CEO: A Triangulated Evaluation of a Modeling-Based CT-Infused CS Activity for Non-CS Middle Grade Students

Nicholas Lytle, Veronica Cateté, Yihuan Dong, Danielle Boulden, Bita Akram, Jennifer Houchins, Tiffany Barnes, Eric Wiebe
{nalytle,vmcatete,ydong2,dmboulde,bakram,jkhouchi,tmbarnes,wiebe}@ncsu.edu

ABSTRACT

With the increased demand for introducing computational thinking (CT) in K-12 classrooms, educational researchers are developing integrated lesson plans that can teach CT fundamentals in non-computing specific classrooms. Although these lessons reach more students through the core curriculum, proper evaluation methods are needed to ensure the quality of the design and integration. As part of a research practice partnership, we work to infuse research-backed curricula into science courses. We find a three-pronged approach of evaluation can help us make better decisions on how to improve experimental curricula for active classrooms. This CEO model uses three data sources (student code traces, exit ticket responses, and field observations) as a triangulated approach that can be used to identify programming behavior among novice developers, preferred task ordering for the assignment, and scaffolding recommendations to teachers. This approach allows us to evaluate the practical implementations of our initiative and create a focused approach for designing more effective lessons.

CCS CONCEPTS

• **Social and professional topics → Computational thinking**; *K-12 education*;

KEYWORDS

Computational Thinking, Modeling and Simulation, Assessment

ACM Reference Format:
Nicholas Lytle, Veronica Cateté, Yihuan Dong, Danielle Boulden, Bita Akram, Jennifer Houchins, Tiffany Barnes, Eric Wiebe. 2019. CEO: A Triangulated Evaluation of a Modeling-Based CT-Infused CS Activity for Non-CS Middle Grade Students. In *ACM Global Computing Education Conference 2019 (CompEd '19), May 17–19, 2019, Chengdu,Sichuan, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3300115.3309527

1 INTRODUCTION

Computer science and computing technologies have become recognized vehicles for economic growth and development across the globe. In order to meet the demands of this growing workforce with well-qualified employees, more countries are focusing on increasing their primary and secondary grade computing courses. Computational knowledge has far-reaching benefits beyond standard technology companies. Computational thinking (CT) [17] and other aspects of computing are being used to solve advanced, multi-faceted, problems around climate change, marine ecosystems, and beyond. Introducing computing into the science classroom provides equitable access to computing training as well as demonstrates to students the usefulness of computing in outside fields.

Previous research has examined teacher preparedness and professional development on the impact of classroom implementation and teacher confidence [3, 5, 13]. Studies show that students with teachers more interested in facilitating CT exhibit higher levels of time on task and engagement than those with disengaged teachers. While previous research focuses primarily on teacher outcomes, more research is needed on student outcomes (i.e. learning and programming behaviors) in order to successfully gauge the curriculum and support materials.

We propose a CEO model to assess the implementation of a CT-infused science curriculum. The model examines student **C**ode traces, **E**xit tickets, and field **O**bservations in order to triangulate the effectiveness of curriculum implementation and student learning outcomes. In order to effectively infuse computing, students must achieve both an understanding of computer programming as well as reinforcement of the scientific concepts being taught in class.

Using the CEO model, we attempt to identify improvements for the existing curriculum, specifically:

(1) How are students completing the assignment tasks?
(2) What emergent behaviors or observations can we identify in order to better engage students in future iterations?
(3) How do students perceive the science vs CT learning goals?

2 BACKGROUND & RELATED WORK

Research practice partnerships (RPPs) are long-term collaborations between practitioners and researchers that are organized to investigate problems of practice and solutions for improving schools and school districts [4]. One type of RPP focuses primarily on design-based implementation research [6]. This research aims to study solutions implemented in real world contexts, typically utilizing a cycle of developing and testing instructional activities and curricula.

We detail the cycle of creating, piloting, and re-designing a life-sciences curriculum in a prior case study [8]. The grounding decisions for activity changes included realizations that the cognitive demands for learning both CT and science concepts could be too high for the diverse range of middle-grade students. Additionally, students without prior programming exposure would benefit from tutorials and scaffolded directions. Thirdly, interface distractions could derail students and cause extra demands to cognitive load[16]. Many of these realizations however, came from short lab-controlled settings or subjective team reflections and discussion.

