

This was a Science Fair project by Brian Dickens which won 1st place in the computer science category at the Fairfax County Regional Science and Engineering fair in April of 1989. The project work would have been done when Brian was 13 years old.

For a blog entry summarizing interesting and funny parts about the project, see:

<http://hostilefork.com/2013/12/11/funny-maze-solver-from-age-13/>

PURPOSE

The purpose of this project is to design a program capable of simulating artificially intelligent robot "mice," and to compare the maze solving abilities of the basic types. Using data gathered on the different mice, I will try to incorporate their strong points to create a mouse with improved maze solving capabilities.

HYPOTHESIS

My hypothesis was that mice with the ability to learn would solve mazes faster than mice who pick directions at random. I also expected that already trained mice would always follow the same path in the same maze because they would take a previously learned course. Therefore, if they solved the maze the first time, they should solve it in the same way every time.

MATERIALS

- Commodore 128
- 1541 Disk Drive
- 1084 Monitor
- Floppy Disk
- 1525 Printer

PROCEDURE

First, I drafted a flow chart of the way the program would work. Then from this flow chart, I wrote the computer program that created the mazes and ran the mice. One type was the random mouse. It chooses directions for every move at random. The other type was the learning mouse, which records its initial responses which it picks at random, and then recalls them before every subsequent move. I devised a way to give the learning mouse a memory bank that would be large enough for any maze.

Once the learning mouse appeared to be working properly, I had to prove it was learning. The way mice get their directions is by picking a random number from one to eight and then looking ahead in the corresponding direction to see if there is a wall. If there is no wall, the response is valid, and the mouse moves in that direction until it hits another wall. If there is a wall, then the mouse tries to pick a new direction.

I compared the number of times random and learning mice found a valid direction out of the total number of directions tried in a three by four box (successful moves divided by attempted moves). I did this with ten mice for each category of 10, 20, 30, 50, 100, 500 and 1000 attempted moves and took the average. I did one mouse for each category of 5000 and 10000 attempted moves.

Next I put the mice in an obstacle maze to see how they differed and what their maze solving capabilities were. I ran twenty mice in the maze, ten of them were random and ten of them were learning. I then recorded the total moves for each mouse. Then I tried adding a non-backtracking routine to the mice that prevented them from picking a direction that would take them back the way that they had been going. I ran twenty more of these mice in the same obstacle maze, ten non-backtracking random and ten non-backtracking learning. I recorded the total moves for each mouse.

I tried to eliminate the problem of “endless loops” in learning mice by having them modify the maze as they go along with a trail. They would regard each trail square as a wall. Another intention of the trail was to keep the mouse from going back into places it had already been. I tested three learning trail mice and three random trail mice for each of five different mazes and recorded the total moves for each mouse.

When I saw how many moves were wasted by the backtracking trail mouse at dead ends, and how the non-backtracking mouse got in its own kind of mousetraps, I created a new trail mouse that disregarded its trail if the mouse was completely blocked in by it. I tested three of these new learning trail mice and three new random trail mice in each of the same five mazes and recorded the total number of moves taken by each mouse.

After seeing all the problems that all the other mice encountered, I finally developed a “triple trail priority mouse.” This mouse leaves three kinds of trails, uses a priority chart to tell it which trails are oldest, and disregards the oldest trail when it boxes itself in. I ran three learning triple trail priority mice and three random triple trail priority mice in each of the five mazes and recorded the total number of moves for each mouse. Just for comparison, I ran three of the original learning mice and three of the original random mice in each of the same five mazes and recorded the total number of moves for each mouse.

METHODS

The way I made it so that learning mice would learn was to assign a value from 0-255 to every possible situation it could come across. I had 2048 bytes of memory to use. A byte is made up of eight bits, which can be either off or on. Binary is the way of representing bits, with ones being on and zeros being off. Binary place values are powers of two, like tens in our decimal system. In decimal, 1234 is the same as $1000+200+30+4$. In binary, $1111=8+4+2+1$, $00000000=0$, and $11111111=255$.


