Probabilistic Search (and Destroy)

CS 520 Assignment-3

Haotian Xu(hx105), Chaoji Zuo(cz296), Xuenan Wang(xw336)

Spring 2019

Probabilistic Search

CS 520 Assignment-3

1. A Stationary Target

A. Building the map:

Given a size n for the map, program is supposed to generate a **n*n** map. For each grid in the map, the probability of grid being a certain type of terrain is different. Using *random.random()* to generate a number for a grid, function determine the number in which interval and assign terrain type to the grid.

In this example of **50x50** size map, the **white** grids are flat areas; the **brown** grids are hills; the **green** grids are forests; the **black** grids are caves.

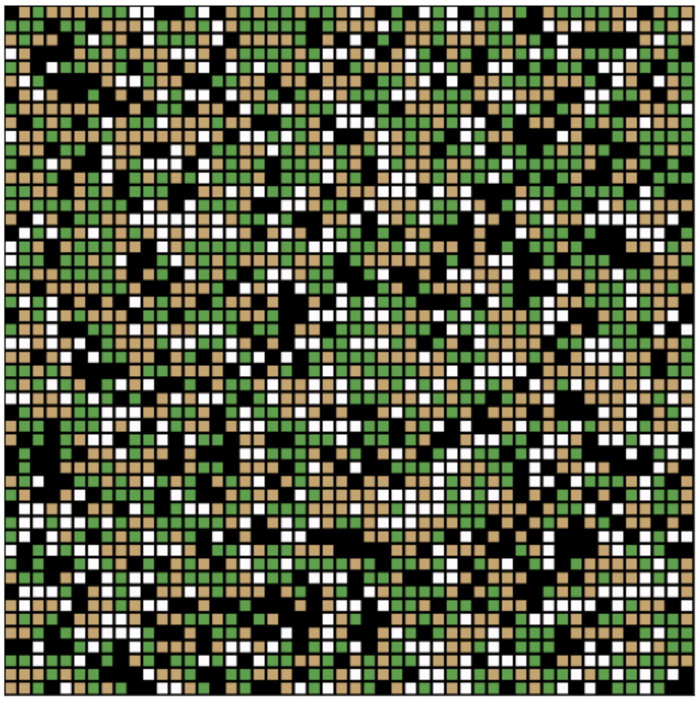

Figure 1. Example of Search Map

B. Formula simplification using Bayesian method (Rule 2):

At any time, search cells with the highest probability of finding the target.

Given the formula of each grid's belief (the probability of containing the target). These simplification also works in rule 1 and moving target.

$$Belief_{(t+1)}[i] = P(Target\ in\ Cell_i \mid Observations_i \cap Failure\ in\ Cell_i)$$

We can simplify the equation to:

$$\frac{P(Target\ in\ Cell_i\ \cap\ Observations_t\ \cap\ Failure\ in\ Cell_j)}{P(Observations_t\ \cap\ Failure\ in\ Cell_j)}$$

Continue simplifying the numerator:

$P(Target\ in\ Cell_i\ \cap\ Observations_t\ \cap\ Failure\ in\ Cell_j) =$

$P(Observations_t)P(Target\ in\ Cell_i \mid Observations_t)\times$

$P(Failure\ in\ Cell_j \mid Target\ in\ Cell_i\ \cap\ Observations_t)$

Failure in $Cell_j$ only depends on whether the target is in $Cell_j$ or not:

$$P(Observations_t)P(Target\ in\ Cell_i \mid Observations_t)P(Failure\ in\ Cell_j \mid Target\ in\ Cell_i)$$

$$P(Target\ in\ Cell_i \mid Observations_t) = Belief_t[i]$$

Finally, we have:

$$Belief_t[i] \times P(Failure\ in\ Cell_j \mid Target\ in\ Cell_i)$$

These two are the only terms that associate with $grid_i$, and we have the recursive formula.

