Homework 2 Mine Sweeper

1 Work Distribution

Yuhan Cao(yc992): Responsilble for the environment creation and basic algorithm.

Weikai Mao(wm282): Responsible for the improved algorithm.

2 Funtions and Parameters explanation

We have 3 python files totally.

- Environment.py It creates an mine sweeper environment like border.

  1. `init` initialize the environment

  `self.winFlag`: If winFlag is true, then the program will show the window.

  `self.mine`: all the mines' position in the box.

  2. `autoReact`

     if we do not reveal window(winFlag = False), then we use autoReact function to **autoplay** sweeper.

  3. `baseShow`

     Show function when we use basic algorithm.

  4. `bomb`

     Draw bombed mines on the window.

  5. `drawBox`

     Draw grids on the window.

  6. `getScore`

     return the final score of current AI mine swepper.

  7. `loadImg`

     load GUI image into the programe.

  8. `numMine`

     return the number of mines around current cell (0-8).

  9. `put_mine`

     After loading grids, we randomly put n mines among them.

```
self.box[x][y] = 1
self.mine.add((x,y))
```

10. `react`

    When we create GUI and reveal the window, we could click the mouse the check each step of mine sweeper or play it manually.

11. `recover`

    When there is already a flag on the cuurent cell, then we recover it to uncovered status.

12. `recursion`

    recursively to cover all cells(numMine>=0) that around the current cell.

13. `setFlag`

    Draw flag on the window.

14. `setMine`

    draw a mine on the window.

15. `setNull`

    Draw the clue of cells whose number of around mines is 0.

16. `setNum`

    Draw the clue of cells.

17. `show`

    when we use algrothim we improved in agent.py.

    isRobot = True: autoplay mine sweeper with AI algorithm.

    needClick = True: we need to click the mouse to get next step.

- Agent.py realize improved algorithm

  1. `init`

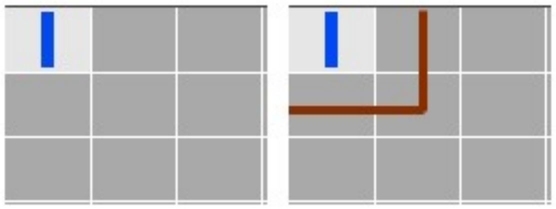
     `def __init__(self, n:int, boxView:list, knowN:bool, splitBorders:bool, uncertainModel:str):`

     To initialize the object. It have 5 parameters: `n:int`, `boxView:list`, `knowN:bool`, `splitBorders:bool`, `uncertainModel:str`, where `n` is the number of mines in the map; `boxView` is the information given by `environment.py`; `knowN` means whether the agent know the number of mines or not; `splitBorders` means whether the agent will split the borders or not; `uncertainModel` specifies the uncertainty model.

  2. `getBorders`

     `def getBorders(self) -> list:`

This function used to return borders by the information of `boxView`. The border is a list of connective border nodes in surrounding of clue (num) nodes. For example, in the plot below, the clue node is `(0,0)` and the border is `[(0,1), (1,1), (1,0)]`.



3. `check`

   `def check(self, i:int, j:int, borderSet:set, borderDict:dict) -> bool:`

   The border node `(i,j)` has been set a value in `borderDict`. This function check whether this value can satisfy the clue (num) nodes in the surrounding of node `(i,j)`. Take the plot above for example, `{(0,1):1, (1,1):0, (1,0):0}` can satisfy the clue node `(0,0)`, but `{(0,1):1, (1,1):1}` cannot because 1+1>1.

4. `dfs`

   `def dfs(self, k:int, border:list, borderSet:set, borderDict:dict) -> None:`

   We use DFS to do exhaustivity. For example, in the plot above, the border have 3 border nodes, and each of them can be 0 or 1, so there are 8 cases for the values of the values of these border nodes. We traverse all posssible cases by DFS, and use the `check` function to check whether this case is valid. If not valid, we backtrack to previous case. By using backtracking, we can reduce time complexity dramativcally.

   For example, in the plot above, `{(0,1):1, (1,1):1}` is not valid, so we do not need to check `{(0,1):1, (1,1):1, (1,1):0}` and `{(0,1):1, (1,1):1, (1,1):1}`.

