Investigating Patterns of Iterative Code Changes in the VEXcode VR environment by Combining Heuristic Patterns with Hidden Markov Modeling

Hyeongjo Kim, Luc Paquette, Jennie Lee
University of Illinois at Urbana-Champaign
{hk61, lpaq, jl294}@illinois.edu

ABSTRACT

Computational problem solving (CPS) is a core component of computational thinking and is increasingly supported through game-based learning environments. However, understanding learners' problem-solving processes in such environments remains challenging especially when they engage students in open-ended problem-solving with no unique solution pathway. In this study, we examine learners' CPS processes in VEXcode VR, a block-based robotics programming environment, by investigating iterative code modification behaviors. Drawing on in situ observations, we first identified three heuristic problem-solving patterns: step-by-step, relying on fortune, and strategy change. These patterns were operationalized using a code change score and modeled using a categorical Hidden Markov Model (HMM), enabling the representation of latent problem-solving states and their temporal dynamics. The results demonstrate strong alignment between the latent states inferred by the HMM and the heuristic patterns identified by researchers, supporting both the validity and interpretability of the model. The interpretations of the identified patterns highlight the importance of considering the unique modality of game-based environments when understanding learners' computational problem-solving processes.

Keywords

Computational Problem Solving; Game-Based Learning; Hidden Markov Models; Heuristic Modeling; Sequential Analysis

1. INTRODUCTION

Computational problem solving (CPS) is recognized as a core competency in contemporary education, particularly in technology-rich and game-based learning environments [2]. CPS involves a set of cognitive processes, including problem formulation, algorithmic thinking, iterative testing, and debugging, through which learners construct and refine solutions to complex problems [10]. To support CPS, game-based learning (GBL) has been used as scaffolding for computational problem solving through providing immediate and continuous feedback, reducing cognitive load, and simulating real-world systems [2, 9, 10].

Hyeongjo Kim, Luc Paquette, and Jennie Lee. Investigating Patterns of Iterative Code Changes in the VEXcode VR environment by Combining Heuristic Pat-terns with Hidden Markov Modeling. In Anthony Botelho, Maria Mercedes T. Rodrigo, Adish Singla, Hiroaki Ogata, Hyojeong So, and Young Hoan Cho (eds.) Proceedings of the 19th International Conference on Educational Data Mining, Seoul, Republic of Korea, June, 2026, pp. 689–693. International Educational Data Mining Society (2026).

https://doi.org/10.5281/zenodo.21039798

Because both games and coding fundamentally involve problem solving [2], prior studies examining problem-solving patterns in game-based learning provide useful conceptual frameworks for understanding CPS. However, as Matcha and colleagues [5] noted, students' strategic behaviors can vary across learning environments even when guided by the same theoretical framework, largely due to differences in modality. For example, while systematic evaluation is often considered a more advanced and desirable strategy than trial-and-error in traditional problem-solving contexts [4, 7], repeated cycles of trial and error are not merely preliminary strategies but can constitute a core process of solution development, testing, and debugging in CPS contexts [2, 8]. Therefore, in CPS environments, examining how learners iteratively modify their code during trial-and-error processes provides a direct and meaningful way to understand their problem-solving strategies.

Therefore, in this study, we investigate the CPS processes of students using VEXcode VR, a block-based programming environment in which students must program a robot to perform different tasks. Tasks generally do not have one unique solution and may involve randomness which impacts the final outcome of executing a solution. More specifically, we examine how learners iteratively modify their solutions in the VEXcode VR environment and explore how these iterative changes can be modeled to reflect different problem-solving processes. Insights about the students' code change behaviors were obtained through in-situ observations and review of screen-capture video to identify three heuristic problem-solving patterns (step-by-step, relying on fortune, and strategy change) which were modeled using a categorical Hidden Markov Model (HMM), enabling us to capture CPS processes in a game-based learning context while accounting for its unique affordances.

