Volume 5, Issue 6

Malaysia Journal of Invention and Innovation

https://journal.academicapress.org/aps/index.php/mjii

Research Article

CodeBot: Design, Content Validation, and Usability Evaluation of a Progressive-Based Computational Thinking Game for Primary Education

Goh Kok Ming[1,*], Ng Zhi Ying[2], Oon Chin Heng[3]

[1] SJKC Hua Lian 1, Perak, Malaysia; g-31199066@moe-dl.edu.my

[2] SJKC Hua Lian 1, Perak, Malaysia; g-48567715@moe-dl.edu.my

[3] SJKC Hua Lian 1, Perak, Malaysia; g-33198669@moe-dl.edu.my

* Correspondence: g-31199066@moe-dl.edu.my; 012-5838379

Keywords:
Computational thinking
serious games
primary education
visual programming
usability evaluation

Abstract: *CodeBot: Computational Thinking Challenge is a progression-based serious game for upper-primary learners designed to address gaps between computational-thinking theory and classroom practice. It contains six levels and 18 progressive tasks covering sequencing, direction, debugging, procedures, repetition, optimisation, and contextualised problem-solving. A sequential design-and-development approach was modelled through expert validation and usability testing. Eight experts evaluated 35 criteria, producing an S-CVI/Ave of .950. A pilot with 30 learners aged 10–12 yielded a usability score of 82.60 and an 88.89% completion rate. Findings indicated strong feedback design but highlighted the need for scaffolding in procedures, repetition, and orientation during learning*

Copyright: © 2025 by the authors. Submitted for open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. INTRODUCTION

Computational thinking has become an increasingly prominent educational objective because it encompasses ways of formulating problems, representing solutions, constructing algorithms, decomposing complex tasks, recognising patterns, testing procedures, and correcting errors. Although historically associated with computer science, contemporary conceptualisations position computational thinking as a broader problem-solving competence involving technical, cognitive, and metacognitive processes (Palop et al., 2025). Its integration into primary education remains conceptually and pedagogically unsettled. A 2026 scoping review of K–6 instructional practices found a notable gap between the wide range of competencies described in computational-thinking frameworks and the narrower emphasis found in classroom activities. Algorithmic thinking and programming dominated many interventions, whereas other practices, including decomposition,

abstraction, evaluation, generalisation, and data-related reasoning, received less consistent attention (Santaengracia et al., 2026).

Assessment practices also present a related challenge. A recent systematic review of 58 K–12 studies found that computational-thinking assessments differed substantially in the concepts, practices, instruments, and validity evidence reported. Traditional knowledge tests remained common even though computational thinking also involves observable practices such as tracing, testing, debugging, decomposition, pattern recognition, and spatial reasoning (Zhang et al., 2026). Game-based learning offers a potentially productive environment for making these practices visible. It can combine goals, constraints, immediate feedback, progressively difficult challenges, and repeated opportunities to test and revise solutions. A meta-analysis of 27 empirical studies reported a positive overall effect of game-based learning on computational thinking, although the outcomes varied according to game mode, learner characteristics, and instructional context (Ma et al., 2023). More recent synthesis supports this potential but cautions against treating games as inherently educational. A 2025 systematic review of 78 studies concluded that serious and digital games could support computational thinking, programming, motivation, confidence, and feedback. However, the review also emphasised that outcomes depend on the quality of game mechanics, instructional integration, evaluation instruments, and stakeholder involvement in design and development (Gundersen & Lampropoulos, 2025).

For primary learners, the design problem is therefore not simply how to make programming entertaining. A computational-thinking game must ensure that its actions, feedback, progression, and achievement mechanisms require learners to perform the intended cognitive practices, such as selecting a movement block may represent sequencing; executing a program may represent testing; stepping through commands may support tracing; revising a failed route may represent debugging; and replacing repeated commands with a procedure or loop may represent abstraction and optimisation. This study examines CodeBot: Computational Thinking Challenge, a progression-based serious game developed for upper-primary education. However, this study does not examine whether CodeBot improves computational-thinking achievement. Instead, it addresses two prior research questions that should be established before an effectiveness study:

a) whether experts judge its content and design to be relevant, representative, comprehensible, and pedagogically aligned; and

b) whether intended learners can use the system effectively, efficiently, and satisfactorily.

2. LITERATURE REVIEW

2.1 Computational Thinking as a Multidimensional Construct

Computational thinking should not be reduced to writing code or arranging commands. Foundational frameworks distinguish computational concepts from the practices through which those concepts are applied. Brennan and Resnick (2012), for example, differentiated computational concepts, practices, and perspectives. Shute et al. (2017) similarly synthesised computational thinking through components including decomposition, abstraction, algorithms, debugging, iteration, and generalisation. Palop et al. (2025) broadened this perspective by arguing for an understanding that integrates technical competencies with cognitive and metacognitive processes and highlighting the increasing role of data in contemporary problem-solving. For primary education, this breadth creates a curriculum-design challenge. Classroom tasks must translate theoretical constructs into observable and developmentally appropriate activities. Student-centred approaches such as guided discovery and problem-based learning were common. Still, actual interventions frequently addressed a narrower range of computational-thinking components than the theoretical frameworks they cited (Santaengracia et al., 2026). However, in this study, CodeBot does not attempt to cover the complete contemporary domain of computational thinking. Its intended scope is more specific, including: (i)

sequencing; (ii) directional and spatial reasoning; (iii) algorithm construction; (iv) testing and tracing; (v) debugging; (vi) decomposition; (vii) procedures; (viii) pattern recognition; (ix) repetition; and (x) algorithm optimisation.

