

Manhattan:

```
def __manhattan__(self, other):
    return sum(list(map(abs, self.__sub__(other))))
```

Manhattan max:

```
def __manhattan_max__(self, others):
    return max(list(map(self.__manhattan__, others)))
```



ii)

Manhattan Sum

```
def __manhattan_sum__(self, others):
    return sum(list(map(self.__manhattan__, others)))
```

Collected coin ids: 0, 1, 2, 3

Solution found!

Plan length: 108

States expanded: 3532

States visited: 3925

Total time: 69.873s

iii) Manhattan Ordered Sum:

```
def __manhattan_ordered_sum__(self, others):
    if len(others) <= 0:
        return 0
    ordered = sorted(others, key=lambda x: self.__manhattan__(x))
    total = self.__manhattan__(ordered[0])
    for index, val in enumerate(ordered[1:]):
        total += self.__manhattan__(ordered[index + 1])
    return total
```

```
graphical illustration of the initial state
of parameters:
heuristic_search.py den009d.map 20,10
must be compatible with Python 3!
heuristics make use of the Manhattan d
 $h_1(x',y') = |x-x'| + |y-y'|$ . Let
 $(x,y)$  denote its position on the grid.
 $s$  the agent's current position, and  $C' \subseteq C$ 
functions are formally defined as follows:
Manhattan Man
has not been collected. Formally,  $h_1(s) =$ 
 $|\text{col}(s) - \text{col}(c)|$  and  $h_2(s) =$ 
of the Manhattan distances to all c
 $h_2(s) = \sum_{c \in C'} d(s, c)$ .  $(x,y)$  is
Manhattan to the current li
2
1
for all  $1 \leq i \leq n$ , it holds that  $d(s, s_i) \leq d(s, c_i)$ .
Debug: heuristic_search
Control Frames/Variables
[Out[3]: <__main__.SearchNode at 0x7f055061fe48>
Out[4]: '<__main__.SearchNode object at 0x7f055061fe48>'
8
CHECKING LE, 9 11
Traceback (most recent call last):
Solution found!
Plan length: 128
States expanded: 3597
States visited: 3806
Total time: 74.098s
Collected coin ids: 0, 1, 2, 3
```

iv)

My Fancy Heuristic:

```
def __sub__(self, other):
    return self.x - other.x, self.y - other.y

def __l2_sq__(self, other):
    d = self.__sub__(other)
    sq = d[0] ** 2, d[1] ** 2
    return sum(sq)

def __l2_sq_min__(self, others):
    return min(list(map(self.__l2_sq__, others)))

def __l2_min_coins_penalty__(self, uncollected, all_coins, coins_collected, max_dist):
    if(len(uncollected)<=0):
        return 0
    return self.__l2_sq_min__(uncollected) + ((len(all_coins) - len(coins_collected)) *
max_dist)

Heuristic Call:
return agent_pos.__l2_min_coins_penalty__(uncollected, all_coins, collected,
                                            (state.grid.width ** 2 + state.grid.height **
2))
```

1)

Explanation for admissibility:

The heuristic is as follows:

Squared manhattan min (closest coin)+ num uncollected coins *max possible manhattan min distance

This shows that distance will never be over estimated as it is always at least:

num_coins * (grid.width**2+grid.height**2)

Hence it is impossible to underestimate the distance.

2)

Works arbitrarily on any state of this problem. Nothing is assumed.

3) Results: time: 12s, states visit: 915, states expand: 476: plan:124 (should be less)

Motivations: Provides penalty for states where coins are not collected, this way we just greedily collect the closest coin, and find a plan quickly, an identified plan can then be optimised. The tree could also be pruned to not re-explore states that have already been explored after collecting a coin, this would be faster as well.

```
Solution found!
Plan length:      124
States expanded:  476
States visited:   615
Total time:       11.630s
```

[illegible]