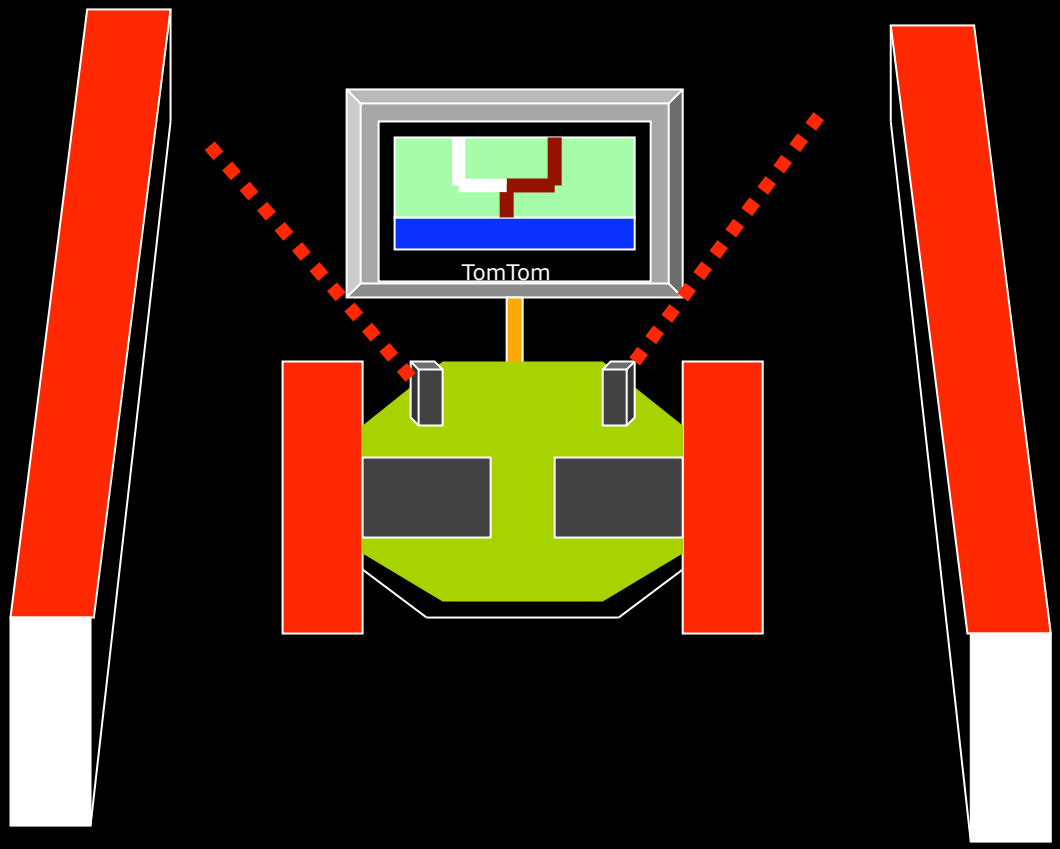
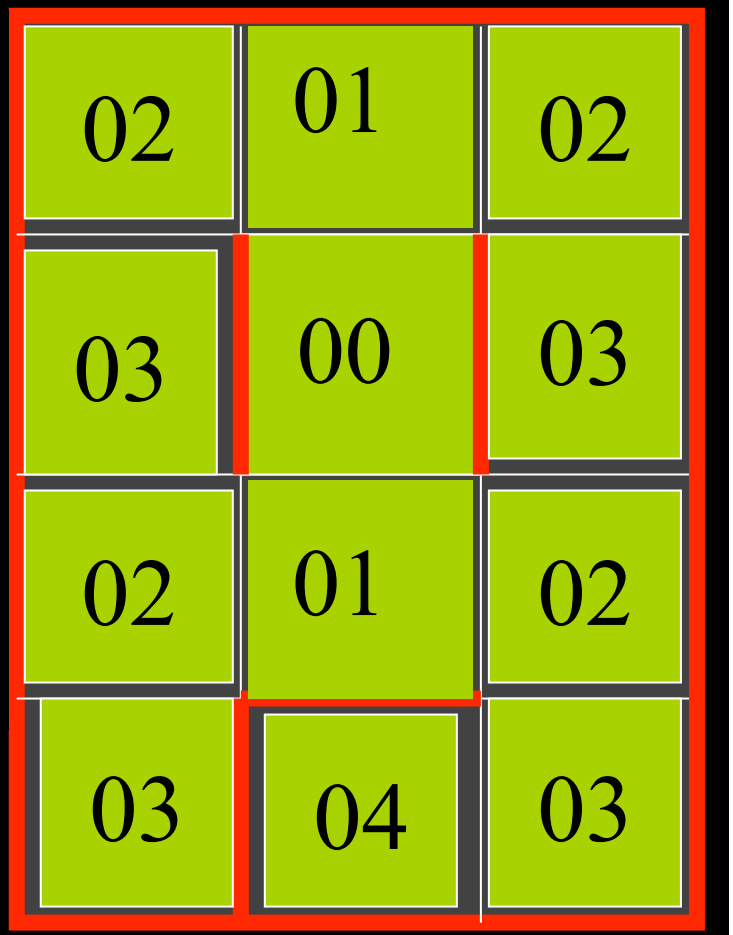
Flood algorithms

