International Journal of Evaluation and Research in Education (IJERE)
Vol. 15, No. 1, February 2026, pp. 424~435
ISSN: 2252-8822, DOI: 10.11591/ijere.v15i1.35542

Evaluating the impact of game development-based learning on programming skill acquisition and student engagement

Nurul Hazlina Noordin[1], Gajendran Karunanithi[2]
[1]Universiti Malaysia Pahang Al-Sultan Abdullah (UMPSA) STEM Lab, Faculty of Electrical Electronics Engineering Technology, Universiti Malaysia Pahang Al-Sultan Abdullah, Pekan, Malaysia
[2]International Business Machines (IBM) Malaysia, Petaling Jaya, Malaysia

Article Info

Article history:

Received Apr 19, 2025
Revised Dec 9, 2025
Accepted Jan 1, 2026

Keywords:

Digital embodiment
Educational approach
Game development learning
Learner engagement
Programming education

ABSTRACT

This study examines the challenge of limited engagement and conceptual understanding among school children in introductory programming education. To address this, the research evaluates the impact of game development-based learning using the slider game module. The objective is to assess how developing a simple game can support programming skill acquisition and enhance learner engagement. A total of 310 participants, aged 11 to 17, were selected through purposive sampling from various schools involved in programming classes. The research design included pre- and post-test assessments, demographic analysis, and Likert-scale surveys to gauge learner perceptions. Quantitative analysis was conducted using paired sample t-tests and descriptive statistics. The results show improvements in learners' coding abilities and increased confidence and motivation across all age groups. The findings highlight the effectiveness of game development-based learning as a pedagogical approach for teaching programming in an engaging and impactful way.

Corresponding Author:

Nurul Hazlina Noordin
Universiti Malaysia Pahang Al-Sultan Abdullah STEM Lab
Faculty of Electrical Electronics Engineering Technology, Universiti Malaysia Pahang Al-Sultan Abdullah
26600 Pekan, Pahang, Malaysia
Email: hazlina@umpsa.edu.my

1. INTRODUCTION

In programming education, there is a growing interest in using game development to support the acquisition of coding skills and promote computational thinking [1]. Despite the increasing importance of programming literacy in the 21st century, many novice learners—particularly school children—struggle to grasp programming fundamentals [2]. Traditional methods of teaching programming often focus heavily on syntax and rote memorization, which can hinder learners' motivation, engagement, and ability to solve problems creatively. A disconnect between abstract programming logic and real-world application continues to present a barrier, especially for younger or first-time coders. While many studies emphasize the benefits of computational thinking, few have examined how game development can be structured as a core method for introducing these concepts effectively in early programming education [3].

Game development learning provides an experiential, hands-on approach that allows learners to create, test, and interact with their own coded projects [4]. It aligns with constructivist learning theory, cognitive apprenticeship (CA), and situated learning theory, which together emphasizes active engagement, real-world context, and social interaction in knowledge construction. Constructivism focuses on active learning and scaffolding, while CA emphasizes modeling, coaching, and fading. On the other hand, situated learning theory promotes contextual and authentic problem-solving. The slider game module is designed

based on the combined principles of these three theories. Figure 1 illustrates how the module supports each theory through strategies such as modeling game logic (CA), scaffolded Python tasks (constructivism), and applying code to real game design contexts (situated learning). Games are naturally motivating, goal-oriented, and often require logical problem-solving, making them suitable tools for teaching abstract programming concepts in a concrete and interactive manner. By focusing on the development of a slider game, learners not only apply programming concepts like loops, conditions, and variables, but also develop algorithmic thinking in a fun and meaningful context. This study investigates whether such an approach could provide a viable alternative to traditional programming instruction for school learners.

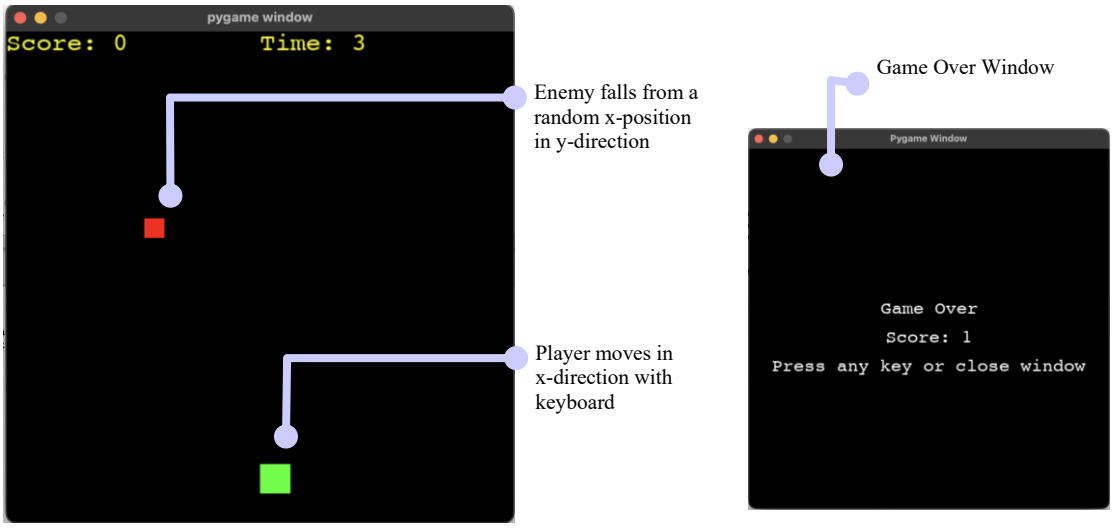

Figure 1. Slider game

One of the persistent challenges in teaching programming lies in the disconnect between programming syntax and the underlying logic it represents [5]–[11]. The programming-by-demonstration paradigm was developed to address this gap, offering learners a more intuitive entry point into coding through observation and replication [12]. Reinfelds [13] further emphasizes that programming should be taught as an engineering discipline, grounded in problem-solving and conceptual understanding rather than syntax memorization. Efforts to address these challenges have included tiered scaffolding and constructivist-based interventions that leverage physical computing tools such as Arduino and project-based modules. These interventions, as well as game development-based ones, align with the CA model by offering demonstration (game demo), coaching (game building support), and reflection (debugging and replay). These interventions have demonstrated success in enhancing students' engagement and digital skill development through hands-on learning environments. Similar observations have been reported in the broader constructivist and physical computing literature, which highlights the effectiveness of such approaches in promoting active learning and science, technology, engineering and mathematics (STEM) competencies [14]–[17].

