15 REM VARIATIONS IN BASIC

EMULATING THE COMMODORE 64
UNBALANCED
WEAVE
CORNERS
CORNERS AND DIAGONALS
FOUR WALLS
TWO WALLS
POKE
RANDOM SOUNDS

Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

Even small changes to the 10 PRINT code can have a significant impact on the visual output and the pattern produced. The output of 10 PRINT has a unique visual appeal that can be understood in terms of design (a diagonal vs. an orthogonal composition, for instance), and in terms of how it plays against the contextual expectations of the historical period when it emerged (all-text BASIC programs on the one hand and graphical software, particularly videogames, on the other).

To understand more about this, it's possible not only to read the program the way one might go over a poem or other literary text, but also to modify the program and see what happens, as the *Commodore 64 User's Guide* and *RUN* magazine explicitly invite programmers to do. Writing code can be a method of reading it more closely, as was recognized decades ago. The text accompanying the first two printed variants suggested modifying the distribution of characters (in *Commodore 64 User's Guide*) and adding code to cause random color changes (in the magazine *RUN*). This section shows the results of doing the first of these, explores what happens if other PETSCII characters are chosen for display, and finally gives a one-line variation that uses POKE to directly write to screen memory.

As tweaking the program will show, 10 PRINT is a kind of optimal solution that is uniquely elegant in its design space, that of the Commodore 64 BASIC one-line maze generator. Any similar attempt is both less concise (it requires more code) and less expressive (it resembles a maze less or produces a less interesting visual pattern). In fact, the concision of the code and the expressiveness of the image are tightly related. They arise out of a unique set of constraints and interactions, particularly the interaction between the desire to constrain the program code to a single line and the sequence of adjacent characters in the PETSCII table.

EMULATING THE COMMODORE 64

The Commodore 64 was an extremely popular computer; many millions of units were sold and many remain in working condition. It is still possible to cheaply acquire a Commodore 64, hook it to a television, and operate it as users of the 1980s did. When one's goal is to provide a classroom of students with access to the platform, however, or when one wishes to be able to play with and program for the Commodore 64 in many differ-

Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

ent locations on one's own contemporary notebook computer, there is a more practical alternative to finding, setting up, and starting up the classic taupe unit.

This alternative is a Commodore 64 emulator, a software version of the computer that runs on contemporary hardware and functions in the way the original Commodore 64 did. In 1983, a Commodore 64 could be purchased for $600. Today, for those who already have Internet-connected computers, it costs nothing to download and use an emulator. Emulators have been disparaged as inadequate attempts to mimic computers; while they do not capture the material aspects of older computers, they need not be considered as poor substitutes. Instead, an emulator can be usefully conceptualized as an edition of a computer.

When developers produce a program, such as the free software emulator VICE, that operates like a Commodore 64, it can be considered as a software edition of the Commodore 64. It isn't an official or authorized edition—only being a product of Commodore would allow for that. (There are official, authorized emulators for some systems, but VICE and many of the most frequently used emulators are not official.) An emulator like this is an attempt—more or less successful—to produce a system that functions like a Commodore 64. The development of an emulator typically takes a great deal of effort and can be extremely effective, as it is in the case of VICE. Thinking of this as an edition of the system seems to be a useful way to frame emulation, as it allows users to compare editions and usefully understand differences and similarities. Some emulators (like some editions) may be better for teaching, for casual reading or play, or for research and study. Instead of dismissing the emulator as useless because it isn't the original hardware, it makes more sense to consider how it works and what it affords, to look at what sort of edition it is.

