Rodrigo, M. M. T. et al. (Eds.) (2021). Proceedings of the 29th International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Adopting No-Code Methods to Visualize Computational Thinking

Derrick HYLTON[a], Shannon SUNG[b*] & Charles XIE[b]
[a]*Physics Department, Spelman College, USA*
[b]*Institute for Future Intelligence, USA*
*shannon@intofuture.org

Abstract: In this paper, we describe the application of iFlow, a no-code programming platform, in visualizing foundational computational thinking (CT) as a precursor to learning disciplinary concepts. Research shows that it is possible to learn both science concepts and computational skills through building computer simulations and solutions of problems related to natural phenomena. CT could be operationally defined as the cognitive processes involved in problem formulation, in which its solutions could be effectively carried out by an information-processing agent (Wing, 2010). We introduce two modules as examples to facilitate the visualization of computational processes that are frequently adopted in teaching physical sciences. The first module was designed for students to analyze and evaluate data, and the second module allowed students to generate simulated data and scaffold questions to predict testable outcomes. In this workshop, we will demonstrate how to use iFlow to teach data analysis and problem evaluation that takes advantage of the no-code programming by means of the functional blocks, which contain operational codes to vividly visualize the input, process, and output.

Keywords: Computational thinking in STEM education, graph-based programming, no-code programming, visualizing computation

1. Introduction

Computational thinking (CT) could be operationally defined as the cognitive processes involved in problem formulation, in which its solutions could be effectively carried out by an "information-processing agent" (Wing, 2010). A major avenue to build CT is to integrate it into STEM disciplines, especially in mathematics and the physical sciences. In a 2013 report, The American Association for Physics Teachers (AAPT) has recommended that computational skills be incorporated into the undergraduate physics curriculum (AAPT, 2016), and it must include three of the following: spreadsheets, integrated mathematical computing packages, general-purpose programming language, and special-purpose software. In the 2015 Undergraduate Professional Education in Chemistry guidelines, the American Chemical Society (ACS) recommends that students should have access to computing facilities and chemistry software (ACS, 2015). Regarded as “the third pillar of science” alongside theory and experimentation (Skuse, 2019; Wing & Stanzione, 2016), computation is so fundamental in science that some even argue that “computational thinking emerged from within the scientific fields—it was not imported from computer science” (Denning, 2017, p. 14). CT is now a foundational skill for STEM workers and must be included in the undergraduate STEM curriculum, since Computer Science departments alone cannot meet the United States’ need for data science/computer professionals (US Department of Commerce) and probably for any other nation.

Research shows that it is possible to learn both science concepts and computational skills through building computer simulations and solutions of natural phenomena (e.g., Aksit & Wiebe, 2020; Dwyer et al., 2013; Hutchins et al., 2020; Sengupta et al., 2013). If so, we need to define and build these skills with relevant applications to create meaningful and effective pathways within STEM courses to also teach CT in addition to the standalone disciplinary knowledge. We introduced a graph-based programming platform—iFlow—to help users develop CT skills as well as programming skills. Inspired by the Unified Modeling Language and dataflow programming paradigm, this constructionist environment (Papert, 1991), named iFlow, models a program as an executable directed graph depicting the structure of a computational solution and the interactions among its constituents. The results emerge

as data flow through these interconnected elements. While there exist successful dataflow programming products such as Grasshopper, LabVIEW, and Simulink, most of them are tailor-made for specific applications that may not be appropriate for introductory courses. For example, Grasshopper only works within the Rhinoceros 3D computer-aided design software, LabVIEW is mostly used in data acquisition and instrument control, and Simulink focuses on modeling multi-domain dynamic systems. By comparison, iFlow is a general-purpose, Web-based, and integrated computational environment designed for students to analyze and solve common problems encountered in the math and science curriculum, with an objective to meet diverse educational needs of students with various backgrounds and interests in science.