Further studies suggest that programming shares cognitive processes with natural languages [18]. However, many novice learners struggle with understanding dynamic programming concepts due to static instructional methods, overemphasis on syntax, and mismatches between teaching strategies and student learning styles [19]. This study introduces learners to programming through the process of developing a slider game, using a game development learning approach. This method goes beyond conventional instruction by fostering algorithmic thinking, logical reasoning, and creative problem-solving through hands-on coding activities. Different from visual-based tools such as Scratch, the slider game module is text-based and bridges visual logic with actual Python scripting. Table 1 compares the slider game module with Scratch and similar tools across five key aspects, which include level of abstraction, interactivity, scaffolding mechanism, cognitive load, and customizability. The module offers lower abstraction and higher transfer potential than Scratch, while maintaining beginner accessibility.

Table 1. Comparison between slider game module and other tools

Feature	Slider game module (this work)	Scratch [20]	HTML5 Canvas [21], [22]
Level of abstraction	Low (Python-based)	High (block-based)	Moderate
Type of interactivity	Real-time keyboard/game logic	Drag-and-drop	Event-driven
Scaffolding mechanism	Embedded hints and syntax walkthrough	Visual blocks	Manual
Cognitive load	Moderate	Low	High
Customizability	High (edit Python code)	Moderate	High

The main objectives of this study are: first is to assess the extent to which the slider game module improves learners' understanding of Python programming, including the application of coding concepts and problem-solving skills. Secondly is to examine how the game development learning approach affects learners' motivation, engagement, and ability to solve real-world programming tasks. Besides, this research addresses two primary questions:

i) To what extent does the slider game module improve learners' understanding of Python programming, application of coding concepts, and problem-solving skills?
ii) How does the game development learning approach influence learners' motivation, engagement, and ability to tackle real-world programming tasks?

Programming education is evolving through the incorporation of engaging methods such as game development, which aim to enhance students' understanding of coding and computational thinking. This section presents existing literature on the challenges in programming education, the potential of game development learning. The review also considers how game development learning affects both cognitive skill acquisition and learner engagement. Identifying programming difficulties early and responding with effective strategies is crucial to improving learning outcomes and reducing dropout rates [23]. Programming languages are often abstract, and learners may find it difficult to move from understanding syntax to applying programming logic [24], [25]. Additional challenges include the use of static materials to teach dynamic content and the lack of alignment between instructional methods and learners' preferences [26], [27]. These factors can hinder skill retention and development [28].

Skill transfer is a key component of effective programming education. Zhao *et al*. [29] highlighted the value of educational game development in supporting learners' understanding of abstract programming ideas. Personalization and well-designed in-game instructions help bridge knowledge gaps and improve skill transfer by aligning learning content with individual needs. Near transfer involves applying knowledge in similar contexts, while far transfer relates to using learned skills in different or novel situations. Lee *et al*. [30] emphasized the importance of near transfer for K–12 students through the use of tools like PETIS, which offer real-time feedback to reinforce skills. In contrast, far transfer promotes broader application of coding knowledge across languages and problem domains [18].

Situative transfer theory focuses on the relationship between learning context and knowledge application [31]. In programming education, it supports creating environments that simulate authentic problem-solving scenarios [32]. Effective transfer is more likely when instruction emphasizes principles over memorization and fosters collaborative learning, coaching, and reflection. Models such as problem-based learning (PBL), community of practice (CoP), CA, and game development learning integrate these elements [31], [33]. Chichekian *et al*. [34] proposed a problem-solving model for knowledge transfer in programming. Other frameworks include the classification of language transition concepts [35] and analogical transfer models that support programming pedagogy [36]. Project-driven learning has also been shown to promote deeper conceptual understanding [37].

The ability to abstract core programming concepts and apply problem-solving skills is central to skill transferability [38]–[40]. Learners who can understand the underlying principles of programming can effectively adapt their skills to new languages and coding tasks. This concept highlights the importance of teaching coding as a problem-solving process [29]. The use of abstraction and structure in programming styles is important for expertise and the benefits of object-oriented programming [41]. In a similar context, the use of puzzles as a hands-on approach to teaching abstraction and problem-solving concepts in information technology (IT) education is explored [42]. It was concluded that solving puzzles not only simplifies abstract ideas but also encourages students to develop and employ effective problem-solving strategies, ultimately improving their grasp of IT concepts. A pattern-oriented instruction (POI) and abstract data type (ADT)-oriented instruction were explored in Haberman and Muller [43], where practical methods were deployed for teaching problem-solving processes and abstraction. The study highlighted the value of conceptual models and research tools in analyzing students' abstraction skills.

Innovative pedagogical strategies, such as blended learning and flipped classrooms, are gaining traction in programming education. Blended learning provides flexibility and accommodates diverse learning styles [44], while flipped classrooms allow students to learn theory independently and engage in practical

tasks during class [45]. Programming-by-demonstration simplifies learning by showing the logic behind programming steps, helping students grasp abstract concepts more effectively [46]. Cognitive studies show similarities between programming and language acquisition, highlighting the importance of structured thinking and problem-solving [47]. Study by Fedorenko *et al*. [18] found that programming shares processing mechanisms with natural language, suggesting that programming education can build on students' existing cognitive abilities. This connection implies that the cognitive skills employed in learning programming holds similarities to those employed in language acquisition and comprehension. Recognizing these parallels is a critical aspect of addressing the cognitive dimensions of programming education. It allows educators to leverage students' existing cognitive abilities, such as pattern recognition and abstraction, to facilitate their understanding of programming concepts.

The cognitive challenges encountered by both students and instructors in the process of programming teaching are explored in Elçiçek and Karal [48]. These challenges emphasized the need to identify and address cognitive factors that may impede effective programming education. This awareness is crucial for developing pedagogical strategies that align with students' cognitive processes and for providing instructors with the tools to support their students more effectively. The integration of game development into programming courses led to an improvement in students' comprehension of programming concepts and increased their engagement with the subject [49]. This sets the potential of game development as effective educational tools in programming instruction.

