Project 1: Solving Mazes with A*

Paul Beggs
GitHub Link

September 19, 2025

1 Heuristic Descriptions & Implementations

The Manhattan heuristic (Listing 1) calculates the Manhattan distance between the current location of the `MazeExplorer` and the end location of the maze. If we let n_1 and n_2 be the coordinates of the current location and e_1 and e_2 be the coordinates of the end location, then the Manhattan distance is calculated as $|n_1 - e_1| + |n_2 - e_2|$. This implementation is admissible because it never overestimates the true cost to reach the goal. Thus, the shortest path will never be larger than the Manhattan distance, as the shortest path may involve moving diagonally or taking a more direct route that is not captured by the Manhattan distance.

```java
public class Manhattan implements ToIntFunction<MazeExplorer> {
    @Override
    public int applyAsInt(MazeExplorer value) {
        return value.getLocation().getManhattanDist(value.getM().getEnd());
    }
}
```

Listing 1: Manhattan-Only Heuristic

The Combined Distance heuristic (Listing 2) works by starting with the current location of the `MazeExplorer`, the end location, and also initializes 3 sets of treasures: all treasures in the maze, treasures that have been found, and treasures that have not yet been found. Then, it begins by checking if there are any remaining treasures. If there are no remaining treasures, it returns the Manhattan distance to the end location. If there are remaining treasures, it calculates the minimum distance from the current location to any of the remaining treasures and the minimum distance from any of those treasures to the end location. Finally, it returns the sum of these two minimum distances.

```java
public class CombinedDist implements ToIntFunction<MazeExplorer> {
    @Override
    public int applyAsInt(MazeExplorer value) {
        Pos currentLocation = value.getLocation();
        Pos endLocation = value.getM().getEnd();

        Set<Pos> allTreasures = value.getAllTreasureFromMaze();
        Set<Pos> foundTreasures = value.getAllTreasureFound();

        Set<Pos> remainingTreasures = new HashSet<>(allTreasures);
        remainingTreasures.removeAll(foundTreasures);

        if (remainingTreasures.isEmpty()) {
            return currentLocation.getManhattanDist(endLocation);
        }

        int minDistFromCurrent = Integer.MAX_VALUE;
        int minDistToEnd = Integer.MAX_VALUE;

        for (Pos treasure : remainingTreasures) {
            minDistFromCurrent = Math.min(minDistFromCurrent,
             ↪  currentLocation.getManhattanDist(treasure));
            minDistToEnd = Math.min(minDistToEnd,
             ↪  treasure.getManhattanDist(endLocation));
        }
        return minDistFromCurrent + minDistToEnd;
    }
}
```

Listing 2: Combined Distance Heuristic

2 Experimentation and Analysis

```java
public class MaxDist implements ToIntFunction<MazeExplorer> {
    @Override
    public int applyAsInt(MazeExplorer value) {
        Pos currentLocation = value.getLocation();

        Set<Pos> allTreasures = value.getAllTreasureFromMaze();
        Set<Pos> foundTreasures = value.getAllTreasureFound();

        Set<Pos> remainingTreasures = new HashSet<>(allTreasures);
        remainingTreasures.removeAll(foundTreasures);

        int maxDist =
         ↪  currentLocation.getManhattanDist(value.getM().getEnd());

        for (Pos treasure : remainingTreasures) {
            int distToTreasure = currentLocation.getManhattanDist(treasure);
            if (distToTreasure > maxDist) {
                maxDist = distToTreasure;
            }
        }
        return maxDist;
    }
}
```

Listing 3: Max Distance Heuristic

```java
public class OverlyConfident implements ToIntFunction<MazeExplorer> {
    @Override
    public int applyAsInt(MazeExplorer value) {
        int manhattanDistance =
         ↪  value.getLocation().getManhattanDist(value.getM().getEnd());
        return manhattanDistance * 3 + 1;
    }
}

```

Listing 4: Overly Confident Heuristic