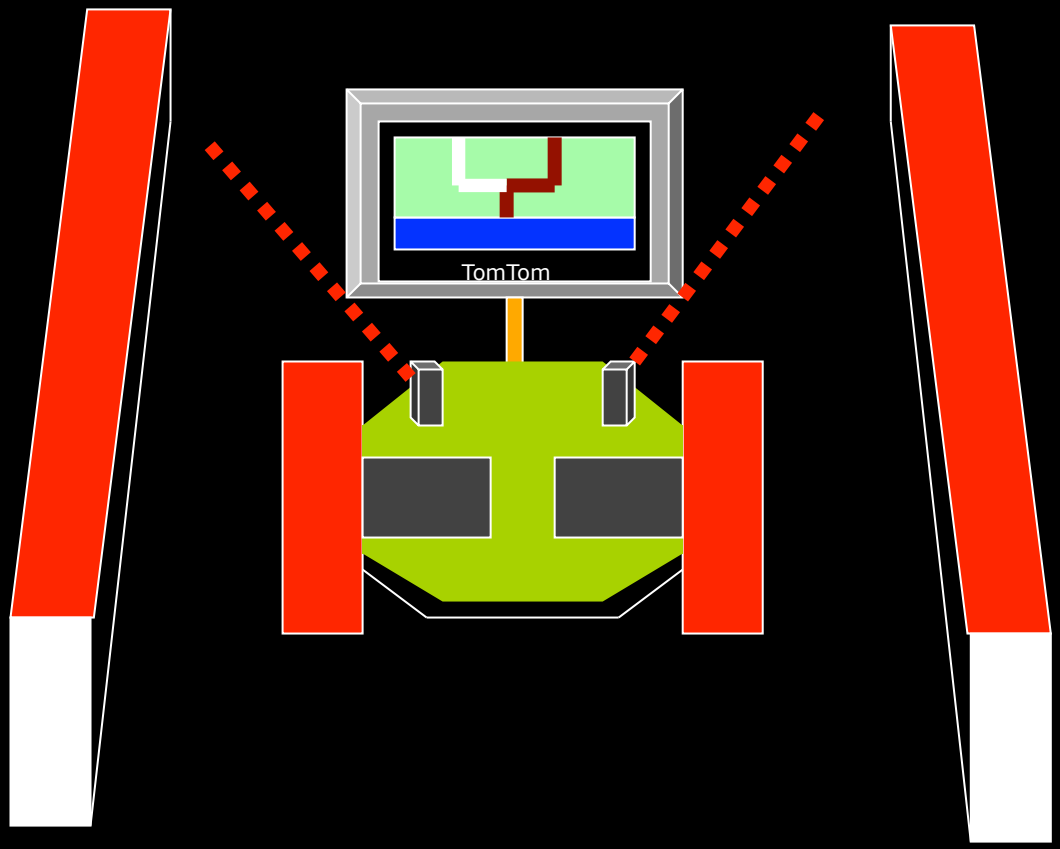
Tuning Flood algorithms

Making them 10 times faster than last year!

