Raspberry Pi: Exploring Computing's Past, Present and Future

William Van Loo
Eastern Michigan University
EDMT632: Fall 2013

Raspberry Pi: Exploring Computing's Past, Present and Future

Many children of the 1980s grew up in one type of computer household or the other. For those in the first type of household, that meant an Atari 2600 game console (or ColecoVision, or other, similar systems). A smaller percentage, however, grew up in households with actual home computers - machines that could be programmed in a variety of languages, and had diverse and often baffling peripherals such as tape-based storage, CRT monitors, and floppy drives (I now count myself lucky to have grown in the second type of house, although convincing my 11-year old self that our home computer was better than the opportunity to play River Raid for the Atari 2600 might have been a hard sell).

The difference between these two types of computing machines was in the intended usage. The Atari 2600, for all its groundbreaking and addictive qualities, was primarily a machine designed to be used by consumers. The competing home computers of the time, such as the Commodore 64, TRS-80, and related machines, were designed to be programmed by their owners, or at least to run software programmed by others, often amateurs.

It's interesting to reflect on those times now. We have iPads with computing power far superior to the supercomputers of the 1980s, but the idea of programming a computer still seems as far-fetched now for most people as it did in, say, 1986. While devices such as the iPad bring with them ease of use, flexibility and user-centered design, their primary function is to be *used* by consumers (though having said that, there are many, many wonderful ways to use devices like the iPad as content-creation devices). What there is a distinct lack of, however, is the easy ability to program these devices. For example, it costs significant time and money to even get started programming for iOS devices.

On the other end of the field from the iPad is a humble new computing device, designed by four researchers at University of Cambridge's Computer Laboratory, called the Raspberry Pi. Their vision, in their own words, states "We want to see cheap, accessible, programmable computers everywhere" (Raspberry Pi Foundation, 2013, "About us: Raspberry Pi" para. 6). Their computer costs around $35, and is designed for educational purposes. Most importantly for purposes of this investigation, the Raspberry Pi is designed to be *programmed*, not simply used.

Programming in Education: How and When?

While describing Scratch, the graphical programming environment for children produced by MIT's Media Lab, Mitchel Resnick writes, "As we see it, digital fluency requires not just the ability to chat, browse, and interact but also the ability to design, create, and invent with new media...To do so, you need to learn some type of programming." (Resnick, et al, 2009, p. 62). Unfortunately, there are no standards or benchmarks specifically aimed at encouraging programming in the Michigan Educational Technology Standards for Students (MET-S), the Common Core Standards for Mathematical Practice, or the Career And Employability Skills Content Standards & Benchmarks. The International Technology and Engineering Educators Association's Standards for Technological Literacy (ITEEA STL) makes a single, passing mention ("Models can take many forms, such as physical replicas of artifacts, computer programs, conceptual and mathematical modeling, and simulated products."). While the International Society for Technology in Education (ISTE)'s National Educational Technology Standards for Computer Science Educators has a specific set of standards for the teaching of programming and computer science, it is totally separate from the more broadly used National Educational Technology Standards for Students (NETS-S).

With the value placed on standards-aligned curriculum and high-stakes testing, there is a very real concern that the lack of any standards related to programming will cause students not to attempt or be formally exposed to programming until high school or college at the earliest. This would push it past the point where many students will be interested or able to take a programming course, as the demands of high school limit the number of electives most high schoolers can take.

Why does this matter? To quote from Seymour Papert (2000), one of the founding voices of modern educational technology, "In Mindstorms I made the claim that children can learn to program and that learning to program can affect the way they learn everything else. I had in mind something like the process of re-empowerment of probability: the ability to program would allow a student to learn and use powerful forms of probabilistic ideas." (p. 727). This ability to make meaning and use the kind of logical thinking that programming requires is exactly what students need in the 21st century.