For $P(Failure\ in\ Cell_j \mid Target\ in\ Cell_i)$, when $grid_i$ is the $grid_j$, we have the false negative rate table. When $grid_i$ and $grid_j$ are not the same, the $P(Failure\ in\ Cell_j \mid Target\ in\ Cell_i)$ will be 1.

For the denominator, it can be omitted, because it only function on normalizing the probability, and we can also simplify the denominator into:

$$P(Observations_t)P(Failure\ in\ Cell_j \mid Observations_t)$$

$P(Observations_t)$ cancels with the one in numerator, which leaves only $P(Failure\ in\ Cell_j \mid Observations_t)$.

$P(Failure\ in\ Cell_j \mid Observations_t) =$

$P(Failure\ in\ Cell_j\ \cap\ Target\ in\ Cell_j \mid Observations_t)\ +$

$P(Failure\ in\ Cell_j\ \cap\ Target\ not\ in\ Cell_j \mid Observations_t)$

Continue simplifying the denominator, we get:

$$Belief_t[j] \times (Probability\ of\ False\ Negative) + 1 - Belief_t[j]$$

For $P(Target\ Found\ in\ Cell_i \mid Observations_t)$, simplify this formula:

$P(Target\ Found\ in\ Cell_i \cap Target\ in\ Cell_i \mid Observations_t) +$

$P(Target\ Found\ in\ Cell_i \cap Target\ not\ in\ Cell_i \mid Observations_t)$

If target not in $Cell_i$, then $P(Target\ Found\ in\ Cell_i) == 0$

$P(Target\ Found\ in\ Cell_i \cap Target\ in\ Cell_i \mid Observations_t) =$

$P(Target\ Found\ in\ Cell_i \cap Target\ in\ Cell_i \cap Observations_t)P(Observations_t)$

$P(Observations_t) \times P(Target\ in\ Cell_i \mid Observations_t) \times$

$P(Target\ Found\ in\ Cell_i \mid Target\ in\ Cell_i)P(Observations_t) =$

$P(Target\ in\ Cell_i \mid Observations_t) \times P(Target\ Found\ in\ Cell_i \mid Target\ in\ Cell_i) =$

$Belief_t[i] \times (1 - [False\ Negative\ Rate\ of\ Cell_i])$

C. Strategy (Rule 2):

Strategy for the agent chooses grid: always picking the areas with highest probability and filter those options to leave only the one type terrain with the lowest false negative rate. Therefore, each $grid_i$ will times the probability $P(Target\ Found\ in\ Cell_i \mid Target\ is\ in\ Cell_i)$.

Explanation about the strategy.

Imagine two cases:

First one is that the target is in the flat areas, which are the terrains with the lowest false negative rate. If we search those flat area prior to other type terrains, we may have the positive result more quickly, because it is in flat areas and the probability for getting false negative information is relatively small. It is rational for agent to search flat areas first, if the target is in a flat grid.

Second case is that the target is in a cave. Agent will still search flat areas first, because the probability of flat areas will drop in a larger scale compared to other type of terrains. For example, the agent search a flat grid and find nothing in it. Because it is a flat grid, it is more likely that the target is not in the cell compared to agent report a false negative result. Therefore, we might not search the same grid in a relative short period, which means the probability of that certain cell drops and approaches to zero. We lift other grids probability with a large degree. However, if we search caves first, the significance of the probability drops will not be same as the flat areas. Then we cannot converge the probability to zero as quickly as we search flat areas.

