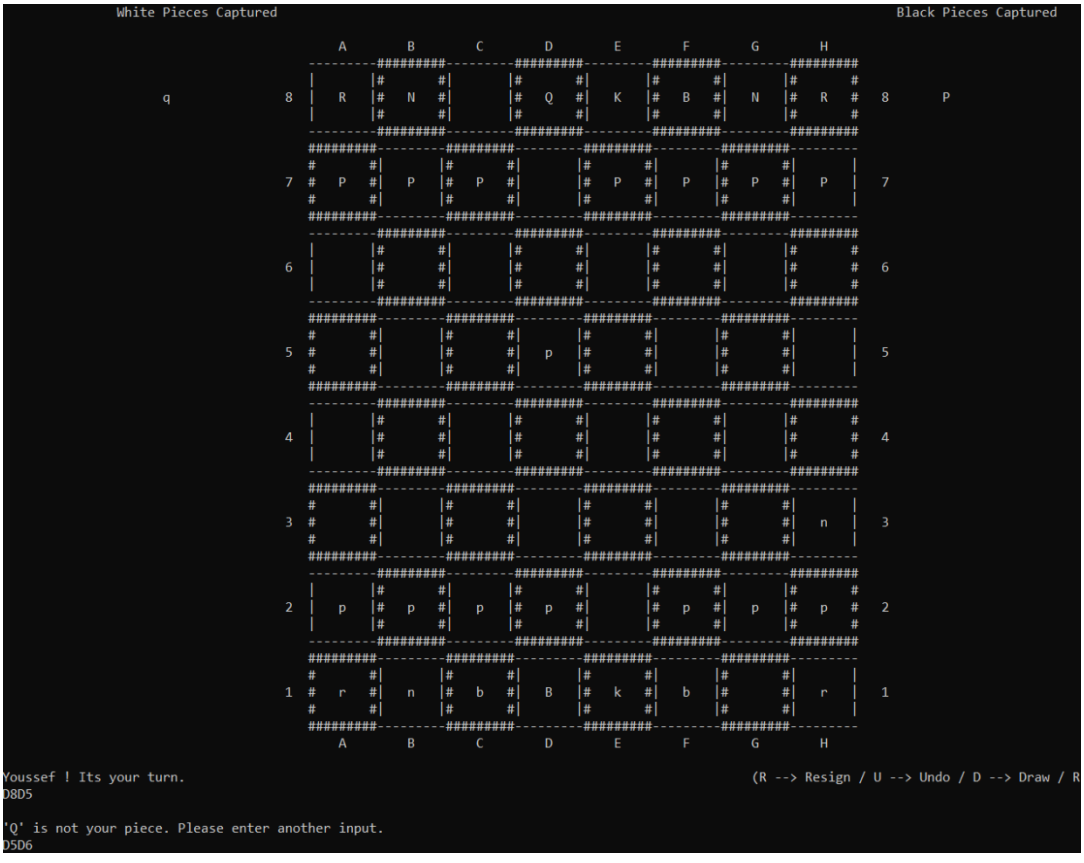





White Pieces Captured
Black Pieces Captured
A B C D E F G H
8 R N Q K B N R 8
7 P P P P P P P P 7
6 6
5 p 5
4 B 4
3 3
2 p p p p p p p p 2
1 r n b q k b n r 1
A B C D E F G H
Youssef ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / RE
G1H3