CompEd '19, May 17–19, 2019, Chengdu,Sichuan, China

ACM ISBN 978-1-4503-6259-7/19/05.
https://doi.org/10.1145/3300115.3309527

As Fishman points out, there is a broad challenge to gather and interpret evidence of effectiveness in the field as this differs greatly from a controlled setting [6]. Strategies for gathering evidence include student assessments, self-report surveys, rigorous observations [11], along with trace data analyses and video-recorded sessions. Although many of these practices are common in the field of educational research, they are often difficult to set up and take place primarily in closed lab environments [14, 15].

Upon entering the implementation cycle of design-based research, additional data collection methods should be used. In a previous report, educational researchers described the implementation of computing oriented science activities in middle grades classrooms [3]. This research focused primarily on the teacher's willingness to adopt the activity and effects of just-in-time professional development, noting that teacher buy-in had a large impact on student perception of usefulness and active on-task engagement. Although this research is well documented, it relied solely on qualitative data, including teacher interviews and observations. The report did little to convey the student learning outcomes or other evidence of student success.

Conversely, work by Grgurina et al. focuses on assessing modeling activities in a secondary grade computing classroom [7]. The assessment uses a combination of the Revised Bloom's Taxonomy [9] and SOLO taxonomy [2] to evaluate students' written answers to a number of questions regarding their models on multiple dimensions ranging from prestructural (information makes no sense) to extended abstract (generalization and transfer) for areas of design, experimentation, and reflection on the model. This assessment, however, is summative in nature using a multi-week modeling and research homework assignment as it's main vessel. The assessment provides less formative feedback on their specific skills and knowledge and instead measures larger concepts.

As our current research concerns middle grades students who are still developing their scientific knowledge and coding abilities, more formative feedback is needed. Consequently, we need to be able to better assess their programming behaviors so that we may provide more support or scaffolding for emergent behaviors in future iterations. Prior research in assessing science and modeling activities in secondary classrooms have focused on teaching, adoption, and summative assessments. In our new research, we look at evaluating an experimental curriculum through both empirical and qualitative evidence of student understanding.

3 METHODS

We created and utilized a CEO model to assess student outcomes, our curriculum as a whole, and the effectiveness of our implementation. Each of the elements of CEO describe a different data set and methodology for curricula analysis. Code Traces (C) describe the actions within the coding environment used by the students, Exit Tickets (E) describe post-activity written responses, and Observations (O) are researcher notes taken during implementation. In the following sections, we describe the curriculum and implementation, followed by our data collection methods.

3.1 Curriculum & Implementation

The curriculum development team, composed of educational psychologists and computing education researchers, created an 8th-grade life science lesson on Epidemic Diseases infused with computational thinking. The lesson was designed to be aligned to national and state-level science standards as well as the k12cs.org Computational Thinking Framework. This five-day unit focused on modeling the spread of epidemic diseases like the flu, see Figure 1.

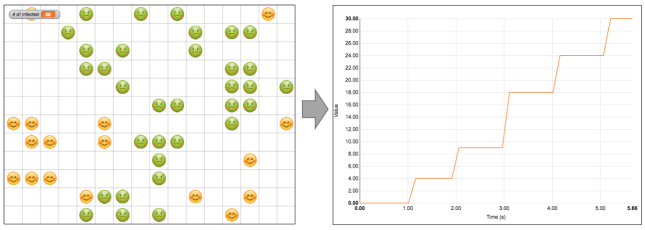


Figure 1: A Cellular representation of an epidemic.

Individual activities are described as "plugged" if students used a programming environment, or "unplugged" if students were learning without a computer. Students were given a block-based programming tutorial prior to the run of the unit. The unit features 2 unplugged days focused on modeling agent-host relations in the transmission of disease. There were also 3 plugged days: 2 of which had students develop a simulation based off the model discussed on the "unplugged" day using the block-based programming environment, Cellular [1], and 1 day where students used the environment to solve scientific research questions about the spread of diseases. An overview of the activities are shown in Table 1.

