VISHWAKARMA INSTITUTE OF TECHNOLOGY

COMPUTER ENGINEERING

Name: Abhinav Mahajan

Division: TY-C

Roll No: 15

Subject: Artificial Intelligence (AI)

LAB ASSIGNMENT NO – 2

Implementation of **Uninformed Strategies** for **8-Puzzle Game**.

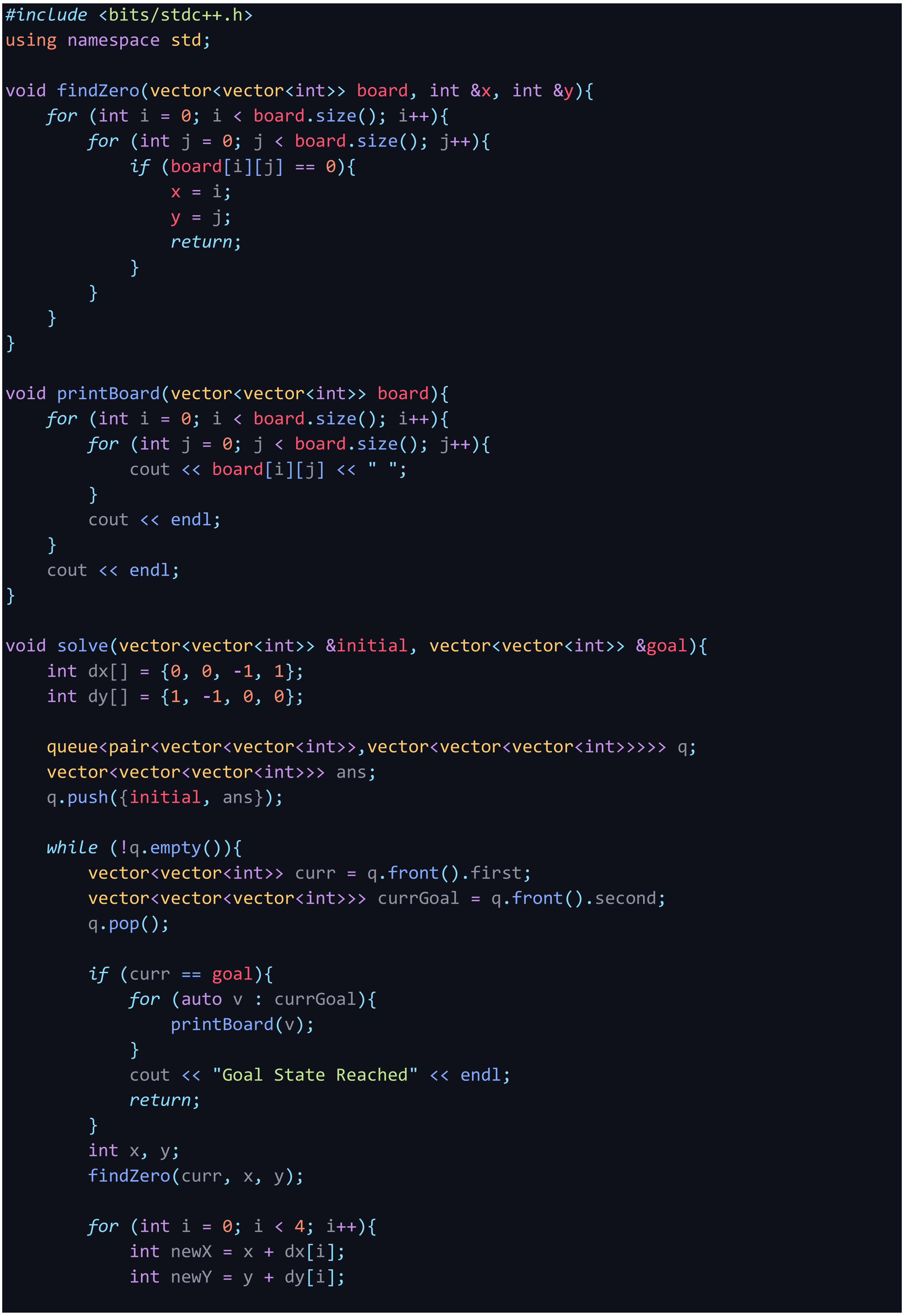
The 8-puzzle problem is a sliding puzzle that involves arranging 8 numbered tiles in a 3x3 grid with one empty space, which aims to reach a specific goal configuration through tile swaps.

