My education in programming truly began when I traded my hard-earned money for a copy of “The Creative Atari”. Sure, I knew my around the BASIC programming language, but so did most kids back in the mid-eighties. And like most kids back then, I was far more interested in typing in the games than I was in actually learning something. Luckily for me, this new tome didn’t just contain a few BASIC programs. Each programmer had been thoughtful enough to include a write up with his contribution that not only explained the nuts and bolts of the code, but took us deep into the development process. And they did it in such a way that it stuck with me for the next thirty years.

“Maze Master” was a fun little offering by Fred Brunyate, which would give the player a new maze each time it was run. Having a limited supply of quarters had already prodded me to attempt to program my own games. A program that generated a random maze with only one path from start to finish would give me lot to work with. Even better, was that Fred’s explanation gave me everything I needed to adapt this concept to an entertaining dungeon-crawler game. Years later, while learning C, I’d write a version of the maze builder in this new language. I would repeat that process when learning Python. Clearly, Fred had provided me with a fun little exercise in learning new programming languages. I’d like to pass it on now, so that others may benefit.

The process Fred provided is pretty straight-forward. Start at any point within a 2 dimensional array. Mark the block you are standing in, so you cannot move back into it. Now look at the four squares around you, and see which ones you can move into. Once you have made that list of potential moves, pick one at random. Repeat that process until there is nowhere left for you to move. Once you have boxed yourself into a dead end, simply choose a new position in the array that you have already been in and continue moving. Choosing a location, in which you have already been, insures that all the maze branches are connected together. Repeat this process until all the squares in the array have been tagged.

Python is a great choice for this exercise. If you look at the program listing, you can see that I have divided it up into three main parts: build\_maze, print\_maze, and main. Starting with main, we can see that the first thing that happens is the program asks the user how big the maze should be. With that information, the code then initializes the 2-D array of the requested size to all ‘0’s.

The main loop in maze\_builder first checks to see if there are any cells left in the array that have not been moved into. If there are still some available, it makes a list of possible moves (left, right, up, and down) based on what is around the cell it is in. The program then makes a random choice based on the list is has compiled. Now things begin to get a little confusing. The same array that keeps tracks of whether or not a cell has been moved into, also keeps track of how the walls on each cell are set up. This program will only concern itself with the lower wall and the right wall of each cell, with the upper wall being handled by the cell above and the left wall being defined by the cell to the left. A value of 4 means both the lower and the right wall of a cell are in place. A value of 1, means that both the lower wall and right wall are absent. Values of 2 and 3 correspond to only the right wall or lower wall respectively. Based on with random direction the program has decided move in, it will update both the cell is it currently in and the cell it moves into.

The last part of the program is print\_asci\_maze. All this does is print out the completed maze based on the array it is fed. Back in my Atari days, I had solid block graphics that made excellent walls and intersections. As I am still feeling out Python’s capabilities, I have decided to just use the characters +, -, and | for now.

Hopefully you have enjoyed this exercise as much as I have. But more importantly I hope that any budding programmers will come away with an understanding of the application of a simple concept can be applied to help you learn new languages. Keep on coding!