Table 1: Epidemics outline, P: plugged, U: unplugged

Day	Learning Goals
1: U	Able to explain a simple model. Understand Hosts and Agents share properties with modified values.
2: P	Define infection and infection rate. Demonstrate understanding of agent properties. Understand and use loops and conditionals.
3: P	Understand disease spread and rate of transmission/infection. Use variables to maintain count. Analyze trends in data to identify patterns. Demonstrate understanding of how interaction properties can affect simulations.
4: U	Understand Morbidity/Mortality rates and their influence on spread. Understand and use Finite State Machines to model logic flow. Model algorithmic thinking through transition modeling.
5: P	Understand how environmental factors affect disease spread; Learn experimental procedure for hypothesis testing. Visualize data; Use simulations to test hypotheses.

We tested the 5-day Epidemics activity with two 8th grade science teachers at a local middle school. Each teacher taught 5 sections of students with class sizes ranging from 20 to 25 students. Each of the teachers' classes were presented the same activity on the same day. Teachers taught every class period on unplugged days of the unit. On plugged days (2, 3, and 5), teachers had one observation period at the start of the day where they experienced the content as a student, following along as the researchers led the first class. They

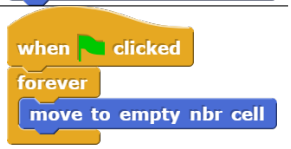
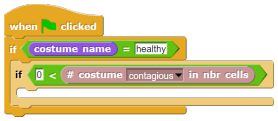
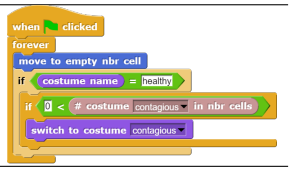
then taught the remaining 4 periods of the day, guiding students through program implementation. 61 students returned consent forms for us to analyze their data.

3.2 Code Traces

On days 2 and 3, students' actions within the Cellular environment were logged in a database for later examination. We focus our analysis on the actions that make changes to their code. Specifically, we call a student's code at any point a *code state*, and the sequence of code states that leads to the final solution a *code trace*.

For days 2 and 3, students were given instructions with individual tasks to complete within the environment. These tasks were presented sequentially, but the order in which students could complete the assignment did not have to match the intended order. These tasks are outlined in Tables 2 and 3. In examining student code traces, we can identify whether or not a specified task has been completed at any given point in the code trace. We refer to the instance in which a specified task has been completed as a *task completion state*. The second column of Tables 2 and 3 provides one of the many solutions that would result in the completion of that task.

Table 2: Task milestones for day 2 of the epidemics activity

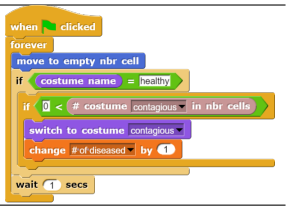
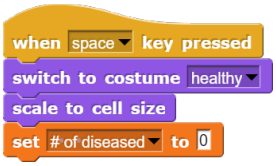
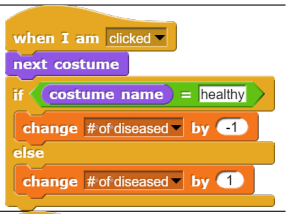
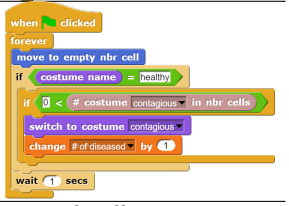
Task/Feature	Example Solution
(1) Write a program that moves your sprite to an empty neighbor cell.	
(2) Use a control loop to have your sprites move to empty neighbor cells forever.	
(3) Add an if control block to your main script that checks whether your sprite is healthy before checking its neighbor cells for an infected sprite.	
(4) Add a script to your forever loop that makes your sprite infected if an infected sprite (a sprite with infected costume) is in a neighbor cell.	
All	

Task completion states were individually tagged by a member of the research team and build sequentially as students complete additional tasks. When a student has a task completion state with all tasks present, they have fully completed the assignment. To illustrate, take two students who completed day 2's activity, one by entering completion state "1234" and another by entering completion state "1243". The presence of all four tasks (i.e. tasks 1,2,3,4) in both states signify both students completed the full assignment. The order of the numbers in the state tells the sequence of tasks the students completed in the assignment (notice the order difference in how they completed the last two tasks - 3 and 4).

