Exploring the Use of Games and a Domain-Specific Teaching Language in CS0

Jennifer Parham-Mocello
Oregon State University
Corvallis, USA
parhammj@oregonstate.edu

Aiden Nelson
Oregon State University
Corvallis, USA
nelsonai@oregonstate.edu

Martin Erwig
Oregon State University
Corvallis, USA
erwig@oregonstate.edu

ABSTRACT

University students learning about computer science (CS) can be intimidated and frustrated by programming, and to make matters worse, the general-purpose programming languages chosen for introducing students to programming contain too many features that have the potential to overwhelm and distract students. We hypothesize that by using a delayed-coding approach with a language designed for teaching a smaller set of features focused on the fundamental CS concepts, such as types, values, conditions, control structures, and functions, student retention and success would improve, especially for those with no or little prior programming experience.

To test this hypothesis, we split a college computer science orientation class into two sections. One section began programming with a general-purpose language, Python, during week 1. The second section used a new, functional domain-specific teaching language themed around programming simple, well-known physical games. A group of researchers designed the new language with the purpose of giving students a more focused approach to learning basic computer science concepts and emphasizing good programming practices early, such as working with user-defined types and decomposition. Based on student survey responses before and after the two sections and their grades through the two subsequent CS courses, we find that students in the delayed-coding section using the new language had lower engagement in their class. In addition, we find no evidence of a higher pass rate for students from this section in their subsequent computer science courses.

CCS CONCEPTS

• **Applied computing** → **Education**; • **Social and professional topics** → *CS1*; **Computational thinking**; • **Software and its engineering** → **Domain specific languages**.

KEYWORDS

domain-specific language, functional programming, games, unplugged, CS0 and CS1

ITiCSE 2022, July 8–13, 2022, Dublin, Ireland

ACM ISBN 978-1-4503-9201-3/22/07…$15.00
https://doi.org/10.1145/3502718.3524812

ACM Reference Format:
Jennifer Parham-Mocello, Aiden Nelson, and Martin Erwig. 2022. Exploring the Use of Games and a Domain-Specific Teaching Language in CS0. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol 1 (ITiCSE 2022), July 8–13, 2022, Dublin, Ireland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3502718.3524812

1 INTRODUCTION

We developed a two-level curriculum for introducing computer science (CS) based on identifying computing concepts in well-known non-electronic games, which we refer to as the ChildsPlay approach. Our approach is similar to the approaches taken in CS For Fun (CS4FN), the Teaching London Computing, and CTArcade [13, 15, 32], which also employ physical games to teach CS concepts, but it differs in a fundamental way: Instead of focusing on the strategy for winning games or students playing against the computer, we use the instructions and rules for playing games without the use of a computer to introduce students to CS concepts, such as representation, algorithm, and computation before introducing them to programming. More specifically, our approach has the following five features.

- *Non-Coding First.* We deliberately avoid the teaching of a programming language in the first part of the curriculum.
- *Unplugged.* We don't use technology and embrace physical artifacts as teaching devices.
- *Games.* We employ physical games as the conceptual framework and metaphor for computing concepts.
- *Well-Known.* We use games well-known for being simple.
- *Domain-Specific Language (DSL).* We introduce students to a formal programming notation employing a newly developed DSL for describing board games.

One major goal of the ChildsPlay approach is to debunk negative perceptions of CS early by demonstrating to students that understanding basic concepts of computer science is as easy and as fun as playing games. We believe choosing well-known, simple games, such as Nim, Tic-Tac-Toe, and Rock, Paper, Scissors, makes CS more widely accessible for students.

In this research study, we use our approach in a university CS Orientation course to provide students with a gentle slope into the concepts of CS and programming. We use students' pre and post survey responses, prior programming information, and grades to determine if students in the section using the new approach with a new functional DSL like the new approach and are impacted differently than those in the traditional taught section using Python. We answer these questions by addressing the following:

(1) How do students feel about the new approach and does this differ from the traditional approach?

(2) Does the new approach influence students' decisions to study CS differently than the traditional approach?
(3) Do students using the new approach perform differently in the orientation course than those students in the traditional approach?
(4) Does the new approach with a gentle slope into programming better prepare students for the subsequent C++ classes than the traditional programming-first approach?

In the rest of the paper, we first describe the motivation and related work behind developing the ChildsPlay approach in Section 2. Then, we describe the Level 1 and Level 2 curriculum in more detail in Section 3, and we provide our research method in Section 4. In Section 5, we provide the results, and finally, we present conclusions and directions for future work in Section 6.

