IJCSC International Journal of Computer Science and Communication Vol. 2, No. 2, July-December 2011, pp. 445-449

A NEW SHORTEST PATH FINDING ALGORITHM FOR A MAZE SOLVING ROBOT WITH SIMULATOR

Amanpreet Singh[1] and Gianetan Singh Sekhon[2]

[1]Student of Masters of Technology, Yadavindra College of Engineering, Talwandi Sabo, Bathinda, India.
[2]A.P., Department of Computer Engineering,Yadavindra College of Engineering, Talwandi Sabo, Bathinda, India.
E-mail: amangenious@gmail.com

ABSTRACT

The problem of Micromouse i.e. A Maze Solving Robot is about 32 years old but its importance in the field of Robotics is unparalleled because it requires thorough analysis and paper work[2]. Before it is implemented on the hardware it requires a great deal of planning i.e . an algorithm. The algorithm is the way or scheme using which the machine will solve the maze . The Algorithm for a Maze solving robot to find the shortest path in an unknown maze from a starting position to a destination probably the center. The algorithm plays a great role in the efficiency of the robot to solve the maze.

The Proposed Algorithm is Designed to reduce the complexity of understanding the algorithm and can reduce the memory usage and time taken for processing at each step. The proposed algorithm is just straight forward and a robot programmer can easily implement on any maze solving robot.

Keywords: Algorithm, Shortest Path, Maze Solving, Robot.

1. INTRODUCTION

One of the Main part for the design of maze solving robot is the ability of the robot to plas its motion and the method[1]. The are some algorithms available to be implemented like Wall Followers, Tremaux's method and Flood fill method. The wall followers are not that efficient because they never gaurantees to find the destination , sometimes they are caught in a loop. The Flood fill method is the most efficient one. There is no such major limitation in this algorithm or we can say that there is no such space to improve it[1]. The only thing it requires high power of hardware like high processing power and memory.At each step it requires to process all the maze in memory and have high level of complexity.

2. THE PROPOSED ALGORITHM

The algorithm that I have designed is technique in which the robot will traverse or visit almost every block of maze and finds all the available paths to destination. Each path will be compared the previously found path and the shortest path is selected. It guarantees to find the shortest path. This method may not be the fastest in some cases but it ensures to find the shortest path and have the less complexity level and will considerably less amount of memory space as compared to flood fill algorithm.

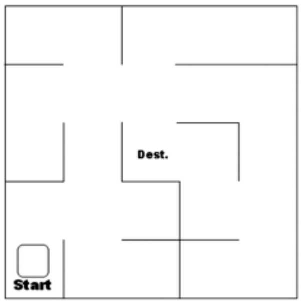
3. THE ALGORITHM RULES

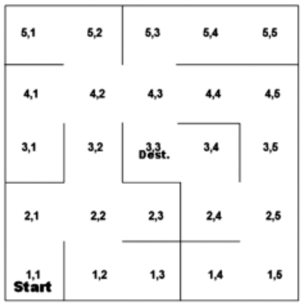
1. Almost each block of the maze will be visited. Only the blocks which are closed and the block on which the only one available path comes only through the destination position will not be visited.
2. Each block will be given the numbering in increasing order counted as the number of steps. The numbering of current route and found shortest path will be kept in memory.
3. The robot will move forward if there is no option available to move to any other side i.e. the only one path is available.
4. When there are options available on some place i.e. move left , move right or move straight means more than one routes available, then that place will be assigned a checkpoint with the number of checkpoint i.e. *c*1 , *c*2 etc. The checkpoint will also be kept in memory along with numbering.
5. From the checkpoint all available paths will be visited one by one considered as routes of the tree. The series of priority can be LCR , LRC , RCL etc. In my case it is taken Left-Centre-Right.
6. If there is a dead end on the one route, then the robot will move back to checkpoint reversing the route till checkpoint of that route blocking that route till checkpoint and will move to next route of that checkpoint and so on. And if there is another checkpoint comes on any route then all the routes of new checkpoint will be completed and then it will back to the previous one. There will be also considered as dead end when any previous incomplete checkpoint comes in the way and when

it reaches to other side the marked dead end will be considered deadend from other side.