I assigned each of the binary digits in a byte (represented here in decimal powers of two) to the surrounding walls:

128	64	32
16		8
4	2	1

When the learning mouse bumps into a wall, he remembers the direction he came from. Then he looks one square in each direction, and if there is a wall he adds the value for the corresponding box to B. If the mouse is surrounded entirely by walls (a hopeless situation) the value is $128+64+32+16+8+4+2+1$, or 255. If there are no walls at all (also impossible, since a mouse must bump into a wall before it will ever reach this routine, and there would be no walls for it to have bumped into), the value would be $0+0+0+0+0+0+0+0$, or 0.

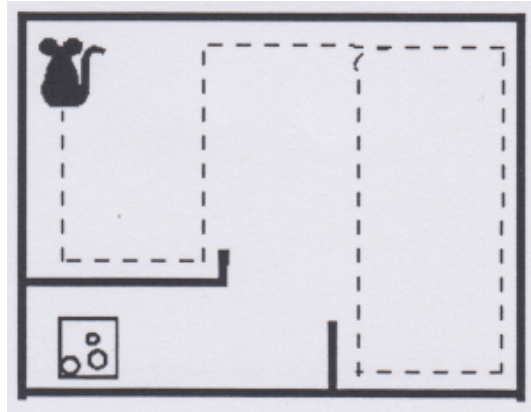
Once the learning mouse has its situation value, it still needs to get a new direction from memory. Learning mice, as mentioned earlier, have 2048 bytes of memory. This memory is divided into eight banks of 256 bytes each. Mice know what directions they are going in by storing it as a number between one and eight.

This figure shows which numbers represent which directions:

1	2	3
8		4
7	6	5

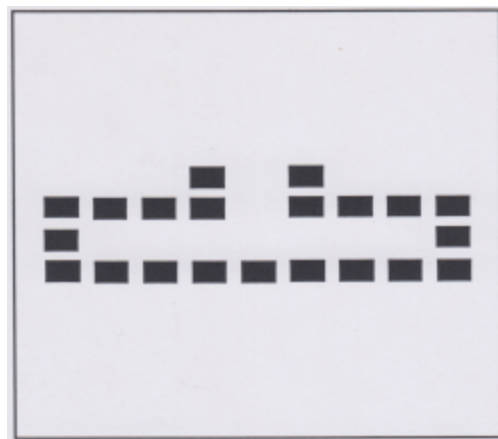
When the mouse got its situation value, it remembered the direction it was going in. It looks in the memory bank of that direction (0 - 8), at the situation specified location (0 - 255), and in that memory location it finds the direction it remembered. If it finds a value of zero in that location, it picks a valid response randomly and stores it in that location.

The learning system allows for any size maze that a mouse could ever be put in. However, it also opens it up to the deadly problem of endless loops. An example of an endless loop is below:

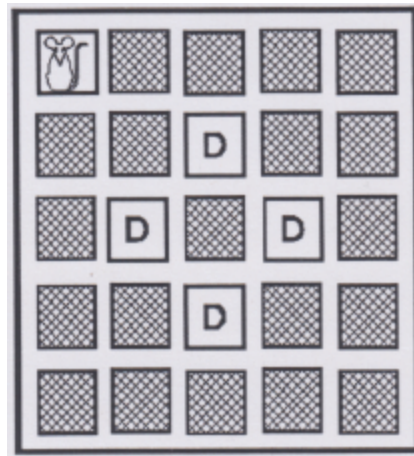


When the mouse went upward the first time, he came upon a binary situation of $128+64+32$, or 224, he was going in direction two and he picked direction four. When he came up the second time, he found that he had been going in direction two and the situation value was 224. He got the direction which he had stored (direction five) and is from then on doomed to forever repeat his responses, and he never got the cheese.

Another problem faced by mice was the mousetrap. The basic mouse trap formation is shown below:



Yet another obstacle for mice are dead squares. A cheese placed on a dead square cannot be eaten. Dead squares are indicated by the squares with the Big D's in this diagram



Since the mouse travels corner to corner, it can never change direction unless it is in a corner. There are other boxes in which dead squares can be found, and they are classified in graph four. The first type of box is one that has a side of three.

Since the mouse can pace back and forth, it can cover every square, and the efficiency rating for all such boxes should be about the same. The second class of boxes is those with both sides equal. There are dead squares in all those larger than six by six, and the binary value for all of them should be the same. The third class is boxes similar to a four by five, they can be found by adding a number to four and adding twice that number to five. They travel around the corners and to the two midpoints of the long sides. Their efficiency rating should be the same. There are dead squares in all of them except four by five.

