Scaffolding Expertise: Evaluating Scaffolds for Block-Based Coding Among Experts and Novices

Yifan Zhang*
Smart Learning Institute, Beijing Normal University
China
zhangyifan@bnu.edu.cn

Teomara Rutherford
University of Delaware
United States
teomara@udel.edu

Abstract

Coding for computer programming is a common way to support and assess computational thinking (CT) skills, a set of problem-solving skills essential for computer science (CS) and STEM more broadly. We designed a coding platform, Fox and Field, to determine what kinds of scaffolds can encourage novice coders to behave more like experts. We compared actions between 221 upper-division undergraduate CS/engineering (expert, n=106) and social science (novice, n=115) majors randomized to scaffolding conditions from four universities in the United States. Overall, experts used statistically significantly more practices aligned with CT skills, such as those that increased code efficiency (e.g., non-right angle turns, loops). This difference disappeared when novices were scaffolded by being told to use fewer codes or by priming with critical features. Both experts and novices were equally likely to be swayed by purposefully distracting features within the platform, such as a drawn path to deflect them from the most efficient solution. Results present initial evidence regarding which features of coding platforms can direct students to exercise CT-linked practices, leading to recommendations regarding platform development to better support learning.

CCS Concepts

• **Social and professional topics** → Professional topics; Computing education; Computational thinking; Professional topics; Computing education; Student assessment; Professional topics; Computing education; Computing education programs; Computer science education; • **Software and its engineering** → Software organization and properties; Contextual software domains; Virtual worlds software; Interactive games; • **Applied computing** → Education; Interactive learning environments.

Keywords

computational thinking, scaffolding, game-based learning

*Corresponding author.

ISAIE 2024, September 06–08, 2024, Xi'an, China

ACM ISBN 979-8-4007-0710-0/24/09
https://doi.org/10.1145/3700297.3700345

ACM Reference Format:
Yifan Zhang and Teomara Rutherford. 2024. Scaffolding Expertise: Evaluating Scaffolds for Block-Based Coding Among Experts and Novices. In *2024 International Symposium on Artificial Intelligence for Education (ISAIE 2024), September 06–08, 2024, Xi'an, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3700297.3700345

1 Introduction

Computational thinking (CT) is a critical set of problem-solving skills that should be introduced and developed not only in computer science (CS) but also in other STEM subjects [1, 2]. The CT skills which include algorithms, patterns, abstraction, as well as decomposition, are fundamental for addressing complex problems and improving efficiency [3]. In the practical education context, coding usually serves as one basic pedagogical approach to instruct and assess CT competencies [4]. The iteration behavior of writing code, testing the model, and debugging errors could indicate the cognitive processes introduced in CT [5].

However, as coding involves certain levels of cognitive ability and abstraction, experts and novices perform tasks much differently in coding and cognitive strategies [6]. Moreover, experts leverage efficient logical and relatively systematic writing behaviors, demonstrating deep understanding and higher-level abstraction [7]. In contrast, novices often rely on trial and error, displaying superficial problem interpretation and difficulty in utilizing abstract concepts effectively [8]. Therefore, identifying differences in problem-solving behaviors between experts and novices could better design associated support and scaffolding for novices.

In this study, we designed a coding environment, Fox and Field, to test whether experts and novices perform differently in problem-solving (e.g., code efficiency) and whether certain scaffolds can prompt novice coders to perform expert-like behaviors. In this environment, learners need to control a virtual fox to follow certain instructions and reach the final goal point through block-based coding. Learners could use fewer non-right angle turns to save the code length compared to more right angle turns and demonstrate code efficiency. Scaffolds, in this context, refer to prompts and conditions that provide guidance and opportunity to learners, enhancing their task performance [9]. Our environment incorporated scaffolds, such as marking critical features and providing prior practice opportunities, aiming to guide learners' focus and facilitate skill transfer.

To this end, we ask two research questions:

- Are participants more likely to use the non-right angle turn (NRAT) when they are given an efficiency instruction?
- Are participants more likely to use the NRAT depending on their training level condition?

