Student Understanding of Recursion: A Review of the Literature

Kenneth Pat (A14950024), Roger Cheng (A14743993), William Hogan (A53305475)

ABSTRACT

In this work, we present a review of research on student understanding of recursion. Recursion is an important and challenging topic in computer science education. Student understanding of recursion is a well-studied research area yet there is a lack of literature reviews summarizing the work. Specifically, no literature reviews report on the tools and software used to help students learn recursion. We attempt to fill that gap with this work. We address the following four research questions: (1) What concepts do students struggle with most when learning recursion? (2) What are the mental models of recursion? (3) What are effective teaching methods of recursion? (4) What tools have been used to teach recursion, and are they publicly accessible? To conduct our literature review, we limited our paper search to a few major conferences that feature computer education research. We employed a snowballing technique to expand the set of papers considered for the review. As presented, this literature review will assist instructors in finding tools and methods to effectively teach recursion. We hope it will also help to improve the teaching experience and student understanding while teaching and learning recursion.

ACM Reference Format:
Kenneth Pat (A14950024), Roger Cheng (A14743993), William Hogan (A53305475). 2020. Student Understanding of Recursion: A Review of the Literature. In *Proceedings of the 2020 International Computing Education Research Conference (ICER '20), August 10–12, 2020, Virtual Event, New Zealand.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3372782.3406266

1 INTRODUCTION

Recursion is a cornerstone topic in the education of students studying computer science. It is a notoriously difficult concept for instructors to teach and for novice programmers to learn. Fortunately, there is a large body of research that focuses on student understanding of recursion. The earliest studies date back to 1982 [4] with topics ranging from understanding common student misconceptions to developing teaching methods to help overcome these misconceptions. Much work has been done to accurately capture student understanding of recursion with mental models that describe the exact concepts students struggle with [8, 16, 17]. Thorough and complex concept inventories have been developed to help further identify the areas students struggle with the most when learning recursion [9].

Interestingly, despite the abundance of computer education research papers on student understanding of recursion, very few literature reviews have focused on this topic. In our search, we found only a single literature review that touched on student understanding of recursion. This gap in literature reviews makes it difficult for researchers to get a top-level understanding of the field. It also presents a challenge to instructors who hope to leverage research findings in their classrooms.

Our interest in this work is driven by our own experiences as computer science students who recall learning and wrestling with the difficult concepts of recursion. The motivation for this work is to assist instructors, teaching assistants, and tutors who teach recursion by providing a summary of the recent research that examines student understanding of recursion and the tools used to overcome student misconceptions. Ideally, this will lead to better learning outcomes for novice programmers who struggle with learning recursion, which, as evident in the literature, are a majority of novice programmers [5]. We also hope this work will provide an overview for researchers looking to identify future areas of research.

We begin our literature review with an overview of the common challenges students as well as the mental models students construct when learning recursion. We then present an updated inventory on the software and tools used to help students understand recursion.

1.1 Terminology

Given the large body of research articles on student understanding of recursion, there are a couple of terms used to describe identical concepts. For clarity, we define the terms we use in this literature review and map them to their corresponding synonyms found in the literature.

- Active flow: Active flow is when control is passed forward to new instantiations of a recursive function. Synonyms include "forward flow" and "head recursion."
- Passive flow: Passive flow is when control flows back from terminated instantiations of a recursive function. Synonyms include "backward flow" and "tail recursion."
- Base case: A condition that ends the recursive process and may or may not trigger a passive flow.