7. When the destination is found then the number of that block and all the checkpoints of that path are noted. That gives the total no of steps taken to reach the destination using that path.
8. On the destination if all checkpoints are completed then it moves forward otherwise it moves back to complete the routes.
9. If then another path to destination is found to reach the destination then the total number steps will be compared with previous completed path. The shortest on will be kept and larger one will be set to waiting state.
10. If the destination is found using one path and another path comes on the way of that already found destination path then the current number of that path will be compared with the found number. If the previous one is less than the new path will be cancelled moving backward. Otherwise old path will be deleted till the checkpoint where both paths meet. It will be checked from the memory instead of moving the robot back.
11. If it meets the checkpoint from where it starts or passes on current route say Cx, which is not completed yet , then it will not add it again to stack. It will complete the previous stack and complete the Cx on previous position where it comes before.

For example the 5x5 maze is taken as below:



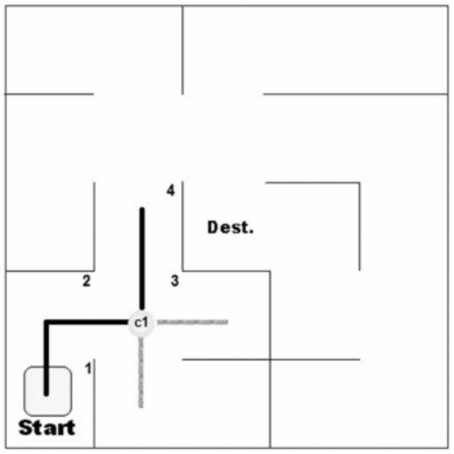


The each block of the maze is identified as the matrix as shown above.

Steps of the Algorithm for above Example Maze

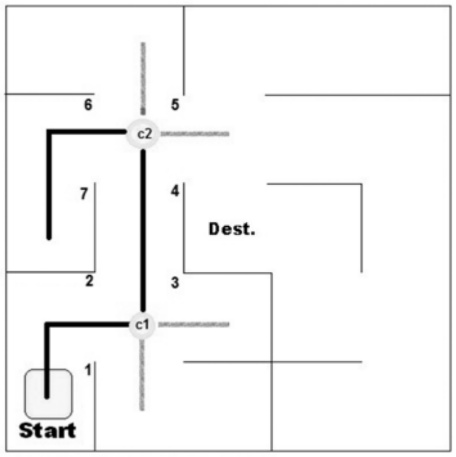
In the image 1 the robot takes move till 3rd step along with numbering the blocks until there is no option available. In the 3rd step all three options are available marked as checkpoint $c1$ marked yellow and moves to 4th. As in Stack(table) $C1$ has all 3 options. The left is in progress so it is marked D(Done) and other 2 are remaining(R).

If the checkpoint have only one or 2 options, then the route which is not there will be denoted by NA.



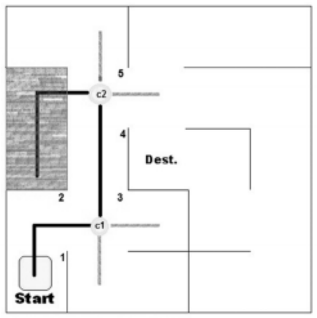
Checkpoint	*Left*	*Centre*	*Right*
C1	D	R	R

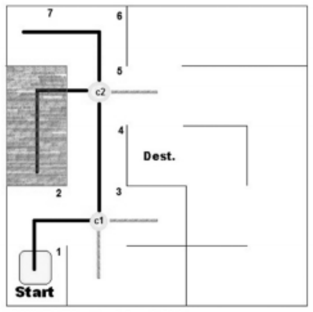
In the 2nd image in 5th block another checkpoint $c2$ comes. First of all it moves left till step 7.