2 Literature Review

Papert coined the term CT in the 1980s and claimed it as a powerful literacy [10]. He also described it as one of the critical learning outcomes through Logo programming. Decades later, Wing, in 2006, promoted CT to the field of science, technology, engineering, and mathematics (STEM) education and also linked it to broader subjects [3]. Ever since then, CT has been defined with various versions, but generally focuses on problem-solving [11]. Several researchers defined CT in the programming context. For example, Brennan and Resnick developed the block-based programming environment, Scratch, and showed their three assessment dimensions for CT, including concepts, practices, and perspectives [5]. Weintrop et al. classified CT into four categories including data practices, modeling & simulation practices, computational problem-solving practices, and systems thinking practices [2]. Although CT can be instructed and assessed by programming, CT has also been defined more broadly as competencies needed in both specific knowledge and general problem-solving, which include decomposition, abstraction, algorithmic thinking, and debugging [12].

Problem-solving behaviors can vary depending on domain knowledge. For example, experts often exhibit functional and behavioral understanding that novices lack [13]. Similarly, Jessup et al. investigates how expert and novice programmers differ in their code comprehension and perceptions of code [14]. Leveraging eye-tracking data, in source code reading behavior, Aljehane et al. showed that experts and novices read source code statistically significantly differently, where experts use more efficient eye movements [15]. Considering debugging behaviors, experts utilize more complicated debugging strategies that novices usually cannot handle [16]. These studies demonstrate the difference between experts and novices and provide the foundation for developing associated scaffolding.

Experts and novices face different struggle points and misconceptions during problem-solving. Denny et al. discussed some cognitive difficulties novices face in automated assessment tools and called for associated scaffolding [17]. Scaffolding could benefit the learning outcomes and perceptions by reducing of degree of freedom, demonstration, and marking critical features [9]. For example, Rego et al. investigated how various feedback modalities (such as visual, auditory, and haptic feedback) influence the cognitive load on novice users [18]. One great effort is the research and practices in block-based programming environments, which removes syntax errors and focuses learners on logical thinking and problem-solving [4, 19]. In block-based programming, novices could benefit from structured guidance, including worked examples and annotation styles [20]. Besides reducing problem complexity, demonstrating or modeling solutions is another promising method. Denney et al. showed that solving test cases before programming could significantly improve novices' problem-solving process, which provides evidence that prior condition could impact their behavior [17]. Lastly, some studies demonstrated that marking critical features could highlight the current task and guidance learners away from distraction [21, 22]. These studies together showed the benefit of scaffolding to novices.

3 Methodology

3.1 Fox and Field

The main objective of our coding environment is to create a condition-controlled platform to isolate various conditions as experiment variables and provide tasks that could intuitively visualize differences in problem-solving strategies among experts and novices. Therefore, we developed a few fundamental design principles: (1) The coding language should be in the block-based structure to provide a low floor (i.e., user-friendly) for different learners [23]. Study on block-based environments also indicate that by eliminating syntax errors and reducing the degree of freedom, learners can focus on problem-solving and logical thinking [24]. (2) All levels should map to CT skills. Table 1 lists core CT skills that are commonly defined in literature [5, 20, 25, 26] and involved in our coding environments.

For our research question, we designed a train and test level with different conditions, which consisted of different maps and level instructions. The conditions are listed in Table 2 and the maps are shown in Figure 1. Experiment conditions came from three dimensions, the first condition is whether the train level show connected path (Figure 1a and 1b), the second condition is whether the test level contains distract path (Figure 1c and 1d), the third condition is whether the level instruction prompt for few codes (Table 2, column *Level Instruction*). Scaffoldings came from two aspects, the first is train level (i.e., structured practice opportunities and demonstrating solutions to a task), the second is the level instruction prompt (i.e., marking critical features).

