**Nawal**

**23F-0776**

**BSCS-2B**

**24-Puzzle Game**

#include<iostream>

#include<conio.h>

#include<cstdlib>

#include <algorithm>

#include<ctime>

#include<string>

#include<fstream>

using namespace std;

void init(int initial[5][5]);

void goal(int goal[5][5]);

bool IsSolveable(int solve[5][5]);

bool is\_goal(int current[5][5], int goal[5][5]);

void init(int initial[5][5])

{

cout << "\t\t\t\t\t\t\t\t\t\t\t" << endl;

cout << "\t\t\t\t\t" << "Welcome! to the 24-Puzzle Game :)" << "\t\t\t\t" << endl;

cout << "\t\t\t\t\t\t\t\t\t\t\t" << endl;

cout << "\t " << "-------------------------------------" << "\t\t\t\t" << endl;

srand(time(0));

// Generate shuffled numbers from 1 to 24

int num[24];

for (int i = 0; i < 24; ++i)

{

num[i] = i + 1;

}

random\_shuffle(num, num + 24);

// Assign shuffled numbers to grid cells

int index = 0;

int emptyRow = rand() % 5; // Randomly select a row

int emptyCol = rand() % 5; // Randomly select a column

for (int i = 0; i < 5; ++i)

{

for (int j = 0; j < 5; ++j)

{

if (i == emptyRow && j == emptyCol)

{

initial[i][j] = 0; // Leave this cell empty

}

else

{

initial[i][j] = num[index++];

}

}

}

// Display the grid

for (int i = 0; i < 5; ++i)

{

for (int j = 0; j < 5; ++j)

{

if (initial[i][j] == 0)

{

cout << "\t" << " |" << " " << " | "; // Display empty cell

}

else

{

cout << "\t" << " |" << initial[i][j] << " | ";

}

}

cout << endl;

cout << "\t " << "-------------------------------------" << endl;

}

cout << "\t\t\t" << "Initial State" << endl;

}

void goal(int goals[5][5])

{

cout << "\t\t\t\t\t\t\t\t\t\t\t" << endl;

srand(time(0));

cout << "\t " << "-------------------------------------" << endl;

int num = 1;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (i == 4 && j == 4)

{

cout << "\t" << " |" << " " << " |";

goals[i][j] = 0;

}

else

{

goals[i][j] = num++;

cout << "\t" << " |" << goals[i][j] << " |";

}

}

cout << endl;

cout << "\t " << "-------------------------------------" << endl;

}

cout << "\t\t\t" << "Goal State" << endl;

}

bool isSolvable(int solve[5][5])

{

int inversions = 0;

int flatArr[5 \* 5];

int num = 0;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (solve[i][j] != 0)

{

flatArr[num++] = solve[i][j];

}

}

}

for (int i = 0; i < 5 \* 5 - 1; i++)

{

for (int j = i + 1; j < 5 \* 5; j++)

{

if (flatArr[i] > flatArr[j])

{

inversions++;

}

}

}

if (5 % 2 == 1)

{

return inversions % 2 == 0;

}

}

bool is\_goal(int initial[5][5], int goal[5][5])

{

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (initial[i][j] != goal[i][j])

{

return false;

}

}

}

return true;

}

int\* legal\_moves(int\* init\_state)

{

int empty\_row = -1, empty\_col = -1;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (init\_state[i \* 5 + j] == 0)

{

empty\_row = i;

empty\_col = j;

break;

}

}

}

int\* moves = new int[4]; // Up, Down, Left, Right

for (int i = 0; i < 4; i++)

{

moves[i] = 0;

}

if (empty\_row > 0) {

moves[0] = 1; // Up

}

if (empty\_row < 4) {

moves[1] = 1; // Down

}

if (empty\_col > 0) {

moves[2] = 1; // Left

}

if (empty\_col < 4) {

moves[3] = 1; // Right

}

return moves;

}

void make\_move(int(&init\_state)[5][5], char move)

{

int empty\_row = -1, empty\_col = -1;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (init\_state[i][j] == 0)

{

empty\_row = i;

empty\_col = j;

break;

}

}

}

switch (move)

{

case 'U': // Move up

if (empty\_row > 0)

{

swap(init\_state[empty\_row][empty\_col], init\_state[empty\_row - 1][empty\_col]);

empty\_row--;

}

break;

case 'D': // Move down

if (empty\_row < 4)

{

swap(init\_state[empty\_row][empty\_col], init\_state[empty\_row + 1][empty\_col]);

empty\_row++;

}

break;

case 'L': // Move left

if (empty\_col > 0)

{

swap(init\_state[empty\_row][empty\_col], init\_state[empty\_row][empty\_col - 1]);

empty\_col--;

}

break;

case 'R': // Move right

if (empty\_col < 4)

{

swap(init\_state[empty\_row][empty\_col], init\_state[empty\_row][empty\_col + 1]);

empty\_col++;

}

break;

}

}

void print\_path(const char moves[], int num\_moves)

{

cout << "Moves taken to reach the goal state:" << endl;

for (int i = 0; i < num\_moves; ++i)

{

cout << moves[i] << " ";

}

cout << endl;

}

int main()

{

char moves[100];

int initial[5][5];

int goals[5][5];

init(initial);

cout << "\t\t";

goal(goals);

cout << endl;

int num\_moves = 0;

int empty\_row = 0, empty\_col = 0;

if (isSolvable(initial))

{

cout << "\t\t\t\t\t" << "Puzzle is solvable!" << endl;

}

else

{

cout << "\t\t\t\t\t" << "Puzzle is not solvable." << endl;

}

ofstream outFile("moves.txt"); // Open file for writing moves

int\* Moves = legal\_moves(initial[0]);

cout << "Legal moves:\n";

if (Moves[0] == 1) {

cout << "Up\n";

}

if (Moves[1] == 1) {

cout << "Down\n";

}

if (Moves[2] == 1) {

cout << "Left\n";

}

if (Moves[3] == 1) {

cout << "Right\n";

}

// Main game loop

while (!is\_goal(initial, goals))

{

char move;

bool valid\_move = false;

// Loop until a valid move is entered

while (!valid\_move)

{

cout << "\nEnter your move (U/D/L/R for Up/Down/Left/Right): ";

cin >> move;

// Validate the move

if (move == 'U' || move == 'D'|| move == 'L'|| move == 'R') {

valid\_move = true;

}

else {

cout << "Invalid move! Please enter a valid move from (U/D/L/R)." << endl;

}

make\_move(initial, move);

moves[num\_moves] = move;

++num\_moves;

outFile << move << " ";

}

cout << "\nUpdated State after moving " << move << ":" << endl;

init(initial);

}

print\_path(moves, num\_moves);

cout << "\nCongratulations! You solved the puzzle!" << endl;

system("pause");

return 0;

}