2 MOTIVATION AND RELATED WORK

Playing games helps develop problem-solving skills and creativity, which are fundamental to computational thinking [22, 42, 43]. Thus, it is not surprising that games have a long tradition as learning tools in education, especially in the form of gamification, which is the idea of representing a learning process as playing a game [28]. While studies have shown that playing board games improves math skills in elementary school students [8] and involves computational thinking activities [5, 6, 24, 29], simply playing games does not increase one's computational thinking skills, unless guided instruction about the skills is given [33]. Our curriculum goes beyond just playing games by teaching core CS concepts using well-known physical games for explaining computation.

Several new board and card games have been invented specifically to teach computational thinking (CT), such as RaBit EscAPE (ages 6-10), Cubetto (ages 3-6), and Crabs and Turtles [1, 40, 46], but new games present two disadvantages over focusing on well-known and familiar games. First, learning the rules of a new game can create unnecessary extraneous cognitive load on the learner, taking away cognitive resources from the learning of the computational concepts. Second, schools, kids, and families might not have access to the new games. We believe using well-known games instead broadens participation and shifts the focus to the computational concepts being taught.

The idea of using games well-known to be simple, such as Nim and Tic-Tac-Toe to explain computational concepts is not new [12, 14, 32], and researchers understand that playing games unsupported by an appropriate framework may be ineffective at teaching the computational concepts [33]. Researchers in the CS4FN and Teaching London Computing projects have shown that the use of games with well-developed lesson plans are effective for teaching specific computational concepts [13, 15], and Lee et al. show that their educational software called CTArcade enables children to articulate CT-related thinking patterns while playing Tic-Tac-Toe and Connect Four [32].

We do not use CTArcade, because we want students to play games with their peers to promote social interaction and communication, and we want students and teachers to practice concepts learned in one game by identifying them in other games, which would have to be first implemented in CTArcade. Our curriculum fits into the landscape of game-based CT teaching approaches by using familiar games, instead of new ones, and we use the same games to teach new computational concepts.

CS Unplugged [3, 4] has been shown to broaden participation [11], and several studies have demonstrated that unplugged activities, such as games, puzzles, and storytelling, can be a viable alternative to traditional programming activities for teaching introductory computational skills and algorithms [3, 16, 17, 37–39, 41, 45]. Supporting studies have shown the positive impacts unplugged activities have on students' perspectives of, engagement in, and motivation to study CS [2, 16, 18, 34, 44].

After introducing students to fundamental concepts in CS using physical games in an unplugged environment, students can then apply the concepts to programming. However, Brusilovsky et al. argue that one of the obstacles general-purpose languages pose to beginning students includes being too large and cognitively overwhelming [7]. Likewise, educators have shown success using domain-specific languages for introducing programming [30, 31]. Other researchers argue that the reduced complexity and natural relationship to familiar mathematical concepts, in addition to leveling the playing field, makes functional languages a better choice for teachers and students [9, 20, 21, 25, 27, 35], which is what spearheaded the successful Bootstrap Algebra project [10, 47]. For these reasons, we think it is important to use a functional, domain-specific language as the beginning language for students. While we understand that block-based languages have been shown to help students understand some programming concepts better [26, 36], in this project, we are specifically interested in introducing a text-based language for expressing algorithms in a formal notation.

3 OUR CHILDSPLAY APPROACH

Our approach is unique by embracing all of the following features. (1) Focusing on game rules and not strategy; (2) Connecting game descriptions to CS concepts; (3) Playing games socially to promote communication and terminology; (4) Reusing the same simple games as common threads; (5) Delaying a programming language to promote algorithmic thinking without the use of technology; (6) Employing a text-based, functional domain-specific language for programming board games. In the following subsections, we discuss the non-programming curriculum and programming curriculum in more detail.

3.1 The Non-Programming Curriculum

The goal of the non-programming part of the curriculum is to introduce CS concepts, such as representation, abstraction, algorithm, types, values, names, input, output, instruction, control structures, conditions, and computation, without the use of a computer. First, we motivate the concept that representations are a form of abstraction using stories to represent games. Then, we focus on the representations, which are the types of things and the actual values, used by the game instructions (or algorithm), and transformed during game play (or computation). Next, we focus on the concept of algorithms being a set of instructions to perform some task, and we further the concept of representation through thinking of pros and cons of using other representations in a game and how the instructions/rules of a game (*have to*) change based on representations. At this time, we also introduce the idea of placeholders

for values in algorithms and how to formalize their IF-THEN-ELSE and WHILE-DO constructs with conditions. Lastly, we explore the idea of computation and computational resources, such as time and space, by comparing games such as Nim with one heap versus several heaps and Tic-Tac-Toe versus Connect Four.

