Analogies and Active Engagement: Introducing Computer Science

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

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

Margaret Niess
niessm@oregonstate.edu
Oregon State University
Corvallis, Oregon, USA

ABSTRACT

We describe a new introductory CS curriculum initiative that uses analogies and active engagement to develop students' conceptual understanding before applying the concepts to programming. We believe that traditional coding approaches to introducing computer science concepts rely on students to build their own conceptual understanding, rather than grounding their understanding of concepts in what they know from everyday experiences. Using constructivism as a foundation for this curriculum initiative, our approach builds a framework for student understanding anchored in the physical world using simple games and stories to stimulate mental engagement through embodied learning.

For example, we teach the concept of abstraction and representation by presenting the game of Tic-Tac-Toe as an island divided into nine regions, but the middle one you cannot get to by boat, which is the way two teams arrive to the island. After playing the game once and realizing the game is really just Tic-Tac-Toe, the students understand the example is a representation with modified rules and game pieces. Then we talk about how the set of rules for a simple game like Tic-Tac-Toe is an algorithm with instructions for how to play the game, and we use playing the game to explain computation as the execution of an algorithm.

Based on observations using analogies and active engagement in 6th grade classrooms, we provide many examples explaining how this curriculum initiative is an engaging, effective, and flexible approach for introducing CS concepts.

CCS CONCEPTS

• **Applied computing** → **Interactive learning environments**; **Collaborative learning**; • **Social and professional topics** → **K-12 education**; **Computational thinking**.

KEYWORDS

constructivism, games, stories, unplugged, embodied learning

ACM Reference Format:
Jennifer Parham-Mocello, Martin Erwig, and Margaret Niess. 2024. Analogies and Active Engagement: Introducing Computer Science. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630777

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM ISBN 979-8-4007-0423-9/24/03…$15.00
https://doi.org/10.1145/3626252.3630777

1 INTRODUCTION

We discuss a curriculum initiative for introducing computer science (CS) centered on using analogies and active engagement to convey basic computing concepts. We define CS as a discipline requiring computational thinking to study the foundation of computing and all related concepts, and we use non-programming computational thinking activities and examples to illustrate four, fundamental CS concepts: *abstraction*, *representation*, *algorithm*, and *computation*.

One major goal of our approach was to debunk negative perceptions that CS is socially isolating, lacks creativity or fun, and is better suited for white, male students [9, 19]. We demonstrated that learning basic concepts of CS is as fun, social, and gender-neutral as reading a story or playing a simple non-electronic game, and we used constructivism as a foundation for our curriculum initiative.

We aimed to create an accessible and adoptable CS curriculum initiative that built a framework for student understanding anchored in the physical world using simple games and stories to stimulate mental engagement that was meaningful to teachers and students in the 6th grade. Our approach was inspired by previous work of ours and others that introduced CS concepts before programming using unplugged approaches [2, 10, 15, 16, 29, 48], simple stories [7, 18, 26, 34, 34–36] and simple, physical games [6, 11, 13, 17, 27, 32, 33, 37–40].

2 MOTIVATION AND RELATED WORK

The constructivism learning theory was fundamental to our curriculum initiative. We used the idea of "unplugged" computational thinking to actively engage students in the cognitive and social construction of computer science knowledge, and we used simple stories and games as analogies for explaining computing concepts.

2.1 Constructivism

Constructivism is a well-known theoretical framework for learning in K-12 education grounded in the construction of new knowledge based on one's prior knowledge and experiences [31]. Cognitive constructivism was based on Piaget's stages of cognitive development among children [42], and Lev Vygotsky's social constructivism was based on development occurring when people interact within cultures and societies [50]. Our curriculum initiative incorporated both cognitive and social constructivism through simple, unplugged stories and games to introduce CS concepts.

2.2 Unplugged Computational Thinking

Many approaches introducing computer science to students are predicated on programming and require an understanding of how to code an algorithm in a programming language. Efforts like code.org use programming/coding to promote computer science to young children. However, many studies showed that comfort level and

willingness to learn programming were the strongest predictors of success in programming classes [45, 46, 51]. Since computer science is not synonymous with programming, there is no inherent necessity to tie the orientation to computer science to coding activities.