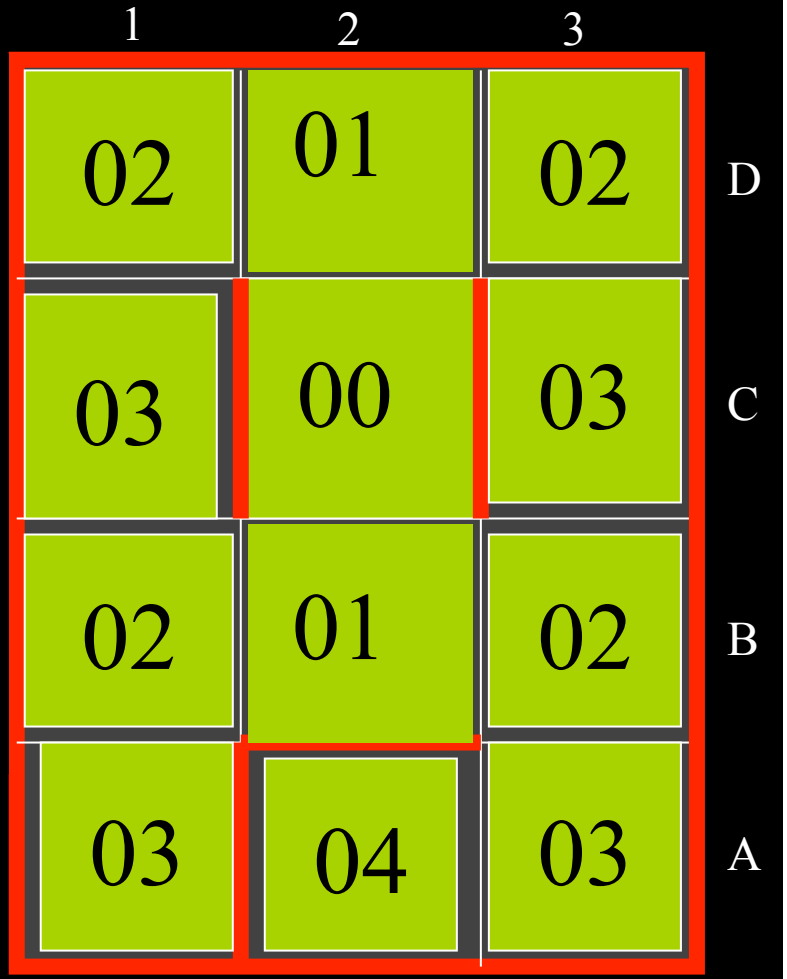


Recap on MINOS 2009

- Last year Pete and I explored ways of making the flood algorithm very efficient so as to get the CPU time down to a few milliseconds
- This involved creating a list which held pointers to the cells as we flooded them out from the centre

