AI assignment 3: heuristics

Laurens De Wachter - 20214686

October 29, 2023

1 Corners problem

For the corners problem I used a heuristic that gives the manhattan distance to the furthest unvisited corner. This is admissible because this distance will always need to be traveled by pacman to reach the goal, therefore always being less or equal to the actual distance. The consistency of this heuristic is intuitive and follows the same reasoning as it's admissability. When writing this heuristic in code it looks like this:

```python
    return max(util.manhattanDistance(state[0], corner) for corner in
    unvisited_corners)
```

2 Food search problem

For the food search problem I used the following heuristic:
Take the longest possible real distance between two dots and call it **max_distance**
Now take the real distance between pacman and the closest of these two dots and call it **closer_food**
The heuristic is now simply **max_distance** + **closer_food**

This heuristic is admissible because you will always have to travel **max_distance** in the end and it is beter to travel the shortest distance first since you then have to backtrack the least distance. It is consistent since **closer_food** will always become smaller when moving closer to this dot.

In code the heuristic looks like this:

```python
    # If more than 2 food left, calculate the position of the 2 foods that are the
    furthest apart from each other
    max_distance = float("-inf")
    pos1, pos2 = None, None
    for i in range(len(food_list)):
        for j in range(i + 1, len(food_list)):
            distance = mazeDistance(
                food_list[i], food_list[j], problem.startingGameState
            )
            if distance > max_distance:
                max_distance = distance
                pos1, pos2 = food_list[i], food_list[j]

    # Calculate the distance the closest food and pacman
    closer_food = min(
        mazeDistance(pos1, position, problem.startingGameState),
        mazeDistance(pos2, position, problem.startingGameState),
    )

    # Return the max distance between the 2 foods and the distance to the closest food
    return max_distance + closer_food
```

This heuristic will not be able to solve *MediumSearch* wuickly because of the amount of calculations it must do a lot of calculations and the *mazeDistance* function is not super fast.

3 Suboptimal search

The suboptimal search uses the ClosestDotSearchAgent problem. This will of course not alway be the most optimal algorithm. Think of the following situation. The maze consists of a single line with a row of 10 dots starting directly left of pacman and a single dot 3 spaces to the right of pacman. Using this search it will first start eating the entire row and have to backtrack this row to eat the dot on the right. It would of course be far more optimal to first eat the dot on the right before eating the row of dots.