1. Breadth-First Search

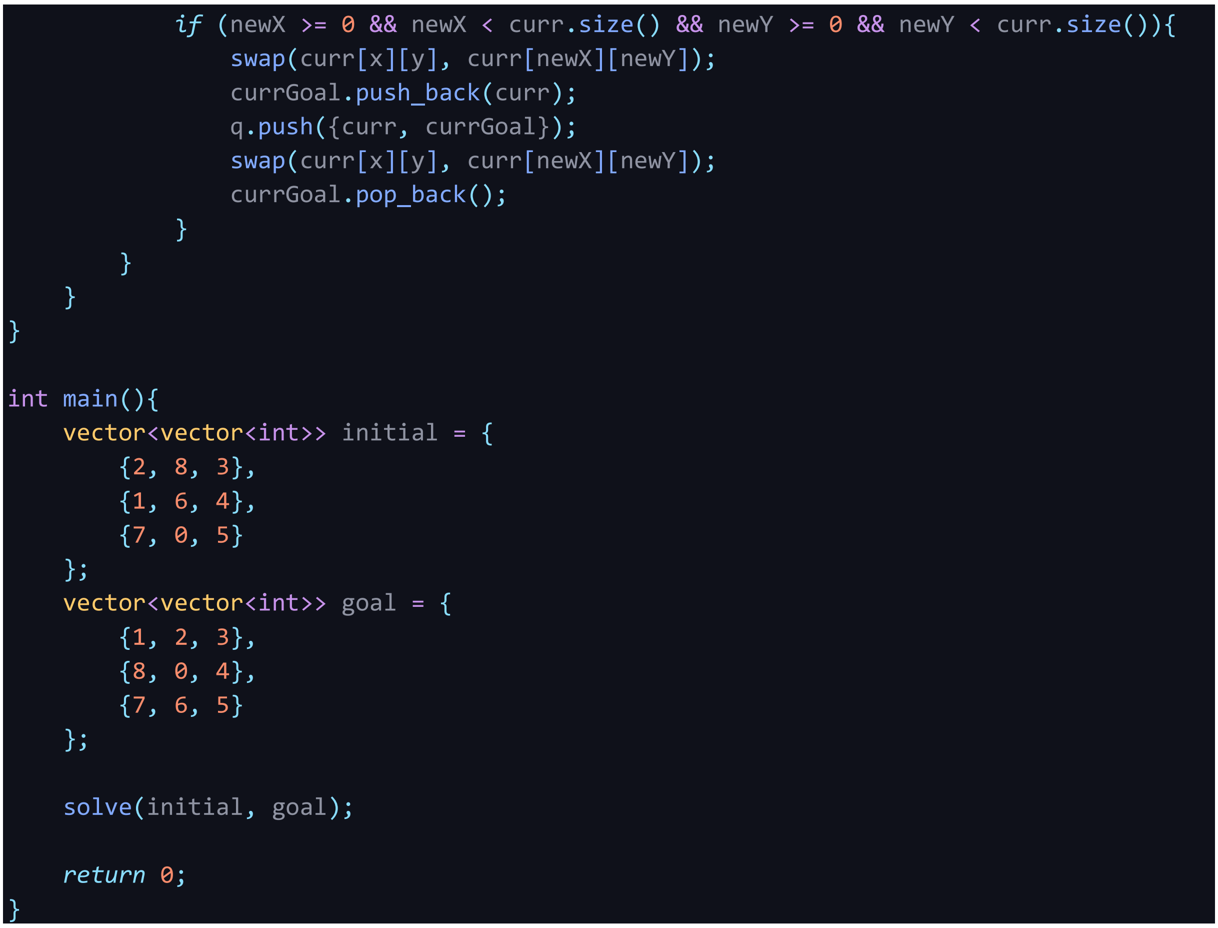
Approach:

1. Set up initial and goal states as 2D vectors.
2. Initialize a queue containing pairs of current state and its path.
3. Breadth-First Search (BFS): Start a loop with the queue. Pop current state and path.
4. Iterate over the four possible neighbour positions around the empty cell: up, down, left, and right
5. If current state matches goal state, print the path and stop.
6. Explore Neighbours: Find empty cell position, swap it with valid neighbours, and store resulting state and path in queue.
7. After exploring neighbours, backtrack by undoing swaps to explore other paths.
8. Continue this process for all possible paths, storing the unique paths in the vector.
9. When goal is unreachable or found, print paths leading to the goal state.