D. Analysis (Rule 2):

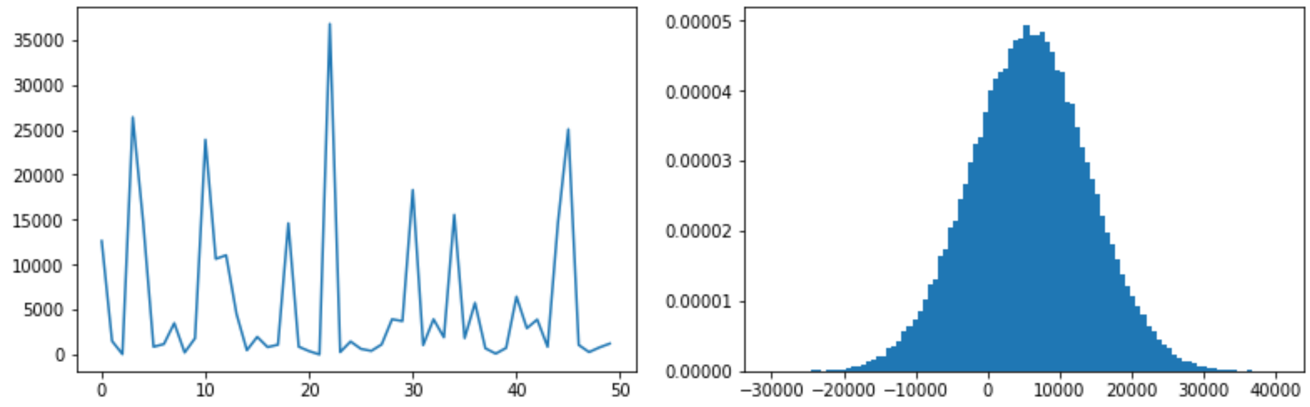

Figure 2. Number of Steps of Finding Target (Left) & Normal Distribution Right)

The left diagram is the number of steps to find target in each individual experiment. The average step is **5779**, and the standard derivation is **8250.5**. For stationary target, we can only lower one grid's probability and lift others by certain amount of degree; however, there is not other information provided by the map. Therefore, the probability will not drop or lift explosively in a short period of time.

The righthand side diagram is the normal distribution based on the expected value and its standard derivation. Ignore the negative side, the probability of steps between 0 ~ 20000 domains the distribution, which conform to what we believe. For the extreme cases, like found the target less than 1000 steps or larger than 20000 steps, they are due the random picking mechanism and the probability of false negativity.

E. Strategy (Rule 1):

At any time, search the cell with the highest probability of containing the target.

For rule 1, the only difference with rule 2 is that we do not need to worry about the terrain type, which means that agent only consider about the probability of grids containing the target. There is no strategy any more.

F. Analysis (Rule 1, discuss about Question 5):

Like rule 2, the number of steps are spread, and the average step is also large. The mean value of rule 1 is a little bit larger than rule 2, which is **8400**. The standard derivation is similar in scale, which is 8592.7. Because there is not a strategy in rule 1, the average search times should be more than rule 2 which uses a strategy to select those more likely of finding a target. However, this is just a strategy, and a strategy can sometimes lead the agent to a wrong place; therefore, the difference between two rules is not ought be big. The normal distribution also gives us a similar distribution to rule 2, and it also conforms to what I have said. Strategy will not provide a guaranteed useful direction; only new data or information will change the probability distribution in a dramatic way.