Students curious about computer science should not be excluded because they are reluctant to the idea of having to learn programming as a prerequisite to understanding computer science. This was why efforts to explain computer science without a computer, such as csunplugged.org [3, 4], gained popularity, especially among the K-12 community [2, 15, 16, 29, 48], and studies showed that the unplugged approach broadened participation [10]. For these reasons, we believed that 6th grade students and teachers could benefit from a non-coding alternative using everyday understanding of the world as a way to introduce basic fundamental CS concepts.

2.3 Stories

The use of stories to explain computing is not new. Computational Fairy Tales described algorithms and data structures as part of a story about a princess on a quest to save her father's kingdom [26]. The target audience was middle school children. Lauren Ipsum [7] employed a similar approach telling a story about a girl who got lost in a forest and wanted to find her way back home. In her adventure, she had to solve several problems, which served as a hook to introduce concepts of algorithms and math on a very high level. The target audience was also middle school children, and the story was like Alice in Wonderland with its playful and clever use of names. It contained an appendix that provided additional explanations of the concepts mentioned in the story.

At the university level, one study used Computational Fairy Tales to help the retention and academic performance of computer science majors, mostly aimed at students with little to no programming experience [30]. It found that "CS0 students without prior programming experience got significantly higher grades in CS1 than CS0 students who had programmed before"; the students were split on how useful the book was to their learning. Another study determined that Story Programming was a viable alternative to the traditional programming-focused approach for teaching a computer science orientation class [34, 35], and students in another study argued that computer science or programming concepts could be explained effectively using stories if the connection between concept and story was strong [23]. Similarly, one study claimed that using a story to learn a concept was easily accessible because stories helped many people learn in general [22], and another study showed that "unplugged" activities involving storytelling was useful for introducing teachers to computational thinking [15]. For these many reasons, we believed stories provided a viable means for conveying CS concepts to 6th grade students and teachers with any background in computation.

2.4 Games

Likewise, playing games develops problem-solving skills and creativity, which are fundamental to computational thinking [20, 44, 47]. 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 [24]. While studies showed that playing board games improved math skills in elementary school students [8] and involved computational thinking activities [5, 6, 21, 25], simply playing games did not increase one's computational thinking skills, unless guided instruction about the skills was given [28].

The idea of using simple, existing physical games to explain computational concepts is not new [11, 13, 27], and researchers understand that playing games unsupported by an appropriate framework may be ineffective at teaching the computational concepts [28]. Researchers in the CS4FN and Teaching London Computing projects showed that the use of games with well-developed lesson plans were effective for teaching specific computational concepts [12, 14], and Lee et al. showed that their educational software called CTArcade enabled children to articulate computational thinking patterns while playing Tic-Tac-Toe and Connect Four [27].

However, CTArcade was software, and it was not flexible with the implementation of games. We wanted students to play games in an unplugged environment with their peers to promote social interaction and communication, as well as practice concepts learned in one game by identifying them in other games. While we recognized that several new board and card games were invented to teach computational thinking, such as RaBit EscAPE (ages 6-10), Cubetto (ages 3-6), and Crabs and Turtles (ages 8-9) [1, 43, 49], these games did not come with lesson plans for teachers or research studies on whether students actually learned the computational concepts. Additionally, schools, kids, and families did not have access to the new games. Therefore, a major goal for this curriculum initiative was to use and create simple, physical games that anyone could access for teaching CS concepts.

3 OUR CURRICULUM INITIATIVE

While the use of stories or games to teach CS is not new, our curriculum initiative combining stories and games with the use of analogies and active engagement without a machine or programming to teach 6th grade students is new. We created the curriculum grounded in constructivism to introduce four basic CS concepts: *representation*, *abstraction*, *algorithm*, and *computation*. Our curriculum went beyond just creating stories or playing games by teaching four core CS concepts using stories of simple, physical games.

One key contribution to the development of the curriculum was the active research to make continuous improvements based on observations, interviews, and survey information from teachers and students. In the following sections, we provide examples of how we used a research practice partnership to develop our curriculum initiative.