I did not classify any further, although I can think of other boxes with dead squares, these are the basic types which show up most often.

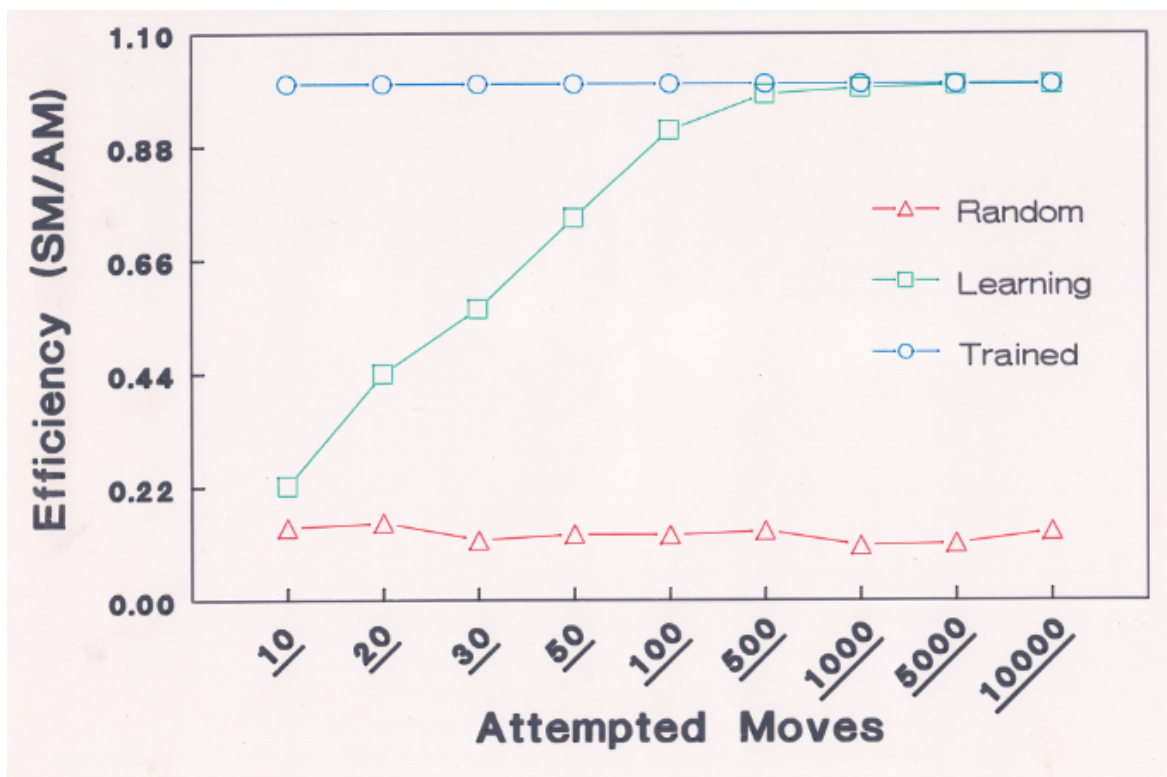
The triple trail priority mouse uses a priority chart, given here:

Trail Leaving	1	2	3
1st (Blank)	B	B	B
2nd (2 Back)	2	3	1
3rd (Last)	3	1	2

When the mouse finds that it has a binary situation value of 255 (trails counted as walls), it disregards the trail which is second on its chart. If it still sees the situation as 255, it disregards the trail which is third on its chart. If the situation is still 255, it switches to the next trail and continues. This mouse was based on the non-sidetracking mouse in its policy of “if you are surrounded by trails, treat them as blank spaces.”