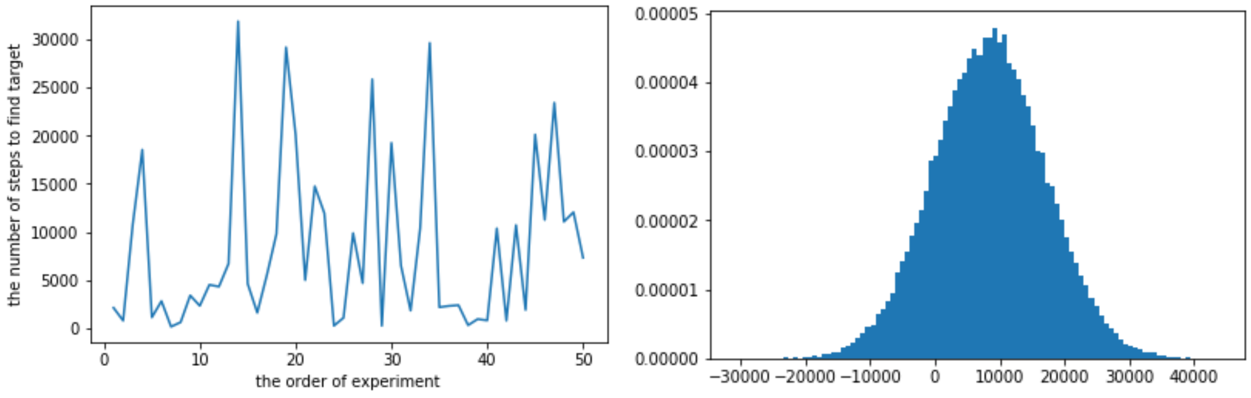

Figure 3. Number of Steps of Finding Target (Left) & Normal Distribution (Right)

G. Move Constraint Case(Question 4)

Under the constraint of calculating moving motions, we came up with a new rule based on rule but consider the distance between two grids. We named it as Rule 3: At any time, search the cell with the highest probability of finding the target meanwhile the nearest cell to the last cell.

And we run all the rules 1000 time to get a more general result.

	Rule 1	Rule 2	Rule 3
Average search times	6536.919	5484.980	5810.661
Average move times	217901.731	182788.689	33840.689

Table 1. Different rules' comparison

As the table 1. shows, our Rule 3 have a significant advantage in reducing the moving motions. Theoretically, the search times for rule 1 and rule 2 without applying the constraint in question should have the same number as the one using the constraints. Because, the randomly pick just ignores the counting of the agent's actions.

2. A Moving Target

Moving target has the same mechanism of building a map, and the functions (Rule 1 and Rule 2)of updating probability, belief of each grid, would have a different update way. The grids whose terrain are neither Type1 nor Type2 would have no probability of containing a target. Therefore, in this case, we need to relocate the probability after every motion. Those grids which satisfy the Type1 x Type2 would have the same probability of containing a target, and the sum of their probability would be 1.

A. Strategy (Rule 2):

Because map will inform us the previous location type and the new location terrain, the option will be all the gird which is the same terrain with the new location type and connected to a previous location type. We can shrink our options into a relatively small set. At first, the program just applies the same strategy whenever the agent need to generate a new possible option list. This method gives me an average with less than 1000 steps, which is a huge improvement compared to the stationary cases. However, I use the method of local optimization, which is that when agent did the t^{th} search, it always memorize the options it has in $(t-1)^{th}$ search. Due to the continuity of the target movement, the previous location in t^{th} observation should be the same type of terrain with the new location in $(t-1)^{th}$ search. Thus, we shrink the option set furthermore, and it gives me an even better number of steps.

B. Analysis (Rule 2):

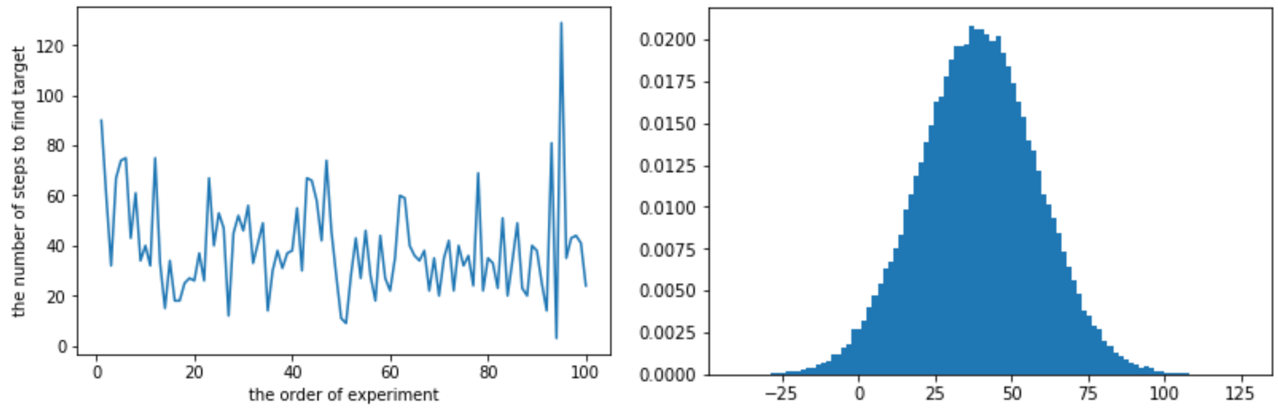

Figure 4. Number of Steps of Finding Target (Left) & Normal Distribution (Right)