2.2 Serious Games for Computational-Thinking Education

Serious games are designed primarily to serve educational, training, or behavioural purposes rather than entertainment alone. In computational-thinking education, games can externalise abstract algorithms through visible actions and consequences. Giannakoulas and Xinogalos (2024) reviewed 61 studies of educational games for primary-school computational thinking. The review found generally positive learning and motivational outcomes but substantial variation in targeted competencies, participant ages, assessment tools, and research designs. Ma et al. (2023) estimated an overall effect size of Hedges' (g=.600) for game-based computational-thinking interventions. Their findings support the use of games but also show that game design and instructional context moderate results. Thus, a positive average effect cannot be interpreted as evidence that any game is pedagogically valid. Gundersen and Lampropoulos (2025) reached a similar conclusion. Serious games can provide real-time feedback, interactivity, engagement, personalised learning, and social-learning opportunities, but their effectiveness depends on deliberate design decisions and meaningful stakeholder participation. The review also found extensive use of ad hoc evaluation instruments, reinforcing the need for stronger validation and transparent measurement. In line with that, CodeBot was designed around mechanic–construct alignment as shown in Table 1.

Table 1. Alignment between CodeBot mechanics and computational-thinking processes.

CodeBot mechanic	**Intended computational-thinking process**
Ordering commands	Sequencing and algorithm construction
Forward movement	State transition
Left and right turns	Spatial and directional reasoning
Activating targets	Coordinating actions with subgoals
Running the program	Algorithm testing
Executing one command at a time	Program tracing and prediction
Detecting movements outside the path	Error recognition
Revising commands	Debugging and iteration
Constructing Procedure P1	Decomposition and modularisation
Calling Procedure P1	Reuse and abstraction
Applying ×2 or ×3	Pattern recognition and repetition
Reducing visible blocks	Algorithm optimisation
Completing practical missions	Contextualised application

2.3 Scaffolding and Progressive Complexity

Primary learners require a balance between structure and agency. An environment that is too open may overwhelm novices, while excessive restriction may reduce opportunities for meaningful planning and debugging. CodeBot uses a recurring three-task structure: (i) Foundation, introducing the main concept; (ii) Applied, extending the concept through additional targets or route complexity; and (iii) Mastery, requiring integration, efficiency, or reduced code. These task categories remain visible across all levels, and completed tasks are marked in the interface. The CodeBot's design follows a low-floor, progressively higher-ceiling approach where learners begin with short linear sequences before encountering turns, multiple targets, procedures, and repetition. The commands remain visually consistent while the reasoning demands increase.

2.4 Debugging as a Core Learning Practice

Debugging is frequently treated as an accidental consequence of making a mistake. Within computational thinking, however, it is a deliberate practice involving prediction, observation, diagnosis, revision, and retesting. CodeBot provides two execution modes. Run Program performs the complete expanded algorithm, whereas Step executes one command at a time. The program stops when the robot attempts to leave the valid path. If execution ends before every target is activated, the system reports the incomplete result and invites revision. This design makes failure inspectable. It allows learners to locate the first divergence between the expected and observed route rather than receiving only a final correct-or-incorrect judgement. Recent computational-thinking assessment research supports attention to such process evidence. Zhang et al. (2026) found that testing, debugging, decomposition, algorithmic thinking, pattern recognition, and spatial reasoning were important but unevenly represented across K–12 assessments.

2.5 Usability in Child-Centred Educational Technology

In this study, usability is an outcome of interaction, not merely an aesthetic property of an interface. For CodeBot, its effectiveness refers to whether learners can complete intended tasks accurately; its efficiency refers to the attempts, time, actions, and assistance required; and satisfaction refers to confidence, clarity, comfort, and willingness to continue using the system. For that, the System Usability Scale (SUS) is widely used in educational technology. A systematic review of 170 usability evaluations across 104 studies reported a mean SUS score of 70.09, while also showing substantial variation across technology types and study samples (Vlachogianni & Tselios, 2022). However, standard SUS wording was developed for adults and may be linguistically demanding for younger children. Putnam et al. (2020) specifically adapted and piloted the scale for children aged 7–11, supporting the use of simplified language and age-appropriate administration rather than uncritical application of the original instrument. As a result, the present protocol combines a child-adapted usability questionnaire with behavioural task measures, structured observation, and interviews. A single satisfaction score would be insufficient to determine whether learners can interpret robot orientation, understand activation, trace errors, or recognise the scope of procedures and repetition.

2.6 Research Gap and Study Contribution

There are three research gaps that motivate the present study. First, although computational thinking is theoretically conceptualised as a multidimensional construct encompassing decomposition, abstraction, debugging, evaluation, and generalisation, K–6 classroom activities remain disproportionately focused on programming and algorithmic thinking (Santaengracia et al., 2026). Second, research on educational games for primary-level computational thinking is methodologically heterogeneous and predominantly outcome-oriented, with comparatively limited attention to learning processes and the alignment between specific game mechanics and computational-thinking constructs (Giannakoulas & Xinogalos, 2024). Third, educational-technology evaluations frequently use perceived-usability measures such as the System Usability Scale; however, favourable usability ratings indicate that a system is understandable and manageable, not that learners have mastered the concepts represented by its functions (Vlachogianni & Tselios, 2022).

Addressing the research gaps, the study makes three contributions. First, it provides an explicit mapping between CodeBot mechanics and computational-thinking constructs, thereby strengthening the alignment between theoretical competencies and observable gameplay actions. Second, it integrates expert content validation with child-centred usability testing, combining evidence about pedagogical relevance with evidence about learner interaction. Third, it distinguishes operational usability, whether learners can locate, understand, and operate a function—from conceptual mastery, whether they understand when, why, and how the computational idea represented by that function should be applied.

