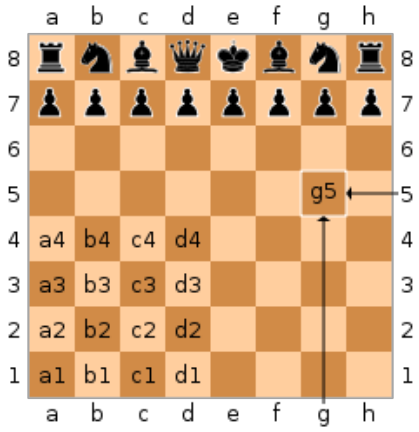
# DOCUMENTATION

Chess is a two-player strategy board game played on a chessboard, a checkered game board 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
- Eight pawns.

These pieces are used to attack and capture the opponent's pieces, with the objective to checkmate the opponent's king by placing it under an inescapable threat of capture. In addition to checkmate, the game can be won by the voluntary resignation of the opponent, which typically occurs when too much material is lost, or if checkmate appears unavoidable. A game may also result in a draw in several ways, where neither player wins.

Chess is played on a square board of eight rows (called *ranks* and denoted with numbers *1* to *8*) and eight columns (called *files* and denoted with letters '*a*' to '*h*') of squares. The colors of the 64 squares alternate and are referred to as "light" and "dark" squares. The chessboard is placed with a light square at the right-hand end of the rank nearest to each player, and the queen sits on a square of its own color at the start of the game. Each square has a name, as depicted.

# MINIMUM SYSTEM REQUIRMENT

- Intel Pentium IV
- 1 Gb Ram
- 5 MB Hard disk Space
- WINDOWS XP
- DEV C++ 4.90 or Codebocks 11.0

# HEADER FILES USED

- IOSTREAM.H
- CONIO.H
- FSTREAM.H
- STRING.H
- MATH.H
- CTYPE.H
- WINDOWS.

# CLASSES USED

```
class chess
{
   private:
        int signed board[8][8];
        int game_status ,  current_player ,  king_move;
        int log_file;
   public:
      int rules();
      int about();
      int help();
      void reset();
      void load();
      void save();
      void log();
      int opponent();
      string current_color();
      string piece_display(int piece );
      void display_board();
      void display_board(int i);
      void promotion(int y, int x);
      int validity(int y, int x,int dy,int dx);
      void tutorial();
      int get_number(string letter);
      int check();
      int move(string p_move);
      int status();

}p1;
```

# FUNCTIONS USED

1. **void delay(int seconds)**

   This function takes the value of seconds and produces a delay for that much time.

2. **int chess:: rules()**

   This function displays the contents of the file “rules.txt” on the screen.

3. **int chess::about()**

   This function displays the contents of the file “about.txt” on the screen.

4. **int chess::help()**

   This function displays the contents of the file “help.txt” on the screen.

5. **void chess::reset()**

   This function, when invoked resets all the pieces back to their initial positions, changes the current player to white.

6. **void chess::tutorial()**

   This function gives a step by step tutorial to the user on how to play the game of chess. It uses the delay() and display_board() functions.

7. **int save()**

   This function saves the current positions of all the pieces, so that the game can be reloaded from that point.

8. **int load()**

   This function loads the pieces in their last saved positions.

9. **void log()**

   This function records all the moves played by both the players in a file called “game.txt”.

10. **int opponent()**

    This function changes the current player to

11. **string current_color()**

    This function returns “white” or “black” depending on the value of current_player.

12. **string piece_display(int piece )**

    This function uses the value of piece and displays:

    r- Rook.

    k- Knight.

    b- Bishop.

    Q- Queen.

    K- King.

    p- Pawn.

13. **void display_board()**

    This function displays all the pieces and the board.

14. **void display_board(int i)**

    This function is an overloaded function of display_board with a slight variation

15. **void promotion(int y, int x)**

    This function takes in the values of variables x and y (i.e. positions of the pawn) and promotes it to a piece of the users choice when it reaches the other end of the board.

16. **int validity(int y, int x,int dy,int dx)**

    This function takes in the values of variables x, y, dx and dy. And checks if a move is legal from (x,y) to (dx,dy). It has a different algorithm for each piece.

17. **int get_number(string letter)**

    This function takes a string letter and return the corresponding integer number.

18. **int check()**

    This function uses the values of kx, ky, x and y to determine whether the king is under check. After every move, the validity of all the pieces is checked from its current position to the position of the opposing king. If there exists a legal move, a true value is returned.

19. **int move(string p_move)**

    This function analyses the move entered by the player and separates it into x, y, dx and dy components. It then passes the values to the validity and check functions. This function is also responsible for the execution of special commands like reset, save, load etc.

20. **int status()**

    This function checks the status of the game and decides the winner.