The BASIC programs printed in this chapter can be run on a Commodore 64 emulator. The reader is encouraged to download an emulator, run the programs, and imagine how various differences between emulation and the original hardware influence the experience. For instance, the modern PC keyboard does not have the Commodore 64 graphics characters printed on the keys, and mapping the Commodore 64 keys to a modern keyboard layout is not straightforward. Graphically, a composite video monitor or television display attached to a Commodore 64 do not function exactly like a modern LED flat panel; the pixels drawn by an emulator are

Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

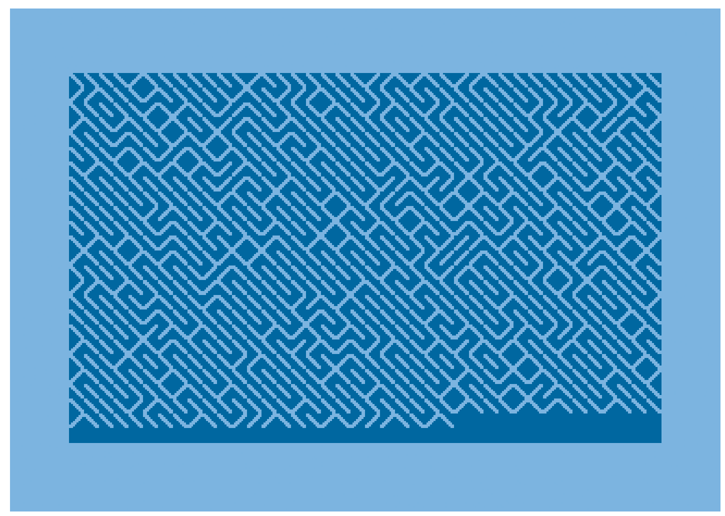

Figure 15.1

```
10 PRINT CHR$(205.25+RND(1)); : GOTO 10
```

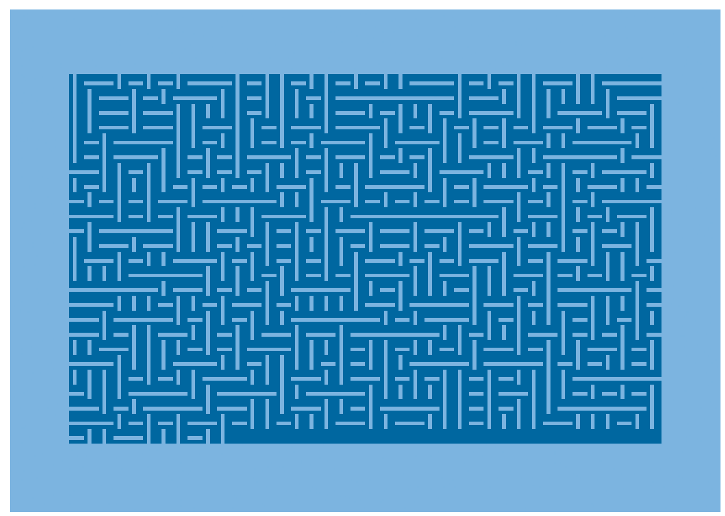

Figure 15.2

```
10 PRINT CHR$(198.5+RND(1)); : GOTO 10
```


Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

overly crisp when compared to those seen on an early display. An emulator lets the user to save the current state of memory, registers, and so on more easily than BASIC programs can be saved to and loaded from disk on the hardware Commodore 64.

UNBALANCED

The *Commodore 64 User's Guide* encourages users to modify its version of 10 PRINT in this way: "If you'd like to experiment with this program, try changing 205.5 by adding or subtracting a couple tenths from it. This will give either character a greater chance of being selected" (1982, 53).

Figure 15.1 shows the effect of changing the ".5" to ".25." As one diagonal predominates, the perceived architecture of the maze tends to long corridors along that direction. More extreme variations, such as going to or beyond 0.95 or below 0.05, present what looks like a regular diagonal pattern with a very few lines going the other way, as if they were occasional defects.

WEAVE

There are no other adjacent characters in the PETSCII data set that, when substituted for the diagonal ╲ and ╱, will result in the construction of a traditional orthogonal maze, one that is aligned to the vertical and horizontal axes of the screen. Using vertical and horizontal bars, for example, results in a disconnected weave (figure 15.2), while solid and empty squares result in a pattern similar to rough static.

Though the result certainly does not suggest a maze as strongly, this "Weave" version of the program is not without visual interest. The output imparts a three-dimensional impression, as if someone had woven bands of material over and under one another.

Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

CORNERS

The Commodore 64 PETSCII character set includes corner characters, such as 204 and 207, which correspond to lower-left and upper-right corner pieces. Randomly selecting either 204 or 207, as is done in this program, produces an image similar to a honeycomb. Diagonal mazes are particularly efficient ones to produce on a Cartesian grid. If a diagonal line is used, four characters can meet at the corners, whereas only two meet along an edge when tiles touch left-to-right or top-to-bottom. This pattern (see figure 15.3) does not offer as many meeting points, but has some of its own interesting visual properties.

CORNERS AND DIAGONALS

A simplification of the program above involves dropping the `INT` function, so that the program chooses at random between other characters in addition to 204 and 207, the two corners; this "Corners and Diagonals" version can also choose the two characters in between. These characters are, of course, 205 and 206, which are the ╲ and ╱ characters that are invoked by `10 PRINT`. The result (see figure 15.4) does not have the clear structure of the `10 PRINT` maze and its pathways run for shorter stretches, appearing to be blocked more frequently. Nevertheless, the pattern that is produced is somewhat compelling in its confusion of elements.

FOUR WALLS

A reasonably intuitive method of constructing a maze-grid is to fill in one edge of each square on a sheet of graph paper. That is, when considering any specific square, fill in the top, right, bottom, or left to form a "wall," then move to the next square and repeat. The four characters in this program correspond to a top-wall, bottom-wall, left-wall or right-wall. Such characters exist in PETSCII in both "thick" and "thin" variants; the ones used in figure 15.5 are the thick ones. Such a process is unfortunately less elegant, as these characters are not (in either variety) placed adjacent to one another in the PETSCII character set—for instance, the ones used here

Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

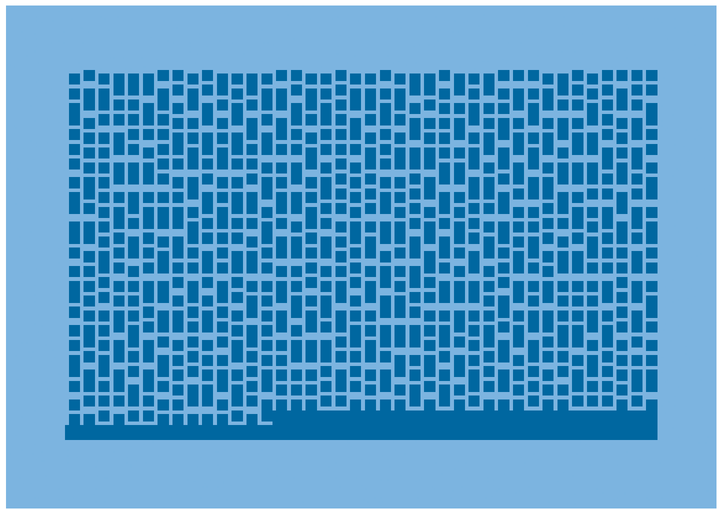

Figure 15.3

```
10 PRINT CHR$(204+(INT(RND(1)+.5)*3)); : GOTO 10
```

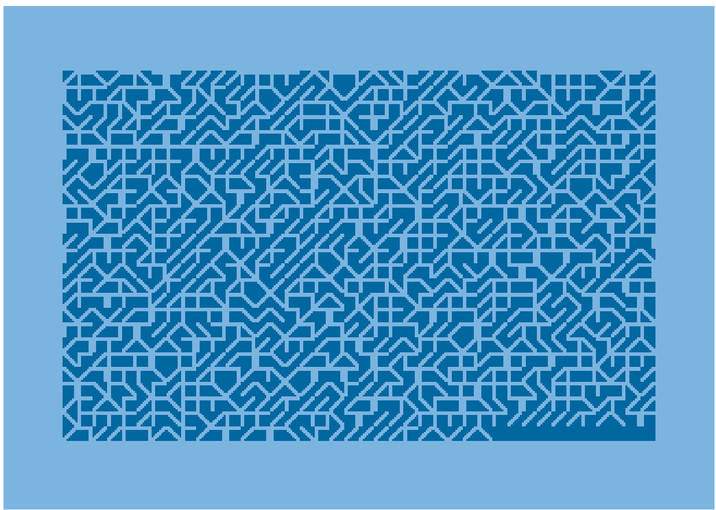

Figure 15.4

```
10 PRINT CHR$(204+(RND(1)+.5)*3); : GOTO 10
```


Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

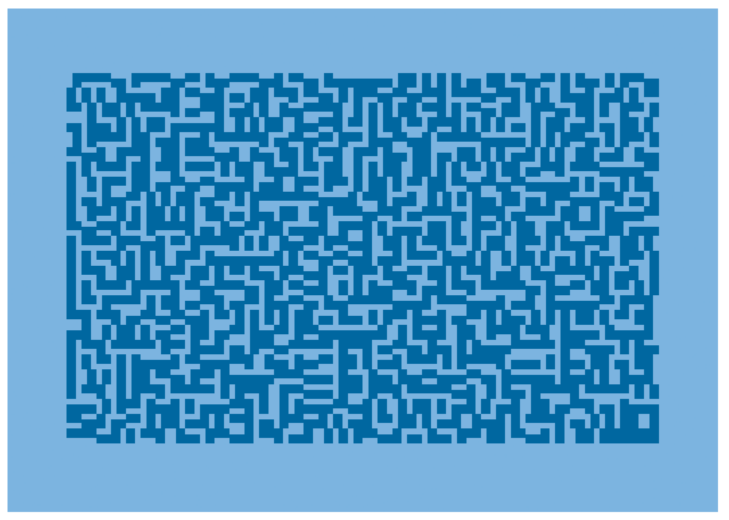

Figure 15.5

```
10 PRINT CHR$(181+(INT(RND(1)+.5)*3)+(INT(RND(1)+.5))); : GOTO 10
```

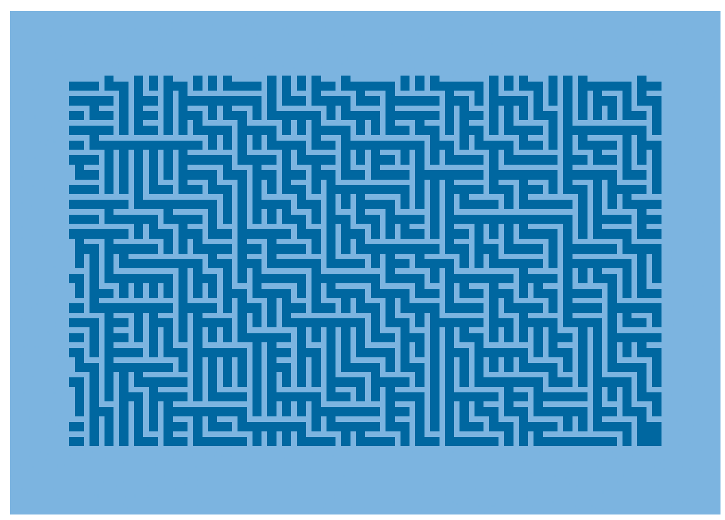

Figure 15.6

```
10 PRINT CHR$(181+(INT(RND(1)+.5)*3)); : GOTO 10
```


Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

are 181, 182, 184, and 185—and so cannot be addressed with a single base value plus an offset, as was done in the previous program.

The image that emerges is indeed mazelike, but this image, like the underlying code, lacks simplicity and elegance. Since top and bottom and left and right lines can be printed up against each other, a variation in the thickness of the walls appears—a noticable but potentially distracting implication of messiness and texture.

TWO WALLS

The selection of characters 181 and 184, a thick left line and thick top line (figure 15.6), provides the best approximation of the classic orthogonal maze that is seen in arcade, console, and computer games. Producing it is still less elegant than selecting between 205 and 206 as PETSCII values. The characters used are not adjacent, so some trick, such as this one involving the use of `INT`, must be used to select one of the two at random. The resulting output is less visually interesting. It is a maze, but is both less formally dynamic (being aligned to the screen) and less contextually unexpected (being typical of familiar game mazes).

