Graphical Qualities of Educational Technology
Using Drag-and-Drop and Text-Based Programs for Introductory Computer Science

Betsy DiSalvo
Georgia Institute of Technology

The Glitch Game Testers, a Georgia Tech and Morehouse College collaboration, trained and employed African-American male high school students as game testers. We aimed to leverage the passion that many young urban African-American males have for video games into a greater agency with technology. Whereas the participants spent most of each day testing prereleased digital games, they spent an hour a day in computer science (CS) workshops.

In our exploratory work, we had observed that young African-American men had a strong interest in digital games but weren’t leveraging this into an interest in CS. We also found that these men valued learning about technology through what they considered real work practices. Because we sought to leverage their interest in games and use authentic practices, we were torn about which language to use to introduce programming. A visual-programming language offered a low barrier to entry and compelling visuals similar to digital games. In contrast, a text-based language offered an authentic experience of programming like “real” computer scientists and game developers.

In the program’s first year, to better inform our curriculum design, Kenneth Perry, chair of Morehouse’s Department of Computer Science, taught the first four weeks of workshops using Alice, a visual-programming language. This instruction employed project-based activities that emphasized software development (see Figure 1). In the final four weeks, Mark Guzdial, a professor at Georgia Tech’s School of Interactive Computing, developed and taught a curriculum using text-based Jython, a version of Python (see Figure 2). The related programming activities let students integrate media they created in Jython into their Alice projects.

Glitch’s informal learning environment provided a freedom and agency for learning that let the participants choose when and how to focus their learning on the basis of their interests. (For more on Glitch, see the related sidebar.) It also let us observe their choices and conduct inquiries about their interaction with both programming languages. Using observations, pre-interviews and post-interviews, design-based research methods, and survey tools, we looked at how the participants responded and learned using the two programming environments. Although participants often preferred one language, that language wasn’t necessarily the one from which they learned the most. Also, we found that when selecting the graphical qualities of educational technology, educators must take into account the interplay between access to learning concepts and participants’ value of authenticity and ownership.

Programming Language Preferences
In the interviews, we asked the participants what they knew about computing, what they learned in the CS workshops, and what they were proudest of. Most participants volunteered that they preferred one of the two programming languages. Three participants preferred Alice, six preferred Python, and three indicated no preference. A pattern emerged between the preferred language and the participants’ career intentions (see Figure 3).

Preference for Alice
Those participants who wanted to pursue media and design careers might have preferred Alice for its visual interface and low entry barrier. Max (all...
participant names are pseudonyms) told us he found Alice to be an easy way to do media and design projects:

I was amazed by Alice. I didn’t know it was going to be so easy. I thought I was going to have to type a million codes just to make [the characters] walk, but you can just drag and make your own methods and your own properties, change the colors, and you can drag pictures from Jython into the Alice world.

When we asked those participants why Alice was easier or more enjoyable, they mentioned that its visual nature let them understand what was happening in the program. They told us that debugging and knowing “when it worked” was easier. For example, Jacob said that

Jython is not exactly as clear just because you can’t see it; you have to think to yourself about what is actually happening inside the computer. ... I don’t like to read code that much; I just like to write the code and look that it came out correctly. With Alice, you can really see that it came out correctly.

Preference for Jython

The participants who preferred Jython mentioned that it was an authentic, challenging programming experience. Carver noted that Alice was too easy for people of his age:

I mean, to make a character move [in Alice], I feel like a six-year-old could do that. ... With JES, that takes time, and you have to put a lot of work into it and type all the stuff out.

Everett made a distinction between Alice being time-consuming and Jython being challenging:

Alice is just really time-consuming. If you mess up, sometimes you have to go all the way back and change what you did. ... So, like in Jython, when you have the code on the top half, you have to type in, like increase max, or increase blue, or something. And then you have to load [the] program, click explore, picture, all of that. It is pretty confusing, but I like being confused.

The challenge of textual programming also seemed to motivate learning. When we asked Oscar what accomplishment he was proudest of, he talked about how difficult programming in Jython was at first, and his pride in accomplishment in working with the textual programming:

When I first started [Jython] it was ridiculously hard. I didn’t understand any of it; I felt like I was at school ... but I finally got it.

Others felt that Jython was more authentic because it was closer to “real” coding. We suspect the four participants who expressed a strong interest in programming careers preferred Jython on the basis of the perception that textual coding was more authentic programming. Elijah specifically addressed how it was more like game development to him:

Figure 1. Alice is a drag-and-drop programming language that lets users modify and link lines of prewritten code to control animated characters and objects.