3.2 Participants

Our participants recruitment was conducted at four universities in the United States. We distributed flyers around the campus, emailed through listservs and faculty, and compensated participants accordingly. We only included upper-division undergraduate students to ensure they had already taken their major courses to distinguish experts and novices. In total, 233 students were enrolled and scheduled time, and 221 students actually showed up and completed the study. We categorized 106 students who majored in CS, engineering, and similar subjects as experts and 115 students in other majors (e.g., psychology, education, and humanities) as novices. Detailed demographic information is listed in Table 3. All of our experiment procedures were approved by the IRB in advance.

3.3 Experiment context

All of the experiments were conducted through online meeting software and participants were randomly assigned to different game conditions. Participants login their pre-assigned account using their devices. Before the experiment began, two researchers briefly introduced the coding environment and participants signed an electronic consent form and took a survey that included demographic questions. We asked participants to share their screens while also showing their faces through the webcam, and the shared screens were videotaped. The experiment lasted for 1 hour and stopped immediately regardless of whether the participants finished all tasks or not. During the experiment, researchers did not provide any help to participants. Participants could try as many times as they

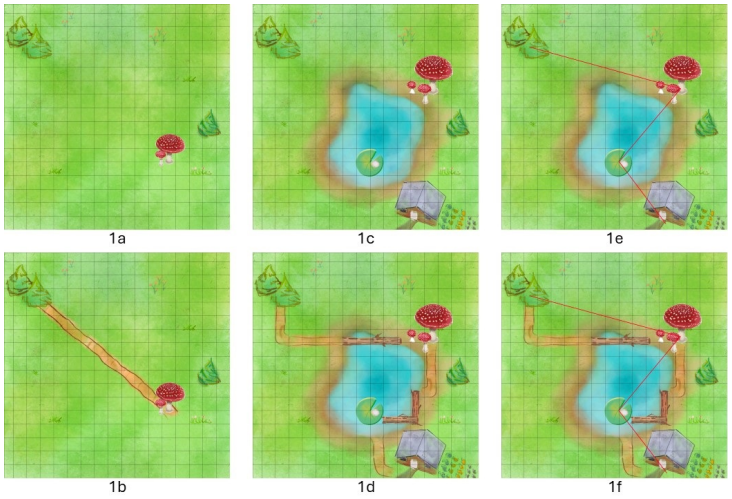

Figure 1: Game level conditions. 1a: Train 1 with no path shown; 1b: Train 2 with directly connected path; 1c: Test 1 and 2 with no path shown; 1d: Test 3 and 4 with connected path shown; 1e and 1f: Efficient paths in red lines.

Table 1: Core CT skills involved in our environment.

CT Skills	Game Features
Algorithmic Thinking	Players learn to create step-by-step instructions to solve problems
Decomposition	Players break down complex problems into smaller, more manageable parts
Pattern Recognition	Players identify patterns or similarities in the tasks they are performing
Abstraction	Players learn to focus on important information while ignoring irrelevant details
Debugging	Players identify and fix errors in their code
Loop	Players use repetitive sequences of blocks until a certain condition is met
Iteration	Players refine their solutions through repeated cycles of testing and improvement

Table 2: Game conditions for distract path and efficiency instruction. Distract path indicates that the map has some connected paths but not the fewest codes. Efficiency indicates a text prompt in the instruction for the fewest codes.

Conditions	Map Conditions	Instruction Condition	Level Instruction
Train 1	No path	No efficiency	Get the fox to the base of giant mushroom
Train 2	Connected path	No efficiency	Follow the path to the base of giant mushroom
Test 1	No distract path	No efficiency	Touch the base of the giant mushroom, then the lilypad, then go to the front door
Test 2	No distract path	Efficiency	Touch the base of giant mushroom, then the lilypad, then go to the front door. Try to use as few codes as possible
Test 3	Distract path	No efficiency	Touch the base of the giant mushroom, then the lilypad, then go to the front door
Test 4	Distract path	Efficiency	Touch the base of giant mushroom, then the lilypad, then go to the front door. Try to use as few codes as possible

needed until they finished each level. Once finished, they could not go back to that level.

