<https://ieeecharusat.wordpress.com/2010/09/23/the-flood-modified-flood-fill-algorithm-micromouse/>

## The Flood & Modified Flood Fill Algorithm – MICROMOUSE

***FLOOD FILL ALGORITHM***

The best way to understand the flood fill algorithm is the water-in-the-maze analogy. This is how it goes,

Suppose you start pouring water in the center square of the maze (I would love to know who actually tried it first). The water will ‘flood’ the center of the maze and then will start flowing in all the adjacent squares which are ‘not separated by a wall’. If I mark the first square, that is the center, as 0 and mark the next square(s) where the water goes as 1 and keep on increasing the number as the water ‘floods’ the maze, I’ll get something like this

The water is now flooding the maze, and I am numbering the squares in ascending order. That is, water reached the square numbered 4 before reaching the square numbered 5 or 6 or 7.

Done flooding, with water! (You got a dryer or something?)

Now taking the bottom left square as the starting point, if I follow the numbers in descending order, you can see that I will reach the center through the shortest path! Amazing….yes….hard to digest….YES!

This was my reaction when I first got the hang of flood fill.

Flood Fill can be understood in another manner, something that I call the ‘potential difference’ analogy. Water is still used here [:-)] !

Suppose there are two squares A and B. I put A at a higher potential than B and pour a glass of water in A. It is obvious that the water will reach the square B taking the shortest path, given there exists a path (i.e. no wall present) between A and B.

Above point is shown at a larger scale in the following picture and this should clear any doubts in your mind regarding this algorithm (hopefully).

It should be noted here that this illustration has no connection with the ones shown before. Here also, it is easy to see if squares are chosen in ascending order of their potential from the starting square, the center can be reached by the shortest path.

This algorithm has a relatively simple implementation in the software. Two arrays should be maintained in the memory, one for keeping the flood fill numbers and the other for keeping the wall information of the maze. Every time the mouse enters a new square, it stops and checks for the walls, runs the flood fill algorithm, decides where to go depending upon the information given by the flood fill and moves on till it finds the center. Needless to say, this is not a great approach, firstly because your mouse is stopping at every square and secondly, you are flooding the maze from the center every time. This is where the modified flood fill kicks in.

### The Modified Flood Fill Algorithm 

The modified form of flood fill is nothing different from the original one, just that we don’t flood the whole maze every time. It changes only those flood values which need to be changed. Have a look at the link given below,

<http://www.micromouseinfo.com/introduction/algorithm.html>

This is a very good demonstration of modified flood fill for maze solving.

### Algorithm Formulation 

Before I start describing the various parts of my modified flood fill simulator, we need to formulate the various parts of our algorithm in simple English language.

This part has been taken from micromouseinfo.com.

Whenever we enter a cell, we check for the walls and update the wall map accordingly.

***Update the wall map***:

Is the cell to the North separated by a wall?Yes -> Turn on the “North” bit for the cell we are standing on andNo -> Do nothing

Is the cell to the East separated by a wall?Yes -> Turn on the “East” bit for the cell we are standing on andNo -> Do nothing

Is the cell to the South separated by a wall?Yes -> Turn on the “South” bit for the cell we are standing on andNo -> Do nothing

Is the cell to the West separated by a wall?Yes -> Turn on the “West” bit for the cell we are standing on andNo -> Do nothing

Now we need to call the modified flood fill to flood only that part of the maze which is required. The cell values are updated according to the following rule,

If a cell is not the destination cell, its value should beone plus the minimum value of its open neighbors.

When the cell values violate the above rule, they need to be updated.

**Update the distance values (if necessary)**

Make sure the stack is empty

Push the current cell (the one the robot is standing on) onto the stack

Repeat the following set of instructions until the stack is empty:

{Pull a cell from the stackIs the distance value of this cell = 1 + the minimum value of its open neighbors?

No -> Change the cell to 1 + the minimum value of its open neighbors and  
push all of the cell’s open neighbors onto the stack to be checked  
Yes -> Do nothing  
}

The stack empty check is not required in the program presented in this article.

The wall map is updated, required part of the maze is also flooded now what? Guess we need to make the right move.

***Decide which neighboring cell has the lowest distance value:***

Is the cell to the North separated by a wall?Yes -> Ignore the North cellNo -> Push the North cell onto the stack to be examined

Is the cell to the East separated by a wall?Yes -> Ignore the East cell

No -> Push the East cell onto the stack to be examined

Is the cell to the South separated by a wall?Yes -> Ignore the South cellNo -> Push the South cell onto the stack to be examined

Is the cell to the West separated by a wall?Yes -> Ignore the West cell

No -> Push the West cell onto the stack to be examined

Pull all of the cells from the stack (The stack is now empty)Sort the cells to determine which has the lowest distance value

***Move to the neighboring cell with the lowest distance value.***

After making the move, the above process is repeated again and again till the mouse reaches the center/runs out of gas/stops to say ‘Hi!’ to you/stops for a photograph/bangs into the wall!

<https://github.com/adam2392/ieee_micromouse>

<http://students.iitk.ac.in/projects/roboticsclub_micro2012>

<http://www.alexhadik.com/micromau5/>

what violates the number sequence is the center cell!

<https://www.gamedev.net/forums/topic/695522-how-can-i-optimize-this-flood-fill-pathfinding-algorithm/>

Arduino robot car with interrupt.