RESULTS

The results of my first experiment are shown in the following graph:

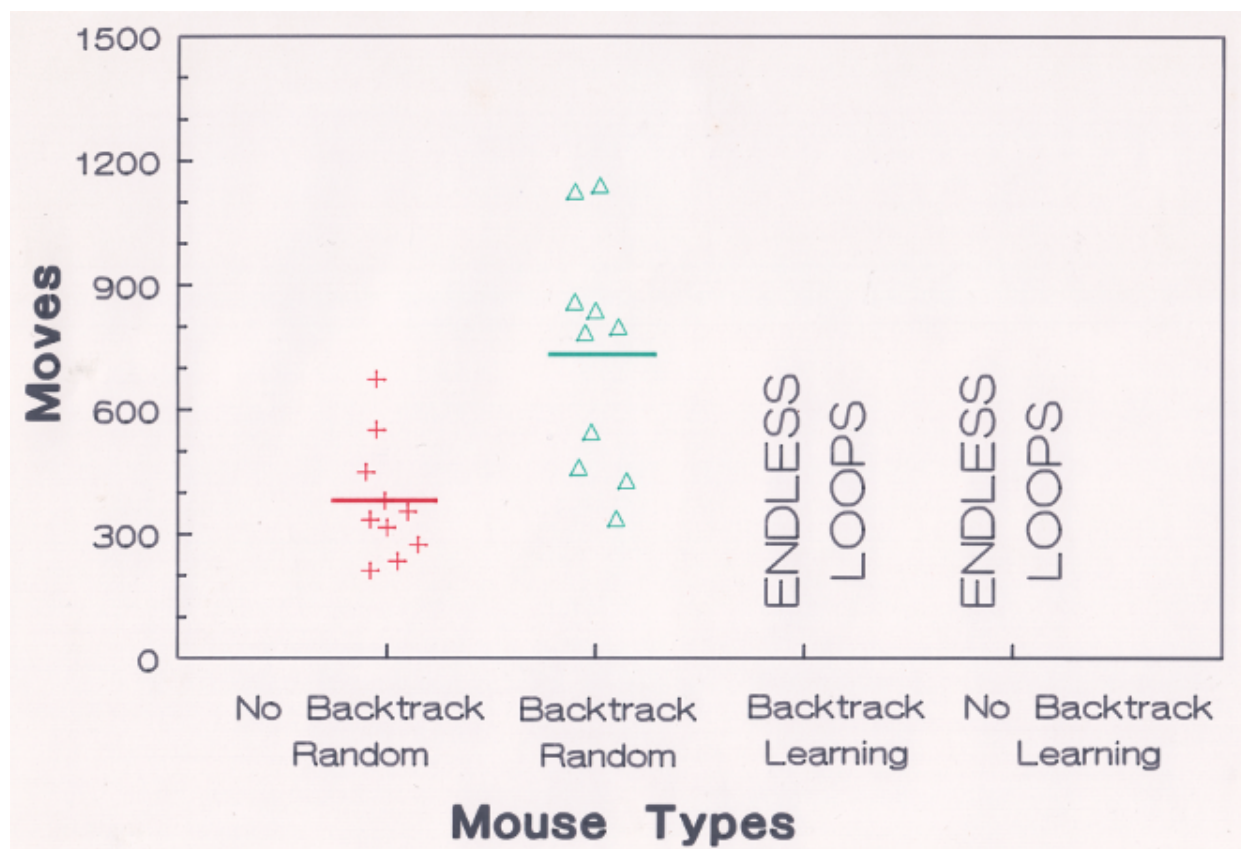


Graph 1: Learning ability of Random, Learning, and Trained Mice

In a three by four box, the mouse can only move back and forth, and for both of the situations there is a one in eight chance that the mouse will pick a valid direction at random. The decimal value is .125, and that was approximately the ratio of successful moves/attempted moves for the random mouse. However, as the graph shows, the ratio for the learning mouse quickly jumped up to almost one. This was because after the learning mouse had already gone through the process of finding the valid directions for the two situations once, it was able to pick a valid move every time. The mice who had already undergone training could call up valid responses every every time, so since their attempted moves were equal to their successful moves, the ratio was one.

When I put the random mice in the obstacle maze, they seemed to be taking too many moves. Most of the moves were wasted when the mice backtracked. However, when I put the learning mice in the maze, they were incapable of solving it at all. They got into what I called “endless loops”, constantly recalling the same responses over and over again even when they don’t work. Endless loops could have hundreds of steps, and if you put a learning mouse in a maze it makes a pattern it cannot break. This was also true of an already trained mouse.

