

Assignment 2.5 - Decision Making

By Tatsat Vyas, Sri Jay Adarsh Gogineni, Andy Guo

Introduction

This project builds off of the previous minesweeper algorithm that was created from assignment 2. In our previous attempt to solve the board, our algorithm started to randomly select after no safe cells could be identified. However, in this assignment we are required to intelligently select the next best move possible. To do this we can define the next best move to be: minimizing cost or minimizing risk. To minimize cost this means the best choice of square is the one that minimizes the expected number of mines stepped on. As minimizing risk, means the best choice of square is the one that minimizes the expected number of squares you have to step on.

Assessing the Future

In order to solve this problem we must first determine the probability that a square would be a mine. This would give us a way to choose the most likely square each time around. If we calculate the probabilities correctly, this agent will have the most information possible to make its decision, and will make the best decision every time with the available knowledge base.

To do this, we need to know a few things as our knowledge base. First, we need to know all the combinations of mines that are in the number of tiles we have undiscovered. Next, for each tile we need to know the number of combinations in which a mine occurred there. So, in order to find the probability of each unknown tile being a mine, we divide the number of combinations the current tile/total number of combinations. After we find the probabilities, we have all the information we need to make our next decision.

To obtain these two information our algorithm will first take our current knowledge of the board and separate each of the squares based on the adjacent cells that we have already visited. This will help us separate our board into “groups”.

A C1	AB C2	AB C3	B C4
A C5	1	3	B C6
A C7	AB C8	AB C9	B C10

For example, if we have a board and we discover two cells. One cell contains a with a 1, then we will mark all of its surrounding cells A. And another cell we discovered was 3, we will mark B to all its surrounding cells. We can continue this for the next cell we discover and mark C to all of its neighbors and so on. This information will help us to determine the equation. As you can see cells 1, 5, and 7 (c1, c5, and c7) will be in one group. C2, c3, c8, c9 is another group. And the last group is c4, c6, c10. We can then set them to two different equations

$$\text{Equation 1: } (C1+C5+C7) + (C2+C3+C8+C9) = 1$$

$$\text{Equation 2: } (C2+C3+C8+C9) + (C4+C6+C10) = 3$$

Now that we have the equations we can start analyzing these groups. As we can see from equation 1, there are two possible solutions to this problem. (C1+C5+C7) can either have 0 mines or 1 mine. This will affect the solutions of the other two equations.

$$\text{Solution 1: } (C1+C5+C7) = 1$$

$$(C2+C3+C8+C9) = 0$$

$$(C4+C6+10) = 3$$

$$\text{Solution 2: } (C1+C5+C7) = 0$$

$$(C2+C3+C8+C9) = 1$$

$$(C4+C6+10) = 2$$

The more groups we have the more solutions we will have to branch off of but in this case there are only two solutions possible for this board. Now we can use this information to determine the total number of combinations which mines can occur. To do this we will be using binomial coefficient to determine this (nCk). Where n is the number of cells in that group and k is the number of mines we determined in that solution:

Formula

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Solution 1:

$$(C1+C5+C7): \quad 3 \quad nCk \quad 1 = 3$$

$$(C2+C3+C8+C9): \quad 4 \quad nCk \quad 0 = 1$$

$$(C4+C6+10): \quad 3 \quad nCk \quad 3 = 1$$

$$3 * 1 * 1 = 3 \text{ total combinations}$$

Solution 2:

$$(C1+C5+C7): \quad 3 \quad nCk \quad 0 = 1$$

$$(C2+C3+C8+C9): \quad 4 \quad nCk \quad 1 = 4$$

$$(C4+C6+10): 3 \text{ nCk } 2 = 3$$

$$1 * 4 * 3 = 12 \text{ total combinations}$$

By adding the two combinations together ($3 + 12 = 15$) we get there are a total of 15 different placements that the mine can occur.