2. COMPUTATIONAL PROBLEM SOLVING IN GAME-BASED LEARNING

Computational problem solving (CPS) refers to a logical and abstract reasoning process for addressing problems, involving understanding a problem, devising a solution plan, and evaluating outcomes through programmed procedures [2, 10]. CPS is considered a core component of computational thinking (CT), reflecting the centrality of problem solving in CT definitions. To support CPS and CT, a learning-by-doing approach has been widely adopted as a constructionist alternative to direct instruction [2, 6]. While this approach aligns with the exploratory nature of CPS, it poses challenges for novice learners who often lack sufficient knowledge and skills in both computation and problem solving to effectively address ill-structured problems [3]. As a result, learners may struggle to engage productively without appropriate guidance.

To address this gap, research has emphasized the importance of scaffolding, particularly through game-based learning (GBL) environments. Games provide several affordances that support CPS. First, they enhance motivation by offering immediate and continuous feedback during problem solving [1]. Second, games sometimes employ designs to reduce cognitive load such as block-based or card-based coding interfaces, allowing learners to focus on the logical structure of solutions without being hindered by syntax and semantic errors [2]. Third, games enable simulations of complex real-world systems, such as space or physical laws, thereby situating problem solving within meaningful and interactive contexts [1, 2].

Beyond these affordances, both games and CPS fundamentally center on problem solving [2]. In game environments, players either attempt to solve predefined challenges or construct their own goals in open-ended contexts, utilizing available resources within the system. In CPS contexts, these resources correspond to programming systems and interfaces, which serve as tools for constructing and testing computational solutions. This structural alignment suggests that GBL environments function not only as motivational tools but also as spaces where CPS processes are actively enacted.

Based on this alignment, studies have proposed integrated frameworks that connect CPS processes in GBL environments [2, 8]. They commonly include identifying problems, designing and developing solutions, testing and debugging the solutions. Each stage is associated with specific CT practices [8]. For example, in the identifying problem stage, learners engage in decomposition, abstraction, and generalization to break down and simplify problems. These stages provide a structured lens for understanding CPS, but understanding how learners enact these processes in practice has led researchers to examine learners' behaviors through log data.

Previous studies have examined learners' problem-solving processes through log data to reveal how they develop and test solutions. For example, in Zoombinis, a mathematics problem-solving game, researchers have identified structured problem-solving processes and investigated how specific behaviors correspond to different stages, as well as how these behaviors can be modeled to represent sequential progress and strategies [4, 7]. Although the identified strategies such as trial-and-error and systematic testing in problem solving are also relevant to CPS, their roles may differ depending on the learning context. Matcha and colleagues [5] demonstrated that students' strategic behaviors can vary across learning environments even when guided by the same theoretical framework, largely due to differences in modality. CPS environments present a distinctive modality in which learners construct solutions through code and iteratively test and revise them. In the context of Zoombinis, systematic evaluation is often considered a more advanced and desirable strategy than trial-and-error, and transitions from trial-and-error to systematic evaluation are treated as indicators of improved problem-solving competence [4, 7]. In contrast, within CPS contexts, repeated cycles of trial and error are not merely preliminary strategies but constitute a core process of solution development, testing, and debugging [2, 8]. Therefore, in CPS environments, examining how learners iteratively modify their code during trial-and-error processes provides a more direct and meaningful way to understand their problem-solving processes and strategies.

3. CONTEXT

3.1 VEXcode VR

The study was conducted using the VEX Robotics programming environment, specifically VEXcode VR, a block-based platform designed to foster computational thinking and problem-solving skills. Within this environment, learners control a virtual robot by creating sequences of commands, including movement, rotation, and control structures, to navigate simulated spaces. The system provides immediate visual feedback through simulation, enabling students to iteratively test, debug, and improve their solutions. Our study employed a custom-made version of VEXcode VR which automatically logged all student interactions, such as code creation and execution, supporting fine-grained analysis of learners' problem-solving processes over time.

