English 1076

17 February 2014

Learning Computer Science:

A Secret Wall of Zeros and Ones

When I first applied to Mizzou, I was a biology major. I thought I wanted to work in a genetics research lab because my DNA science class was lab intensive, and I was good at it. But one day my Dad told me,

> If you want to make any respectable amount of money with a biology major, the only option is to become a medical physician. Do you want to be a doctor?

I replied without hesitation,

> No.

He thought a moment, then said,

> You should do computer science; you're good with computers.

I quickly yelled,

> But I don't know anything about programming!

He smiled and said,

> You'll learn.

The first day of classes, I was really scared. I was terrified that I wouldn't be any good at computer science and that I was going to fail. But to my surprise, I loved it. It was a new world; a hidden world all around us that nobody ever thinks about. The idea of

the first person that invented the concept of computer programming baffles me. Using zeros and ones to tell a computer to do simple things like turn on or off seems impossible. To this day, I still don't fully understand that. Every time I go to computer science class, I learn something new that makes everything seem more complicated. But then I realize that the old programming functions and techniques that once seemed complicated became easy.

I notice the same thing when I play video games; it seems really difficult in the beginning, but as you progress through the game, the old levels are no longer a challenge, and certain moves and tactics become reflexes that are attuned like real life reflexes such as catching a ball. Although programming is a life skill and playing video games is a form of entertainment, they are both comprised of the same learning aspects.

I have never had to think like this before. The code is logical, yet so cryptic. During labs my friends and I look at each other thinking the same thing; will this function work, does this line make sense? It looks even weirder to people who don't program. After a lab one day, I showed my friend Giselle my program:

I told her,

> It's a grade-averaging program. It asks for two mid-term exam grades and the final exam grade, and then it tells the user what grade they get in the class.

She stared at it and said,

> That's really cool!

But then I showed her the code. Her eyes grew big and she said,

> Wow. It took over two hundred and fifty lines.

That program wasn't very challenging but at the time it was a bit frustrating. It reminded me of a game I played when I was very young called *Snood*. This game involves shooting these weird balls with faces on them to make groups of three or more of the same color to make them disappear before they get too close. When I first started playing I couldn't even get past the first level. I couldn't get rid of the faces fast enough before they were pushed into the bottom of the screen. But then I started noticing that when I bounced the ball around a bunch of the other balls to match a group near the top, it would make all the balls connected to it below fall. Once I learned this technique, I advanced much farther in the game and I never lost on the first level again. The same learning principle applies to coding: I practiced writing the true/false statements I used in my lab enough to the point that I no longer had to contemplate over what outcome would occur. I stopped making careless syntax errors too.

Every web page you visit, every movie you watch, every time you use an A.T.M.; anything with a screen is secretly a giant wall of zeros and ones represented by numbers, variables, and algorithms. Somebody had to first think up the idea for that program, develop the algorithms for the different functions the program has to perform, and then type it all up in the computer language of their choice. People had to invent each language too. They had to turn binary (zeros and ones) into English words that programmers could actually use to develop useful applications. And when I think about how binary works my brain dies a little. I don't understand how the computer knows that the zeros and ones represent certain numbers and characters or how someone created a machine to interpret all of it. My professor tells us that we don't really need to understand how machine code works because people already spent a generation working on it. The

only way to advance technology is to use the tools we are given to create new things. After all, that's how most of society works. A man who wants to hang a picture on his wall shouldn't have to melt iron into the shape of a hammer and sharp points for nails. He only has to purchase them at a hardware store. In Malcolm Gladwell's book *Outliers: The Story of Success,* he discusses "the ten thousand hour rule," which states that to become an expert at a particular skill, one must practice for ten thousand hours: "In fact, researchers have settled on what they believe is the magic number for true expertise: ten thousand hours" (40). Nobody has enough time in his or her life to become an expert at everything, and neither do I. The purpose of my computer science education is mostly because I want to be able to create a video game. But before I do that I need to master programming. And to master programming, I need to apply the same learning aspects I experience when learning how to play a video game.