3.1 A Research Practice Partnership

For our curriculum initiative we used a research practice partnership (RPP) between researchers from education and computer science departments, two middle school math teachers (with no background in computer science), and the school principal to iteratively develop our ideas and curricular material. The RPP was critical for creating lesson plans that were useful to the teachers with objectives, content/subject matter, learning experiences, and assessments and teaching material that was age-appropriate, inclusive, and scaffolded for students (see Table 1).

Table 1: Example Lesson Plan for Representation.

Lesson Overview		
Total Time: 36 minutes **Lesson Materials:** Lesson Overview (this document), PowerPoint slides, student worksheet, Students' homemade representations of the players for the Treasure Hunt game. **Focused Topics**: Benefits and disadvantages of representations **Summary:** Students revisit the Treasure Hunt game, this time viewing different representations of the game's components, such as using shapes instead of ropes and a single line instead of a grid for a board. Students will have to analyze the new representations to figure out how the game rules change with different representations. **Student Learning Objectives (SLOs) (assessment provided at end of unit):** Students will work on learning how to… • SLO 1: List advantages, disadvantages, and differences for various representations of game objects. • SLO 2: Understand how rules change to accommodate different representations.		
Lesson Details	**Teacher (T)**	**Student (S)**
Time: 5-10 min **Purpose:** SLO 1. Summarize ideas learned from Days 1 and 2 to refresh Ss understanding.	T has Ss place their different team characters on the classroom display as they come into the classroom (get them out of their hands!) T reminds Ss that these characters are representations for pieces of the game. Then T reminds them of some of the other representations they identified for this game, like ropes for in a row. Revisit from Day 2: Can you describe the different **representations** in this story or game and the details that were omitted by the **representation**? Today we are going to think about some of these representations and see if the game rules must change.	Ss place their team characters on the classroom display On another display are the Ss ideas for representations from yesterday such as story, island representation, ropes, symbols for ropes.

For example, through the RPP we identified how to change the language in a story we created to be more inclusive, and we learned that we needed to use technical vocabulary in more than 10 different ways to reinforce and scaffold student and teacher understanding. Additionally, we learned that a teacher's knowledge for teaching CS needs to go beyond the material they are teaching, and a lesson plan should contain the objectives for the students and the teachers, learning activities for engaging the students, subject matter content knowledge for the teachers and students, and assessment information. Each lesson had a review at the beginning.

We used the same example in multiple different analogies and activities to reinforce a concept or teach new concepts without adding extrinsic cognitive load to the learner. For example, we created a story of two teams looking for a treasure on an island divided into nine regions, representing the game of Tic-Tac-Toe in many different analogies and activities to introduce and reinforce each of the four concepts.

3.2 Analogies and Active Engagement

The goal was to use analogies and active engagement with stories of simple, physical games and the human as the computer to introduce the CS concepts of representation, abstraction, algorithm, and computation (see Table 2). We wanted to leverage what students were already familiar with, eliminate the distraction of a machine, and level the playing field between those with or without prior programming experience or who identify with digital technology.

Table 2: Examples of Analogies and Active Engagement for CS Concepts.

Stories and Games		
CS Concept Order	**Analogies**	**Active Engagement**
Abstraction and Representation	Story Characters Game Pieces Stories for Games	*Story Representation* *Floor Activity* *Read Story/Play Game* *Change Story/Game* *Formalize Instructions* *Classroom Poster*
Algorithm (correctness and complexity)	Text for Stories Rules for Games	
Computation (execution time)	Reading a Story Playing a Game	

We used the same stories and games but with different analogies and active engagement to introduce and reinforce all four basic CS concepts in our curriculum initiative (see Table 2). We began with abstraction and representation before algorithms, since algorithms use abstraction and representations and are themselves examples of abstraction and representation. Similarly, the concept of an algorithm was discussed before computation, since computation requires executing an algorithm. The analogies were specific to the concept being conveyed, but the same active engagement used different analogies and concepts.

We created stories for the games of Tic-Tac-Toe, tossing a coin to see who goes first, Nim, and a new game called MoveIt! to introduce, reinforce, and scaffold student learning of all four basic concepts in our curriculum, and we used many analogies and active engagement with the stories and games for students to further construct their conceptual understanding. We found that this initiative created a foundation for the teachers to easily incorporate other stories and games that they and their students knew.

