Assignment 3 Report

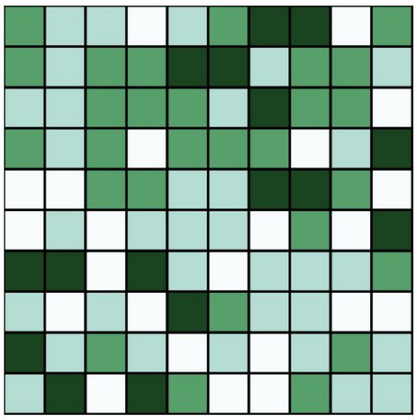
Ziqi Wang
Zekun Zhang
Richa Rai

The Problem

In assignment 3, we need to find the specific target hidden in a map. The map contains four different terrains: flat, hill, forest and caves. As the complexity of the given terrains is different, it is likely that we can't find the target in the complex terrains even the target is in this grid.

Implementation

Map initialize



[1]

We initial a map contains 4 different terrains. and randomly choose a target in a grid.

Stationary Search

Separate with the terrain map, we have another belief map. In each grid, it contains how much we believe the target is in this grid. While we searching through the terrain map, we will update the belief map at the same time.

Belief updating policy

After we search a grid Celli, it can contain two situation2:
1)Success.

[1] map with 4 terrains

The search process is over since we find the target.

2)Failure.

At this situation, we find a grid which may contains target or not, but we definately didn't find the target, so the intuition of this observation is target not in Celli or target in Celli but we didn't find it.

Observation = P(target not inCelli) + P(target in Cellibut not found)

P(target not in Celli) = 1 - P(target in Celli)

According to the conditional probability, we can use Bayesion formula to get this equation.

P(target in Celli but not found) = P(target in Celli) * P(Target not found in |Target is in Celli)

So after this observation, we can update the whole belief map, Cellj means other cells:

P(target in Celli) = P(target in Celli) * P(Target not found in |Target is in Celli) / Observation

P(target in Cellj) = P(target in Cellj) / Observation

Through this updating process, we can increase the possibility of finding the target.

Searching policy

In the searching part, the belief updating policy will keep the same, the only thing we concern is how to determine the next grid to search. According to the assignment, there are 2 basic stationary searching policy(Search Rule 1 & Search Rule 2).

1)Search Rule 1: searching grid has highest probability containing the target.

In this policy, we go to the cells which has highest belief, considering that there may have multiple highest value, we randomly choose one and search it.

2)Search Rule 2: searching grid which is most likely to find the target.

In this policy, we need also considering about the P(Target not found in |Target is in Celli). If it is low, we are less likely to go to this grid. So in the searching process, we give the simple terrain grid higher priority and more likely to search these grid. So in the implementation, we keep 4 lists containing 4 different terrains' grids, only if the simpler terrains's list is empty(don't have highest belief grids), we move to other lists and randomly pick one grid to search.

Stationary Search with Actions

On the basis of the searching policies above, if we need to consider about the cost of traveling from one grid to another, we call it searching with actions.

What is called an 'action' in the searching process?

In this part, determination of next grid to go is considered as one action.

The Manhattan distance x between the current grid and the destination grid is considered as x actions.

So the overall cost from traveling from Celli to Cellj is:

Cost = determination + Manhattan Distance(Celli, Cellj)

What is our goal in under this policy and how to achieve that?

The goal we want to achieve is to minimize the number of actions we take to find the target. In order to accomplish this goal, in each determination, we need to consider two aspects:

1) How likely we can find the target in the destination grid?

2)The Manhattan Distance between current grid and the destination grid should be minimize.

Based on that, we would like to choose these grid which is more likely to find the target, and the searching policy is slightly different from the former two. Under this searching with action policy, instead of randomly choosing a grid with highest possibility to find the target, we will choose the nearest grid to minimize the Manhattan distance.

Bonus Part

Preparing work

How to make target dynamic?

In this part, the target is no longer stationery and it would move to a neighbor cell every unit time. As we've mentioned, a neighbor cell is a cell which would be up, down, left or right. So, initially we should keep every movement of target is actually in the map. We choose to use a direction array: [[1, 0], [-1, 0], [0, 1], [0, -1]] and add a random element in this array to the current position of target. After every movement, we must keep the position of the target is valid or we would roll back and choose another direction. In this way, we make the target move randomly and keep every movement valid. And we invoke the target-move function after every search we make to make the target move in every unit time.

How to use the additional infomation?

Fortunately, in the bonus part, we still have a tracker on the taget. But there's still a problem that the tracker only can return a terrain type which the target isn't on. So, we utilize a funtion to locate the position of the target, and randomly return a terrain type except the one which the current target is on. Then, in the search funtion, we create a new copy of the possibility map and make all of values of the terrain cells which are the exact terrain type to zeros.

Searching Implementation

How to implement the searching?

After we have our utility functions, we are following these steps to implement our project.

1. Initially,we initialize the possiblity map and choose a target randomly.
2. Choose a grid with the Stationery Search Rule 2 in the possibility map.
3. Then, make the target move randomly, and get infomation from the tracker on the target.
4. Next, make a new copy of the possibility map and set zeros for the cells which couldn't contain the target and use the copy map to choose a grid with the Stationery Search Rule 3 we've mentioned.
5. Finally, modify the original possibility map with this observation.
6. Repeat step 3 ~ step 5 until we successfully find the target.

Evaluation

Algorithm Complexity

In this project, it’s no longer like the first two we’ve done, which is really easy for us to analyze the time complexity and space complexity. As far as we concerned, our project requires at least one N * N array, in this way the space complexity should be O(N^2). For the time complexity, it’s really hard to say because this is a possibility issue.

Result

For map 50 * 50, search times and actions:
https://github.com/zzkzzk1996/CS520/blob/master/Project3/data/r1_50.jpg
We’ve attach all of our test data in our project file as .xlsx.