Title: Micromouse

Author: K Sudheer Kumar, G Hrudil

Introduction:

The aim of this course project was to work on the Firebird as a team and use it to create several useful projects. We chose Micro Mouse to do as our project since it was quite an intriguing topic and had a scope for many optimizations speaking with respect to the Firebird.

There will be a maze of 7x7 cells with walls in it. The bot has to explore the maze, find out the information about the maze, the location of the walls, the turnings, etc., and had to generate a path such that it takes least time to reach from the starting point to the goal. The goal cell is predefined to be the center of the maze. Once the bot is placed inside the machine, it is to be left on its own until it completes the task.

Description:

We used a well known algorithm called Bellman-flood algorithm. The algorithm is similar to a flood-fill algorithm except that it floods the values of a cell with time instead of distance. This means we do not get the shortest path but the fastest path. The concept behind this algorithm is to use weighted cells. The cells adjacent to each other will be incremented by '1' during the flooding. But when there is a turning, the cell value is incremented by a value of '3'. This is done in order to avoid paths with more number of turnings as the bot takes more time to complete a turning.

We maintain a 'struct'(maze) to store 'gTime'(which is the flood value), dir(the direction the bot should be moved in), str(the straight cells the bot has moved so far). Apart from this, we maintain an array(wall array) to store the wall information which is to be constantly updated. The initial array contains walls only at the borders.

Our approach to the solutions was to explore the maze as much as it is necessary to find the fastest path and generate it. At every cell, the wall

information is updated, the flood values are filled and a new shortest path to the center from “that point” is calculated. The bot reaches the center along some path(need not be the fastest path) exploring the walls. After reaching the center, the goal is changed to be the starting point we started our previous journey and the algorithm is repeated again. This recursion is carried on until the same path recurses. This final path will be the required fastest path. This is because, after several recursions, the bot would have explored all the necessary parts of the maze required to find a fastest path and hence always generate the fastest path to be same from the starting point.

Conventions used in coding:

A cell has a value 0 in wall array if it has no walls in its left or bottom direction.

A cell has a value 1 in wall array if it has a wall in the bottom and no wall in the left.

A cell has a value 2 in wall array if it has a wall in the left and no wall in the bottom.

A cell has a value 3 in wall array if it has walls in its left and bottom directions.

We only save left and bottom values to avoid redundancy of storing these wall values. Also in order to store the border walls, the wall array is a 8x8 array instead of being a 7x7 array. The last row and the first column are kind of dummy, only there to store the wall information of the borders.

References:

Firebird V Hardware Manual
Firebird V Hardware Manual
http://www.wikipedia.com
http://madan.wordpress.com/2006/07/24/micromouse-maze-solvingalgorithm/