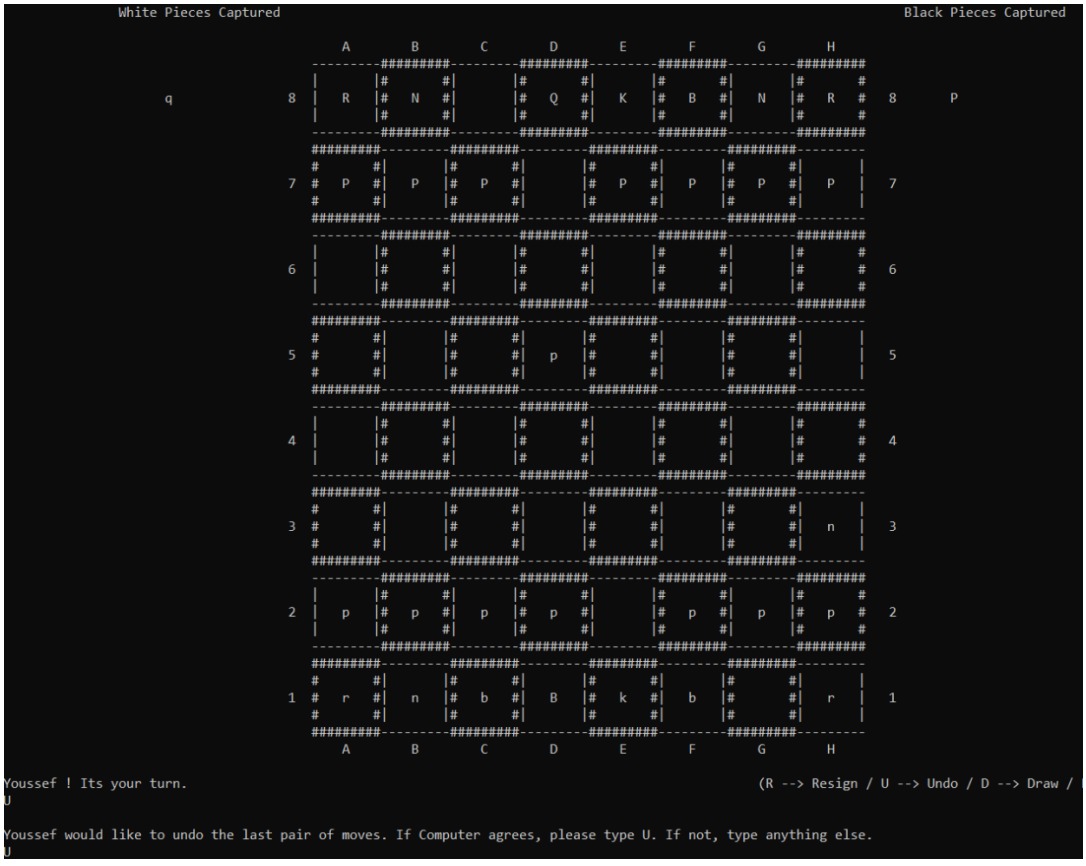

White Pieces Captured
Black Pieces Captured
q
A B C D E F G H
8 R N Q K B N R 8
7 P P P P P P P P 7
6 6
5 p 5
4 4
3 n 3
2 p p p p p p p p 2
1 r n b B k b r 1
A B C D E F G H
Youssef ! Its your turn.
(R --> Resign / U --> Undo / D --> Draw / R
S
Game is saved

- Loading the last saved game:

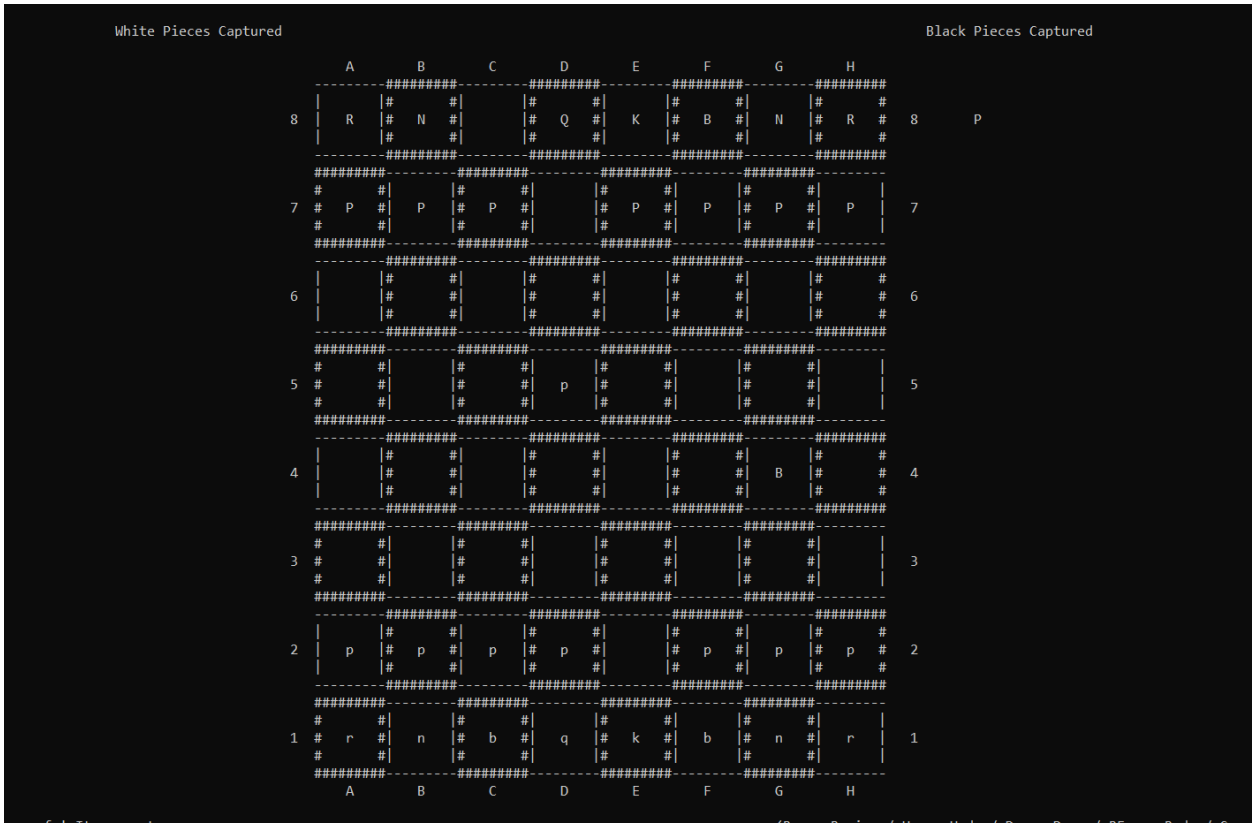


- Executing move 5 after loading:

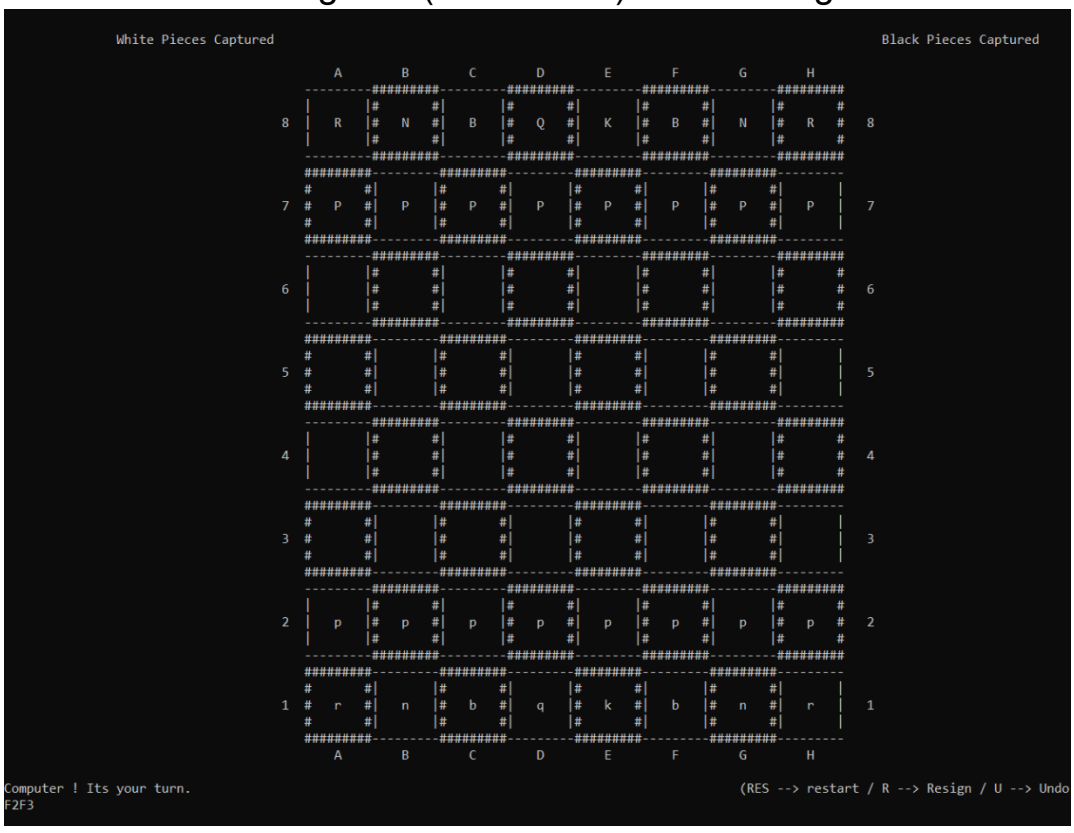
- Undo to move 4 (first move after load):