Checkpoint	*Left*	*Centre*	*Right*
C1	D	R	R
C2	D	R	R

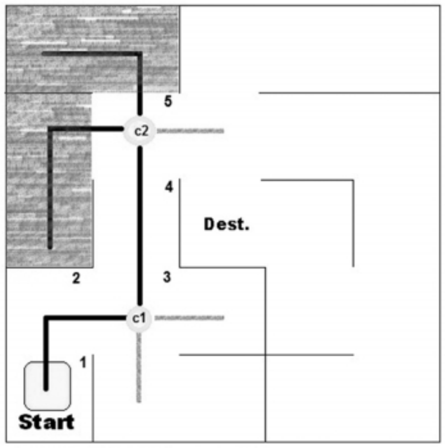
There is a dead end at step 7 so that path is blocked and marked as dead (marked red) and numbering is also reversed.





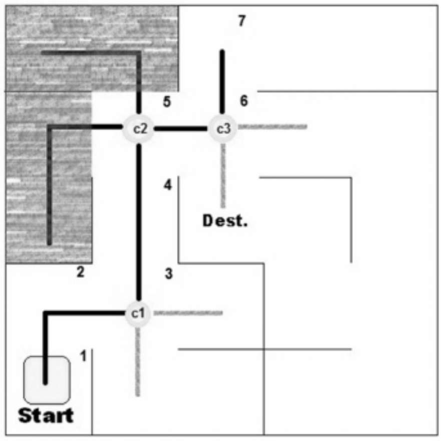
In the next picture after blocking the left side route it comes back to $c2$ and goes to next route i.e. centre.Moves till step 7 above in second image.

Here the another dead end comes same happens here. Again the numbering is reversed. Again to *c*2 i.e. the step 5. and move to next route of *c*2.

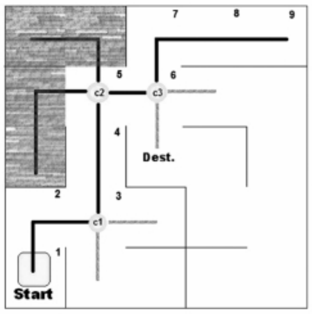


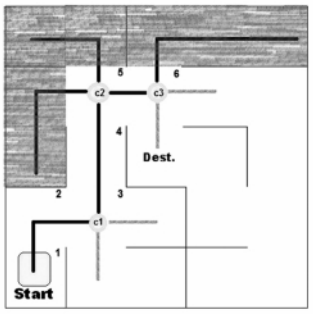
Checkpoint	*Left*	*Centre*	*Right*
C1	D	R	R
C2	D	D	R

On the step 6 another checkpoint is found. Looking at next 3 pictures the left route of *c*3 is also blocked. Because all routes of *c*2 are done so it is deleted from the stack.

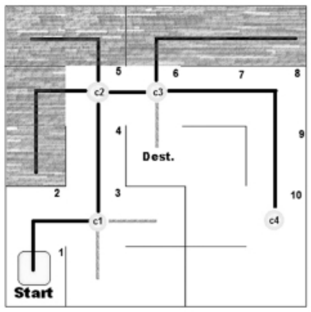


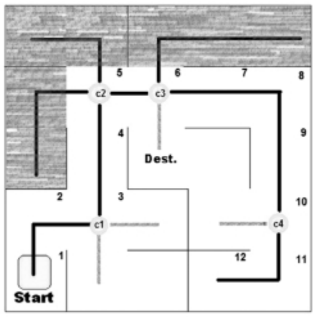
Checkpoint	*Left*	*Centre*	*Right*
C1	D	R	R
C2	D	D	D
C3	D	R	R