The study was conducted across three different playgrounds (simulated 3 dimensional spaces), each presenting distinct problem-solving goals. In Coral Reef Cleanup, students programmed a robot to collect and remove trash in an underwater environment within limited battery constraints, emphasizing efficient navigation and resource management. In Castle Crasher, learners developed programs to knock down castle structures and transport debris into designated areas, requiring strategic planning. In Rover Rescue, goal prioritization was highlighted as students programmed a rover with the goal to maximize mission duration through three strategy choices: collecting minerals, neutralizing enemies, and returning minerals to base. . After executing each code, the system assigns a score based on predefined rules (e.g., the amount of trash collected) that involve randomness, meaning that an identical code could yield different outcomes. Across all playgrounds, tasks were open-ended, allowing multiple solution paths and encouraging iterative refinement.

3.2 Data

A total of 18 students participated in a one-and-a-half-hour session per day over three days. Students played on one playground each day in the following order: Coral Reef Cleanup – Castle Crasher - Rover Rescue. Each day, students first received a brief introduction to the environment and core programming commands before engaging in individual coding activities. During each session, learners iteratively developed and executed their programs, receiving immediate feedback from the simulation and refining their solutions accordingly. While collaboration and discussions were allowed, each student constructed their own solution. Instructors helped students mostly when they were asked by students. Throughout the process, all programming actions and execution events were logged, enabling the analysis of learners' iterative problem-solving behaviors over time.

Two types of data were collected for this study: screen capture recordings and programming log data. Screen capture recordings without sounds were obtained during participants' interactions with the VEXcode VR programming environment. In addition, detailed coding logs were automatically recorded. These logs include various user actions, such as adding, editing, and moving blocks, as well as executing programs (e.g., "Run project"). All interaction data were stored in JSON format, enabling fine-grained analysis of learners' computational problem-solving processes over time. The Institutional Review Board (IRB) at University of Illinois Urbana-Champaign reviewed the data collection protocol and made the determination that it did not meet criteria for "human-subject research" since all collected data was fully anonymized with no identifier linking the data to participants.

4. MODELING ITERATIVE CODE CHANGES

The modeling process integrated human insights with data-driven analysis to identify patterns of iterative code changes in VEXcode VR. First, in situ observations and reviews of screen-capture recordings of problem-solving activities were used to identify meaningful code-change patterns while accounting for contextual factors and the environment's unique modality. Then, we defined how these patterns could be operationalized using the VEXcode VR interaction log data. Finally, a Hidden Markov Model (HMM) was fit to categorize student problem-solving interactions as one of three identified code change patterns.

4.1 In situ Observations

Learners' behavioral patterns were heuristically identified by two researchers (first and third authors) who participated in the camp as facilitators and had direct experience observing and interacting with learners during the activities. Drawing on their domain knowledge and in situ experiences, they collaboratively articulated patterns they perceived in students' iterative code-change processes.

Screen capture recordings were revisited as needed to refine these interpretations. However, this study did not employ systematic video coding procedures, as the primary goal of this phase was not to generate exhaustive behavioral coding, but rather to elicit and formalize contextual understandings of learner behaviors. Through this process, we identified three overarching patterns that were consistently observed across the three playgrounds: step-by-step, relying on fortune, and strategy change.

The step-by-step pattern is characterized by students making small, incremental changes to their code, executing it, and then modifying it based on the last state of the VEX robot in each execution.

The relying on fortune pattern is characterized by repeated execution of the same code without any modification. Unlike typical iterative testing where code is refined between runs, students repeatedly run an identical program in an attempt to obtain better outcomes. This behavior is closely related to the design of the VEXcode VR environment, where stochastic elements can produce different outcomes. For example, the Coral Reef Cleanup playground randomizes when and where pieces of trash fall into the playground, whereas in the Castle Crasher playground, physics simulations determine how the castle structure reacts to collisions with the robot, which can result in greatly differing outcomes when repeatedly executing the same program.

Within this context, students appear to rely on randomness to achieve higher scores rather than refining their code. This pattern often emerged after a period of step-by-step iteration. It may reflect a temporary shift in strategy, such as taking a cognitive break, exploring outcome variability, or attempting to maximize performance from existing solutions. After several repetitions, students typically returned to modifying their code.

The strategy change pattern involves substantial revisions to the structure of the code, including modifying or removing large portions of previously constructed solutions. Compared to the other patterns, this behavior was observed relatively infrequently. We observed that students were generally reluctant to abandon or significantly alter existing code, which may explain the rarity of this pattern. Although we observed instances of strategy change, it was uncertain from our observation what constitutes an appropriate threshold for distinguishing it from incremental (step-by-step) modifications.