5. `mergeDict`

   `def mergeDict(self, dictList: list) -> dict:`

   This function is used to merge the values in different dictionary to a mean value, which is the probability of this `pos` to be 1 (mine). Take the same example as before, `dictList=[{(0,1):1, (1,1):0, (1,1):0}, {(0,1):0, (1,1):1, (1,1):0}, {(0,1):0, (1,1):0, (1,1):1}]`, and this function will return `{(0,1):0.3333, (1,1):0.3333, (1,1):0.3333}`, which means the probability of this `pos` to be 1 (mine) is 0.3333.

   If the probability of the `pos` to be 1 (mine) is 1, we will know this `pos` must be 1, so we should set this `pos` as flag. Similarly, if the probability of the `pos` to be 1 (mine) is 0, we will know this `pos` must be 0 (no mine), so we should uncover this node.

6. `tank`

   `def tank(self) -> tuple:`

This function calls all the functions we defined above. If some nodes are certain to be 1, we set flags on them. If some nodes are certain to be 0, we uncover them in next step. If there is no node is certain to be 1 or 0, we uncover the node with lowest probability, which means this node is most unlikely to be mine.

- Baseline.py realize basic agent algorithm
    1. init
    2. base

2.1 Basic Agent Algorithm

Basic algorithm is realized in baseline.py

By setting parameters in environment.py main function, we can choose to create window or not.

How we do better?

Without picking a cell to reveal randomly, we could check the border of current cell and cumpute all cells' probability near the border. If probalility is 0,we uncover it, if it is 1,we set it as flag. If there is no node is certain to be 1 or 0, we uncover the node with lowest probability

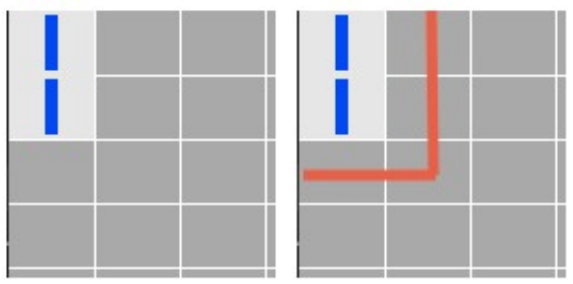
3 Questions and Writeup

- Representation:

  We use two dimension array to represent box. In our program, self.box[][][x][y] =1 represents the there is am mine in (x+1) row, (y+1) column. self.boxView[x][y] = 9 means there is mine or flag in current cell, self.boxView[x][y] = 10, it means that current cell is uncovered. self.boxView[x][y]= 0....8, reprsents the clue of current cell. We use self.box to represent board and use self.boxView to save updated information.

- Inference:

  We perform DFS (backtracking) to traverse all possible cases of border nodes.



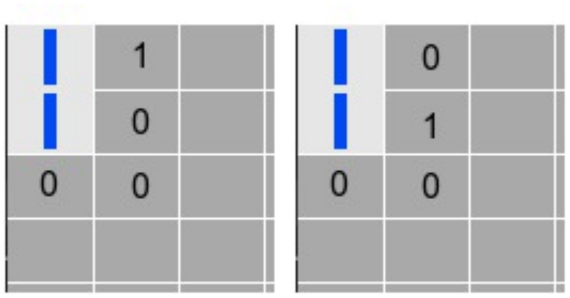
Take the plot above for example. The red line is the border. There are 4 nodes on border: `(0,1), (1,1), (2,1), (2,0)`, and each of can be 0 or 1, so there are 2^4 possible cases if there are no constraints. However, the clue nodes are the constraints of the values of these border nodes.

Denote $val(i,j)$ as the value of node `(i,j)`. $val(i,j) = 0 \; or \; 1$.

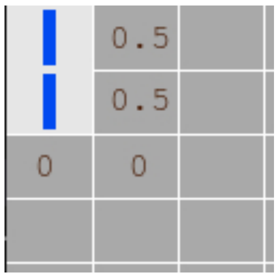
$$val(0,1) + val(1,1) = 1$$
$$val(0,1) + val(1,1) + val(2,1) + val(2,0) = 1$$

We perform DFS (backtracking) to search valid cases with constraints. There are 2 possible cases that satisfy the two constraints above:

$$\{val(0,1) = 1, val(1,1) = 0, val(2,1) = 0, val(2,0) = 0\},$$
$$\{val(0,1) = 0, val(1,1) = 1, val(2,1) = 0, val(2,0) = 0\}.$$



The average of these two cases indicates the probability of a node to be 1 (mine):



This algorithm deduce everything it can from a given clue, because it traverse all possible cases and do inference on these cases.