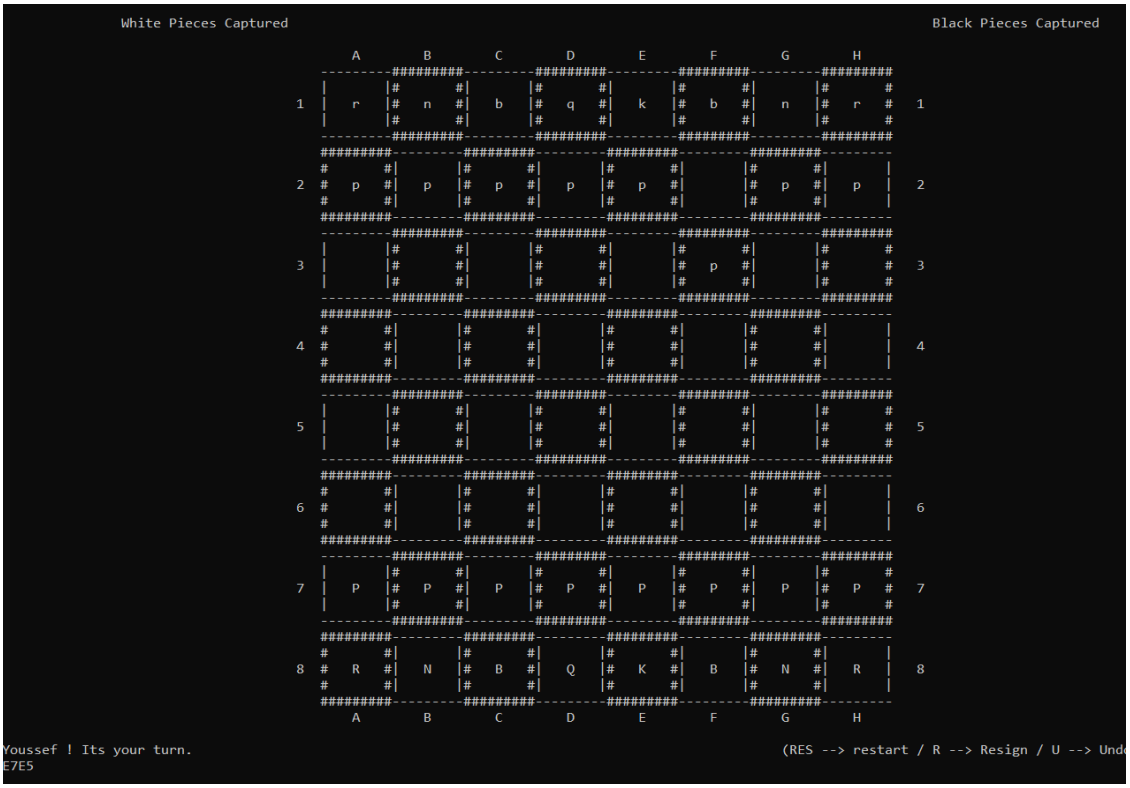


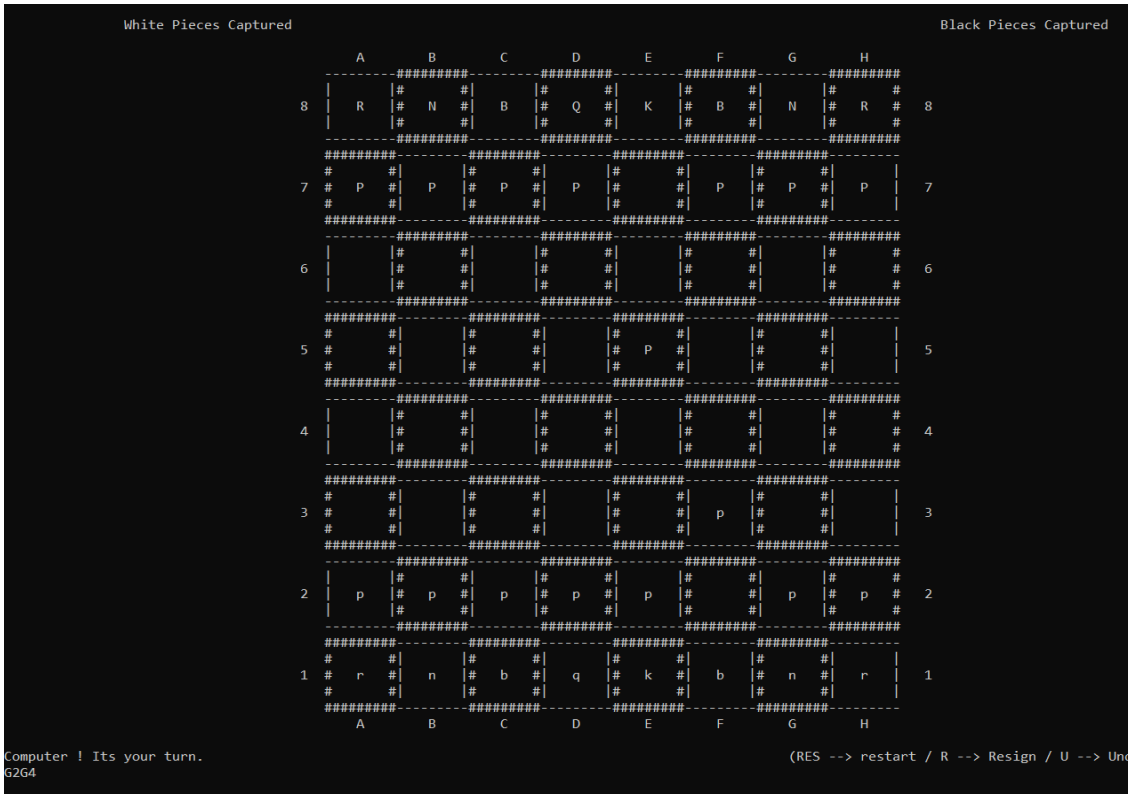
Undo to move 3 (last move before load):