Since computational problems can easily escalate into a complex mental model consisting of many abstract, intertwined moving parts that are often difficult for students to imagine and think through, the iFlow platform helps externalize learners' complicated mental processes of CT. Ideally, the thought processes may be subsequently profiled into systematic CT strategies. In other words, making students' computational thinking visible while they are shaping it can be cognitively offloading (Ainsworth, 2006; Ainsworth, et al., 2011; Schmidgall, et al., 2019). Visualizing computational processes could also be done through conventional flowcharts but they cannot be tested without actually programming an implementation. However, iFlow diagrams are palpably "live," where they can run immediately after each action students take while constructing and tinkering with artifacts. The learning environment can then automatically generate formative feedback to the student and to the researcher. Similar to students' concept map drawings that encode their understanding of specific concepts and their relationships (e.g., McClure, Sonak, & Suen, 1999), students' iFlow diagrams reflect their CT in solving specific science or engineering problems using a variety of building blocks that correspond to different computational concepts. Therefore, these graphic artifacts of students can also be collected and analyzed by researchers to test a hypothesis and by teachers to assess student learning (Jonassen & Cho, 2008), even for tacit knowledge that would be difficult to measure otherwise (e.g., Ahmad, Ahmad, & Rejab, 2011). Compared with text-based code, the same strengths of iFlow that assist student learning can also reduce assessment burdens. In summary, we anticipate iFlow to be an invaluable tool in identifying students' strategies and barriers to CT.

As an example, we found that students struggle with fitting data to a curve and developing a mathematical model, one of the first tasks in data analysis in an introductory physics course. We are interested in tackling the following questions: *What exactly is(are) the barrier(s)*? *Is it the coding*? *Is it the mathematics*? *Is it the numerical process*? *Is it the algorithm*? *Is it all of these to varying degrees*? The problem with text-based languages, such as Python, is that the computational processes take place in the background and students are left without any visualizations of data flow. We envision that iFlow can be used, by carefully constructing specific tasks, to identify some of these problems, and to inform intervention mechanisms.

2. Module Demonstration

2.1 Curve Fitting

The objective of curve fitting is to find a mathematical representation or function for a data set. One method to determine the goodness of the fit, suitable for educational purposes, is for students to visually examine how close the function aligns with the data. This method, which we call the "eyeball test", elicits students' problem analysis and evaluation skills that are fundamentally built on their visual examination, as they manipulate the parameters of the fitting function.

To facilitate curve fitting, we first provided several data points for them to paste into the array input block and outputted the data on the Space2D graph. Figure 1 shows the artifact of one student who engaged in the problem analysis and evaluation using the curve-fitting task. The data has been entered into an Array Input block (top left) and the output port on the right of this block is connected to a Space2D block (top right) throught port A on the left side of the block. This connection allows the data to be plotted in a straightforward manner. The remaining blocks in Figure 1 deals with the curve fitting. Parameters of the quadratic fit has been connected to sliders (bottom blocks). The values of the parameters are input to the defined function (block directly above the block with parameter *a*). Points

for this function can be generated via the Worker block, which is inputted to Space2D for graphing via port B.

In order to accomplish this activity, we first introduced some of the available blocks (shown in the panel in the left side of Figure 1 under the title Blocks) and contextualized the task for students to analyze the problem and evaluate how a proper function line or curve that best describe the relationship among these data points look like. Students should determine what constitutes a "good fitting function" by visualizing the trend to describe these non-linear data points. They may start with a simple equation, such as $y = mx + b$ for a straight line (the value of x and y maps to the data points, where m is the slope, and b is the intercept) and they may find that the straight line does not represent the plotted data. Thus, they need to resort to the quadratic equation $ax^2 + bx + c = y$ and manipulate the parameters (a, b, c) using slider blocks in iFlow. Since each slider is connected to a parameter via actionable nodes and arcs, students can simply drag the slider to get synchronous change in the curve.