# FILES USED

- About.txt
- Save.txt
- Help.txt
- Load.txt
- Game.txt

# SOURCE CODE

```
#include <iostream>
#include <fstream>
#include <windows.h>
#include <dos.h>
#include <string.h>
#include <time.h>
#include <vector>
#include <math.h>
#include <cstdlib>
#include <sstream>
#include <conio.h>
#include <ctype.h>
#include <windows.h>

using namespace std;
void delay ( int seconds )
{
  clock_t endwait;
  endwait = clock () + seconds * CLOCKS_PER_SEC +5;
  while (clock() < endwait) {}
}

const int pawn = 1, rook = 2, knight = 3, bishop = 4, queen = 5, king = 6;
```

```
int game_exit = 0,turn_counter =0;

class chess
{
   int signed board[8][8];


   int game_status ,  current_player ,  king_move ,log_file;

   public:
      int rules();
      int about();
      int help();
      void reset();
      void load();
      int win();
      void save();
      void log();
      int check(int ky,int kx);
      int opponent();
      string current_color();
      string piece_display(int piece );
      void display_board();
      void display_board(int i);
      void promotion(int y, int x);
      int validity(int y, int x,int dy,int dx);
      void tutorial();
      int get_number(string letter);
      int check();
      int move(string p_move);
      int status();
      int check_mate();

}p1;
int chess::rules()
      {
```

```
        char arr[1000];
        ifstream fin;
        fin.open("rules.txt");
        system("cls");                          //Opening text File
        cout<<"\n";
        while(!fin.eof())
        {
          fin.getline(arr,1000,'\n');                                   //Counting the no of words

          if(fin.eof())
          {
            break;
          }
          cout<<endl<<arr<<endl;

        }
        return 0;
    }
int chess::about()
      {

        char arr[1000];
        ifstream fin;
        fin.open("about.txt");
        system("cls");
        cout<<"\n";
        while(!fin.eof())
        {
          fin.getline(arr,1000,'\n');

          if(fin.eof())
          {
            break;
          }
          cout<<endl<<arr<<endl;

        }
```

```
            return 0;
        }
int chess::help()
    {
            char arr[1000];
            ifstream fin;
            fin.open("help.txt");
            system("cls");
            cout<<"\n";
            while(!fin.eof())
            {
               fin.getline(arr,1000,'\n');

               if(fin.eof())
               {
                  break;
               }
               cout<<endl<<arr<<endl;

            }
            return 0;
    }
void chess::reset()
        {
            board[0][0] = rook;
            board[0][1] = knight;
            board[0][2] = bishop;
            board[0][3] = queen;
            board[0][4] = king;
            board[0][5] = bishop;
            board[0][6] = knight;
            board[0][7] = rook;
            board[1][0] = pawn;
            board[1][1] = pawn;
            board[1][2] = pawn;
            board[1][3] = pawn;
            board[1][4] = pawn;
```

```
board[1][5] = pawn;
board[1][6] = pawn;
board[1][7] = pawn;
board[2][0] = 0;
board[2][1] = 0;
board[2][2] = 0;
board[2][3] = 0;
board[2][4] = 0;
board[2][5] = 0;
board[2][6] = 0;
board[2][7] = 0;
board[3][0] = 0;
board[3][1] = 0;
board[3][2] = 0;
board[3][3] = 0;
board[3][4] = 0;
board[3][5] = 0;
board[3][6] = 0;
board[3][7] = 0;
board[4][0] = 0;
board[4][1] = 0;
board[4][2] = 0;
board[4][3] = 0;
board[4][4] = 0;
board[4][5] = 0;
board[4][6] = 0;
board[4][7] = 0;
board[5][0] = 0;
board[5][1] = 0;
board[5][2] = 0;
board[5][3] = 0;
board[5][4] = 0;
board[5][5] = 0;
board[5][6] = 0;
board[5][7] = 0;
board[6][0] = -pawn;
board[6][1] = -pawn;
```

```
            board[6][2] = -pawn;
            board[6][3] = -pawn;
            board[6][4] = -pawn;
            board[6][5] = -pawn;
            board[6][6] = -pawn;
            board[6][7] = -pawn;
            board[7][0] = -rook;
            board[7][1] = -knight;
            board[7][2] = -bishop;
            board[7][3] = -queen;
            board[7][4] = -king;
            board[7][5] = -bishop;
            board[7][6] = -knight;
            board[7][7] = -rook;
            current_player = 1;
            king_move = 0;
            game_status =  0;
            turn_counter = 1;
        }

void chess::save()
        {
             ofstream fout("save.txt",ios::trunc);
             for(int i=0;i<8;i++)
                 for(int j=0;j<8;j++)
                 {
                    //board[i][j]=k;
                    fout<< board[i][j]<<" ";
                    //k++;
                 }
              fout<<current_player<<" "<<game_status<<" "<<king_move<<" "<<log_file;
              fout.close();


        }
void chess::load()
        {
```

```
        string v;
        ifstream fin("save.txt",ios::app);
        for(int i=0;i<8;i++)
          for(int j=0;j<8;j++)
          {


             fin>>v;                                  //Counting the no of characters
             board[i][j] = atoi (v.c_str());
             if(fin.eof())
             {
                break;
             }

          }
        fin>>v;
        current_player = atoi (v.c_str());
        fin>>v;
        game_status = atoi (v.c_str());
        fin>>v;
        king_move = atoi (v.c_str());
        fin>>v;
        log_file = atoi (v.c_str());
        fin.close();

    }
void chess::tutorial()
    {
        for(int i=0;i<8;i++)
        {
            for(int j=0;j<8;j++)
            {
                board[i][j]=0;
            }
        }
        cout<<"THE GAME OF CHESS IS PLAYED ON A BOARD THAT HAS 8 ROWS AND 8 COLOUMS.";
        delay(1);
```

```
cout<<".";
delay(1);
cout<<".";
cout<<"\n\nLIKE THIS       ";
display_board(1);
delay(3);
system("cls");
cout<<"\nThe white pieces are placed on row no 1 and 2\n";
delay(3);
system("cls");
board[0][0] = rook;
board[0][7] = rook;
cout<<"\nThe Rook are placed in the 2 corners.\n";
delay(2);
display_board(1);
delay(5);
system("cls");
board[0][1] = knight;

board[0][6] = knight;
cout<<"Followed by the Knight And Bishop on both the sides\n\n";
delay(2);
display_board(1);
delay(2);
system("cls");
cout<<"Followed by the Knight And Bishop on both the sides\n\n";
board[0][2] = bishop;
board[0][5] = bishop;
 display_board(1);
delay(5);
system("cls");
cout<<"The Queen is placed on the 'd' coloum.\n\n";
delay(2);
board[0][3] = queen;
display_board(1);
delay(5);
system("cls");
```

```
cout<<"The King is placed on the 'e' coloum.\n\n";
delay(2);
board[0][4] = king;
display_board(1);
delay(5);
system("cls");
cout<<"The Pawns are placed in the second row, in front of every other white piece.\n\n";
delay(2);
board[1][0] = pawn;
board[1][1] = pawn;
board[1][2] = pawn;
board[1][3] = pawn;
board[1][4] = pawn;
board[1][5] = pawn;
board[1][6] = pawn;
board[1][7] = pawn;
display_board(1);
delay(5);
system("cls");
board[6][0] = -pawn;
board[6][1] = -pawn;
board[6][2] = -pawn;
board[6][3] = -pawn;
board[6][4] = -pawn;
board[6][5] = -pawn;
board[6][6] = -pawn;
board[6][7] = -pawn;
board[7][0] = -rook;
board[7][1] = -knight;
board[7][2] = -bishop;
board[7][3] = -queen;
board[7][4] = -king;
board[7][5] = -bishop;
board[7][6] = -knight;
board[7][7] = -rook;
cout<<"The Black Pieces are placed similarly on the 7 & 8 rows.\n\n";
delay(2);
```

```
        display_board(1);
        delay(7);
        system("cls");
        reset();
        cout<<"     Now for the way each piece move\n\n";
        delay(3);
        system("cls");
        cout<<"                          PAWN";
        delay(2);
        system("cls");
        cout<<"The pawn moves forward exactly one space, or optionally, two spaces when on its
starting square, toward the opponent's side of the board.\n\n ";
        display_board(1);
        delay(3);
        system("cls");
        board[1][4] = 0;
        board[3][4] = pawn;
        //cout<<"PAWN";
        cout<<"The pawn moves forward exactly one space, or optionally, two spaces when on its
starting square, toward the opponent's side of the board. \n\n";
        display_board(1);
        delay(3);
        system("cls");
        board[3][4] = 0;
        board[4][4] = pawn;

        cout<<"The pawn moves forward exactly one space, or optionally, two spaces when on its
starting square, toward the opponent's side of the board. \n\n";
        display_board(1);
        delay(3);
        system("cls");
        board[5][5]=-1;
        board[6][5]=0;
        cout<<"When there is an enemy piece one square diagonally ahead of the pawn, either    left or
right, then the pawn may capture that piece.\n\n";
        delay(3);
        display_board(1);
```

```
        delay(3);
        board[5][5]=1;
        board[4][4] = 0;
        system("cls");
        cout<<"When there is an enemy piece one square diagonally ahead of the pawn, either    left or right, then the pawn may capture that piece.\n\n";
        display_board(1);
        delay(3);
        reset();
         system("cls");
        cout<<" If the pawn reaches a square on the back rank of the opponent, it promotes to the player's choice of a queen, rook, bishop, or knight\n\n";
        board[0][2] = 0;
        board[1][3] = 0;
        board[3][5] = 4;
        board[1][1] = -1;
        board[6][1] = 0;
        delay(3);
        display_board(1);
        delay(3);
        system("cls");
        cout<<" If the pawn reaches a square on the back rank of the opponent, it promotes to the player's choice of a queen, rook, bishop, or knight\n\n";
        board[1][1] = 0;
        board[0][0] = -1;
        display_board(1);
        cout<<"\n Change pawn to : \n1.Queen \n2.Rook \n3.Bishop\n4.Knight\nEnter your choice : ";
        delay(3);
        cout<<"1";
        delay(3);
        system("cls");
        board[0][0] = -5;
        display_board(1);
        delay(3);
        reset();
        system("cls");
        cout<<"\n The Knight moves in a L shape.And can jump over pieces that come in its path.\n\n";
```

```
        delay(2);
        display_board(1);
        delay(2);
        board[2][2] = 3;
        board[0][1] = 0;
        system("cls");
        cout<<"\n The Knight moves in a L shape.And can jump over pieces that come in its path.\n\n";
        delay(3);
        reset();
        system("cls");
        cout<<"The bishop moves any number of vacant squares diagonally in a straight line.\n\n";
        board[1][3]=0;
        delay(2);
        display_board(1);
        delay(2);
        system("cls");
        board[0][2]=0;
        board[3][5]=4;
        cout<<"The bishop moves any number of vacant squares diagonally in a straight line.\n\n";
        display_board(1);
        delay(2);
        system("cls");
        reset();

    }
void chess::log()
    {


        cout<<" \nDo You Want The Game To Be Logged In A File (y\\n) ";
        char choice;
        cin>> choice;
        if (choice == 'y'||choice == 'Y')
        {
            log_file = 1;
            ofstream fout("game.txt");
        }
```

```
            else if (choice == 'n'||choice == 'N')
               log_file = 0;
            else
            {
               cout<<"\nInvalid Move Enter Again";
               log();
            }

         }
int chess::opponent()
         {

            if(current_player == 1)
               return 2;
            else
               return 1;

         }
string chess::current_color()
         {
            if(current_player == 1)
               return "White";
            else
               return "Black";
         }
string chess::piece_display(int piece = 0)
         {
               string str="";
               if(piece>0)
               {
                     switch(piece)
                     {
                        case rook:    str = " r "; break;
                        case knight:  str = " k "; break;
                        case bishop:  str = " b "; break;
                        case queen:   str = " Q "; break;
                        case king:    str = " K "; break;
```

```
                case pawn:    str = " p "; break;
                }
            }

            else if(piece<0)
            {
               switch(-piece)
               {
                  case rook:    str = " r "; break;
                  case knight:  str = " k "; break;
                  case bishop:  str = " b "; break;
                  case queen:   str = " Q "; break;