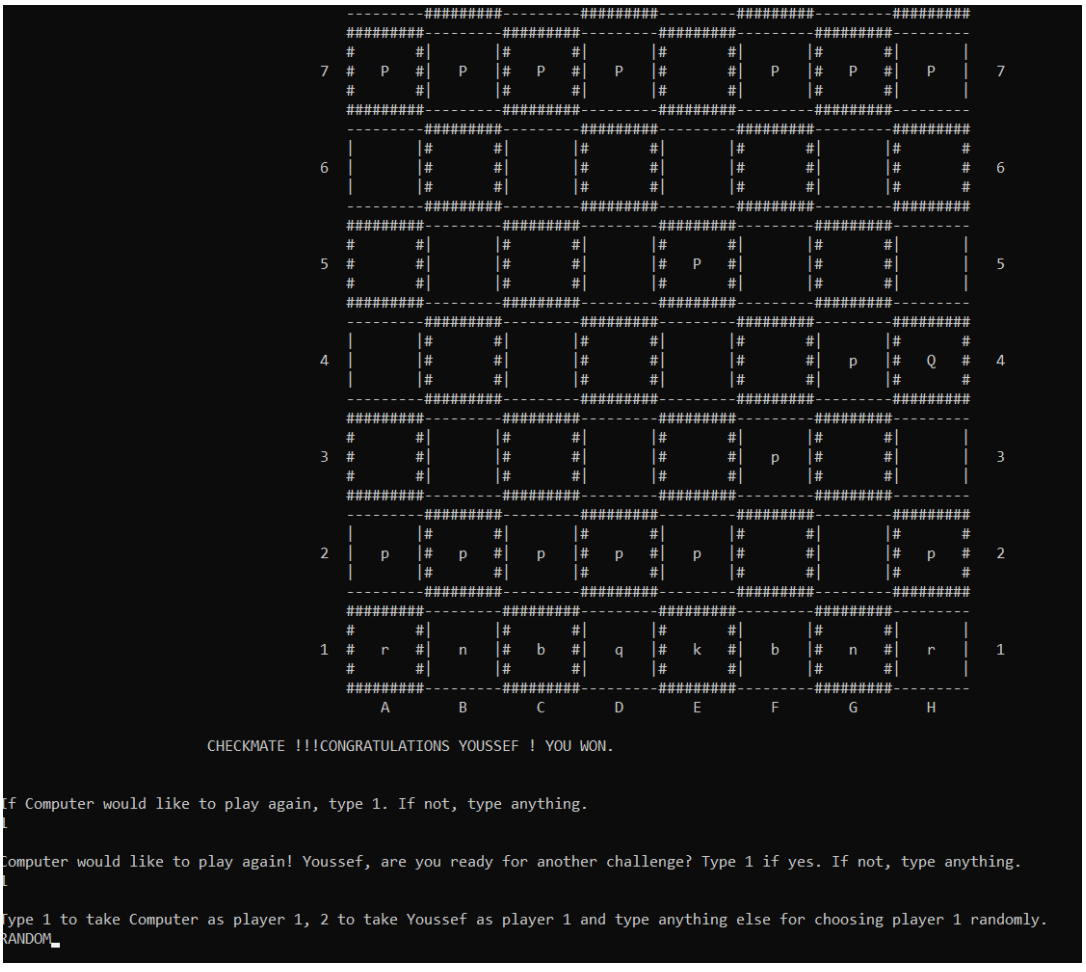
Showing a detailed 2 move game (fool's mate) after adding the reflection feature:

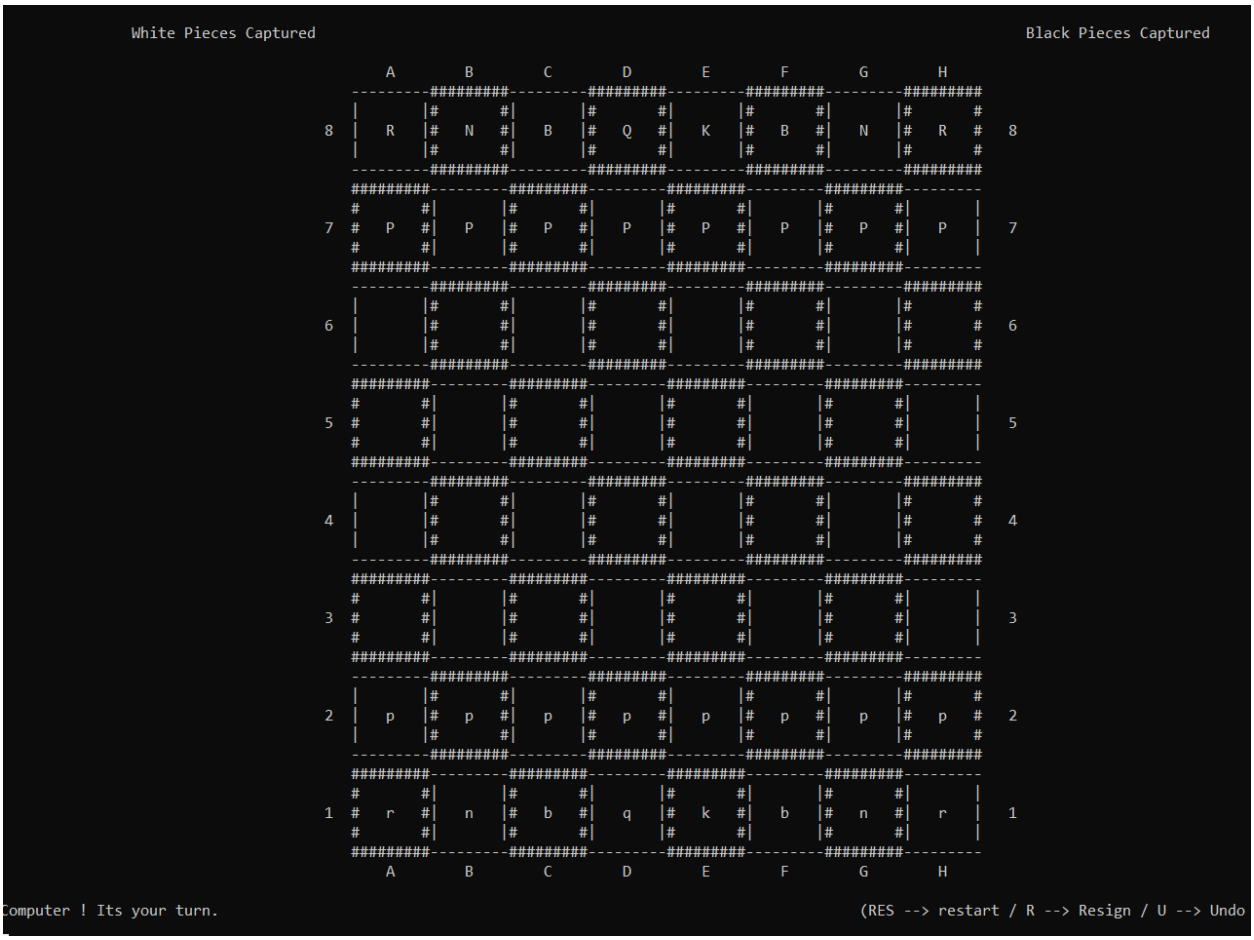



White Pieces Captured
Black Pieces Captured
A B C D E F G H
1 r n b q k b n r 1
2 p p p p p p p p 2
3 3
4 4
5 5
6 6
7 P P P P P P P P 7
8 R N B Q K B N R 8
A B C D E F G H
Youssef ! Its your turn.
(RES --> restart / R --> Resign / U --> Und
E7E5

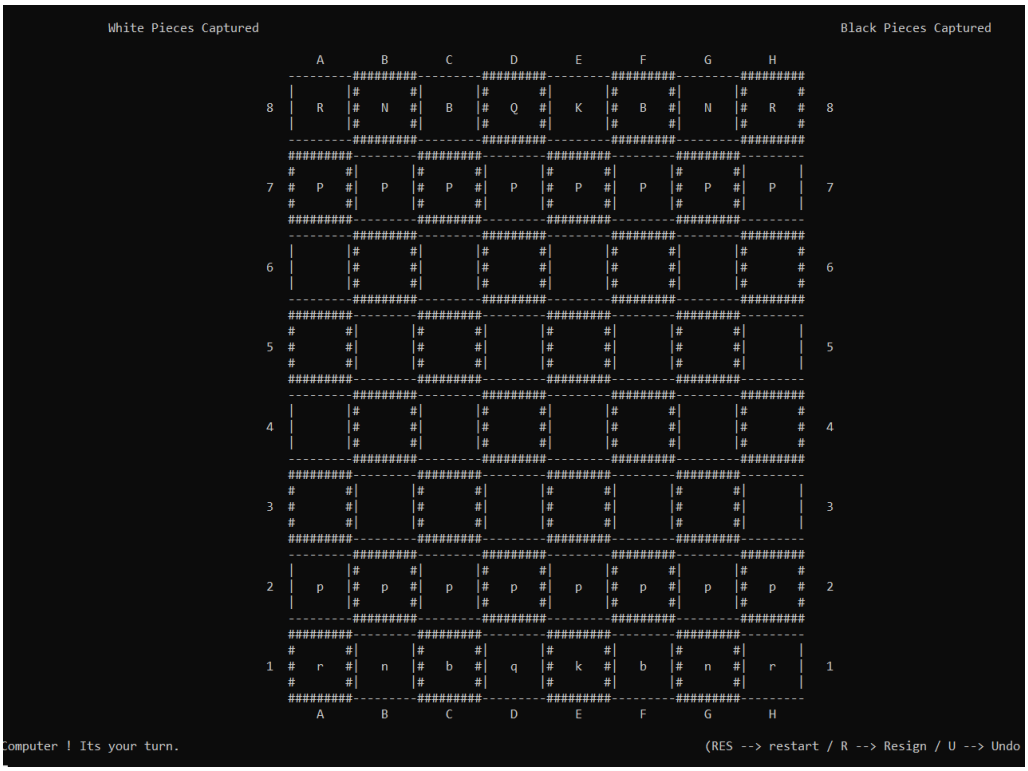

White Pieces Captured
Black Pieces Captured
A B C D E F G H
8 R N B Q K B N R 8
7 P P P P P P P P 7
6 6
5 P 5
4 4
3 p 3
2 p p p p p p p p 2
1 r n b q k b n r 1
A B C D E F G H
Computer ! Its your turn.
(RES --> restart / R --> Resign / U --> Und
G2G4

Playing again after the game had ended:




White Pieces Captured
Black Pieces Captured
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 P P 7
6 6
5 5
4 4
3 3
2 p 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
Computer ! Its your turn.
(RES --> restart / R --> Resign / U --> Undo

Restarting the game in the middle of a current one:



Program termination:

References:

- https://stackoverflow.com/questions/9241025/wait-pause-an-amount-of-seconds-in-c/9241040
- https://en.wikipedia.org/wiki/Chess