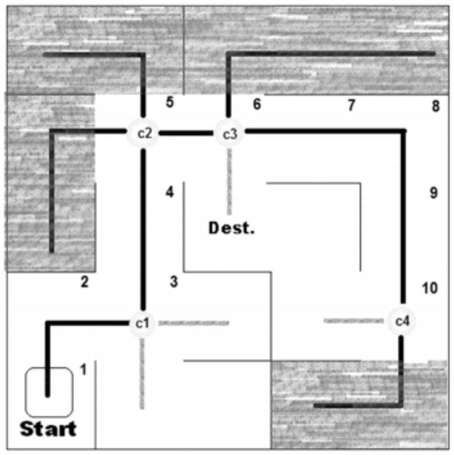




At step 10 there is another checkpoint c4 comes. There is no left node then it moves to centre. That is also dead end so blocked.

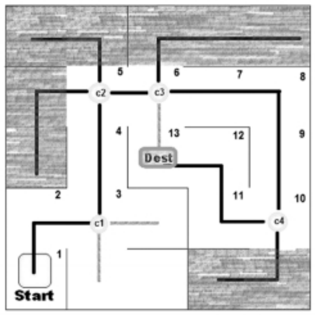


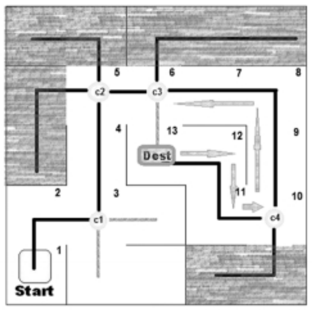




Checkpoint	*Left*	*Centre*	*Right*
C1	D	R	R
C2	D	D	D
C3	D	D	R
C4	NA	D	R

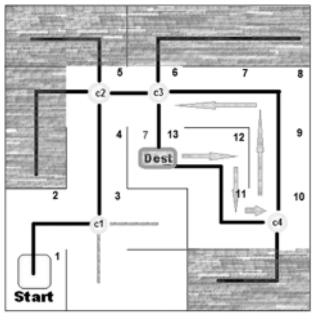
On the right node the destination is found at step 13. After Marking that position it moves back to previous checkpoint *c*4. All nodes of *c*4 are visited so again it moves back to *c*3. At *c*4 The right node is remaining.

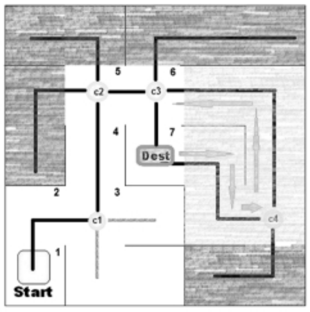




Checkpoint	*Left*	*Centre*	*Right*
C1	D	R	R
C2	D	D	D
C3	D	D	R
C4	NA	D	D

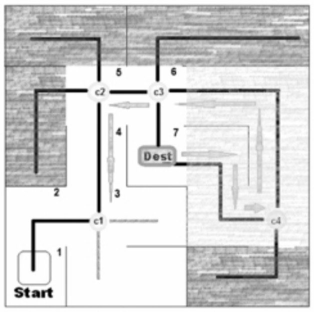
The *c*3 is on 6th step so the next step from *c*3 will be considered as 7th. On the 7th it again finds the destination. It is on 7th step so it is compared with the previous one that was 13. So the 7th is shortest, now this is shortest and previous is set to waiting state marked green considered priority path.

