#include <iostream>

#include <vector>

#include <queue>

#include <cmath>

#include <algorithm> // For std::reverse

using namespace std;

const int mapWidth = 10;

const int mapHeight = 10;

int playerX = 2;

int playerY = 2;

const int goalX = 9;

const int goalY = 0;

vector<vector<char>> map = {

{'.', '.', '.', '.', '.', '.', '.', '.', '.', 'G'},

{'.', 'X', '.', 'X', '.', '.', 'X', '.', 'X', '.'},

{'.', 'X', 'P', 'X', '.', '.', 'X', '.', 'X', '.'},

{'.', '.', '.', '.', '.', '.', '.', '.', '.', '.'},

{'.', 'X', '.', 'X', '.', '.', 'X', '.', 'X', '.'},

{'.', '.', '.', '.', '.', '.', '.', '.', '.', '.'},

{'.', 'X', '.', 'X', '.', '.', 'X', '.', 'X', '.'},

{'.', '.', '.', '.', '.', '.', '.', '.', '.', '.'},

{'.', 'X', '.', 'X', '.', '.', 'X', '.', 'X', '.'},

{'.', '.', '.', '.', '.', '.', '.', '.', '.', '.'},

};

struct Node {

int x, y;

float g, h;

Node\* parent;

Node(int x, int y, float g, float h, Node\* parent) : x(x), y(y), g(g), h(h), parent(parent) {}

float f() const {

return g + h;

}

bool operator>(const Node& other) const {

return f() > other.f();

}

};

float heuristic(int x1, int y1, int x2, int y2) {

return abs(x1 - x2) + abs(y1 - y2);

}

vector<pair<int, int>> get\_neighbors(int x, int y) {

vector<pair<int, int>> neighbors;

if (x > 0) neighbors.push\_back({x - 1, y});

if (x < mapWidth - 1) neighbors.push\_back({x + 1, y});

if (y > 0) neighbors.push\_back({x, y - 1});

if (y < mapHeight - 1) neighbors.push\_back({x, y + 1});

return neighbors;

}

vector<pair<int, int>> a\_star(int startX, int startY, int goalX, int goalY) {

priority\_queue<Node, vector<Node>, greater<Node>> openSet;

openSet.emplace(startX, startY, 0, heuristic(startX, startY, goalX, goalY), nullptr);

vector<vector<bool>> closedSet(mapHeight, vector<bool>(mapWidth, false));

while (!openSet.empty()) {

Node current = openSet.top();

openSet.pop();

if (current.x == goalX && current.y == goalY) {

vector<pair<int, int>> path;

for (Node\* node = &current; node != nullptr; node = node->parent) {

path.emplace\_back(node->x, node->y);

}

reverse(path.begin(), path.end());

return path;

}

closedSet[current.y][current.x] = true;

for (const auto& neighbor : get\_neighbors(current.x, current.y)) {

int nx = neighbor.first;

int ny = neighbor.second;

if (closedSet[ny][nx] || map[ny][nx] == 'X') continue;

float g = current.g + 1;

float h = heuristic(nx, ny, goalX, goalY);

openSet.emplace(nx, ny, g, h, new Node(current));

}

}

return {};

}

// Function to display the map and distance to the goal

void displayMap() {

for (int y = 0; y < mapHeight; y++) {

for (int x = 0; x < mapWidth; x++) {

if (x == playerX && y == playerY) {

cout << "P "; // Player's position

} else {

cout << map[y][x] << " "; // Map content

}

}

cout << endl;

}

// Calculate and display the distance to the goal

float distance = heuristic(playerX, playerY, goalX, goalY);

cout << "Distance to goal: " << distance << endl;

}

bool isValidMove(int newX, int newY) {

if (newX < 0 || newX >= mapWidth || newY < 0 || newY >= mapHeight) {

return false;

}

if (map[newY][newX] == 'X') {

return false;

}

return true;

}

int main() {

char move;

bool quit = false;

bool goalReached = false;

vector<pair<int, int>> path;

while (!quit && !goalReached) {

displayMap();

if (path.empty()) {

path = a\_star(playerX, playerY, goalX, goalY); // Find path to goal (G)

if (path.empty()) {

cout << "No path found!" << endl;

return 1;

}

}

cout << "Enter a move (WASD to move, Q to quit): ";

cin >> move;

int newX = playerX;

int newY = playerY;

switch (move) {

case 'W': case 'w': newY--; break;

case 'S': case 's': newY++; break;

case 'A': case 'a': newX--; break;

case 'D': case 'd': newX++; break;

case 'Q': case 'q': quit = true; break;

default: cout << "Invalid input. Use WASD to move or Q to quit." << endl;

}

if (isValidMove(newX, newY)) {

playerX = newX;

playerY = newY;

if (map[playerY][playerX] == 'G') {

goalReached = true;

cout << "Congratulations! You've reached the goal!" << endl;

}

} else {

cout << "Invalid move. There's an obstacle there!" << endl;

}

}

cout << "Goodbye!" << endl;

return 0;

}