```
import java.util.*;

class Node {
    int x, y;
    double gCost, hCost;
    Node parent;
    boolean isWalkable;

    public Node(int x, int y, boolean isWalkable) {
        this.x = x;
        this.y = y;
        this.isWalkable = isWalkable;
    }

    public double getFCost() {
        return gCost + hCost;
    }


    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Node node = (Node) obj;
        return x == node.x && y == node.y;
    }


    public int hashCode() {
        return Objects.hash(x, y);
    }
}

class AStarPathfinder {
    private Node[][] grid;
    private int gridWidth, gridHeight;
    private Set<Node> openSet = new HashSet<>();
    private Set<Node> closedSet = new HashSet<>();
    private PriorityQueue<Node> priorityQueue;
```

```
public AStarPathfinder(int width, int height) {
    gridWidth = width;
    gridHeight = height;
    grid = new Node[width][height];

    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            grid[x][y] = new Node(x, y, true); // Assume all nodes are walkable initially
        }
    }

    priorityQueue = new PriorityQueue<>(Comparator.comparingDouble(Node::getFCost));
}

public void setWalkable(int x, int y, boolean isWalkable) {
    grid[x][y].isWalkable = isWalkable;
}

public List<Node> findPath(Node startNode, Node endNode) {
    openSet.clear();
    closedSet.clear();
    priorityQueue.clear();

    startNode.gCost = 0;
    startNode.hCost = heuristic(startNode, endNode);

    openSet.add(startNode);
    priorityQueue.add(startNode);

    while (!priorityQueue.isEmpty()) {
        Node currentNode = priorityQueue.poll();

        if (currentNode.equals(endNode)) {
            return constructPath(currentNode);
        }

        openSet.remove(currentNode);
        closedSet.add(currentNode);

        for (Node neighbor : getNeighbors(currentNode)) {
```

```
            if (!neighbor.isWalkable || closedSet.contains(neighbor)) {
                continue;
            }

            double tentativeGCost = currentNode.gCost + distance(currentNode, neighbor);

            if (tentativeGCost < neighbor.gCost || !openSet.contains(neighbor)) {
                neighbor.gCost = tentativeGCost;
                neighbor.hCost = heuristic(neighbor, endNode);
                neighbor.parent = currentNode;

                if (!openSet.contains(neighbor)) {
                    openSet.add(neighbor);
                    priorityQueue.add(neighbor);
                }
            }
        }
    }

    return null; // No path found
}

private double heuristic(Node a, Node b) {
    return Math.abs(a.x - b.x) + Math.abs(a.y - b.y); // Manhattan distance
}

private double distance(Node a, Node b) {
    return Math.hypot(a.x - b.x, a.y - b.y);
}

private List<Node> getNeighbors(Node node) {
    List<Node> neighbors = new ArrayList<>();

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

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

```
            if (newX >= 0 && newX < gridWidth && newY >= 0 && newY < gridHeight) {
                neighbors.add(grid[newX][newY]);
            }
        }

        return neighbors;
    }

    private List<Node> constructPath(Node endNode) {
        List<Node> path = new ArrayList<>();
        Node currentNode = endNode;

        while (currentNode != null) {
            path.add(currentNode);
            currentNode = currentNode.parent;
        }

        Collections.reverse(path);
        return path;
    }
}

public class Main {
    public static void main(String[] args) {
        AStarPathfinder pathfinder = new AStarPathfinder(10, 10);

        // Set some nodes as unwalkable (representing dynamic obstacles)
        pathfinder.setWalkable(3, 3, false);
        pathfinder.setWalkable(3, 4, false);
        pathfinder.setWalkable(3, 5, false);

        Node startNode = new Node(0, 0, true);
        Node endNode = new Node(7, 7, true);

        List<Node> path = pathfinder.findPath(startNode, endNode);

        if (path != null) {
            for (Node node : path) {
                System.out.println("Path: (" + node.x + ", " + node.y + ")");
            }
```

```
        } else {
            System.out.println("No path found");
        }
    }
}
```