Using code traces to identify student programming behaviors gives light to RQ1 as they can illustrate common solution strategies, most difficult tasks, and common pitfalls in student implementations. As these states and actions are also time-stamped, specific strategies by students can be compared to see whether or not certain strategies are easier (i.e. faster to complete) than others.

To complement our task completion state analysis, we went through traces and made qualitative observations of other behaviors in the environment we found interesting. These included behavior taken by the students after the assignment was complete, resets of the environment (giving up and starting over again), spots where students found difficulty in completing tasks, and off-task behavior.

Table 3: Task milestones for day 3 of the epidemics activity

Task/Feature	Example Solution
(1) Create a variable to hold number of infected sprites and name it # of diseased. Set this variable to zero in the beginning of your program (just after the "when green flag clicked" block)	
(2) Write a script that increases the value of the infected people by one when a sprite gets infected.	
(3) Write a script that restarts your simulation when you hit space. The restart script should make all sprites healthy and set the number of infected to 0. Add your restart script to this part of the code on the screen.	
(4) Write a script that changes the value of the variable: # of diseased when you click on a sprite.	
(5) Remove the set # of diseased from your main script (the script after "when green flag clicked")	
All	3,4, and 5 all co-present.

3.3 Exit Tickets

On plugged days, students were asked to take an end of class "Exit Ticket". This series of questions measured self-reported affect, student engagement, and perception of what they learned. The 61 consenting students generated 127 responses over three days of activity. The relevant questions are listed below:

(1) What did you learn today?
(2) What was the most helpful to you: the tutorials, your classmates, your teacher?
(3) Did you find anything difficult or frustrating? Please explain.
(4) Did the lesson go too fast, too slow, or just right?
(5) Do you have any suggestions to improve the activities?

For question 1, our aim was to answer RQ3 by analyzing student perception of the intent of the activity. As this was a CT-infused science lesson, we aimed to see whether or not the perception of the activity leaned more towards computing or science. Our approach consisted of having two researchers individually tag responses as being more focused on computing, science, both, or neither. After individual tagging, the researchers compared their tags and discussed cases where conflict existed until agreement could be reached. For questions 2 and 4, our aim was to see which of the three choices or combinations thereof were most present within an assignment. Finally, for questions 3 and 5, researchers counted and grouped the most common answers to use as feedback for the next development cycle.

3.4 Observations

During the implementation, at least one member of the research team acted as an observer on each plugged day. Each observer had prior classroom experience leading computing activities with middle and secondary age students (with three of the observers having 5+ years of experience). Our goal with classroom observations was to experience and record how the activity was conducted across different classrooms and to gather both insights for improving teacher training practices as well as improving the actual activity and its implementation environment. In most activity sessions, we assigned one observer to watch the teacher, and others to observe the students. We also had additional support to help students stuck on programming tasks, so that observers could stay focused on the field. Observers recorded student affect, behaviors, and interactions of note. After each session, observers conferred and noted interesting results to follow up with.

4 RESULTS

4.1 Code Traces

In Figures 2 and 3, we represent the paths students took to complete the assignment as a transition of progressive task completion states. These states correspond to the subset of tasks that have been completed thus far (e.g. state "12" represents finishing task 1 and then 2). The size of the shape corresponds to the number of individuals who entered that specific task completion state, and the size of the edges corresponds with the number of individuals who made that unique transition. Task completion states can often be stuck states (octagons in this representation) meaning that some students made no further progress in the assignment after reaching this state. This representation was adapted from a similar analysis of student programming paths(see [18]).

4.1.1 Day 2. The code traces for 61 participants on day 2 are shown in Figure 2. The task completion was mostly uniform, with participants only differing on whether or not they completed task 3 or 4 first. Most students, 33, completed the tasks in the intended order of 1234 while 22 completed the ordering as 1243. The third feature was the most missed (7 students omitted).