It is important to remember that the algorithms developed as part of the curriculum are *not* strategies for winning games, but rather for capturing the rules that describe games as computational processes.

3.2 The Programming Curriculum

The goal of the programming curriculum is to introduce a formal notation for algorithms within the scope of board games. To this end, we designed a new DSL targeted at the particular application domain of board games with a web interface to make it accessible by anyone on any platform with internet access (see Figure 1). More specifically, the DSL is a *domain-specific teaching language* (DSTL) created to be very small with only the basic features for teachers and students and shaped by the goal of moving students to a general-purpose language. The new DSTL for board games is primarily a functional, text-based language syntactically similar to Haskell [23] or Elm [19], but with a significantly simplified syntax and type system.

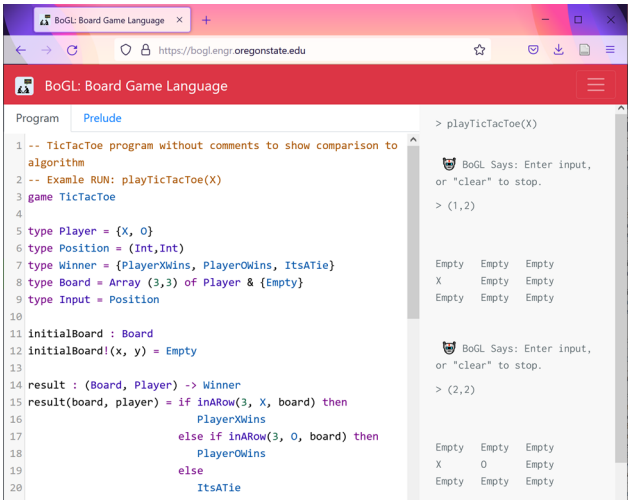

Figure 1: Screenshot of TicTacToe in the new DSTL.

Mirroring the non-programming curriculum, which begins with the concept of representation, in the programming curriculum, we introduce types and values before functions, which are the formal equivalent of algorithms in the DSTL. We start with simple functions to illustrate the use of parameters, followed by simple expressions, control structures, and conditions. The curriculum ends with introducing repetition and the concept of an array data structure to represent game boards. We also support the smooth transition from algorithmic notation to a program in the DSTL through demonstrating a systematic process during lecture to help students turn their algorithm into a program.

4 RESEARCH METHOD

To answer our research questions, we split a CS Orientation course into two sections, and the same instructor taught each section. Both sections had a 110-minute lecture, one two-hour lab, and one assignment each week. One section used the new ChildsPlay approach with a gentler slope into programming and the new DSTL for programming board games. The ChildsPlay section did not start programming until the fourth week of a 10-week quarter, and the traditional section began using the Python programming language in week 1.

Based on our prior research on delayed-programming approaches and the purpose of the new approach to offer a gentler slope into programming, we advertised the section with the new approach for those with no or little prior programming and the traditional section for those with lots of prior programming. Advisors suggested which section to take based on students self-reported prior programming experience during registration, and the course catalog reflected these differences in the notes section of the class. We acknowledge the threat to validity this causes, but it is important to remember that the ChildsPlay approach is for those with no or little experience with programming. Additionally, moving forward, all engineering students will be made to take this course, and the intention is to offer 12 sections of the course taught using different themes to better fit students' varied interests and backgrounds. This research study more accurately represents reality for our university, and we present this study as an experience report with quantitative support for its findings, rather than a scientific experimental study.

Interestingly, the two sections did not differ significantly in the number of students with and without prior programming. However, there was a difference in the amount of programming experience students had in each section. The traditional section had more students with moderate to significant experience than the other section, but many students reported selecting the section based on scheduling considerations, rather than advisor input.

The instructor in the ChildsPlay section covered the concepts of representation, algorithm, and computation without programming for the first four weeks. Beginning in week four until the end of the 10-week quarter, students used the DSTL to reinforce the non-programming concepts of types, names, values, functions, control structures with conditions, and arrays.