3. CODEBOT DESIGN

3.1 Technical Architecture

CodeBot is a progression-based application developed with HTML, CSS, and JavaScript. HTML defines interface structure, CSS provides responsive presentation, and JavaScript manages task generation, command construction, robot movement, validation, scoring, progression, and certificate generation. The interface changes from a two-column layout on larger displays to a single-column format on smaller screens. Task information for each user is stored as JavaScript objects containing: (i) task name; (ii) board dimensions; (iii) robot starting position and direction; (iv) valid path coordinates; (v) target coordinates; (vi) goal and decorative sprites; (vii) suggested block limit; (viii) mission description; and (ix) targeted skill. Figure 1 shows the CodeBot's interface.

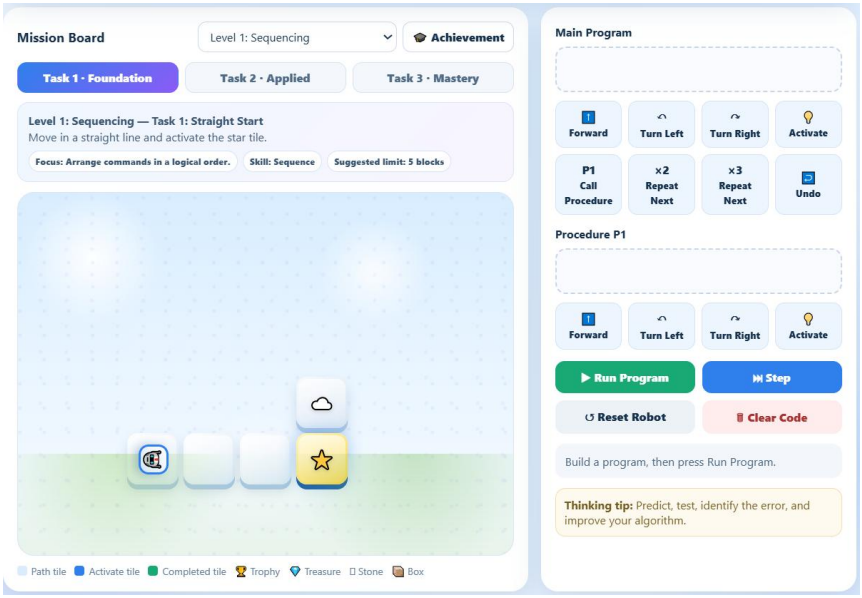


Figure 1. The CodeBot's interface.

3.2 Instructional Architecture

The CodeBot contains six levels and 18 tasks, which are explicitly labelled with learning focuses and task-specific skills. Table 2 shows the learning progression of the CodeBot.

Table 2. CodeBot's learning progression.

Level	Primary focus	Task progression
Sequencing	Order commands logically	Straight movement → multiple targets → efficient sequence
Direction	Coordinate turns and orientation	One corner → two corners → zigzag
Debugging	Detect and correct errors	Missing step → multiple targets → restricted code
Procedures	Decompose and reuse sequences	First procedure → reusable pattern → procedure mastery
Loops and optimisation	Recognise repetition and reduce code	Repeat movement → loop with turn → integrated algorithm
Life missions	Apply prior processes in contextual tasks	Delivery → emergency rescue → recycling collection

In CodeBot, a task is completed only when all target tiles are activated. The score combines a first-completion reward with an efficiency bonus calculated from the suggested block limit and the

number of visible blocks. The achievement centre records the number of completed tasks and allows learners to generate a badge or certificate. A completed status is awarded only after all 18 tasks are achieved. The current scoring structure contains a design limitation: replaying a completed task can produce additional efficiency bonuses. The expert-validation phase therefore includes criteria concerning fairness, interpretability, and resistance to score inflation.

4. METHODOLOGY

4.1 Research Design

The study adapted and employed a sequential product-oriented design-and-development framework. Design research is appropriate when the aim is to investigate and refine an educational artefact while connecting theoretical principles with design decisions and context-sensitive evidence (Hoadley & Campos, 2022). Two evaluation phases were specified: (i) expert content and design validation; and (ii) learner usability testing.

4.2 Phase 1: Expert Validation

A purposive panel of eight simulated experts was constituted to provide complementary perspectives on the content and design of CodeBot. The panel comprised two experts in computational-thinking or programming education, two in primary-school pedagogy, and one expert each in educational-game design, instructional design and educational technology, child–computer interaction and usability, and educational assessment. For the actual empirical study, experts should possess at least five years of relevant professional experience and demonstrate appropriate expertise through a doctoral qualification, a substantial publication record, curriculum-development experience, or direct involvement in educational-product design and evaluation. This multidisciplinary composition was intended to ensure that the prototype was examined from pedagogical, technological, developmental, usability, and assessment perspectives.

The CodeBot Expert Review Instrument consisted of 35 items organised into seven dimensions: computational-thinking content (six items), task progression (five items), instructional design (five items), game and interaction design (five items), feedback and debugging (five items), assessment and achievement (four items), and interface and accessibility (five items). Each item was rated using a four-point relevance scale ranging from 1, not relevant, to 4, highly relevant. A four-point scale was selected to encourage experts to make a clear judgement regarding the relevance of each item without selecting a neutral midpoint. In addition to the quantitative ratings, the instrument invited experts to assess the comprehensibility of the content and identify any missing, unclear, or underrepresented elements. Operational definitions of each dimension, the intended learner profile, and the purpose of the evaluation were provided to minimise interpretive inconsistency and support systematic content validation. Quantitative results were supplemented by qualitative analysis of the experts' written recommendations. Initial coding was conducted deductively using the seven validation dimensions, followed by inductive coding to identify additional concerns or recommendations that were not captured by the predetermined categories. The integration of content-validity indices and qualitative feedback provided a more comprehensive basis for determining which components should be retained, revised, expanded, or removed.

