Chess Project

CSED 2024

AMR MAGDY MAHMOUD HANAFY MAHMOUD 19016116

YOUSEF SABER SAEED MOHAMED MOHAMED 19016924

Description:-

Chess is a two-player board game played on a chessboard, a checkered game board (usually black and white) with 64 squares arranged in an eight-by-eight grid. Each player begins the game with 16 pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns. Each of the six piece types moves differently. The most powerful piece is the queen and the least powerful piece is the pawn. The objective is to checkmate the opponent's king by placing it under an inescapable threat of capture.

Features:-

We've designed a chess program that support all the basic chess rules :

moves rules, check, draw situations:[stalemate - dead positions -50 moves draw] , checkmate and other rules like: en passant, castling and promotion.

- supports saving and loading.

- supports undoing moves up-to the very first move in the game (this includes loaded games) & supports redoing as well.

- has a friendly user interface.

- supports mouse & keyboard controls.

- sound effects.

Design Overview:

The players are first greeted with the main menu where they can choose between starting a new game, loading a previous one or viewing information and learn how to play.

.The `start game` section asks the players to enter their names, making the game more user-friendly and appealing.

. In the 'load a game' section, the two players are welcomed back before continuing their loaded game just from the point where they left off.

Then the game window appears !

let's take a quick dive into the back-end of our game :

Every half-play "**turn**" is stored in an independent place in the memory with its properties :

the board at that turn, the captured pieces & other special properties like en-passant & castling …

Controlling the flow :

-using 64 bits unsigned integers was so helpful to deal with complex positions relations as every square of the board is represented by a digit in the 64 unsigned int.

-beside using 64 bit integers, 2D arrays are used to represent each turn's board.

Assumptions:-

1. The game is designed to satisfy all the requirements stated in the project pdf.

2. We use character arrays to store the players' names & if they decided to proceed without typing their names they will be called as player_One & player_Two.

3. They can't save more than one game at a time {one save slot} which just satisfies the requirements, however we're looking forward to making more versions with more save slots and more enhancements.

4. each one of the players can go back to main menu , so that might be infuriating for one player if the other has gone back to main-menu without saving in the middle of a running game.

5. Concerning this report "white player" means the player who controls the white pieces & the same thing for "black player".

Data structures:-

We're dealing with a respectively bigger amounts of data than we used to, so choosing to use structures was a great decision that made it easier to enable more features.

Imagine an **array of turns**; where every element of the array of index i has the following information:

- **2D array representing** the chess board at the i^{th} turn: [array of characters]

- **1D array representing** the captured pieces at the i^{th}turn:[array of characters]

Each piece's represented by a character (i.e: 'q' represents the white queen)

- **castling permissions** : 8-bit char (- - - - xxxx) :-

8421

The 4 least-most significant bits are used as flags (permissions) like this :

The bit (1) represents the white[small letters] short castling

The bit (2) represents the white[small letters] long castling

The bit (4) represents the black[capital letters] short castling

The bit (8) represents the black[capital letters] short castling

Using bit-wise operations: bit-wise AND(&), bit-wise OR(|) and bit shifting operations, it's possible to read, manipulate and change this **char** according to certain game actions (king & rook movements).

- **En-passant information** : stored in a defined as data type :

a **structure** storing

```
typedef struct {
    int on;
    int x;
    int y;
}enpassant;
```

1- a permission flag : int on;

2- The position of the pawn, which could possibly be en-passed in the following turn, represented by two integer values: co-ordinates (x,y)

- **The 50 moves draw counter** : integer counter that depends on game actions, when the game reaches 100 half-plays :: 50 plays, then it's a Draw.

- **cap_count** : integer holds the count of the captured pieces at the i^{th} turn.

- **turn** : is simply i (integer holds the turn number).

Defining a new data type : [game]

```
typedef struct{
    int turn;
    char board[8][8];
    enpassant enpas;
    int draw_50;
    char castle;
    int cap_count;
    char captured[30];
}game;
```

And .. declaring an array of up to 3000 games

```
game saved[3000];
```

This array to make it easier to enable (saving & loading , undoing & redoing).

2D integer arrays are also used side by side with the 64 bit unsigned integers to control the movements and provide solid algorithms to control the flow of the game.

Why use 64 bit integers while you can fully depend on arrays ?

-One single unsigned 64-bit integer (fills 64 bits in memory) can hold a full board information (i.e : threatened squares), on the other hand int arrays need way more room at least 64*8 bits.
-- less memory needed = more speedy & efficient --

-using bit-wise operations instead of nesting more loops and complicating the code.