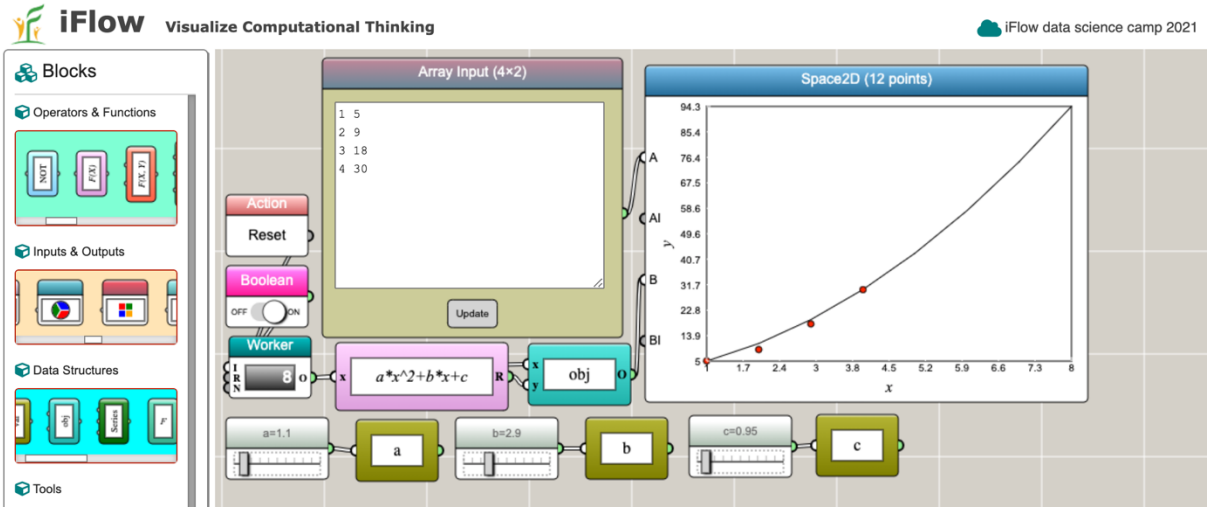

Figure 1. The graph-based module shows the visual code for curve fitting. Students were provided with four coordinate points in the array input block (top left) and were tasked to find a curve that describes these data points.

It is better if students first plot the data and put some thought into the type of function that would best fit the data. The platform will not choose the function; students must choose the function themselves and use the eyeball test to determine the best-fit parameters. Since we want to emphasize no-code programming, we need to give them the programming platform for them to compute without writing a single line of code. What we offer is to give them something to "program," using blocks and the algorithm that explicitly shows them how the actionable nodes and arcs are connected.

The advantage of adopting iFlow in curve-fitting is so that each change of the parameters can be easily computed and visualized quickly. In addition, students can observe how the data "flow" into a series of operational blocks and outputted on the graph, which makes the data transformation process explicit.

In traditional science class, this is often accomplished on Excel or on graphing calculator. *What are the advantages of iFlow*? First, students can clearly see the independence of the two graphs of the data and the fitted function. They are separated and come together in the graphing block. This independence of the fitting function to represent the data does not clearly come across in other platforms, even in text-based programming, as the "x" values of the fitted function are usually chosen to be the same as the "x" values of the data points. There is often confusion between the data points and the fitted function. Second, the connection of each parameter to a feature of the fitted function can be made clear, simple, and direct. Sliders can be done in other platforms and similar connections can be made, but with more distractions, such as columns of numbers or strange syntax. Third, students can powerfully visualize how everything works together, such as the "obj" block taking "x" values from the worker and "y" values from the function to create points to plot.

2.2 Projectile Motion

CT is not limited to data analysis such as curve fitting, it is also applicable in making predictions from a theory. Another module is the projectile module that showcases how scientists use computation to make predictions. It can also be applied to reverse engineer a curve fitting activity, in which we can generate simulation data from projectile module and feed the data into array input block in the curve fitting module to demonstrate how the scientists come up with a predicted relationship that can be tested experimentally. For instance, we can change the speed or angle of the projectile to determine the range or distance of the soccer ball (see Fig. 2) and fit this to a function. Since it may be difficult for experimenters to replicate the exact speed and angle, it is better to predict the relationship rather than the particular numbers.

The problem analysis is still applicable in this module, and the students need to synthesize the given problem by identifying answers to questions, such as "*What are the variables*? *What should be the input and output*? *How should the blocks be connected to obtain expected output*?" Also, the visualization helps users understand the underlying algorithms and deepening their learning of the associated variables in the equation. For example, *Why g is associated with the projectile movement*? *Why t is needed to depict the desirable output*? *Does mass matter*? Students can query the program with questions such as *Does the initial angle impact the horizontal distance, and if so, how? Does the height impact the horizontal distance, and if so, how*? These may be scaffolded questions.

