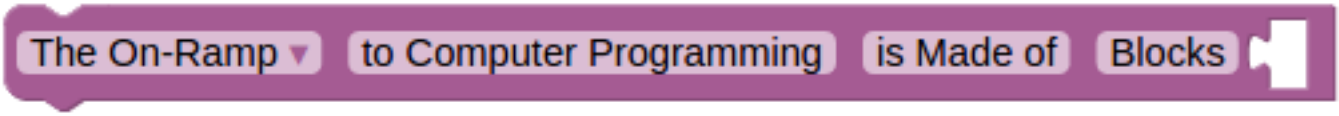


An Analysis of the Effectiveness of Blocks-based Programming Languages in Classrooms

Maxwell Lancaster

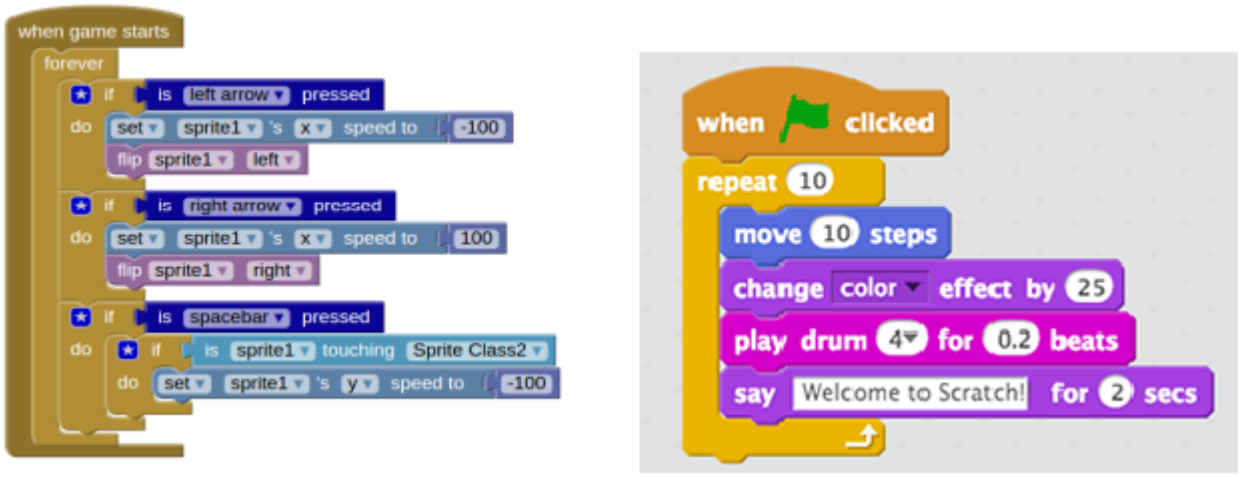
mlancast@mit.edu

Today's technologies play roles in nearly everything we do. From shopping for clothing or groceries, to social media and online banking, the field of Computer Science has trickled into every aspect of our lives. As such, the ability to effectively communicate within and contribute to this field will soon be essential to having a foundational understanding of how our society functions. The method by which we do this communicating and the tool that we use to make contributions to the field of Computer Science are known as computer programming. Programming is a discipline that requires critical thinking, creativity and a level of organization that is often not present in other disciplines. For this reason, it serves as an excellent complement to the traditional foundational subjects taught in early education: math, reading, and writing. There is general value associated with the ability to think like a programmer does. Programmers are exceptionally good at breaking problems down into their essential components, and solving each of these components individually. A basic, working, programming knowledge will soon be an essential piece to the larger puzzle of intelligence.

Another major problem – which is perpendicular to the problem of getting programming courses into our schools – is convincing students to pursue careers in computer science. According to the U.S. Bureau of Labor Statistics, there were 913,000 computer programmer jobs in 2010. That number is expected to grow by 30% by 2020 [1]. The U.S. will not produce enough programmers to fill these jobs. Because of this, programmers are in high demand. So how do we inspire students to pursue a career in computer science? This has been the goal of a new style of programming languages - known as blocks-based programming languages - which are geared to provide easy on ramps to the programming world. Blocks-based programming languages are typically graphical environments that allow users to create programs with only their mouse by dragging and dropping blocks which correspond to an instruction or a set of instructions. These blocks usually snap together to indicate that the user is putting together a cohesive program. Figure 1 depicts two examples of blocks-based programming languages.