A refresher on 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

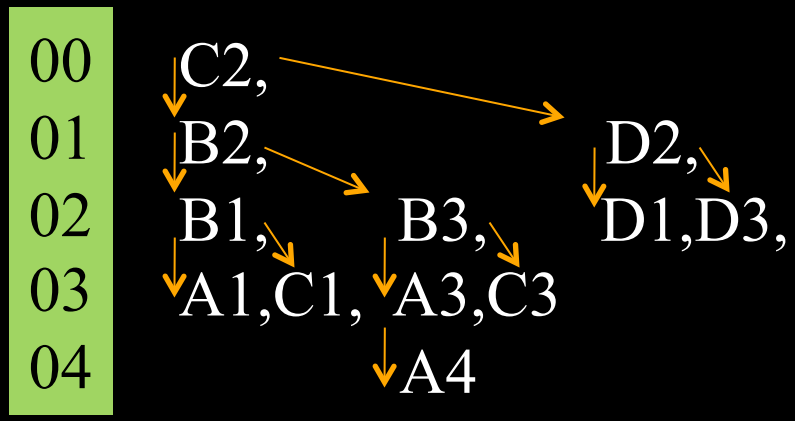


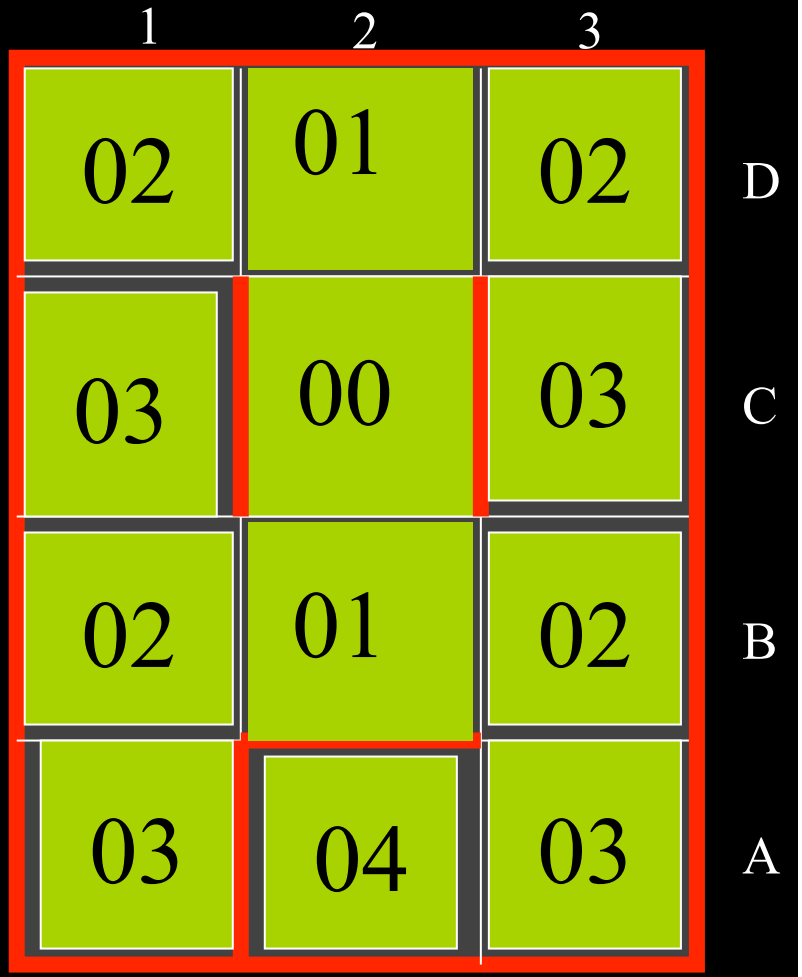
Using the pointer list

- The pointer list looks like :

C2, B2,D2, B1,B3,D1,D3, A1,C1,A3,C3, A4

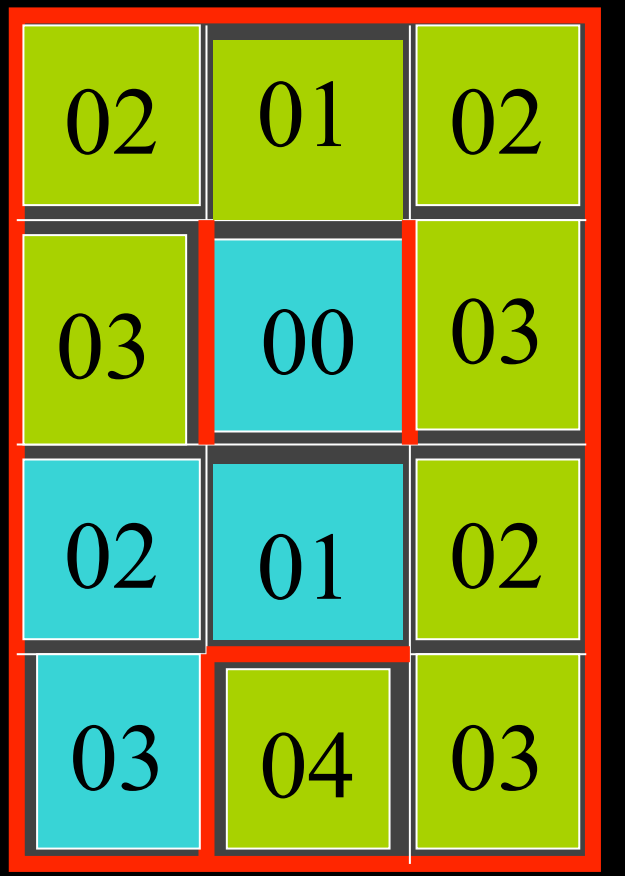
- Which makes more sense if shown as a tree structure:





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)

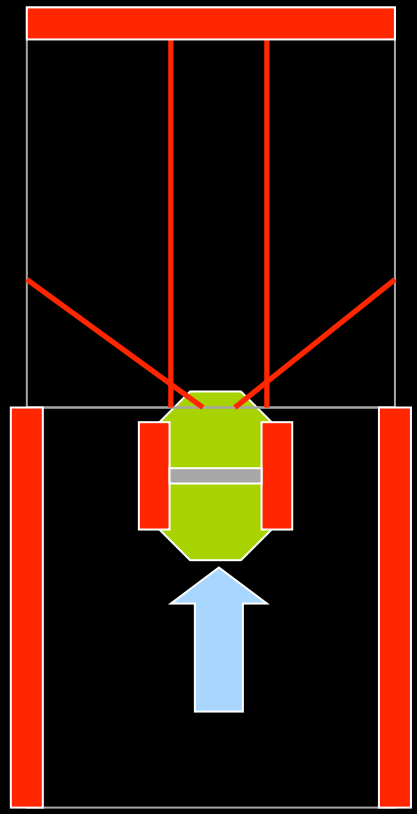


So why update this algorithm?