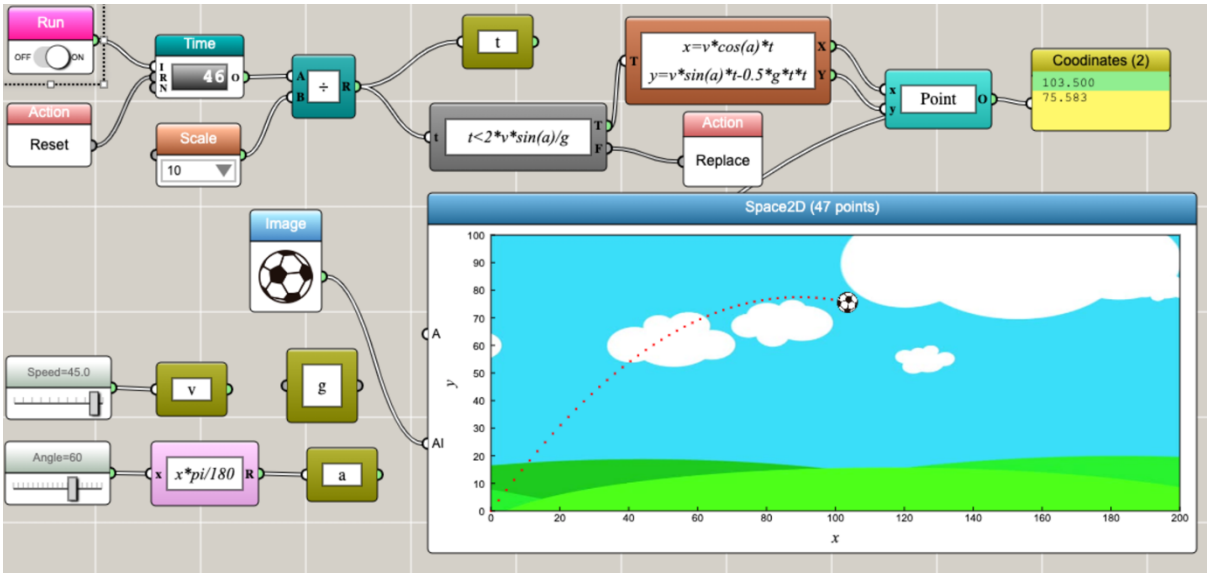

Figure 2. The module shows the visual code for projectile motion. The x and y coordinates as a function of time are determined by the following equations: $x = v\cos(\theta)t$, $y = v\sin(\theta)t - gt^2/2$ where v is the initial speed, θ is the initial angle, and g is the gravitational acceleration. The time (T) it takes the object to hit the ground is: $T = 2v\sin(\theta)/g$

Students can come away with several outcomes in using iFlow to study projectile motion. First, students can readily and quickly see how a change of angle or initial speed affect the motion all in one place. Second, students can clearly see that as time advances (from the Time block), the ball moves via its effect on x and y that come together to form a point along the path of the ball. Third, extraneous syntax does not cloud the essential features of the problem and how everything is connected and works together.

3. Significance of the Workshop

Based on the two modules described above, we conclude that students can apply the problem analysis and evaluation practices learned during the computational activities. We are not saying that students cannot come to a good understanding using other platforms, but visual stimulus is powerful in

promoting understanding. The connectivity and the necessity of certain functional blocks also demonstrate the essential ingredients of developing a computational model of a particular problem and thus, it promotes CT.

The iFlow platform is designed to visualize users' CT, so it is more versatile than scientific simulation platforms, particularly in eliciting students' procedural knowledge in computing, problem analysis, and evaluation. iFlow offers a more powerful platform to empower students in taking the role of programmers, which in turn gives them more agency in creating the artifact that can execute the student developers' commands, not only interacting with the predefined interface.

Computational tasks, such as physics simulations, can be captured in system-provided or user-defined blocks that are accompanied by simple graph-based user interfaces. This philosophy of iFlow resonates with the no-code movement in empowering people's development of apps without coding, thus unleashing their creative potential and inviting more to join the software workforce — irrespective of their programming experience

We plan to collect more empirical data that will focus primarily on students' computational problem-solving practices (Weintrop et al., 2016), consisting of *preparing problems for computational solutions, programming, choosing effective computational tools, assessing different approaches/solutions to a problem, developing modular computational solutions, creating computational abstractions*, for curriculum design.

We will investigate students' barriers to acquiring adequate problem analysis competencies that especially involve computational thinking in the future studies. It is important to understand these barriers for students taking a natural science course, in order to target our intervention strategies for the development of CT in these courses. The targeted intervention will include iFlow activities which should empower students to better transition to text-based programming, such as Python.

Acknowledgements

This project is supported by the National Science Foundation (NSF) under grant numbers #2131097 and #2107104. Any opinions, findings, and conclusions or recommendations expressed in this material, however, are those of the authors and do not necessarily reflect the views of NSF. We thank Kyla Price for attempting the creation of the curve-fitting sample module shown in Figure 1.

