

AlgoQuest: A Web-Based and Mobile-Based Educational Game for Teaching Algorithmic Thinking

Ms. Divya Patil¹, Ms. Chetana Kawale²

Research Scholar, Department of Computer Applications, SSBT's COET, Jalgaon, Maharashtra, India¹

Professor, Department of Computer Applications, SSBT's COET, Jalgaon, Maharashtra, India²

Abstract : Game-based learning can close the gap between what students hear in lectures and what they can actually do when they write algorithms. In practice, though, educational software runs into three recurring problems: keeping progress in sync across devices, getting acceptable performance on phones as well as desktops, and keeping learners with very different backgrounds motivated. This paper describes the design, implementation and evaluation of AlgoQuest, an educational game that runs in web browsers (HTML5/WebGL) and on Android and iOS from one shared codebase. A lightweight serverless backend stores progress. A Dynamic Difficulty Adjustment (DDA) engine reads each learner's completion time, then adjusts puzzle difficulty and the amount of scaffolding. We tested the game in a four-week controlled study with 80 MCA students, using a pre-test/post-test design. The experimental group improved more than the control group (Hake's normalized gain $g = 0.581$ vs. 0.297 ; $p < 0.0001$; Cohen's $d = 1.81$). The experimental group also reported much lower frustration on the NASA-TLX and rated usability at 81.42 on the System Usability Scale. On a mid-range Android phone the game held about 58 FPS, with cloud synchronization under 180 ms. These results suggest that a single architecture can support both browser and mobile delivery of an algorithm-learning game. Because the study covers one department and one term, the results should be read as promising rather than final.

Index Terms—digital game-based learning, cross-platform architecture, WebGL, dynamic difficulty adjustment, flow theory, System Usability Scale, instructional technology.

I. INTRODUCTION

Data structures and algorithms sit at the centre of a computer applications degree, and MCA students are expected to master them before moving on to serious software development. Many of them struggle anyway. Recursive backtracking, pointer manipulation and graph traversal are abstract, and the usual workflow makes them harder: a student writes code, submits it to an automated judge or waits for a grader, and gets feedback long after the thinking has moved on. For beginners, a cryptic syntax error often does more damage than the algorithm itself, because frustration sets in before the logic has been understood.

Digital game-based learning (DGBL) is one response to this. When an algorithmic idea is built into the rules of a game, learners can try things out and see the result straight away. Recent studies of web-based educational games report gains in learning outcomes, shorter task times and high learner interest [18], and expert-validated game media have been rated feasible and effective in classroom trials [20]. Most of this work, however, targets school pupils in subjects such as mathematics, language and chemistry. Very little addresses postgraduate computing students, whose difficulty is conceptual abstraction rather than basic engagement.

Two practical obstacles also keep showing up when educational games are deployed at scale.

1. **Platform fragmentation.** Students use lab desktops, mid-range Android phones and iPads, often in the same week. Maintaining separate native codebases leads to version mismatches, uneven features and learning data scattered across devices. Mobile delivery has clear benefits, since learners can study anywhere and at any time [19], but it only works if the game behaves the same on every device.

2. **Fixed difficulty.** Many educational games use a static difficulty curve. In a class where prior skill varies widely, this frustrates weaker students and bores stronger ones, which runs against Csikszentmihalyi's flow theory [2]. Cahyana et al. found something related in chemistry: a mobile game helped learners with high learning independence but did worse than traditional teaching for learners with low independence [19]. A game that treats every learner the same can therefore widen gaps instead of narrowing them.

Moreno-Ger et al. argue that educational games need built-in *adaptation* and *assessment* if they are to be more than entertainment, and that they should be designed as systems whose state can be monitored and steered [8]. We follow that view here. This paper presents AlgoQuest, a cross-platform game for algorithmic thinking. Its main contributions are:

- A decoupled architecture that runs the same HTML5/WebGL code in browsers and in native Android/iOS packages, with cold-start times under 2 seconds and about 58–60 FPS on the hardware we tested.
- A mathematical specification of a Dynamic Difficulty Adjustment engine for computational-thinking puzzles.
- A four-week controlled evaluation with 80 MCA students, covering learning gain (Hake's g), perceived workload (NASA-TLX) and usability (SUS).