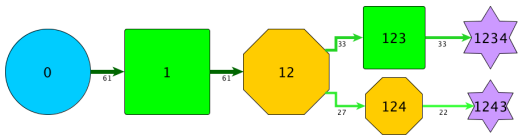


Figure 2: Task completion strategies for day 2.

Timing trends were also recorded and analyzed within the environment. The average time spent in the environment was 36 minutes. However, average time in environment varied considerably by class period. Periods 2 and 3 spent the most amount of time (41 and 39 minutes respectively) and period 4 spent the least (31 minutes). The last two periods spent 36 and 33 minutes within the environment. This difference in timing is important as all students within period 4 completed the features in the order 1243.

In looking through the traces, there is clear evidence for student confusion. 14 students reset the environment and started over at least once. Many students tried to complete their implementation with *distractor blocks* (blocks not useful in final solution). The most common distractor block was the "Touching" block, which was used to check if two sprites were overlapping, instead of checking if an agent was in a "neighboring cell". Another common block that students attempted to use in the solution was the "Move Step" block which was used in lieu of the "move to empty neighbor cell" block.

Students largely were able to complete the assignment with relative ease as there was 17 minutes on average left from the last feature was completed to close of the system. Due to the amount of time left at the end, many students elected to complete additional extension features with the most common (39/61) being a "reset functionality" that resets the simulation to its initial position.

4.1.2 Day 3. Among the 59 code traces, the average time spent in the environment was 39 minutes, ranging from 34 to 42 minutes. There were markedly fewer resets (only 6 students) than the previous day's programming assignment and many of these occurred after students made an error such as creating a new sprite (and losing track of their current sprite).

What is easily apparent by comparing the graphs of this assignment with the other is the incredibly varied paths that students take through the assignment. 11/59 students were unable to finish the assignment with five of these students missing all but Task 5. The 48 students who did finish the assignment took a myriad of paths. The majority of these solutions (34) did NOT include tasks 1 and 5. In fact, 44 students did not complete task 1 at all and of the 15 that did, nearly half (7) did not complete task 5 afterwards (and thereby not having correct final code).

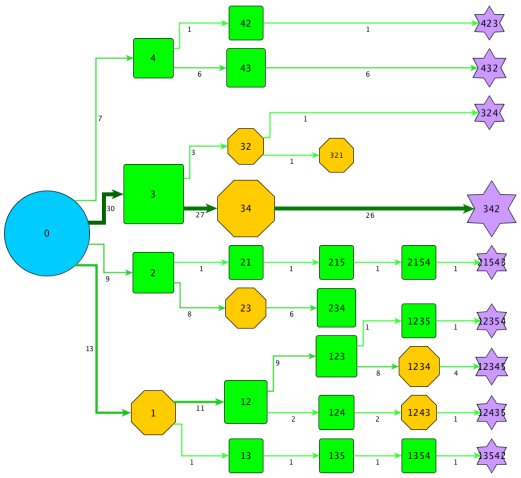


Figure 3: Task completion strategies for day 3.

Students who completed the assignment had less time on average (10 minutes) between finishing and close of the environment than those working on assignment 2, but this split also varied by solution strategy with those who did tasks 1 and 5 having less time (around 8.5 minutes) than those who just did only tasks 2, 3, and 4 (around 11 minutes). With the remaining time, students attempted a number of self-created extensions. Some students attempted to make other sprites in their model such as hospitals. Others attempted to extend the logic of their modeling by either stopping the simulation once the entire population was infected or keeping track of and graphing the remaining healthy population. Another student attempted to add interactivity into their model with key press events.

4.2 Exit Tickets

In addition to the main breakdown of students reporting learning science vs computing (see Table 4), we also wanted to examine how students articulate their new computing knowledge. We classified their learning statements by whether they mention a computing term or mention the term in connection to a specific goal. On day 2, 15 students mention sprites and costumes, 10 with specific goals in mind. Seven students mention programming blocks, but only two with a specific use in mind. On day 3, 23 students mention variables and plotting, 13 of which with a specific goal for use. This time, only four students mentioned sprites or costumes. Day 5, showed the biggest swing with most students reporting science, though 5 students also mention programming in general.