Now we are able to determine the probability that each cell will contain a mine. To do this we will take the number of mines we have determined and divide it by the number of cells in that group times by the combination found in that solution. For example in our (C1+C5+C7) case:

In solution 1: 0 mines in 3 fields, with 12 combinations

In solution 2: 1 mines in 3 fields, with 3 combinations

This means we do the following equations:

$(0/3) * 12 + (1/3) * 3 = 1$ and we use this information to divide it by the total number of combinations to get $1/15$ or 0.06667 . By continuing this we get the following probability:

0.067	0.2	0.2	0.73
0.067	1	3	0.73
0.067	0.2	0.2	0.73

As you can see the probability of choosing the left side is far safer than choosing the right side of the board. After revealing a new tile our algorithm will start the process again to determine the new probabilities.

Minimizing Cost

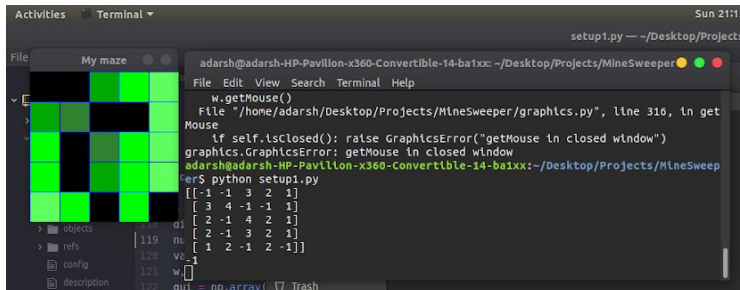
When we are talking about minimizing cost when it comes to improving our agent from project 2, we do it in the following way.

As explained above, our goal is to first find the bordering tiles that are still unknown. From our current board, our first order of business is to find out which of the cells in our board are our unknown neighbors that our current clues are giving hints about. We can do this by going through our knowledge base so far and getting a list of all the bordering cells that our clues are referring to. Once we have these, we are in the beginning stages of being able to understand how each cell must be chosen.

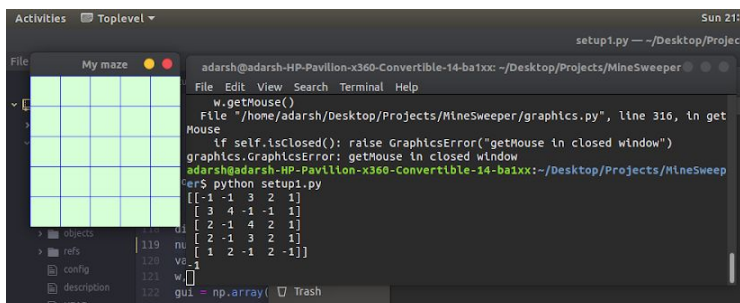
Once we have our bordering list of unknown cells, what we need to do now is figure out all the possible combinations of mines that could possibly go into these neighboring cells. We can do this by grouping these cells into different groups based off of their "parent". If the cell has only 1 parent, it will be in a group, if it has more than 1 parent, it will go into its own group. What we mean by the parent of the cell is in this case, the cell that the hint came from is the parent. So in the example above, group A and AB have the same parent because they both share the cell that contains 1. In doing this, we have created groupings of our outer-unknown cells.

We now know that if a mine appears in one of these groupings, we can subtract this number from the number within the hint. This is because if a mine appears within the group, the hint will be referring to that mine, thus it will be subtracted. In this way, we are able to "make" our system of equations as shown above. With these "equations", once we set a mine into the group, we can run our old algorithm from project 2 again, and see what configurations we can make from assuming the mine is in that cell. With this, we can make the algorithm for Minimizing Cost.

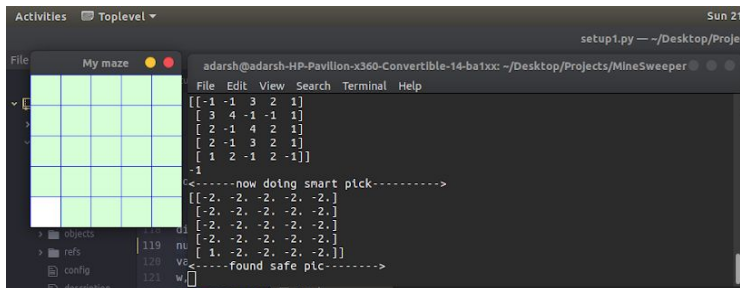
Once all the probabilities are found out by our algorithm, by assigning the groupings all different probabilities, we can pick the lowest one and minimize our cost. A run through of our algorithm is described below:



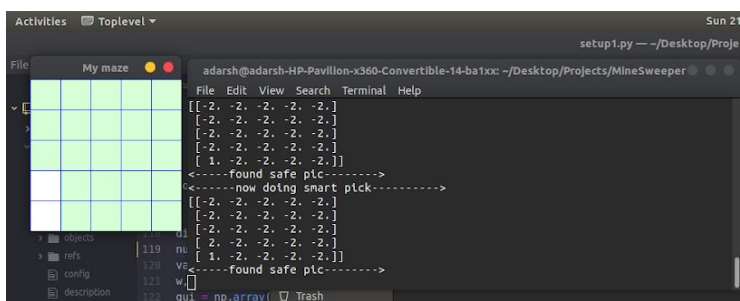
```
File Edit View Search Terminal Help
File "/home/adarsh/Desktop/Projects/MineSweeper/graphics.py", line 316, in get
Mouse
    if self.isclosed(): raise GraphicsError("getMouse in closed window")
graphics.GraphicsError: getMouse in closed window
adarsh@adarsh-HP-Pavillon-x360-Convertible-14-ba1xx:~/Desktop/Projects/MineSweep
er$ python setup1.py
[[ -1 -1 3 2 1]
 [ 3 4 -1 -1 1]
 [ 2 -1 4 2 1]
 [ 2 -1 3 2 1]
 [ 1 2 -1 2 -1]]
119 nu
120 vo
121 w
122 gui = np.array([ 7 Trash
```



```
File Edit View Search Terminal Help
File "/home/adarsh/Desktop/Projects/MineSweeper/graphics.py", line 316, in get
Mouse
    if self.isclosed(): raise GraphicsError("getMouse in closed window")
graphics.GraphicsError: getMouse in closed window
adarsh@adarsh-HP-Pavillon-x360-Convertible-14-ba1xx:~/Desktop/Projects/MineSweep
er$ python setup1.py
[[ -1 -1 3 2 1]
 [ 3 4 -1 -1 1]
 [ 2 -1 4 2 1]
 [ 2 -1 3 2 1]
 [ 1 2 -1 2 -1]]
119 nu
120 vo
121 w
122 gui = np.array([ 7 Trash
```



```
File Edit View Search Terminal Help
File "/home/adarsh/Desktop/Projects/MineSweeper/graphics.py", line 316, in get
Mouse
    if self.isclosed(): raise GraphicsError("getMouse in closed window")
graphics.GraphicsError: getMouse in closed window
adarsh@adarsh-HP-Pavillon-x360-Convertible-14-ba1xx:~/Desktop/Projects/MineSweep
er$ python setup1.py
[[ -1 -1 3 2 1]
 [ 3 4 -1 -1 1]
 [ 2 -1 4 2 1]
 [ 2 -1 3 2 1]
 [ 1 2 -1 2 -1]]
119 nu
120 vo
121 w
122 gui = np.array([ 7 Trash
```



```
File Edit View Search Terminal Help
File "/home/adarsh/Desktop/Projects/MineSweeper/graphics.py", line 316, in get
Mouse
    if self.isclosed(): raise GraphicsError("getMouse in closed window")
graphics.GraphicsError: getMouse in closed window
adarsh@adarsh-HP-Pavillon-x360-Convertible-14-ba1xx:~/Desktop/Projects/MineSweep
er$ python setup1.py
[[ -1 -1 3 2 1]
 [ 3 4 -1 -1 1]
 [ 2 -1 4 2 1]
 [ 2 -1 3 2 1]
 [ 1 2 -1 2 -1]]
119 nu
120 vo
121 w
122 gui = np.array([ 7 Trash
```

Activities Toplevel Sun 21:14

setup1.py -- ~/Desktop/Project

File Edit View Search Terminal Help

My maze

objects refs config description

119 nu
120 V6
121 w
122 gui = np.array()

```
[ -2. -2. -2. -2. -2.]  
[ 2. -2. -2. -2. -2.]  
[ 1. -2. -2. -2. -2.]  
-----found safe pics----->  
-----found mines----->  
[(3, 1)]  
-----now doing smart pick----->  
[[-2. -2. -2. -2. -2.]  
[-2. -2. -2. -2. -2.]  
[-2. -2. -2. -2. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 1. 2. -2. -2. -2.]]
```