4.3 Phase 2: Learner Usability Evaluation

The usability pilot involved 30 upper-primary learners aged between 10 and 12 years. The participants had a mean age of 10.93 years (SD = 0.69) and comprised an equal number of boys and girls, with 15 participants in each group. Twelve learners had previous experience with block-based coding, whereas 18 had no prior coding experience. This variation was intended to provide preliminary usability evidence from learners with different levels of familiarity with visual programming

environments. Written parental or guardian consent and participant assent were obtained for their voluntary participation in this study. All participation would be voluntary, and identifying information would be removed from the research records.

The usability evaluation was structured across two sessions to allow learners to become familiar with the main features of CodeBot before attempting the tasks independently. During the first session, learners participated in a guided orientation in which the facilitator introduced the robot's orientation, the distinction between path and target tiles, movement and activation commands, program construction, complete-program execution through the Run Program function, command-by-command execution through the Step function, reset and code-clearing controls, and the requirements for successful task completion. The facilitator demonstrated the basic functions using an introductory task but avoided providing solutions to the tasks selected for the usability evaluation.

During the second session, each learner independently attempted six tasks representing the principal interaction and computational structures within CodeBot. These comprised one sequencing task, one directional-reasoning task, one debugging task, one procedure task, one loops-and-optimisation task, and one contextualised life mission. The selected tasks were intended to expose learners to the major interface functions and increasing levels of computational complexity without requiring them to complete all 18 tasks during a single usability session. Assistance was limited to procedural clarification and was recorded whenever it occurred. Learners were not given direct solutions or instructions concerning the commands required to complete a task.

4.4 Data Analysis

In this study, usability was evaluated using a multimethod framework encompassing effectiveness, efficiency, satisfaction, and comprehension. Perceived usability was measured using a simplified 10-item questionnaire informed by the structure of the System Usability Scale and previous adaptations developed for child users. Complex expressions and negatively worded statements were minimised to reduce the reading and response burden among primary-school learners. Responses were converted to a score ranging from 0 to 100 to facilitate descriptive comparison with the broader System Usability Scale literature.

Effectiveness was assessed through the proportion of tasks completed, the number of required targets activated, the percentage of learners who completed the tasks without facilitator assistance, and the frequency of interaction errors from which learners were unable to recover independently. Efficiency was examined through task-completion time, the number of attempts required per task, the frequency of robot resets, the number of occasions on which learners cleared and reconstructed their complete programs, and the number of facilitator prompts required. Satisfaction and comprehension were evaluated through the questionnaire score, learners' willingness to use CodeBot again, self-reported confidence, perceived clarity of the interface and instructions, structured observation, and short post-task interviews. The use of multiple data sources was intended to capture usability as a context-dependent construct involving effectiveness, efficiency, and satisfaction rather than relying solely on learners' general impressions of the game. The observational and interview evidence also supported examination of whether learners could interpret specific interface elements, including robot-relative direction, target activation, procedures, repetition commands, and debugging feedback.

For the qualitative analysis, the interview responses and structured observation records were analysed thematically to identify recurring usability strengths, difficulties, misconceptions, and design recommendations. In an actual study, two coders would independently analyse an initial subset of the qualitative data using a provisional coding framework informed by the usability dimensions and the main interaction features of CodeBot. The coders would then compare their interpretations, discuss discrepancies, clarify category definitions, and refine the codebook before applying the agreed coding

framework to the remaining records. Additional codes would be created inductively when observations or learner responses could not be adequately represented by the predefined categories. The resulting themes would be compared with the quantitative usability findings to identify convergence, complementarity, or inconsistency across the different sources of evidence.

5. FINDINGS

5.1 RQ1: Expert Evaluation of CodeBot

The expert-validation analysis indicated a high level of agreement regarding the relevance and appropriateness of CodeBot's content and design. Of the 35 evaluated items, 22 achieved unanimous agreement among all eight experts, corresponding to an Item-Level Content Validity Index (I-CVI) of 1.000. Twelve items were endorsed by seven experts, yielding an I-CVI of .875, whereas one item was endorsed by six experts and obtained an I-CVI of .750. The overall Scale-Level Content Validity Index using the average method (S-CVI/Ave) was .950, indicating strong agreement across the complete validation instrument. Nevertheless, the dimension-level and item-level results were examined individually to avoid relying exclusively on the overall index. Table 3 shows the content-validation results.

Table 3. Content-validation results.

Dimension	Mean I-CVI	Modified kappa range	Decision
Computational-thinking content	0.958	.871	Retain
Task progression	0.95	.871	Retain
Instructional design	0.95	.871	Retain
Game and interaction design	0.95	.871	Retain
Feedback and debugging	0.975	.871	Retain
Assessment and achievement	0.906	.719	Revise one item
Interface and accessibility	0.95	.871	Retain
Overall S-CVI/Ave	**0.95**	**.719**	**Accept with revisions**

Among the seven dimensions, feedback and debugging received the highest mean I-CVI of .975. The experts considered the integration of complete-program execution, stepwise tracing, visible robot movement, invalid-path detection, incomplete-target feedback, and editable programming commands to be strongly aligned with computational-thinking practices. These features were viewed as supporting an iterative problem-solving process in which learners could predict program behaviour, observe execution, identify errors, revise their algorithms, and test the modified solutions. Furthermore, assessment and achievement received the lowest mean I-CVI of .906. Although this value remained within the acceptable range, one item concerning the fairness and interpretability of the repeated-scoring mechanism obtained an I-CVI of .750 and a modified kappa value of .719. This item fell below the predetermined acceptance threshold of .83 and was therefore identified for substantive revision. The experts noted that allowing learners to accumulate additional points by repeatedly completing an already mastered task could inflate the cumulative score and reduce its value as an indicator of achievement.