References

Anderson, R. E. (1992). Social impacts of computing: Codes of professional ethics. *Social Science Computing Review*, *10*(2), 453-469.

Chan, T. W., Roschelle, J., Hsi, S., Kinshuk, Sharples, M., Brown, T. et al. (2006). One-to-one technology-enhanced learning: An opportunity for global research collaboration. *Research and Practice in Technology-Enhanced Learning, 1*(1), 3-29.

Ahmad, K. B., Ahmad, M., & Rejab, M. M. (2011). The Influence of Knowledge Visualization on Externalizing Tacit Knowledge Paper presented at the Proceeding of the *International Conference on Advanced Science, Engineering and Information Technology*, Hotel Equatorial Bangi-Putrajaya, Malaysia.

Ainsworth, S. (2006). DeFT: A Conceptual Framework for Considering Learning with Multiple Representations. Learning and Instruction, 16(3), 183-198. doi:https://doi.org/10.1016/j.learninstruc.2006.03.001

Ainsworth, S., Prain, V., & Tytler, R. (2011). Drawing to Learn in Science. Science, 333(6046), 1096-1097. doi:10.1126/science.1204153

American Association of Physics Teachers (AAPT). (2016). Recommendations for Computational Physics in Undergraduate Physics Curriculum. Retrieved from https://www.aapt.org/resources/upload/aapt_uctf_compphysreport_final_b.pdf

American Chemical Society (ACS) (2015). Undergraduate Professional Education in Chemistry. Retrieved from https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiAo5nGisHsAhUHR6wKHbg5CkQQFjABegQIAxAC&url=https%3A%2F%2Fwww.acs.org%2Fcontent%2Fdam%2Facsorg%2Fabout%2Fgovernance%2Fcommittees%2Ftraining%2F2015-acs-guidelines-for-bachelors-degree-programs.pdf&usg=AOvVaw2zS-S7CDNXzLj52gE35bbw

Aksit, O., & Wiebe, E. N. (2020). Exploring Force and Motion Concepts in Middle Grades Using Computational Modeling: a Classroom Intervention Study. *Journal of Science Education and Technology*, 29(1), 65-82. doi:10.1007/s10956-019-09800-z

Denning, P. J. (2017). Computational Thinking in Science. *American Scientist*, 105(January-February), 13-17.
Dwyer, H., Boe, B., Hill, C., Franklin, D., & Harlow, D. (2013). Computational Thinking for Physics: Programming Models of Physics Phenomenon in Elementary School. Paper presented at the *Physics Education Research Conference*, Portland, OR.
Hutchins, N. M., Biswas, G., Maróti, M., Lédeczi, Á., Grover, S., Wolf, R., . . . McElhaney, K. (2020). C2STEM: a System for Synergistic Learning of Physics and Computational Thinking. *Journal of Science Education and Technology*, 29(1), 83-100. doi:10.1007/s10956-019-09804-9
Jonassen, D., & Cho, Y. H. (2008). Externalizing Mental Models with Mindtools. In D. Ifenthaler, P. Pirnay-Dummer, & J. M. Spector (Eds.), *Understanding Models for Learning and Instruction* (pp. 145-159). Boston, MA: Springer US.
McClure, J. R., Sonak, B., & Suen, H. K. (1999). Concept Map Assessment of Classroom Learning: Reliability, Validity, and Logistical Practicality. *Journal of Research in Science Teaching*, 36(4), 475–492.
Papert, S. (1991). Situating Constructionism. In I. Harel & S. Papert (Eds.), *Constructionism*. Norwood, NJ: Ablex Publishing Corporation.
Schmidgall, S. P., Eitel, A., & Scheiter, K. (2019). Why do learners who draw perform well? Investigating the role of visualization, generation and externalization in learner-generated drawing. *Learning and Instruction*, 60, 138-153. doi:https://doi.org/10.1016/j.learninstruc.2018.01.006
Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, 18(2), 351-380. doi:10.1007/s10639-012-9240-x
Skuse, B. (2019). The third pillar. Physics World, 32(3), 40-43. doi:10.1088/2058-7058/32/3/33
Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5
Wing, J. M. (2010). Computational Thinking: What and Why? Retrieved from https://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf
Wing, J. M., & Stanzione, D. (2016). Progress in Computational Thinking, and Expanding the HPC Community. *Communication of the ACM*, 59(7), 10-11.