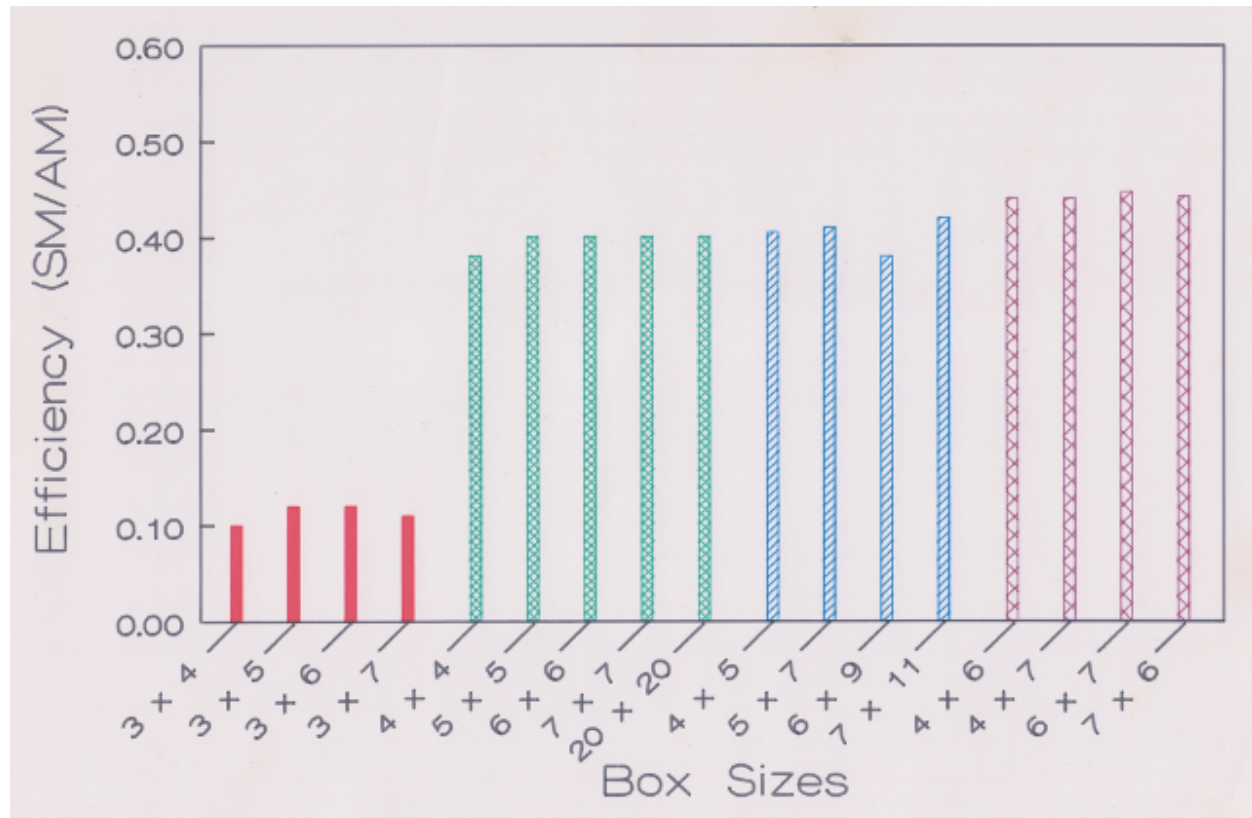
When I added the no-backtracking routine, I saw that the average score for the random mice improved noticeably. The learning mice still were not able to solve the maze, but they managed to do better since the simplest and most common of endless loops had been eliminated. This is demonstrated in the following graph:



Graph 2: The advantages of using a non-backtracking mouse.

When I began running the mice in mazes. I became aware of some of the problems that they faced. Mousetraps were one of these problems. The standard mousetrap is in the shape of a T. Since my mice can only change their direction when they hit a wall, they are forced to pace back and forth endlessly in these situations.