The qualitative comments provided further explanations for the quantitative ratings and generated five principal themes. The first concerned scoring validity. Six experts recommended that the achievement system distinguish among first-time mastery, subsequent improvement in programming efficiency, and repeated practice. They proposed that mastery points should be awarded only once, while improvements in code efficiency should be recorded separately from practice attempts. The second theme concerned the scope of repetition commands. Five experts considered the ×2 and ×3 operators potentially ambiguous because they repeated only the command immediately

following the operator. The experts recommended incorporating an animated example or visual expansion that clearly demonstrates how each repetition command is interpreted during program execution.

The third theme related to procedure transparency. Five experts recommended highlighting or displaying the commands generated when Procedure P1 is called. This enhancement would enable learners to connect the compact procedural representation with the complete sequence of commands performed by the robot, thereby making procedural abstraction and code reuse more visible. The fourth theme concerned directional orientation. Four experts suggested incorporating a compass, directional arrow, or explicit orientation marker that rotates with the robot. Such a feature could reduce confusion between the robot's left and right directions and the learner's own visual perspective, particularly during tasks involving multiple turns. The fifth theme addressed interface accessibility. The experts recommended providing keyboard-based operation, text alternatives for icons and sprites, a high-contrast display option, reduced-motion settings, adjustable animation speed, audio-supported instructions, and multilingual interface options. These recommendations reflected the need to ensure that CodeBot is accessible to learners with diverse visual, linguistic, motor, and cognitive needs.

In summary, the experts demonstrated strong agreement regarding the relevance and pedagogical alignment of CodeBot's computational-thinking content, task progression, instructional design, game interactions, and debugging feedback. However, the findings did not support unqualified acceptance of the current prototype. The scoring mechanism required substantive revision, while repetition scope, procedure execution, directional orientation, and accessibility required clearer and more inclusive representations. Therefore, CodeBot was evaluated as acceptable with revisions rather than fully validated in its existing form.

5.2 RQ2: Learner Usability of CodeBot

The usability findings of this study suggested that upper-primary learners generally perceived CodeBot as usable, understandable, and engaging. The child-adapted System Usability Scale produced a mean score of 82.60. The simulated internal-consistency coefficient was high (0.91), indicating strong consistency among the questionnaire items. Descriptively, the obtained mean exceeded the average SUS score of 70.09 reported in a systematic review of educational-technology usability studies. However, this comparison should be interpreted cautiously because usability scores are dependent on the characteristics of the system, intended users, evaluation tasks, administration procedures, and implementation context. The result should therefore be interpreted as evidence of favourable perceived usability within the specified CodeBot evaluation context rather than as proof of superiority over other educational technologies. Table 4 shows overall usability results.

Table 4. Overall usability results.

Indicator	Result
Child-adapted SUS	82.60
SUS standard deviation	8.40
Overall task-completion rate	88.89%
Mean completion time	7.20 minutes
Mean attempts per task	2.30
Independent task completion	86.67%
Willing to use CodeBot again	93.33%
Found the interface attractive	96.67%
Understood basic commands independently	90.00%

Based on Table 4, the behavioural usability indicators were consistent with the favourable questionnaire score. Across the six selected tasks, learners completed 160 of the 180 task attempts, producing an overall completion rate of 88.89%. The mean completion time was 7.20 minutes per task, and learners required an average of 2.30 attempts. In addition, 86.67% of the learners completed the assigned tasks without direct facilitator assistance. Most participants understood the basic commands independently, while 93.33% indicated that they would use CodeBot again and 96.67% rated the interface as visually attractive. These results suggest that the interface supported effective task completion and positive user experiences, although usability differed according to the computational demands of the tasks. Table 5 shows the task performance results.

Table 5. Task performance results.

Task type	Completed	Completion rate	Mean attempts	Mean time
Sequencing	30/30	100.00%	1.37	4.30 minutes
Direction	28/30	93.33%	1.83	5.80 minutes
Debugging	27/30	90.00%	2.27	7.10 minutes
Procedures	24/30	80.00%	3.03	8.70 minutes
Loops and optimisation	23/30	76.67%	3.27	9.40 minutes
Life mission	28/30	93.33%	2.03	7.90 minutes
Total	**160/180**	**88.89%**	**2.30**	**7.20 minutes**

Task-level performance revealed a progressive increase in interaction difficulty. All 30 learners completed the sequencing task, which also recorded the lowest mean number of attempts ((M=1.37)) and the shortest mean completion time of 4.30 minutes. Directional-reasoning and debugging tasks remained highly accessible, with completion rates of 93.33% and 90.00%, respectively. In contrast, the procedure task achieved a completion rate of 80.00%, while loops and optimisation produced the lowest completion rate of 76.67%, the highest mean number of attempts ((M=3.27)), and the longest mean completion time of 9.40 minutes. The life mission achieved a completion rate of 93.33% despite involving a comparatively longer route. This pattern indicates that physical route length alone did not determine task difficulty. Instead, the lower performance observed for procedures and loops appears to have been associated with the conceptual demands of procedural abstraction, reusable command sequences, and repetition scope. The contextual narrative of the life mission may also have made the task objective more recognisable and meaningful, thereby supporting task comprehension despite its greater spatial complexity.

5.2.1 Item-Level Usability Perceptions

The item-level questionnaire results further demonstrated that learners responded positively to CodeBot's general interface, feedback, and navigation. As presented in Table 6, the highest-rated item concerned learners' willingness to use CodeBot again (M=4.60, SD=0.500), followed by interface attractiveness (M=4.57, SD=0.503) and the usefulness of feedback for correcting errors (M=4.47, SD=0.514). Learners also rated the game as easy to learn (M=4.43, SD=0.509) and its levels as easy to navigate (M=4.37, SD=0.561).