The evidence presented here is intended to show that programming is an essential component of any student's early education, right alongside mathematics and reading/writing. In addition, blocks-based programming tools facilitate a faster and easier introduction to the field of computer science in middle-school aged students than their text-based counterparts. Lastly, educators are better equipped to engage and inspire their students to pursue careers in computer science when using blocks-based programming tools.

Figure 1: Two examples of blocks-based programming languages. (Left) Gameblox, (right) MIT Scratch.



Improvements on Traditional Programming Education

Led by popular tools such as MIT Scratch [2], Google Blockly [3], and Android App Inventor [4], blocks-based programming languages have already propelled themselves into today's classrooms featuring syntax-free program construction; visual, real-time, constructive feedback; user empowerment; and inherent teachings about abstraction.

User Experience

Most blocks-based programming tools utilize several different techniques to help users construct programs. Typically, blocks are organized by color to indicate function. A block's shape indicates which other blocks it may be appropriate to use with. Similar to traditional programming indentation, blocks can be nested to denote the scope of the instruction. More importantly, most blocks-based programming tools impose certain constraints on the construction of programs to encourage users to create syntactically correct programs. If two blocks cannot be joined to create a syntactically correct instruction, then the system does not allow them to be attached. In effect, this only allows users to create syntactically correct programs and eliminates all syntax-related, debugging headaches that are all-too-common for software developers. Further, this approach encourages experimentation from the user

by allowing them to repeatedly select and try to join different blocks. It is precisely this experimentation that allows a beginner-level programmer to easily and fearlessly create unique programs that they would otherwise not be able to create (at least, certainly not as quickly) in a traditional text-based programming language. Lastly, though the user may not create a syntactically incorrect program, he or she must still select each and every block in the program, effectively preserving the traditional instruction-by-instruction program construction used in all programing languages. Fred Martin from the Computer Science Teachers Association argues that the single largest benefit of blocks-based programming languages is their ability to encourage quick and syntax-free program construction, while maintaining the genuine intellectual challenge of programming [5].

User Empowerment

In general, blocks-based programming tools allow users to reach the end-goal, or finished version of their program, much more quickly than traditional text-based programming. This can be largely attributed to to the aforementioned syntax-free construction, which eliminates potentially painstaking debugging. In addition, most blocks-based programming tools provide immediate access to powerful APIs (tools for building applications) such as Gameblox's physics-enabled sprite movements, and App Inventor's message sending or web database querying. These tools are more readily accessible in blocks-based programming tools like Gameblox and App Inventor because the creators have pre-packaged these functionalities into the respective toolkits. In essence, users of these blocks-based programming tools have, at their fingertips, the tools to make powerful programs that do real things. However, creating a program that achieves similar results in a text-based programming language from the ground up often requires many lines of code that can be far beyond the capabilities of a beginner [5].

Further, many of these tools, such as Gameblox, Scratch, and App Inventor, encourage the sharing of work between users. Scratch and Gameblox feature gallery-like pages where a user can run and edit another user's programs. Similarly, much of the professional software engineering community is built around open-source projects. By encouraging a user to share their work, and view other peoples' work, these programming tools are empowering users to take pride in the programs they build, and to learn from the work of others - two important concepts in the field of computer science that are best learned early on.

Inherent Abstraction

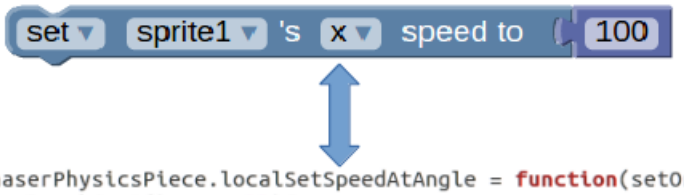
One of the most important topics in the field of computer science is known as abstraction. Abstraction is the single most effective solution to complexity [6] because it allows a system or program to be broken down into its most essential components. Each of these components is responsible for taking a certain input and producing a certain output. That's all.

For example, a car is an excellent showcase of abstraction that many people interface with on a daily basis. When a driver steps on the gas pedal, the throttle valve opens and

more fuel enters the engine. The driver does not need to manually inject fuel into the engine; instead, he simply steps on the gas pedal and relies on the corresponding components to perform their jobs. In addition, when the driver turns the steering wheel, this effort is passed through a system of joints down to the wheels. Each of these joints (components) executes its function and relies on the next one to do the same. In the end, the steering system is rather simple and very well engineered because each component is highly effective at performing a simple task. This is the beauty of abstraction: the driver need not understand how the fuel is injected into the engine, or how the steering box connects to the drop arm to drive a car correctly. The system works because each of its components performs its job correctly.