3.4 Data Collection and Analysis

We collected background surveys, coding artifacts, coding process logs, screen recording videos, and reflection surveys students' answered after finishing each level. For this study, we chose coding artifacts as our main data source and analyzed artifacts of 221 students who completed level 4. We ran every coding artifacts and simulated the fox's paths. We coded the paths as using right-angle turns (RT) and using non-right-angle turns (NRAT). We leveraged chi-square and logistic regression for quantitative analysis. For those quantitative analysis, experts were coded as 1 and novices

Table 3: Demographic of participants.

Demographic	N(221)	Percentage(100%)	Demographic	N(221)	Percentage(100%)
Gender			Years in studying		
Woman	137	62%	2 years	47	21%
Man	64	29%	3 years	85	38%
Others	2	1%	4 years	73	33%
Prefer not to say	18	8%	4 years +	16	7%
Race/Ethics					
White	123	56%			
Africa America	13	6%			
Asian	50	23%			
Hispanic	15	7%			
Others	2	1%			
Prefer not to say	18	8%			

were 0; NRATs were 1 and RTs were 0; efficiency instruction as 1 and no efficiency instruction as 0.

4 Results

4.1 RQ1: Are participants more likely to use the non-right angle turn (NRAT) when they are given an efficiency instruction?

In comparing the three conditions, a chi-square test indicates differences in the use of the NRAT between conditions ($p < .001$), see Figure 2. From a logistic regression of whether the participant used the NRAT on dummy variables for whether the participant had an efficiency instruction or distractor, the distractor was a statistically significant negative predictor of the use of NRAT ($p < .001$); efficiency instruction had a positive non-statistically significant association with the use of NRAT ($p = .07$).

Using a chi-square test of difference, majors were equally distributed across the test conditions (test of difference, $p = .56$). On average, experts used NRATs more than did novices ($p = .02$), see Figure 3. However, this difference did not come into play in the non-efficiency, non-distractor path condition ($p = .75$). Experts were directionally more likely to use NRATs in the other two conditions, but these differences did not arise to statistically significant levels when examined separately (efficiency, no distractor, $p = .10$; efficiency with distractor, $p = .08$). When both efficiency instructor conditions were considered together, the percentage difference between experts and novices who used NRATs (42% vs. 22%) was statistically significant ($p = .01$).

A logistic regression of conditions and majors predicting NRAT revealed major as a positive predictor of use of NRAT ($p = .04$) and distractor condition as a negative predictor ($p < .001$), with a similar coefficient to the model above without the major control (-1.5 in each). Entering interactions between major and condition, none of the interaction terms were statistically significant (ps > .05).

The directional results and the statistically significant results from consideration of both efficiency conditions together are opposite our hypothesis—the efficiency instruction scaffold appeared to benefit experts more than it did novices.

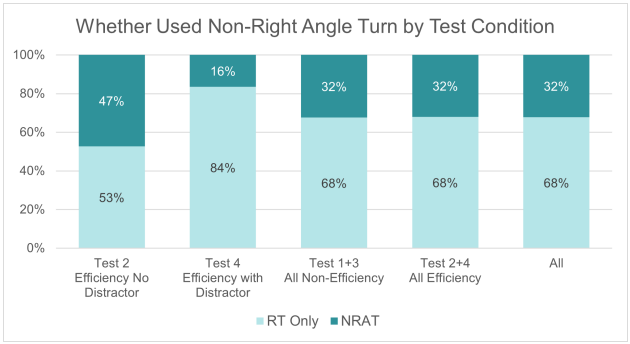

Figure 2: Distribution of using RT and NRAT under different test conditions.

4.2 RQ2: Are participants more likely to use the NRAT depending on their training level condition?

In our dataset, two training conditions are equally distributed among four testing conditions (test of difference, $p = .53$). Figure 4 shows whether participants used an NRAT, depending on their training condition. Those with condition training 2 (the connected path) were more likely to use an NRAT on testing. This is statistically significant from a chi-square test ($p < .001$).