The embodiment of game development in programming education can have a holistic impact on students' overall experience and engagement [50]–[52]. Other research supports using learning analytics to personalize game-based instruction and integrating game elements to enhance comprehension [32]. Despite this, few studies have evaluated game development learning, especially in text-based environments, as a constructivist strategy for early-stage programming education. This study addresses that gap by introducing a structured module based on the constructivist theory. It focuses on skill transfer in particular the differentiation between near and far transfer. Near transfer involves applying game logic in similar Python tasks, such as conditions or scoring mechanics, while far transfer includes using skills from the game module to create new applications which include game modification scripts.

2. RESEARCH METHOD

This study involved 310 participants from diverse educational backgrounds. The goal was to examine how engaging in game development supports programming skill acquisition and learner engagement. Participants included students aged 12 to 22 and a small number of teachers. All participants were enrolled in programming workshops facilitated by the Universiti Malaysia Pahang Al-Sultan Abdullah (UMPSA) STEM Lab. No prior programming experience was required to participate, although basic computer literacy was expected. The sample was intentionally broad to reflect a range of educational levels and learning experiences. A mixed-methods approach is adopted, combining quantitative and qualitative techniques to obtain a comprehensive understanding of learning outcomes and learner experiences. The design was quasi-experimental, using pre-test and post-test comparisons to measure programming skill improvement.

2.1. Study context and sample selection

The study was conducted in a series of hands-on Python programming workshops held at the UMPSA STEM Lab between March and December 2024. A purposive sampling technique was used to ensure a mix of primary, secondary, and tertiary-level students, as well as a small group of school teachers. Participants were invited through school collaboration programs, online registration, and university outreach events. The inclusion criteria were minimal, where participants had to have basic digital literacy but no prior programming experience.

2.2. Data collection instruments

The study utilized a combination of pre-test and post-test assessments, demographic questionnaires, and post-activity Likert-scale surveys. The pre-test and post-test, consisting of 26 multiple-choice questions, were designed to measure participants' understanding of basic Python concepts, including syntax, control structures, variable usage, and debugging. The Likert-scale survey, administered post-activity, consisted of seven items measured on a 5-point scale to assess participants' engagement, confidence, and perceived learning. To ensure reliability, Cronbach's alpha was calculated using SPSS, with a strong internal consistency score (α=0.924). Semi-structured interviews were also conducted with a subset of participants to gather qualitative insights about the learning experience and challenges encountered.

2.3. Research procedures

Each workshop followed a consistent structure. The first is to what extent does the slider game module improve learners' understanding of Python programming, application of coding concepts, and problem-solving skills. Secondly followed with pre-test to assess baseline programming knowledge. Next one is instructional sessions using the slider game module with live coding, demonstrations, and hands-on practice. Further step is completion of coding tasks to apply learned concepts in developing the game. In the fifth step is post-test and survey to evaluate learning gains and learner perceptions. Move on to next step which is about interviews after sessions for voluntary participants and the final step involved facilitating the activities using the Thonny IDE, with sessions conducted in computer labs under guided instruction and individual support.

2.4. Data analysis

Quantitative data from pre-test and post-test assessments were analyzed using descriptive statistics and inferential statistics, including paired-sample t-tests to measure learning gains. Effect size (Cohen's d) was calculated to assess the magnitude of learning improvements. Qualitative data from semi-structured interviews were transcribed and thematically analyzed using an inductive coding approach to uncover recurring themes related to engagement, motivation, and challenges. Triangulation between quantitative and qualitative data was employed to increase the trustworthiness of findings.

2.5. Game development learning as a pedagogical approach: the slider game as a case study

The slider game was used as a scaffolded, game development-based learning activity designed to support novice learners in acquiring fundamental programming skills. This module provided participants with an interactive and hands-on experience in which they gradually built a playable game using Python. The learning process was structured around five core programming principles: event handling, control structures, variable usage, function design, and logical debugging. The game development process emphasized event handling through keyboard-controlled player movement, control structures such as loops and conditionals to manage object behaviors (such as enemy movement), variables and functions to handle scoring, game timers, game logic, and, logical thinking and debugging, especially in managing collisions and runtime behavior.

Participants followed a tiered instructional design, beginning with small tasks such as displaying objects on screen, followed by user-controlled movement, and later incorporating complex game logic such as scoring, timers, and collision detection. This tiered approach aligned with the constructivist learning framework, enabling students to build upon prior knowledge while solving new programming challenges. Facilitators provided live demonstrations and real-time feedback during coding, encouraging trial and error and collaborative learning. From an engagement perspective, the slider game module was designed to encouraged creativity, problem-solving, and immediate feedback. These characteristics are known to increase learner motivation and persistence in programming education [9], [11]. Quantitative performance metrics, from pre-post test scores and practical coding task scores, were complemented by qualitative reflections collected through interviews and surveys to assess how well the pedagogical method contributed to learners' programming acquisition.

2.6. Game mechanics

An illustration of the slider game is shown in Figure 1. In this game, players control a character represented by a colored square within a rectangular game window. The objective is to navigate the player character, using keyboard inputs, to hit (collide) the incoming enemy obstacles descending from the top of the screen. These enemies, depicted as red squares, move in a continuous downward motion, creating a challenging environment for the player. As the game progresses, players must maneuver their character to collide with the descending enemies. Upon collision detection, the player's score increases, rewarding successful interactions and encouraging continued gameplay. Additionally, a timer function tracks the elapsed time since the start of the game. If the game runs for more than 30 seconds, it triggers a game over condition, prompting the display of a "game over" message along with the player's final score. This intuitive gameplay mechanic was designed not just for entertainment, but also as an embedded learning tool to reinforce coding logic. It allowed learners to practice core programming constructs such as keyboard events, collision detection, loops, timers, and variable tracking.