Blocks-based programming tools inherently teach students about abstraction. The blocks that a student uses in building his or her program often represent multiple lines of code and multiple different function calls. For illustrative purposes, Figure 2 shows the code that underlies a single block related to setting a sprite's[1] speed. Notice that this one block which sets a sprite's horizontal speed to a value of 100 actually corresponds to 10+ lines of code, including reference to several other functions. In a text-based programming language, a user would have to write all of this code to achieve his or her goal of setting a sprite's speed. In Gameblox, the user simply needs to grab this one block. By abstracting away the specific implementation, Gameblox allows the user to get to the solution as quickly as possible. This allows the user to focus on developing a creative, unique solution to a problem, instead of focusing on specific syntax details.

Figure 2: The underlying code for a speed block in Gameblox.




```
bd.methodTag.phaserPhysicsPiece.localSetSpeedAtAngle = function(setOrChange,speed,angle) {
  if(this.getDisplayObject() != null){
    var viewComponentObject = bd.component.lookup(this.model.viewId);
    var phaserGameObject = viewComponentObject.getPhaserGameObject();

    var velocityVector = this.getVelocityFromAngleAndSpeed(angle, speed);
    if(setOrChange == "set") {
      this.getDisplayObject().body.velocity.x=velocityVector.x;
      this.getDisplayObject().body.velocity.y=velocityVector.y;
    } else {
      this.getDisplayObject().body.velocity.x += velocityVector.x;
      this.getDisplayObject().body.velocity.y += velocityVector.y;
    }
  }
}
```

In another sense, by breaking the large problem down into its essential components, and then solving each of those components individually, abstraction often contributes to the elegance and organization of a particular solution. In the same way that middle-school

[1]A sprite is a 2D figure or character in a game

students are taught to organize their thoughts in the most understandable and logical way when writing an essay, blocks-based programming tools inherently teach students about composing their programs in the most logical and easy-to-follow way.

Gameblox

At the time of writing this (April 2016), I have two years of experience working in the MIT Scheller Teacher Education Program Lab (the Education Arcade). The STEP Lab builds games that help educators improve classroom learning environments, and help students build math and science skills. In particular, the project that I have worked on is called Gameblox (mentioned throughout in the preceding sections) a blocks-based programming tool designed specifically for making and playing games [11]. Gameblox is a unique blocks-based programming tool for a few reasons:

1. it teaches students about Object Oriented Programming
2. it will soon implement the above-mentioned blocks-to-text conversions
3. it is strictly designed to create games, instead of all purpose programs such as those created in MIT Scratch or App Inventor

Regarding (1), Gameblox is set up with difference classes, such as "Sprites", "Labels", "Sounds", etc., which can be thought of as types of objects. Then, the user can create instances of these different classes by simply dragging them onto the editing stage [12]. In using this system, users learn about Object Oriented Programming topics such as inheritance. For example, the user has the ability to set various properties of a Sprite Class in a sidebar on the left side of the interface. The user quickly learns that these properties apply to all instances of that class. So, each instance of the Alien Class that the user drags into his or her game, inherits the properties that he or she set for the entire Alien Class. In addition, the individual instances of the Alien Class (now located in the editing stage) can have instance-specific properties such as their x- and y-positioning, height, and width. These concepts are very similar to the true Object Oriented Programming Concepts that are taught in, say, an introductory Java course.

Gameblox and TAIL (Textual App Inventor Language), in particular, are developing tools to allow users to convert between blocks and text programs. Both of these languages will feature isomorphic text-based languages that allow programmers to convert between blocks and text as desired [9]. In addition, these languages will allow users to make edits to the program in text-form, and convert back to blocks [10]. This last feature facilitates more syntax-based learning, and will appeal to many users who are turned off by blocks-based languages in fear that they are not "real" programming languages [5].