The left diagram shows the number of steps agent takes for each experiment. Compared to the stationary case, moving target problem will normally have a way more less number of steps. The average steps in moving target problem is 39 (in 100 different experiments), and the standard derivation is 19.4, which also shows that the moving target problem will have a more stable steps.

On the righthand side, it is the normal distribution of the moving target search times. We can see that the interval of $(0, 100]$ almost covers the whole area of the distribution.

C. Strategy (Rule 1):

For moving target, because we already know which kind of terrain target shows up in observation_t, the meaning of finding will be different to the stationary case, in which we multiple with the probability

$$P(Target\ Found\ in\ Cell_i \mid Target\ in\ Cell_i)$$

However, in moving target, the next optional states we come up are the same type of terrain, which means they will have probability of finding target; thus, it will be meaningless if we just apply the same mechanism as we do in the stationary target problem. For rule 2, we do use some strategy which is the local optimization. In order to further explain the application of local optimization in rule 2, we can imagine the target travels in a series of locations, which is a list of node that each node will have at most edge connected to other nodes. The graph we come up is a digraph, we also have an ordered edge respect to the time change. Therefore, the path that the target has traveled is consist of a bunch of triples. The three nodes in a triple represent past, present, and future separately. When we collect the latest report, the option list we had from the last time gives the memory of past type terrain and present type terrain, the latest report tells us the future type terrain. By using the strategy, we can shrink the options list into a much smaller set, which can enhance the performance of the agent and speed up the search process. However, in rule 1, the agent will ignore the strategy of using triples, and it just simply looks for the location which is the future terrain type and connected to a past terrain type location.

D. Analysis (Rule 1):

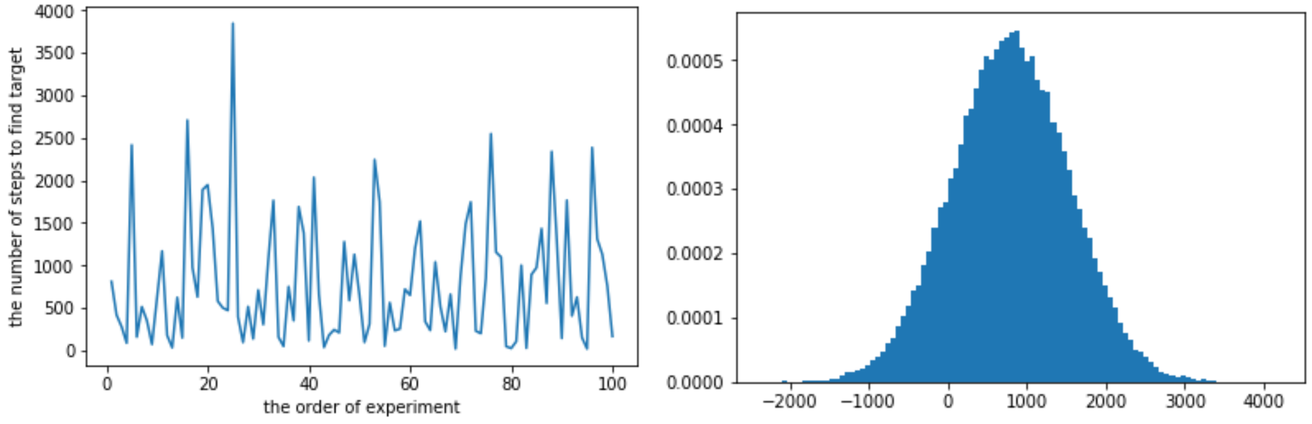

Figure 5. Number of Steps of Finding Target (Left) & Normal Distribution (Right)

For containing target probability, the average step is **795**, bigger than the one using rule 2. The standard derivation is 743.4, compared to rule 2 the variance also becomes larger, and the diagram on the lefthand side also shows the characteristic.