When controlling for testing conditions, a logistic regression estimated NRAT as a function of training conditions, testing efficiency instruction, and the presence of the testing level's distractor path. A connected angle path in training (i.e., training 2) was a statistically significant positive predictor of using an NRAT on testing. This was consistent across all testing conditions, see Figure 5.

5 Dicsussion

We answered our research questions through statistical analysis on two kinds of scaffolding methods, which are efficiency instruction and prior conditional training level. Our results indicate that explicit instruction and training level could both scaffold learners

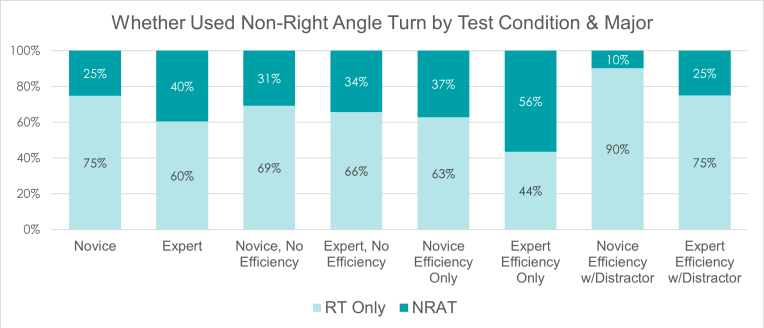

Figure 3: Distribution of using RT and NRAT under different test conditions and majors.

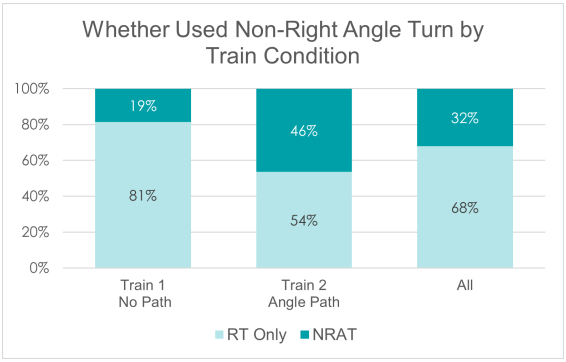

Figure 4: Distribution of using RT and NRAT under different train conditions.

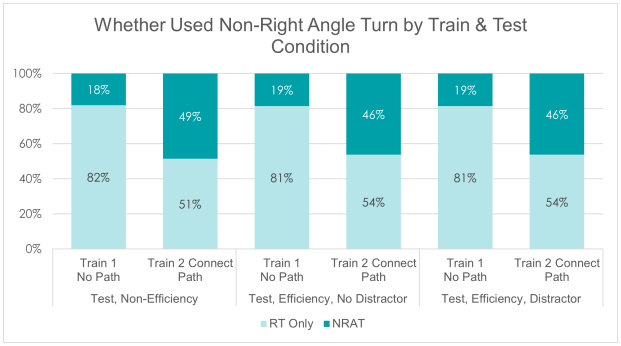

Figure 5: Distribution of using RT and NRAT under different train and test conditions.

yet to different degrees. Our directional results and statistically significant results proved that efficiency instruction scaffold benefits experts more. Similarly, Sinha and Kapur suggested that novice were largely unsuccessful at recognizing the value of scaffolding while experts succeeded [27]. Considering the potential reasons, research has consistently proven that when learners lack prior domain-specific knowledge (i.e., NRAT in our context), they face difficulties when solving well-structured problems [28]. This may further reveal our research question, where the efficiency instruction only suggests learners use as few code as possible, yet does not help them how to do so.

On the other hand, we observed that the difference in instructional scaffolding disappeared when participants were involved in certain prior training levels, where the directly connected NRAT path in training conditions indeed helps learners to perform better than no path in the training condition. Our designed prior training level serves as marking critical features and demonstrating solutions to a task in scaffolding strategy [9], as well as providing structured practice opportunities [29]. Although such scaffolding helps to improve performance by nearly 30% percent, distracting paths at the test level still play an important role. Research suggests several scaffoldings in addition to those involved in our study. Guided attention scaffolding directs learners' attention to the most relevant information in the problem-solving process and minimizes the chances of following distracting paths [30, 31]. Feedback loops indicate regular and timely feedback to help learners stay on course when they interact with distracting paths [32, 33]. More explorations on combinations of these scaffolding are needed.