Past and Future of Programming in Education

One microcosm that demonstrates the philosophical shifts that have occurred in computing tools over the past 25 years can be seen in the United Kingdom. "The BBC Micro and the Acorn Atom - which in many ways was very similar - inspired a generation....There were many other machines around, all capable of being programmed, like the Commodore Pet, Sinclair ZX80 and ZX81, Apple 2, and then in the following years Spectrum, Amstrad CPC, Commodore 64, even curios like the Oric Atmos, Tatung Einstein, and MSX. But it was the BBC Micro that had the impact - mainly because of its spread throughout UK schools." (Braben, 2011, para 5).

Contrast that world of possibility with the current ICT curriculum of the UK, which focuses primarily on the use of office-oriented computer applications. Thankfully, the British government's education officials have recognized the issue with this. British Education Secretary Michael Gove, in announcing a radical overhaul of the ICT curriculum, wrote, "Imagine the dramatic change which could be possible in just a few years, once we remove the roadblock of the existing ICT curriculum. Instead of children bored out of their minds being taught how to use Word and Excel by bored teachers, we could have 11-year-olds able to write simple 2D computer animations using an MIT tool called Scratch. By 16, they could have an understanding of formal logic previously covered only in University courses and be writing their own Apps for smartphones" ("'Harmful' ICT curriculum", 2012).

The Raspberry Pi and Minecraft Pi Edition: A look at the future

Throughout my work with elementary and middle school students over the past seven years, I have had the opportunity to teach with a number of the type of open-ended programming tools that Gove describes, including Scratch, the Lego Mindstorms robotics packages, and Processing, a visual programming language based on Java. This summer, I took on a new challenge by running a week-long summer camp where campers would use the Raspberry Pi computer and a special version of Minecraft to learn programming, networking and computer maintenance basic in a collaborative environment. Minecraft is an incredibly popular 3D game in which players can build their own worlds and structures using simple blocks of diverse materials; the analogy to building with Lego bricks is often made here. The

special version of Minecraft used in our camp, Minecraft Pi Edition, was designed specially for the Raspberry Pi and allows players to manipulate the game world through programming instead of strictly by clicking the mouse to place blocks and move. You can see an example of this in the photo below (Figure 1).

Figure 1: A camper's creation built in Minecraft Pi Edition

On the first day of camp, I gave the campers the opportunity to decide how to arrange the classroom. After deciding to assemble their desks into four pods of four desks each, campers started getting out the things they'd need to set up their Pi systems, including monitors, keyboards, power strips, and so on. This is a far different experience from being handed a laptop or tablet where all the components are neatly bundled together and ready for use - the students were forced to understand the relationships among the various parts of the computer system. There was tremendous excitement when each camper had assembled all the components of the system and finally got to start it up.

Once the campers had their Raspberry Pi systems running, we quickly jumped into playing the Minecraft Pi Edition game, and did this until lunchtime. After lunch, I introduced Python, the programming language we'd be using for the rest of the week. We wrote some basic "Hello World" programs, and campers got started learning how to use loops in Python to make things happen over and over.

The second day of camp, the campers started learning Python code that was designed specifically for the Minecraft Pi Edition, building on Monday's first Python programs. There was great excitement when campers saw the power that a few lines of Python code could wield in the Minecraft world - instead of clicking 1000 times to make a 10x10x10 cube, they could write four or five lines of code and have that same cube generated for them.

During the final three days of the camp, we saw our work proceed in a similar fashion: I would demonstrate a programming technique on the projection system, and give campers time to find out how to use it on their own. A number of particularly interesting phenomenon were observed during this time. I wish to make special note of two behaviors: pseudocode, and pair programming.

Pseudocode is the practice of writing out a desired behavior in natural language before worrying about writing code that the computer will understand. For example, one might write the following as pseudocode:

- Get the player's current position
- Calculate the position of the block 5 spaces away from the player
- Build a tower 10 blocks high starting at this new position.
- Move the player on top of this new tower

After working out the basic logic to be followed, the programmer can then translate this into actual working code. Pseudocode proved to be a wonderful technique for talking through ideas with students; the mental process of talking through what it was they wanted to accomplish forced them to carefully consider the desired result without getting bogged down in syntax.