                  case king:    str = " K "; break;
                  case pawn:    str = " p "; break;
               }
            }
            else
               str = " ";

            return str;
        }
void chess::display_board()
        {
           system("cls");
           int k =0;
           HANDLE hConsole;
           system(" color 31");
           hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
           SetConsoleTextAttribute(hConsole, 49);
           cout<<"\n \t\t   a   b   c   d   e   f   g   h \n\n";
           for(int i=7;i>=0;i--)
           {
              switch(i)
              {
                 case 0: cout << "\t\t1 ";
                      break;
                 case 1: cout << "\t\t2 ";
```

```
            break;
    case 2: cout << "\t\t3 ";
            break;
    case 3: cout << "\t\t4 ";
            break;
    case 4: cout << "\t\t5 ";
            break;
    case 5: cout << "\t\t6 ";
            break;
    case 6: cout << "\t\t7 ";
            break;
    case 7: cout << "\t\t8 ";
            break;
}
cout<<" ";
if(i%2==0)
{
    k=0;

}
else
{
    k=1;
}
for( int j = 0;j<8;j++)
{
    if(board[i][j]<0)
    {
      if(k%2==0)
      {
         SetConsoleTextAttribute(hConsole, 96);
         cout<<piece_display(board[i][j]);
      }
      else if(k%2==1)
      {
         SetConsoleTextAttribute(hConsole, 128);
         cout<<piece_display(board[i][j]);
```

```
  }

}
if(board[i][j]>0)
{

   if(k%2==0)
   {
    SetConsoleTextAttribute(hConsole, 111);
    cout<<piece_display(board[i][j]);
   }
   else
   {
    SetConsoleTextAttribute(hConsole, 143);
    cout<<piece_display(board[i][j]);
   }

}
if(board[i][j])
 {
   if(k%2==0)
   {
    SetConsoleTextAttribute(hConsole, 97);
    cout<<" ";
    k++;
   }
   else
   {
    SetConsoleTextAttribute(hConsole, 129);
    cout<<" ";
    k++;
   }
 }
else
   if(k%2==0)
   {
    SetConsoleTextAttribute(hConsole, 97);
```

```
        cout<<"   ";
        k++;
       }
       else
       {
        SetConsoleTextAttribute(hConsole, 129);
        cout<<"   ";
        k++;
       }
   if(j==7)
      cout<<"\n";
}
SetConsoleTextAttribute(hConsole, 49);

if(i%2==1)
{
   {
      cout<<"\t\t  ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 49);
      cout<<" ";
```

```
            }
          }
          else
          {
            {
              cout<<"\t\t   ";
              SetConsoleTextAttribute(hConsole, 97);
              cout<<"    ";
              SetConsoleTextAttribute(hConsole, 129);
              cout<<"    ";
              SetConsoleTextAttribute(hConsole, 97);
              cout<<"    ";
              SetConsoleTextAttribute(hConsole, 129);
              cout<<"    ";
              SetConsoleTextAttribute(hConsole, 97);
              cout<<"    ";
              SetConsoleTextAttribute(hConsole, 129);
              cout<<"    ";
              SetConsoleTextAttribute(hConsole, 97);
              cout<<"    ";
              SetConsoleTextAttribute(hConsole, 129);
              cout<<"    ";
              SetConsoleTextAttribute(hConsole, 49);
              cout<<" ";

            }
          }
          cout<<endl;
        }
      }
void chess::display_board(int i)
      {

        int k =0;
        HANDLE hConsole;
        system(" color 31");
        hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
```

```
SetConsoleTextAttribute(hConsole, 49);
cout<<"\n \t\t   a   b   c   d   e   f   g   h \n\n";
for(int i=7;i>=0;i--)
{
   switch(i)
   {
      case 0: cout << "\t\t1 ";
            break;
      case 1: cout << "\t\t2 ";
            break;
      case 2: cout << "\t\t3 ";
            break;
      case 3: cout << "\t\t4 ";
            break;
      case 4: cout << "\t\t5 ";
            break;
      case 5: cout << "\t\t6 ";
            break;
      case 6: cout << "\t\t7 ";
            break;
      case 7: cout << "\t\t8 ";
            break;
   }
   cout<<" ";
   if(i%2==0)
   {
      k=0;

   }
   else
   {
      k=1;
   }
   for( int j = 0;j<8;j++)
   {
      if(board[i][j]<0)
      {
```

```
 if(k%2==0)
 {
    SetConsoleTextAttribute(hConsole, 96);
    cout<<piece_display(board[i][j]);
 }
 else if(k%2==1)
 {
    SetConsoleTextAttribute(hConsole, 128);
    cout<<piece_display(board[i][j]);
 }

}
if(board[i][j]>0)
{

  if(k%2==0)
  {
    SetConsoleTextAttribute(hConsole, 111);
    cout<<piece_display(board[i][j]);
  }
  else
  {
    SetConsoleTextAttribute(hConsole, 143);
    cout<<piece_display(board[i][j]);
  }

}
if(board[i][j])
 {
  if(k%2==0)
  {
    SetConsoleTextAttribute(hConsole, 97);
    cout<<" ";
    k++;
  }
  else
  {
```

```
        SetConsoleTextAttribute(hConsole, 129);
        cout<<" ";
        k++;
      }
    }
   else
      if(k%2==0)
      {
        SetConsoleTextAttribute(hConsole, 97);
        cout<<"   ";
        k++;
      }
      else
      {
        SetConsoleTextAttribute(hConsole, 129);
        cout<<"   ";
        k++;
      }
   if(j==7)
      cout<<"\n";
}
SetConsoleTextAttribute(hConsole, 49);

if(i%2==1)
{
   {
      cout<<"\t\t  ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
```

```
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 49);
      cout<<" ";

   }
}
else
{
   {
      cout<<"\t\t  ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 97);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 129);
      cout<<"   ";
      SetConsoleTextAttribute(hConsole, 49);
      cout<<" ";

   }
}
cout<<endl;
```

```
            }
        }
void chess::promotion(int y,int x)
        {
            cout<<"\n Change pawn to : \n1.Queen \n2.Rook \n3.Bishop\n4.Knight\nEnter your choice : ";
            int choice;
            cin>>choice;
            if(current_player == 1)
            {
               switch(choice)
               {
                  case 1: board[y][x] = 5;
                        break;
                  case 2: board[y][x] = 2;
                        break;
                  case 3: board[y][x] = 4;
                        break;
                  case 4: board[y][x] = 3;
                        break;
               }
            }
            else if(current_player == 2)
            {
               switch(choice)
               {
                  case 1: board[y][x] = -5;
                        break;
                  case 2: board[y][x] = -2;
                        break;
                  case 3: board[y][x] = -4;
                        break;
                  case 4: board[y][x] = -3;
                        break;
               }
            }
        }
int chess::validity(int y,int x,int dy,int dx)
```

```
{
   int source = board[y][x];
   int target = board[dy][dx];
   switch(abs(source))
   {
      case 1:
           {
              if(current_player==1 && dy>y)
              {
                 if(dx==x && abs(dy-y)==1 && target == 0)
                   {
                      if( dy == 7)
                      {
                         cout<<"promotion w\n";
                         promotion(y,x);

                      }
                      return 11;

                   }
                 if(abs(dx-x)==1 && abs(dy-y)==1 && target != 0)
                   {
                      if( dy == 7)
                      {
                         cout<<"promotion w\n";
                         promotion(y,x);

                      }
                      return 12;
                   }
                 if(y==1 && dx==x && abs(dy-y)==2 && target == 0)
                 {

                      return 13;
                 }
                  return 0;
```

```
        }
        if(current_player == 2 && dy<y)
        {
          if(dx==x && abs(dy-y)==1 && target == 0)
          {
                if(dy == 0)
                {
                  cout<<"promotion w\n";
                  promotion(y,x);

                }
                return 21;
          }
          if(abs(dx-x)==1 && abs(dy-y)==1 && target != 0 )
          {
                if(dy == 0)
                {
                  cout<<"promotion w\n";
                  promotion(y,x);

                }
                return 22;
          }
          if(y==6 && dx==x && abs(dy-y)==2 && target == 0)
          {

             return 23;
          }

          return 0;
        }
        return 0;
    }

case 2:
    {
```

```
if(dx==x)
{
   if(dy>y)
   {
      for( int ch = (y+1);ch < dy;ch++)
      {
            if (board[ch][dx]!=0)
               return 0;
      }
      return 21;
   }
   else
   {
      for ( int ch = dy+1;ch < dy;ch++)
      {
            if (board[ch][dx]!=0)
               return 0;
      }
      return 22;

   }


}
if(dy==y)
{
   if(dx>x)
   {
         for ( int ch = (y+1);ch < dy;ch++)
         {
            if (board[dy][ch]!=0)
               return 0;
         }
         return 23;
   }
   if(dx<x)
   {
```

```
            for ( int ch = dx+1;ch < dy-1;ch++)
            {
                if (board[dy][ch]!=0)
                    return 0;
            }
            return 24;
          }

        }
        return 0;
    }
case 3:
    {
        if((abs(dy-y) == 2 && abs(dx-x) == 1)||(abs(dx-x) == 2 && abs(dy-y) == 1))
        {
          return 3;
        }
        return 0;

    }

case 4:
    {
        if(abs(dy-y) == abs(dx-x))
        {

            if((dy>y && dx>x) || (x>dx && y>dy))
            {
              int xyz = min(dx,x)+1;
              int ch = min(dy,y)+1;

              for(; ch < dy; ch++)
              {
                if(board[ch][xyz] != 0)
                    return 0;
                xyz ++;
              }
```

```
                return 4;
            }
            if((dx>x && y>dy) || (dy>y && dx<x))
            {
                int xyz = max(dx,x)-1;
                int ch = min(dy,y) +1;
                for(; ch < y; ch++)
                {
                    if(board[ch][xyz] != 0)
                        return 0;
                    xyz --;
                }
                return 4;
            }
          }
          return 0;
          // break;
      }
case 5:
      {
        if(abs(dy-y) == abs(dx-x))
        {
            if((dy>y && dx>x) || (x>dx && y>dy))
            {
                int xyz = min(dx,x)+1;
                int ch = min(dy,y)+1;
                for(; ch < y; ch++)
                {
                    if(board[ch][xyz] != 0)
                        return 0;
                    xyz ++;
                }
                return 1;
            }
            if((dx>x && y>dy) || (dy>y && dx<x))
            {
                int xyz = max(dx,x)-1;
```

```
            int ch = min(dy,y) +1;
            for(; ch < y; ch++)
            {
               if(board[ch][xyz] != 0)
                  return 0;
               xyz --;
            }
            return 4;
      }
}

if(dy==y || dx==x)
{
   if(dx==x)
   {
      if(dy>y)
      {
            for(int ch = (y+1); ch < dy; ++ch)
            {
               if(board[ch][dx] != 0)
                  return 0;
            }

      }
      if(y>dy)
      {
            for(int ch = dy; ch < (y-1); ++ch)
            {
                  if(board[ch][dx] != 0)
                     return 0;
            }
      }
      return 5;
   }
   if(dy==y)
   {
      if(dx>x)
```

```
                    {
                        for(int xyz = (x+1); xyz <dy; ++xyz)
                        {
                            if(board[dy][xyz] != 0)
                                return 0;
                        }
                    }
                    if(dx<x)
                    {
                        for(int xyz =dx; xyz < (x-1); ++xyz)
                        {
                            if(board[dy][xyz] != 0)
                                return 0;
                        }
                    }
                    return 5;
                  }
                }
                return 0;

            }
        case 6:
            {
                if (abs(dy-y)<=1 && abs(dx-x)<=1)
                {
                  //king_move=1;
                  return 6;
                }
                if(current_player==1)
                {
                 //  cout<<"white castel\n";
                  if(abs(dx-x==2)&& king_move == 0 && (board[0][0]==2 || board[0][7]==2) &&
dy==y)
                  {
                    cout<<"step 2";