4.2 Operationalization of Identified Patterns

The patterns we heuristically identified were used as an initial foundation for pattern recognition, serving as a rough yet meaningful basis for subsequent systematic analysis. Operationalizing these patterns through a data-driven approach enables the development of models grounded in empirical evidence. To operationalize these patterns, it was necessary to translate them into quantifiable representations. As the identified patterns primarily relate to how students modify their code across program execution, we computed a code change score to capture the degree of difference between consecutive code submissions.

We computed the code change score by combining a position-weighted change rate with a length-based adjustment. First, differences between two consecutive code sequences were evaluated at each sequential position. A binary change indicator was assigned for each position, taking the value of 1 if the blocks differed (or if one sequence was shorter), and 0 otherwise. To reflect the structural importance of earlier code segments, we applied an exponentially decaying weight to each position, giving higher importance to changes occurring at the beginning of the sequence. The weighted changes were then normalized by the sum of positional weights to obtain a position-weighted change rate.

To account for the overall size of the program, this rate was further adjusted using a length-based weight (square root). Finally, the code change score was computed as the product of the position-weighted change rate and the normalized length weight, resulting in a measure that captures both the structural impact and the extent of code modifications. This formulation allows the score to reflect both the structural significance of changes (via positional weighting; $e^{-position}$) and the magnitude of edits (via length adjustment; $\sqrt{length}$).

An initial examination of the distribution of code change scores suggested the presence of distinct behavioral tendencies (Figure 1). The histogram revealed a substantial proportion of zero-change cases, along with a concentration of small changes and relatively fewer large changes. Specifically, over 40% of the observations showed no change in code, and even when changes occurred, they were typically minimal. This distribution provides preliminary support for the existence of the three heuristic patterns: minimal or no change (relying on fortune), incremental change (step-by-step), and substantial change (strategy change).

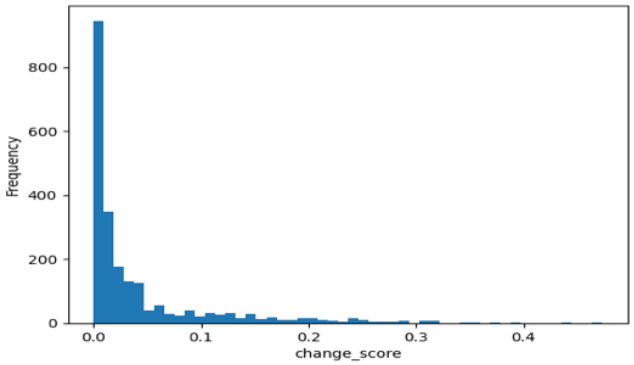

Figure 1. Distribution of code change scores across consecutive submissions.

Importantly, our observations emphasized that individual code change values should not be interpreted in isolation. Rather, patterns emerge over sequences of actions. For example, a sequence consisting predominantly of no-change trials with a single instance

of code modification would still be interpreted as relying on fortune, rather than as a mixture of patterns. This highlights a limitation of approaches that rely solely on observed transitions, such as Markov chains.

To address this limitation, we adopt Hidden Markov Modeling (HMM), which enables patterns to be conceptualized as hidden states that generate observable behaviors (i.e., code changes). From this perspective, the code change score serves as an observable indicator, while the underlying problem-solving patterns are inferred as hidden states. By modeling temporal sequences in this way, HMM provides a more nuanced representation of learner behaviors, consistent with instructors' interpretations.

To align with the heuristic categories and facilitate interpretation of the HMM results, we constrained the model by discretizing the continuous variable into categorical states and employing a categorical HMM instead of a continuous HMM. Code change scores were discretized into three levels: no change (0), small change (below the 95th percentile), and large change (above the 95th percentile). The 95th percentile corresponds to substantial modifications affecting a large portion of the code; in practice, these cases typically involve changes to more than 85% of blocks when the program contains at least 10 blocks, indicating a clear restructuring of the program.