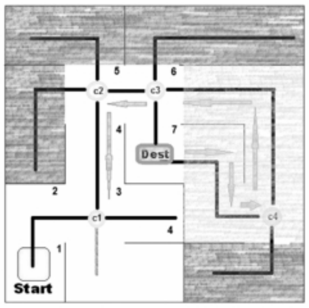




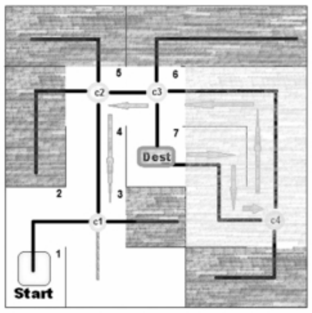
Checkpoint	*Left*	*Centre*	*Right*
C1	D	R	R
C2	D	D	D
C3	D	D	D
C4	NA	D	D

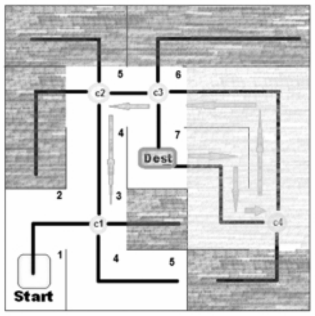
So all routes of *c*3 are completed so it has to move back to *c*2. Because the all routes of *c*2 are completed so it moves to *c*1 where centre and right nodes are still unvisited. Both are dead ends.



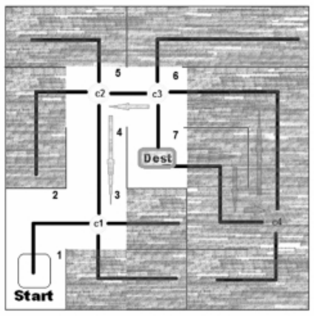


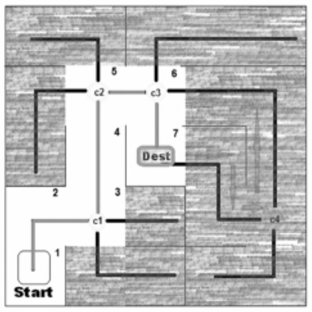
Checkpoint	*Left*	*Centre*	*Right*
C1	D	D	R
C2	D	D	D
C3	D	D	D
C4	NA	D	D





So all the blocks are checked and the remaining area is the shortest path marked with green line





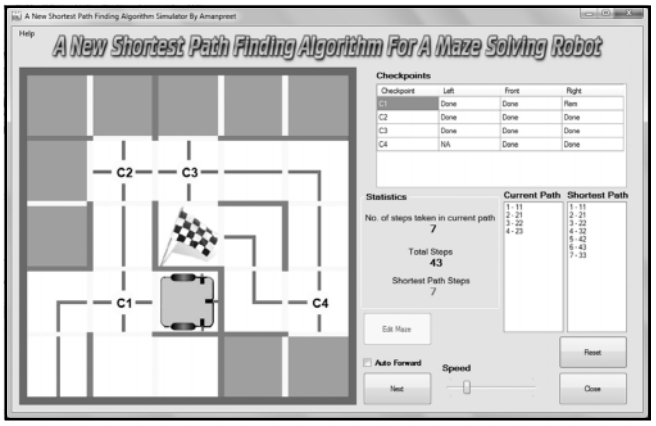
Checkpoint	*Left*	*Centre*	*Right*
C1	D	D	D
C2	D	D	D
C3	D	D	D
C4	NA	D	D

I will simulate algorithm mentioned above either in some simulator already available or will make an application to simulate. In the simulation the all process that is route, pathm calculations and number of turns will be shown.

4. THE SIMULATOR

The simulator for this algorithm is designed using vb.net. It simulates this algorithm on 5x5 maze. The maze editor is also provided so the user can design any type of maze. It provides results like total steps, steps from starting to current position including full path in matrix form, shortest path, shortest path steps, checkpoints etc. There so many features are provided in the simulator like the speed of the robot and auto or manual movement of robot steps. The code is fully dynamic and is about 3000 coding lines long and is all new idea, means it is not the modification or upgrade of any previously available simulator.