How ?

0000000000000000000000000000000000000000000000000000000000000000

These are 64 zeros, but when we look at them like this :

0 1 2 3 4 5 6 7 #

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 2

0 0 0 0 0 0 0 0 3

0 0 0 0 0 0 0 0 4

0 0 0 X 0 0 0 0 5

0 0 0 0 0 0 0 0 6

0 0 0 0 0 0 0 0 7

so say we want to represent the square at (1,3) : row 1 & column 3 :

-it's represented by the 11th bit (1*8+3)

what if we want to represent the square at (3,1) row 3 & column 1 ?

-it's represented by the 25th bit (8*3+1)

so the (8*row+column)th bit represents the square at (row , column)

To set a single bit in 64-bits integer X to one or zero :

to 1 : shift one to that bit then bitwise OR [I.e : **X|=1<<(5*8+3)**]

to 0 : shift one to that bit using bitwise not , and:

[I.e : **X&=~(1<<(5*8+3))**]

SDL data:

Sdl structures & unions that were used :

-SDL_Surface :

A structure that contains a collection of pixels used in software blitting, used to store bitmaps of chess board and pieces to blit on the screen (for more info).

-SDL_Rect:

A structure that contains the definition of a rectangle, with the origin at the upper left(for more info).

-SDL_Event:

A union that contains structures for the different event types(for more info).

-SDL_MouseButtonEvent:

A structure that contains mouse button event information(for more info).

-SDL_KeyboardEvent:

A structure that contains keyboard button event information(for more info).

-SDL_AudioSpec:

A structure that contains the audio output format. It also contains a callback that is called when the audio device needs more data(for more info).

These are also used (to load & run sound effects):

-Uint32 ;

-Uint8 ;

-SDL_AudioDeviceID :

to store the audio device id returning from **SDL_OpenAudioDevice**

function (for more info).

Description of files and functions:-

Note:- the files have been written in the order of compiling :

Starting with **primitive** files (definitions & global variables & the basic functions) going all the way up to [**main.c**]:

1-definition.h

Contains the definitions and the global variables that need to be included in all the other headers.

Functions:

Has one function : **MsgBox(char *s)** which pops out a massage box to the screen with the characters it takes as an argument.

2-moves.h

Contains all movement-related functions :

Move_'character' : the function that calculates all the possible moves for the piece represented by that char.

Move_'integer' : the function that performs a single movement according to that integer

examples :

move_q : calculates the possible moves for the white queen. (Small= white).

move_K : calculates the possible moves for the black KING. (Capital = black).

move_3 : movement with a capture.

move_11 : pawn promotion .

Move_12 : pawn promotion with a capture.

---

get_possible_moves(int x, int y, game*s, int moves[8][8], int*can_move) ;

'The most called function in the program'

Takes :

-2 integers x,y (a position on the board) ,

- a pointer to a game structure,

- a pointer to a 2D integer array to fill it with integers representing the specific type of movement - as said before (I.e: 11 = promotion),

- a pointer to an integer to provide an answer if it has any possible moves or not.

Returns :

a 64 bit unsigned integer representing all the squares the piece can move to.

3-menu.h

Simply contains the functions needed to display the main menu.

4-head.h

We can say that head is the brain of the game engine that controls the flow of the game and applies its rules, mostly works on the 64 bit unsigned integers and it has the following functions :

Common parameters :

Most of the following functions take a pointer to a game & integer 'who' :

'who' is either [0] for the white player, or [non-zero usually 1] for the black player.

(1)- **attack_zone**(game *s,int who) **&** **modified_attack_zone**(game *s,int who): U.L.L int

Calculates and returns a 64-bit integer (unsigned) representing all the pieces the the player 'who' can move to /or capture.

(2) who_is_attacking(game*s,int a,int b,int who,int*howmany): U.L.L int

calculates and returns a 64-bit integer representing the positions of all the pieces of the opponent that can reach and capture the piece at (a,b) that should be belonging to 'who'… also puts their count in *howmany.

(3) where_is_my_king(game*s,int who,int*m,int*n): void

As you may have guessed, it loops searching for the king of 'who'

Putting the co-ordinates in (*m,*n).

(4) is_it_checked(game*s,int m,int n,int who): int

Is the piece at(m,n) that belongs to 'who' in danger of being captured the following move ? … returns **1** if yes, **0** otherwise.

(5) is_ok_to_move(game*s,int a,int b,int x,int y,int o,int p,int who): int

It's **who**'s turn, and the question is: "Is it ok to move his piece at (a,b) to (x,y) given that his king is at (o,p) ? "… returns **1** if yes, **0** otherwise.