                    if(dx == 6)
                    {
```

```
            cout<<"step 3";
            if(board[0][5]!= 0)
               return 0;
            if(board[0][6]!=0)
               return 0;
            board[0][5]=2;
            board[0][7]=0;
            return 6;
         }
         if(dx == 2)
         {
            if(board[0][1]!= 0)
               return 0;
            if(board[0][2]!=0)
               return 0;
            if(board[0][3]!=0)
               return 0;
            board[0][3]=2;
            board[0][0]=0;
            return 6;
         }
      }
   }
   else if(current_player == 2)
   {
      if(abs(dx-x==2)&& king_move == 0 && (board[7][0]== -2 || board[7][7]== -2))
      {
         if(dx == 6)
         {
            if(board[7][5]!= 0)
               return 0;
            if(board[7][6]!=0)
               return 0;
            board[7][5]=2;
            board[7][7]=0;
            return 6;
         }
```

```
                        if(dx == 2)
                        {
                            if(board[7][1]!= 0)
                                return 0;
                            if(board[7][2]!=0)
                                return 0;
                            if(board[7][3]!=0)
                                return 0;
                            board[7][3]=2;
                            board[7][0]=0;
                            return 6;
                        }
                      }
                    }
                    return 0;
                }
            }
            return 0;
        }
int chess::get_number(string letter)
        {
            if(letter == "a")
                return 0;
            if(letter == "b")
                return 1;
            if(letter == "c")
                return 2;
            if(letter == "d")
                return 3;
            if(letter == "e")
                return 4;
            if(letter == "f")
                return 5;
            if(letter == "g")
                return 6;
            if(letter == "h")
                return 7;
```

```
        }
int chess::check()
        {
           int kx,ky,i,j,k=0;
           for(i=0;i<8;i++)
           {
                 for(j=0;j<8;j++)
                 {
                    if(current_player==1)
                    {
                       if(board[i][j] == king)
                       {
                          kx=j;
                          ky=i;
                          cout<<"king poss "<<kx<<" "<<ky<<endl;

                       }
                    }
                    else
                    {
                       if(board[i][j]== -king)
                       {
                          kx=j;
                          ky=i;
                          cout<<"king poss "<<kx<<" "<<ky<<endl;

                       }
                    }
                 }
           }
          //  ctr = 0;
           for(i=0;i<8;i++)
           {
              for(j=0;j<8;j++)
              {
                 if(board[j][i]!= 0)
                 {
```

```
                if (current_player == 1)
                {
              // cout<<"o"<<k++<<" "<<board[i][j]<<" ";
                    if(board[j][i] < 0)
                    {
                       cout<<++k<<"w ";
                      // ctr = validity(i,j,ky,kx);
                       //cout<<"valid "<<validity(i,j,ky,kx);
                       if(validity(j,i,ky,kx))
                       {
                          if(check_mate())
                             game_status=current_player;
                          //cout<<"white CHECK  y= "<<j<<" x = "<<i<<" board = "<<board[j][i]<<" kx =
"<<kx<<" ky = "<<ky<<" valid "<<validity(j,i,ky,kx)<<endl;
                          return 1;


                       }

                    }

                }
                else if ((current_player == 2))
                {

                       if(board[j][i] > 0)
                       {
                          cout<<++k<<"b ";
                         // ctr = validity(i,j,ky,kx);
                          if(validity(j,i,ky,kx))
                          {
                             if(check_mate())
                             game_status=current_player;
                             //cout<<"BLACK CHECK  y= "<<j<<" x = "<<i<<" board = "<<board[j][i]<<" kx =
"<<kx<<" ky = "<<ky<<" valid "<<validity(j,i,ky,kx)<<endl;
                             return 1;
```

```
                        }
                    }
                }

            }
        }


    }
    return 0;
}
int chess::check(int ky,int kx)
{
    int i,j;
    for(i=0;i<8;i++)
    {
        for(j=0;j<8;j++)
        {
            if(board[j][i]!= 0)
            {

                if (current_player == 1)
                {
            // cout<<"o"<<k++<<" "<<board[i][j]<<" ";
                    if(board[j][i] < 0)
                    {
                      // cout<<++k<<"w ";
                      // ctr = validity(i,j,ky,kx);
                       //cout<<"valid "<<validity(i,j,ky,kx);
                       if(validity(j,i,ky,kx))
                       {

                           //cout<<"white CHECK  y= "<<j<<" x = "<<i<<" board = "<<board[j][i]<<" kx =
"<<kx<<" ky = "<<ky<<" valid "<<validity(j,i,ky,kx)<<endl;
                           return 1;
```

```
                        }

                    }

                }
                else if ((current_player == 2))
                {

                    if(board[j][i] > 0)
                    {
//                          cout<<++k<<"b ";
                       // ctr = validity(i,j,ky,kx);
                        if(validity(j,i,ky,kx))
                        {
                            //cout<<"BLACK CHECK  y= "<<j<<" x = "<<i<<" board = "<<board[j][i]<<" kx =
"<<kx<<" ky = "<<ky<<" valid "<<validity(j,i,ky,kx)<<endl;

                            return 1;


                        }
                    }
                }

            }
        }


    }
    return 0;
}
int chess::move(string p_move)
{
    lbl:
    int ctr= 0;
```

```
if (p_move == "reset")
{
   reset();
   display_board();
   return 0 ;
}
if (p_move == "save")
{
   save();
   return 0;
}
if (p_move == "load")
{
   load();
   return 0;
}
if(p_move == "exit" || p_move == "quit")
{
   game_exit = 1;
   return 0;
}
if(p_move == "load")
   return 0;
if(p_move == "save")
   return 0;

if (p_move == "board")
{
   cout<<"\nEnter the cordinates ";
   string corrd;
   cin>>corrd;
   string xx = corrd.substr(0,1);
   string yy = corrd.substr(1,1);
   int xxx = get_number(xx);
   int yyy =  atoi(yy.c_str())-1;
   cout<<" = = "<<board[yyy][xxx]<<" y = "<<yyy<<" x = "<<xxx;
   return 0;
```

```
}
if (p_move == "rules")
{
   rules();
   return 0;
}
int x=0,y=0,dx=0,dy=0;
string from_x = p_move.substr(0,p_move.find("-")).substr(0,1);
string from_y = p_move.substr(0,p_move.find("-")).substr(1,1);
string to_x = p_move.substr(p_move.find("-")+1,p_move.length()).substr(0,1);
string to_y = p_move.substr(p_move.find("-")+1,p_move.length()).substr(1,1);
x = get_number(from_x);
y = atoi(from_y.c_str())-1;
dx = get_number(to_x);
dy = atoi(to_y.c_str())-1;

int source = board[y][x];
int target = board [dy][dx];
if(current_player == 1 && (source<=0 || target >0) )
{
   cout<<"ILLEGAL MOVE 1";
   getch();
   return 0;
}
if(current_player == 2 && (source>=0 || target <0) )
{
   cout<<"ILLEGAL MOVE 2";
   getch();
   return 0;
}
// cout<<"\nvalid - "<<validity(y,x,dy,dx)<<" ";

if(validity(y,x,dy,dx))
{

   source =  board[y][x];
```

```
            board[y][x] = 0;
            board[dy][dx] = source;

            if(check())
            {
               board[y][x] = source;
               board[dy][dx] = target;
               cout<<"INVALID MOVE KING UNDER CHECK  "<<ctr;
               getch();
               return 0;
            }
//             takenlog(target);
            if(check_mate())
               {
                  game_status=current_player;
                  getch();
               }
            if(abs(target)== king)
            {
               game_status = current_player;
               return 0;
            }
            turn_counter++;

            display_board();
            if(log_file)
            {
               ofstream fout("game.txt",ios::app);
               fout<<current_color()<<" plays "<<p_move<<"\n";
            }
            current_player = opponent();

         }
         else
         {
            cout<<"ILLEGAL MOVE 3";
            getch();
```

```
            getch();


        }
        return 0;
    }
int chess::win()
    {

            system("cls");
            char arr;
            ifstream fin("win.txt");
            while(!fin.eof())
            {
               fin.get(arr);
               if(fin.eof())
               {
                  break;
               }
               cout<<arr;                    //Printing the content of the file

            }
            delay(3);
            cout<<endl<<endl;
            if(game_status==1)
            {
               ifstream white("white.txt");
               while(!white.eof())
               {
                  white.get(arr);
                  if(white.eof())
                  {
                     break;
                  }
                  cout<<arr;                    //Printing the content of the file

               }
```

```
                }
                else
                {
                   ifstream black("black.txt");
                   while(!black.eof())
                   {
                      black.get(arr);
                      if(black.eof())
                      {
                         break;
                      }
                      cout<<arr;                    //Printing the content of the file

                   }
                }
                getch();

        }
int chess::check_mate()
        {

              int i,j,kx,ky;
              for(i=0;i<8;i++)
              {
                    for(j=0;j<8;j++)
                    {
                       if(current_player==1)
                       {
                          if(board[i][j] == king)
                          {
                             kx=j;
                             ky=i;
                             cout<<"king poss "<<kx<<" "<<ky<<endl;

                          }
                       }
                       else
```

```
        {
            if(board[i][j]== -king)
            {
                kx=j;
                ky=i;
                cout<<"king poss "<<kx<<" "<<ky<<endl;

            }
        }
    }
}
if(current_player==1)
{
   if(kx+1<=7)
   {
      if(board[ky][kx+1]<=0)
      {
         if(!check(ky,kx+1))
         {
            return 0;
         }

      }
      if(ky-1>=0)
      {
         if(board[ky-1][kx+1]==0 || board[ky-1][kx+1]<0)
         {
            if(!check(ky-1,kx+1))
            {
               return 0;
            }
         }
      }
      if(ky+1<=7)
      {
         if(board[ky+1][kx+1]==0 || board[ky+1][kx+1]<0)
         {
```

```
            if(!check(ky+1,kx+1))
            {
               return 0;
            }
         }
      }

}

{
   if(board[ky][kx]==0 || board[ky][kx]<0)
   {
      if(!check(ky,kx))
      {
         return 0;
      }

   }
   if(ky-1>=0)
   {
      if(board[ky-1][kx]==0 || board[ky-1][kx]<0)
      {
         if(!check(ky-1,kx))
         {
            return 0;
         }
      }
   }
   if(ky+1<=7)
   {
      if(board[ky+1][kx]==0 || board[ky+1][kx]<0)
      {
         if(!check(ky+1,kx))
         {
            return 0;
         }
      }
```

```
        }

    }
    if(kx-1>=0)
    {
        if(board[ky][kx-1]==0 || board[ky][kx-1]<0)
        {
            if(!check(ky,kx-1))
            {
                return 0;
            }

        }
        if(ky-1>=0)
        {
            if(board[ky-1][kx-1]==0 || board[ky-1][kx-1]<0)
            {
                if(!check(ky-1,kx-1))
                {
                    return 0;
                }
            }
        }
        if(ky+1<=7)
        {
            if(board[ky+1][kx-1]==0 || board[ky+1][kx-1]<0)
            {
                if(!check(ky+1,kx-1))
                {
                    return 0;
                }
            }
        }

    }
}
else
```