Most students on each day reported that they did not find anything frustrating with the activity. Of those that did, eight found programming errors and debugging frustrating, while five students found system related errors (problems saving, crashing, etc.) each day frustrating to deal with. Five students on day 2 and day 3 specifically mentioned adding the hospital was difficult, but this was an extension activity not explicitly supported in the curriculum. Finally, day 5's specific activity brought new challenges for five students as some were frustrated and confused about the terms introduced (e.g. independent variable) as well as using the tools in order to find the relationships in the assignment.

Table 4: Student responses to Exit Ticket questions 1, 2, and 4. Answers of none are omitted from the table.

Day	Balance	Most Helpful	Pacing
2 N=39	Computing: 23 Science: 3 Both:12	Teacher: 28 Tutorial: 5 Classmates:2	Perfect: 29 Slow: 5 Fast: 3
3 N=49	Computing: 34 Science: 5 Both:7	Teacher: 32 Tutorial: 8 Classmates:6	Perfect: 30 Slow: 13 Fast: 2
5 N=39	Computing: 0 Science: 26 Both: 7	Teacher: 24 Tutorial: 3 Classmates:6	Perfect: 23 Slow: 4 Fast: 5

Finally, students had a number of suggestions on how to improve the curriculum. Suggestions can be broadly categorized as having to do with the instruction (providing more support or improving the pacing); fixing technical problems with the programming environment (e.g. crashes, bugs); and activity-related suggestions (e.g. more extensions, time for group work, creative time).

4.3 Observations

Observers identified general patterns on all three days. First, concerns were raised about the task sequence in both days 2 and 3. For day 3, observers noted that the action of setting # of diseased to 0 under the Green Flag in task 1 didn't seem to serve a purpose and caused confusion in some cases. In later periods, teachers skipped this step entirely.

Observers noted that the ordering of tasks in day 2 was off. When comparing periods, the class seems to run smoother when students work to iteratively solve a problem and the nested-if (task 3) does not produce any visible change. Observers also noted that teachers adopted different strategies in different periods, resulting in the difference in pacing of the assignments. In some periods, the activities were led in an incredibly instructionist manner, leading students through each step. As teachers got more comfortable with the assignment, they began to add their own pacing and scaffolding approaches such as having students explain previous days code or creating parsons problems [12] for students to reason through as a group. Towards the end periods, teachers found ways to explicitly connect science concepts into the programming instruction.

Researchers observed the majority of students actively engaged during all three days of instruction with minimal off-task behavior. While students were generally able to code within the environment, some confusion was observed during day 5 of the activity. Students struggled to setup the simulation environment in a way needed to answer their group's research questions. This was alleviated in later periods with deliberate instructor support.

5 DISCUSSION & LIMITATIONS

In addressing the question "*How are students completing the assignment tasks?*", we examined the connection between code traces and observations. Overall, day 2 and 3 completion rates were strong with 55/61 and 42/59 students fully completing the assignment respectively. Those that completed the assignment tended to have extra

time left, suggesting mid to high performing students need additional activities and extensions. Initial observer concerns about task sequencing were corroborated by examining students progression through the completion states. For day 2, the quicker completion time as well as observations suggest that the task sequencing should be 1243 instead of 1234. Task 4 is the behavior that actually visually changes the sprites to infected, which as observers noted, is critical in engaging students. Moreover, task 3 requires a doubly nested if, while task 4 only requires a single if, which is a better scaffolding strategy and smoother sequence than the original task sequence (as noted by field observations of teachers guiding the instruction). The final recommendation for day 2's sequence is informed by the large number of field observed and code trace observed end of day behaviors of adding in the reset functionality. This should be explicitly added as a day 2 task, as the behavior and functionality is necessary for day 3's activity.

In examining day 3's sequencing, the most apparent result was that the placing of the set block under the green flag in task 1 and the removal of this block in task 5 were unnecessary additions to the assignment. Not only did it impede progress in many cases as evident by the code traces, observations from researchers showed that teachers and students would often forego actually completing the tasks at all. This lead to faster overall completions of the assignment as well as fewer stuck states. As the plurality of students completed the task by completing task 3 first and without doing tasks 1 and 5 fully, the results strongly support removal and reordering of the tasks.