2 RELATED WORK

In this section, we detail the single related literature review that focuses on student understanding of recursion, namely *Teaching and learning recursive programming: a review of the research literature* [15]. In it, the authors focus on three aspects of student understanding of recursion: misconceptions, mental models, and teaching methods used to introduce concepts of recursion. The literature review is split evenly into these three areas of focus. Notably, this work intentionally leaves out any summary or analysis of the software and tools used to help students learn recursion. They instead focus on presenting and analyzing various approaches used to introduce recursion to novice programmers. With this focus, they do briefly touch on one tool that has been used to help students write recursive programs, namely the Cargo-Bot, a commercial iPad video game [14]. They mention that the Cargo-Bot has been shown

ICER '20, August 10–12, 2020, Virtual Event, New Zealand
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7092-9/20/08…$15.00
https://doi.org/10.1145/3372782.3406266

to improve a student's ability to *write* a recursive function but has little to no effect on improving their ability to *trace* a recursive function.

In [15], the authors note that, while some conflicts exist in the literature, several effective teaching strategies have emerged. Students learn better when presented with a variety of recursive analogies, examples, and contexts. Recursive mathematical functions are useful for teaching recursion in that they side-step the need for programming proficiency. Concrete examples of recursion, such as Russian nesting dolls, are helpful aids when explaining the concepts of recursion. Students typically understand recursion better when they are first introduced to looping and interactive functions. And, it is beneficial to start teaching recursion with examples that feature both the active and passive flows instead of simpler examples that only feature the active flow. These findings are the results of their review of 49 research papers studying student understanding of recursion.

To the best of our knowledge, the aforementioned literature review is the only review on student understanding of recursion. Since its publication in 2015, hundreds of research papers and journal articles have been published on student understanding of recursion. We focus our literature review on works that are not reviewed in [15]. Our main focus is on works that feature software and tools used to teach recursion, something that has not been covered in any literature review up to this point.

3 METHOD OF CONDUCTING THE REVIEW

3.1 Research Questions

Our literature review examines four research questions:

(1) RQ1: What concepts do students struggle with most when learning recursion?
(2) RQ2: What are the mental models of recursion?
(3) RQ3: What are effective teaching methods of recursion?
(4) RQ4: (a) What tools have been used to teach recursion? (b) Are they publicly accessible?

The corresponding motivations for these research questions are as follows:

(1) To discover the recursive concepts students struggle with the most.
(2) To describe the established mental models students develop when learning recursion.
(3) To explore which teaching methods are or are not effective in helping students understand recursion.
(4) To compile a list of tools and software that exist to help students learn recursion and, to report on their public availability.

3.2 Paper Search and Filtering

We limit our paper search to papers featured in the following major conferences: SIGCSE, SIGCHI, ITiCSE, ICER, ACE, Koli Calling, ACM SIGGRAPH.

We also include papers from the *IEEE Transactions on Education* journal. Given the scope of this project, we employed a limited version of the snowballing technique described in [23]. Snowballing is a method for collecting papers for a thorough and systematic literature review. Instead of a typical database-centric paper search, snowballing involves tracing the reference list of an initial "start-set" of core relevant papers. For our start-set, we collect significantly relevant papers (i.e. papers which focus exclusively on student understanding of recursion) from the aforementioned conferences and journals. We then employed the snowball method to expand the body of papers considered for our review.

3.3 Limitations

Given the context of this project, we could not conduct an exhaustive literature review of student understanding of recursion. This work is considered a draft that will require significant expansion to be considered complete. There were hundreds of relevant papers that we were unable to review for this project. As such, our results should be considered in this context.

4 RESULTS

4.1 RQ1: What concepts do students struggle with most when learning recursion?

In this section, we present an overview of works that focus on analyzing the concepts students struggle with the most when learning recursion. This helps to build the context for the following sections which focus on mental models that describe these common misunderstandings (Section 4.2), and teaching methods (Section 4.3) and tools (Section 4.4) used to avoid and/or overcome these misconceptions.