Since the university was online due to COVID-19, the instructor used Zoom for class lecture and labs, and used Canvas to organize all the information for the class, such as the syllabus, grades, etc. Initially, there were 150 students in the ChildsPlay section, and there were 126 students in the traditional section. Out of those students, 2 withdrew from the section, and 6 withdrew from the traditional section, which left 148 and 130 students respectively. With IRB approval, we collected consent, pre/post survey responses, and demographic information from 48 students in the ChildsPlay section and 41 in the traditional section.

5 RESULTS

In this section, we present survey results from those participating in the research study, but we provide performance information for all students taking the class. We used observational statistics to present the distribution of responses to survey questions and

grades in both sections. We used paired, two-sided, 2-sample t-tests for equality of proportions with continuity correction to compare the mean responses before and after participating in the different sections. We used unpaired two-sided, 2-sample t-tests for equality of proportions with continuity correction to compare the mean responses and grades between the two sections. For all statistical tests, we selected $\alpha = 0.05$ to be the significance level for reporting results with 95% confidence.

5.1 How do students feel about the new approach and does this differ from the traditional approach?

To answer this question, we asked students how they felt about using games to explain computation, how they felt about the new DSTL for learning programming, how much interest they had in the class before and after, and how much they overall enjoyed the approach used for their the course.

5.1.1 Do students like using games as a way to learn fundamental CS concepts? We asked students how using games as a way to teach computation affected their engagement in the class and motivation to learn more CS and programming. The majority of the students (30 out of 48) said that using games greatly or slightly increased their engagement in the class (see Figure 2). Whereas, an equal number of the students were either neutral or positive about the use of games impacting their motivation to learn more about CS or programming (see Figure 3).

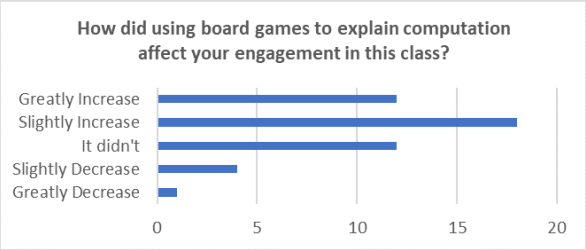

Figure 2: Impacts of Using Games on Engagement.

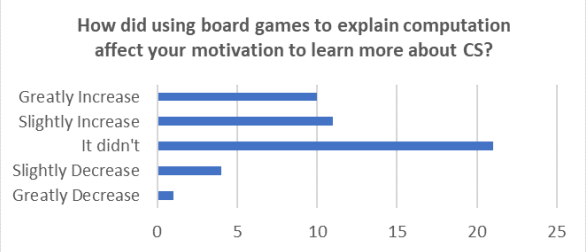

Figure 3: Impacts of Using Games on Motivation.

5.1.2 Do students enjoy programming in a new functional, text-based domain-specific board game language? Next, we asked students if they enjoyed using the new DSTL for programming board games in the class. The majority of the students either only somewhat liked the language or did not like the language at all (see Figure 4). We believe this was not necessarily a bad outcome considering the language was very new with little documentation on how to use it, except for the class notes and example programs with a tutorial being developed alongside the class.

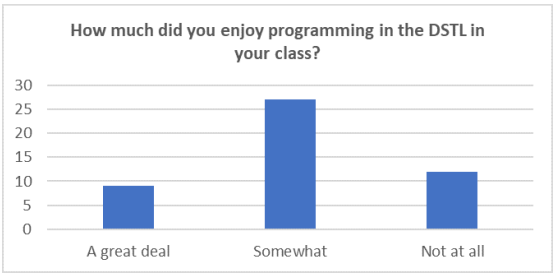

Figure 4: Student Feelings Toward Using the New DSTL.

5.1.3 What are students interest in the class before and after and is this different for the traditional section? We asked students how interested they are in the class before and after taking the CS orientation class and how much they enjoyed the overall approach in their class. We compared these results from the two sections to see if there was a difference in the way students felt about their class in the ChildsPlay versus the traditional section. As Figure 5 shows, unfortunately, there was a significant decline in students' interests in the ChildsPlay section after taking the class (p-value = 0.005757). Whereas, Figure 6 show that there was no difference in the mean response to students' interests before and after taking the traditional section (p-value = 0.2004).

Figure 5: Student Class Interests in ChildsPlay Section.

