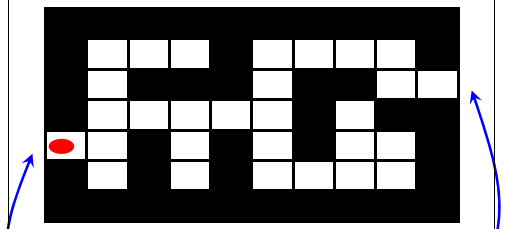
Maze

Subroutine named *nextval*
guides mouse thru a 15 x 30 maze



- move 1 position left, right, up, down
- may only move to an empty location
- north & south will be blocked
- east & west blocked except entry & exit

Specification

File = *nextval.asm* Entry = *nextval*

When called
make 1 move
then return

Specification

File = *nextval.asm* Entry = *nextval*
When called make 1 move and return

Input is four pointers

- **bp** address of the maze
- **di** address of current value of y unsigned byte in the range 1 to 15
- **si** address of current value of x unsigned byte in the range 1 to 30
- **bx** address of the current value of direction of travel of the mouse unsigned byte E=1 S=2 W=3 N=4

The output is the new mouse data

- Determine where the mouse should move
- Set **x** and **y** values in the **caller's data seg** to the new mouse location
- Set **direction** in the **callers data seg** to the new direction of travel

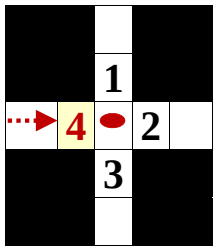
Important programming notes

- no error checking is needed
- driver detects if mouse traversed maze
- mouse is not allowed to stay in a square
- mouse may not return to start square
- do not do any file I/O
- do not keep history info between calls
- do not modify the maze
- mouse must work for all 15 x 30 mazes

How to select the next location

A ***left*** turning mouse works as follows

- try to turn left
- else try to go forward
- else try to turn right
- else go backwards



How to select the next location

A ***left*** turning mouse works as follows

- try to turn left
- else try to go forward
- else try to turn right
- else go backwards

A ***right*** turning mouse works as follows

- try to turn right
- else try to go forward
- else try to turn left
- else go backwards

How HLL access arrays

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4

Apparent 2D arrays are really linear strings in memory

0	1	2	3	4	5	6	7	8	9	10	11
1,1	1,2	1,3	1,4	2,1	2,2	2,3	2,4	3,1	3,2	3,3	3,4

*offset = (y-1) * #cols + (x-1)*

offset (1,1) = (1-1) * 4 + (1-1) = 0

offset (3,4) = (3-1) * 4 + (4-1) = 11

Can a smart assembler programmer simplify that process and get rid of those multiplications and additions?

Yes !

- the key to efficiency -

advanced indirect addressing techniques
&
a solution more clever than brute force

If you know the array size
The offsets are fixed constants

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4

Apparent 2D arrays are really linear strings in memory

- 4

0	1	2	3	4	5	6	7	8	9	10	11
1,1	1,2	1,3	1,4	2,1	2,2	2,3	2,4	3,1	3,2	3,3	3,4

+ 1

Files You Need

Retrieve *unpack.exe* from *maze* locker

- ***nextval.m*** **is the model for your subr**
- **rename *nextval.m* to *nextval.asm***

  all source code must be in *nextval.asm*
- ***mazedrvr.obj*** **is the driver program**
- **link your *nextval.obj* with *mazedrvr.obj***

Testing

testmaze maze.nn

The driver program will
- **read a file that describes the maze**
- **build and display the maze**
- **display the mouse**
- **call your subroutine for a move**
- **move the mouse to the new position**
- **check for completion and mistakes**

Six mazes named
maze.01 - maze.06

Grading

gradmaze

The final grade
- **50 correctly traversing the maze**
- **20 documentation**
- **15 number of instructions *written***
- **15 number of instructions *executed***

Submit ***maze.ans***