The next best thing to Sat Nav for mice !!



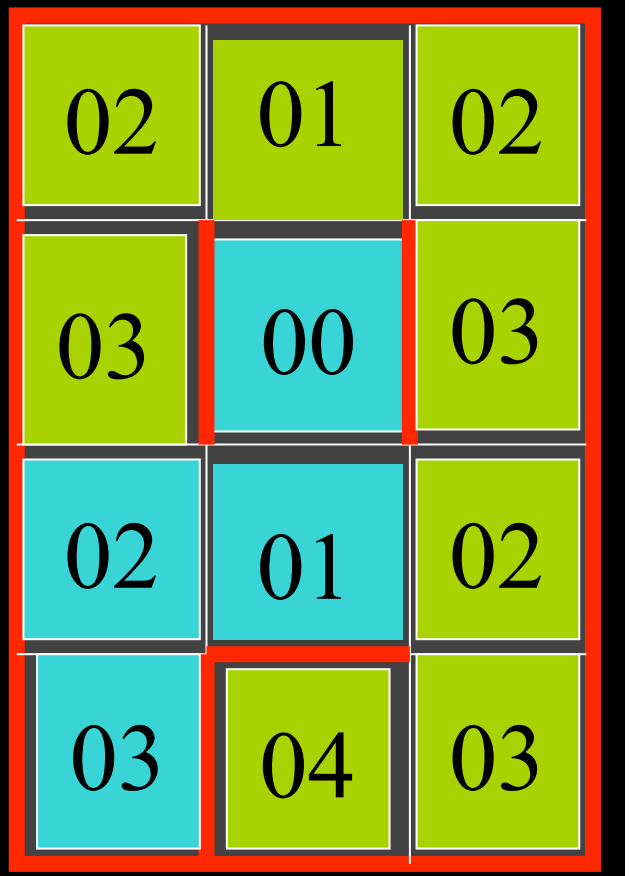
Basics – the flood algorithm

- Set all cells to maximum flood distance and middle square(s) =0. Start at centre
- Find all squares adjacent to ones just filled in & not blocked by walls. Fill them in with adjacent square value + distance, if lower than existing.
- Repeat previous step until at start



How to cross the maze using the flood info

- Set up at start & pointing North
- Get flood no from current cell
- Look for lowest flood no in adjacent cells you can get to.
- If several equal, choose the one in same direction as going now and save as the next cell to go to
- Make this one the new current cell and loop round from step 2 until until at end cell (middle square if going to centre)

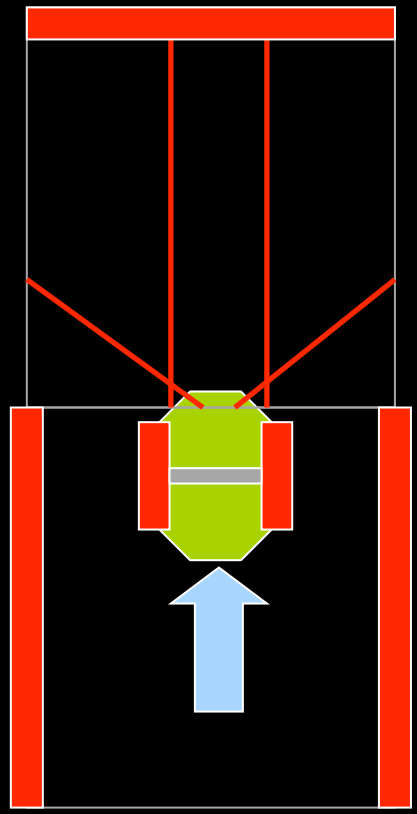


When to finish searching

- After you get to the centre or back to the start, if you run through the flood derived route and you have visited every cell in it, there is no point in exploring any more.

