#include "graphics.h" ***//To use the inbuilt graphics function***

#include <iostream>

#include<thread>

#include<time.h>

#include<stdlib.h>

using namespace std;

class Board{

public:

bool Check\_Up(char);

bool Check\_Down(char);

bool Check\_Left(char);

bool Check\_Right(char);

bool Is\_Board\_Solved();

void Initialize\_Board();

void Scramble\_Board();

void Move\_Up(char move);

void Move\_Down(char move);

void Move\_Left(char move);

void Move\_Right(char move);

void Display\_Board();

void Refresh\_Board();

Board();

char move;

bool move\_up, move\_down, move\_right, move\_left;

int row, column, emp\_row, emp\_column;

int board[3][3], solved\_board[3][3];

int counter;

}Brd;

**//================================================================//**

**//------------------------------------Constructor for the class Board-----------------------------------//**

Board:: Board()

{

row= 0;

column= 0;

emp\_row=0;

emp\_column=0;

counter= 0;

move\_up= true;

move\_down= true;

move\_left= true;

move\_right= true;

for(row= 0; row<=2; row++)

{

for(column= 0; column<=2; column++)

board[row][column]= 0;

}

}

**//================================================================//**

**//-----------------------------Function to initialize the board with values--------------------------//**

void Board:: Initialize\_Board()

{

int i=0;

for(row= 0; row<=2; row++)

{

for(column= 0; column<=2; column++)

{

board[row][column]= i;

solved\_board[row][column]= (i+1);

i++;

}

}

solved\_board[2][2]=0;

}

**//================================================================//**

**//------------------------------------Function to scramble the board-----------------------------------//**

void Board:: Scramble\_Board()

{

time\_t t;

int rnd\_num, rnd\_num2, temp=0;

srand((unsigned) time(&t));

do{

rnd\_num= rand()%3;

rnd\_num2=rand()%3;

for(row=0;row<=2;row++)

{

for(column=0;column<=2;column++)

{

if(column==rnd\_num)

{

temp= board[row][column];

board[row][column]= board[row][rnd\_num2];

board[row][rnd\_num2]= temp;

}

}

}

for(column=0;column<=2;column++)

{

for(row=0;row<=2;row++)

{

if(row==rnd\_num)

{

temp= board[row][column];

board[row][column]= board[rnd\_num2][column];

board[rnd\_num2][column]= temp;

}

}

}

}while(Is\_Board\_Solved());

for(row= 0; row<=2; row++)

{

for(column= 0; column<=2; column++)

{

if(board[row][column]==0)

{

emp\_row= row;

emp\_column= column;

}

}

}

}

**//================================================================//**

**//------------------------------------------Function to check for victory---------------------------------//**

bool Board:: Is\_Board\_Solved()

{

bool board\_solved = false;

bool flag=FALSE; ***//To get game board status***

for(row=0; row<=2; row++)

{

for(column=0; column<=2; column++)

{

if(board[row][column]!=solved\_board[row][column])

{flag=TRUE;break; }

}

if(flag==TRUE)

break;

}

if(flag==FALSE)

board\_solved=TRUE;

return board\_solved;

}

**//================================================================//**

**//----------------------------Function to check for valid move upwards-----------------------------//**

bool Board:: Check\_Up(char move)

{

if(emp\_row==2 && (move== 'w' || move== 'W'))

move\_up= false;

else

move\_up=true;

return move\_up;

}

**//================================================================//**

**//----------------------------Function to check for valid move downwards-------------------------//**

bool Board:: Check\_Down(char move)

{

if(emp\_row==0 && (move== 's' || move== 'S'))

move\_down= false;

else

move\_down=true;

return move\_down;

}

**//================================================================//**

**//---------------------------Function to check for valid move rightwards---------------------------//**

bool Board:: Check\_Right(char move)

{

if(emp\_column==0 && (move== 'd' || move== 'D'))

move\_right= false;

else

move\_right= true;

return move\_right;

}

**//================================================================//**

**//----------------------------Function to check for valid move leftwards----------------------------//**

bool Board:: Check\_Left(char move)

{

if(emp\_column==2 && (move== 'a' || move== 'A'))

move\_left= false;

else

move\_left= true;

return move\_left;

}

**//================================================================//**

**//----------------------------------------Function to move upwards-------------------------------------//**

void Board:: Move\_Up(char move)

{

int temp;

while (Check\_Up(move) && counter<1)

{

temp = board[emp\_row][emp\_column];

board[emp\_row][emp\_column]= board[emp\_row+1][emp\_column];

board[emp\_row+1][emp\_column]= temp;

emp\_row+= 1;

counter++; //To limit the sliding of the tile to only once

Display\_Board();