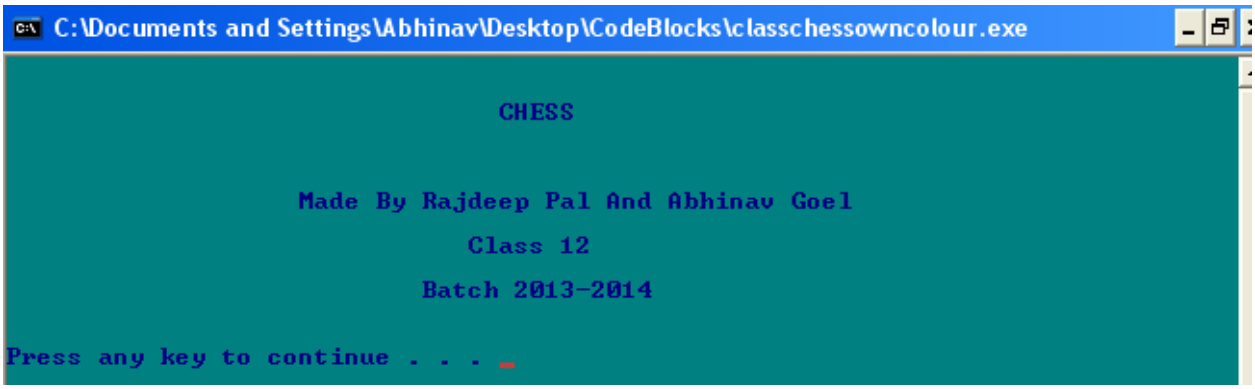
```
{

   if(kx+1<=7)
   {
      if(board[ky][kx+1]>=0)
      {
         if(!check(ky,kx+1))
         {
            return 0;
         }

      }
      if(ky-1>=0)
      {
         if(board[ky-1][kx+1]==0 || board[ky-1][kx+1]>0)
         {
            if(!check(ky-1,kx+1))
            {
               return 0;
            }
         }
      }
      if(ky+1<=7)
      {
         if(board[ky+1][kx+1]==0 || board[ky+1][kx+1]>0)
         {
            if(!check(ky+1,kx+1))
            {
               return 0;
            }
         }
      }

   }

   {
      if(board[ky][kx]==0 || board[ky][kx]>0)
```

```
    {
        if(!check(ky,kx))
        {
            return 0;
        }

    }
    if(ky-1>=0)
    {
        if(board[ky-1][kx]==0 || board[ky-1][kx]>0)
        {
            if(!check(ky-1,kx))
            {
                return 0;
            }
        }
    }
    if(ky+1<=7)
    {
        if(board[ky+1][kx]==0 || board[ky+1][kx]>0)
        {
            if(!check(ky+1,kx))
            {
                return 0;
            }
        }
    }

}
if(kx-1>=0)
{
    if(board[ky][kx-1]==0 || board[ky][kx-1]>0)
    {
        if(!check(ky,kx-1))
        {
            return 0;
        }
```