Code:

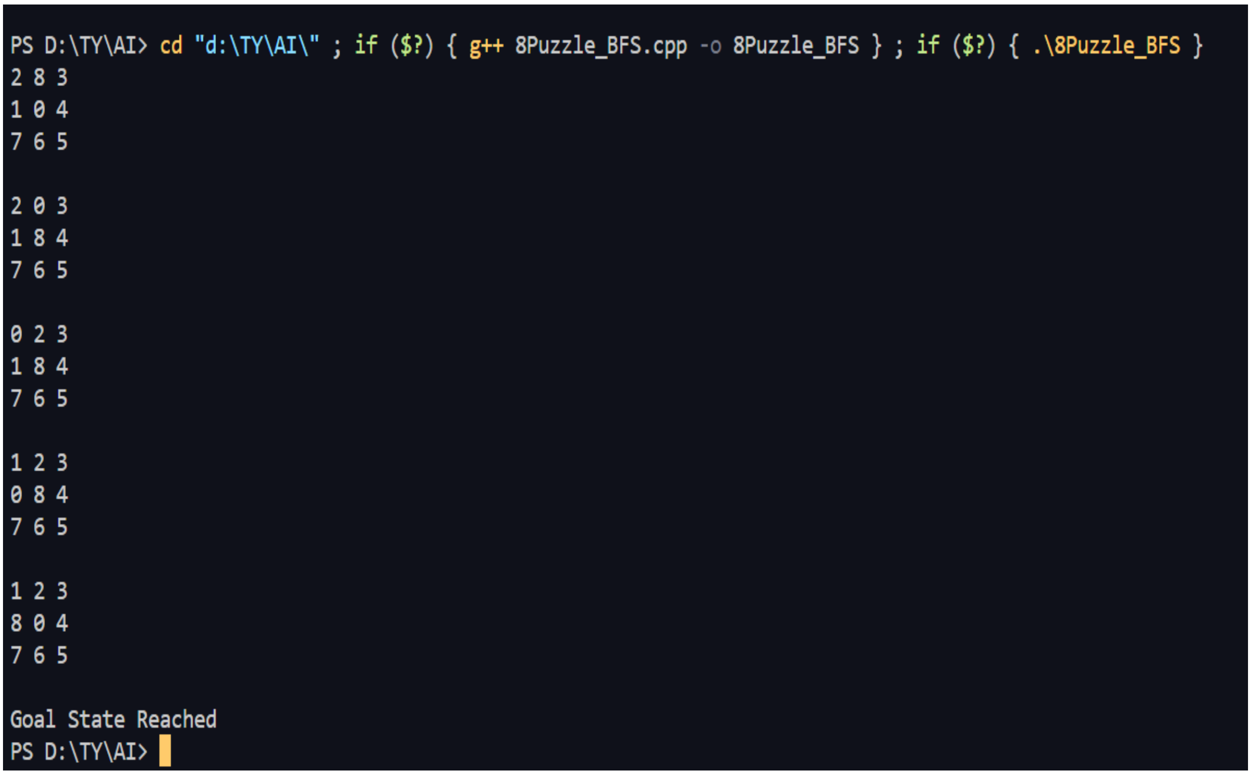


```cpp
#include <bits/stdc++.h>
using namespace std;

void findZero(vector<vector<int>> board, int &x, int &y){
    for (int i = 0; i < board.size(); i++){
        for (int j = 0; j < board.size(); j++){
            if (board[i][j] == 0){
                x = i;
                y = j;
                return;
            }
        }
    }
}

void printBoard(vector<vector<int>> board){
    for (int i = 0; i < board.size(); i++){
        for (int j = 0; j < board.size(); j++){
            cout << board[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

void solve(vector<vector<int>> &initial, vector<vector<int>> &goal){
    int dx[] = {0, 0, -1, 1};
    int dy[] = {1, -1, 0, 0};

    queue<pair<vector<vector<int>>,vector<vector<vector<int>>>>> q;
    vector<vector<vector<int>>> ans;
    q.push({initial, ans});

    while (!q.empty()){
        vector<vector<int>> curr = q.front().first;
        vector<vector<vector<int>>> currGoal = q.front().second;
        q.pop();

        if (curr == goal){
            for (auto v : currGoal){
                printBoard(v);
            }
            cout << "Goal State Reached" << endl;
            return;
        }
        int x, y;
        findZero(curr, x, y);

        for (int i = 0; i < 4; i++){
            int newX = x + dx[i];
            int newY = y + dy[i];
```



```
            if (newX >= 0 && newX < curr.size() && newY >= 0 && newY < curr.size()){
                swap(curr[x][y], curr[newX][newY]);
                currGoal.push_back(curr);
                q.push({curr, currGoal});
                swap(curr[x][y], curr[newX][newY]);
                currGoal.pop_back();
            }
        }
    }
}

int main(){
    vector<vector<int>> initial = {
        {2, 8, 3},
        {1, 6, 4},
        {7, 0, 5}
    };
    vector<vector<int>> goal = {
        {1, 2, 3},
        {8, 0, 4},
        {7, 6, 5}
    };

    solve(initial, goal);

    return 0;
}
```