POKE

A similar maze pattern can be drawn by directly placing characters in video memory using the `POKE` command, which writes directly to memory—screen memory, in this case, which is mapped to the decimal addresses 1024–2024 (see figure 15.7). The `1024+RND(1)*1000` selects a random number in this range as the first argument to `POKE`, pointing that command at some specific location on the screen. The `77.5+RND(1)` selects ╱ or ╲. It should seem odd that after using 205.5 (and thus the values 205 and 206) to refer to these two characters, this program refers to them using the values 77 and 78. It is, indeed, odd. This difference is due to the PETSCII codes for characters not corresponding to their *screen codes*—each character has a different address for `PRINT`ing and for `POKE`ing into screen memory. This rather esoteric feature of the Commodore 64 is discussed in the final chapter of this book, The Commodore 64.

Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

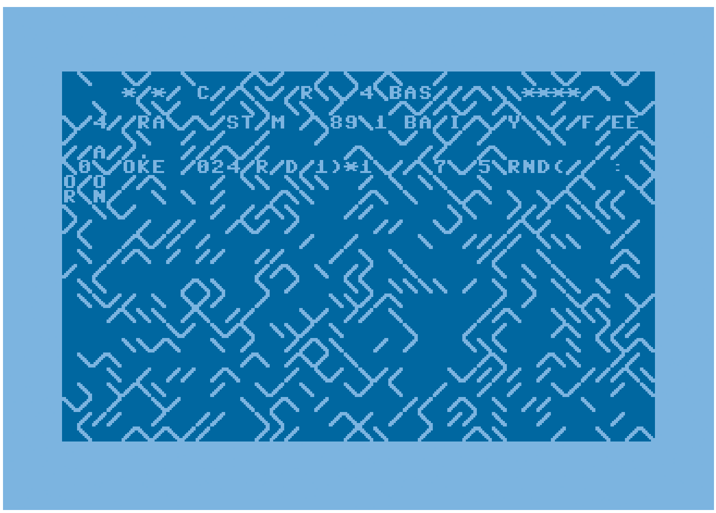

Figure 15.7

`10 POKE 1024+RND(1)*1000,77.5+RND(1) : GOTO 10`

This “POKE” program works by randomly selecting one of the one thousand positions on the screen, randomly selecting the screen code for ╱ or ╲, and placing that code in that memory location. Then, of course, it uses `GOTO 10` to loop back to the beginning and do everything again. While the steady-state output is a full screen of characters changing one at time, the program overwrites the existing contents of the screen slowly, filling in the maze pattern at random.

Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

RANDOM SOUNDS

Finally, consider this considerably more complex program, an audio analogue of 10 PRINT. It plays a sequence of tones chosen from a distribution of two, both of which have the same timbre that approximates that of a piano. The selection is done using the same pseudorandom pattern that 10 PRINT uses, thanks to the invocation of RND(1) in line 30:

```
10 S=54272 : POKE S+24,15 : POKE S+5,190 : POKE S+6,248
20 A(0)=17 : A(1)=37 : A(2)=21 : A(3)=76
30 Q=INT(.5+RND(1)) : POKE S+1,A(Q*2) : POKE S,A(Q*2+1)
40 POKE S+4,17 : FOR T=1 TO 75 : NEXT
50 POKE S+4,16 : FOR T=1 TO 150 : NEXT
60 GOTO 30
```

In Commodore 64 BASIC, one can point into a table of PETSCII characters by simply using 205 and 206 as indices. But there is no similar built-in way to index into a table of notes. After setting up the sound chip in line 10, this program builds such a table using the array A in line 20. Furthermore, the sound chip requires two POKE commands—the ones on line 30—to change the note frequency. Although this is because the chip is extremely accurate in its pitch control, it does make for longer and more involved programs.

This book does not cover arrays (which are not part of the canonical 10 PRINT) in any detail; it would move the discussion quite far afield to explain exactly what is happening in each invocation of POKE in this program. Suffice it to say that POKE is being used to set the sound chip's registers, causing the Commodore 64 to emit musical sounds in a stright-forward way—the standard way one would produce music in BASIC. The invocations of POKE are not simply storing values in memory for later use, nor are they placing values in screen memory, as in the previous example—yet all of this is necessary to move from a randomized generator of block graphics to a randomized generator of tones. This program shows how much easier it is for Commodore 64 BASIC to work on graphic, rather than musical, elements.

Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022

Downloaded from http://direct.mit.edu/books/chapter-pdf/2025471/c000500_9780262305501.pdf by guest on 04 December 2022