Activities Toplevel Sun 21:14

setup1.py -- ~/Desktop/Project

File Edit View Search Terminal Help

My maze

objects refs config description

119 nu
120 V6
121 w
122 gui = np.array()

```
[ -2. -2. -2. -2. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 1. 2. -2. -2. -2.]  
-----found safe pics----->  
-----found mines----->  
[(2, 1)]  
-----now doing smart pick----->  
[[-2. -2. -2. -2. -2.]  
[-2. -2. -2. -2. -2.]  
[-2. -2. -2. -2. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 1. 2. -2. -2. -2.]]
```

Activities Toplevel Sun 21:14

setup1.py -- ~/Desktop/Project

File Edit View Search Terminal Help

My maze

objects refs config description

119 nu
120 V6
121 w
122 gui = np.array()

```
[ 2. -1. -2. -2. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 1. 2. -2. -2. -2.]  
-----found safe pics----->  
-----Stepped on a mine----->  
-----now doing smart pick----->  
[[-2. -2. -2. -2. -2.]  
[-2. -2. -2. -2. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 1. 2. -1. -2. -2.]]  
error  
-----no safe pics----->
```

Activities Toplevel Sun 21:14

setup1.py -- ~/Desktop/Project

File Edit View Search Terminal Help

My maze

objects refs config description

119 nu
120 V6
121 w
122 gui = np.array()

```
[ 2. -1. -2. -2. -2.]  
[ 1. 2. -1. -2. -2.]  
error  
-----no safe pics----->  
-----Stepped on a mine----->  
-----now doing smart pick----->  
[[-2. -2. -2. -2. -2.]  
[-2. -2. -2. -2. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 1. 2. -1. -2. -1.]]  
error  
-----no safe pics----->
```

Activities Toplevel Sun 21:14

setup1.py -- ~/Desktop/Project

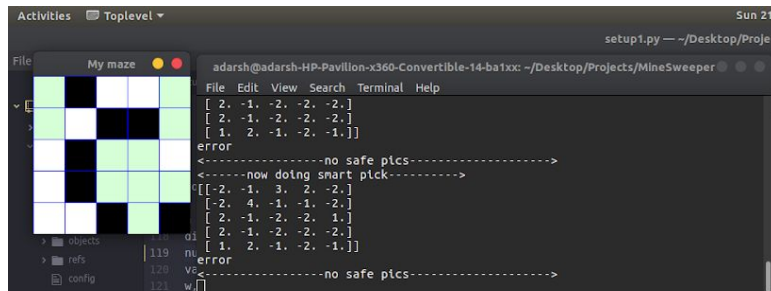
File Edit View Search Terminal Help

My maze

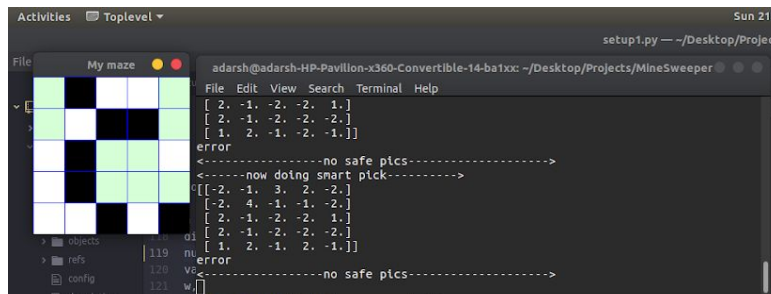
objects refs config description

119 nu
120 V6
121 w
122 gui = np.array()