In addition, we asked students how they felt about the approach used to teach their class, and there was a significant difference between the way the students felt about their approach used in the ChildsPlay versus traditional section (p-value = 0.01593). Students in the traditional section liked the approach used in their class more than those in the ChildsPlay section (see Figure 7). The majority of the students in the ChildsPlay section only somewhat liked the approach used in their class; whereas, the Python section had more students who liked the approach a great deal.

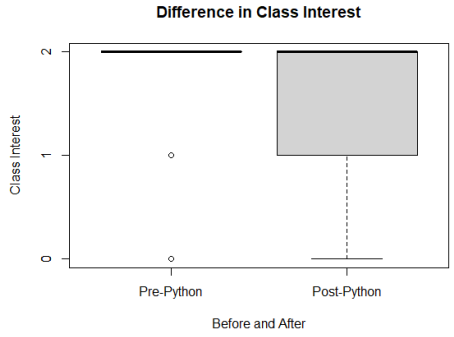

Figure 6: Student Class Interests in Traditional Section.

Figure 8: Student Job Interest in ChildsPlay Section.

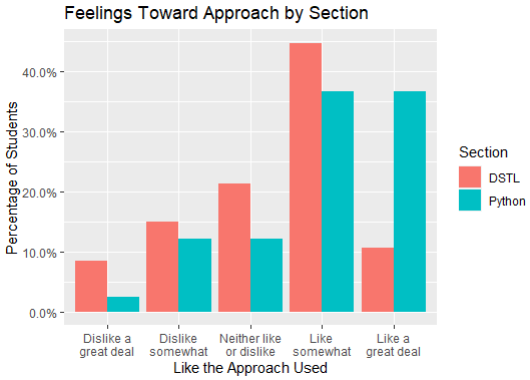

Figure 7: Student Feelings Toward Approach Used for Class.

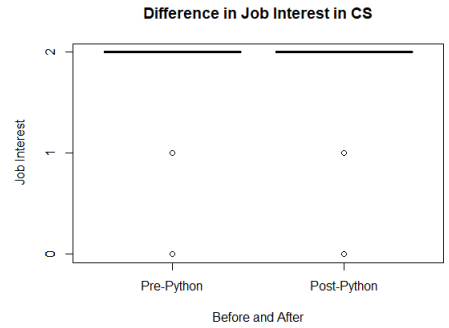

Figure 9: Student Job Interest in Traditional Section.

5.2 Does the new approach influence students' decisions to study CS differently than the traditional approach?

We asked students about their pre- and post-interests in 1) learning more about CS, 2) majoring in CS, 3) taking more CS classes, 4) programming, and 5) using computation in their job after college. The first four interests did not significantly change in the ChildsPlay section or in the traditional Python section. However, students' interest in using computation in a job after college did change before and after taking the ChildsPlay section (p-value = 0.01926). More students were extremely interested in using computation in their job before taking the ChildsPlay section than after taking the class (see Figure 8). In contrast, mean student interests in computational jobs before or after taking the traditional Python section remained exactly the same (see Figure 9).

5.3 Do students using the new approach perform differently in the orientation course than those students in the traditional approach?

In order to answer this question, we used the grades and drop, fail, withdraw rates from all students in the class, not just those who consented to the research study. We did not need consent for the course-level data from the registrar, because our IRB determined that existing, de-identified course-level data with unique ids to follow students across courses did not involve human subjects according to the Dept. of Health and Human Services.

We did not find a significant difference in the performance of students in the ChildsPlay section versus the traditional section of the CS Orientation course (see Figure 10). The majority of the students in both sections got an A in the class, and approximately 8% in each class did not make a C or above, which was the grade required to pass the class. At our university, all CS classes require a C or above to count toward the CS degree. There were more students who withdrew from the traditional Python section (6 out of 126) than the ChildsPlay section (2 out of 150) with a gentler slope into programming. From this analysis, it appears that the gentler slope retained more students in the orientation class.

We found that 135 out of 150 students enrolled in the ChildsPlay section passed (90% pass rate), and 110 out of 126 students enrolled in the Python section passed (~87% pass rate). We found no evidence of a difference between the two pass rates (p-value = 0.606).

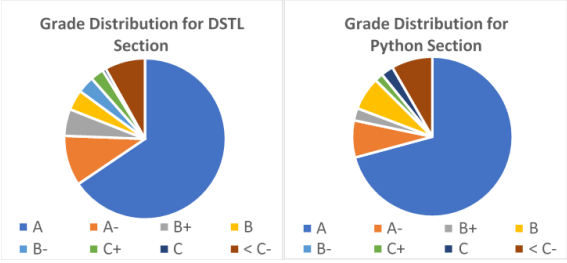

Figure 10: Student Grades in the CS Orientation Sections.