The slider game learning objectives, as shown in Table 2, focuses on establishing the core competencies in introductory programming education. These objectives are carefully aligned with fundamental programming concepts, which are data structures, functions, loops, conditional statements, and variables. List data structures were used to store object positions and manage the enemy entities, allowing learners to grasp indexing and iteration. These structures are vital for managing player and enemy properties,

as well as scores within the game environment. Another concept is functions, which is incorporated in tasks such as initializing the game window and defining player movements. Through these activities, learners develop proficiency in creating and utilizing functions to streamline code organization and execution.

In programming, control statements like iterative, conditional, and sequential operations are crucial for directing the flow of a program [53]–[55]. In the slider game module, these concepts are integrated into various aspects such as player movements, enemy property management, enemy falling movements, and game timing coordination. By engaging with loop structures embedded within these functionalities, learners gain hands-on experience in iterating over game elements, thereby deepening their comprehension of loop functionalities and reinforcing their programming skills. Throughout the module, learners encounter conditional statements that dictate player and enemy behaviors based on user input or game conditions. It enables learners to implement decision-making logic within their code, helping them to develop logical thinking and problem-solving strategies.

Another concept that has been emphasized thoroughly in this module is variables. Variables are implicitly embedded into various tasks, serving as fundamental components for storing and manipulating game properties such as player positions, enemy attributes, and game scores. Learners were guided to use print statements and variable tracing techniques to observe how values change during game execution. By working with variables, learners develop an understanding of data manipulation and management within programming contexts [56]. This hands-on experience enables learners to grasp the importance of variables in storing dynamic data and facilitates their transition to more complex programming tasks.

Table 2. Slider game learning objectives and programming concepts

Game properties	Learning objectives	Fundamental learning concepts
Create game window	Understanding the setup of game environments	Basics of function definition and program initialization
Describe player	Defining and customizing player attributes	Utilizing Python operators and expressions for character design
Player movements	Implementing player controls and interactions	Application of control flow concepts for character movement
Enemy properties	Defining enemy characteristics and behaviors	Applying control flow principles to manage enemy actions
Enemy falling movement and scoring system	Managing enemy descent and tracking player progress	Reinforcement of control flow and introduction to variable usage
Game timer	Coordination of game events and timing	Understanding the importance of timing in game development
Timing systems	Maintaining smooth gameplay flow	Reinforcing the concept of timing and event coordination
Assignment 1	Encouraging creative problem-solving in game design	Application of various programming concepts in practical challenges

3. RESULTS AND DISCUSSION

This section presents the results of the quantitative and qualitative analyses and discusses the implications of the game development learning approach, as implemented through the slider game module. The focus is on assessing its impact on programming skill acquisition and learner engagement across different age groups and educational backgrounds. The statistical analyses were conducted using SPSS v26.0.

3.1. Overall learning gain

A total of 310 participants completed both the pre-test and post-test assessments. These tests measured understanding of Python syntax, control structures (loops and conditionals), variable usage, and debugging. As shown in Table 3, the overall average learning gain across participants was +8.4%, with pre-test scores improving from a mean of 35.4% to 43.8% in the post-test, improvement across the sample. A paired-sample t-test was conducted revealing a significant improvement ($t=17.45$, $p<0.001$). The calculated Cohen's $d=0.99$ indicates a large effect size, highlighting the practical significance of the intervention. Though average, this gain is significant given the short-term nature of the intervention and the wide range of prior programming experience among participants. Studies on similar short-form interventions report comparable results, typically within the 5–10% range [29], [57]. This supports the effectiveness of scaffolded, hands-on tasks in producing measurable learning outcomes even over a brief duration.

3.2. Gender-based comparison

An analysis of learning gains by gender indicated that both male and female participants benefited from the module is presented in Table 4, with male participants showing a slightly higher average improvement (9.3%) compared to females (6.9%). Despite the difference in gain, female participants had

higher overall pre- and post-test scores. These findings suggest that while prior exposure or readiness may have influenced baseline scores, the game development approach effectively engaged and supported both groups in learning. This aligns with previous research highlighting the importance of inclusive and equitable practices in computing education [33].

Table 3. Pre-test and post-test scores

Evaluation item	Min score	Max score	Mean	Std. Deviation
Score (Post %)	13.8	72.4	43.8	12.1
Score (Pre %)	0.0	67.9	35.4	14.1
Score (Post – Pre %)	-0.5	48.3	8.4	8.5

Table 4. Pre-test and post-test scores by gender

Gender	Score (Post %)		Score (Pre %)		Score (Post – Pre %)	
	Mean	Std	Mean	Std	Mean	Std
Female (*n*=113)	47.4	11.9	40.5	13.5	6.9	6.6
Male (*n*=197)	41.8	11.6	32.4	13.6	9.3	9.4

3.3. Age-based analysis

Table 5 presents the pre-test and post-test scores, as well as the score differences among participants grouped by age. Participants aged 12–15 showed the highest average score gain of +8.7%, suggesting that the intervention was particularly impactful for younger learners who may have had limited prior exposure to programming. This may also be attributed to their lack of prior exposure to programming. On the other hand, participants aged 23–25 also showed a significant improvement of an average +10.5% score gain, possibly due to greater maturity in logical reasoning and task ownership. This pattern reflects both near transfer for younger learners [30] and elements of far transfer for older learners adapting programming logic to new contexts [18].

Effect size analysis supports these findings. Cohen's d values across all age groups ranged from 0.946 to 1.491, indicating large effects above the 0.8 threshold for a large effect. The 23+ age group demonstrated the largest effect (d=1.491), highlighting the practical significance of the intervention for mature learners. These results align with constructivist learning theory, where learners actively build knowledge through meaningful and engaging tasks. The slider game module, structured around incremental problem-solving and application of core concepts, such as loops, variables and conditionals, likely facilitated cognitive engagement that contributed to learning gains. This is consistent with situated learning and transfer of learning frameworks, where contextual, scaffolded tasks support meaningful knowledge transfer and retention [30], [31].