Kahney (1983) [13], is one of the early seminal works in student understanding of recursion research. In it, Kahney compares a novice programmer's understanding with an expert programmer's understanding of the process of recursion. The study aims to uncover exactly what a novice programmer understands about recursion. To do this, they first test the hypothesis that novices and experts differ in their conceptual models of recursion and their understanding of the execution of a recursive function. They, then, try to discriminate between the mental models that students develop to understand recursion. The study features a recursive question that is given to a group of novices (N=30) and a group of experts (N=9). They evaluate all responses in terms of correctness and in terms of the corresponding written explanation of the problem. They find that, as expected, most novices construct the "looping" model of recursion that views a recursive procedure as a single object instead of a series of new instantiations. Using this model, students incorrectly assume that recursive variables can be overwritten by subsequent recursive calls. All but one expert was shown to possess the viable "copies" model of recursion that views recursive calls as separate "copies" of the function. These mental models are discussed further in Section 4.2 A few students (3 of 30) were able to develop a "copies" model of recursion but, upon deeper analysis of video recordings of students solving the problem, it was shown that only one student in the group actually acquired an expert's understanding of recursion. The author notes that arriving at the correct answer for a recursive function is not sufficient in assessing understanding. The author concludes that mental models of novices vary but they may still arrive at the same solution to a problem. Deeper analysis (video recording, written explanation, etc.) is needed to assess understanding.

Sally, Stephen, Hicham, Jeremy and Clifford (2017) [9] came together and developed a concept inventory that provides measurement to student's misconceptions on recursion topics. This inventory presented a collection of misunderstandings and difficulties encountered by students when they are learning introductory recursion in a basic computer science course. From the inventory, the researchers developed a series of questions to evaluate the student's understanding of the concept. The paper talked about how the recursion concept inventory was built, which is starting by having common problematic topics found in recursion teaching literature which were, backward flow, active flow, recursive calls, limiting case, infinite recursion, confusion with loop structure, and variable updating. Then they asked 20 instructors to rank their difficulties. After that, they analyzed around 8000 responses to recursion questions that were given to students during tests and filtered out common misconceptions related to the topics. Then they created the CI questions that are given as "fill in the blank" and validated the questions by giving them to the students. Professors can now use this CI to measure student understandings of basic recursion.

R. Sooriamurthi (2001) [18] conducted a study of why upper-division undergraduate students were having difficulty in comprehending the ideas behind recursion. The paper states that students are having troubles understanding recursion due to 3 reasons: insufficient exposure to declarative thinking in a programming context, inadequate appreciation of the concept of functional abstraction, and lack of a proper methodology to express a recursive solution. The paper states that two tools that can help with understanding recursion complexity are an abstraction, which helps to reduce it, and modularity, which helps to better manage it. To master a recursion, students would need to master and acquire a fundamental understanding of functional abstraction and fully know the functionalities of the method. Modularity is achieved when having the idea of divide and conquer and divide the problem into subproblems of the same type. One example would be breaking a problem down by templating it into a base case, simplifying routine, natural recursion, and completion. This approach aids students in formulating recursive solutions very easily.

As reported in the literature, many students struggle with understanding recursive functions. The most common mistake is to construct a "looping" model of recursion where the function is viewed as a single object, capable of sharing variable values between different recursive instantiations. We also observe that, although students are capable of accurately tracing a recursive function, they, in most cases, still lack an expert level understanding of recursion. There are thorough concept inventories that are available for instructors to accurately assess student understanding of recursion. The concept inventory should help instructors measure the effectiveness of the teaching methods and tools they use to teach recursion. Students struggling with the core concepts of recursion may benefit from learning how to construct an abstract representation of recursive functions, as well as how to modularize the functions into easier-to-understand sub-functions.

4.2 RQ2: What are the mental models of recursion?

Many of the studies that attempt to assess student understanding of recursion also classify student understanding with mental models. These mental models help researchers and instructors better understand the issues students face when learning recursion. Important to note is that "mental models" differ from "conceptual models." Mental models are constructed by the students whereas conceptual models are the tools an instructor uses to teach the concepts of recursion. Below, we discuss some of the seminal works that focus on the development and use of student mental models of recursion.