In the following subsections, we explain some of the analogies and active engagement that we used to teach the four basic CS concepts and the reasons why we used them.

3.2.1 Story Representation (of Anything). We *motivated* the concepts of representation (an entity that stands for something else), abstraction (the process of omitting detail), algorithm (a sequence of instructions in a language understood by a computer (which isn't necessarily a machine)), and computation (when a computer executes an algorithm) using story representations for Tic-Tac-Toe, Tossing a Coin, and Nim (see Figures 1, 2, and 3).

For Tic-Tac-Toe, we presented the game as a story of two teams looking for a treasure on an island divided into nine sections. The teams arrived at the island by boat, which means they could not start in the middle section (see Figure 1). The team members took turns sending team members to different sections of the island, and the winning condition for a treasure was when a team had three team members on the same rope. This allowed us to motivate the introduction of the concept of abstraction and representation with little cognitive load and later use the same story representation in other activities to convey different concepts in multiple contexts.

For the game of tossing a coin, we presented tossing a coin as a way to win a dispute over which sibling, Rosa and Jack, would have to do the dishes (see Figure 2). We used the story and game to motivate reinforcing abstraction and representation and to introduce

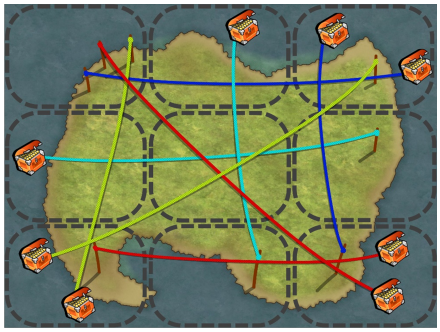
if-then-else control structures in algorithms.

Lesson 1 Worksheet:

Let's suppose, there are two teams of people searching to find treasure chests buried on multiple islands. Both teams want to be the first to find a treasure chest! Each island has its own map and directions for retrieving the treasure chests.

This is the map of the first island the teams will encounter in their quest for treasure chests. The map shows the island with its treasure chests buried in the ocean off the island shore.

The island is marked into nine sections, and each section can only be occupied by one member of a team at a time. Since there is nowhere to land a plane on the island, the teams arrive by boat, and each team chooses a different section of the island for their boat to land.



In addition to the island being marked into different sections, the island is covered with ropes traversing different parts of the island and connected to the treasure chests buried in the ocean.

The ropes traversing the island cross different sections, and the treasure chests at the end of the ropes are very heavy requiring 3 members on a team to pull out the treasure from the ocean. A team must place three members in different sections for pulling on the same rope, and the teams must take turns in claiming specific sections on the island.

Figure 1: Story Representation of Tic-Tac-Toe

Figure 2: Story Representation of Tossing a Coin

For the game of Nim, which involves people taking turns selecting between one and three objects from a heap of objects until someone either takes or is left with taking the last object(s), we presented the game as siblings, Rosa and Jack, who knock over a stack of boxes into a pile (or heap) onto the garage floor (see Figure 3). We used the story and game to once again motivate the concepts of abstraction and representation and introduced the looping control structure that can impact the time it take to run (or compute) the algorithm.

Even though this curriculum initiative created stories representing games to teach about basic CS concepts, stories about *an activity*, like getting ready in the morning or moving items to homes for

Figure 3: Story Representation of Nim

disaster relief, or *a well-known fictional event/tale*, like Hansel and Gretel leaving bread crumbs to mark their trail, could also be used to teach basic CS concepts. We correlated the categories/kinds of things (types) and the actual things (values) that were in stories, games, activities, events, etc. to representations and abstractions. Students got to play with and think about different representations in the story, game, or activity to motivate the appropriate choice of a representation that is abstract enough to omit unnecessary detail but remains easily distinguishable, such Xs and Os versus pictures of team members.

3.2.2 Floor Activity. We *engaged* students with the concepts of abstraction, representation, algorithm, and computation using a floor-sized version of the treasure hunt representation of Tic-Tac-Toe (see Figures 4). This was an excellent way to engage the entire class, and the students did not want to quit playing as a class and break into small groups. The students also saw the importance of representation when they played the story version of Tic-Tac-Toe and had to remember who was on which team (see Figure 4). The picture in Figure 4 was taken before we changed the language from diamonds to treasure chests to be more inclusive.