Table 5. Pre-test and post-test scores by age group with effect sized (Cohen's d)

Age group	Score (Post %)		Cohen's d	Score (Pre %)		Score (Post – Pre %)	
	Mean	Std		Mean	Std	Mean	Std
12 – 15 (*n*=207)	45.1	11.7	0.946	36.4	14.5	8.7	9.2
16 – 17 (*n*=48)	42.6	12.3	1.007	34.3	13.4	8.3	8.3
18 – 22 (*n*=37)	35.7	10.8	1.289	29.2	11.6	6.5	5
23 + (*n*=37)	48.9	10.5	1.491	38.4	11.9	10.5	7

3.4. Engagement and perception analysis

Post-activity Likert-scale surveys, as shown in Table 6, reveals strong participant agreement with positive statements related to learning and engagement. The mean response scores ranged from 3.98 to 4.20 (on a 5-point scale), indicating that most participants found the activity motivating, enjoyable, and helpful in improving their understanding of programming. To evaluate the reliability of the Likert scale responses, Cronbach's alpha was calculated. The resulting coefficient of 0.924 indicates a high level of internal consistency among the survey items. This reliability suggests that the survey effectively measured participants' perceptions and attitudes regarding the game development learning experience. The effect size between pre- and post-test scores was calculated using Cohen's d, resulting in a value of 0.82, which indicates a large effect size. This suggests that the learning intervention had a strong impact on participants' understanding and confidence in programming. Participants agreed with statements related to increased confidence in programming and enjoyment in learning through game development. These results are

consistent with prior work [58], which demonstrated that PBL contribute to improved learner satisfaction, motivation, and confidence in programming contexts.

Table 6. Descriptive statistics for Likert scale responses

	Item	Mean	Std	Variance
A1	I understand better about Python programming after attending the course	4.08	0.908	0.825
A2	I learn about the basics of physical computing after attending the course	4.2	0.906	0.821
A3	I am more interested in programming and physical computing after attending the course	4.15	0.922	0.85
A4	I am able to explore innovative solutions using Python programming after attending the course	4.1	0.923	0.851
A5	I am confident to program in Python after attending the course	3.98	0.953	0.909
A6	This course has provided opportunities for me to improve my technical skills in preparation for my school projects	4.19	0.838	0.703
A7	I would recommend this course to my colleagues	4.19	0.895	0.802

3.5. Demographic trends in perception (age and gender)

Analysis of Likert data by age and gender revealed consistently positive perceptions across all demographic groups, as shown in Table 7. Female participants in the 16–17 and 18–22 age groups reported higher interest and confidence scores. Overall, participants in the 12–15 group showed the highest overall perception scores, particularly in engagement and enjoyment. This supports the idea that game development learning is well-suited for diverse learners, offering an accessible and motivating environment regardless of prior experience.

Table 7. Impact of slider module course across age groups and genders

Age group gender	12-15				16-17				18-22				23+			
	F (n=79)		M (n=128)		F (n=14)		M (n=34)		F (n=12)		M (n=25)		F (n=7)		M (n=10)	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
A1	4.06	1.13	3.94	0.88	4.29	1.07	4.41	0.5	4.58	0.52	4.04	0.74	4.14	1.07	4.2	0.42
A2	4.19	1.01	4.14	0.93	4.29	0.83	4.44	0.5	4.5	0.52	4.16	0.8	4.43	0.54	4	1.16
A3	4.08	1.13	4.14	0.9	4.43	0.51	4.26	0.79	4.33	0.89	4.08	0.91	4.43	0.54	4.1	0.32
A4	4.1	1.01	4.01	0.96	4.29	0.47	4.29	0.76	4.42	0.9	4.08	0.76	4.43	0.54	4	0.82
A5	3.81	1.17	3.93	0.92	4.5	0.52	4.15	0.82	4.33	0.89	3.92	0.81	4.29	0.49	4	0.82
A6	4.14	0.98	4.11	0.9	4.57	0.51	4.41	0.5	4.58	0.52	4.04	0.74	4.29	0.49	4.2	0.42
A7	4.08	1.1	4.11	0.84	4.36	1.08	4.59	0.5	4.67	0.49	4.08	0.76	4.29	0.49	4.1	1.2

3.6. Correlation between perception and learning outcomes

Pearson correlation analysis revealed a strong positive relationship between understanding Python programming and learning about physical computing (r=0.684, p<0.001), as well as between interest in programming and physical computing (r=0.654). This suggests that participants who were more interested in programming were also more likely to engage with physical computing concepts. A moderate positive correlation was also observed between participants' confidence in programming and their perception of the course as an opportunity to enhance technical skills (r=0.568, p<0.001). Furthermore, participants who felt more confident in their programming abilities were more likely to recommend the approach of learning programming through game development to others (r=0.532, p<0.001).

3.7. Qualitative insights from interviews

Semi-structured interviews revealed that participants valued the hands-on nature of learning programming through game development, particularly the creative freedom involved in designing and building their own game. Many reported a sense of accomplishment and satisfaction from completing a playable game and felt more confident in coding concepts after the session. However, some participants highlighted difficulties in debugging and implementing scoring logic, indicating that more scaffolding or guided support in these areas may be beneficial in future iterations of the module.

3.8. Contextualizing the learning gain

While the average learning gain was 8.4%, this result should be interpreted in context. The intervention was conducted as a short-duration, and many participants had little to no prior experience with Python programming. Learning gains of this magnitude are consistent with findings from similar beginner-level interventions in game-based programming education [29]. As shown in Table 5, learners in the 12–15 and 23+ age groups achieved gains exceeding 10%, suggesting that the scaffolded game development learning approach is especially impactful for both early-stage learners and more mature

participants. These results support the idea that short, constructivist-based programming interventions can yield meaningful outcomes when aligned with learner needs and cognitive readiness.

4. CONCLUSION

This study evaluated the impact of game development learning, implemented through the slider game module, on programming skill development and learner engagement. The intervention involved 310 participants from a wide range of age groups and backgrounds, including secondary and pre-university learners. Key findings include consistent improvements in programming skills across all age groups, with the most significant gains seen in participants aged 12–15 and 23+. The gender-based analysis further demonstrated that while male participants showed slightly higher learning gains, female participants achieved higher overall test scores, confirming the inclusive and balanced nature of the module. The Likert-scale responses reflected a strong consensus regarding the effectiveness of the course, with participants reporting increased understanding, interest, and confidence in Python programming. The high internal reliability (Cronbach's alpha=0.924) validates the consistency of these responses. Correlation analysis confirmed strong relationships between programming understanding, technical skill development, and participant confidence. This research addressed two key objectives, which are to explore the impact of game development learning on programming skills, and to examine learner perceptions and experiences. These objectives were achieved through a mixed-methods approach, integrating demographic data, pre- and post-test comparisons, survey analyses, and qualitative feedback from interviews. The findings support game development learning as a constructivist and scaffolded strategy for introductory programming education.