- It used 1 byte cell pointers, so couldn't handle a 32x32 maze or flooding where the "flood distance" between cells was other than 1 – This stops us taking account of the longer time for corners or the shorter time for diagonals while flooding
- I wanted to get the flood time down below a millisecond
- I really felt that we were missing a trick or two (or three)

When did I run the flood?

- While exploring, every time the mouse enters a new cell it checked for walls either side and ahead, then recorded these in the maze array and recalculated 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



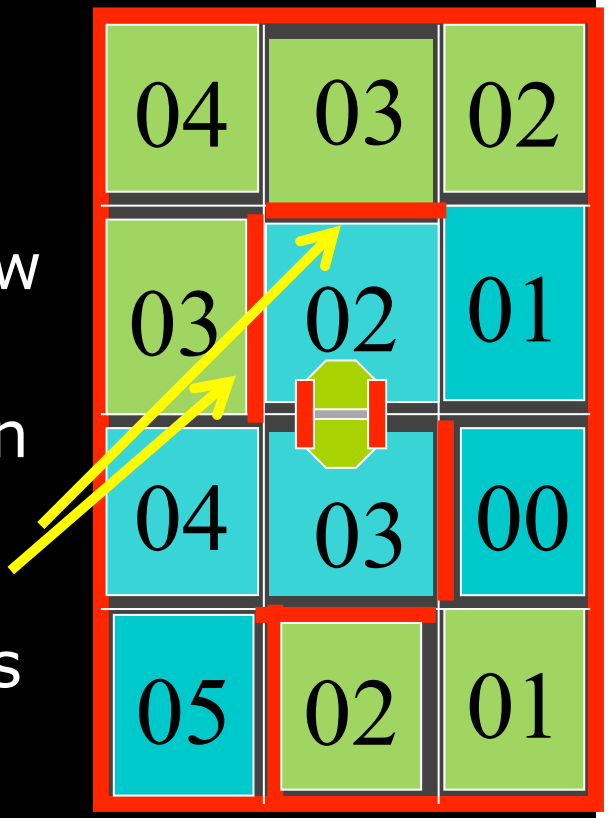
An interesting fact

- When you re-flood a maze (e.g. after seeing a new wall) the flood value of EVERY cell in the maze can only be the same or higher. It can never be lower.
- Why? – Well if could be lower you could have got to this cell by the same route before you saw the new wall, which cannot be true.
- So what !! – Well it turns out that you can use this fact to significant advantage as you will see in the next few slides

Do you always need to re-flood?

Only if you go into a cell and discover new walls on the optimum flood route

- So if you have visited this cell before you don't need to re-flood
- If you don't see any new walls in a new cell you don't need to re-flood
- If the only new walls seen are between the current cell and ones that have a higher flood number than the current cell then you don't need to re-flood, as it can only make them the same or a higher no - e.g. 03 to 03 or 03 to 05



How much time do we save?

By detecting that we don't need to re-flood:

The % times we don't need to flood depends on how often we find a wall that blocks the optimum route

The maths:

Every post must have at least 1 wall so a 16 x16 cell maze must have 17x17 posts, so at least 289 walls