The normal distribution of the rule 1 also tells us that the interval of (0, 2000] domain the most area.

The reason of difference(for rule 1 and rule 2, and moving and stationary) is that the moving strategy provide more implicit information than the stationary strategy. The more information, the more accurate probability we will get. By using the triple idea, we can more quickly locate the target and find it. Therefore, in the world which is not determinist anywhere, the statistics(data) and the probability based on statistics will form a pretty accurate prediction.

Source Code

```
#!/Usr/Bin/Env Python3
# -*- Coding: Utf-8 -*-
"""
Created On Mon Mar 25 22:49:49 2019

@Author: Himmel
"""

Import Random
Import Numpy As Np
Import Matplotlib.Pyplot As Plt

Class Map:
    Def __Init__(Self, Size):
        Self.Size = Size;
        Self.Terrain = {}
        For X In Range(Self.Size):
            For Y In Range(Self.Size):
                Self.Terrain[(X, Y)] = 0
        Self.Target = (0,0)
        Self.Typereport = ("", "")

    Def Generatemap(Self):
        Locations = List(Self.Terrain.Keys())
        Random.Shuffle(Locations)
        For Coordinate In Locations:
            Probability = Random.Random()
            If Probability <= 0.2:
                Self.Terrain[Coordinate] = 1000 #Flat
            Elif Probability <= 0.5:
                Self.Terrain[Coordinate] = 800 #Hilly
            Elif Probability <= 0.8:
                Self.Terrain[Coordinate] = 500 #Forested
            Else:
                Self.Terrain[Coordinate] = 0   #Cave
        Random.Shuffle(Locations)
        Self.Target = Random.Choice(Locations)

    Def Printmap(Self):
        Graph = Np.Zeros((Self.Size, Self.Size), Dtype =
Int)
        For (X, Y) In Self.Terrain.Keys():
            Graph[X, Y] = Self.Terrain[(X, Y)]
        Plt.Figure(Figsize=(7.5,7.5))

Plt.Pcolor(Graph[::-1],Edgecolors='Black',Cmap='Gist
_Earth', Linewidths=2)
        Plt.Xticks([]), Plt.Yticks([])
        Plt.Tight_Layout()
        Plt.Show()

    Def Targetmove(Self):
        Choice = []
        (X, Y) = Self.Target
        If X - 1 >= 0:
            Choice.Append((X - 1, Y))
        If Y - 1 >= 0:
            Choice.Append((X, Y - 1))
        If X + 1 < Self.Size:
            Choice.Append((X + 1, Y))
        If Y + 1 < Self.Size:
            Choice.Append((X, Y + 1))
        Self.Target = Random.Choice(Choice)
        Self.Typereport = (Self.Terrain[(X, Y)],
Self.Terrain[Self.Target])

Class Searchrobot:
    Def __Init__(Self, Map):
        Self.Map = Map
        Self.Belief = {} #Containing Belief
        Self.Choices = []
        Self.Probability = {} #Finding Probability
        For (X, Y) In Self.Map.Terrain.Keys():
            Self.Belief[(X, Y)] = 1/(Self.Map.Size ** 2)
        Self.Observation = ()
        Self.Ptype = 0

    Def Randompick(Self):
        '''
        Choosing The Locations With The Highest
Probability Of Having Target.
        Then Take The Best Terrain(Order: Flat, Hilly,
Forest, Cave).
        Randomly Picking One From The These Choices
        '''
        Highestprobability = Max(Self.Belief.Values())
        Choices = {}
        For Coordinate In Self.Belief.Keys():
            If Self.Belief[Coordinate] ==
Highestprobability:
                Choices[Coordinate] =
Self.Map.Terrain[Coordinate]
        Betterchoices = []
        Terrain = Max(Choices.Values())
        For Coordinate In Choices.Keys():
            If Choices[Coordinate] == Terrain:
                Betterchoices.Append(Coordinate)
