“VOTERS” for Commodore 64 – Category “PUR-80”

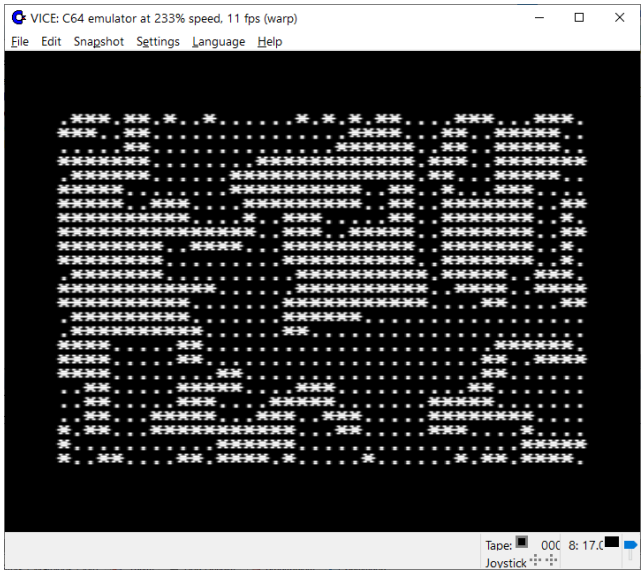
by Eugenio Rapella

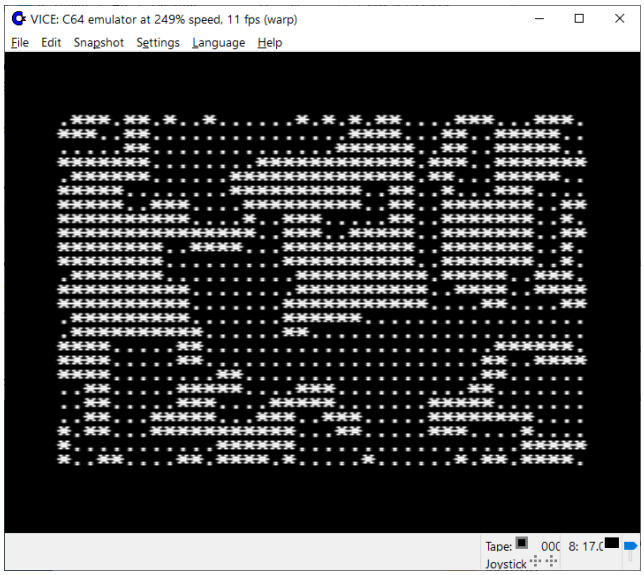
At the beginning of the program, the screen of the Commodore 64 is randomly filled with two symbols, "*" and ".", which represent two states, somehow "in conflict" (they could be conservative/reformist; normal skin cell/altered cell; Bayern Munich supporter/1860 Munich supporter – well, some years ago –… Virus/No Virus). For each of the 1000 locations on the C64 screen (25 rows, 40 columns) the symbol is chosen randomly (with a probability of 50% "*", with a probability of 50% "."). We therefore start from a situation in which the two symbols are well mixed, both as numerical consistency and as geographical disposition.

At this point the main cycle opens: an element of the grid, *alpha*, is chosen at random (those of the first and last row, of the first and last column are excluded) and we count how many asterisks are present among the four "neighbors", ie those located at north, south, east, west. If the number of asterisks is greater than 2, the *alpha* symbol turns into an "asterisk" (regardless of the original content), if it is less than 2 the alpha symbol becomes a "dot" while if it is exactly 2 the content *alpha* remains unchanged. In other words, the neighborhood situation affects *alpha* and causes *alpha* to assume the "position" of the majority of its neighbors (for the *alpha*s the edge locations have been excluded so that each *alpha* has the four "neighbors").

Once the transformation is performed, the program restarts with the random choice of a new *alph*a and a new, possible transformation. What happens in the long run? It is clear that areas where everyone is of the same opinion remain stable because the *alpha* fished in there maintain their state. In the "border" areas there are configurations that, somehow, stabilize.

Here an example of output at the beginning, after 2 minutes, after 10 minutes:





Here is the code:

```
10 w=rnd(-ti):poke 53280,0:poke 53281,0:fort=0to999:h=46:if rnd(0)<.5 then h=42
20 poke 1024+t,h:poke 55296+t,1:next
30 ns=0:x=int(rnd(1)*38)+1:y=int(rnd(1)*23)+1:n=1024+x+40*y
40 if(peek(n-40)=42)then ns=ns+1
50 if(peek(n+40)=42)then ns=ns+1
60 if(peek(n-1)=42)then ns=ns+1
70 if(peek(n+1)=42)then ns=ns+1
80 if ns>2 then poke n,42
90 if ns<2 then poke n,46
100 goto 30
```

Lines 10, 20. Poke 53280,0: black background, Poke 53281,0: black border. The FOR-NEXT cycle takes care of the 1000 screen locations of the C64: location 1024 corresponds to the upper left corner and is filled with a "point" (poke ..., 46) or with an "asterisk" (poke…., 42), with probability ½, when t = 0, then it's the turn of location 1025 when t = 1, and so on. We take advantage of the cycle to enter "white" as the color associated with the location (in the C64, the color map of the screen fonts starts at location 55296).

Line 30. After this preliminary operation, the program begins: *ns* is the variable that contains the number of neighborhood asterisks; initially it is set equal to zero. Then we want to create a random number which will be the number of the column of the location chosen for the game, our *alpha*. The columns are 40 and are numbered from 0 to 39. Since we want to avoid the first and last, we want x to be from 1 to 38. The *rnd(1)* function provides a random, decimal number from 0 to 1: 0 included, 1 excluded: $0 \leq rnd(1) < 1$. Multiplying by 38, you get a decimal number between 0 and 38 with 38 excluded; taking the integer part *int(rnd (1)*38)* we have an integer from 0 to 37; with the final "+ *1*" we finally arrive at the random value from 1 to 38. Same idea for the following instruction that provides a random value (y) from 1 to 23 that allows to determine the *alpha* line excluding the first and the last (the 25 lines are numbered from 0 to 24).
Then the screen memory location corresponding to row x and column y is calculated, this number that is assigned to variable n.

Lines 40-70. The four instructions operate in the same way: each one controls the content of the location of one of the 4 "neighbors", if it is an asterisk the variable *ns* is increased by one unit so that, at the end, *ns* contains the total number of asterisks among the *alfa* neighbors, an integer between 0 and 4.

Lines 80-90. We are at the end, if *ns* exceeds 2, an asterisk is inserted in the *alpha* location *n* (regardless of what was there at the time); if, on the other hand, the number of asterisks of the neighbors is less than 2 (which means that the dots will be 3 or 4), in *n* goes a dot (if *ns* = 2, the content of *n* remains unchanged).

Line 100. Let's go back to line 30 for a new *alpha*.

The program does not have an "end" (it will have to be stopped manually); the C64 screen will show the evolution of the situation. The symbols that appear on the border will never be changed, but will influence their neighbors.