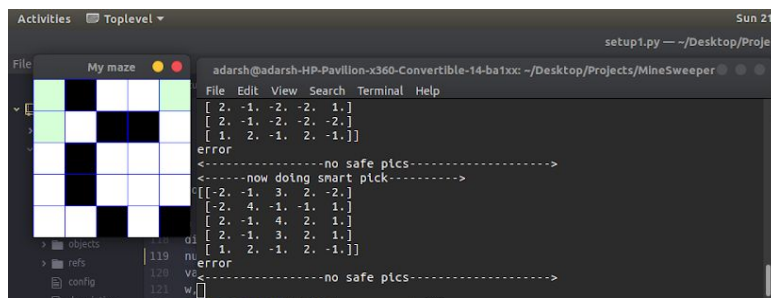
```
error  
-----no safe pics----->  
-----Stepped on a mine----->  
-----Stepped on a mine----->  
-----Stepped on a mine----->  
-----now doing smart pick----->  
[[-2. -1. 3. 2. -2.]  
[ 2. 4. -1. -1. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 2. -1. -2. -2. -2.]  
[ 1. 2. -1. -2. -1.]]  
error  
-----no safe pics----->
```



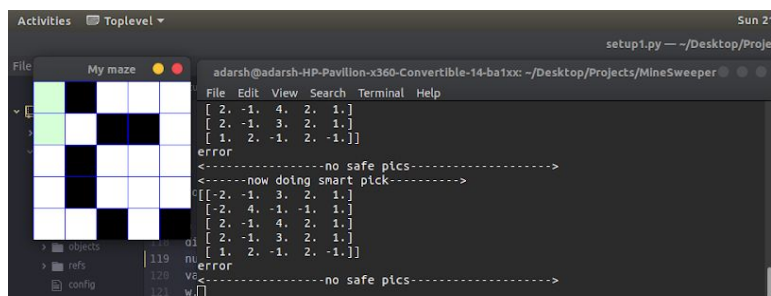
```
File Edit View Search Terminal Help
[ 2, -1, -2, -2, -2.]
[ 2, -1, -2, -2, -2.]
[ 1, 2, -1, -2, -1.]]
error
<-----no safe pics----->
<-----now doing smart pick----->
[[[-2, -1, 3, 2, -2.]
[-2, 4, -1, -1, -2.]
[ 2, -1, -2, -2, 1.]
[ 2, -1, -2, -2, -2.]
[ 1, 2, -1, -2, -1.]]]
error
<-----no safe pics----->
```



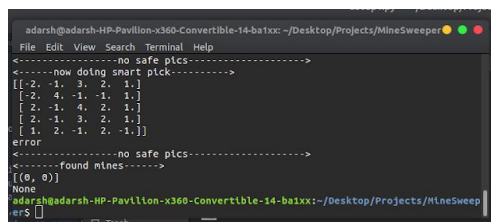
```
File Edit View Search Terminal Help
[ 2, -1, -2, -2, 1.]
[ 2, -1, -2, -2, -2.]
[ 1, 2, -1, -2, -1.]]
error
<-----no safe pics----->
<-----now doing smart pick----->
[[[-2, -1, 3, 2, -2.]
[-2, 4, -1, -1, -2.]
[ 2, -1, -2, -2, 1.]
[ 2, -1, -2, -2, -2.]
[ 1, 2, -1, 2, -1.]]]
error
<-----no safe pics----->
```



```
File Edit View Search Terminal Help
[ 2, -1, -2, -2, 1.]
[ 2, -1, -2, -2, -2.]
[ 1, 2, -1, 2, -1.]]
error
<-----no safe pics----->
<-----now doing smart pick----->
[[[-2, -1, 3, 2, -2.]
[-2, 4, -1, -1, 1.]
[ 2, -1, 4, 2, 1.]
[ 2, -1, 3, 2, 1.]
[ 1, 2, -1, 2, -1.]]]
error
<-----no safe pics----->
```