This work aligns with global computational thinking and coding education frameworks, including the ACM K–12 Computing Curricula and UNESCO STEM competencies, by nurturing creativity, problem-solving, and learner autonomy. Feedback from facilitators emphasized the module's ease of use and adaptability across different learner levels, highlighting its potential for scalability within formal and informal learning settings. Based on these outcomes, we recommend that the slider game module be integrated into school lesson plans or extracurricular digital clubs, where sustained engagement can extend learning beyond the workshop setting.

FUNDING INFORMATION

This work has been supported by the UIC231519/RDU232710 grant, Universiti Malaysia Pahang Al-Sultan Abdullah (UMPSA), IBM Malaysia and the Ministry of Higher Education Malaysia (MoHE).

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Nurul Hazlina Noordin	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
Gajendran Karunanithi	✓									✓		✓		✓

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

INFORMED CONSENT

We have obtained informed consent from all individuals included in this study.

ETHICAL APPROVAL

Ethical approval for this study was obtained from the Institutional Research Ethics Committee (IREC 2023-173). Participants were informed of the study's purpose and provided written consent. Participation was voluntary, and all responses were anonymized to protect privacy.

DATA AVAILABILITY

The data that support the findings of this study are available on request from the corresponding author, [NHN]. The data, which contain information that could compromise the privacy of research participants, are not publicly available due to certain restrictions.

REFERENCES

[1] P. Vostinar, "MakeCode Arcade: Interesting environment for programming 2D games," in *2021 IEEE World Conference on Engineering Education (EDUNINE)*, 2021, pp. 1–6, doi: 10.1109/EDUNINE51952.2021.9429132.

[2] L. Hu, "Programming and 21st century skill development in K-12 schools: a multidimensional meta-analysis," *Journal of Computer Assisted Learning*, vol. 40, no. 2, pp. 610–636, 2024, doi: 10.1111/jcal.12904.

[3] T. C. Hsu, S. C. Chang, and Y. T. Hung, "How to learn and how to teach computational thinking: suggestions based on a review of the literature," *Computers and Education*, vol. 126, pp. 296–310, 2018, doi: 10.1016/j.compedu.2018.07.004.

[4] A. Palmquist, J. Alves, R. Baranyi, R. Munkvold, V. Carvalho, and E. Oliveira, "Blended realities - higher education student reflections on acquiring skills for game creation in a project-based- and blended learning environment," *Games: Research and Practice*, vol. 3, no. 1, pp. 2–26, Mar. 2024, doi: 10.1145/3704414.

[5] S. Muggleton, "Inductive logic programming: issues, results and the challenge of learning language in logic," *Artificial Intelligence*, vol. 114, no. 1–2, pp. 283–296, 1999, doi: 10.1016/s0004-3702(99)00067-3.

[6] B. du Boulay, "Some difficulties of learning to program," in *Studying the Novice Programmer*, E. Soloway and J. Spohrer, Eds., New York: Psychology Press, 1989, pp. 283–299.

[7] J. Jakubovic, J. Edwards, and T. Petricek, "Technical dimensions of programming systems," *The Art, Science, and Engineering of Programming*, vol. 7, no. 3, p. 13, Feb. 2023, doi: 10.22152/programming-journal.org/2023/7/13.

[8] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: a literature review," *ACM Transactions on Computing Education*, vol. 18, no. 1, pp. 1–24, 2017, doi: 10.1145/3077618.

[9] C. S. Cheah, "Factors contributing to the difficulties in teaching and learning of computer programming: a literature review," *Contemporary Educational Technology*, vol. 12, no. 2, p. ep272, 2020, doi: 10.30935/cedtech/8247.

[10] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: a review and discussion," *Computer Science Education*, vol. 21, no. 2, pp. 137–172, 2003, doi: 10.1076/csed.13.2.137.14200.

[11] X. M. Wang and G. J. Hwang, "A problem posing-based practicing strategy for facilitating students' computer programming skills in the team-based learning mode," *Educational Technology Research and Development*, vol. 65, no. 6, pp. 1655–1671, 2017, doi: 10.1007/s11423-017-9551-0.

[12] T. B. Weidmann, S. Thorgeirsson, and Z. Su, "Bridging the syntax-semantics gap of programming," in *Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Nov. 2022, pp. 80–94, doi: 10.1145/3563835.3567668.

[13] J. Reinfelds, "The next generation of programming," in *Networking the Learner*, D. Watson and J. Andersen, Eds., Boston, MA: Springer US, 2002, pp. 967–968, doi: 10.1007/978-0-387-35596-2_105.

[14] N. D. Dat, N. V. Bien, N. T. T. Khuyen, N. T. V. Ha, H. T. T. An, and N. T. P. Anh, "Arduino-based experiments: leveraging engineering design and scientific inquiry in STEM lessons," *International Journal of STEM Education for Sustainability*, vol. 4, no. 1, pp. 38–53, 2024, doi: 10.53889/ijses.v4i1.317.

[15] C. C. Hu, Y. F. Yang, Y. W. Cheng, and N. S. Chen, "Integrating educational robot and low-cost self-made toys to enhance STEM learning performance for primary school students," *Behaviour and Information Technology*, vol. 43, no. 8, pp. 1614–1635, 2024, doi: 10.1080/0144929X.2023.2222308.

[16] H. Y. Lee, T. T. Wu, C. J. Lin, W. S. Wang, and Y. M. Huang, "Integrating computational thinking into scaffolding learning: an innovative approach to enhance science, technology, engineering, and mathematics hands-on learning," *Journal of Educational Computing Research*, vol. 62, no. 2, pp. 431–467, 2024, doi: 10.1177/07356331231211916.