Min % that wall blocks route =¼ x 289/256 =28%

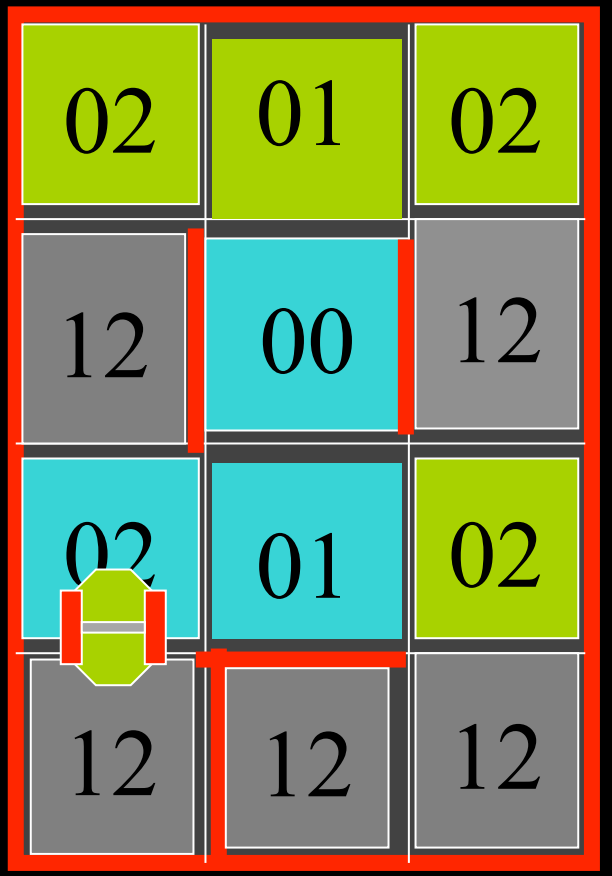
Every cell must have 1 missing wall to get into it, so max % that wall blocks route is ¾ x 289/256 =85%

Mean probability is ½ x 289/256 = 56%

So 44% of the time we don't need to re-flood at all

Do you need to re-flood the whole maze?

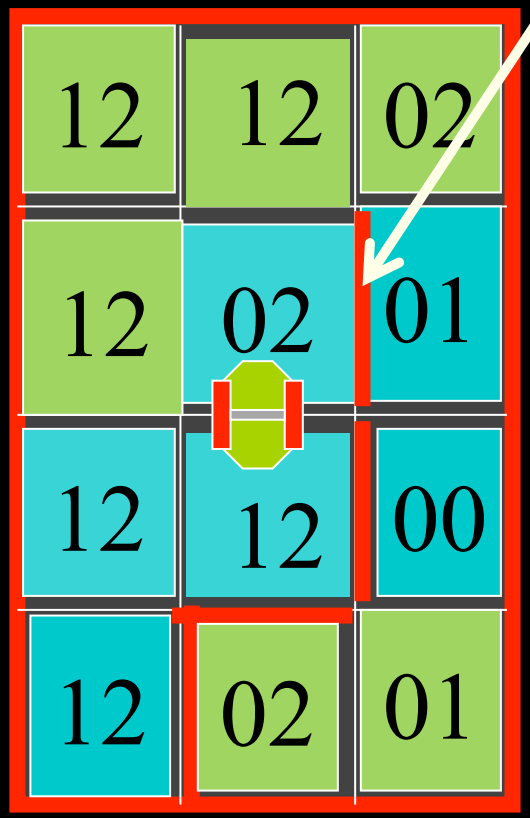
- If you are exploring, and only use the flood numbers to move to the next lowest flood number towards the centre, then when the flood reaches the cell the mouse is going into you can stop. In this case shown, on only the second cell we only have to flood 7 from 12 cells (approx 60%) and as we get nearer the centre this % reduces even more



Do we even need to always flood from the centre?

No, but where do you need to re-flood from?