Another technique that worked well was the use of collaboration, specifically pair programming. This is a technique I first encountered during my time as a programmer before switching careers to education, and the fact that my campers started using it spontaneously was a huge thrill to see (Figure 2). As Anderson describes it, "…Pair programming isn't one programmer looking over another's shoulder as she bangs in code. Pair programming is two programmers, side by side, working together to write the program…when they're finished, two people understand everything, instead of understanding just half of everything." (Anderson, et al, 2000).

Figure 2: Two campers use pair programming to solve a problem

It would be wrong to imply that there were no issues faced during this week of programming and exploration. One problem is that of gender imbalance. Out of 16 students who enrolled, 14 were boys, and two were girls. One of the girls dropped out of the camp after the first two days, and regrettably, the other girl was sick for four out of the five days of camp and was unable to attend. This is a common problem in high-tech settings, and one for which there seem to be no easy answers except to approach the problem with much more intentionality in terms of attracting girls to these types of opportunities.

Conclusions

The experience of running the Minecraft Pi camp reinforced to me the importance of building in opportunities for our students to explore programming and for computing hardware that supports it. If my summer camp program had focused on, say, making electronic music, I could have used any number of devices - iPads, laptops, hardware synthesizers, and so on. In order to give students the experience of programming, however, the hardware used must be able to be programmed, and for this use the Raspberry Pi shines. Its ability to add capabilities by being programmed meant that campers could explore their

ideas freely, whether that meant plugging in custom USB controllers made from Play-doh or networking many Pi computers together over Ethernet to collaborate on world-building.

During the writing of this paper, an email arrived from a former camper, telling of the way he was planning on using his Raspberry Pi as a Web development platform over the upcoming weekend. While we as educational technologists continue to look for devices that are easy to use, we should also look for devices like the Raspberry Pi that empower our students to do the kind of "idea work" that Papert's pedagogy of technology-supported constructivism is based upon.

References

About us: Raspberry pi. (n.d.). Retrieved from http://www.raspberrypi.org/about

'Harmful' ICT curriculum set to be dropped to make way for rigorous computer science. (2012, January 11). Retrieved from https://www.gov.uk/government/news/harmful-ict-curriculum-set-to-be-dropped-to-make-way-for-rigorous-computer-science

Braben, D. (2011, December 1). The bbc microcomputer and me, 30 years down the line. Retrieved from http://www.bbc.co.uk/news/technology-15969065

Jeffries, R., Anderson, A., & Hendrickson, C. (2000).Extreme programming installed. (p. 88). Boston, MI: Addison-Wesley Longman Publishing Co.

Papert, S. (2000). What's the big idea? toward a pedagogy of idea power. IBM SYSTEMS JOURNAL, 39(3&4), 720-729.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E.,Brennan, K., Millner, A.,Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y. (2009). Scratch: Programming for all. Communications of the ACM, 52(11), 62.

About The Author

Bill Van Loo is a teacher, musician and photographer. He is currently employed as the Technology Specialist for Honey Creek Community School in Ann Arbor, Michigan. He teaches technology and engineering to students in grades 2-8, as well as planning and supporting the school's technology assets and infrastructure. As a teacher, he values cross-curricular learning experiences, project-based learning, and use of the design process to help students learn how to solve problems. He supports multiple computing platforms used by staff and students, and sets technology policy and direction for the school. Bill presented at MACUL in 2010 and was published by the MACUL journal in the same year. In addition, he has presented multiple times at the International Technology and Engineering Educators Association conference as well as authoring articles for the ITEEA journal "Children's Technology and Engineering".

Bill is currently in the planning stages for the next iteration of the Raspberry Pi-based summer camp, with plans to expand to include an advanced session. Follow his adventures on Twitter @billvanlooteach.