4.3 Categorical Hidden Markov Modeling

Each student's interaction sequence was represented as a series of discrete observations derived from code changes categories between consecutive runs. Using the operationalization described in Section 4.2, each action was categorized into one of three observable types: no change, small change, and large change.

The HMM assumes that the observed sequence is generated from an underlying sequence of latent states, where transitions between states follow a first-order Markov process. Emission probabilities represent the likelihood of observing each type of code change given a latent state. Researchers should set the number of hidden states while the hyperparameter can be decided by comparing various models with different numbers of hidden states. However, this study first sets the number of hidden states at three to implement the heuristic patterns we observed.

When fitting the model, to address the issue of local optima due to random initialization, each model was trained with 100 random restarts, and the solution with the highest log-likelihood was selected. This procedure also allows us to assess the robustness of the identified patterns and reduce the likelihood of reporting coincidental results. The result shows that the average consistency across the 100 restarts is 84%, indicating robustness. We averaged the emission and transition probabilities to set the final model (Table 1).

Table 1. Emission and Transition Table for HMM model.

Emission			
Hidden state	No change	Small change	Large change
0	0.13	0.86	0.00
1	0.25	0.59	0.15
2	0.82	0.17	0.00
Transition			
0	0.94	0.00	0.04
1	0.03	0.91	0.05
2	0.09	0.05	0.84

Inspection of the emission probabilities enabled interpretation of the latent states in terms of observable behaviors. One latent state was characterized by a high probability of no change, corresponding to the relying on fortune pattern. Another state predominantly emitted small changes, aligning with the step-by-step pattern. The third state was associated with large changes, reflecting the strategy change pattern. Importantly, this state was not limited to large code modifications alone; it also included small adjustments following major changes, as well as immediate re-execution without modification. These results demonstrate that the latent states inferred by the HMM correspond closely to the heuristic patterns articulated by researchers.

After implementing our heuristic model into the HMM, we examine whether the model can be expanded and elaborated by relaxing the constraints we imposed for HMM. To determine the optimal number of hidden states, we fitted models with two to five states and evaluated them using the Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC). The results were as follows: 2 states (AIC = 3136.84, BIC = 3176.82), 3 states (3064.28, 3144.23), 4 states (3031.67, 3163.01), and 5 states (3034.39, 3228.55). BIC was minimized at three states, whereas AIC favored a four-state model. These results suggest that additional patterns may emerge by relaxing the constraints of the HMM, such as increasing the number of hidden states or modeling the code change score as a continuous variable rather than a categorical one.

5. DISCUSSION AND CONCLUSION

This study aimed to understand computational problem solving (CPS) processes in VEXcode VR, a game-based learning (GBL) environment, by examining how learners iteratively engage in testing and revision across trials using a code change score. Three dominant CPS patterns were identified in this environment: step-by-step, relying on fortune, and strategy change. These patterns reflect both general problem-solving strategies and the unique affordances of coding in a game-based context.

The step-by-step pattern reflects how learners in VEXcode VR construct partial solutions, test them, and incrementally refine their code. This behavior is closely tied to the environment, which allows a code to be executed and evaluated at intermediate stages, thereby providing built-in scaffolding for iterative development.

The relying on fortune pattern highlights game-specific characteristics. Due to stochastic elements, identical code executions can yield different outcomes, leading learners to repeatedly run the same code to achieve better results. While step-by-step reflects formative, incremental refinement, relying on fortune resembles summative evaluation under varying conditions without immediate modification. These patterns are also shaped by the ill-structured nature of the problem space. With no single correct solution and evaluation based on relative performance (e.g., scores), learners must interpret feedback and make decisions under uncertainty.

The strategy change patterns are consistent with general problem-solving process. This represents a substantial shift in approach, involving major modifications or entirely new solutions, and aligns more closely with general problem-solving processes than with game-specific mechanics.