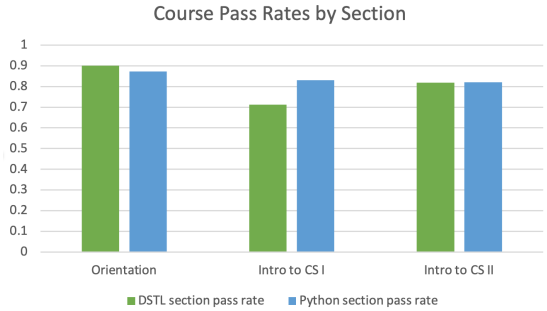

Figure 11: Student Pass Rates in Subsequent CS Classes

5.4 Does the new approach with a gentle slope into programming better prepare students for the subsequent C++ classes than the traditional programming-first approach?

We determined if there was a difference in the way students from the ChildsPlay versus traditional section performed in their subsequent Introduction to Programming I and II courses by collecting grade information from each subsequent course. Using this information, we determined pass rates for each group of students (DSTL and Python) in each of the subsequent courses (Intro to Programming I and II). We then compared these rates for differences between the two student groups in each class. We categorized passing grades as letter grades that are a C or higher, and we categorize non-passing grades to include anything less than a C, withdrawals, incompletes, and S/U (satisfactory/unsatisfactory option) grades. This categorization was consistent with the college's policy on passing grades for CS courses for CS majors.

We found that 72 out of the 101 students from the ChildsPlay section enrolled in the Introduction to Programming I in the winter passed (~71% pass rate), while 78 out of 94 students from the Python section passed the same course (~83% pass rate) (see Figure 11). We found weak evidence of a difference between these two pass rates (p-value ≈ 0.077), but the pass rate of the students from the ChildsPlay section was lower than the pass rate of the students from the Python section. This difference could be because 15 students from the ChildsPlay section withdrew, as opposed to only one from the Python section. Counts of other grades were not drastically different. We expected that the students from the ChildsPlay section would be better prepared for the first subsequent CS course and have a higher pass rate than the students from the Python section.

In the second subsequent class, we found that 54 out of 66 students from the ChildsPlay section, who enrolled in the Introduction to Programming II in the spring, passed (~82% pass rate), and 64 out of 78 students from the Python section passed (~82% pass rate) (see Figure 11). These results showed no difference in students' success in the second course depending on the approach used in the orientation course.

6 CONCLUSION

From this research study, we conclude that most university students found using board games in a CS orientation course to be engaging and that many also liked using board games as way to learn more about CS and programming. However, most university students only somewhat enjoyed programming in a new DSTL in their orientation course. Some students stated that this was because they liked other languages based on what they already knew or what their peers were learning in the other section. We believe that some students influenced their peers' views in the class. Some students commented on disliking the language because there was little documentation or resources, including TAs, to help them or that the error messages were unclear. In the future, this is something to consider before using a new language with 150 students.

While students' interests in the class and using computation in a job after college was negatively impacted by the approach using a new DSTL, the new approach did not negatively impact any of their other interests in CS. Therefore, we do not see this as a primary concern for adopting the approach with first-year university students. In fact, most students somewhat liked the new approach or liked the new approach a great deal, and we found no correlation between their prior programming experience and how they felt about the approach. However, in the future, we would like to determine the type of student this approach is best for.

Even though student grades and drop, fail, withdraw was not different between the new approach and the traditional approach, it did appear that there was a slight difference in students performance in the first subsequent C++ class but not in the second one. We understand that some of these differences might be attributed to the way we recruited students to the different sections, but based on students' survey responses and prior programming experience, we do not believe this was a contributing factor.

We recognize that we did not control for the increase in familywise error rate across the series of reported statistical tests. In other words, there is a likelihood that we incorrectly rejected a null hypothesis in this study. In addition, we recognize the smaller number of traditional section responses could also contributed to the acceptance of a null hypothesis. However, we consider this research to be an experience report with support for its findings and encourage replication.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under the grant #1923628.

REFERENCES

[1] P. Apostolellis, M. Stewart, C. Frisina, and D. Kafura. 2014. RaBit EscAPE: A Board Game for Computational Thinking.. In *Confón Interaction Design and Children*. 349–352.

