Search and Simulation Algorithms in Java

Search and simulation algorithms tackle pathfinding and probabilistic problems. Let's explore A* Search and Monte Carlo Simulation.

1. A* Search: Heuristic Pathfinding

A* is an informed search algorithm that uses a heuristic to find the shortest path in a graph, combining the strengths of Dijkstra's and greedy search. It's widely used in games and navigation.

Java Implementation

```java
import java.util.*;

public class AStar {
    static class Node implements Comparable<Node> {
        int x, y, g, h, f;
        Node parent;
        Node(int x, int y) {
            this.x = x;
            this.y = y;
            this.g = 0;
            this.h = 0;
            this.f = 0;
        }
        public int compareTo(Node other) { return this.f - other.f; }
    }

    static int heuristic(int x1, int y1, int x2, int y2) {
        return Math.abs(x1 - x2) + Math.abs(y1 - y2); // Manhattan distance
    }

    static void aStarSearch(int[][] grid, int[] start, int[] goal) {
        int rows = grid.length, cols = grid[0].length;
        PriorityQueue<Node> open = new PriorityQueue<>();
        boolean[][] closed = new boolean[rows][cols];
        Node startNode = new Node(start[0], start[1]);
        Node goalNode = new Node(goal[0], goal[1]);
        startNode.h = heuristic(start[0], start[1], goal[0], goal[1]);
        startNode.f = startNode.h;
```

```
        open.add(startNode);

        // int[][] dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
        int[][] dirs = {};
        while (!open.isEmpty()) {
            Node current = open.poll();
            if (current.x == goal[0] && current.y == goal[1]) {
                printPath(current);
                return;
            }
            closed[current.x][current.y] = true;
            for (int[] dir : dirs) {
                int newX = current.x + dir[0], newY = current.y + dir[1];
                if (newX >= 0 && newX < rows && newY >= 0 && newY < cols && grid[newX][newY] != 1 && !closed[n
                    Node neighbor = new Node(newX, newY);
                    neighbor.g = current.g + 1;
                    neighbor.h = heuristic(newX, newY, goal[0], goal[1]);
                    neighbor.f = neighbor.g + neighbor.h;
                    neighbor.parent = current;
                    open.add(neighbor);
                }
            }
        }
        System.out.println("No path found!");
    }

    static void printPath(Node node) {
        List<int[]> path = new ArrayList<>();
        while (node != null) {
            path.add(new int[]{node.x, node.y});
            node = node.parent;
        }
        Collections.reverse(path);
        System.out.println("Path:");
        for (int[] p : path) System.out.println("(" + p[0] + ", " + p[1] + ")");
    }

    public static void main(String[] args) {
        int[][] grid = {
            {0, 0, 0, 0},
```

```
            {0, 1, 1, 0},
            {0, 0, 0, 0}
        };
        int[] start = {0, 0}, goal = {2, 3};
        aStarSearch(grid, start, goal);
    }
}
```

Output:

```
Path:
(0, 0)
(1, 0)
(2, 0)
(2, 1)
(2, 2)
(2, 3)
```

2. Monte Carlo Simulation: Probabilistic Estimation

Monte Carlo methods use random sampling to estimate results, like approximating π by simulating points in a square and circle.

Java Implementation

```
import java.util.Random;

public class MonteCarlo {
    static double estimatePi(int points) {
        Random rand = new Random();
        int insideCircle = 0;
        for (int i = 0; i < points; i++) {
            double x = rand.nextDouble();
            double y = rand.nextDouble();
            if (x * x + y * y <= 1) insideCircle++; // Inside unit circle
        }
        return 4.0 * insideCircle / points; // Ratio * 4 approximates
    }

    public static void main(String[] args) {
```

```
        int points = 1000000;
        double pi = estimatePi(points);
        System.out.println("Estimated   with " + points + " points: " + pi);
        System.out.println("Actual  : " + Math.PI);
    }
}
```

Output (varies due to randomness):

```
Estimated   with 1000000 points: 3.1418
Actual  : 3.141592653589793
```