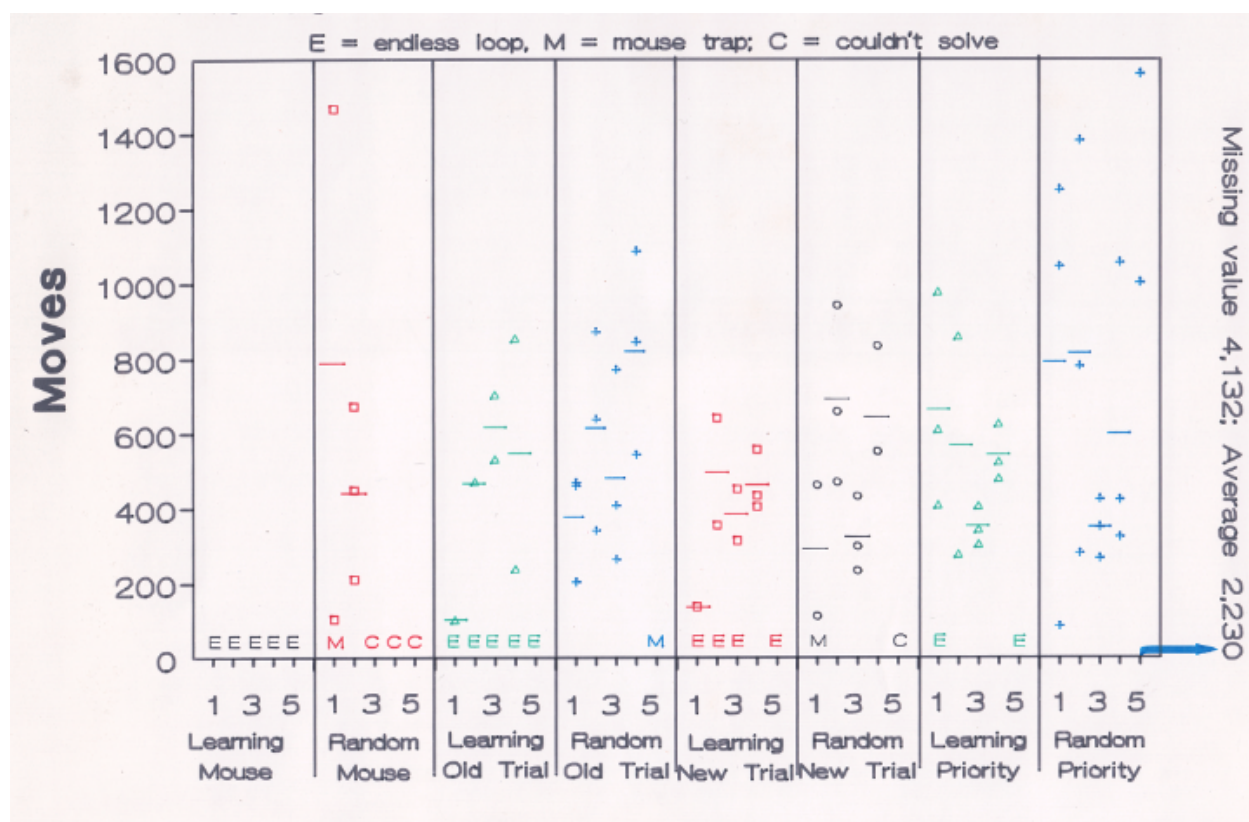
Another problem mice encountered were dead squares. These are certain squares in certain mazes, located in open areas, which the mouse has no way to reach. A cheese placed on one of these squares would be impossible to get. I collected some more data on types of boxes so that I could classify boxes that form dead squares:



Graph 3: The existence of dead squares.

The results showed associations for four major classes of boxes, and the learning ratios for the first three classes should all be the same. If a large box and a small box both have fractional equivalents, then the large box has dead squares in it because it is only using the squares like the ones it used in the small box, leaving gaps or dead squares.

The trail mice, which I programmed to clear all trails in their immediate vicinity when they boxed themselves in, had some problems. When I tried to run them in a maze while allowing backtracking, they took an unreasonable amount of time to get out of dead ends. The learning mice almost always got into endless loops in this situation, and I figured that it is nearly impossible for a random trail mouse to choose the combination of responses that will get it out of a dead end. However, they were able to evade the standard mousetrap.



Graph 4: Comparison of required moves in 5 mazes for the different mice.

When I collected the data in the graph above, I used the no-backtracking routine with the trail mice. This appeared to be working very well until the mice were put in maze 80. They could still evade the standard mousetrap, but they got caught in their own special type of mousetrap and could not solve maze 80 any of the three times they tried. However, the trail mice did well in the other mazes.