6 Conclusion

We designed a coding platform, Fox and Field, to determine what kinds of scaffolds could encourage novice coders to perform more like experts in problem-solving tasks. We recruited 106 experts and 115 novices for quantitative comparison. We found that explicit instruction and critical features could scaffold novices to increase code efficiency. We also found that purposefully distracting features could deflect both experts and novices from the most efficient solution. Our findings could provide insights into learning environment development to scaffold learning.

Acknowledgments

We would like to acknowledge the University Development Research Fund from University of Delaware for supporting this research and the research assistants from The Rutherford Lab, for their work in collecting and cleaning data.

References

[1] National Research Council, Division on Engineering, Physical Sciences, Computer Science, Telecommunications Board, and Committee for the Workshops on

Computational Thinking. 2010. Report of a workshop on the scope and nature of computational thinking. National Academies Press.
[2] David Weintrop, Elham Beheshti, Michael Horn, Kai Orton, Kemi Jona, Laura Trouille, and Uri Wilensky. 2016. Defining computational thinking for mathematics and science classrooms. Journal of science education and technology 25, 2016, 127-147.
[3] Jeannette M Wing. 2006. Computational thinking. Commun. ACM 49, 3, 2006, 33-35.
[4] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, *et al.* 2009. Scratch: programming for all. Commun. ACM 52, 11, 2009, 60-67.
[5] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada, Vol. 1. 25.
[6] Caren A Pacol, Maria Mercedes T Rodrigo, and Christine Lourrine S Tablatin. 2024. A Comparative Study of High and Low Performing Students' Visual Effort and Attention When Identifying Syntax Errors. In International Conference on Human-Computer Interaction. Springer, 77-94.
[7] Philipp Kather, Rodrigo Duran, and Jan Vahrenhold. 2021. Through (tracking) their eyes: Abstraction and complexity in program comprehension. ACM Transactions on Computing Education (TOCE) 22, 2, 2021, 1-33.
[8] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. 2023. A think-aloud study of novice debugging. ACM Transactions on Computing Education 23, 2, 2023, 1-38.
[9] David Wood, Jerome S Bruner, and Gail Ross. 1976. The role of tutoring in problem solving. Journal of child psychology and psychiatry 17, 2, 1976, 89-100.
[10] Seymour Papert. 1980. Mindstorms: children, computers, and powerful ideas.
[11] National Research Council, Division on Engineering, Physical Sciences, Computer Science, Telecommunications Board, and Committee for the Workshops on Computational Thinking. 2011. Report of a workshop on the pedagogical aspects of computational thinking. National Academies Press.
[12] Xiaodan Tang, Yue Yin, Qiao Lin, Roxana Hadad, and Xiaoming Zhai. 2020. Assessing computational thinking: A systematic review of empirical studies. Computers & Education 148, 2020, 103798.
[13] Annie Li, Madeline Endres, and Westley Weimer. 2022. Debugging with stack overflow: Web search behavior in novice and expert programmers. In Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training. 69-81.
[14] Sarah Jessup, Sasha M Willis, Gene Alarcon, and Michael Lee. 2021. Using eye-tracking data to compare differences in code comprehension and code perceptions between expert and novice programmers. In Proceedings of the 54th Hawaii International Conference on System Sciences. 114-123.
[15] Salwa Aljehane, Bonita Sharif, and Jonathan Maletic. 2021. Determining differences in reading behavior between experts and novices by investigating eye movement on source code constructs during a bug fixing task. In ACM Symposium on Eye Tracking Research and Applications. 1-6.