[2] T. Bell, P. Curzon, Q. I. Cutts, V. Dagiene, and B. Haberman. 2011. Overcoming Obstacles to CS Education by Using Non-Programming Outreach Programmes. In *Int. Conf. on Informatics in Schools (LNCS 7013)*. 71–81.

[3] T. Bell, I. H. Witten, and M. Fellows. 2015. *CS Unplugged. An Enrichment and Extension Programme for Primary-Aged Students.*

[4] T. C. Bell, I. H. Witten, and M. Fellows. 1998. *Computer Science Unplugged: Off-line Activities and Games for All Ages.* Computer Science Unplugged.

[5] M. Berland and S. Duncan. 2016. Computational Thinking in the Wild: Uncovering Complex Collaborative Thinking through Gameplay. *Educational Technology* 56, 3 (2016), 29–35.

[6] M. Berland and V. R. Lee. 2011. Collaborative Strategic Board Games as a Site for Distributed Computational Thinking. *Int. Journal of Game-Based Learning* 1, 2 (2011), 65–81.

[7] Peter Brusilovsky, Eduardo Calabrese, Jozef Hvorecky, Anatoly Kouchnirenko, and Philip Miller. 1997. Mini-languages: a way to learn programming principles. *Education and Information Technologies* 2, 1 (1997), 65–83. https://doi.org/10.1023/A:1018636507883

[8] S. Cavanagh. 2008. Playing Games in Class Helps Students Grasp Math. *Education Digest: Essential Readings Condensed for Quick Review* 3 (2008), 43–46.

[9] Manuel M. T. Chakravarty and Gabriele Keller. [n.d.]. The risks and benefits of teaching purely functional programming in first year. 14, 1 ([n. d.]), 113–123. https://doi.org/10.1017/S0956796803004805

[10] Bootstrap Community. [n.d.]. Bootstrap. https://bootstrapworld.org/materials/algebra/

[11] T. J. Cortina. 2015. Reaching a Broader Population of Students Through "Unplugged" Activities. *Commun. ACM* 58, 3 (2015), 25–27.

[12] CS For Fun: Queen Mary, University of London. 2011. Noughts and Crosses. http://www.cs4fn.org/programming/noughtscrosses/. Accessed: 2021-01-07.

[13] CS For Fun: Queen Mary, University of London. 2011. Welcome to cs4fn : the fun side of Computer Science. http://www.cs4fn.org/. Accessed: 2021-01-07.

[14] CS For Fun: Queen Mary, University of London. 2011. Winning at Nim: computers outwitting humans. http://www.cs4fn.org/binary/nim/nim.php. Accessed: 2021-01-07.

[15] CS For Fun: Queen Mary, University of London. 2015. Teaching London Computing: A Resource Hub from CAS London & CS4FN. https://teachinglondoncomputing.org/. Accessed: 2021-01-07.

[16] Paul Curzon, Peter W. McOwan, Nicola Plant, and Laura R. Meagher. 2014. Introducing teachers to computational thinking using unplugged storytelling. *In Proceedings of the 9th Workshop in Primary and Secondary Computing Education (WiPSCE '14)*, 89–92.

[17] Q. Cutts, Q. Connor, G. Michaelson, and P. Donaldson. 2014. Code or (not code): separating formal and natural language in CS education. *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*, 20–28.

[18] Q. I. Cutts, M. I. Brown, L. Kemp, and C. Matheson. 2007. Enthusing and informing potential computer science students and their teachers. In *SIGCSE Conf. on Innovation and Technology in Computer Science*. 196–200.

[19] Elm. 2021. A delightful language for reliable web applications. https://elm-lang.org. Accessed: 2021-08-11.

[20] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. [n.d.]. The structure and interpretation of the computer science curriculum. 14, 4 ([n. d.]), 365–378. https://doi.org/10.1017/S0956796804005076

[21] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. [n.d.]. The TeachScheme! Project: Computing and Programming for Every Student. *Computer Science Education* 14, 1 ([n. d.]), 55–77. https://doi.org/10.1076/csed.14.1.55.23499

[22] C. Harris. 2009. Meet the New School Board: Board Games Are Back–And They're Exactly What Your Curriculum Needs. *School Library Journal* 5 (2009), 24–26.

[23] Haskell. 2019. An advanced, purely functional programming language. https://www.haskell.org. Accessed: 2020-08-24.

[24] N. R. Holbert and U. Wilensky. 2011. Racing games for exploring kinematics: a computational thinking approach. *7th Int.l Conf. on Games + Learning + Society*, 109–118.