        Return Random.Choice(List(Choices.Keys()))

    Def Specificpick(Self):
        '''
        Choose A Certain Kind Of Area, Based On The
Type Report.
        Pick The Location With Highest Probability
        '''
        If Self.Map.Typereport == ("",""):
            Return Self.Randompick()
        (Origination, Destination) = Self.Map.Typereport
        Choices = {}
        Self.Choices = []
        For Coordinate In Self.Belief.Keys():
            If Self.Map.Terrain[Coordinate] ==
Destination:
                (X, Y) = Coordinate
                If Self.Choices == []:
                    If X - 1 >= 0 And Self.Map.Terrain[(X -
1, Y)] == Origination:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                    Elif Y - 1 >= 0 And Self.Map.Terrain[(X,
Y - 1)] == Origination:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                    Elif X + 1 < Self.Map.Size And
Self.Map.Terrain[(X + 1, Y)] == Origination:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                    Elif Y + 1 < Self.Map.Size And
Self.Map.Terrain[(X, Y + 1)] == Origination:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                Else:
                    If (X - 1, Y) In Self.Choices:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                    Elif (X, Y - 1) In Self.Choices:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                    Elif (X + 1, Y) In Self.Choices:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
                    Elif (X, Y + 1) In Self.Choices:
                        Choices[Coordinate] =
Self.Belief[Coordinate]
        Self.Choices = List(Choices.Keys())
        Return Random.Choice(Self.Choices)


    Def Search(Self, Location):
        '''
        Pick A Location With Highest Probability And
Search It.
        Give The False Negative Table To Generate
Certain Feeback
        '''
        (X, Y) = Location
        Self.Observation = (X, Y)
        Probability = Random.Random()
        If Self.Map.Terrain[(X, Y)] == 1000:
            Self.Ptype = 0.1
            If Probability <= 0.1 Or (X, Y) !=
Self.Map.Target:
                Return "Failure"
            Else:
                Return "Success"
        Elif Self.Map.Terrain[(X, Y)] == 800:
            Self.Ptype = 0.3
            If Probability <= 0.3 Or (X, Y) !=
Self.Map.Target:
                Return "Failure"
            Else:
                Return "Success"
        Elif Self.Map.Terrain[(X, Y)] == 500:
            Self.Ptype = 0.7
            If Probability <= 0.7 Or (X, Y) !=
Self.Map.Target:
                Return "Failure"
            Else:
                Return "Success"
        Else:
            Self.Ptype = 0.9
            If Probability <= 0.9 Or (X, Y) !=
Self.Map.Target:
                Return "Failure"
            Else:
                Return "Success"

    Def Stationarysearch(Self):
        '''
        If Agent Currently Does Not Give Us The
Positive(Success) Feeback, Keep Searching
        Updating The Belief Of Each Location During
The Searching Process
        '''
        Number = 0
        While Self.Search(Self.Randompick()) !=
"Success":
            Denominator = 1 -
Self.Belief[Self.Observation] +
Self.Belief[Self.Observation] * Self.Ptype
            For Coordinate In Self.Belief.Keys():
                If Coordinate == Self.Observation:
                    If Self.Map.Terrain[Coordinate] == 1000:
                        Self.Belief[Coordinate] *= 0.1
                    Elif Self.Map.Terrain[Coordinate] ==
800:
                        Self.Belief[Coordinate] *= 0.3