In answering the question "*What emergent behaviors or observations can we identify in order to better engage students in future iterations?*", we identify several behaviors of actionable interest. First, blocks that are identified as distractors impede progress and can lead to student frustration and confusion, noted by both observers and exit tickets. Within the stuck states, it is also observable when students don't know which block to use. Since the goal of the curriculum is for the student to build the correct logic for the epidemics model instead of identifying the difference between blocks, we will hide the distractor blocks in future iterations in order to focus attention solely on blocks necessary for completing the assignment. Second, student exit tickets, observations and code traces all support the idea that students want more opportunities for open-ended exploration. With many extensions attempted in the extra time on days 2 and 3, the curriculum should explicitly provide additional scaffolding for students to be able to complete these extensions, including the reset functionality completed by the majority of students on day 2. Supporting these extra activities will also alleviate pacing issues with students finishing early and displaying disruptive/bored behavior.

In answering the question "*How do students perceiving the science vs computational learning goals?*", we primarily examine exit tickets and observations. Students self-reported learning topics correlate strongly with the amount of programming on a particular day. On days 2 and 3, which focused creating the simulation to determine rate of infection, the majority of students reported learning computing (89.7% and 83.7% respectively). On these days, students were able to articulate the connections between the sprites/costumes and the resulting agent behavior in the simulation as well as the usefulness of variables in being able to track the rate of infection. On day 5, which focused on solving research questions within the simulation the majority of students (84.6%) reported learning science. This trend in perceived learning, is further supported by observations that suggest teachers make deliberate breakpoints to connect the computing to science concepts previously covered in class as teachers introduced more guided reflections and connections with material as the day progressed. These moments of connection, created by teachers, should be explicitly supported within the curriculum in order to be able to truly promote the lesson as an infused activity.

Furthermore, several students reported the tutorial being helpful for each day, however, the majority of students report help from the teacher being most useful. Observers noted the teacher had a good sense of pace of the students on programming days, and helped students construct hypotheses and explain terms like susceptibility which they only briefly covered prior. The teacher also provided logical reasoning and discussions for students to understand why the simulation behaved they way it did.

The scope of this study is limited by mainly focusing on plugged days and not unplugged. We also did not record the action logs of students during the final day as no coding occurred in the environment. An issue that was not addressed in this study was student assessment of scientific concepts as this was also beyond the scope of our research. Finally, as is the case in much classroom research, an observer effect may have influenced findings.

6 CONCLUSIONS & FUTURE WORK

In this analysis of a CT-infused life science lesson, we created and used the CEO model to triangulate data from code traces, exit tickets, and field observations to evaluate the success of student outcomes in the experimental curriculum. Using these data measures, we identified popular student programming pathways that lead to successful completion of the assignment and pathways that lead to stuck states. The code traces provide empirical support for the subjective field observations and student exit tickets.

Other observations and suggestions corroborated by code traces include staging tasks in such a way that each milestone has a visual component or perceivable outcome by the students. Additionally, on days where students complete the main tasks quickly, additional content can be included, specifically, the functionality (such as reset) that most students already take upon themselves to complete. In general, these findings suggest a role for code traces as a initial attempt to empirically corroborate purely qualitative-based evidence. Thus, promoting a more robust pathway for iteratively improving computing-infused activities for non-computing students. While this process was done through inspection by a member of the research team, this process could be automated through the creation of unit tests for intended code behavior for each task.

A natural progression of this work is to investigate the transferability of these activities. Following Means and Penuel 2005, we want to further identify, "what works where, when, and for whom" [10]. We have piloted the epidemics curriculum in two additional schools within the same county. However, we would like to investigate if the lessons will also work at another institution, with a different school culture at differing times in the school schedule (i.e. before/after lesson plan; exploration vs. review activity).

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under 1742351 and 1742332. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Bernd Meyer Aidan Lane and Jonathan Mullins. 2012. *Simulation with Cellular A Project Based Introduction to Programming* (first ed.). Monash University, Melbourne, Australia. Online: https://github.com/MonashAlexandria/snapapps.