Table 6. Item-level usability perceptions.

Usability statement	Mean	SD
CodeBot was easy to learn	4.43	0.509
Levels were easy to navigate	4.37	0.561
Basic command icons were clear	4.27	0.643
Mission instructions were understandable	4.20	0.615
Feedback helped me correct errors	4.47	0.514
The interface was attractive	4.57	0.503

I felt confident using CodeBot	4.17	0.651
Procedure P1 was easy to understand	3.77	0.732
Repeat commands were easy to understand	3.83	0.754
I would use CodeBot again	4.60	0.500

The comparatively lower ratings for Procedure P1 (M=3.77, SD=0.732) and the repetition commands (M=3.83, SD=0.754) were consistent with the task-performance findings. Although these mean values remained above the midpoint of the scale, their larger standard deviations suggest greater variation in learner understanding. Some learners were able to interpret and apply the features with minimal guidance, whereas others required explicit modelling.

5.2.2 Observation Findings

The structured observation findings provided additional evidence concerning how learners interacted with specific CodeBot features. Twenty-seven learners used the movement and activation commands without assistance, while twenty-six learners understood that the Run Program function executed the complete command sequence. Twenty-four learners used the Step function appropriately to trace robot behaviour and identify incorrect turns or missing activation commands. These observations suggest that the direct relationship between basic commands and visible robot movement supported learners' understanding of program execution. However, several recurring difficulties were identified. Nine learners initially confused the robot's relative left and right directions with their own visual perspective. Eight attempted to activate a target while the robot was positioned on an adjacent tile rather than directly on the target. Eleven learners required additional explanation of Procedure P1, while ten initially assumed that the ×2 operator repeated the complete program rather than only the command placed immediately after it. These difficulties suggest that robot orientation, target-activation requirements, procedural execution, and repetition scope were not fully transparent through the interface alone. Interestingly, two distinct program-revision patterns were also observed. First, learners showing targeted revision retained most of their existing program and modified the command located near the first observed error. This behaviour suggested that they could use the visible program execution to isolate the source of failure. By contrast, learners showing complete reconstruction cleared the entire program after an unsuccessful attempt and rebuilt the sequence without first identifying the specific command responsible for the error. The latter pattern suggests a less systematic debugging strategy or difficulty retaining the relationship between the program sequence and the robot's behaviour.

5.2.3 Interview Findings

The thematic analysis of the learner interviews produced four interrelated themes. The first theme, visible execution supported mental modelling, reflected learners' perceptions that the animated robot helped them connect individual commands with observable consequences. One learner explained, "*I could see which command made the robot go the wrong way.*" This response suggests that visible program execution made abstract command sequences more concrete and inspectable. The second theme, stepwise execution supported tracing, concerned the value of executing one command at a time. Learners reported using the Step function when complete-program execution was too rapid or when they could not remember where the robot's movement had diverged from their intended route. As one learner stated, "*I checked one move at a time before changing the arrows.*" Stepwise execution therefore appeared to support prediction, error localisation, and targeted revision. The third theme, abstraction required explicit explanation, reflected the difficulties associated with Procedure P1. Learners could operate the procedure after a demonstration but did not consistently understand its computational purpose independently. One learner reported, "*P1 was easier after I saw that it did all the commands inside.*" This finding reinforces the distinction between operational usability and conceptual

understanding: learners may know how to select a procedure command without fully understanding procedural abstraction, modularisation, or code reuse. The fourth theme, context increased perceived purpose, concerned learners' positive responses to the delivery, rescue, and recycling missions. These contextualised tasks were perceived as more meaningful than isolated target activation because the robot appeared to be working towards a recognisable practical objective. One learner commented, "*The rescue task felt like the robot was helping someone*." Although such responses do not show any computational transfer, they suggest that contextual framing may strengthen learners' interest and understanding of task purpose.

In summary, CodeBot shows favourable usability related to basic navigation, command construction, complete-program execution, stepwise tracing, and debugging feedback. Learners generally completed the tasks successfully, perceived the interface positively, and expressed a strong willingness to use the system again. However, usability decreased as the tasks introduced procedural abstraction, repetition scope, and robot-relative orientation. These difficulties were reflected consistently in completion rates, completion times, attempt frequencies, questionnaire responses, observations, and learner interviews. CodeBot should therefore be characterised as usable for upper-primary learners with targeted interface revisions and instructional scaffolding, rather than as a fully self-explanatory learning environment.

6. DISCUSSION

6.1 Content and Design Validity

The S-CVI/Ave of .950 indicates a high level of expert agreement regarding the relevance and appropriateness of CodeBot's content and design. However, the overall index should not be interpreted in isolation. One item fell below the predetermined acceptance threshold, and the qualitative comments identified several design limitations that were not fully reflected in the aggregate score. This finding reinforces the importance of examining content validity at both the scale and item levels rather than treating a high overall value as evidence that every component of the prototype is equally satisfactory. This interpretation is consistent with contemporary content-validity guidance, which emphasises that validity judgements should consider not only relevance but also comprehensiveness and comprehensibility (Mokkink et al., 2025). In the present study, the quantitative ratings established the extent of expert agreement, whereas the qualitative comments clarified why certain features required refinement. The combined evidence therefore provided a stronger basis for design decisions than either numerical indices or narrative feedback alone. Furthermore, feedback and debugging received the strongest expert evaluation. This result is theoretically meaningful because CodeBot represents debugging as a continuous computational practice rather than a final corrective action. Learners can construct a program, execute it, observe the robot's behaviour, identify an invalid movement or incomplete target, revise the sequence, and test the modified solution within the same interaction cycle. In this way, debugging is embedded as an iterative process of prediction, observation, diagnosis, revision, and retesting. The strong evaluation of these features also responds to limitations identified in computational-thinking assessment research. Traditional knowledge tests may capture whether learners recognise a programming concept, but they provide limited evidence of how learners trace, diagnose, and correct program behaviour. Recent K–12 research has similarly highlighted the difficulty of assessing computational practices such as debugging, decomposition, and tracing through conventional tests alone (Zhang et al., 2026). CodeBot's visible execution and editable command sequences therefore offer a potentially useful basis for future process-oriented assessment, although actual learning evidence would require learner-generated programs, behavioural logs, and explanation data.