}

counter=0;

}

**//================================================================//**

**//--------------------------------------*Function to move downwards*-------------------------------------//**

void Board:: Move\_Down(char move)

{

int temp;

while (Check\_Down(move) && counter<1)

{

temp = board[emp\_row][emp\_column];

board[emp\_row][emp\_column]= board[emp\_row-1][emp\_column];

board[emp\_row-1][emp\_column]= temp;

emp\_row-= 1;

counter++; //To limit the sliding of the tile to only once

Display\_Board();

}

counter=0;

}

**//----------------------------------------Function to move leftwards------------------------------------//**

void Board:: Move\_Left(char move)

{

int temp;

while (Check\_Left(move) && counter<1)

{

temp = board[emp\_row][emp\_column];

board[emp\_row][emp\_column]= board[emp\_row][emp\_column+1];

board[emp\_row][emp\_column+1]= temp;

emp\_column+= 1;

counter++; //To limit the sliding of the tile to only once

Display\_Board();

}

counter=0;

}

**//================================================================//**

**//-----------------------------------Function to move rightwards---------------------------------------//**

void Board:: Move\_Right(char move)

{

int temp;

while (Check\_Right(move) && counter<1)

{

temp = board[emp\_row][emp\_column];

board[emp\_row][emp\_column]= board[emp\_row][emp\_column-1];

board[emp\_row][emp\_column-1]= temp;

emp\_column-= 1;

counter++; //To limit the sliding of the tile to only once

Display\_Board();

}

counter=0;

}

**//================================================================//**

**//--------------------------------------Function to refresh the board------------------------------------//**

void Board:: Refresh\_Board()

{

cleardevice();

int buf[]={95,35,

730,35,//l

730,405,//l

95,405,

95,395,

720,395,//l

720,45,//l

105,45,

105,395,

95,395,

95,35

};

//rectangle(105,45,695,395);

line(525,45,525,395);

rectangle(255,45,380,70);

rectangle(125,80,505,385);

line(525,80,720,80);

fillpoly(11,buf);

settextstyle(10,0,3);

outtextxy(580,55,"OPTIONS");

settextstyle(COMPLEX\_FONT,0,1);

outtextxy(535,109," 1.RESTART LEVEL");

outtextxy(535,142," 2.PAUSE / PLAY");

outtextxy(530,160," (Press Spacebar)");

outtextxy(535,188," 3.EXIT GAME");

rectangle(145,85,245,175);

rectangle(265,85,365,175);

rectangle(385,85,485,175);

rectangle(145,185,245,275);

rectangle(265,185,365,275);

rectangle(385,185,485,275);

rectangle(145,285,245,375);

rectangle(265,285,365,375);

rectangle(385,285,485,375);

}

**//================================================================//**

**//----------------------Function to display the refreshed board every time-----------------------//**

void Board:: Display\_Board()

{

settextstyle(9,0,3);

if(board[0][0]==0)

{

bgiout<<" ";

outstreamxy(195,120);

}

else

{

bgiout<<board[0][0];

outstreamxy(195,120);

}

if(board[0][1]==0)

{

bgiout<<" ";

outstreamxy(315,120);

}

else

{

bgiout<<board[0][1];

outstreamxy(315,120);

}

if(board[0][2]==0)

{

bgiout<<" ";

outstreamxy(435,120);

}

else

{

bgiout<<board[0][2];

outstreamxy(435,120);

}

if(board[1][0]==0)

{

bgiout<<" ";

outstreamxy(195,220);

}

else

{

bgiout<<board[1][0];

outstreamxy(195,220);

}

if(board[1][1]==0)

{

bgiout<<" ";

outstreamxy(315,220);

}

else

{

bgiout<<board[1][1];

outstreamxy(315,220);

}

if(board[1][2]==0)

{

bgiout<<" ";

outstreamxy(435,220);

}

else

{

bgiout<<board[1][2];

outstreamxy(435,220);

}

if(board[2][0]==0)

{

bgiout<<" ";

outstreamxy(195,320);

}

else

{

bgiout<<board[2][0];

outstreamxy(195,320);

}

if(board[2][1]==0)

{

bgiout<<" ";

outstreamxy(315,320);

}

else

{

bgiout<<board[2][1];

outstreamxy(315,320);

}

if(board[2][2]==0)

{

bgiout<<" ";

outstreamxy(435,320);

}

else

{

bgiout<<board[2][2];

outstreamxy(435,320);

}

settextstyle(11,0,0);

}

**//================================================================//**