Sanders et al. [8] investigate the mental models students use to represent the mechanics of recursion. The authors describe two categories of mental models: viable and non-viable. Viable models allow a student to accurately and consistently represent the mechanics of recursion. Non-viable models consist of one or many misconceptions about the fundamental concepts of recursion. They list eight distinct mental models: "copies", "looping", "active", "step", "return value", "magic", "algebraic", and "odd." The "copies" model is the only viable model. The other models feature some degree of misunderstanding. The authors research multiple groups of first year students tracing two different recursion problems on an exam. Their study shows that many students struggle with the passive flow of a recursion problem and with the passing of arguments from one instantiation to another. They also show that some students use an IF/THEN conceptual framework to understand recursion which leads to the misconception that each recursive iteration is a complete operation and those operations are simply combined to create the program's output. The authors claim that, armed with this information, instructors can tailor curricula to better address student misconceptions.

In [16], Sanders et al. build on their 2003 research paper and, in it, they expand their test to include a more complicated list calculation recursion problem. Since the original study the authors, who are also instructors of an introductory algorithms course, changed the way they introduced the concepts of recursion to their students. Instead of first introducing basic recursive problems that only require an active flow (e.g. adding the first N numbers), they begin with problems that require both active and passive flows (e.g. reversing a list). They suggest that students who first learn "simpler" problems without a passive flow are likely to create what they call an "active" mental model of recursion. With the active mental model, students stop when they reach the base case and fail to trace through the passive flow part of the recursive problem. In their study, they show that students are more likely to develop a viable mental model of recursion when they are introduced to recursive problems that have both active and passive flows.

Scholtz and Sanders [17] attempt to determine whether viable trace mental models of recursion show students' true understanding of recursion. Other research has suggested that a student may be able to accurately trace a recursive function without fully understanding recursion. This paper builds off of previous papers by Ian Sanders in that it uses the same eight types of mental models to categorize students' understanding of recursion – copies, active, looping, odd, algebraic, return-value, step, and passive. To determine student understanding of recursion vis-a-vis their ability to accurately trace a recursive problem, the authors design a three-part assessment. The first part examines a students' ability to trace a recursive function. The second part assesses their ability to prove the termination of a function. And, the third part tested a student's ability to describe the execution of the recursive function. They specifically examine the results of the students who are able to pass the first part of the assessment (e.g. students that possess a "viable" trace mental model, namely the "copies" mental model). They report that most of the students who had viable trace mental models failed at other measures of recursive understanding. The students were unable to clearly explain the execution of the recursive functions. The main weakness students had was their ability to understand the passive flow of a recursive algorithm. The authors conclude that obtaining a viable trace mental model that allows students to accurately predict the outcome of a recursive function is not a sufficient measure of a student's understanding of recursion. They recommend that instructors include a wide range of recursive functions in their curricula, especially functions that have commands that must be executed after the recursive call.

Student mental models of recursion are well-studied and well-established. They can be used by researchers and instructors alike to categorize student understanding of recursion. They can also be used to illustrate the progress made by students within a study or a computer science course. Understanding common student mental models of recursion can inform teaching methods and better development of learning tools.

4.3 RQ3: What are effective teaching methods of recursion?

Over the years, computer science educators developed different methods to teach recursion for students to better understand the concepts of recursion.

Velázquez-Iturbide, J. Ángel, M. Eugenia, and Raquel developed (2015) [21] developed an instructional method to translate linear recursion algorithms into equivalent, iterative ones to enhance students' understanding of recursion tracing. The paper gave out an example of translating recursion into two loops, one representing the forward phases while the other representing the backward phases. Students appreciate this method because they can achieve a deeper understanding of the algorithmic knowledge they had already acquired. A study was made by first teaching normal recursion to the students and then test them on the topic. Later on, recursion removal methods were taught, and they were given two linear recursive algorithms asking to trace and translate into iterative algorithms. After that, they were given the same test again as before. The results show that 47 or 59% of students improved their mental models, but there was no significant improvement for the design of recursive. Students would need to fully understand the mechanics of recursions to apply the translation rules.