The new trail mice had a slightly better average score than the old trail mice. The new learning trail mice only got into endless loops seven times while the old learning trail mice got into endless loops nine times. However, the new trail mice, unlike the old trail mice, could not get out of the standard mousetrap. Both new and old trail mice were able to defeat the problem of dead squares, since dead squares can only be in open areas and trail mice fill up open areas with trails.

The results from running the triple priority mice against the others were encouraging. Even the self-programming triple priority mice were able to solve the mazes almost all of the time. The triple trail priority mice sometimes took more moves than the other types, but the random triple priority mouse could solve every maze. The triple priority mouse could overcome both mousetraps and dead squares.

TRIAL #	LEANING TRIPLE TRIAL PRIORITY	RANDOM TRIPLE TRIAL PRIORITY
1	567	377
2	567	191
3	567	378
4	567	562
5	567	977

Table 1: Demonstration of the learning ability of the triple priority learning mouse versus the random triple priority mouse in Maze #7.

CONCLUSION

My first conclusion is that learning mice learn. This was proven by the results given in graph one.

My second conclusion is that learning mice are faster and more efficient in their responses, because they take less attempts (proven by graph one) and, unlike random mice, they do the same thing in the same maze (proven by table one). This proved a part of my hypothesis. However the part of my hypothesis about learning mice solving the mazes faster than random mice was incorrect. This was due to the learning mice getting into “endless loops.”

My third conclusion is that although the learning mice have the advantage of consistency, they are handicapped by their inability to vary their responses (see graphs two and four). Since even the triple priority mouse sometimes took over 2000 moves to solve a really tough maze, I conclude that a memory system based upon a sequential list of all moves made would be impractical.

Since the random triple priority mice were able to solve every maze they were put into, the triple priority mouse is the best maze solver I’ve created to date. I also conclude that the learning trail mice--and particularly triple priority learning mice--were able to solve mazes (whereas non-trail learning mice can’t) because when they come back to a particular point in the maze, it has been changed by their trail so that the endless loop squares have been filled in so it can’t follow the same path again.

My final conclusion is that picking responses at random will always lead to eventual success if success is possible, and that a system of unchanging responses is not always beneficial.

FURTHER RESEARCH

These experiments opened up the possibility of new experiments. One task would be to create a learning mouse that can vary its responses and still be consistent. A mapping mouse could be created that makes a map relative to its starting position and keeps in memory areas it has not explored yet and tries to choose directions that will get it there. A helpful mouse would be one that would know when a maze is impossible and would quit if it was.

There are already robots that are programmed by humans to perform specific tasks, such as those on assembly lines. There are also remote controlled robots that humans control directly, used to perform tasks flexibly in dangerous situations. The third type of robots are a new field, artificially intelligent robots, that do not need any special programming or direct control. They adapt to the situation.

A robot controlled by a program like mine would have this level of intelligence and would be able to perform special tasks. A deep space satellite when it gets far away from Earth, would encounter new situations and since instructions from Earth would take too long to reach it, it would have to respond on its own. This is the type of instance in which my type of program would be applicable.

BACKGROUND RESEARCH

Artificial intelligence is, in electronics, the means by which computers, robots, and other devices perform tasks which normally require human intelligence. My program was going to use artificial intelligence to perform the usually human task of maze solving.

Although I had written computer programs before, I had to read about certain BASIC commands that I had never used before, such as INT and RND(-TI). I also had to write some Machine Language routines for special tasks that would be too bulky and slow in BASIC. These routines handled such things as transferring data for the maze from memory to the screen. I also had to read about screen memory, which was the basis of my entire program. Screen memory was used to move the mouse on the screen, allow the mouse to check ahead for walls or cheese, and several other vital parts of the program.

An article in POWER PLAY magazine on editing the character set helped me create realistic looking mice and cheeses. This was much better than the alternative; a letter "M" roving around the screen in search of a letter "C."

BIBLIOGRAPHY

Butterfield, Jim. Machine Language for the Commodore 64, 128, and Other Commodore Computers. New York, New York: Prentice Hall Press, 1986.

Commodore 64 User's Handbook. New York: Ballantine Books, 1983.

Golden, Keith. "Editing Characters on the 64," Power Play. Vol. 3 (June/July, 1984), pp. 100-103.