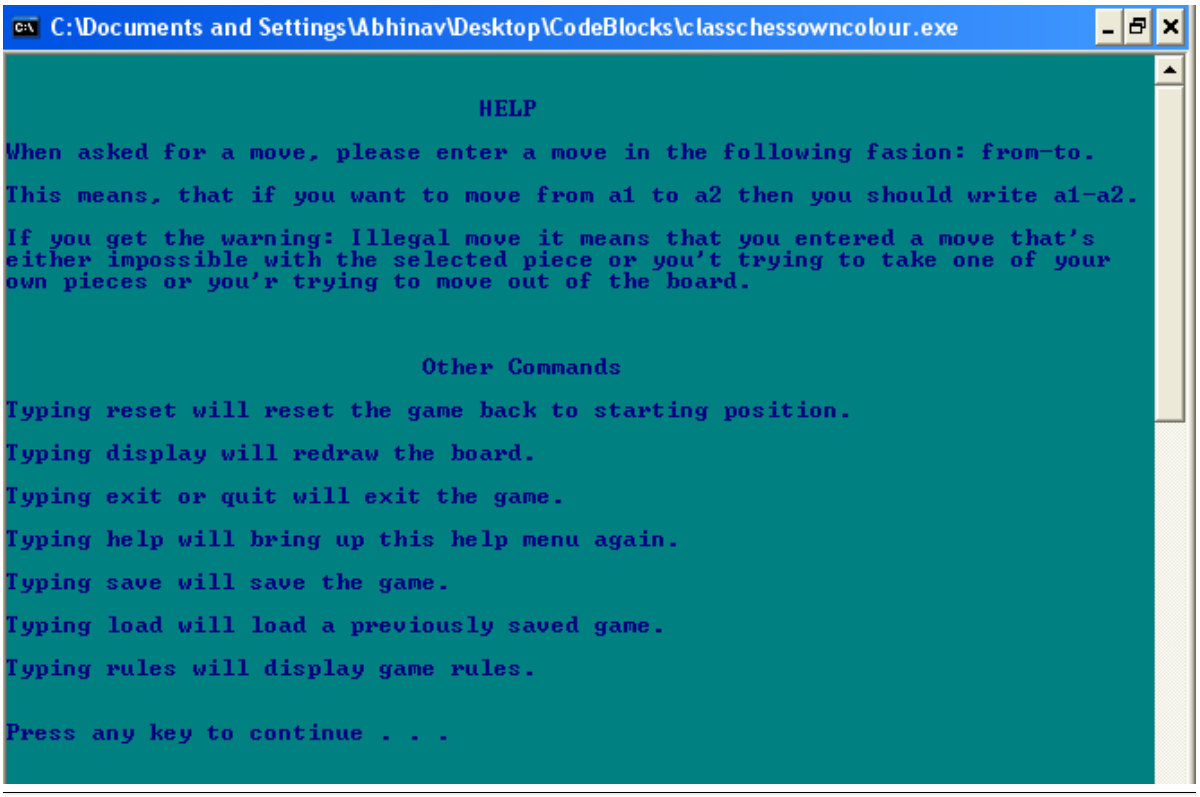
```
                }
                if(ky-1>=0)
                {
                   if(board[ky-1][kx-1]==0 || board[ky-1][kx-1]>0)
                   {
                      if(!check(ky-1,kx-1))
                      {
                         return 0;
                      }
                   }
                }
                if(ky+1<=7)
                {
                   if(board[ky+1][kx-1]==0 || board[ky+1][kx-1]>0)
                   {
                      if(!check(ky+1,kx-1))
                      {
                         return 0;
                      }
                   }
                }

             }
          }

          return 1;

       }
int chess::status()
       {
          if(game_status == 0)
             return 0;
          else
             {

                win();
```

```
                return 1;
            }
        }
main()
{
   system("color 31");
   p1.reset();
   p1.about();
   system("pause");
   system("cls");
   p1.help();
   system("pause");
   system("cls");
   p1.log();
   system("cls");
   //p1.tutorial();
   p1.reset();
   p1.display_board();
   while(game_exit == 0)
   {
        p1.display_board();
        if(p1.status())
        {
           break;
        }
        cout<<"\nTurn "<<turn_counter<<"  "<<p1.current_color()<<" Move : ";
        string p_move = "";
        cin>> p_move;
        p1.move(p_move);

   }
}
```