Wu et al. [24] conducted a study by designing an experimental research design, which consisted of different types of research models and cognitive learning styles. The goal of this study was to examine the impact of learning styles on the contribution of performance on students. There are two main types of conceptual models and learning styles, concrete and abstract. Students were formed into four groups, according to the models and types of learners: abstract learners with abstract models, abstract learners with concrete models, concrete learners with abstract models, and concrete learners with concrete models. Students who said they fall into the concrete learning style would typically prefer to sense and feel their way, whereas students who are in abstract learning style prefer to think their way through. Categorizing students' learning styles is an important key in the study because it tells us about the style of information processing, which is directly related to knowledge acquisition. During the study, there were five conceptual models used in introducing recursion to students, three of them were concrete, and two of them were abstract. The study concludes that using concrete models to teach novice learners about recursion is effective, and it is supported by the investigation from this study. However, the study does not support the assertion that students' learning styles and the style in conceptual models should be matched.

Ginat and Shifroni [7] examined how confident students are after they learned about recursion in programming. The study started as a test and found that students still prefer to use the iterative pattern in problem solving, due to the lack of trust when building the recursive solution. As a result, the authors designed a study to continue to teach students about recursion more systematically. Students were divided into two groups, with one group focusing on the mechanism of recursion and the other focusing on the declarative level. The study concluded that students tend to have trouble writing recursive functions, especially on getting the logic part of the code working. The study also suggests that teaching recursion with the declarative and abstract level of thinking would considerably improve students' ability to write recursion code. The difficulties in writing recursive functions have made many students prefer the iterative method instead because oftentimes, students were worried that the recursive method would not be able to execute correctly.

In the book "Guide to Teaching Computer Science" written by Hazzan et al. [10], it was stated that the familiarity with research works on teaching recursion is an important factor to contribute to improving teaching experience. Specifically, the book mentions four strategies a teacher could do to improve teaching experience: joining an education community, increasing teachers' awareness, strengthening pedagogical knowledge, as well as broadening the teaching toolbox. The book provides four activities that correspond to each of the strategies aforementioned. Each activity has a worksheet containing a list of activities for teachers to do, and they are directly related to building up the skills of instruction on teaching recursion.

A lot of research on recursion teaching methods has been done over the years. Students tend to understand recursion better by thinking in an iterative matter, so researchers developed ways of translating recursion into linear algorithms. Researchers also found that different student learning style fits different teaching models. Moreover, studies have shown that teaching recursion with declarative and abstract level thinking may also improve student's skill of developing recursive functions. As a result, different teaching strategies and class activities were created to further assist computer science educators in teaching recursion.

4.4 RQ4: (a) What tools have been used to teach recursion? (b) Are they publicly accessible?

Here are some of the tools that were invented to further help students see the workflow of recursion functions.

George(2000) [6] introduced a publicly accessible novel software visualization aid, the EROSI(Explicit Representer Of Subprogram Invocations) that helps students understand recursion and apply recursive principles to problems. George argues that a major problem of understanding recursion is subsumed into the problem of having an effective mental model of subprogram invocations. Novels need to visualize a more sophisticated subprogram execution model which emphasizes various dynamic processes logically representing a subprogram execution. The EROSI tutor uses visualization and animation to simulate the dynamic-logical model of subprogram calls which represents the model of recursion. By showing what parts of the code were being executed, the software demonstrated both dynamic code visualization and dynamic algorithm visualization. Tests have shown that EROSI helped students significantly read and understand recursive programs as well as write recursive programs. Studies have shown although the software helped students in understanding the copies model and flow of control, students are still having problems with understanding variable updating and computer memory storage.