- Decisions:

  For those border nodes with probability 0, we know this node must be 0 (no mine), so we should uncover this node. For those border nodes with probability 1, we know this node must be 1 (mine), so we should set this node as flag.

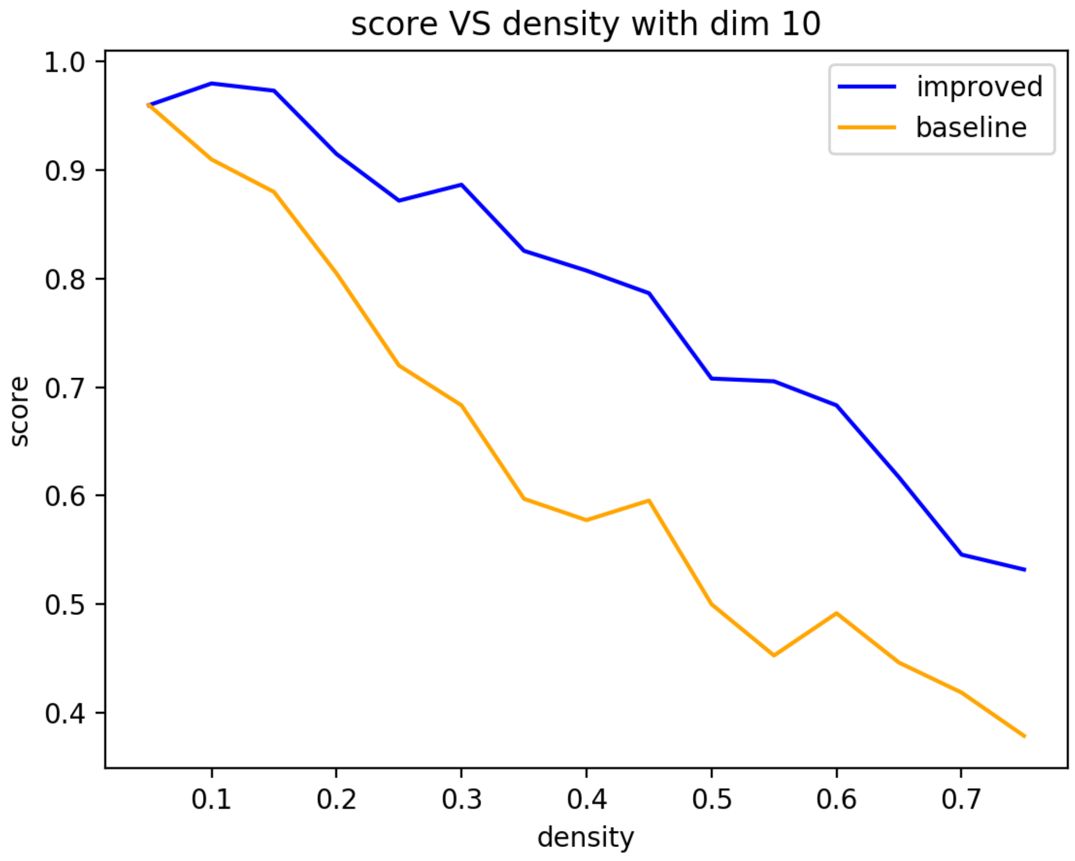
  If there is no border nodes with probability 0 or 1, we will make a guess. We uncover the node with lowest probability, which means this node is most unlikely to be mine.

- Performance:

  I agree with the decisions made by our program. These decisions make sense.

- Performance:

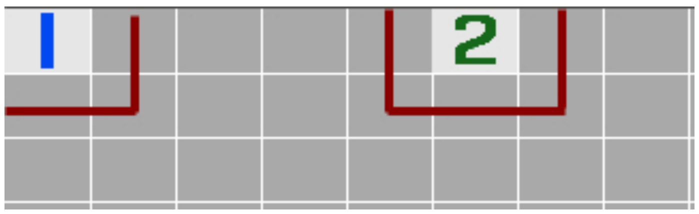
  We test these two algorithm with density between 0.05-0.8. The board dimension is 10, and each algorithm we test 10 times to get average score.



- Efficiency:

  A map may have several borders. If the total number of nodes in these borders is m. The space complexity is $O(8m) + O(dim^2)$, and it is acceptable. The time complexity of our algorithm is $O(m) = 2^m$. The computation is expensive although we use backtracking to reduce it to some extent.