New wall seen here

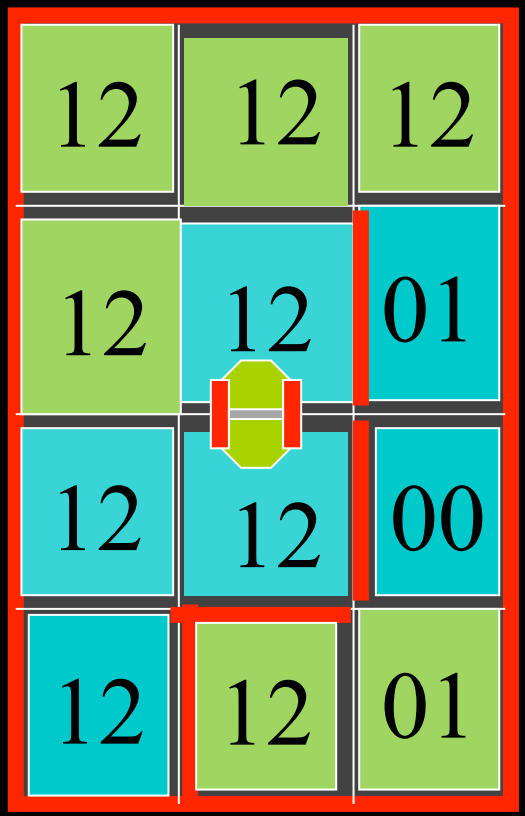


My new rules

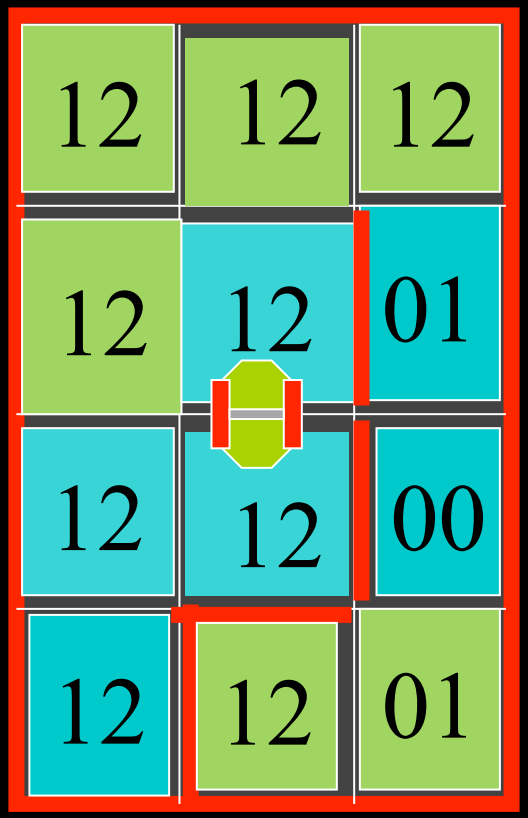
Get current cell flood no = 02

Then adjacent one with lowest no = 01

Clear all cells with current flood no and above – set a high number in them =12



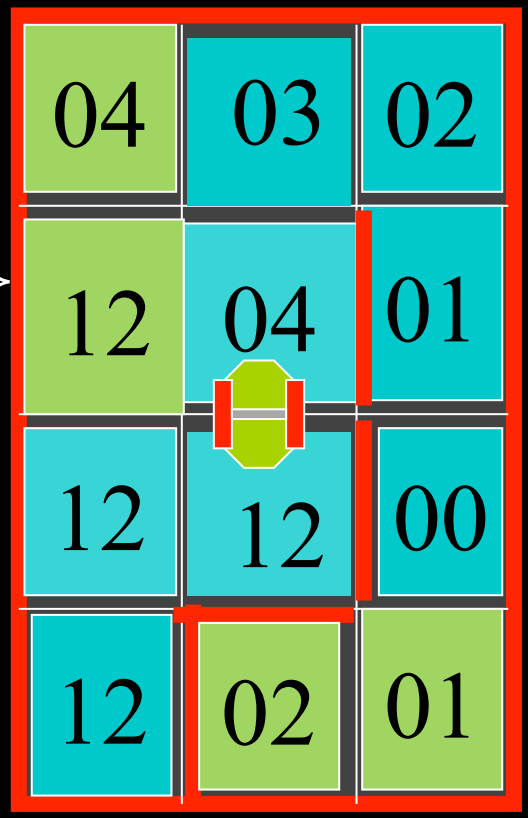
We re-flood from the lowest adjacent cell flood number