I'd say it's one of the most powerful functions I have ever thought of (due to its utility) as it actually performs the move virtually and sees if the king would be in check or not.

(6) can_it_really(game*s,int who,unsigned long long u,int *a,int *b): int

As said before **get_possible_moves** can tell us if a piece can move or not but it has a missing puzzle :

What if the piece can move but all the moves it can perform would put the king in danger, (which means : they are invalid moves !).

"**can_it_really**" is really that missing part as it checks the validity of each possible move of the piece at (*a,*b) that belongs to 'who' with the help of the 64-bit integer u returning from **get_possible_moves !**

(7) perform_move(game *s,int who,int a,int b,int *x,int *y,int moves[8][8]): void

It performs the move for the player 'who' from (a,b) to(*x,*y) using the 2D integer array & the **move_'int'** that were mentioned before (included from moves.h).

(8) rook_defence(int i,int j,int y,int x,game* s,int who,int*dead):

It's who's turn and he is under an opponent's rook's check given that :

The threatening rook is at (i,j) & the (player "who")'s king is at (y,x):

The function looks if another who's piece can defend the king by blocking the way between them. If there's none : *dead = 1 , *dead = 0 otherwise.

bishop_defence has the same purpose, but for a threatening bishop.

*The rest of the functions in head.h are (take_input,prepare_the_game,…):

These are used in the **play()** function, so it's better to talk about them there.

5-over.h

Contains the [game ending] scenarios :

(1) how_is_my_king(game*s,int who,int* checked,int*wayout,int*dead): void

Designed to fill these three integer addressees :

-*checked= 1 if who's king is checked, 0 otherwise.

-*wayout = 1 if the king has any valid move, 0 otherwise.

-*dead = 1 if the king is checkmate, 0 other wise.

---

(2) **nolegalmove**(game *x,int who): int

Returns 1 if the player 'who' doesn't have any valid move and 0 otherwise.

(3)fiftymove(game *x): int

Returns 1 if it's a (50 moves draw) , 0 otherwise.

(4)noleftpiece(game *s): int

Returns 0 if it's a dead position draw, 1 otherwise.

(5)beSureKingIsDead(game *s,int checked,int who): int

Was introduced later in the game, as it was noticed that in addition to how_is_my_king(), nolegalmove() can also determine if the king was dead or not, so it was added to support the certainty of our game .

Note: it doesn't add any time complexity as nolegalmove() would have been called anyway.

6-save.h

Contains the saving & loading functions.

7-Draw.h

Contains the drawing on the SDL_Window functions (most of the SDL work is there).

8-play.h

Has the most main functions (After the SDL_main) :

(1) play(int turn,SDL_Surface* screenSurface,SDL_Window* window,char* t0,int* f1,int* f2,int* f3,int* f4) : int

Let's just -for now- worry about that first parameter (turn) :

The purpose of play() : take a turn .. play that turn and it's briefly goes like this :

//make a new game(a.k.a turn)

//prepare the turn (according to the previous one) (using **prepare_the_game** function)

//draw it on the screen.

//check the game-ending scenarios. (Stop here if ended)

//take input from the player

//perform the command the player gave you

The SDL_Surface & SDL_Window pointers are for drawing matters.

The last 5 pointers are to store the performed move (the piece as a character , 4 integers represent the co-ordinates of the (From&To squares) to display it on the console for the opponent in the following turn.

Play() Returns an integer depending on the command of the player :

- 0 if the game ended;
- The returned integer from take_input();
  - unless it's a (-1)and in this case: [perform the move & return 5]

Take_input() :

- returns 2 if the player command was to undo.
- returns 3 if it was to redo.
- returns 6 if it was to go back to main menu.
- returns -1 if he commanded a valid move.

(2) gameplay(SDL_Surface*screenSurface,int turnn,SDL_Window*window) : void

Has the game loop .. doesn't stop playing turns Sequentially, until one of the players chose to go back to the main menu.

Note (If the game ended the players still can see through the game to analyze it or undo & change middle game actions).

9-Main.c

1- Start the main menu to either start a new game or load a previous one.

2- Initialize the sdl window.

3- Enter the game loop starting with either turn = 0 (if they chose a new game) or a loaded turn.

4- destroy the sdl window, free the memory and end the program.