  We noticed that different borders will not influence each other since they do not have a common node. We can reduce the time complexity dramatically if we perform DFS on the borders separately. If we separate m nodes into two borders, and each border contains $m/2$ nodes. The time complexity will be $O(m) = 2^{m/2} + 2^{m/2} = 2^{\frac{m}{2}+1}$, which is much lower than 2^m .



  In the plot above, there are 8 border nodes. We can split these 8 nodes into 2 borders (red lines) since they do not affect each other. The time complexity of our algorithm after split is $2^3 + 2^5$, which is much lower than 2^8.

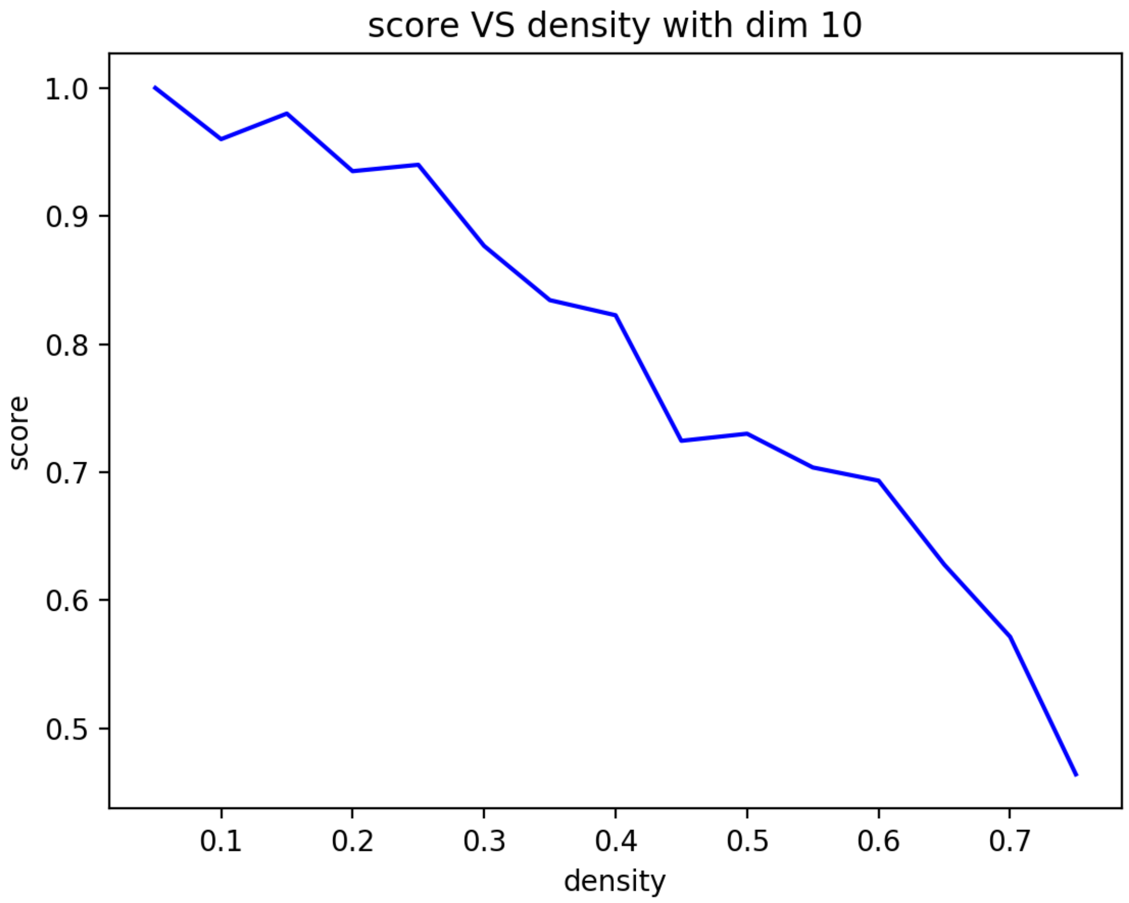
- Improvements

  We test these two algorthm with density between 0.05-0.9.

With the information of the number of mines in the environment, we can estimate the probability of a unvisited node to be mine. The probability of a unvisited node to be mine:

$$P = \frac{all\ mines - dectected\ mines}{unvisited\ nodes}$$

If the estimated probability of a uncovered node to be mine is lower than the lowest probability of all border nodes, then we should randomly uncover an unvisited node that is not a border node.