Lowest adjacent flood no is 01

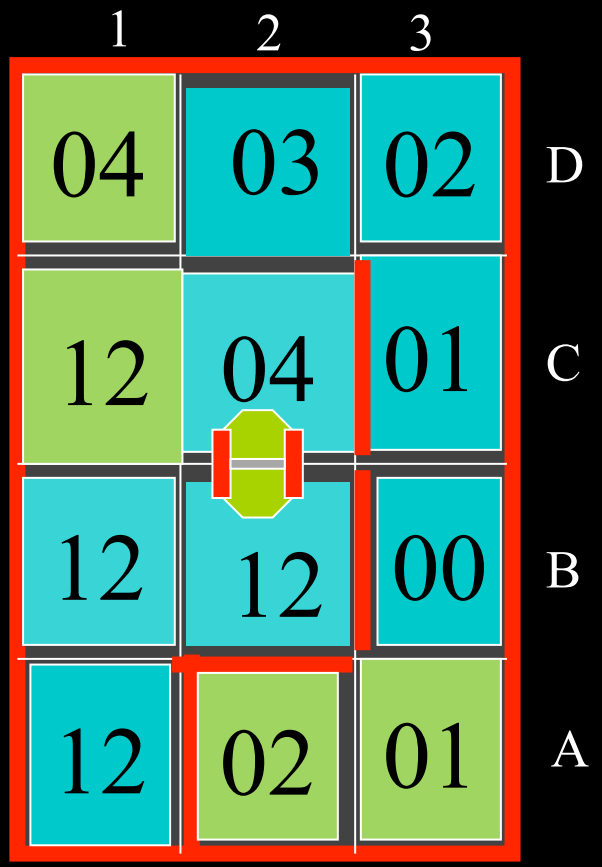
Re-flood out from cells with 01 in them to the current mouse cell

Only 4 cells re-flooded in this case



How do we get to the right cells to re-flood from?

We go back to the pointer list



00 B3, *
01 A3,C3 *
02 A2, C2, D3 *

If we add a marker * in the list every time the flood number changes we just need to go back to the relevant * in the list and restart from there. Note: The marker also holds the flood no for that mini list,

02 A2, D3 *
03 D2 *
04 C2, D1 *

Does this still work with variable distance flooding?

Rework the flood where:

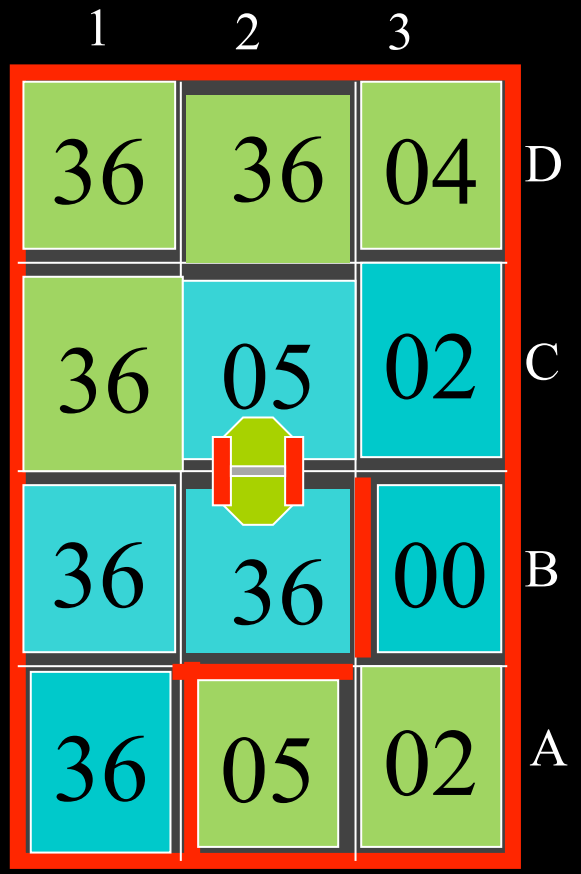
Straight ahead =2 units, Round a corner =3

- Need new highest possible no (36)
- Pointer list is:
- 00 B3 *
- 02 A3,C3 *
- 05/04 A2, C2, D3 *

Can we be sure about D2 not being better?

Only if we flood the D3 cell to give D2 as 07

The rule is you have to keep flooding until all the pointer cells flood values equal or exceed the current mouse cell flood value



