

Chessmate

Description of the game and features it provides

Chessmate is a console chess game written in C.

Features:

- **Two players: Human VS Human.**
- **User interface:**
 - 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.
 - Removed pieces are displayed next to the board, The white pieces are next to player one and the black pieces are next to player two.
 - Rows and columns are represented by characters from “A” to “H” on top and bottom of the board and numbers from “1” to “8” on the left and right of the board.
 - A proper message is displayed in necessary situations like an invalid input or a checkmate...etc.
- **Colors:** The displayed characters on the screen are colored.
- **Promotion.**
- **Game is won by checkmate.**
- **Game ends in a draw in case of stalemate.**
- **Undo moves.**
- **Redo moves.**
- **Save game.**
- **Load last game.**

Overview of game design

Player one and player two keep exchanging turns until the game ends in either checkmate or stalemate.

An input from the user is tested whether it is valid or not, if it is valid then it is executed, if it's not valid a proper message is displayed accordingly.

A move is checked whether it is right or not according to the rules of chess, if so it is executed.

Examples:

- A piece can be moved only in the range it should move in.
- A piece can't move to a place if there is another piece in the way except for the knight.
- A piece can't move if it is protecting the king from a check.

Any move is executed in a virtual game board then checked if it results in check on the player who made that move and if so, it is refused.

When a piece is moved to a certain square, this square is checked if it has a piece that belongs to the other player(so it is added to the removed pieces next to the board), or if it has a piece of the same player(the move is refused then), or if it is empty(the move is executed).

After every move the game checks if one of the two players is checked/checkmated/stalemated and displays a message accordingly.

The game is saved in a text file if the user asks to.

When a game is loaded, the player who was to play before saving is the player that has the first turn after loading.

Moves are saved so that players can undo/redo their moves.

The screen is cleared after every turn to give the feeling of a dynamically changing game board.

Used Data Structures

Defines:

ROW 8, COL 8 : Represents the length of the row and column of the board.

MAX_PIECES 30 : Represents the maximum number of removed pieces.

Global variables:

char removedPieces[MAX_PIECES] : Represents the removed pieces of the 2 players.

char playerWhoSaved: Represents which player saved the game so that he can continue his turn when the game is loaded.

virtualBoard[ROW][COL] : Represents a virtual board where a move is executed and checked whether it is valid or not.

int indexRP : Represents the length of the removedPieces array and it's index.

int flagx : Represents a flag used in saving the removed pieces.

int pointer; Represents a pointer to the move that is to be undo-ed or redo-ed.

Int maxMoves : Represents the number of moves until the last possible redo.

char moves[][] : An array where the done moves are saved for undo/redo.

Int firstMoveK1,firstMoveR1,firstMoveR12,firstMoveR2,firstMoveR22,firstMoveK2 : flags for the first moves of kings and rooks (needed for castling).

Int numMovesK1,numMovesK2 : number of moves of each king since the game started.

Local variables:

char board[ROW][COL] : The main board of the game which has the positions of the pieces.

int i,j,k,l,rpCounter : Counters for looping.

int input[] : An array where a valid move is stored for execution.

int flag, flag2,flag3,flag4 : Flags for exiting a loop when a condition occurs.

char handle[] : The user input is stored here for checking it.

FILE * save : A pointer to the file where the game is saved in.

FILE * load : A pointer to the file where the game is loaded from.

Functions used

printBoard : Prints the board, pieces on it and removed pieces.

initialize_board : Gives the board its initial value (The normal distribution of pieces in the start of a game).

player1 & **player2** : Takes a move from a player and checks if it is valid or not then executes the move if it is valid, also responsible for promotion and displaying proper messages.

checkLastPlaceP1 & **checkLastPlaceP2** : If a move is valid this function checks if it results in removing a piece that belong to the enemy or if the last square contains a piece that belongs to the same player or if it is an empty square.

checkBishopMoveP1 & **checkBishopMoveP2** : Checks if a bishop is moved by a player to a valid square.

checkRookMoveP1 & **checkRookMoveP2** : Checks if a rook is moved by a player to a valid square.