Figure 2. The Jython programming projects used the JES (Jython Environment for Students) integrated development environment. JES is designed to look like DrJava or DrScheme, in which the editor window sits on top of an interpreter pane.
It [Jython] made me feel like I am closer to making a game, like I’m closer to the way that EA makes a game. ... It made me feel like I was close to my dream, like closer to what I want to do in life.

Learning with Both Languages
Student preference is important. However, we also wanted to keep in mind our objective for participants to learn basic computing concepts. As mentioned above, although participants might enjoy one interface more than the other, they might learn more, or different, concepts from the interface they don’t prefer.

In interviews and observations, we found that Alice was more likely to help the participants understand broader CS concepts such as object-oriented programming. Survey findings also supported this (see the related sidebar). Participants expressed a high degree of confidence in understanding Alice’s basic operations and explaining what programming is. However, they had less confidence in explaining more complex ideas such as functions, algorithms, and stepwise refinement.

In contrast, Jython taught in a computational-media context seemed to serve as a vehicle for understanding computing concepts in application rather than abstraction. For example, the participants weren’t confident they could explain an algorithm as taught through Alice but expressed confidence that they could describe more concrete examples of algorithms used with Jython.

When asked how they would apply what they learned, participants gave direct examples of how Jython’s computational-media approach gave them a better understanding of media concepts. Jacob was interested in music and music technology and was well versed in software designed for mixing music. Despite his reported preference for Alice, Jython seemed to provide him with more insights into how sounds are represented in computational language:

He [Mark Guzdial] introduced a lot of new concepts that I really liked; you remember the graph he showed when he increased the frequency. ... I would like to use it one day.

Daniel, who also preferred Alice, talked directly about Jython assisting him with computing concepts concerning visual media, rather than Alice:

I thought at first, a designer has to go pixel by pixel to design a program, but then we used Jython, and I realized it was easier than I thought it would be.

Many of the participants who preferred working with Jython reported learning gains for bigger concepts with Alice. For example, although Isaiah preferred Jython, his use of Alice let him gain insights on higher-level thinking about CS:

I like the top-down design [learned with Alice projects]. We are able to break down the bigger problems into smaller problems and then even smaller problems, so it’s a simpler way to file through. I think you could probably apply top-down design to anything in life.

Glitch
In 2009, Glitch launched with 12 students (ages 16 to 18) working Monday through Friday for approximately 35 hours per week. From this first summer through the three-year program, participants demonstrated a strong connection between Glitch and an interest in CS. More than 65 percent of the participants have gone on to major in computing-related fields.

Reference
Designing GUIs for Learning

When designing for young people, developers often think that highly visual, easy interaction is the way to capture their interest. However, in exploring the Alice interface, we found that the compelling visuals had both advantages and disadvantages. The visual nature provided a lower entry barrier, the ability to visually map complex concepts such as object-oriented programming, and an opportunity to see a project from concept to completion—providing a wider view of software development. Yet, the visual nature made the process appear toy-like and inauthentic. We also saw indications that transferring the knowledge gains from Alice to practice might be difficult.

This study doesn’t suggest eliminating the use of highly visual educational technology such as Alice. However, it does suggest we take into account the intended users and how they’ll translate abstract concepts into concrete skills. For Glitch, we continued to use both tools in the program’s first year because they afforded different kinds of engagement and learning. The young men who continued on for a second year in Glitch concluded that Jython was also not a “real” tool used frequently in development and requested we teach them Java. So we did that by preparing them to take the advanced-placement computer science test in their second year in the program.

In this way, Jython was also toy-like and inauthentic to our participants. But they discovered that only after entering into a community of practice of programming, talking with Morehouse and Georgia Tech CS undergrads, seeking help on the Internet, and looking at other code. In these ways, Jython was a better bridge between the educational context and legitimate participation in computing.

Visually appealing, easy-to-use interfaces and graphics are often our default objectives in designing new technologies. And strong visual interfaces enable modeling of complex ideas and easy entry points to new learning opportunities. However, when designing educational technology, we should also consider authenticity, transfer, and concrete examples. This might result in dirty or clunky interfaces that aren’t pretty but facilitate learning.

Acknowledgments

The Glitch Game Testers project was my dissertation work, conducted under the advisement of Amy Bruckman, a professor in Georgia Tech’s School of Interactive Computing. This work was made possible by US National Science Foundation grant 0837733.

References


Betsy DiSalvo is an assistant professor at the Georgia Institute of Technology’s School of Interactive Computing. Contact her at bdisalvo@cc.gatech.edu.

Contact department editors Gitta Domik at domik@uni-paderborn.de and Scott Owen at sowen@gsu.edu.