The "ten thousand hour rule" can be applied to my favorite gaming franchise of all time, *Halo*, because it takes so long to master. In these *Halo* games, the player takes control of a "Spartan," a super soldier with genetic modifications in a suit of power armor. Players compete against each other by trying to kill one another. There is a lot of strategy involved when playing *Halo*. Contrary to what most people think of first-person shooters, shooting someone first doesn't necessarily mean you'll win, especially when it comes to *Halo*. People don't realize that while they are playing, they are constantly making choices and planning unique ways to outwit their enemy. Experienced players don't just run straight at somebody when they see them. They use their environment and they put themselves in situations that will give them significant advantages and allow effective use of specific weapons. Someone who has a shotgun will often trick somebody

to follow him or her into a room by shooting with a regular rifle and then switching to the shotgun to kill the enemy player. Players learn to time grenades to explode at an adversary's feet when they come up a lift. They are constantly internalizing these strategies that they subconsciously acquire. Of course strategy can only take you so far. Sometimes players need to improvise. But these tactics are still deadly instincts. A large part of why I enjoy programming is because I get a sensation that is similar to what I feel when I play *Halo*. I can program learned functions and techniques without thinking because they have become hardwired into my brain. There is no hesitation. Only what the next move is; what the next line of code will be. And when I do come across a problem that is completely new to me, I have to figure it out, and once I do, I can keep practicing. Eventually, it will become an instinctive thought process in my brain, and I can use it automatically to aid in solving future programming dilemmas.

Halo isn't the only game that requires learning to progress. When I first played a video game called *Catherine*, a block puzzle game, I didn't really know what I was doing. I would just move blocks around randomly until I solved the puzzle. But the puzzles became more complicated, and I was forced to learn new strategies and techniques. Whenever I used these new techniques, I would have to stop climbing and look at the tower of blocks. But eventually after performing them over and over again, they became automatic. I didn't have to think about them anymore. When I am faced with a problem in computer science, such as a specific coding algorithm, it seems impossible at first too. But once I figure it out and use it more often, it slowly becomes a function in my brain. I don't have to stop and think how to do it. I've practiced it enough to the point that I can code in that way subconsciously. A few days ago I wrote a program

in my computer science lab involving linked lists. A linked list is a data structure of a group of nodes that "link" to each other. I think of it as blocks of memory, containing data, that are all linked together like a chain. In this lab, the program had to read in a letter and a number from a file. Based on the letter, the program would perform a certain operation and then read in another letter and number to perform another operation. For instance, if the letter was 'i', the program would insert the number into the linked list. If the number were even, the program would insert it at the beginning of the list, but if the number were odd the program would insert it at the end of the list. But I hadn't used linked lists like this before. On my first attempt, my program would overwrite the number at the end of the list when inserting an odd number. It wasn't supposed to do that, it was supposed to place the number after the end number, making it the new end. I took over an hour to fix this. Like when trying to use a new technique in *Catherine*, I had to stop and stare at my code until I found out what was wrong. After studying my code, I understood the logic and was able to fix my program. Linked lists still confuse me a little but in time I think using them will become instinctive. After all, when I first learned how to read data, such as student grades, from a text file for a program I had trouble, but now it's a piece of cake. The same is true for each tactic I learned in *Catherine*. I'll become a better programmer by learning techniques one at a time and mastering each before moving on to the next one.

Coming to Mizzou to learn computer science has had a strange effect on me. I never thought I would apply processes I learned from playing video games, an activity my parents think is a waste of time, to learning how to program. It's all very ironic. Computer science has changed the way feel about academics; I didn't think I could ever

enjoy learning so much. It's a shame people bash video games all the time. I'm not saying that video games make people smarter. Video games are mostly for enjoyment, but they actually stimulate your brain in a positive way through subconscious learning and thought processes. There are lots of activities like video games that can exercise your brain, and people need to learn to use them to their advantage.