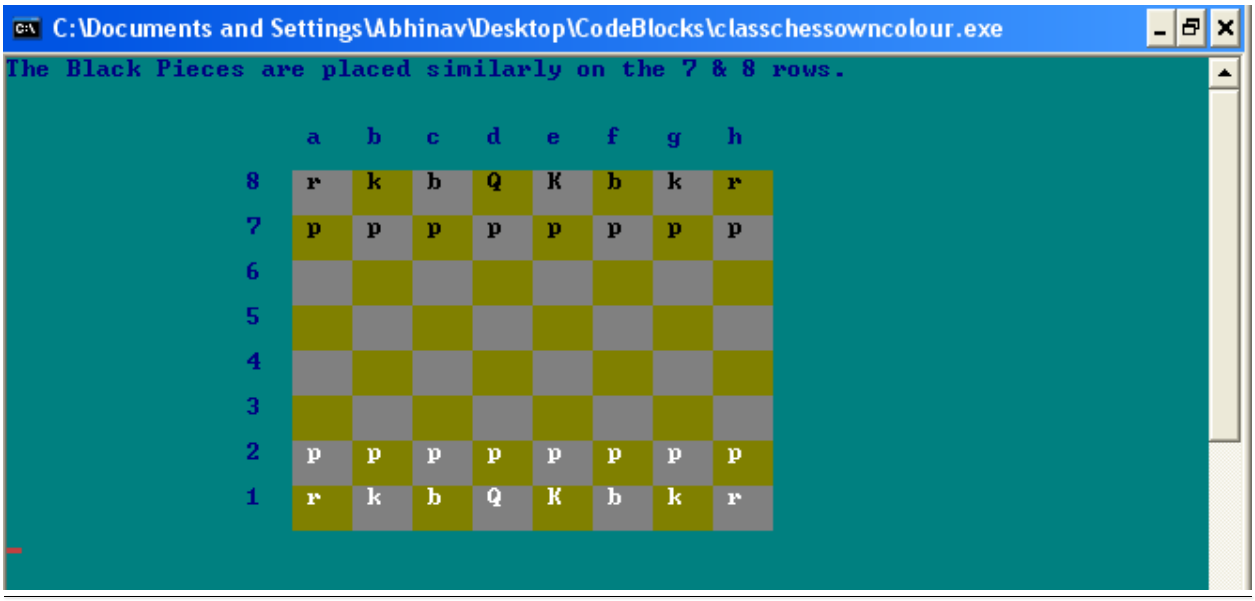
# SAMPLE OUTPUT




ABOUT PAGE

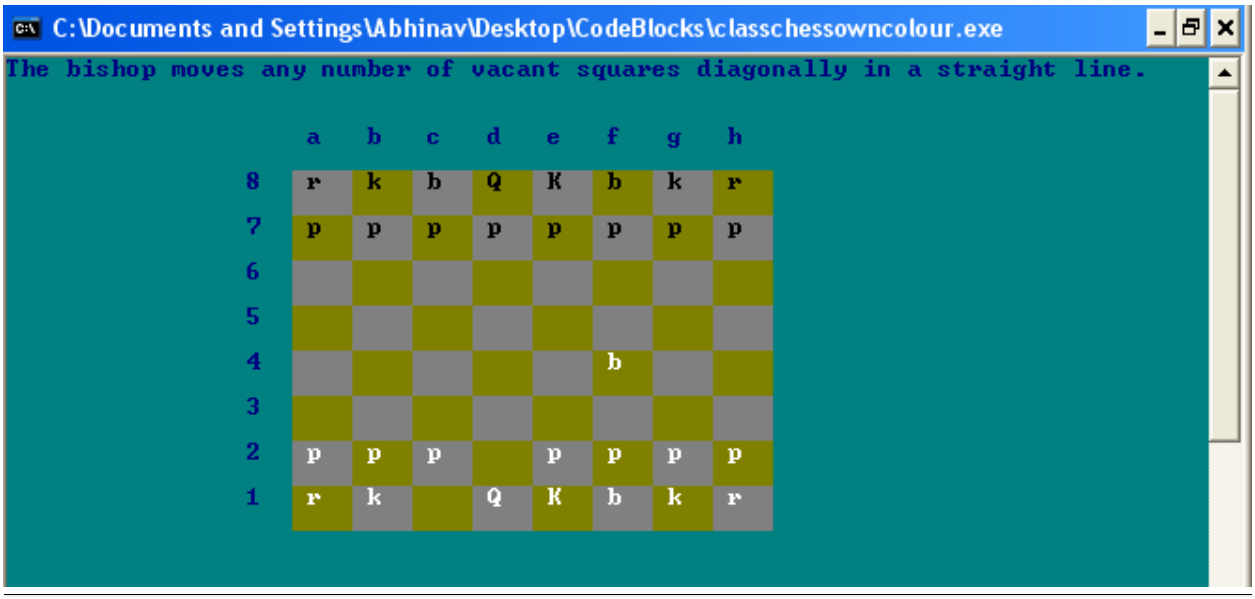



HELP PAGE

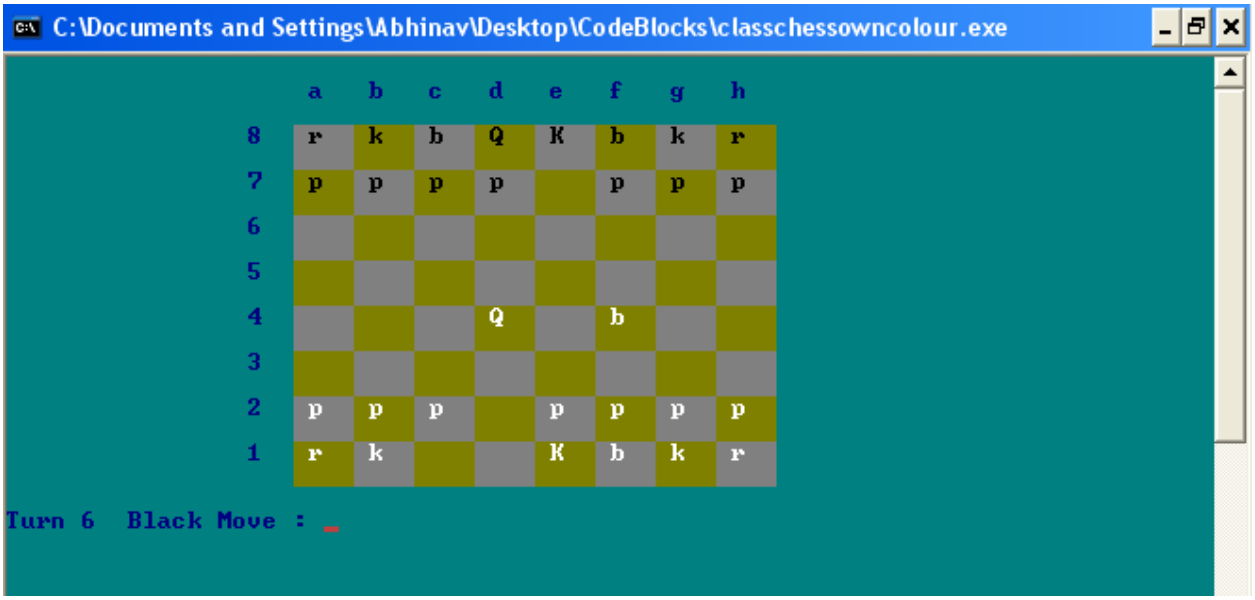



TUTORIAL 1

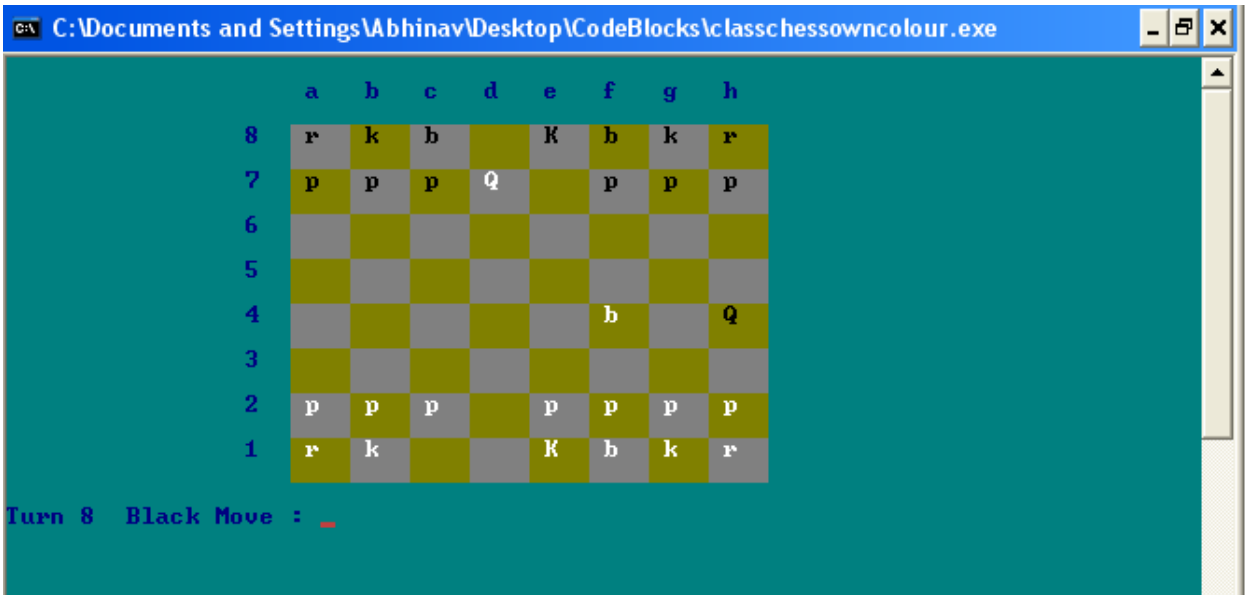

TUTORIAL 2



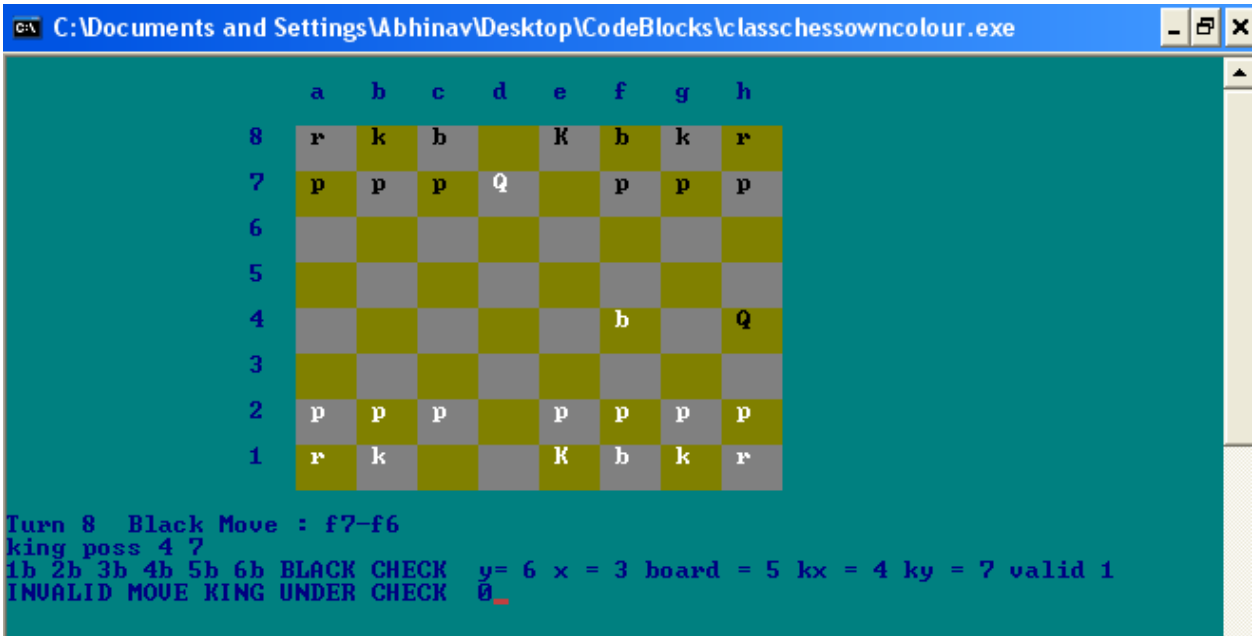

TUTORIAL 3



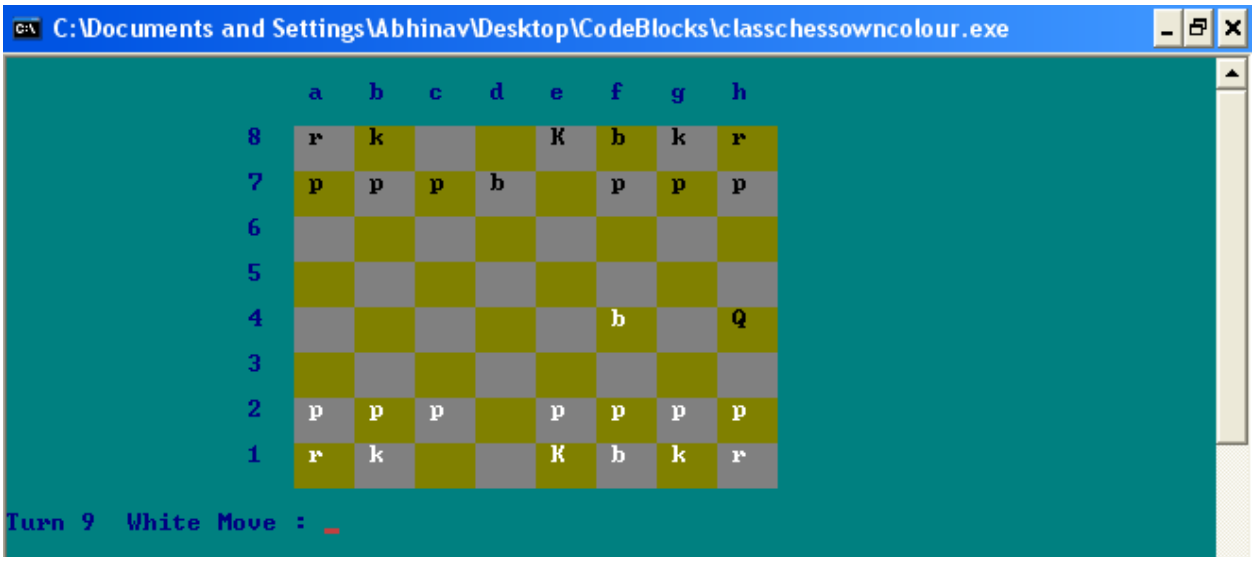

QUEEN MOVE



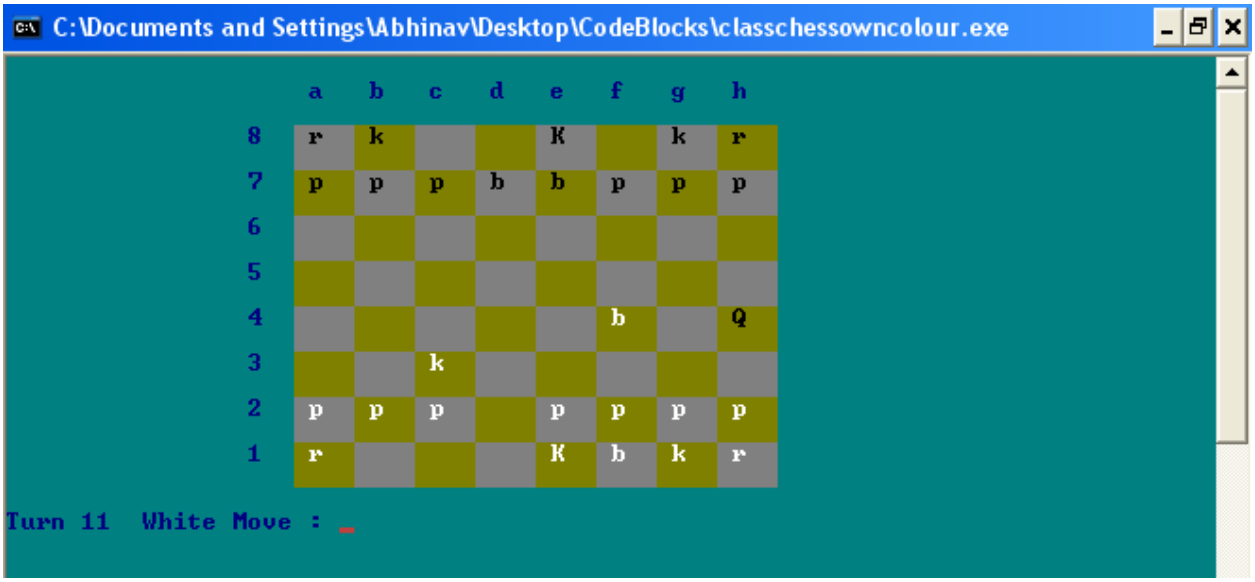