P. Henderson and F. Romero (1989) [11] believed that Standard ML can be used as a recursion problem-solving tool for instructors. ML is a publicly accessible functional programming language with features that make it excellent for teaching recursion. With little formal ML instruction, first year students were able to use recursively defined data structures and to define fairly powerful recursive functions in ML after several weeks. There exists an SML which is an interactive interface that can take in the recursive function and output the result and type. ML had very simple syntax and close to mathematical notation which supports the creation of new abstractions. ML supports exploratory learning, so it can take small textual definitions in the system and get feedback immediately.

Wilcocks and Sanders (1994) [22] introduced how using an online program animator can present recursive functions as dynamic copies of recursive instantiations to further assist students in understanding recursions. This program can dynamically display data structures and help students visualize the flow of control in recursion structures. The Recursion Animator showed the instantiations of a recursive routine as a series of overlaid windows or copies which opens and closes according to the recursive flow of control. The program can go forward and backward and pause to fit the speed of the user. This program was tested with three groups of 10 students, who were first year CS students. They mostly showed a better understanding of recursion after using the program and think it assisted them in acquiring the ability to trace recursive functions.

Omar, Fossati, Eugenio, and Green (2014) [1] also recommended the software ChiQat-Tutor that provides an integrated public accessible environment for learning recursion. ChiQat-Tutor uses recursion graphs to help students visualize, manipulate, and learn recursive processes. Students struggle with understanding recursion due to unfamiliarity with recursive activities, visualization of program execution, and back-flow of execution after reaching the bases, comparison to loop structures, and everyday analogies. ChiQat-Tutor offers an environment for learning recursion by making use of RGraphs as a visual representation of recursive execution. RGraph are directed graphs with two sets of vertices, one representing recursion calls, and the other represents pre/post processing statements of recursive calls. RGraph was built layer by layer with directed edges showing the execution sequence and value being passed in/returned. Students can interact with RGraph to better understand recursive processes. The ChiQat supports five tasks: tracing RGraph, validating RGraph, Constructing Rgraph, animation Rgraph, and answering multiple choice reflective questions.

Zhang, Mustafa, Emanuel, Caldwell, and Jones (2014) [25] presented a game module "Recursive Runner" to reinforce student understanding of recursion. It is publicly accessible and designed for short game play time, and it does not require any prior experience in gaming. This game offers an easy and fun environment for students to visualize and follow the flow of execution of recursive functions. This game also provides helpful guidance when students make mistakes. Four CS undergraduates developed the Recursive Runner with GameMaker Studio for 3 months. The objective of this game is to stop a reactor from exploding by solving Fibonacci and factorial recursive problems. An experiment was conducted by giving a pregame recursion topic test and a post game recursion topic test to entry level CS students and the average improvement was 21%. Moreover, students had very positive feedback and agreed that the game was helpful. A higher learning gain was shown by playing the game multiple times.

Dann et al. (2001) [3] examined the topic of using visualization to teach novice students about the concept of recursion. During the study, students in a pre-CS1 course were introduced to a new method to learn recursion by using 3D animation software as a tool to construct visualization. This helps students to actually see how recursive functions are called each time, level by level, which would give them a more intuitive understanding of how many times a recursive function is called during execution. The paper uses software called "Alice" developed by Stage3 Research Group, which is a 3D interactive programming environment to build 3D objects from code. Examples during the demonstration process include a "Rabbit and Butterfly Chase" and a group of "Towers of Hanoi". The study concludes that using 3D animation while teaching students about recursion is an effective way to introduce them to the topic. 3D animation software provides details in visualization, experimentation, and mathematical explanation, which helps students to understand the fundamentals of recursion more intuitively.