6.2 Usability and the Importance of Context

The System Usability Scale (SUS)s score suggested favourable perceived usability. However, usability should be understood as dependent on the characteristics of the users, the tasks they perform, the goals they attempt to achieve, and the context in which interaction occurs. This interpretation conceptualises usability as related to effectiveness, efficiency, and satisfaction within a specified context of use. CodeBot is usable when commands produced direct and immediately observable consequences. Forward movement, turning, activation, and complete-program execution enabled learners to connect interface actions with visible changes in the robot's position and orientation. These features supported a relatively transparent relationship between the selected command and the resulting behaviour.

By contrast, procedures and repetition required learners to understand a less direct relationship between the visible code and the expanded sequence executed by the robot. Procedure P1 represented several commands through a single call, while the repetition operators affected only the immediately following command. These forms of compact representation introduced additional interpretive demands because learners had to mentally connect the abbreviated program structure with a longer sequence of actions. This distinction explains why CodeBot could receive a high overall usability score while computational features remained difficult. Global satisfaction measures provide a useful summary of user experience, but they should not replace feature-level evidence concerning errors, completion time, facilitator assistance, and misunderstanding. The findings therefore support a multidimensional approach in which perceived usability is interpreted alongside task performance, observations, and interview responses.

6.3 Operational Usability and Conceptual Mastery

A central conceptual contribution of the study is the distinction between operational usability and conceptual mastery. Operational usability refers to a learner's ability to locate, select, remove, and execute a programming block. Conceptual mastery refers to understanding when, why, and how the computational structure represented by that block should be applied. For example, a learner may successfully select Procedure P1 after observing a demonstration but may still be unable to identify a reusable pattern or explain why a procedure provides a more efficient solution. Similarly, a learner may understand that ×3 produces repetition without recognising that its scope applies only to the next command. In both cases, the learner can operate the interface without fully understanding the underlying computational concept. This distinction has important implications for later effectiveness studies. Task completion should not automatically be interpreted as evidence of computational understanding. Stronger evidence would require analysis of the learner's algorithm, explanation, revision strategy, and ability to apply the same concept in a different problem. Future evaluations should therefore distinguish between successfully using a feature and demonstrating the reasoning represented by that feature.

6.4 Progressive Complexity and Scaffolding

The task-performance data supported the intended progression from concrete movement to more abstract computational structures. Sequencing was the most accessible task type, followed by directional reasoning and debugging. Procedures and loops required more attempts, longer completion times, and greater facilitator support. This pattern is consistent with the past literature, which shows that many classroom interventions emphasise programming sequences and algorithmic thinking more frequently than decomposition, abstraction, and generalisation. More advanced computational processes may therefore require explicit modelling and structured scaffolding rather than simple exposure to additional programming blocks. Furthermore, the current progression should be strengthened through several forms of support. For example, a worked example could demonstrate how a repeated sequence is transferred into Procedure P1. A procedure preview could display the commands contained within P1 before execution. Repeated route patterns could be visually highlighted to help learners recognise when a loop or procedure is appropriate. An animation could clarify the

scope of ×2 and ×3, while side-by-side comparison could demonstrate the difference between a literal command sequence and a more abstract solution. Such scaffolding should be gradually reduced as learners develop greater independence. Introductory tasks may provide explicit visual support, whereas later mastery tasks could require learners to identify and apply the relevant structure without prompts.

6.5 Stepwise Execution as a Learning and Usability Feature

The findings identified the Step function as a particularly valuable component of CodeBot. Complete-program execution may be efficient when learners already understand their algorithms, but it can make it difficult for novices to remember where the first error occurred. Stepwise execution reduces this cognitive load by allowing learners to focus on one command and one state transition at a time. This feature appears to support both usability and computational reasoning. From a usability perspective, it enables learners to slow down program execution and inspect the interface more carefully. From a computational-thinking perspective, it supports tracing, prediction, error localisation, and targeted revision. In line with that, future versions should enhance stepwise execution by highlighting the command currently being performed, displaying an execution pointer, and showing the expanded commands produced when Procedure P1 is called. A brief pause before each state change could allow learners to predict the next movement, while a command-history panel could help them compare completed actions with the remaining sequence. These additions would make program execution more transparent and further strengthen the relationship between visible code and robot behaviour.

6.6 Contextualised Missions

Learners' positive responses to the life missions suggest that narrative and contextual purpose may support satisfaction and task comprehension. Delivery, rescue, and recycling scenarios provided recognisable objectives that gave meaning to the robot's movements and target activations. This interpretation is consistent with systematic-review findings indicating that serious games can strengthen engagement, confidence, and interactivity when their game elements are connected meaningfully to the learning activity. However, the life missions remain structurally similar to the earlier grid-navigation tasks. Although the narrative context changes, learners continue to use the same commands, route structure, and activation mechanics. These tasks should therefore be described as contextualised applications rather than evidence of far transfer. A stronger assessment of transfer would require learners to apply the same computational principles to structurally different problems, such as planning a delivery schedule, decomposing an emergency procedure, designing a recycling algorithm, or explaining a solution without the CodeBot interface. Therefore, contextual engagement should not be interpreted as generalised computational competence.