Figure 4: Floor Game of Tic-Tac-Toe

In another game we created, called MoveIt! (see Section 3.2.5), students liked watching their peers create and perform the instructions for moving a set of items from an initial configuration to a target configuration in front of the class. The floor activity engaged the class in acting as a compiler and debugger to find issues with the instructions given by the student instructing another student's

moves for moving the items, such as executing a move that would require going outside the boundaries of the board.

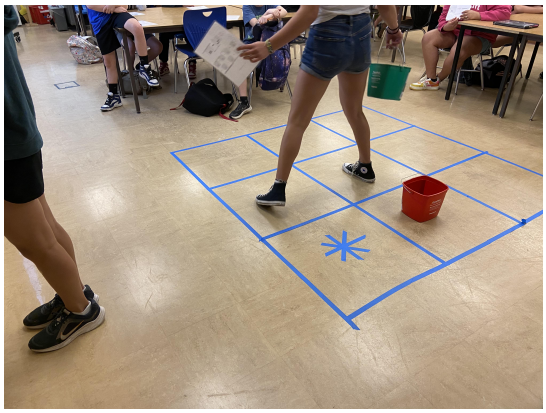


Figure 5: MoveIt! Floor Version

3.2.3 Read Story/Play Game. Concepts were *introduced and reinforced* given simple stories and games to read and play as analogies and active engagement for learning about CS. We related the characters, scenes, pieces, and boards in stories and games to abstractions and representations, and we related events and rules to sequential instructions and control structures that manipulate representations in algorithms.

3.2.4 Change Story/Game. We asked students to change story or game elements to *reinforce* the four basic concepts. For example, students were asked to change the Tic-Tac-Toe board to a line with nine locations, rather than a 3×3 grid (see Figure 6). We asked students if they were able to use the ropes from the treasure hunt story to represent the same winning condition on the line.

Figure 6: Change Tic-Tac-Toe Board Representation

Then, to motivate explicitly stating what three in a row is and the use of "and" and "or" in the condition, we asked students how they would change the winning condition to represent all possible locations where three team members can be on the island to win a treasure, and if the same winning condition could be used with the 3×3 grid. We also presented students with a line containing colored shapes and asked students how the winning condition changed (see Figure 6). This reinforced learning the concept of abstraction and representation, and these activities taught students that the chosen representations and the algorithm were closely connected.

3.2.5 Formalizing Instructions. Beyond motivating and introducing the concept of an algorithm through story representations and floor activities, we *reinforced* the four basic CS concepts through asking students to formalize instructions for a game, activity, etc. To help scaffold students' learning of formalizing instructions, we created a new game called MoveIt! that was based on moving items on a grid from an initial to a target configuration given a limited and extended language that provided the ability to move up, down, right, left, pickup an item, and drop an item (see Figure 7). We created a story about the items representing rocks from a landslide that needed to be moved, but the items in the game could represent anything that would need to be moved for some reason, like disaster relief boxes that need to be delivered to homes.

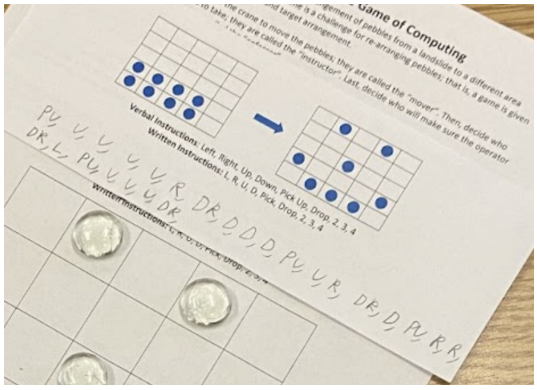


Figure 7: A New MoveIt! Game

We found that having students verbalize the instructions one at a time helped motivate the need to write down instructions to share with others or detect errors. Students liked playing the game, especially when introduced as a floor activity, and the students really liked being given increasingly difficult tasks to solve or creating their own tasks for other students to solve.