Why I need a fast flood algorithm

- While exploring, every time the mouse enters a new cell it checks for walls either side and ahead, then records these in the maze array and recalculates the full flood route
- Need to complete this before mouse needs to decide whether going ahead or turning left or right in new cell
- This means the mouse only explores the potentially best routes, so it should minimise the explore time



Was it complicated to write?

- Clever but not a lot of code
- My current version written in 3 pages of assembler with 117 lines of instructions
- Flood routine available to view at:

http://www.davidhannaford.com/robots/flood.doc

Code 1 - Setting up to start flood

- Set up a blank maze with known walls around the edge
- Set all flood cells to value 255
- Point at the start and end cells
- Your code should be able to flood from any cell to any other, so you can use it to come back from the centre to start, and test it on small mazes
- Put centre square number in pointer list

Code 2 – Processing flood cells

- See if any more cells to process with the current flood no
- If there are :–
- check the 4 adjacent squares

  - if wall in the way ignore it

  - otherwise replace flood cell with current flood no +1 unless cell has already got a lower number in it

  - save cell location in pointer list

Code 3 – Next flood no

- If no more cells to process with the current flood no
  - increase flood number by 1
  - save new end of pointer list
  - check if reached the start, so can stop flooding

More examples of how this works in detail from Peter Harrison in a few minutes

So how fast is it?

- Using 18F4525 PIC processor running at 32Mhz clock - gives 8MIPS
- On worst case of a 16x16 empty maze with no walls seen yet, it takes 2.84 milliseconds. (approx 23,000 ins)
- When all walls seen in 16x16 maze it takes 2.52 milliseconds.
- Even if travelling at 2m/sec we would only go 5mm in this time
- If pushed for time we only need to flood back to point where mouse is, so average flood time can be reduced to half that above. I.e. about 1.4msec

Limitations of code

- Can't handle the new 32x32 mazes
- Just minimises no of squares traversed (distance =1)
- It doesn't optimise routes using information about number of corners in route or exploiting diagonals – I'm hoping to get info on how to do this from some of the other speakers ! – see next slide

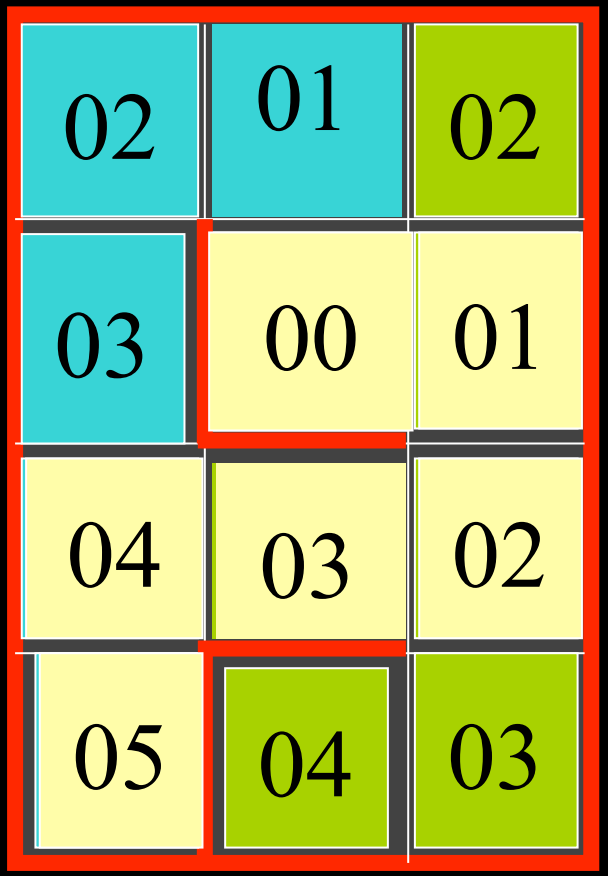
Variations on flood distance between squares

- 2 equal routes blue, yellow

I would like to know how to:

- Penalise no of turns needed
- Take account of diagonals being shorter
- Time rather than distance I.e. can go faster if more straights in a row

Should we do this when doing the flood or when extracting route ?



Questions

Then hand over to Peter Harrison