6.7 Implications for Educational-Game Research

This study offers four implications for the design and evaluation of educational games. First, researchers should map specific game mechanics to clearly define learning constructs rather than presenting features such as points, levels, animations, and rewards as inherently educational. Construct alignment should demonstrate how each interaction requires learners to perform the intended cognitive practice. Second, expert validation should extend beyond numerical content-validity indices. Quantitative agreement should be complemented by qualitative evaluation of missing content, unclear representations, accessibility barriers, and unintended consequences. This combination provides a more informative basis for prototype or product revision. Third, usability research involving children should integrate self-reported perceptions with behavioural evidence. Satisfaction scores alone cannot reveal whether learners misunderstood an operator, required repeated assistance, or completed a task through systematic reasoning. Completion rates, time, errors, prompts, observations, and interviews

should therefore be interpreted together. Fourth, interface usability and conceptual learning should be analysed as related but distinct outcomes. An educational game may be easy and enjoyable to operate without producing understanding of the represented concepts. Conversely, a concept may remain challenging even when the interface is well designed. Future studies should consequently assess both the quality of interaction and the quality of learning rather than treating usability as evidence of educational effectiveness.

7. CONCLUSION

This study presented the design and a preliminary evaluation framework for CodeBot, a progression-based computational-thinking game for upper-primary education. CodeBot operationalises sequencing, spatial reasoning, debugging, decomposition, procedures, repetition, and optimisation through visible robot movement and editable programs. Its six-level structure moves from direct command sequences toward reusable and compressed algorithmic representations. The first research question concerned content and design validity. The expert results suggested strong overall alignment but identified weaknesses in scoring, repetition scope, procedure transparency, orientation, and accessibility. The second research question concerned usability. The findings also suggested strong satisfaction and high completion of basic tasks. However, procedures and loops required more attempts, time, and facilitator support. Furthermore, the findings reinforce a critical distinction: an educational programming environment may be easy to operate without guaranteeing mastery of the computational concepts it represents. Future development and research should therefore assess interface interaction, program quality, learner explanations, debugging processes, and learning outcomes separately.

Acknowledgments: The author gratefully acknowledges the support of schools and institutions in facilitating the development of the CodeBot. Appreciation is also extended to colleagues who provided constructive feedback on the game design, computational-thinking content, and usability-evaluation framework. The author further acknowledges the contributions of the educational-technology and programming-education research community, whose scholarly work informed the conceptual and methodological development of this study.

References

Brennan, K., & Resnick, M. (2012, April). *New frameworks for studying and assessing the development of computational thinking*. Paper presented at the American Educational Research Association Annual Meeting, Vancouver, Canada.

Giannakoulas, A., & Xinogalos, S. (2024). Studying the effects of educational games on cultivating computational thinking skills to primary school students: A systematic literature review. *Journal of Computers in Education, 11*, 1283-1325. https://doi.org/10.1007/s40692-023-00300-z

Gundersen, S. W., & Lampropoulos, G. (2025). Using serious games and digital games to improve students' computational thinking and programming skills in K–12 education: A systematic literature review. *Technologies, 13*(3), 113. https://doi.org/10.3390/technologies13030113.

Hoadley, C., & Campos, F. C. (2022). Design-based research: What it is and why it matters to studying online learning. *Educational Psychologist, 57*(3), 207–220. https://doi.org/10.1080/00461520.2022.2079128.

Ma, J., Zhang, Y., Zhu, Z., Zhao, S., & Wang, Q. (2023). Game-based learning for students' computational thinking: A meta-analysis. *Journal of Educational Computing Research, 61*(7), 1430–1463. https://doi.org/10.1177/07356331231178948.

Mokkink, L. B., Herbelet, S., Tuinman, P. R., & Terwee, C. B. (2025). Content validity: Judging the relevance, comprehensiveness, and comprehensibility of an outcome measurement instrument—A COSMIN perspective. *Journal of Clinical Epidemiology, 185*, 111879. https://doi.org/10.1016/j.jclinepi.2025.111879.

Palop, B., Díaz, I., Rodríguez-Muñiz, L. J., & Santaengracia, J. J. (2025). Redefining computational thinking: A holistic framework and its implications for K–12 education. *Education and Information Technologies, 30*, 13385–13410. https://doi.org/10.1007/s10639-024-13297-4.

Putnam, C., Puthenmadom, M., Cuerdo, M. A., Wang, W., & Paul, N. (2020). Adaptation of the System Usability Scale for user testing with children. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery. https://doi.org/10.1145/3334480.3382840.

Santaengracia, J. J., Palop, B., & Rodríguez-Muñiz, L. J. (2026). Computational thinking in K–6 education: A scoping review of instructional practices. *Journal of Computers in Education, 13*, 983–1016. https://doi.org/10.1007/s40692-025-00380-z.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003.

Tikva, C., & Tambouris, E. (2021). Mapping computational thinking through programming in K–12 education: A conceptual model based on a systematic literature review. *Computers & Education, 162*, 104083. https://doi.org/10.1016/j.compedu.2020.104083.

Vlachogianni, P., & Tselios, N. (2022). Perceived usability evaluation of educational technology using the System Usability Scale: A systematic review. *Journal of Research on Technology in Education, 54*(3), 392–409. https://doi.org/10.1080/15391523.2020.1867938.

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

Yusoff, M. S. B. (2019). ABC of content validation and content validity index calculation. *Education in Medicine Journal, 11*(2), 49–54. https://doi.org/10.21315/eimj2019.11.2.6.

Zhang, Y., Song, Y., & Cutumisu, M. (2026). Assessing computational thinking skills in K–12 education: A systematic review. *Technology, Knowledge and Learning, 31*, 1953–2002. https://doi.org/10.1007/s10758-025-09938-3.