Chaffin et al. [2] conducted a study by introducing a new type of video game to students in computer science courses. During the process of the study, students were introduced to a video game named EleMental. This video game, however, differs from other video games we know because it is an interactive learning environment. It uses visualization to help students to learn recursion by depth-first search of a binary tree. Students were asked to write code for the DFS method of a binary by providing them with a solution but with some lines removed. If students had a wrong solution compiled, the compiler will broadcast an error message by telling them which line was incorrect. The study concluded that using the video game EleMental: The Recurrence to teach students about the fundamentals of recursion is statistically significant. Even though the study was conducted for only 43 students, the difference in the scores between them was large enough to conclude that it is statistically significant. This suggests that EleMental, by providing good visualization and interactive feedback, is an effective method to teach students about recursion in a video game. Besides, this paper also provides much valuable information in researching educational games for computing.

Tessler et. al [19] conducted a new method of recursion by using a video game. The study was based on 47 high school students who were taking AP Computer Science A. Researchers deemed that using the video game "Cargo-Bot" to teach recursion is effective to improve student understanding on this topic. Cargo-Bot is publicly available and can be downloaded for free. The game is designed to help novice programmers visualize recursion. The objective is to let players design algorithms by adding procedures to "teach a robot how to move crates". Students were divided into a control group and an experimental group. The control group would use Cargo-Bot after the mid-test, whereas the experimental group would use Cargo-Bot before taking the mid-test. The study concludes that, by using a two-sample t-test, it is revealed that students in the experimental group have gained larger learning gains than those in the control group. More importantly, by using Cargo-Bot in classrooms, students would have a stronger ability to write recursive functions in the future.

Hsin [12] conducted a study about using Recursion Graph (RGraph) to teach recursion to students. The objective is to help students visualize how recursive functions begin and end, and thus gaining a more thorough understanding of the workflow of recursion. The Recursion Graph introduced in the paper works similar to a flowchart, but instead of letting arrows pointing to one final destination, arrows in a Recursion Graph form a cycle. In other words, the statements and arrows form a loop, where each function call is denoted by a rectangle, and each print statement is denoted by an oval. The paper also compares the difference between a Recursion Graph vs. a Horstmann's graph. It is stated that Horstmann's graph might not be very helpful to students since Horstmann's graph tells an abstract concept, thus making it hard for students to absorb the principle behind it.

Vasić et al. [20] developed a new teaching method to teach recursion by using HTML 5 games. The objective is to ease the learning curve and make recursion a topic that is easier and more interesting for students to understand. The authors first stated that many instructors have misused the concept of "Towers of Hanoi" to teach recursion. Since the principles behind it are still baffling students, students might not understand it thoroughly. The authors conducted a study on first year computer science students to measure the impact of this new game. The statistical results suggest that students who played that game scored higher in their post-test. The study concluded that by designing this HTML 5 game and letting students engage with it, students would have gained the right mental model and learn about recursion more comprehensively.

Throughout the development of computer science education research, various tools have been developed and applied in classrooms to help students understand the fundamental concepts of recursion in easier ways. Many studies conducted were centered around the idea of using different types of tools to demonstrate the principles of recursion. These tools, including but not limited to: graphs, video games, and 3D animation software, all share a common purpose, which is to visualize recursion. They help to divide a recursive algorithm into each iteration, thus allowing students to see how the algorithm is assembled.

5 POTENTIAL IMPACT

As mentioned in Section 3.3, this is a draft literature review. When complete, this literature review will assist instructors in finding tools and methods to effectively teach recursion. It will also help to improve the teaching experience and student understanding while teaching and learning recursion. As a complete and through literature review, it will help other researchers get an overview of the topic and assess the needs for future research. The review may also be helpful to educational tool developers who are interested in learning about the tools that have or have not been successful in teaching recursion. Ultimately, we hope this review will help improve learning outcomes for novice programmers struggling with recursion.

ACKNOWLEDGMENTS

We would like to thank Professor Raj and teaching assistants Ana Selvaraj and Anne Xu for their guidance in shaping this literature review.

REFERENCES

[1] Omar AlZoubi, D. Fossati, B. D. Eugenio, and N. Green. Chiqat-tutor: An integrated environment for learning recursion. 2014.