Output:



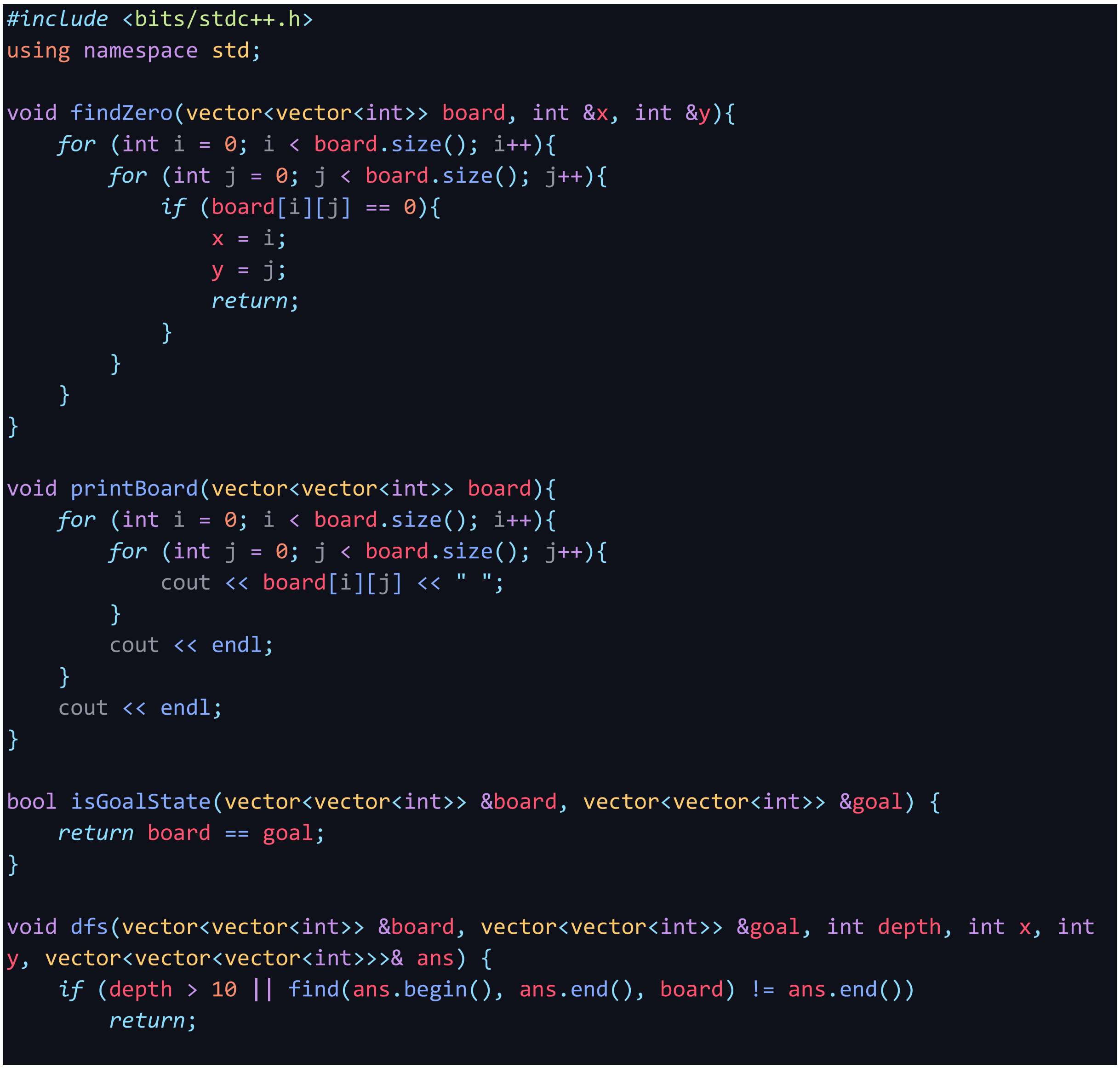
```
PS D:\TY\AI> cd "d:\TY\AI\" ; if ($?) { g++ 8Puzzle_BFS.cpp -o 8Puzzle_BFS } ; if ($?) { .\8Puzzle_BFS }
2 8 3
1 0 4
7 6 5

2 0 3
1 8 4
7 6 5

0 2 3
1 8 4
7 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5

Goal State Reached
PS D:\TY\AI> 
```