SetColor : Sets the color of printing.

saveGame : Saves the game in a text file.

loadGame : Loads the last game from the text file.

isP1Checked & **isP2Checked** : Checks if a player is checked by searching for the king in the range of the pieces of the enemy.

equateArrays : Equates the “board” array and the “virtualBoard” array so that a move can be tested in the virtual board.

canP1Move & **canP2Move** : Checks if there is a valid movement for a player thus determining if the game ended or not.

executeInVirtBoard : Executed a valid move in the virtual board.

canBishopP1Move & **canBishopP2Move** : Checks if the movement of a bishop results in check on the same player(i.e if the bishop was protecting the king from check).

canRookP1Move & **canRookP2Move** : Checks if the movement of a rook results in check on the same player(i.e if the rook was protecting the king from check).

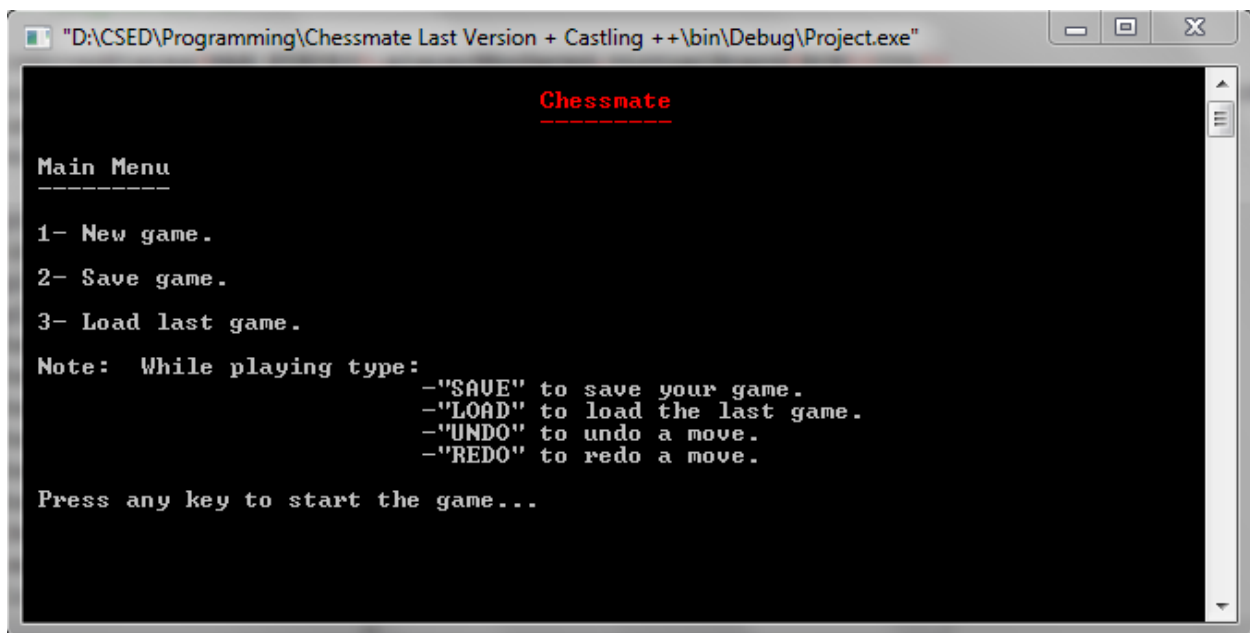
saveMoves : Saves moves for undo/redo.

References

The “SetColor” function is from the following link:

<http://stackoverflow.com/questions/29574849/how-to-change-text-color-and-console-color-in-codeblocks>

