ALGORITHM

MEMORY MAPPING
The example area has a matrix of 16 X 16 cells. The whole game area is mapped into the memory of the robot assigning the values. As the cells are mapped with the numbers as shown in the figure, at each cell the robot is expected to take three decisions.

1) Move to cell which it has gone to least
2) Move to the cell that has minimum cell value
3) If possible the robot must try to go straight

It is evident that these three conditions if followed at each cell position the robot will reach the centre of the maze designated as "0".

The mapping of the cell values in the memory requires huge memory, thus an alternative method was adopted to generate the cell values at runtime.

SAMPLE ALGORITHM TO GENERATE CELL VALUES AT RUNTIME

```c
unsigned short gen(unsigned short row1,unsigned short col1)
{
    if(row1>0x08)
    {
        nr = row1 - 0x09;
        row1 = 0x08 - nr;
    }
    if(col1>0x08)
    {
        nc = col1 - 0x09;
        col1 = 0x08 - nc;
    }
    return(0x0f - (col1-0x01) - (row1 - 0x01));
}
```

Eg: consider the cell location where row = 0x08, col = 0x08. Evaluating in the formula we get the return value as '0' which is the cell value.

Maze Solving
The main objective of the maze-solving algorithm is to quickly determine from the current position of the mouse, where it has to go next in order to get to the centre of the maze and back. It uses wall information stored during the exploration of the maze to determine the quickest route from the start-square.
to the centre of the maze and back.

**Bellman Flooding Algorithm**

The Bellman flooding algorithm is a popular maze solver with micro mouse contestants and has been used by several world championship-winning mice. The standard Bellman algorithm solves the maze for the shortest route, but this is not always the quickest. To find the quickest route to the centre of the maze it is necessary to use an advanced form of the Bellman flooding algorithm, which floods with time instead of distance. The advanced Bellman flooding algorithm is based upon the following:

1) No internal maze walls exist except those specified in the competition rules; normally just the outside perimeter.
2) The target square(s) is/are numbered 0. The squares joining the target square(s), with no separating walls, are numbered by the time taken to travel there from the target square(s).
3) The squares joining the above squares, with no separating walls, are also numbered by the time taken to travel there from the target square(s).
4) The flooding stops when the square being renumbered is the current mouse position.
5) The mouse makes the best move from the current mouse position by selecting the lowest flooding number.
6) If the mouse encounters a situation where there are two possible routes, with the same Bellman number, then the mouse will select the route with the following priority: straight ahead, east, west, north or south.

Examples of the Bellman flooding and direction arrays are given below in Figures 1 and 2. Pseudo code for the iterative Bellman flooding algorithm is shown in Figure 3 outlining the key stages in the process.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>27</td>
<td>24</td>
<td>21</td>
<td>24</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td>26</td>
<td>23</td>
<td>18</td>
<td>15</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td>37</td>
<td>20</td>
<td>17</td>
<td>12</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>43</td>
<td>40</td>
<td>43</td>
<td>14</td>
<td>11</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>46</td>
<td>41</td>
<td>52</td>
<td>49</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>47</td>
<td>42</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>5</td>
</tr>
<tr>
<td>S</td>
<td>43</td>
<td>52</td>
<td>49</td>
<td>10</td>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>

**Figure 1 - Bellman Flooding Array**
Bellman Flooding Program Structure

The block diagram for the top level of the Bellman flooding algorithm (see Figure 4) outlines the five key steps in the iterative process of navigating the maze.

**Step 1:** This determines the next position of the mouse from its current position. It primarily uses information from the Bellman flooding array, along with the mouse's current direction. It only starts this step when the mouse is nearing its required destination.

**Step 2:** It comprises of the task of driving the micro mouse to the new position determined in the last step. Although this is not strictly the task of the maze-solving algorithm, it must inform the Motor Control sub-system where to go.

**Step 3:** This is to update the wall information on the move, which is automatically done by the Sensors and Sensor sub-system. The Sensor sub-system sets a flag if any new wall information is obtained.

**Step 4:** This is only executed if the wall information has changed during the
movement of the micro mouse to its required destination. This cuts down the
time to process the next position.

**Step 5:** This is the last step in the process and checks whether the micro mouse
has reached its ultimate destination. The easiest way to achieve this is to check
the current Bellman number for the current mouse's position. If this value is zero,
then the flooding direction must be switched.

![Block Diagram for the Bellman Flooding Algorithm](http://www.lboro.ac.uk/departments/el/robotics/Maze_Solver.html)

![Source:](http://www.swallow-systems.co.uk/dash/dash10.htm)

![Source:](http://micromouse.cannock.ac.uk/maze/fastfloodsolver.htm)