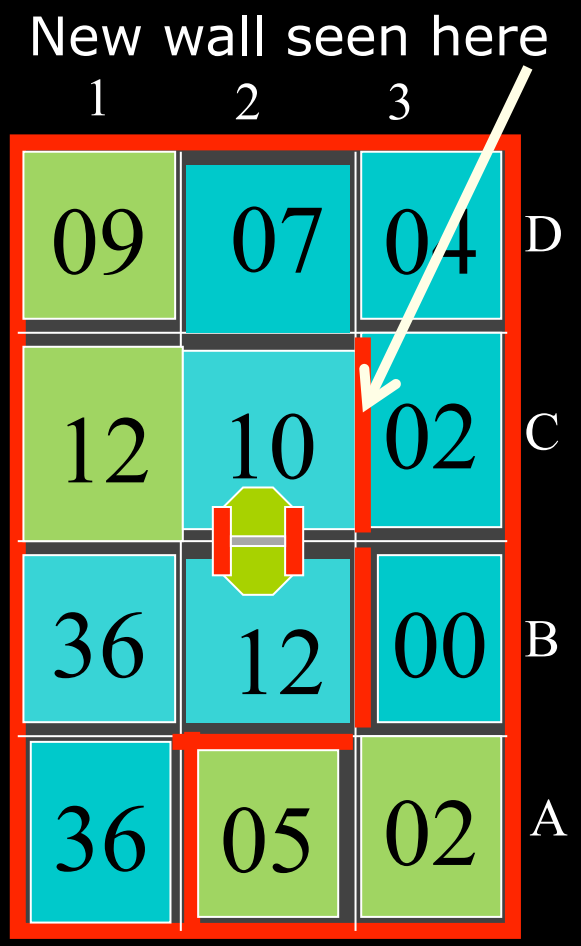
Note that with variable distance flooding you also need some extra flags in each cell to say which direction you are heading, to detect when you are going round corners.

Re-flood after seeing new wall?

Rework the flood where:

Straight ahead =2 units, Round a corner =3

- Lowest adjacent flood no to current is 02
- So re-flood from 02 cells
- Pointer list was:
- 00 B3 *
- 02 A3,C3 *
- 05/04 A2, C2, D3 *
- 07 D2 *
- Re-flood from 02 to get:
- 05/04 A2, D3 *
- 07 D2 *
- 09/10 C2, D1 *
- 12 C1, B2 *

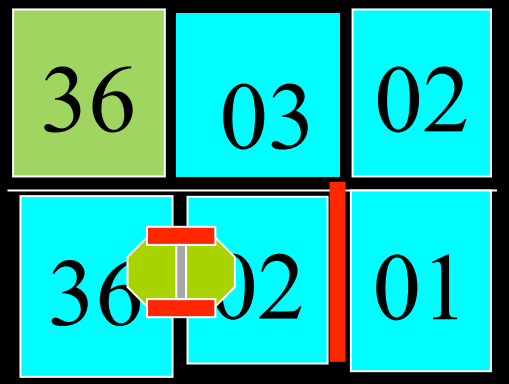


So how much faster is it?

By not flooding the whole maze. I calc 10 times faster

- A full flood on a 16x16 maze is into 256 cells
- We only re-flood if new wall blocks the optimum route, so amount of re-flooding depends on how big a detour the new wall causes.
- Minimum value is flooding 3 steps

to get round the edge of the new wall



But can expect to flood out to 2 cells

per step, so 2x2x2 (8) cells re-flooded

Typical worst case we detour 8 steps which could re-flood 2 to power 8 = 256 cells

If 85% time we re-flood 8 cells, 10% re flood 64 cells and 5% re-flood 256 cells, new average = 26 cells

Summarising

- 44% of times we don't need to do the flood at all
- If we just flood out to the current mouse position we save on average 50% of the time
- If we only re-flood from the lowest adjacent cell to the current one we save on average nearly 90% of the flood time
- And it should work with variable distance flooding

Am I doing it yet?

- Have rewritten and am using the pointer lists flood code with 16 bit cell values in current mouse
- Not flooding when visited cell before, but not detecting yet all cases when don't need to flood
- Only flooding out to the current mouse position is being used in the current mouse and seems to work OK
- Still need to code only re-flooding out from the lowest adjacent cell to the current one
- Haven't tried variable distance flooding yet

What else to speed it up?

- Have changed the processor to a dsPIC33FJ128MC804 running at 80mhz clock
- This has 128k of memory, 16k of data, plus 2 PWMs and 2 encoder trackers
- This does the full flood algorithm for an empty 16x16 maze in 15576 instructions
- Full flood now only takes 0.3894 milliseconds according to the MPLAB simulation
- So re-flood now looking at around 40 microseconds if my calculations are correct

Questions