Taken together, our findings show how the specific characteristics of the VEXcode VR environment, such as stochastic playground outcomes, executable intermediate states, and ill-structured tasks, shaped the way learners develop strategies. This aligns with Matcha and colleagues [5], which highlights that different learning modalities can give rise to distinct strategies even under the same

theoretical framework. For example, while the observation of step-by-step and strategy change patterns aligned with common CPS processes, the strategy of relying on fortune appeared to be unique to VEXcode VR, where the repeated execution of the same program may produce different outcomes. This underscores the importance of understanding the unique modalities of both CPS and game-based environments when analyzing and interpreting learners' behaviors.

However, several limitations should be addressed in future research. First, the sample size of 18 participants limits the generalizability of the findings. Therefore, future studies should include larger and more diverse datasets. Second, the identified heuristic patterns require further validation through methods such as systematic video coding and inter-rater reliability analysis. Third, the use of the 95% threshold for defining strategy change should be theoretically justified rather than relying solely on a data-driven criterion. Finally, to determine whether these patterns are educationally meaningful, future research should examine their relationships with learning outcomes such as task performance, code efficiency, and learning gains.

6. ACKNOWLEDGMENTS

This study was supported by the National Science Foundation (NSF; # 2229612). Any conclusions expressed in this material do not necessarily reflect the views of the NSF. We would also like to thank the entire INVITE project team for their support throughout this project.

7. ACKNOWLEDGEMENT OF GENERATIVE AI USE

This study used generative AI to assist with data analysis programming and draft revision. No study data were directly uploaded to AI systems, and all AI-generated content was carefully reviewed by the first author.

8. REFERENCES

[1] Gao, L., Fabricatore, C. and Lopez, M.X. 2019. Game features in inquiry game-based learning strategies: A systematic synthesis. *Proceedings of the European Conference on Gamesbased Learning* (2019), 854–62.

[2] Jiang, X., Harteveld, C., Huang, X. and Fung, A.Y.H. 2019. The computational puzzle design framework: a design guide for games teaching computational thinking. *Proceedings of the 14th International Conference on the Foundations of Digital Games* (San Luis Obispo California USA, Aug. 2019), 1–11.

[3] Jonassen, D.H. 1997. Instructional design models for well-structured and III-structured problem-solving learning outcomes. *Educational Technology Research and Development*. 45, 1 (Mar. 1997), 65–94. https://doi.org/10.1007/BF02299613.

[4] Liu, T. and Israel, M. 2022. Uncovering students' problem-solving processes in game-based learning environments. *Computers & Education*. 182, (June 2022), 104462. https://doi.org/10.1016/j.compedu.2022.104462.

[5] Matcha, W., Gašević, D., Ahmad Uzir, N., Jovanović, J., Pardo, A., Lim, L., Maldonado-Mahauad, J., Gentili, S., Pérez-Sanagustín, M. and Tsai, Y.-S. 2020. Analytics of Learning Strategies: Role of Course Design and Delivery Modality. *Journal of Learning Analytics*. 7, 2 (Sept. 2020), 45–71. https://doi.org/10.18608/jla.2020.72.3.

[6] Papert, S.A. 2020. *Mindstorms: Children, computers, and powerful ideas*. Basic books.

[7] Rowe, E., Almeda, M.V., Asbell-Clarke, J., Scruggs, R., Baker, R., Bardar, E. and Gasca, S. 2021. Assessing implicit computational thinking in Zoombinis puzzle gameplay. *Computers in Human Behavior*. 120, (July 2021), 106707. https://doi.org/10.1016/j.chb.2021.106707.

[8] Teng, K. and Chung, G.K.W.K. 2025. Measuring Children's Computational Thinking and Problem-Solving in a Block-Based Programming Game. *Education Sciences*. 15, 1 (Jan. 2025), 51. https://doi.org/10.3390/educsci15010051.

[9] Turchi, T., Fogli, D. and Malizia, A. 2019. Fostering computational thinking through collaborative game-based learning. *Multimedia Tools and Applications*. 78, 10 (May 2019), 13649–13673. https://doi.org/10.1007/s11042-019-7229-9.

[10] Wing, J.M. 2006. Computational thinking. *Communications of the ACM*. 49, 3 (Mar. 2006), 33–35. https://doi.org/10.1145/1118178.1118215.