Flowcharts and pseudo-code :

1 Who is attacking function: to get the positions of all the threatening pieces:

```
Get input game s and a and b and howmany
 U=0 and can =1 trac array 8*8 =0 and i=0 and j=0
If  who=0
    For I from 0 to 8
        For j from 0 to 8
          If sboard[i][j] is small letter
               D= "get_possible_move(i,j,js,trash,can) function
                     If( D bitwise and 2^(8*a+b)=0)
                         u=u bitwise or   2^(8*a+b)
                         how many=how many +1
          else
For I from 0 to 8
        For j from 0 to 8
          If sboard[i][j] is capital letter
               D= "get_possible_move(i,j,js,trash,can)function
                     If( D bitwise and 2^(8*a+b)=0)
                         u=u bitwise or   2^(8*a+b)
                         how many=how many +1
return u
End
```

The flow chart:

https://drive.google.com/drive/u/0/folders/1q7LpBXZ8L5NF5UWGlXkmCgovNrlXachn

2_ attack zone is function to get all places that under attack from opposite player

```
Get  input game s and who
  u=0 d=0 tracch array 8*8
      If who=0
                  For i from 0 to 8
                       For j from 0 to 8
                             If sboard[i][j] is small
                                  d=“get_possible_move(i,j,js,trash,can)”
function
                                  u=u bitwiseor d
      Else
                  For i from 0 to 8
                       For j from 0 to 8
                             If s board[i][j] is small
                                  d=“get_possible_move(i,j,js,trash,can)”
function
                                  u=u bitwiseor d
return u
End
```

The flow chart:

https://drive.google.com/drive/u/0/folders/1y1hY2K4lqci-JGTGvZB21EXuZtQVdXaF

3_ is_it_ok_to_move function to know if piece can move from position to other position and king not in danger:

```
   Get input position a and b destination x y and king position o and p and
   Get game s
  Game new_s, m=o,n=p
       For i from 0 to 8
          For j from 0 to 8
            New_s board[i][j]=s board[i][j]
   Move[8][8]=0,can=1
  Z= “get_possible_move(a,b,j,s,moves,can)” function
     If move[x][y]=8
          If who =0.
             new_s board[x][y]= new_s board[a][b]
             new_s board[a][b]= colorboard[a][b]
```

```
                new_s board[x+1][y]= colorboard[a][b]
            Else
                new_s board[x][y]= new_s board[a][b]
                new_s board[a][b]= colorboard[a][b]
                new_s board[x-1][y]= colorboard[a][b]

        Else If

                new_s board[x][y]= new_s board[a][b]
                new_s board[x][y]= colorboard[a][b]
      Else
             new_s board[x][y]= new_s board[a][b]
              new_s board[x][y]= colorboard[a][b]
  u=0
  If who =0
      u=”attack_zone(new_s,1)” function
  Else
       u=”attack_zone(new_s,0)” function

If (2^(8*m+n)bitwise and u=0)

   Return 1

   End

Else

   Return 0

   End
```

The flow chart:
https://drive.google.com/drive/u/0/folders/1y_8J8S64Ag9Fj1HX21IZazGmrr-aa96T

4_is_it_checked is function return 1 if king check and 0 if not

```
Get game s and king postion m and n and who

u=0

If who=0
```

```
   u=” modified_attack_zone(s,1)” function
Else
   u=” modified_attack_zone(s,0)” function
If 2^(m*8+n)bitwise and u=0
   Return 1
   End
Else
  Return 0
  End
```

The flow chart:
https://drive.google.com/drive/u/0/folders/135ZVPcG29HZUqIve1U8yrOBaIY9sMAdp