Sample Runs



```
"D:\CSED\Programming\Chessmate Last Version + Castling ++\bin\Debug\Project.exe"

Chessmate
-----

Main Menu
-----

1- New game.
2- Save game.
3- Load last game.

Note: While playing type:
      -"SAVE" to save your game.
      -"LOAD" to load the last game.
      -"UNDO" to undo a move.
      -"REDO" to redo a move.

Press any key to start the game...
```

```

"D:\CSED\Programming\Chessmate Last Version + Castling ++\bin\Debug\Project.exe"

  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
Player 1 turn, enter your move:

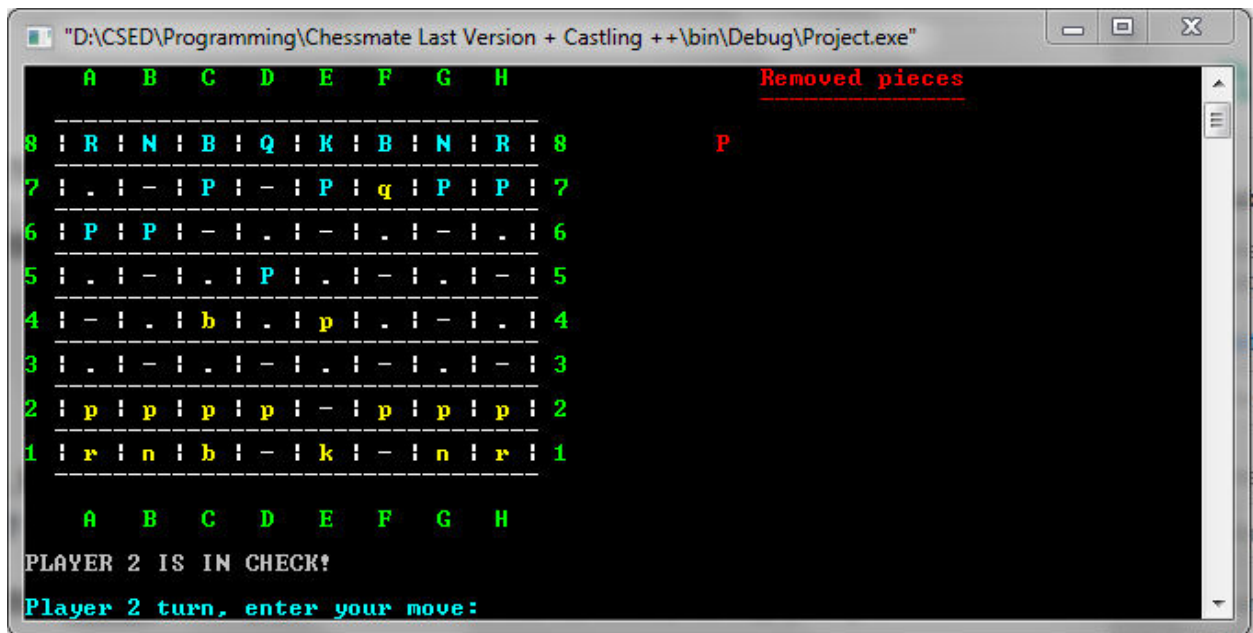
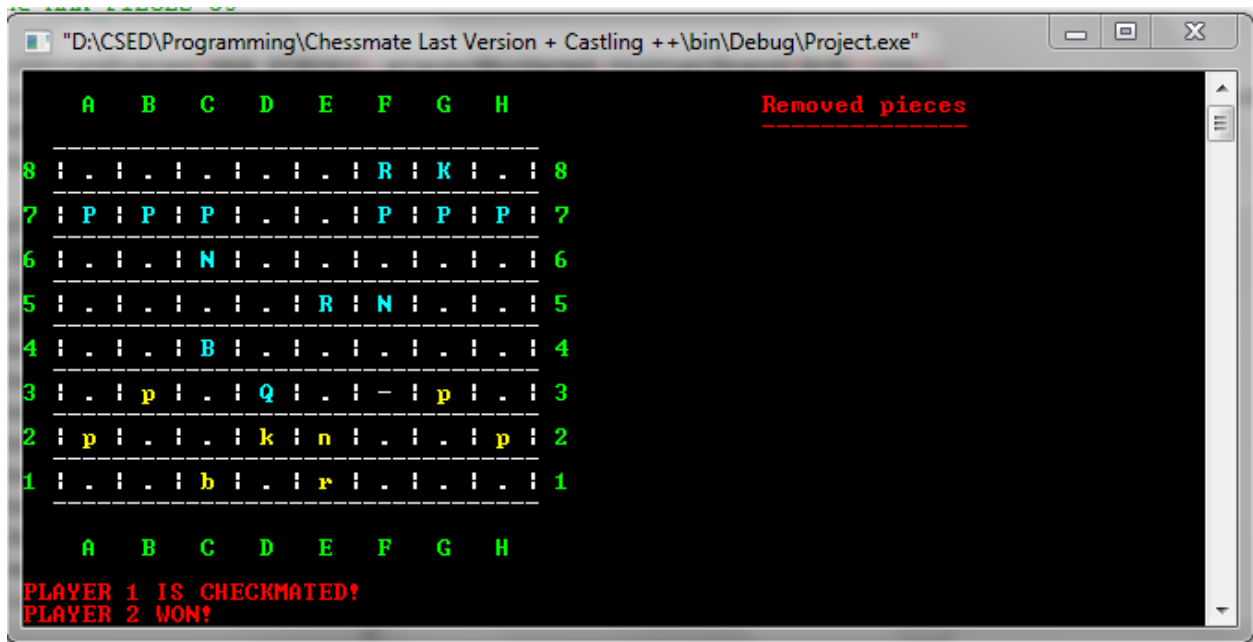
```

```

"D:\CSED\Programming\Chessmate Last Version + Castling ++\bin\Debug\Project.exe"

  A  B  C  D  E  F  G  H
-----
8 | R | N | B | Q | K | B | - | R | 8
7 | . | P | P | P | P | P | P | P | 7
6 | - | . | - | . | - | N | - | . | 6
5 | P | - | . | - | . | - | b | - | 5
4 | - | . | - | p | - | . | - | . | 4
3 | . | - | . | - | . | - | . | - | 3
2 | p | p | p | . | p | p | p | p | 2
1 | r | n | . | q | k | b | n | r | 1
-----
  A  B  C  D  E  F  G  H
Player 1 turn, enter your move:

```



```

"D:\CSED\Programming\Chessmate Last Version + Castling ++\bin\Debug\Project.exe"

  A  B  C  D  E  F  G  H
8 | - | . | - | . | - | B | N | R | 8
7 | . | - | . | - | P | - | P | Q | 7
6 | - | . | - | . | q | P | K | R | 6
5 | . | - | . | - | . | - | . | P | 5
4 | - | . | p | . | - | . | - | p | 4
3 | . | - | . | - | . | - | . | - | 3
2 | p | p | - | p | p | p | p | . | 2
1 | r | n | b | - | k | b | n | r | 1

  A  B  C  D  E  F  G  H
DRAW: THE GAME ENDED IN STALEMATE!

Removed pieces
P  P  P  P  N
B

```

```

"D:\CSED\Programming\Chessmate Last Version + Castling ++\bin\Debug\Project.exe"

  A  B  C  D  E  F  G  H
8 | R | N | B | Q | K | B | q | . | 8
7 | P | P | P | - | . | - | . | - | 7
6 | - | . | - | . | - | . | - | . | 6
5 | . | - | . | - | . | - | b | - | 5
4 | - | . | - | . | - | . | - | . | 4
3 | . | - | . | - | . | - | . | - | 3
2 | p | . | - | . | p | p | p | p | 2
1 | r | Q | . | q | k | b | n | r | 1

  A  B  C  D  E  F  G  H
Player 1 turn, enter your move: D1D2
Invalid move, enter a valid move:

Removed pieces
P  P  P  R  P
N
p  n  p

```


User Manual

Movement : Each square is indexed by a character and a number (e.g. D1,C3,...etc.). 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).

Promotion : 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 the pawn to a bishop).

Save : For a player to save, he should enter the word "SAVE" (capital letters).

Load : For a player to load, he should enter the word "LOAD" (capital letters).

Undo : For a player to undo, he should enter the word "UNDO" (capital letters).

Redo : For a player to redo, he should enter the word "REDO" (capital letters).

Castling : Move your king two squares to the right of the left (with the rules of castling present).

e.g. for player one E1G1 or E1C1 and for player two E8G8 or E8C8.

Invalid moves : In case that a player enters an invalid move(for any reason) a message in red is displayed saying that the move is invalid and asks for a valid move.