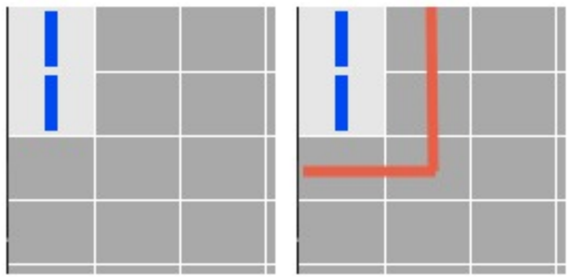


4 Dealing with Uncertainty

We mainly deal with first two situations.

1. Lost information

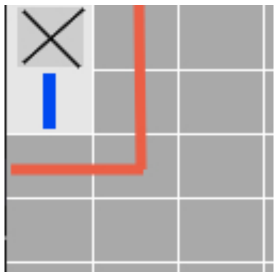
For the first uncertainty model, we have some probability to lost clues. In our algorithm, one clue corresponds to one constraint for the values of border nodes. If we lost a clue, we lost one constraint.



In the example above, we have two clues and two constraints:

$$val(0,1)+val(1,1)=1$$
$$val(0,1)+val(1,1)+val(2,1)+val(2,0)=1$$

If we lost one clue on node `(0,0)`, we have:

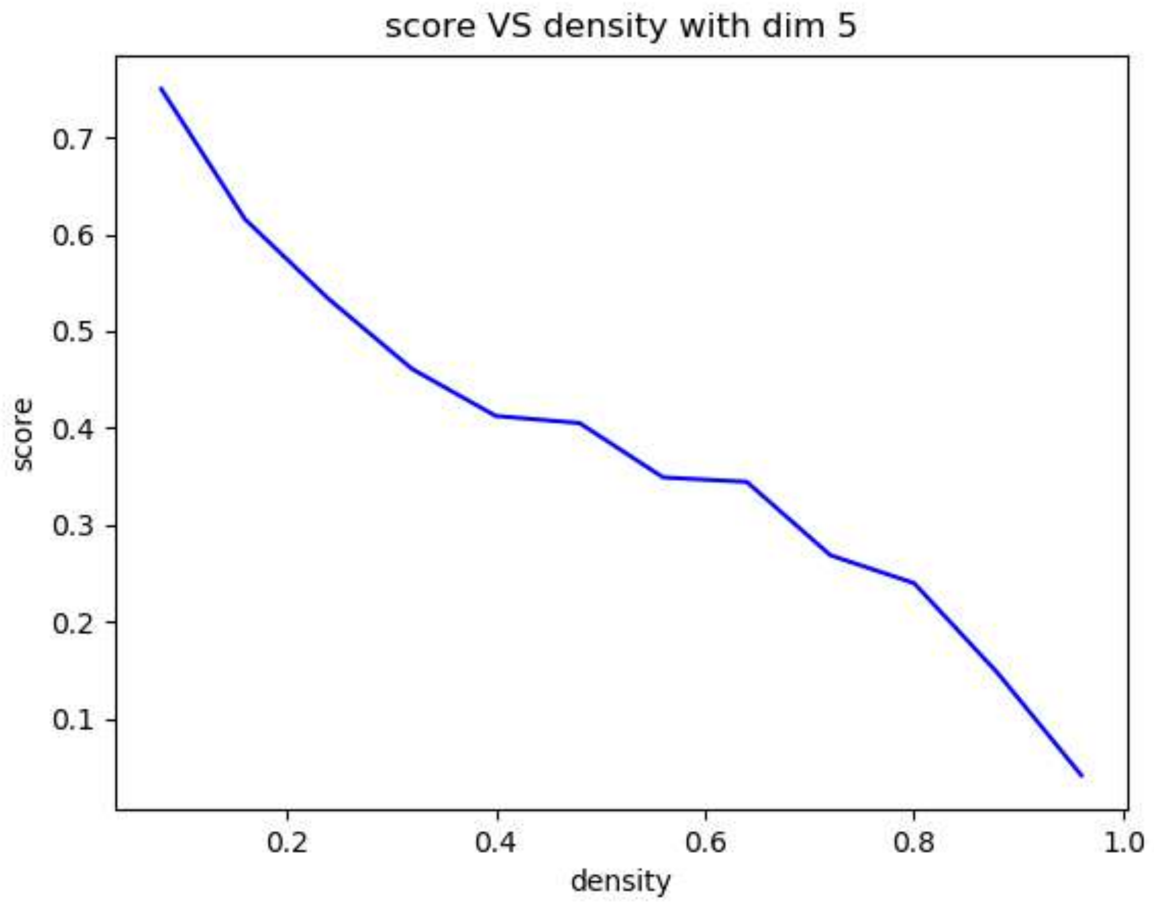


We will lost one constraint. Now we have only one constraint for the values of border nodes:

$$val(0,1)+val(1,1)+val(2,1)+val(2,0)=1$$

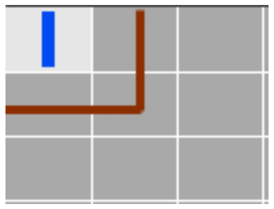
We can still perform our algorithm to estimate the probability of these border nodes to be mines, and then make decisions based on these probabilities.

We set probability of lost information to 0.5, and the plot below showst our algorithm performance.



2. Underestimating

For the second uncertainty model, the clue has some probability of underestimating the number of surrounding mines. We need to relax our constraint to solve this problem.



In the example above, the clue is 1. If this is not a uncertainty problem, the number of total mines in border nodes is 1. We have the constraint:

$$val(0,1) + val(1,1) + val(1,0) = 1$$

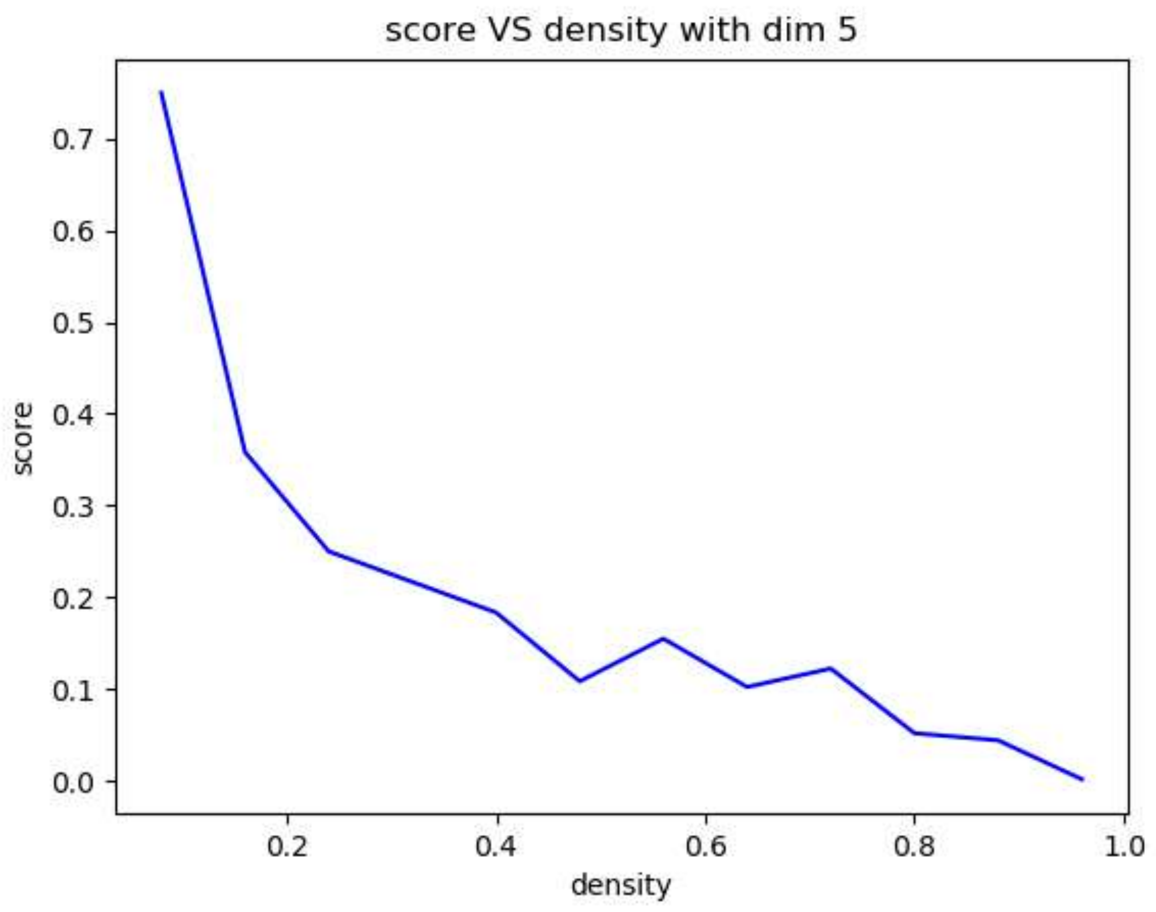
For the underestimating situation, the actual number of total mines in border nodes can be 1, 2, 3. We relax our constraint:

$$val(0,1) + val(1,1) + val(1,0) \geq 1$$

We perform our algorithm to traverse all possible cases based on the new constraint. There are 7 possible cases that satisfy the two constraints above:

(0,1)	**(1,1)**	**(1,0)**
1	0	0
0	1	0
0	0	1
1	1	0
1	0	1
0	1	1
1	1	1

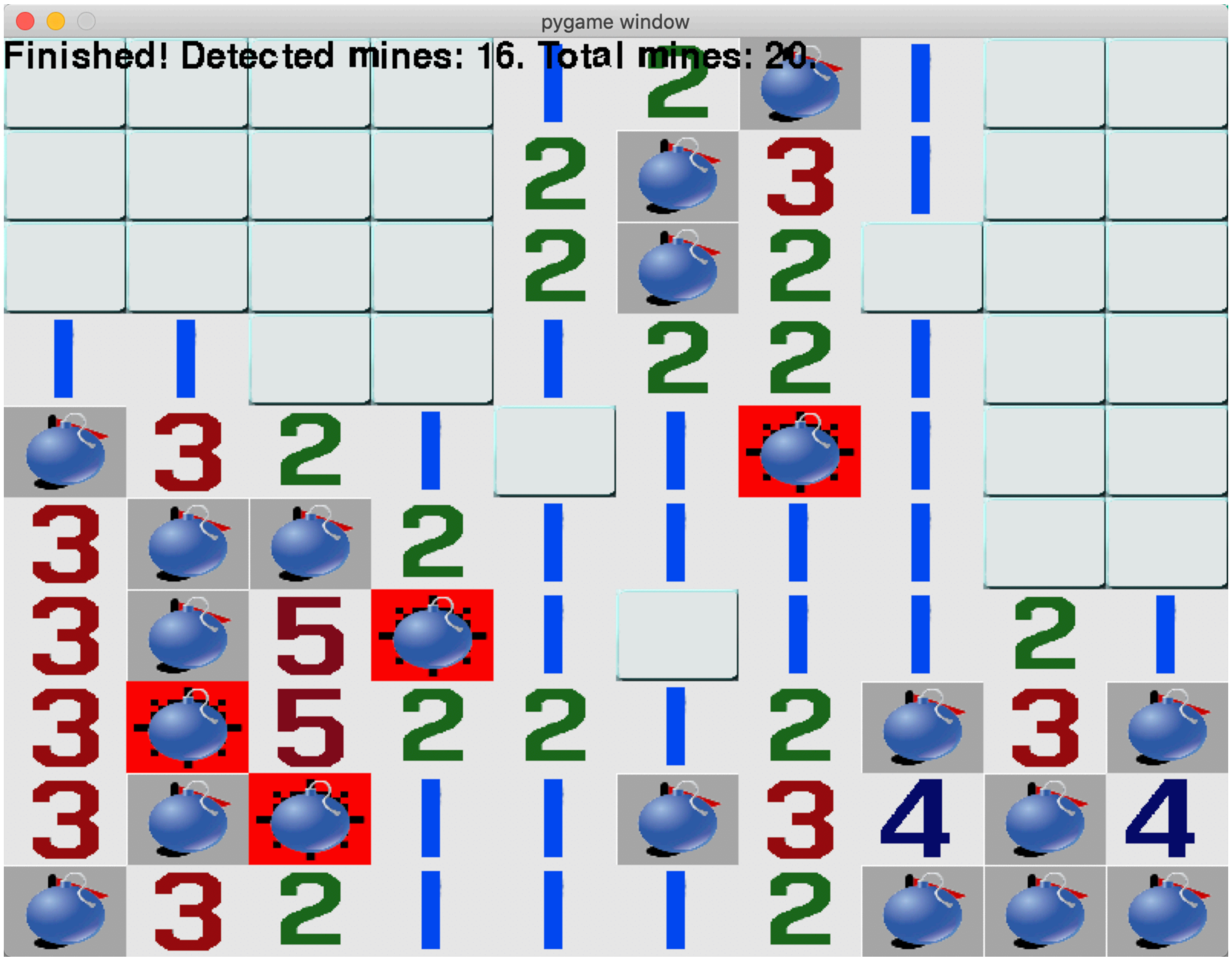
After that, we estimate the probabilities and make decisions based on these probabilities.