5_how_is_my_king in that function we check if king checked or not and if king dead

```
Get game s,who,checked,way out,dead
If who=0
   For i from 0 to 8
      For j from 0 to 8
        If sboard[i][j]=k
            y=I,j=x
Else
   For i from 0 to 8
      For j from 0 to 8
        If sboard[i][j]=K
```

```
            y=I,j=x
u=”attack_zone(s,who-1)”
If (2^(8*y+x)bitwise and u=0)
    Checked=0
      Dead=0
Else
  Checked=1
For xDir from -1 to 2
  For yDir from -1 to 2

   If ((0<=x+xDir<=7)and(0<=y+yDir<=7)and(xDir !=0 and y!=0))
       If (sboad[y+yDir][x+xDir]=’ - ‘ or sboad[y+yDir][x+xDir]=’ .’ )
          If(“is_ok_to_move(s,y,x,yDir+y,xDir+x,yDir+y,xDir+x,who)” function=1)
             Way out =1 dead=0
       Else
         If who=0
          If sboard[y+yDir][x+xDir]is small letter
            If(“is_ok_to_move(s,y,x,yDir+y,xDir+x,yDir+y,xDir+x,who)” function=1)
              Way out =1 dead=0
         Else
          If sboard[y+yDir][x+xDir] is capital letter
            If(“is_ok_to_move(s,y,x,yDir+y,xDir+x,yDir+y,xDir+x,who)” function=1)
              Way out =1 dead=0
  If way out =1
       Break
```

```
If  (way out=0 and checked=1)
  Dead=1,count =0
  q =  “who_is_attacking(s,y,x,who-1,count)” function
  For i from 0 to 8
   For j from 0 to 8
    if (2^(8*i+j)bitwise and q !=0)
      if(2^(i*8+j)bitwise and  “attack_zone(s,who)” function)!=0)
       howmany=0, w=” who_is_attacking(s,i,j,who,howmany)”  function
       for a from 0 to 8
          for b from 0 to 8
              If(2(8*a+b)bitwise and w!=0)
                If(sboard[a][b]= k or sboard[a][b]= K)
                  If(“is_ok_to_move(s,a,b,i,j,i,j,who)” function=1)
                      dead=0
                      End
              Else
                   If(“is_ok_to_move(s,a,b,i,j,i,j,who)” function=1)
                      dead=0
                      End
      If(sboard[i][j]=='N' or sboard[i][j]=='n')
                        dead=1;
                        End
      If(sboard[i][j]=='P' or sboard[i][j]=='p')
                        dead=1;
                        End
```

```
      If(sboard[i][j]=='R' or sboard[i][j]=='r')
                          “rook_defence(i,j,y,x,s,who,dead);” function
      If(sboard[i][j]=='B' or sboard[i][j]=='b')
                          “bishop_defence(i,j,y,x,s,who,dead);” function
      If(sboard[i][j]=='Q' or sboard[i][j]=='q')
                          “rook_defence(i,j,y,x,s,who,dead);” function
                           If(dead=0)
                                    End
                          “bishop_defence(i,j,y,x,s,who,dead);” function
```

The flow chart:
https://drive.google.com/drive/u/0/folders/1LTUfUy5NHNfPCCPCrzGuufkRuKGYJauR

6-beSureKingIsDead function to make us sure king is dead or stale mate

```
Get game s and checked and who
     D=” nolegalmove(s,who)” function
     If(d=1)
          If(checked=1)
               Play check mate audio
                    Print winner name
                  Return 0
                    End
          Else
                  Play stalemate audio
```

```
                Print winner name
               Return 0
                 End
    Else
            Return 120
            End
```

The flow chart:

https://drive.google.com/drive/u/0/folders/1-3E-vEK87--ueutpsk75XAKDCKRPNRVs

7-inherit game prepare game from previous game

```
Get game s1 and game s2
S2_turn=S1_turn+1
For i from 0 to 8
   For j from 0 to 8
      S2_board[i][j]=s1_board[i][j]
S2_castel=s1_castel
S2.enpas.on=0
S2.enpas.x=0
S2.enpas.y=0
S2_draw_50=s1_draw_50+1
For i from  0 to 8
     S2.capture[i]=s1.capture[i]
S2.cap_count=s1.cap_count
```

The flow chart:
https://drive.google.com/drive/u/0/folders/1C6DQeCb5e8HGDY9yV_2W_DMmmYHGY9Zp

8-can it realy

```
Get game s and who and u and a and b

M=0,n=0

 “Where is my king (s,who,m,n)” function

For i from 0 to 8

        For j from 0 to 8

             If(2^(8*i+j)bitwiseand u !=0)

                    If (“is_ok_to_move(s,a,b,i,j,m,n,who)” function=1)

                           Return 0

                          End

Return 1

End
```

The flow chart:
https://drive.google.com/drive/u/0/folders/1haeEYL-Jbgj-qIqlpdtkvOBOqHcWQ-Gl

9-no legal move

check if player has moves or not

```
Get game x who

If who =0

   For k from 0 to 8

        For j from 0 to 8

           Can=1

             if x.board[k][j] is small letter
```

```
                    u=” get_possible_moves(k,j,x,moves,can)”  function
                        if can=0
                                If(  “can_it_really(x,who,u, k, j)”  function =0)
                                    Return 0
                            End
Else
   For k from 0 to 8
         For j from 0 to 8
             Can=1
               if  x.board[k][j] is small letter
                    u=” get_possible_moves(k,j,x,moves,can)”  functon
                        if can=0
                                If(  “can_it_really(x,who,u, k, j)”  function =0)
                                    Return 0
                            End
Return 1
End
```

