```
#include <iostream>
#include <vector>
#include <queue>
#include <ctime>
#include <cstdlib>

using namespace std;

const char WALL = 'X';
const char PATH = ' ';
const char START = '@';
const char END = '*';
const char SOLUTION = '.';


struct Cell {
    int x, y;
};

vector<vector<char>> generateMaze(int rows, int cols, int density, Cell &start, Cell &end) {
    vector<vector<char>> maze(rows, vector<char>(cols, WALL));

    srand(time(0));

    // Fill inner area with random walls and paths
    for (int i = 1; i < rows - 1; ++i) {
        for (int j = 1; j < cols - 1; ++j) {
            maze[i][j] = (rand() % 100 < (100 - density)) ? PATH : WALL;
        }
    }

    start = {1, 1};
    end = {rows - 2, cols - 2};
    maze[start.x][start.y] = START;
    maze[end.x][end.y] = END;

    return maze;
}

bool bfsSolve(vector<vector<char>> &maze, Cell start, Cell end) {
    int rows = maze.size();
    int cols = maze[0].size();
    vector<vector<bool>> visited(rows, vector<bool>(cols, false));
    vector<vector<Cell>> parent(rows, vector<Cell>(cols, {-1, -1}));
    queue<Cell> q;

    q.push(start);
    visited[start.x][start.y] = true;
    int dx[] = {1, -1, 0, 0};
    int dy[] = {0, 0, 1, -1};

    while (!q.empty()) {
        Cell curr = q.front();
        q.pop();

        if (curr.x == end.x && curr.y == end.y) {
            // Trace path back
            Cell p = parent[end.x][end.y];
            while (!(p.x == start.x && p.y == start.y)) {
                maze[p.x][p.y] = SOLUTION;
                p = parent[p.x][p.y];
            }
            return true;
        }

        for (int i = 0; i < 4; ++i) {
            int nx = curr.x + dx[i];
            int ny = curr.y + dy[i];

            if (nx >= 0 && nx < rows && ny >= 0 && ny < cols &&
                !visited[nx][ny] && (maze[nx][ny] == PATH || maze[nx][ny] == END)) {
                visited[nx][ny] = true;
                parent[nx][ny] = curr;
                q.push({nx, ny});
            }
        }
    }

    return false;
}

void printMaze(const vector<vector<char>> &maze) {
    for (const auto &row : maze) {
        for (char ch : row) {
            cout << ch;
        }
        cout << '\n';
    }
}

int main() {
    int rows, cols, density;
    Cell start, end;

    cout << "Maze Generator and Solver\n";
    cout << "Enter number of rows (5-50): ";
    cin >> rows;
    cout << "Enter number of columns (5-50): ";
    cin >> cols;
    cout << "Enter wall density (10-90%): ";
    cin >> density;

    if (rows < 5 || cols < 5 || rows > 50 || cols > 50 || density < 10 || density > 90) {
        cout << "Invalid input parameters!\n";
        return 1;
    }

    vector<vector<char>> maze = generateMaze(rows, cols, density, start, end);

    cout << "\nGenerated Maze:\n";
    printMaze(maze);

    cout << "\nSolving maze...\n";
    if (bfsSolve(maze, start, end)) {
        cout << "\nSolved Maze:\n";
        // Restore start and end markers
        maze[start.x][start.y] = START;
        maze[end.x][end.y] = END;
        printMaze(maze);
        cout << "\nSolution found! Path marked with '" << SOLUTION << "'\n";
    } else {
        cout << "\nNo solution found for this maze.\n";
    }

    return 0;
}
```