5 Some running results

1.Basic algorithm

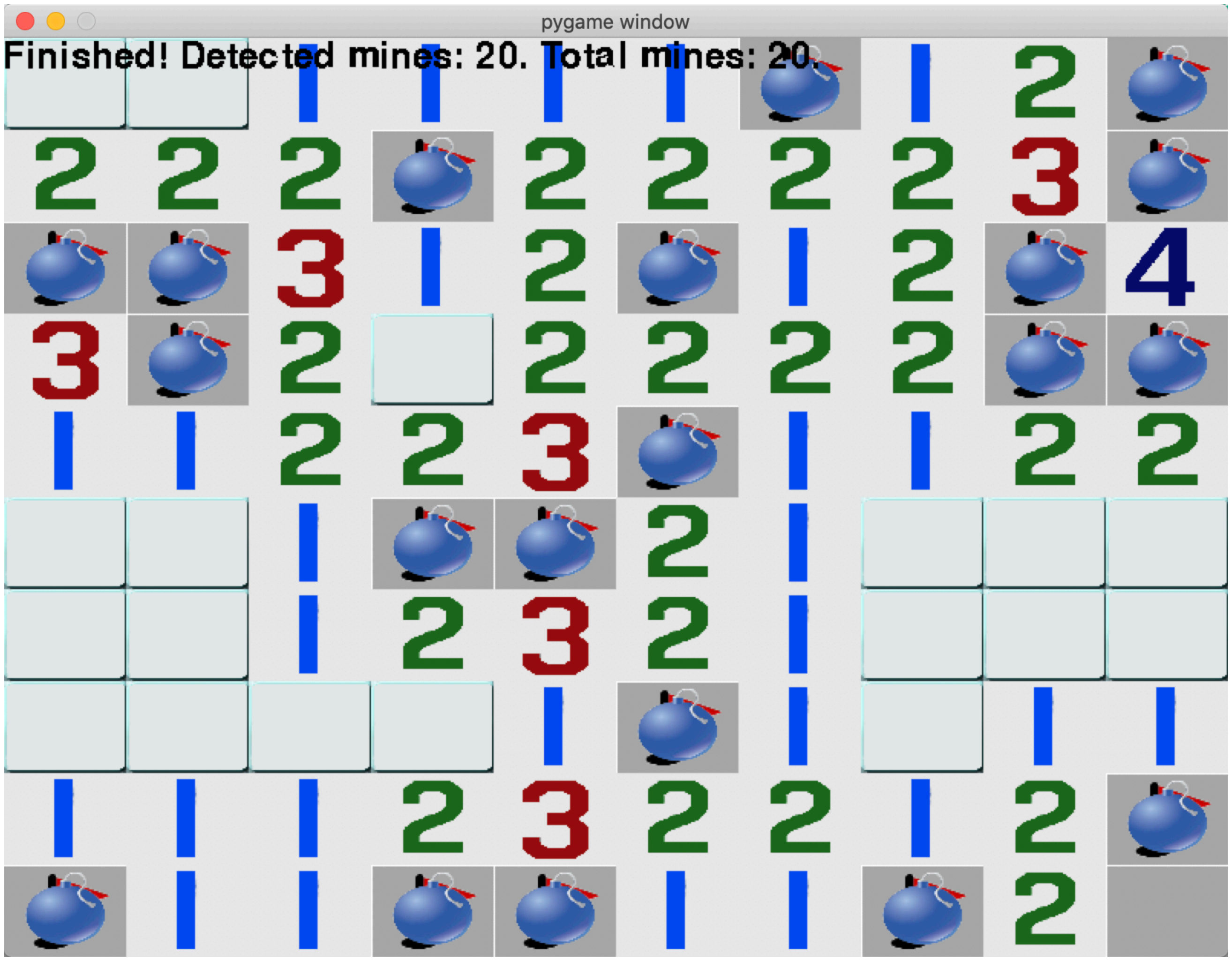
Dimension = 10



Finished! Detected mines: 16. Total mines: 20.

2.improved algorithm

Dimension = 1



Finished! Detected mines: 20. Total mines: 20.