Additionally, since past research showed Parsons Problems to be an effective way to teach programming [41], we created Parson Problems for English-like algorithms with hints and blank lines for the students (see Figure 8). However, the students did not like filling in the blanks with all the words and would rather point out which instructions went where.

Therefore, we created instruction pieces for students to arrange for formally expressing their algorithms without any writing (see Figure 9) or minimal writing (see Figure 10). Students liked the instruction pieces to arrange, but larger, more complex algorithms, such as Rock, Paper, Scissors, were confusing without scaffolding from the teacher. The teacher provided the outline of the algorithm and let the students figure out what the input, conditions, and output were. However, the instruction pieces limited the students to the construction of a pre-designed algorithm or something very similar, unless students were provided with an overwhelming number of pieces that they may or may not use. Even though the instruction pieces could not be used to write any algorithm for any story or game, students liked physically putting together the instruction pieces better than writing instructions.

Below are pieces of the algorithm for playing the coin toss game.

- Rosa tosses the coin
- Rosa does the dishes
- Jack calls Heads or Tails
- the coin toss is equal to the call
- Jack does the dishes

Write each phrase in the space where it belongs in the algorithm.

_____

_____

IF _____ **THEN**

_____

ELSE

_____

Figure 8: Worksheet with Algorithm as Parsons Problem

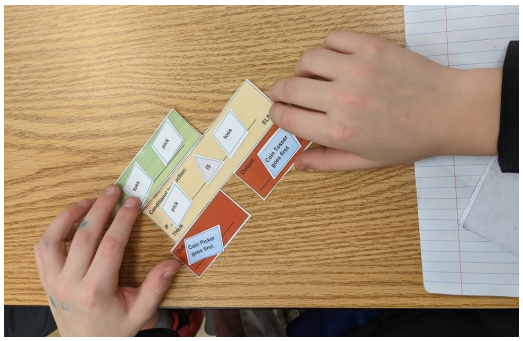


Figure 9: Basic Puzzle Solution for Coin Toss

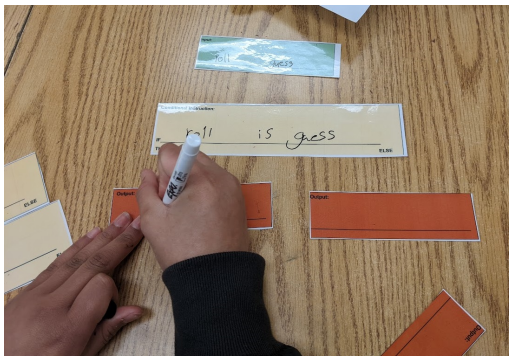


Figure 10: Basic Puzzle Solution for Coin Toss

3.2.6 Classroom Poster. We *reinforced* the concepts of abstraction, representation, algorithm, and computation using a large classroom-sized poster hung on the wall. After learning about the concepts in the curriculum, students were supposed to place a label for where concepts appeared on the poster. For example, on the poster in Figure 11, students were expected to put a label for algorithm on the game instructions and a label for computation at the bottom with the people playing the game. The poster served as a way to reinforce concepts by continuously being viewed and discussed in the class, and students outside the CS class could view the poster to help broaden participation.

Figure 11: Computing explained through games.

4 CONCLUSIONS

This curriculum initiative provided a foundation for combining stories and games to teach about four fundamental CS concepts. We used stories and games for analogies and active engagement, and we provided stories and games to motivate, introduce, and reinforce the basic CS concepts of abstraction, representation, algorithm, and computation. We found that the 6th grade teachers liked teaching the curriculum, and the 6th grade students liked using the curriculum to learn about CS. Using the ideas from this initiative, teachers and researchers can think about new stories and games to teach basic CS concepts, and students can alter or create new stories and games to illustrate their understanding of the concepts.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under the grant DRL-1923628.

REFERENCES

[1] P. Apostolellis, M. Stewart, C. Frisina, and D. Kafura. 2014. RaBit EscAPE: A Board Game for Computational Thinking. In *Conference on 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] C. Bueno. 2014. *Loren Ipsum.* No Starch Press.