[2] John B Biggs and Kevin F Collis. 2014. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome).* Academic Press, New York, New York, USA.

[3] Veronica Cateté, Nicholas Lytle, Yihuan Dong, Danielle Boulden, Bita Akram, Jennifer Houchins, Tiffany Barnes, Eric Wiebe, James Lester, Bradford Mott, and Kristy Boyer. 2018. Infusing Computational Thinking into Middle Grade Science Classrooms: Lessons Learned. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education (WiPSCE '18).* ACM, New York, NY, USA, Article 21, 6 pages. https://doi.org/10.1145/3265757.3265778

[4] Cynthia E Coburn, William R Penuel, and Kimberly E Geil. 2013. *Practice Partnerships: A Strategy for Leveraging Research for Educational Improvement in School Districts.* Technical Report. William T. Grant Foundation, New York, New York, USA.

[5] National Research Council et al. 2011. *Report of a workshop on the pedagogical aspects of computational thinking.* National Academies Press, Washington, DC.

[6] Barry J Fishman, William R Penuel, Anna-Ruth Allen, Britte Haugan Cheng, and NORA Sabelli. 2013. Design-based implementation research: An emerging model for transforming the relationship of research and practice. *National society for the study of education* 112, 2 (2013), 136–156.

[7] Natasa Grgurina, Erik Barendsen, Cor Suhre, Bert Zwaneveld, and Klaas van Veen. 2018. Assessment of Modeling and Simulation in Secondary Computing Science Education. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education (WiPSCE '18).* ACM, New York, NY, USA, Article 7, 10 pages. https://doi.org/10.1145/3265757.3265764

[8] J. K. Houchins, D. C. Boulden, B. Akram, E. Wiebe, V. Cateté, Y. Dong, N. Lytle, A. Milliken, T. Barnes, J. Lester, B. Mott, and K. E. Boyer. 2019. Designing a Computational Modeling Unit for Middle Grades Science Classrooms: Grounding Decisions in Practice. In *2019 American Educational Research Association Annual Meeting.* AIED, Toronto, Canada, 15.

[9] David R Krathwohl. 2002. A revision of Bloom's taxonomy: An overview. *Theory into practice* 41, 4 (2002), 212–218.

[10] Barbara Means and William R Penuel. 2005. Scaling up technology-based educational innovations. *Scaling up success: Lessons learned from technology-based educational improvement* . (2005), 176–197.

[11] Jaclyn Ocumpaugh. 2015. *Baker Rodrigo Ocumpaugh monitoring protocol (BROMP) 2.0 technical and training manual.* Teachers College, Columbia University.

[12] Dale Parsons and Patricia Haden. 2006. Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52 (ACE '06).* Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 157–163. http://dl.acm.org/citation.cfm?id=1151869.1151890

[13] Thomas W. Price, Veronica Cateté, Jennifer Albert, Tiffany Barnes, and Daniel D. Garcia. 2016. Lessons Learned from "BJC" CS Principles Professional Development. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16).* ACM, New York, New York, USA, 467–472. https://doi.org/10.1145/2839509.2844625

[14] Jonathan P Rowe, Scott W McQuiggan, Jennifer L Robison, and James C Lester. 2009. Off-Task Behavior in Narrative-Centered Learning Environments.. In *Proceedings of the 14th International Conference on Artificial Intelligence in Education, AIED 2019.* IOS Press, Brighton, UK, 99–106.

[15] Robert Sawyer, Andy Smith, Jonathan Rowe, Roger Azevedo, and James Lester. 2017. Enhancing Student Models in Game-based Learning with Facial Expression Recognition. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization (UMAP '17).* ACM, New York, NY, USA, 192–201. https://doi.org/10.1145/3079628.3079686

[16] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.

[17] Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.

[18] Rui Zhi, Thomas W Price, Nicholas Lytle, Yihuan Dong, and Tiffany Barnes. 2018. Reducing the State Space of Programming Problems through Data-Driven Feature Detection. In *Educational Data Mining in Computer Science Education (CSEDM) Workshop @EDM 2018.* Educational Data Mining, New York, United States, 7.