The flow chart:

https://drive.google.com/drive/u/0/folders/1hPAQzPWi6ILzZZ__omkdxaAAR-H3409Z

10-fiftymove that function check rule of fifty move that if there have been no captures or pawn moves in the last fifty moves if return 1

```
Get game x;
If x.draw_50=100
      Return 1
```

```
        End
Return 0
End
```

The flow chart:

https://drive.google.com/drive/u/0/folders/1bHxFvLC9jvrnv4yQKfnmjWHTzCJfy06Y

11-noleftpiece check if game end with dead position draw if return 0

```
Get input game x
Wht=-1,bck=-1,bownw=0,knight=0,bownb=0,knight=0
if x.cap_count=30
    print draw
        return 0
          End
Else if x.cap_count>=28
   For k from 0 to 8
        For j from 0 to 8
            If s.board[k][j] is small letter
                wht=wht+1
               If(s.board[k][j] =' n' )
                 knightw=knightw+1
               If(s.board[k][j] =' b' )
                          bownw=bownw+1
           Else s.board[k][j] is small letter
                bck=bck+1
               If(s.board[k][j] =' N' )
```

```
                    knightb=knightb+1
                If(s.board[k][j] =' B' )
                            bownb=bownb+1
If( wht=bck)
   If( knightw=knightb or knightb=bownw or bownb=knightw or bownb=bownw)
         print draw
      return 0
       End
Else If (wht=1 and bck=0)
       If (bownw=1 or knightw =1)
              If( knightw=knightb or knightb=bownw or bownb=knightw or bownb=bownw)
                                    print draw
                                   return 0
                                    End
Else If (wht=0 and bck=1)
       If (bownb=1 or knightb =1)
              If( knightw=knightb or knightb=bownw or bownb=knightw or bownb=bownw)
                                    print draw
                                   return 0
                                    End
Else
                                   return 1
                                    End
```

The flow chart:

https://drive.google.com/drive/u/0/folders/1KzemrNRwsHnKC5ZXFjursGufZbCbcNdF

12-play is function to play one turn and return if game ends or undo or redo & return value to game play menu

```
Get turn ,screen surface ,window,t0,f1,f2,f3,f4

Game x

 “prepare_the_game(turn,x)” function

If(turn!=0)

     If(turn%2=0)

            Print last played player 2 name

     Else

            Print last played player 1 name

     Print last move

If(turn%2=0)

              Print player 1 name have turn

Else

              Print last played player 2 name

checked=0,wayOut=0,dead=1

 “how_is_my_king(x,turn%2,checked,wayOut,dead)” function

If dead=1

    If   “beSureKingIsDead(x,checked,turn%2)”  function=0

                Return 0

                End

Else If   “beSureKingIsDead(x,checked,turn%2)”  function=0
```

```
                Return 0
                End
Else If  “noleftpiece(x)” function=0
                Return 0
                End
Else If  “50move(x)” function=1
                Return 0
                End




If check=1
        play
        check sound
         print checked
a=0,b=0,k=0,i=0,moves[8][8]
g=”take_input(x,moves,a,b,k,l,turn%2,window)” function

If g!=-1
         Return g
         End
t0=x.board[a][b],f1=a ,f2=b, f3=k f4=i

“perform_move(x,turn%2,a,b,k,l,moves)”function
play
move sound
“save_into_array(x,turn)” function
max_turn = turn
return 5
```

End

The flow chart:

https://drive.google.com/drive/u/0/folders/1RGSVF8WBAOoEwAYF-2G-VELltGTeJp8J

13-game play is most important function in game it has the main game loops called in the sdl main

```
Get screen surface ,turn,window

State ,turn=turn

While true

      m=0,n=0,t0=’  ’ ,f1=0,f2=0,f3=0,f4=0

     state=” play(turn,screenSurface,window,t0,f1,f2,f3,f4)” function

    if state=2

       turn= turn-1

    Else if state=3

       turn= turn+1

    Else If state=6

      “end_window”  function

       “Main_menu” function

      SDL_Surface screenSurface;

          Return 0

          End

    Else

        If state=0

         Get input key board SDL_EVENT
```

```
Switch(input  key)
  Case( key down)
          If key u
              Turn=turn-1
          Else If key s or key y
                   “save()” function
                   “End_window()” function
                   Turn=0
                   “ Main_menu(turn)” function
                   SDL_Surface screen surface
                   “gameplay(screenSurface,turn,window)” function
                  End
  Case KEY Up
            print key relase
                 break;
Case Click out
          Return 6
            End
 Default
          Break
```

