Jonathan Speiser

CMSC390

SIMP: A Simplified Computational Thinking and Programming Tool for Children

"Computational thinking," a phrase coined by Carnegie Mellon Professor Jeannette Wing, represents an analytical and scientific approach to solving the challenges of everyday life. At the core of computational thinking is the ability to develop appropriate abstractions[1]. Abstractions convert the often messy and difficult problems of the real world into neater and more tractable problems. For example, long before the develop of today's powerful and ubiquitous multi-core processors, computer scientists developed the basic concept of the Parallel Random Access Machine (PRAM) to evaluate the performance of parallel algorithms. The evaluation of algorithms in the abstract model was enabled by specific assumptions that removed detailed complexities related to challenges in hardware, while retaining the essential high-level, intellectual challenges. It is precisely these higher level challenges that are most interesting and can catalyze dramatic innovation, as PRAM theory has shown.

Today, in introductory computer science classes, college students are challenged to leverage and develop their computational skills through programming. Developing code to successfully solve a particular problem inherently requires abstract analytical thought. For example, if the programming task is to develop a board game, the programmer must consider how to abstractly represent the state of the board over time. While Object-Oriented languages such as Java and C++ may provide suitable foundations for college students to develop appropriate abstractions, the syntax of formal programming languages presents a significant hurdle and a steep learning curve for children[2].

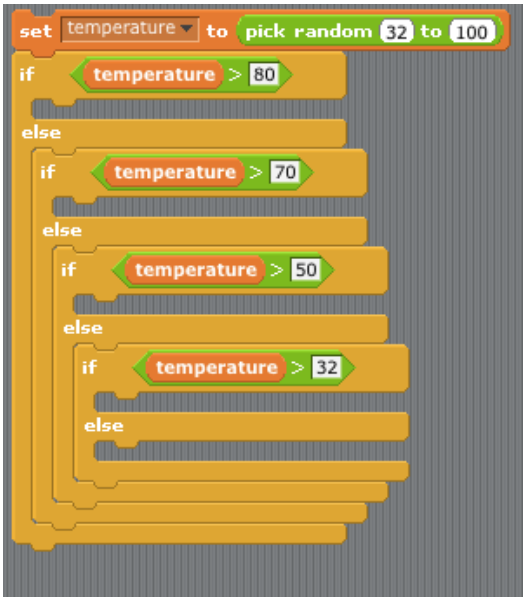
There are a number of existing tools for children to facilitate computational thinking. Some of the more

---

[1] Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.
[2] Utting, I., Cooper, S., Kolling, M., Maloney, J., and Resnick, M. 2010. Alice, Greenfoot and Scratch – A Discussion. ACM Trans. Comput. Educ. 10, 4, Article 17 (November 2010).

widely known paradigmatic implementations include LOGO and Scratch. While these approaches offer novel ways to simplify and alleviate the traditional pains of programming language syntax, it can be difficult in each system to develop a program to the point where critical analytical thought is required. To highlight this point imagine two exercises: first, emulate a trivial thermostat using LOGO and Scratch.

```
1  MAKE "TEMPERATURE (RANDOM 68) + 32
2  IF :TEMPERATURE  >  85 ...
3  IF :TEMPERATURE  >  75 ...
4  IF :TEMPERATURE  >  50 ...
5  IF :TEMPERATURE  >  32 ...
6  IF :TEMPERATURE <= 32  ...
7  PRINT :TEMPERATURE
```



The skeleton programs for the sample exercise may resemble the figures above. With this exercise, Scratch's interface ostensibly provides an intuitive representation of the underlying formal programming language syntax. While the LOGO example only requires several lines of code, simply wrangling with

the nuisances of variable assignment and usage in LOGO may already present a significant challenge for children[3].

For the next exercise develop a program in LOGO or Scratch to play Tic-Tac-Toe perfectly. While at a cognitive level, playing Tic-Tac-Toe perfectly is not an overly burdensome task, representing the required logic in child-friendly programming framework becomes an unwieldy task. It is precisely at this level of advanced cognition where existing solutions begin to show their inadequacies and break down.