CHECK 1



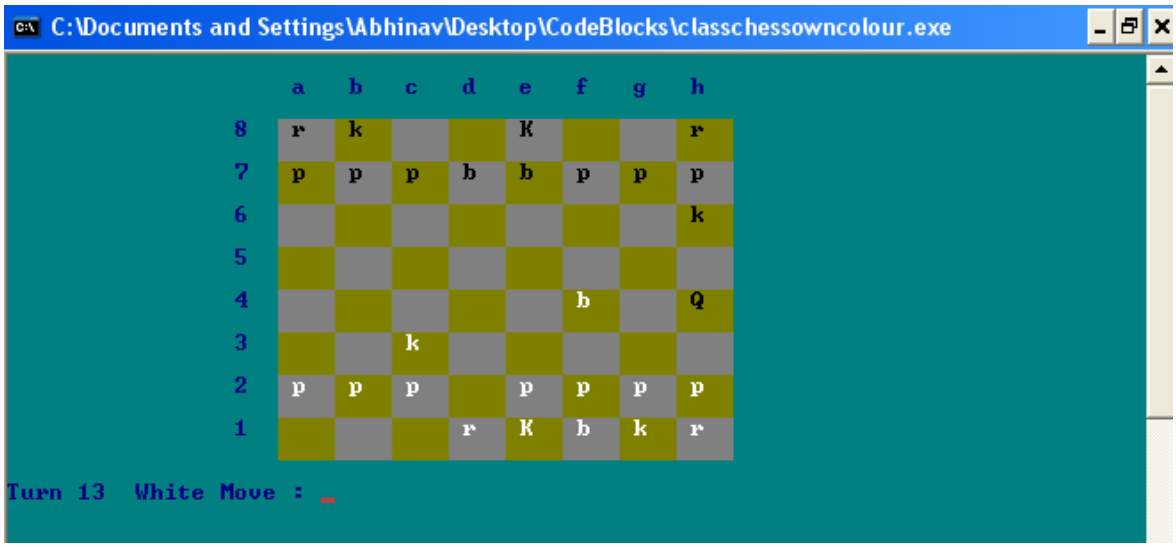

CHECK 2



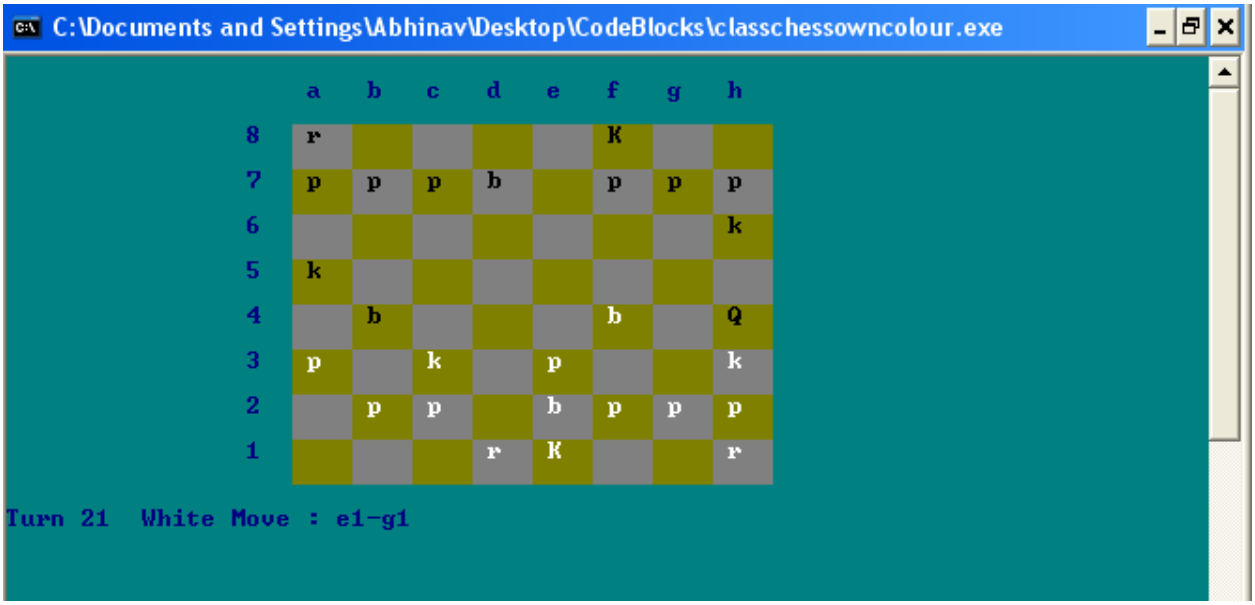

CHECK 3



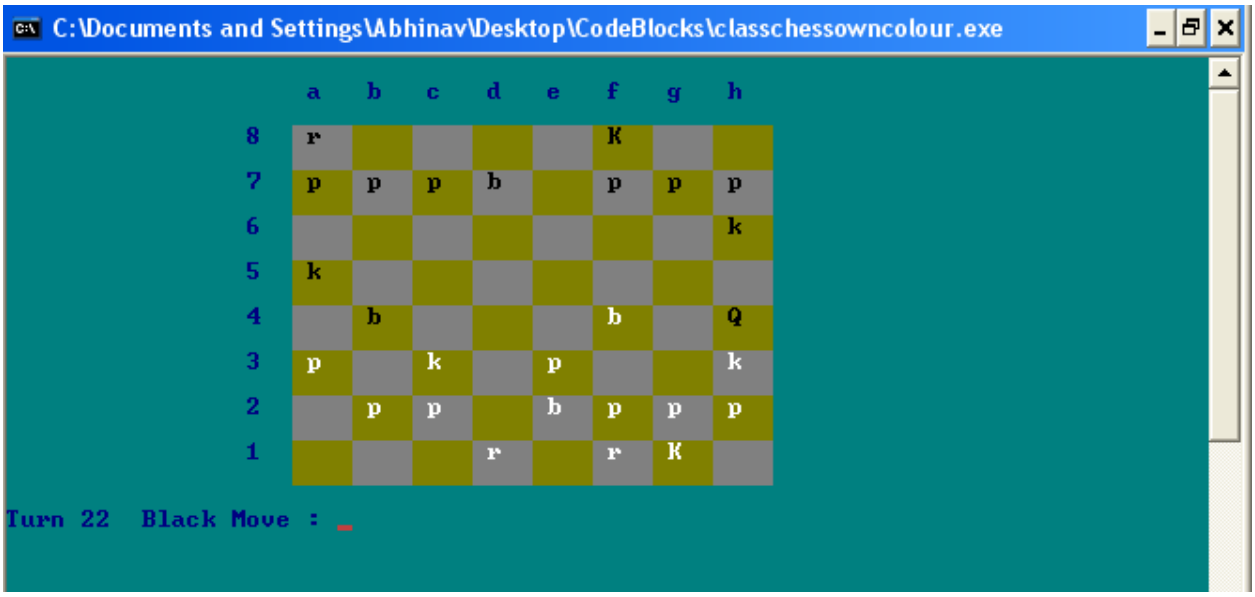

KNIGHT MOVE




ROOK MOVE




CASTLING 1


C:\Documents and Settings\Abhinav\Desktop\CodeBlocks\classchessowncolour.exe
a b c d e f g h
8 r K
7 p p p b p p p
6 k
5 k
4 b b Q
3 p k p k
2 p p b p p p
1 r r K
Turn 22 Black Move :

CASTLING 2