[17] S. L. Martinez and G. Stager, *Invent to learn: making, tinkering and engineering in the classroom*, 2nd ed. Torrance, CA: Constructing Modern Knowledge Press, 2019.

[18] E. Fedorenko, A. Ivanova, R. Dhamala, and M. U. Bers, "The language of programming: a cognitive perspective," *Trends in Cognitive Sciences*, vol. 23, no. 7, pp. 525–528, 2019, doi: 10.1016/j.tics.2019.04.010.

[19] A. Gomes and A. J. Mendes, "Learning to program - difficulties and solutions," in *International Conference on Engineering Education (ICEE)*, 2007, pp. 1–5.

[20] J. Fagerlund, P. Häkkinen, M. Vesisenaho, and J. Viiri, "Computational thinking in programming with Scratch in primary schools: a systematic review," *Computer Applications in Engineering Education*, vol. 29, no. 1, pp. 12–28, 2021, doi: 10.1002/cae.22255.

[21] N. Ahmadi, M. Jazayeri, and A. Repenning, "Engineering an open-web educational game design environment," in *2012 19th Asia-Pacific Software Engineering Conference (APSEC)*, 2012, pp. 867–876, doi: 10.1109/APSEC.2012.88.

[22] G. Stuart, "HTML5 and the Canvas element," in *Introducing JavaScript Game Development: Build a 2D Game from the Ground Up*, G. Stuart, Ed., Berkeley, CA: Apress, 2017, pp. 3–16, doi: 10.1007/978-1-4842-3252-1_1.

[23] J. Figueiredo and F. Garcia-Penalvo, "Teaching and learning tools for introductory programming in university courses," in *2021 International Symposium on Computers in Education (SIIE)*, Sep. 2021, pp. 1–6, doi: 10.1109/SIIE53363.2021.9583623.

[24] M. Fojcik, M. K. Fojcik, S.-O. Høyland, and J. Ø. Hoem, "Challenges in teaching programming," in *Education and New Developments*, Jun. 2022, pp. 160–163, doi: 10.36315/2022v1end034.

[25] B. S. Xia, "A pedagogical review of programming education research: what have we learned," *International Journal of Online Pedagogy and Course Design*, vol. 7, no. 1, pp. 33–42, 2017, doi: 10.4018/IJOPCD.2017010103.