The reasons that LOGO and Scratch break down when it comes to modelling systems that require a greater degree of analytical thought is that they are too broad in focus. Without an intuitive model for the fundamental data-structures users of Scratch and LOGO are forced to develop the data structures modelling the given tasks. This adds another lower-level tedious task that may present a significant barrier for children, who may have a reasonable subconscious grasp of the data model in their heads but would likely face great difficultly in developing or expressing that model. LOGO is, in essence, "a thinly disguised full-strength Lisp development environment."[4] While Scratch on the other hand takes care of challenges with programming syntax and is not merely a thin veil, it too does not adequately address the fundamental issue of making complex computation thought more palpable. Users of Scratch and LOGO are therefore necessarily forced to work at a lower level of abstraction, and consequently, it is difficult for users to reach the point where they can engage in meaningful computational thinking.

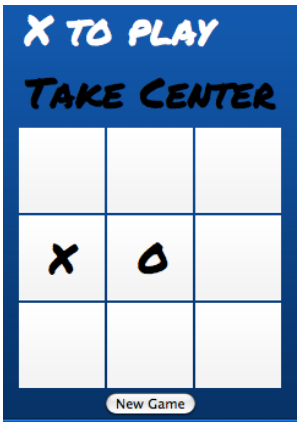
The Simplified Computational Thinking and Programming model (SIMP) represents the prototype of a rule-based programming approach. Instead of having students tackle the challenges of wrangling with a programming language, users are now free to think purely in terms in the domain-specific problem where the more interesting strategic challenges lie. With our SIMP approach, we are building a domain-

---

[3]Utting, I., Cooper, S., Kolling, M., Maloney, J., and Resnick, M. 2010. Alice, Greenfoot and Scratch – A Discussion. ACM Trans. Comput. Educ. 10, 4, Article 17 (November 2010).

[4]Weller, M., Do, E., and Gross, M., Escape Machine: Teaching Computational Thinking with a Tangible Interaction Design and Children State Machine Game. (IDC), ACM, (2008), 283.

specific tool, that allows users to focus on the high-level strategic concepts, rather than the low-level minutiae of programming language syntax. We have also abstracted the internal complexities of the data model away from users so that they need only to be concerned with issues related to high-level strategy. For example, in a Tic-Tac-Toe game, instead of searching through nine elements of a three-by-three grid, users of SIMP can leverage Tic-Tac-Toe specific rules that encapsulates the lower-level tasks of checking or setting the middle square. Additionally, SIMP's "VCR" mode allows users to step forwards and backwards through the high-level logical rules of a program as it runs and see which rules execute under which conditions.



We worked with nine children aged 7 through 10 from the KidsTeam program, to collect feedback on our initial drag-and-drop rule-based prototype system. We tested the system with the games Tic-Tac-Toe and Connect Four. The system turned out to still be challenging for students under the age of 8, while students over 8 were able to successfully develop strategies for the games. The session gave us the opportunity to collect valuable feedback and design suggests from the primary audience.

Based on the feedback we received from the KidsTeam session, we will look to build on the SIMP mode to further engage and educate students. In particular we will look to develop the system in such a way that children on the lower end of the age spectrum, such as 7 and 8 year olds, will be able to hone their computational thinking skills as well. One potential way to address the challenges of engaging the

younger students would be to develop a more robust interactive user-experience though live animations. Such animations could provide debugging real-time debugging capabilities as children assemble their rules. Coupling this real-time animated feedback with a color heat-map indicating the strength of the current program's performance could provide an extremely useful canvas for children. Younger students would be able to tinker and develop programs to the best of their ability by experimentation. Furthermore, we would also like to support the development of computational thinking on the older end of the age spectrum. Consequently, we would like to provide a mechanism by which we could expose the internal building blocks of existing rules to users so that they could easily and visually develop their own more advanced heuristics.

In conclusion, tools such as LOGO and Scratch provide sufficient solutions for students who are either already familiar with programming, or mathematically inclined (and familiar with algorithmic thinking). However, there remains a large void in terms of tools available to children who have no background in areas of computational thought, and who are starting from square one. It is precisely these students with little to no background where the SIMP model could be most effectively used and have the most profound impact.