[2] Amanda Chaffin, Katelyn Doran, Drew Hicks, and Tiffany Barnes. Experimental evaluation of teaching recursion in a video game. In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, Sandbox '09, page 79–86, New York, NY, USA, 2009. Association for Computing Machinery.

[3] Wanda Dann, Stephen Cooper, and Randy Pausch. Using visualization to teach novices recursion. ITiCSE '01, page 109–112, New York, NY, USA, 2001. Association for Computing Machinery.

[4] Gary Ford. A framework for teaching recursion. *SIGCSE Bull.*, 14(2):32–39, June 1982.

[5] Judith Gal-Ezer and David Harel. What (else) should cs educators know? *Communications of the ACM*, 41(9):77–84, 1998.

[6] C. E. George. Erosi—visualising recursion and discovering new errors. In *SIGCSE '00*, 2000.

[7] David Ginat and Eyal Shifroni. Teaching recursion in a procedural environment—how much should we emphasize the computing model? In *The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '99, page 127–131, New York, NY, USA, 1999. Association for Computing Machinery.

[8] Tina Götschi, Ian Sanders, and Vashti Galpin. Mental models of recursion. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 346–350, 2003.

[9] Sally Hamouda, Stephen H Edwards, Hicham G Elmongui, Jeremy V Ernst, and Clifford A Shaffer. A basic recursion concept inventory. *Computer Science Education*, 27(2):121–148, 2017.

[10] Orit Hazzan, Noa Ragonis, and Tami Lapidot. *Guide to Teaching Computer Science*. Springer Nature Switzerland AG, 2020.

[11] P. Henderson and F. Romero. Teaching recursion as a problem-solving tool using standard ml. In *SIGCSE '89*, 1989.

[12] Wen-Jung Hsin. Teaching recursion using recursion graphs. *J. Comput. Sci. Coll.*, 23(4):217–222, April 2008.

[13] Hank Kahney. What do novice programmers know about recursion. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 235–239, 1983.

[14] Elynn Lee, Victoria Shan, Bradley Beth, and Calvin Lin. A structured approach to teaching recursion using cargo-bot. In *Proceedings of the tenth annual conference on International computing education research*, pages 59–66, 2014.

[15] Renée McCauley, Scott Grissom, Sue Fitzgerald, and Laurie Murphy. Teaching and learning recursive programming: a review of the research literature. *Computer Science Education*, 25(1):37–66, 2015.

[16] Ian Sanders, Vashti Galpin, and Tina Götschi. Mental models of recursion revisited. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 138–142, 2006.

[17] Tamarisk Lurlyn Scholtz and Ian Sanders. Mental models of recursion: investigating students' understanding of recursion. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 103–107, 2010.

[18] R. Sooriamurthi. Problems in comprehending recursion and suggested solutions. 2001.

[19] Joe Tessler, Bradley Beth, and Calvin Lin. Using cargo-bot to provide contextualized learning of recursion. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, ICER '13, page 161–168, New York, NY, USA, 2013. Association for Computing Machinery.

[20] Daniel Vasić, Emil Brajković, and Tomislav Volaric. Experimental evaluation of teaching recursion with html5 game. 09 2014.

[21] J. Velázquez-Iturbide, M. Castellanos, and Raquel Hijón-Neira. Recursion removal as an instructional method to enhance the understanding of recursion tracing. *IEEE Transactions on Education*, 59:161–168, 2016.

[22] Derek Wilcocks and I. Sanders. Animating recursion as an aid to instruction. *Computer Education*, 23:221–226, 1994.

[23] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery.

[24] Cheng-Chih Wu, Nell B. Dale, and Lowell J. Bethel. Conceptual models and cognitive learning styles in teaching recursion. In *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '98, page 292–296, New York, NY, USA, 1998. Association for Computing Machinery.

[25] J. Zhang, Mustafa Atay, Emanuel Smith, E. R. Caldwell, and E. Jones. Using a game-like module to reinforce student understanding of recursion. *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–7, 2014.