[25] John Hughes. [n.d.]. Experiences from teaching functional programming at Chalmers. 43, 11 ([n. d.]), 77–80. https://doi.org/10.1145/1480828.1480845

[26] Niklas Humble. 2021. The use of Programming Tools in Teaching and Learning Material by K-12 Teachers. https://doi.org/10.34190/EEL.21.117

[27] Stef Joosten, Klaas Van Den Berg, and Gerrit Van Der Hoeven. [n.d.]. Teaching functional programming to first-year students. 3, 1 ([n. d.]), 49–65. https://doi.org/10.1017/S0956796800000599

[28] K. M. Kapp. 2012. *The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education.* Pfeiffer.

[29] C. Kazimoglu, M. Kiernan, L. Bacon, and L. MacKinnon. 2012. Learning programming at the computational thinking level via digital game-play. *Procedia Computer Science* 9 (2012), 522–531.

[30] Tomaž Kosar, Marjan Mernik, and Jeffrey C. Carver. 2012. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. *Empirical Software Engineering* 17, 3 (2012), 276–304.

[31] Tomaž Kosar, Nuno Oliveira, Marjan Mernik, Matej Pereira, Črepinšek, Daniela Da Cruz, and Pedro Henriques Rangel. 2010. Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems* 7, 2 (2010), 247–264.

[32] T. Y. Lee, M. L. Mauriello, J. Ahn, and B. B. Bederson. 2014. CTArcade: Computational Thinking with Games in School Age Children. *Int. Journal of Child-Computer Interaction* 2, 1 (2014), 26–33.

[33] T. Y. Lee, M. L. Mauriello, J. Ingraham, A. Sopan, J. Ahn, and B. B. Bederson. 2012. CTArcade: Learning Computational Thinking Thile Training Virtual Characters Through Game Play. In *Human Factors in Computing Systems*. 2309–2314.

[34] C. Mano, V. Allan, and D. Cooley. 2010. Effective In-Class Activities for Middle School Outreach Programs. In *Annual Conf. on Frontiers in Education*. F2E–1–F2E–6.

[35] J. Margolis and A. Fisher. 2003. *Unlocking the Clubhouse: Women in Computing.* MIT Press, Cambridge, MA.

[36] Monika Mladenovi, Saa Mladenovi, and Žana Žanko. 2020. Impact of used programming language for K-12 students' understanding of the loop concept. *International Journal of Technology Enhanced Learning* 12 (2020), 79–98.

[37] J. Parham-Mocello, S. Ernst, M. Erwig, E. Dominguez, and L. Shellhammer. 2019. Story Programming: Explaining Computer Science Before Coding. In *ACM SIGCSE Symp. on Computer Science Education*. 379–385.

[38] J. Parham-Mocello and M. Erwig. 2020. Does Story Programming Prepare for Coding?. In *ACM SIGCSE Symp. on Computer Science Education*. 100–106.

[39] J. Parham-Mocello, M. Erwig, and E. Dominguez. 2019. To Code or Not to Code? Programming in Introductory CS Courses. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*. 187–191.

[40] Primo. 2018. Cubetto: Screenless Coding Toy for Girls and Boys Aged 3-6. https://www.primotoys.com.

[41] Primo. 2020. Free beginner's guide to Coding with Kids. https://www.primotoys.com/guide-coding-for-kids-ebook/. Accessed: 2021-01-07.

[42] C. Ragatz and Z. Ragatz. 2018. Tabletop Games in a Digital World. *Parenting for High Potential* 7 (2018), 16–19.

[43] L. A. Sharp. 2012. Stealth Learning: Unexpected Learning Opportunities Through Games. *Journal of Instructional Research* 1 (2012), 42–48.

[44] R. Taub, M. Ben-Ari, and M. Armoni. 2009. The Effect of CS Unplugged on Middle-School Students' Views of CS. In *SIGCSE Conf. on Innovation and Technology in Computer Science*. 99–103.

[45] R. Thies and J. Vahrenhold. 2013. On Plugging "Unplugged" Into CS Classes. 365–370.

[46] K. Tsarava, K. Moeller, and M. Ninaus. 2018. Training Computational Thinking Through Board Games: The case of Crabs Turtles. *Int. Journal of Serious Games* 5, 2 (2018), 25–44.

[47] Geoff Wright, Peter J. Rich, and Robert Lee. 2013. The Influence of Teaching Programming on Learning Mathematics.