Pseudo code

```
While game=1 //the main loop until the game ends
{
    Print board // function save the board in array 2D
    While flag =1 {
        //ends if the player's turn ends
        Read the player move //input
        If input="SAVE"
            Save the board(2D array of pieces) and removed pieces
            in file

        Else if input ="LOAD"
    {
        if found a file saved before
            Load file from folder
        Else {
            print there isn't game saved
        }
    }
}
```

```
    Else if input="UNDO"
    {
        If this is the initial board
            Print no more undo to make
        Else undo to the previous move
    // all moves are saved in array 2D to redo and undo when needed
    }

    Else if input="REDO"
    {
        if this is the final move
            Print no redo to make
        Else redo to the next move
    }

Else if input[0]={A:H} &&input[1]={1:8}&& input[2]={A:H}&&
input[3]={1:8}
{
    If the piece is pawn
    {
        if this is the first move of pawn AND
        (input[0]==input[2]AND(input[1]-input[3])==2)AND the last place is
        empty //move 2square first move
        Flag=0
```

```
else if input[0]=input[2] AND input[1]-input[3]=1
{
    if input[3] =1 OR 3 //promotion
        Make promotion to input[4]
        Flag=0
    Else flag=0 //normal move
}

Else If (input[1]-input[3]=1 AND abs(input[2]-input[0])=1
AND last place contain enemy piece)
{
    if input[3] =1 OR 3 //eat and promotion
    {
        Make promotion to input[4]
        Flag=0
        Save the enemy piece in removed pieces
    }

    Else //eat only
    {
        Flag=0
        Save the enemy piece in removed pieces
    }
```

```
    }  
Else print invalid move  
}  
  
Else If the piece is bishop  
{  
    If abs(input[0]-input[2])=abs(input[3]-input[1]) AND  
    input[0]!=input[2]  
    {  
        //if bishop can move  
        If last place contain enemy piece  
            Save the enemy piece in removed  
pieces  
        //loops that check if there is any piece in bishop way  
        Flag=0  
  
        Else print invalid  
    }  
Else print invalid move  
}
```

Else If the piece is rook

```
{    If (input[0]-input[2]=0 AND abs(input[3]-input[1])>0) OR (input[3]-
    input[1]=0 AND abs(input[0]-input[2])>0)
    {    if rook can move
        {    If last place contain enemy piece
            Save the enemy piece in removed pieces
        }
    }
    //loops that check if there is any piece in bishop way

    Flag=0
}
Else print invalid move
}
```

Else If the piece is queen

```
{    If abs(input[0]-input[2])=abs(input[3]-input[1]) AND
    input[0]!=input[2]
    {    If queen can move
    }
    //loops check the way of queen
    {
        If last place contain enemy piece
    }
```

Save the enemy piece in removed pieces

//loops that check if there is any piece in bishop way

Flag=0

}

Else print invalid

}

//both the movements of the bishop and rook

else if(input[0]-input[2]=0 AND abs(input[3]-input[1])>0) OR
(input[3]-input[1]=0 AND abs(input[0]-input[2])>0)

{

If queen can move

//loops that check if there is any piece in queen way

{

If last place contain enemy piece

Save the enemy piece in removed pieces

Flag=0

}

Else print invalid move

```
}
```

```
//movements of the bishop and rook
```

```
}
```

Else If the piece is knight

```
{   if (abs(input[0]-input[2])=2 AND abs(input[3]-input[1])=1) OR  
    (abs(input[0]-input[2])=1 AND abs(input[3]-input[1])=2)
```

```
        If last place contain enemy piece
```

```
        Save the enemy piece in removed pieces
```

```
        Flag=0
```

```
    Else print invalid
```

```
}
```

Else If the piece is king

```
{   if (abs(input[0]-input[2])=1 AND abs(input[3]-input[1])<2) OR  
    (abs(input[3]-input[1])=1 AND abs(input[0]-input[2])<2)
```

```
        If last place contain enemy piece
```

```
        Save the enemy piece in removed pieces
```



```
    Flag=0
}
Else print invalid move //there is no piece in the square to be moved

If the king checked
{    Print invalid move //the move cases checked on king
    Flag=1
}
//we save all moves in array to make redo and undo
} // end the loop of player and changed to another player flag=0
//else of flag =0 check if the game ended or not
```

```
Else {
    If the other player is checked AND can't move
    // execute all moves in virtual board to know check or can move
        {game =0
        Print player win
        }
    Else If the other player isn't checked AND can't move
```

```
// // execute all moves in virtual board to know check or can move
```

```
    {game =0
```

```
    Print draw (stalemate)
```

```
    }
```

```
Else If the other player is checked only
```

```
    {
```

```
    Game =1
```

```
    Flag=1
```

```
    Print the other player is in check
```

```
    //turn of the other player
```

```
    }
```

```
Else
```

```
{game =1
```

```
Flag=1
```

```
}
```

```
}
```

```
//end of loop of game and the game ends
```

Flow Chart