In the same sense that different text-based programming languages have different syntaxes and different constructs, so do blocks-based programming languages. Some critics argue that the constructs used in blocks-based programming languages are overly simplified, and simply delay the need to learn language-specific syntax [13]. For example, Figure 3 depicts a "forever" block, found in the Google Blockly library and used in Gameblox. It is true that most programming languages do not have a simple statement or construct that

loops through a set of instructions forever, as this block does. However, most programs written for games (as is every project made in Gameblox) make use of a function called "update", which simply runs the game logic once for each frame of the game's execution. This is not unlike the "forever" block used in Gameblox, so there is clearly value in this kind of programming construct in blocks-based programming languages.

Figure 3: The Google Blockly "forever" block which constantly runs the stack of blocks inside of it.

Conclusion

We live in a world where the demand for instantaneous results has become the expectation in almost every facet of our lives. We have smartphone apps which, in addition to providing instantaneous access to the deepest corners of the Internet at our fingertips, have eliminated the wait for tasks such as finding a cab, choosing a restaurant, or even finding a date. By virtue of the shiny new technologies that get rolled out each week and the frequent improvements on computational power and efficiency, humans have become, by nature, less patient. These statements will be even more true for the next generation of students who come through our classrooms. For this reason, it has become increasingly easy for students to lose interest in science, technology, and engineering-related fields [14].

As mentioned above, the number of jobs related to computer science (and most other science/engineering related fields) is expected to surge in the coming years, but the interest levels in our classrooms across the country will not match that surge. The role of blocks-based programming tools is in mitigating this problem. Whether or not you buy into the idea that blocks-based programming tools are more effective at teaching introductory programming skills than their text-based counterparts, research has shown that, in one study, up to 92% of students surveyed cite blocks-based programming as easier to use [13] than text-based programming. Computer programming can be an intimidating concept for many young students [15]. At the very least, blocks-based languages are effective at breaking down that fear by improving ease-of-use.

Tools like Gameblox, and TAIL are leading the force behind block-to-text conversion capabilities, and I imagine this will be the next wave of major changes to the blocks-based programming world. Research has been conducted that shows that there is often difficulty in transitioning from graphical blocks-based programming to traditional text-based programming [13]. Thus, in working towards the perfect solution - the one which is effective at both inspiring students to pursue careers/education in computer science and most effective at teaching students about how to think like a computer scientist - blocks-based programming tools will most likely need to follow the lead of Gameblox and TAIL.

Computer Programming will soon be recognized as one of the core subjects taught in early-education classrooms. As a result, schools that do not respond to this trend by integrating the newest tools to teach computational thinking will be left behind in the 20th century.

References

[1] U.S. Bureau of Labor Statistics, 2015. *Software Developers.*

[2] Resnick, M. et al. 2009. *Scratch: Programming for all.* Communications of the ACM.

[3] Fraser, N. 2013. *Blockly.* Google.

[4] Abelson, H and Frienman, M. 2010. *App Inventor.* MIT Media Lab.

[5] Martin, F. 2014. *Real Programmers Use Blocks - A new definition of who is a programmer.* URL: http://blog.csta.acm.org/2014/10/28/real–programmers–use–blocks–a–new–definition–of–who–is–a–programmer/. Accessed April 4, 2016.

[6] *Abstraction.* Cornell University. CS211 L08 Abstraction.

[7] Karishma, C. 2014. *Improving the Usability of App Inventor through Conversion between Blocks and Text.* Wellesley University

[8] Karishma, C. and Turbak, F. 2014. *Improving App Inventor usability via conversion between blocks and text.* Journal of Visual Languages Computing. Volume 25. December 2014, Pages 1042-1043.

[9] Medlock-Walton, P. 2016. *Gameblox* MIT Scheller Teacher Education Program Lab. URL: http://education.mit.edu/portfolio_page/gameblox/. Accessed April 3, 2016.

[10] Medlock-Walton, P. 2016. *Gameblox.* URL: https://gameblox.org/. Accessed April 4, 2016.

[11] Weintrop, D. 2015. *IGCSE : G : Minding the gap between blocks-based and textbased programming: Evaluating introductory programming tools.* Association for Computing Machinery.

[12] Langfield, A. 2013. *Teens losing interest in science, tech money jobs.* Yahoo! Finance. URL: http://finance.yahoo.com/news/teens–losing–interest–science–tech–100000227.html. Accessed April 4, 2016.

[13] Holden, E and Weeden, E. 2005. *Prior Experience and New IT Students.* Informing Science. Pages 188 – 204.