2. Depth-First Search

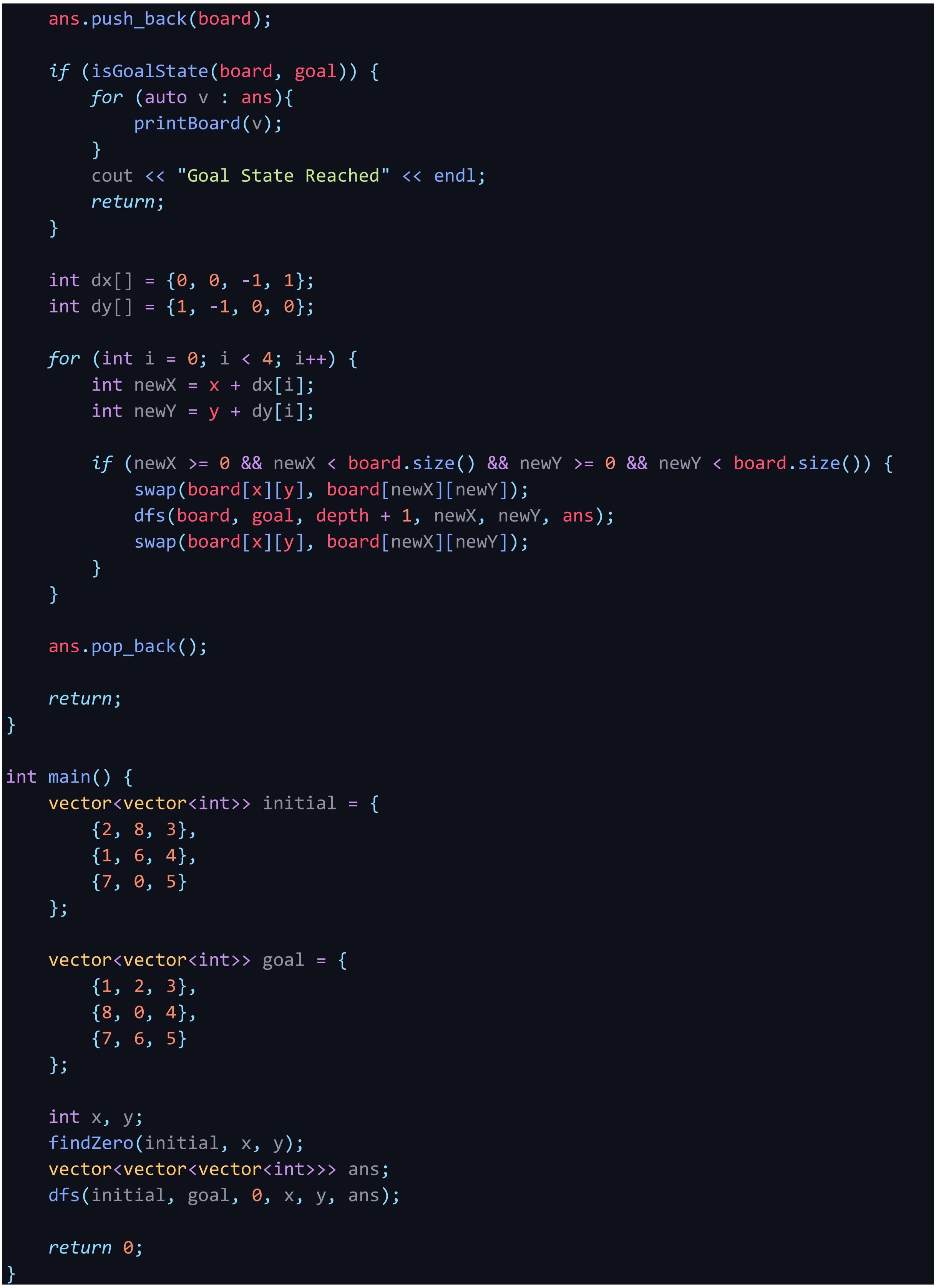
Approach:

1. Set up initial and goal states as 2D vectors.
2. Create a recursive function for DFS.
3. Pass the current state, goal state, depth, empty cell position, and path vector as arguments.
4. Check for base cases: maximum depth reached, out-of-bounds position, state already visited. If any, return.
5. Iterate over valid neighbour positions (up, down, left, right) around the empty cell. Swap, explore, and backtrack.

Code:

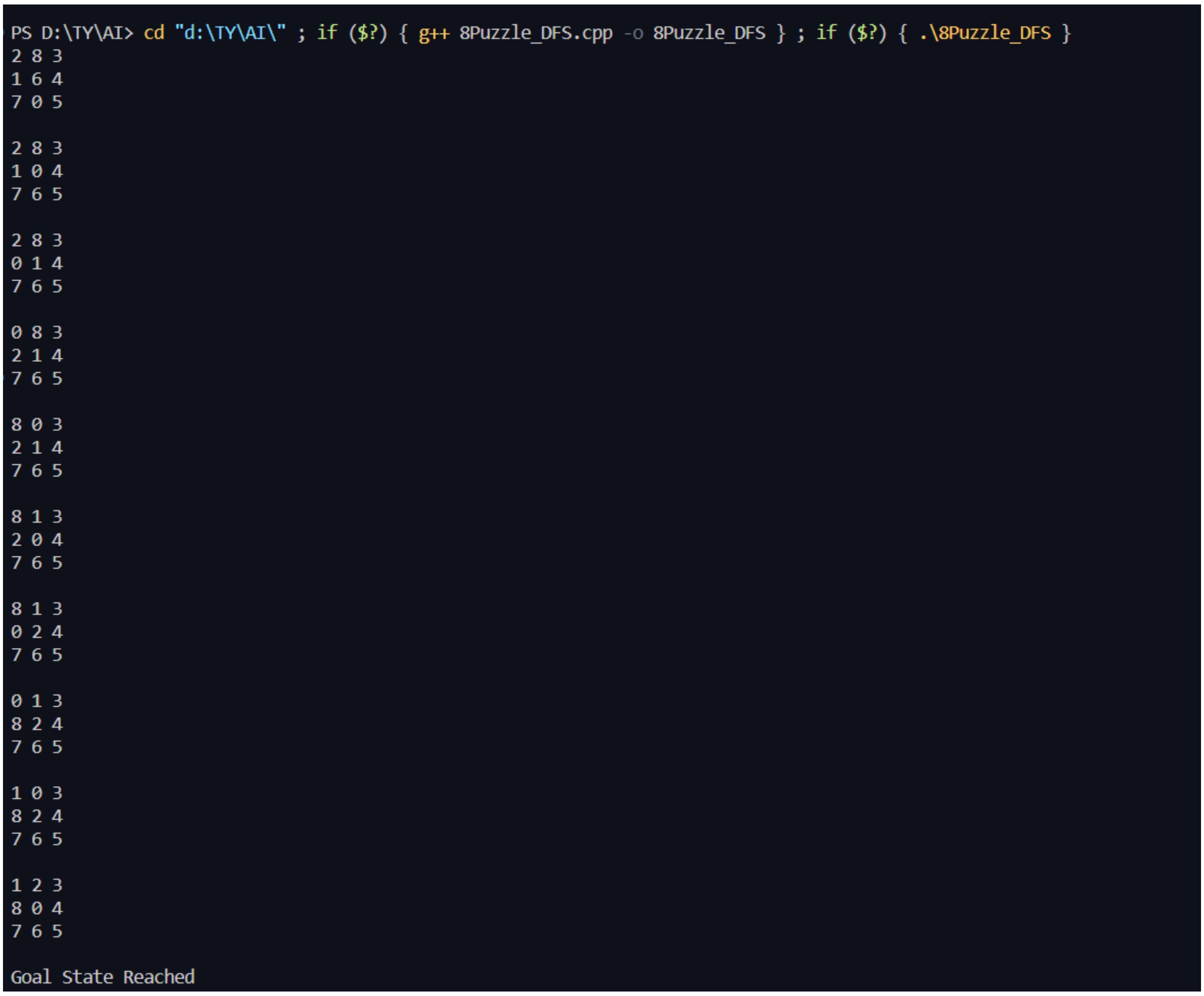


```cpp
#include <bits/stdc++.h>
using namespace std;

void findZero(vector<vector<int>> board, int &x, int &y){
    for (int i = 0; i < board.size(); i++){
        for (int j = 0; j < board.size(); j++){
            if (board[i][j] == 0){
                x = i;
                y = j;
                return;
            }
        }
    }
}

void printBoard(vector<vector<int>> board){
    for (int i = 0; i < board.size(); i++){
        for (int j = 0; j < board.size(); j++){
            cout << board[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

bool isGoalState(vector<vector<int>> &board, vector<vector<int>> &goal) {
    return board == goal;
}

void dfs(vector<vector<int>> &board, vector<vector<int>> &goal, int depth, int x, int
y, vector<vector<vector<int>>>& ans) {
    if (depth > 10 || find(ans.begin(), ans.end(), board) != ans.end())
        return;