[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] B.J. Cheryan, Drury and M Vichayapai. 2013. Enduring Influence of Stereotypical Computer Science Role Models on Women's Academic Aspirations. (2013). https://doi.org/10.1177/0361684312459328

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

[11] CS For Fun: Queen Mary, University of London. 2011. Noughts & Crosses. http://www.cs4fn.org/programming/noughts-crosses. Accessed: 2021-01-07.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] G. Dietz, J. Le, N. Tamer, J. Han, H. Gweon, E. Murnane, and J. Landay. 2021. StoryCoder: Teaching Computational Thinking Concepts Through Storytelling in a Voice-Guided App for Children. In *ACM Conf. on Human Factors in Computing Systems*. 1–15.

[18] M. Erwig. 2017. *Once Upon an Algorithm – How Stories Explain Computing.* MIT Press, Cambridge, MA.

[19] A. Gokhale and K Machina. 2010. Online Learning Communities to Recruit and Retain Students in Information Technology Programs. (2010). https://doi.org/10.1109/ITNG.2010.259

[20] 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.

[21] 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.

[22] W. Joel. 2013. A story paradigm for computer science education.. In *ACM conference on Innovation and technology in computer science education (ITiCSE)*. 362–362.

[23] D. Kafura and D. Tatar. 2011. Initial experience with a computational thinking course for computer science students. In *ACM SIGCSE Symp. on Computer Science Education*. 251–256.

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

[25] 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.

[26] J. Kubica. 2012. *Computational Fairy Tales.* CreateSpace Independent Publishing Platform.

[27] 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.

[28] 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.

[29] 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.

[30] Cindy Marling and David Juedes. 2016. CS0 for Computer Science Majors at Ohio University. *In Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (2016), 138–143.

[31] M. F. Mascolo and K. W. Fischer. 2005. *Constructivist theories.* Cambridge, England: Cambridge University Press. 49–63 pages.

[32] M. L. Niess, J. Parham-Mocello, and M. Erwig. 2021. Reframing Middle School Mathematics Teachers' TPACK for Teaching A New Computer Science Curriculum: Researcher-Practitioner Partnership, Board Games, and Virtual Teaching Experiences. In *Int. Conf. of the Society for Information Technology & Teacher Education*. 1623–1631.

[33] J. Parham-Mocello, G. Barrett, and A. Gupta. 2023. Manipulatives for Teaching Computer Science Concepts. In *IEEE Conf. on Frontiers in Education*.

[34] 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.

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

[36] 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.

[37] J. Parham-Mocello, M. Erwig, and M. Nies. 2022. Using a Text-Based, Functional Board Game Language to Teach Middle School Programming. In *IEEE Conf. on Frontiers in Education*.

[38] J. Parham-Mocello, M. Erwig, and M. Niess. 2021. Teaching CS Middle School Camps in a Virtual World. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*. 1–4.

[39] J. Parham-Mocello, M. Erwig, M. Niess, J. Weber, M. Smith, and G. Berliner. 2023. Putting Computing on the Table: Using Physical Games to Teach Computer Science. In *ACM SIGCSE Symp. on Computer Science Education*. 444–450.

[40] J. Parham-Mocello, A. Nelson, and M. Erwig. 2022. Exploring the Use of Games and a Domain-Specific Teaching Language in CS0. In *ACM Conf. on Innovation and Technology in Computer Science Education*. 351—-357.

[41] D. Parsons and P. Haden. 2006. Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. *Proceedings of the 8th Australasian Conference on Computing Education* 52, 157–163.

[42] Jean Piaget. 1968. *Six Psychological Studies.* New York: Vintage Books.

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

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

[45] Nathan Rountree, Janet Rountree, and Anthony Robins. 2002. Predictors of success and failure in a CS 1 course.. In *SIGCSE Bull.*, Vol. 34. 121–124.

[46] Nathan Rountree, Janet Rountree, Anthony Robins, and Robert Hannah. 2004. Interacting factors that predict success and failure in a CS 1 course.. In *SIGCSE Bull.*, Vol. 36. 101–104.

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

[48] 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.

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

[50] L. S. Vygotsky. 1978. *Mind in society: The development of higher psychological processes.* Cambridge, MA: Harvard University Press.

[51] B. C. Wilson and S. Shrock. 2001. Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors. In *ACM SIGCSE Symp. on Computer Science Education*. 184–188.