[16] Phyllis J. Beck. 2023. IDE-based learning analytics for assessing introductory programming skill. Theses and Dissertations. https://scholarsjunction.msstate.edu/td/5909
[17] Paul Denny, James Prather, Brett A Becker, Zachary Albrecht, Dastyni Loksa, and Raymond Pettit. 2019. A closer look at metacognitive scaffolding: Solving test cases before programming. In Proceedings of the 19th Koli Calling international conference on computing education research. 1-10.
[18] Chryselle Rego, Enid Montague, *et al.* 2023. The Impact of Feedback Modalities and the Influence of Cognitive Load on Interpersonal Communication in Nonclinical Settings: Experimental Study Design. JMIR Human Factors 10, 1, 2023, e49675.
[19] Monika Mladenović, Žana Žanko, and Ivica Boljat. 2019. Programming misconceptions at the K-12 level. Encyclopedia of education and information technologies, 2019, 1-13.
[20] Q Batiha, N Sahari, N Aini, and N Mohd. 2022. Adoption of visual programming environments in programming learning. International Journal on Advanced Science, Engineering and Information Technology 12, 5, 2022, 1921.
[21] Chen-Hsuan Liao and Jiun-Yu Wu. 2022. Deploying multimodal learning analytics models to explore the impact of digital distraction and peer learning on student performance. Computers & Education 190, 2022, 104599.
[22] Chuen-Tsai Sun, Li-Xian Chen, and Hsiu-Mei Chu. 2018. Associations among scaffold presentation, reward mechanisms and problem-solving behaviors in game play. Computers & Education 119, 2018, 95-111.
[23] Dave Kruti. 2022. Comparison of flow-based versus block-based programming for naive programmers. Ph.D. Dissertation. Toronto Metropolitan University.
[24] Ünal Çakıroğlu and Suheda Mumcu. 2020. Focus-Fight-Finalize (3F): Problem-solving steps extracted from behavioral patterns in block based programming. Journal of Educational Computing Research 58, 7, 2020, 1279-1310.
[25] Jan Cuny, Larry Snyder, and Jeannette M Wing. 2010. Demystifying computational thinking for non-computer scientists. Unpublished manuscript in progress, referenced in http://www. cs. cmu. edu/˜ CompThink/resources/TheLinkWing. pdf, 2010).
[26] Shuchi Grover and Roy Pea. 2013. Computational thinking in K-12: A review of the state of the field. Educational researcher 42, 1, 2013, 38-43.
[27] Tanmay Sinha and Manu Kapur. 2021. Robust effects of the efficacy of explicit failure-driven scaffolding in problem-solving prior to instruction: A replication and extension. Learning and Instruction 75, 2021, 101488.
[28] Namsoo Shin, David H Jonassen, and Steven McGee. 2003. Predictors of well-structured and ill-structured problem solving in an astronomy simulation. Journal of research in science teaching 40, 1, 2003, 6-33.
[29] Randall C Gale, JustinaWu, Taryn Erhardt, Mark Bounthavong, CaitlinMReardon, Laura J Damschroder, and AmandaMMidboe. 2019. Comparison of rapid vs in-depth qualitative analytic methods from a process evaluation of academic detailing in the Veterans Health Administration. Implementation Science 14, 2019, 1-12.
[30] Muanchan Utthavudhikorn, Kittitouch Soontornwipast, *et al.* 2023. An exploration of teachers' experience with using scaffolding techniques. Ph.D. Dissertation. Thammasat University.
[31] Zhenhua Xu, Ana Zdravkovic, Matthew Moreno, and Earl Woodruff. 2022. Understanding optimal problem-solving in a digital game: The interplay of learner attributes and learning behavior. Computers and Education Open 3, 2022, 100117.
[32] Tom Beckmann, Eva Krebs, Patrick Rein, Stefan Ramson, and Robert Hirschfeld. 2021. Shortening Feedback Loops in a Live Game Development Environment. In 2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 1-5.
[33] Mathias Mejeh, Livia Sarbach, and Tina Hascher. 2024. Effects of adaptive feedback through a digital tool-a mixed-methods study on the course of self-regulated learning. Education and Information Technologies, 2024, 1-43.