                    Elif Self.Map.Terrain[Coordinate] ==
500:
                        Self.Belief[Coordinate] *= 0.7
                    Else:
                        Self.Belief[Coordinate] *= 0.9
                Self.Belief[Coordinate] =
Self.Belief[Coordinate]/Denominator
            Number += 1
        Print(Self.Observation)
        If Self.Map.Terrain[Self.Map.Target] == 1000:
            Print("Target Is At Flat Terrain")
        Elif Self.Map.Terrain[Self.Map.Target] == 800:
            Print("Target Is At Hilly Terrain")
        Elif Self.Map.Terrain[Self.Map.Target] == 500:
            Print("Target Is At Forested Terrain")
        Else:
            Print("Target Is In A Cave")
        Print("Congratulations! You've Found The
Target!")
        Return Number
    Def Movingsearch(Self):
        '''
        Each Search-Failure Will Make The Target
Moving Into A Neighbor Location
        The New Probabilities Will Be Updated The
Same As Stationary Search
        '''
        Number = 0
        While Self.Search(Self.Specificpick()) != "Success":
            Denominator = 1 -
Self.Belief[Self.Observation] +
Self.Belief[Self.Observation] * Self.Ptype
            For Coordinate In Self.Belief.Keys():
                If Coordinate == Self.Observation:
                    If Self.Map.Terrain[Coordinate] == 1000:
                        Self.Belief[Coordinate] *= 0.1
                    Elif Self.Map.Terrain[Coordinate] ==
800:
                        Self.Belief[Coordinate] *= 0.3
                    Elif Self.Map.Terrain[Coordinate] ==
500:
                        Self.Belief[Coordinate] *= 0.7
                    Else:
                        Self.Belief[Coordinate] *= 0.9
                Self.Belief[Coordinate] =
Self.Belief[Coordinate]/Denominator
            Self.Map.Targetmove()
            Number += 1
        Print(Self.Observation)
        If Self.Map.Terrain[Self.Map.Target] == 1000:
            Print("Target Is At Flat Terrain")
        Elif Self.Map.Terrain[Self.Map.Target] == 800:
            Print("Target Is At Hilly Terrain")
        Elif Self.Map.Terrain[Self.Map.Target] == 500:
            Print("Target Is At Forested Terrain")
        Else:
            Print("Target Is In A Cave")
        Print("Congratulations! You've Found The
Target!")
        Return Number

    Def Regression(Self):
        For Coordinate In Self.Belief.Keys():
            Self.Belief[Coordinate] = 1/(Self.Map.Size ** 2)


M = Map(50)
Stationary = []
Moving = []
'''
For I In Range(50):
    M.Generatemap()
    Agent = Searchrobot(M)
    Stationary.Append(Agent.Stationarysearch())
    #Agent.Regression()
    #Moving.Append(Agent.Movingsearch())
Mean= Sum(Stationary) // 50
Std = 0
For Number In Stationary:
    Std += (Number - Mean) ** 2
Std = (Std / 50)** 0.5

Normaldistribution = Np.Random.Normal(Mean, Std,
100000)
Plt.Hist(Normaldistribution, Bins=100, Normed=True)
Plt.Show()
Plt.Xlabel("The Order Of Experiment")
Plt.Ylabel("The Number Of Steps To Find Target")
Plt.Plot(Range(1,51), Stationary)
'''
M.Generatemap()
For I In Range(100):
    Agent = Searchrobot(M)
#Stationary.Append(Agent.Stationarysearch())
    Moving.Append(Agent.Movingsearch())
Mean = Sum(Moving) // 100
Std = 0
For Number In Moving:
    Std += (Number - Mean) ** 2
Std = (Std / 100)** 0.5
Normaldistribution = Np.Random.Normal(Mean, Std,
100000)
Plt.Hist(Normaldistribution, Bins=100, Normed=True)
Plt.Show()

Plt.Xlabel("The Order Of Experiment")
Plt.Ylabel("The Number Of Steps To Find Target")
Plt.Plot(Range(1,101), Moving)
```