```



```cpp
    ans.push_back(board);

    if (isGoalState(board, goal)) {
        for (auto v : ans){
            printBoard(v);
        }
        cout << "Goal State Reached" << endl;
        return;
    }

    int dx[] = {0, 0, -1, 1};
    int dy[] = {1, -1, 0, 0};

    for (int i = 0; i < 4; i++) {
        int newX = x + dx[i];
        int newY = y + dy[i];

        if (newX >= 0 && newX < board.size() && newY >= 0 && newY < board.size()) {
            swap(board[x][y], board[newX][newY]);
            dfs(board, goal, depth + 1, newX, newY, ans);
            swap(board[x][y], board[newX][newY]);
        }
    }

    ans.pop_back();

    return;
}

int main() {
    vector<vector<int>> initial = {
        {2, 8, 3},
        {1, 6, 4},
        {7, 0, 5}
    };

    vector<vector<int>> goal = {
        {1, 2, 3},
        {8, 0, 4},
        {7, 6, 5}
    };

    int x, y;
    findZero(initial, x, y);
    vector<vector<vector<int>>> ans;
    dfs(initial, goal, 0, x, y, ans);

    return 0;
}
```

Output:



```
PS D:\TY\AI> cd "d:\TY\AI\" ; if ($?) { g++ 8Puzzle_DFS.cpp -o 8Puzzle_DFS } ; if ($?) { .\8Puzzle_DFS }
2 8 3
1 6 4
7 0 5

2 8 3
1 0 4
7 6 5

2 8 3
0 1 4
7 6 5

0 8 3
2 1 4
7 6 5

8 0 3
2 1 4
7 6 5

8 1 3
2 0 4
7 6 5

8 1 3
0 2 4
7 6 5

0 1 3
8 2 4
7 6 5

1 0 3
8 2 4
7 6 5

1 2 3
8 0 4
7 6 5

Goal State Reached
```