The flow chart:

https://drive.google.com/drive/u/0/folders/1DypWbeJFN9xhsgG3WsjrJwLJD1LXItBy

14- SDL_Main

Turn=0

“main_menu(turn)” function

SDL_WINDOW x=NULL

“init_window(x)” function

SDL_Surface screenSurface = NULL

“gameplay(screenSurface,turn,x)” function

“End_window()” function

Out put 0

END

The flow chart:

https://drive.google.com/drive/u/0/folders/1r9Q2Ulw_18EeyW1wDaBk2W5nNqRkCMtI

User Manual

In menu :

Choose either to start a new game, a load a previous game or view info.
// by typing in ‘1’ , ‘2’ or ‘3’ then pressing enter

Mid-game Controls:

To move the pieces :

Left click on the piece you want to move then left click on the distination

---

To undo:

Press the key ‘u’ on your keyboard (make sure the game window is activated)

To redo:

Press the key ’r’ on your keyboard (make sure the game window is activated)

To save:

Press the key ‘s’ on your keyboard (make sure the game window is activated)

To go back to main menu”

Press the key ‘y’ on your keyboard (make sure the game window is activated)

In pawn promotion:

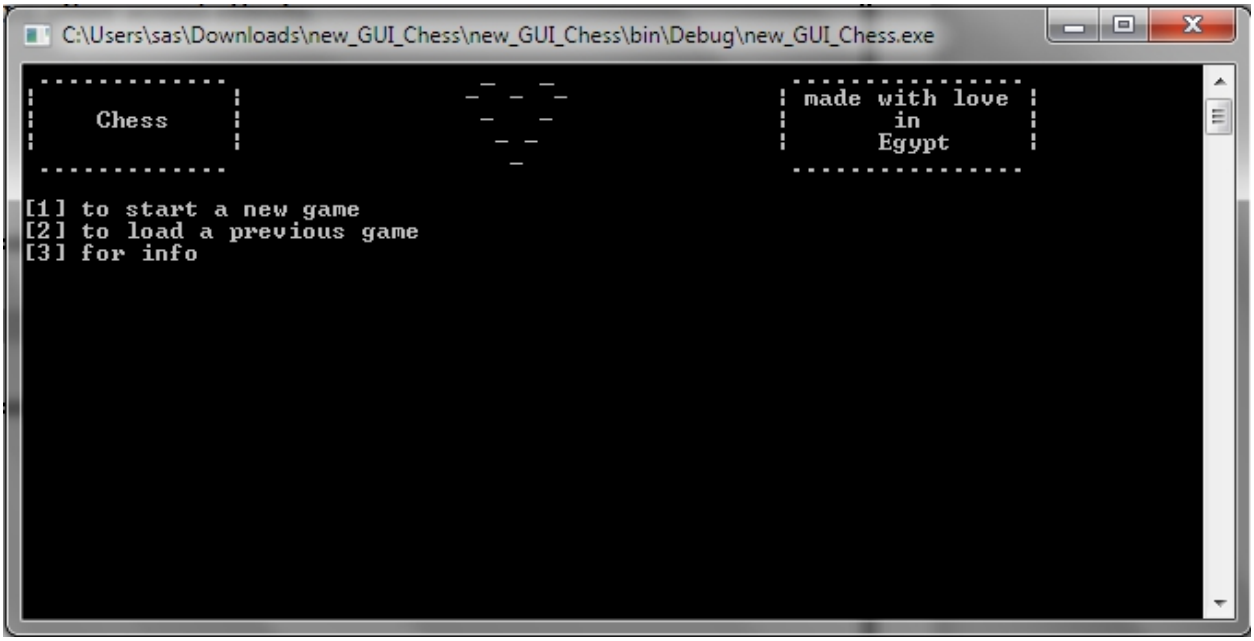
Left click on the piece you wanna promote to.

Sound effects will guide you during the game :

-invalid attempts play a certain sound effect.
-main game actions have other sound effects.