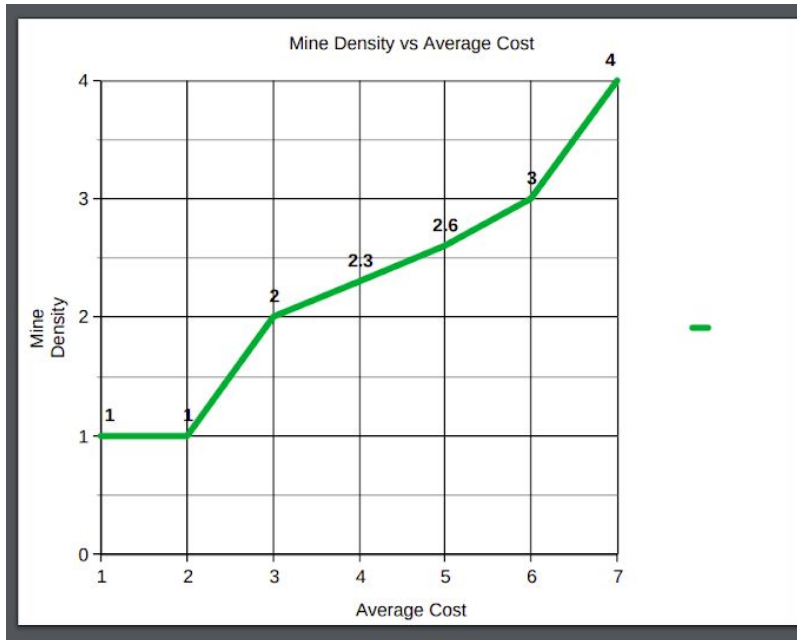


```
File Edit View Search Terminal Help
[ 2, -1, 4, 2, 1.]
[ 2, -1, 3, 2, 1.]
[ 1, 2, -1, 2, -1.]]
error
<-----no safe pics----->
<-----now doing smart pick----->
[[[-2, -1, 3, 2, 1.]
[-2, 4, -1, -1, 1.]
[ 2, -1, 4, 2, 1.]
[ 2, -1, 3, 2, 1.]
[ 1, 2, -1, 2, -1.]]]
error
<-----no safe pics----->
```

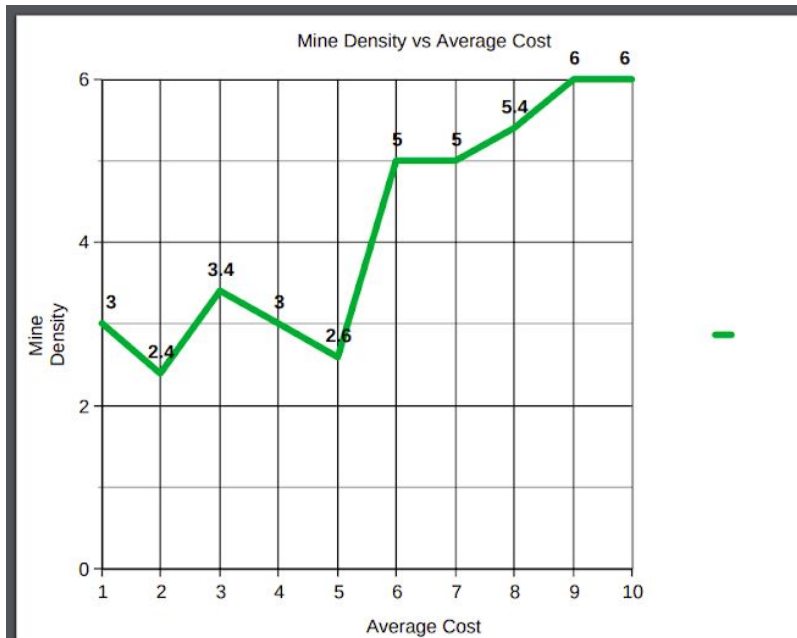


```
File Edit View Search Terminal Help
<-----no safe pics----->
<-----now doing smart pick----->
[[[-2, -1, 3, 2, 1.]
[-2, 4, -1, -1, 1.]
[ 2, -1, 4, 2, 1.]
[ 2, -1, 3, 2, 1.]
[ 1, 2, -1, 2, -1.]]]
error
<-----no safe pics----->
<-----found mines----->
[(0, 0)]
None
adash@adash-HP-Pavillon-x360-Convertible-14-ba1xx:~/Desktop/Projects/MineSweep
ers$
```

Plots:



Minimized Cost



Original Algorithm

Comparisons

When comparing the improved agent with the original agent that randomly chose after it ran out of safe moves, we can see that the improved agent actually has a lower average cost at each mine density. This does not come of complete surprise, as our improved agent is supposed to be more intelligent than our original. This is obvious due to the inclusion of probabilities. Once we have an empirical way to estimate how likely a mine is in a certain cell, picking the least of those probabilities will almost always give the best outcome. Due to

this, no random choices are being made unnecessarily, and are only being made when the probabilities of the mines being in a certain spot are equal.