Mouse in a Maze

Maze Traversal Game

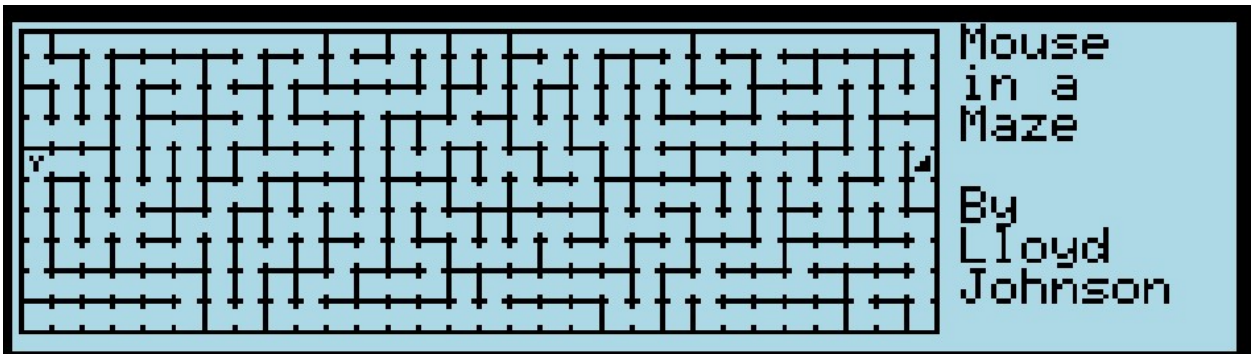
By

Lloyd Johnson

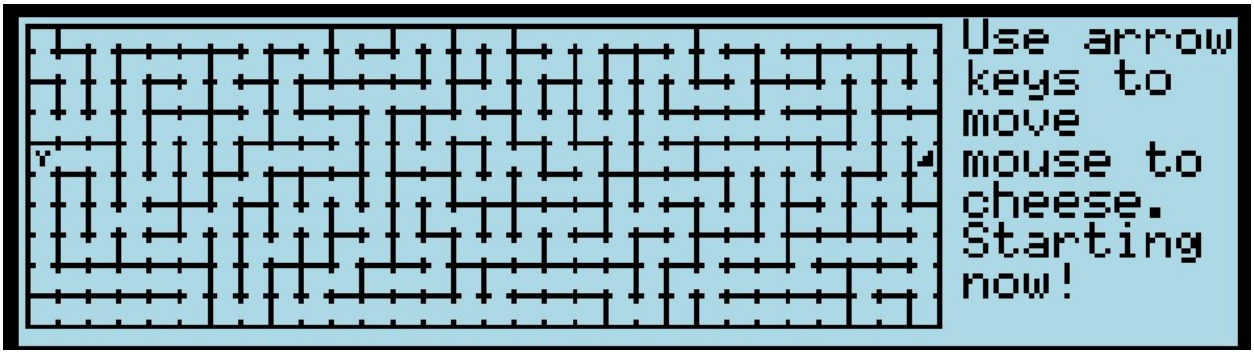
Shortly after I retired from Boeing in August 2020, I acquired an Arduino based emulation of the Altair 8800 computer. I wrote a maze game in 8080 assembly language. I called the game, MAZE029 since you were asked to select a prebuilt maze that was numbered from 0 to 9. Now, several years later, I was planning to rewrite the assembly language program in BASIC and adapt it to the TRS 80 Model 100. As I scoped out the project, I decided instead to create a new maze traversal game called Mouse in a Maze or MIAM for short. It runs on a TRS 80 Model 100.

The Mouse in a Maze game uses ten prebuilt mazes as was done in my Altair 8800 maze game. I modified the C program that generates mazes to resize the maze to10x30 rooms and to output the results as ten BASIC DATA statements that have 30 numbers per statement. And instead of the user selecting a maze, a maze is chosen randomly. The maze starting position is the symbol, ▼ which represents a mouse with Mickey Mouse ears. The goal of this game is to move the mouse towards the cheese, which is represented by the symbol, ▟.

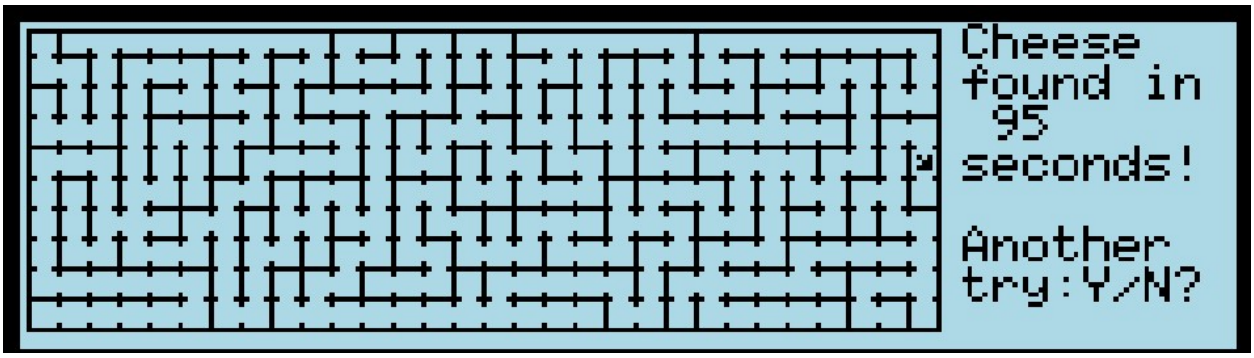
The initial starting screen after the selected maze is drawn may look something like this:



Once the screen is drawn, the right side of the display is updated to provide instructions for using the arrow keys. As stated on this screen, the user is instructed to use the arrow keys to move the mouse. The user is also informed that he should start now. This is a clue that a timer has started. This screen is displayed as follows:



Once the mouse is moved to the room where the cheese is, the message on the far right is updated as shown in the following screen:



This game also includes an autonomous mode. If the user presses the upper-case letter A, instead of an arrow key, the mouse will move on its own and will continue to move until it finds the cheese, or until an arrow key is pressed. If an arrow key is pressed, the game will return to manual mode. In manual mode, the mouse will move in response to arrow keys. It is possible to reenter autonomous mode by pressing the letter A again. If this game is hosted on the TRS-80 Model 100 emulation found at www.bitchin100.com/CloudT/#!/M100Display, it is fun to watch the mouse move at lightning speed by pressing the TURBO button when in autonomous mode.

I initially tried to implement the autonomous mode by simply selecting a door at random for each new room the mouse enters. Far too often the mouse would select the door it just came from resulting in the mouse oscillating back and forth between two rooms until it finally selects a different door. This was too painful to watch so I decided to spiff up the algorithm for mouse movement. This was done by establishing three requirements that are followed whenever the mouse enters a new room.

1. If the new room has doors leading to rooms that have never been visited, one of these doors shall be chosen.
2. If there are no doors to an unvisited room and if the new room has doors other than the one just entered, then one of these doors shall be chosen.
3. If the only door in the new room is the door that was used to enter the new room, then this door shall be chosen.

To evaluate requirement 1, it is necessary to differentiate between the rooms the mouse has visited and the rooms that were never visited. This is done by leaving behind a bread crumb. Whenever a room is visited, a bread crumb is deposited. Each room of the maze is defined by an element of the array, M. M has been dimensioned to 300 with each element corresponding to the room. The bits of the lower byte of this element are defined as follows:

Description	Upper Nibble				Lower Nibble			
Bit Position	7	6	5	4	3	2	1	0
Power of 2	2^7=128	2^6=64	2^5=32	2^4=16	2^3=8	2^2=4	2^1=2	2^0=1
Definition if set	Not Used	Not Used	Not Used	Bread Crumb	North Door	South Door	East Door	West Door

It should be noted that bit position 4 represents the bread crumb (if it exists). Bits are set using the BASIC logic operator OR and are tested using the BASIC operator AND. For example: A bread crumb is dropped at the current position, CP by setting the bit at position 4 with the following BASIC statement:

M(CP)=16 OR M(CP)

The bit at position 4 is set and the other bits of the array element are unaffected. This operation is performed at line 470. The nonexistence of a bread crumb is tested at lines 1110, 1130, 1150 and 1170 by using the AND operator to isolate the bit at position 4 and test if it is 0. If it is 0, then there are no breadcrumbs, and the room has not been visited.

In a similar manner, the AND operator is used on the lower nibble to determine if there are doors in the north, south, east and/or west directions. This is done at lines 390, 400, 410 and 420 and at 660, 670, 680 and 690.

A description of the variables and where they are used is in the table below:

Variable	**Description**	**Where It is Used**
A	Autonomous Mode, =0 not autonomous, =1: autonomous	390 400 410 420 430 450
A$	Key pressed string variable	380(2) 390 400 410 420 430 600(3) 610
AD(4)	Available doors array. Subscripts 0-3 correspond to N, S, E, W accordingly. If a door exists the value is set to 1, 2, 3, or 4	40 660 670 680 690 750
BD	Back Door - Used during autonomous mouse movement to indicate the	700 710 720 730 750

	door to going back to the previous room.	
BP	Beginning Position	240 250
CP	Current Position	250 390 400 410 420 470(2) 500 660 670 680 690 960(2) 1040(2) 1060(2) 1080(2) 1100(2) 1110 1130 1150 1170
D	Dummy variable used to store a random number during the randomizing of the random number sequence.	50
D	Door Selection. Value is 1,2,3 or 4 corresponding to N, S, E or W.	380 390 400 410 420 440 480(2) 740 750(2)
DC	Door Count.	650 660(3) 670(3) 680(3) 690(3) 750(2)
EP	Ending Position	260 500 900(2)
I	Index Variable	50 150(2) 170(3) 180(3) 190(3) 200 210 220 290(2) 520(2)
IX	X coordinate of maze position.	190 200(2) 210(2) 900 910 920(2) 930(2) 960 970(2) 980(2) 1010(2)
IY	Y coordinate of maze position.	190 200(2) 210(2) 900 910 920(2) 930(2) 960 970(2) 980(2) 1010(2)
M (300)	Maze array. Dimensioned to 300. Array elements used are indices 0-299. Value of each element represents which sides of the cell have doors.	40 150 200 210 390 400 410 420 470(2) 660 670 680 690 1110 1130 1150 1170
NC	New Room Door Count.	650 740(2) 1110(3) 1130(3) 1150(3) 1170(3)
NR(4)	New Room array.	40 740 1110 1130 1150 1170
PD	Previous Door.	480 700 710 720 730
S	Seconds.	50(2)
SM	Selected Maze - values range from 0 to 9.	130 780 790 800 810 820 830 840 850 860 870
TB	Time Begin. This is the begin time in seconds.	370 560
TE	Time End. This is the end time in seconds.	550 560

I hope you enjoy the maze.