Test cases

I-The menu




made by youssef saber & amr magdy .. Have fun ❤
turn :
u to undo
r to redo
s to save
y to go back to main menu

En-passant:


made by youssef saber & amr magdy .. Have fun ❤
turn : 5
u to undo
r to redo
s to save
y to go back to main menu


made by youssef saber & amr magdy .. Have fun ❤
turn : 6
u to undo
r to redo
s to save
y to go back to main menu


made by youssef saber & amr magdy .. Have fun ❤
turn : 7
u to undo
r to redo
s to save
y to go back to main menu

Castling:


made by youssef saber & amr magdy .. Have fun ❤
turn : 15
u to undo
r to redo
s to save
y to go back to main menu


made by youssef saber & amr magdy .. Have fun ❤
turn : 16
u to undo
r to redo
s to save
y to go back to main menu


made by youssef saber & amr magdy .. Have fun ❤
turn : 17
u to undo
r to redo
s to save
y to go back to main menu

Promotion:


made by youssef saber & amr magdy .. Have fun ❤
turn : 2 1
u to undo
r to redo
s to save
y to go back to main menu


promotion
click on one of these
made by youssef saber & amr magdy .. Have fun ❤
turn : 2 1
u to undo
r to redo
s to save
y to go back to main menu


made by youssef saber & amr magdy .. Have fun ❤
turn : 2 2
u to undo
r to redo
s to save
y to go back to main menu

Check after the promotion:

Checkmate :

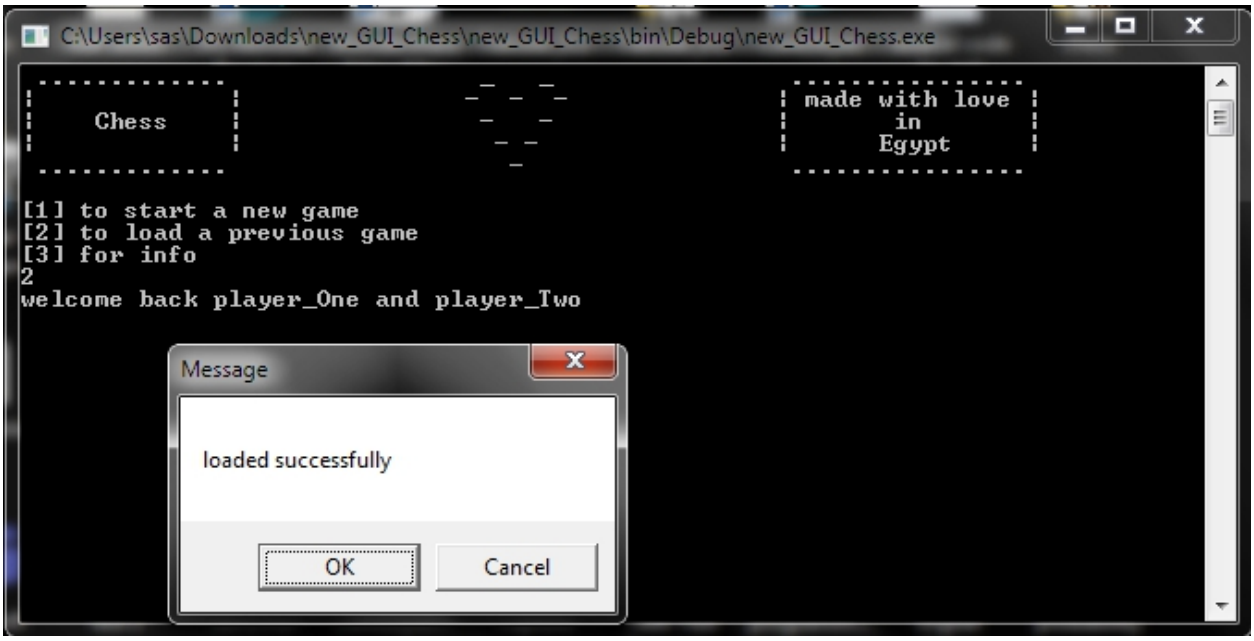

made by youssef saber & amr magdy .. Have fun ❤
turn : 4
u to undo
r to redo
s to save
y to go back to main menu


Message
checkmate !
OK
Cancel
made by youssef saber & amr magdy .. Have fun ❤
turn : 5
u to undo
r to redo
s to save
y to go back to main menu

Draw (Stalemate):


Message
DRAW !
OK
Cancel
made by youssef saber & amr magdy .. Have fun ❤
turn : 2
u to undo
r to redo
s to save
y to go back to main menu

loading:


C:\Users\sas\Downloads\new_GUI_Chess\new_GUI_Chess\bin\Debug\new_GUI_Chess.exe
Chess
made with love in Egypt
[1] to start a new game
[2] to load a previous game
[3] for info
2
welcome back player_One and player_Two
Message
loaded successfully
OK
Cancel

The loaded game(the same previous draw game):

Refrences

- https://wiki.libsdl.org/
- http://stackoverflow.com/
- https://www.geeksforgeeks.org/
- https://gigi.nullneuron.net/gigilabs/