The main screen of the simulator:


A New Shortest Path Finding Algorithm Simulator By Amanpreet
Help
A New Shortest Path Finding Algorithm For A Maze Solving Robot
C1
C2
C3
C4
Edit Maze
Auto Forward
Next
Speed
Reset
Close

Checkpoints

Checkpoint	Left	Front	Right
C1	Done	Done	Rem
C2	Done	Done	Done
C3	Done	Done	Done
C4	NA	Done	Done

Statistics

No. of steps taken in current path
7

Total Steps
43

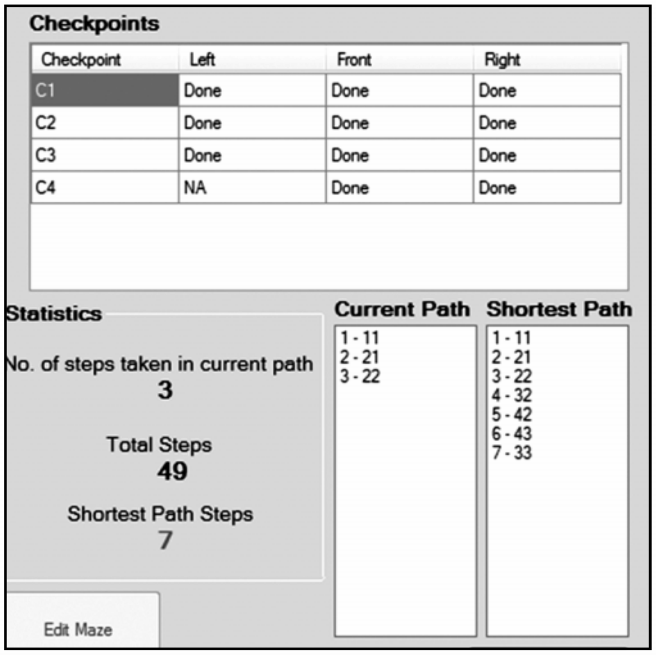
Shortest Path Steps
7

Current Path
```
1 - 11
2 - 21
3 - 22
4 - 23
```

Shortest Path
```
1 - 11
2 - 21
3 - 22
4 - 32
5 - 42
6 - 43
7 - 33
```

Figure 1: The Main Screen of the Simulator

Like for above mentioned example maze it produces the following results:



Checkpoints

Checkpoint	Left	Front	Right
C1	Done	Done	Done
C2	Done	Done	Done
C3	Done	Done	Done
C4	NA	Done	Done

Statistics

No. of steps taken in current path
3

Total Steps
49

Shortest Path Steps
7

Current Path	Shortest Path
1 - 11	1 - 11
2 - 21	2 - 21
3 - 22	3 - 22
	4 - 32
	5 - 42
	6 - 43
	7 - 33

Figure 2: The Results and Stats

As in above image the results are produced for the maze that is taken as example in previous section. The shortest path is 7 steps long and the full path is given in the matrix form. The Total number of steps taken are 49. And the table on the top describes the checkpoint status. All the figures are updated at every step by the simulator.

In the maze editor the user can click on any of the wall to toggle its status , can remove walls and can put new walls.

5. CONCLUSION

The Algorithms already there like Flood fill have no such serious limitation[1],[3] and need only is to improve the hardware and the complexity of the algorithm[2]. The algorithm is the most important part of the project[3]. The algorithm presented here may not better then the flood fill algorithm in all cases but as in flood fill it requires to flood all the maze at each point that requires the amount of processing and memory and have the complex calculations. So the algorithm presented can be used on considerably low memory and processing power. As in wall follower they never guarantees to find the destination. So can't say about the shortest path. But in this method the shortest path is guaranteed. It could be another good algorithm along with the algorithms already available.

REFERENCES

[1] Manoj Sharma and Kaizen Robeonics, "Algorithms for Micro-Mouse", 2009.

[2] Swati Mishra and Pankaj Bande, "Maze Solving Algorithms for Micromouse", 2008.

[3] Babak Hosseini Kazerouni, Mona Behnam Moradi and Pooya Hosseini Kazerouni, "Variable Priorities in Maze Solving Algorithms for Robot's Movement", 2003.