[26] M. F. Yiğit and M. Başer, "Learning difficulties and use of visual technologies in learning to program," *Participatory Educational Research*, vol. spi15, no. 2, pp. 27–34, 2015, doi: 10.17275/per.15.spi.2.4.
[27] S. Xinogalos, "Designing and deploying programming courses: strategies, tools, difficulties and pedagogy," *Education and Information Technologies*, vol. 21, no. 3, pp. 559–588, 2016, doi: 10.1007/s10639-014-9341-9.
[28] J. Bennedsen and M. Caspersen, "Recalling programming competence," in *9th Koli Calling International Conference on Computing Education Research*, 2009, pp. 86–95.
[29] D. Zhao, C. H. Muntean, A. E. Chis, G. Rozinaj, and G. M. Muntean, "Game-based learning: enhancing student experience, knowledge gain, and usability in higher education programming courses," *IEEE Transactions on Education*, vol. 65, no. 4, pp. 502–513, 2022, doi: 10.1109/TE.2021.3136914.
[30] H. Y. Lee, C. J. Lin, W. S. Wang, W. C. Chang, and Y. M. Huang, "Precision education via timely intervention in K-12 computer programming course to enhance programming skill and affective-domain learning objectives," *International Journal of STEM Education*, vol. 10, no. 1, p. 52, 2023, doi: 10.1186/s40594-023-00444-5.
[31] S. Hajian, "Transfer of learning and teaching: a review of transfer theories and effective instructional practices," *IAFOR Journal of Education*, vol. 7, no. 1, pp. 93–111, 2019, doi: 10.22492/ije.7.1.06.
[32] A. L. dos Santos, M. R. de A. Souza, M. Dayrell, and E. Figueiredo, "Exploring game elements in learning programming: an empirical evaluation," in *2018 IEEE Frontiers in Education Conference (FIE)*, Oct. 2018, pp. 1–9, doi: 10.1109/FIE.2018.8658505.
[33] Y. Kafai, C. Proctor, and D. Lui, "From theory bias to theory dialogue: embracing cognitive, situated, and critical framings of computational thinking in K-12 CS education," *ACM Inroads*, vol. 11, no. 1, pp. 44–53, 2020, doi: 10.1145/3381887.
[34] T. Chichekian, J. Trudeau, T. Jawhar, and D. Corliss, "Experimenting with computational thinking for knowledge transfer in engineering robotics," *Journal of Computer Assisted Learning*, vol. 40, no. 2, pp. 859–875, 2024, doi: 10.1111/jcal.12921.
[35] E. Tshukudu, "Towards a model of conceptual transfer for students learning new programming languages," in *Proceedings of the 2019 ACM Conference on International Computing Education Research*, Jul. 2019, pp. 355–356, doi: 10.1145/3291279.3339437.
[36] Y. Kao, B. Matlen, and D. Weintrop, "From one language to the next: applications of analogical transfer for programming education," *ACM Transactions on Computing Education*, vol. 22, no. 4, pp. 1–21, 2022, doi: 10.1145/3487051.
[37] E. C. Miller and J. S. Krajcik, "Promoting deep learning through project-based learning: a design problem," *Disciplinary and Interdisciplinary Science Education Research*, vol. 1, no. 1, p. 7, 2019, doi: 10.1186/s43031-019-0009-6.
[38] R. A. Dixon and R. A. Brown, "Transfer of learning: connecting concepts during problem solving," *Journal of Technology Education*, vol. 24, no. 1, pp. 2–17, 2012, doi: 10.21061/jte.v24i1.a.1.
[39] R. Scherer, F. Siddiq, and B. S. Viveros, "The cognitive benefits of learning computer programming: a meta-analysis of transfer effects," *Journal of Educational Psychology*, vol. 111, no. 5, pp. 764–792, 2019, doi: 10.1037/edu0000314.
[40] R. E. Mayer, "Teaching for transfer of problem-solving skills to computer programming," in *Computer-Based Learning Environments and Problem Solving*, 1992, pp. 193–206, doi: 10.1007/978-3-642-77228-3_9.
[41] D. J. Barnes and D. Shinners-Kennedy, "A study of loop style and abstraction in pedagogic practice," in *Proceedings of the Thirteenth Australasian Computing Education Conference*, 2011, vol. 114, pp. 29–36.
[42] S. E. Cha, D. Y. Kwon, and W. G. Lee, "Using puzzles: problem-solving and abstraction," in *Proceedings of the 8th ACM SIGITE Conference on Information Technology Education*, 2007, pp. 135–140, doi: 10.1145/1324302.1324331.
[43] B. Haberman and O. Muller, "Teaching abstraction to novices: pattern-based and ADT-based problem-solving processes," in *2008 38th Annual Frontiers in Education Conference*, Oct. 2008, pp. F1C–7, doi: 10.1109/FIE.2008.4720415.
[44] A. Alammary, "Blended learning models for introductory programming courses: a systematic review," *PLoS ONE*, vol. 14, no. 9, p. e0221765, 2019, doi: 10.1371/journal.pone.0221765.
[45] L. Zhang and J. Niu, "Research to practice in computer programming course using flipped classroom," in *2022 IEEE Frontiers in Education Conference (FIE)*, Oct. 2022, pp. 1–7, doi: 10.1109/FIE56618.2022.9962430.
[46] O. Graf, S. Thorgeirsson, and Z. Su, "Assessing live programming for program comprehension," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, Jul. 2024, pp. 520–526, doi: 10.1145/3649217.3653547.
[47] R. Gacitúa, M. Diéguez, J. Díaz, and S. Sepúlveda, "A flexible and systematic teaching framework to develop cognitive skills through programming courses," in *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, Nov. 2019, pp. 1–8, doi: 10.1109/SCCC49216.2019.8966409.
[48] M. Elçiçek and H. Karal, "Cognitive problems in the process of programming teaching in higher education: learner-instructor experiences," *Turkish Online Journal of Qualitative Inquiry*, vol. 11, no. 1, pp. 140–160, 2020, doi: 10.17569/tojqi.639139.
[49] J. T. George and M. J. George, "Developing a game in Python," in *Human-Computer Interaction in Game Development with Python: Design and Develop a Game Interface Using HCI Technologies and Techniques*, J. T. George and M. J. George, Eds., Berkeley, CA: Apress, 2022, pp. 139–171, doi: 10.1007/978-1-4842-8182-6_5.
[50] N. Pellas and E. Peroutseas, "Gaming in second life via Scratch4SL: engaging high school students in programming courses," *Journal of Educational Computing Research*, vol. 54, no. 1, pp. 108–143, 2016, doi: 10.1177/0735633115612785.
[51] K. Howland and J. Good, "Learning to communicate computationally with Flip: a bi-modal programming language for game creation," *Computers and Education*, vol. 80, pp. 224–240, 2015, doi: 10.1016/j.compedu.2014.08.014.
[52] G. C. Natucci and M. A. F. Borges, "The experience, dynamics and artifacts framework: towards a holistic model for designing serious and entertainment games," in *Proceedings of the 2021 IEEE Conference on Games (CoG)*, Aug. 2021, pp. 1–8, doi: 10.1109/CoG52621.2021.9619144.
[53] P. A. Buhr, *Understanding control flow*. Cham: Springer International Publishing, 2016, doi: 10.1007/978-3-319-25703-7.
[54] C. A. R. Hoare, "Communicating sequential processes," in *Theories of Programming: The Life and Works of Tony Hoare*, C. B. Jones and J. Misra, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, pp. 306–317, doi: 10.1007/978-3-662-09507-2_19.
[55] M. E. Caspersen and M. Kolling, "STREAM: a first programming process," *ACM Transactions on Computing Education (TOCE)*, vol. 9, no. 1, pp. 1–29, 2009.
[56] J. Sajaniemi and M. Kuittinen, "An experiment on using roles of variables in teaching introductory programming," *Computer Science Education*, vol. 15, no. 1, pp. 59–82, 2005, doi: 10.1080/08993400500056563.
[57] X. Xu, R. Liu, Q. Chen, and H. Yang, "A Digital game-based learning approach to improve students' learning performance of senior high school programming courses," in *Proceedings of the 2023 4th International Conference on Education Development and Studies*, pp. 23–28, 2023, doi: 10.1145/3591139.3591155.
[58] C. S. Chang, C. H. Chung, and J. A. Chang, "Influence of problem-based learning games on effective computer programming learning in higher education," *Educational Technology Research and Development*, vol. 68, no. 5, pp. 2615–2634, 2020, doi: 10.1007/s11423-020-09784-3.

BIOGRAPHIES OF AUTHORS

Nurul Hazlina Noordin holds a Ph.D. in electrical and electronics engineering from the University of Edinburgh in 2013. Currently, she is affiliated with the Universiti Malaysia Pahang Al-Sultan Abdullah (UMPSA), serving as an associate professor in the Faculty of Electrical and Electronics Engineering Technology, and leads UMPSA STEM Lab. Her research focus includes antenna design, FPGA implementation and design, adaptive arrays, beamforming algorithms, and engineering education. She actively collaborates with both local and international agencies, advancing research with institutions including the Ministry of Higher Education Malaysia, Malaysia Digital Economy Corporation, the British Council, UNESCO, and IBM. Her dedication extends to both research and education through her roles in academia and her commitment to STEM outreach programs, especially in nurturing future engineering professionals. She can be contacted at email: hazlina@umpsa.edu.my.

Gajendran Karunanithi is the leader for Corporate Social Responsibility (CSR) at IBM, overseeing initiatives in Vietnam and Malaysia. In this role, he drives strategic social impact programs that align with IBM's commitment to sustainability, education, and digital inclusion. Prior to joining IBM, Gajendran served at the Malaysia Digital Economy Corporation (MDEC), where he was actively involved in shaping national-level digital empowerment programs. With deep expertise in stakeholder engagement, community development, and public-private partnerships, he brings a unique perspective to digital equity and inclusive innovation in Southeast Asia. He can be contacted at email: gajen@ibm.com.