II. REVIEW OF LITERATURE

A. Constructivism and microworlds

AlgoQuest is based on constructivist learning theory and Papert's constructionism [13], which holds that people learn best when they build things inside a small, rule-governed world, or *microworld*. In a lecture, an algorithm is shown as a fixed sequence of steps on a slide, and students cannot change anything mid-run. In a microworld they can. If a student moves a pointer, swaps two array elements or adds a condition, the game shows the effect immediately, so learners can test their own guesses and revise their mental models as they go.

B. Flow and the zone of proximal development

Flow theory [2] describes engagement as a balance between challenge and skill. Too much challenge produces anxiety and too little produces boredom. Vygotsky's zone of proximal development [14] makes a similar point: learners progress most when the task sits just beyond what they can do alone, with support. Keeping a learner in that band means adjusting difficulty during play rather than relying on levels fixed in advance.

C. Design frameworks for educational games

We use the Mechanics–Dynamics–Aesthetics (MDA) framework [3] to separate the game's formal rules (mechanics), the behaviour that emerges when students play (dynamics), and the feelings the play produces (aesthetics), such as a sense of competence and curiosity.

Moreno-Ger et al. describe a game as a state-transition system, so that assessment rules (which states to log or grade) and adaptation rules (which state to move a learner into) can be attached without changing the game itself [8]. AlgoQuest's DDA engine and telemetry follow the same idea: the game state is observed and the next challenge is adjusted based on what is observed.

D. Evidence from comparable studies

Table I summarises three recent studies that are close to ours in method. All three used web or mobile games in a classroom, and all reported gains.

TABLE I
RELATED CLASSROOM STUDIES OF WEB/MOBILE EDUCATIONAL GAMES

Study	Context and sample	Design	Main findings
Qonitattsani & Sukardi [20]	Indonesian language, Grade 5, 24 pupils	ADDIE development; test; expert validation	Media experts 90%, pre/post-material experts 91%; N-Gain 0.566 and 0.573 (medium)
Asri et al. [18]	Mathematics, Grade 4, 32 pupils	Design-and-develop ment; single group, pre/post	Post-test rose from 62.34 to 78.91 (d = 1.64); task time fell from 28.6 to 21.4 min; interest M = 4.21/5
Cahyana et al. [19]	Chemistry, high school, 40 students	Quasi-experiment, 2×2 ANOVA	Mobile game beat traditional teaching only for high-independence learners

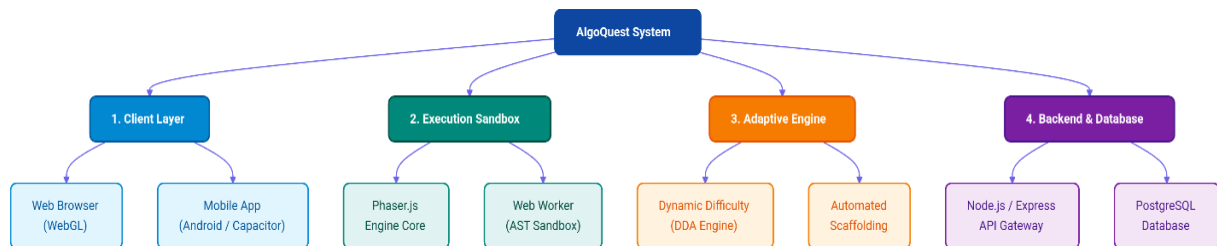
Our work differs in three ways. The learners are postgraduate computing students rather than school pupils. We use a randomized control group instead of a single-group pre/post design. And we build adaptation into the game itself, so that low-independence learners are supported by the system rather than left behind, which addresses the weakness reported in

[19].

Among computing-education tools, Scratch and Alice use block-based interfaces but are not designed for touch-first delivery or for advanced algorithmic content. Grasshopper offers mobile micro-lessons, but relies mostly on multiple-choice questions and guided tasks. Table II gives our own comparison of these tools with AlgoQuest; it reflects the authors' assessment of publicly documented features, not an independent benchmark.

TABLE II

FEATURE COMPARISON (AUTHORS' ASSESSMENT)



III. SYSTEM ARCHITECTURE

AlgoQuest is built in three tiers: a presentation layer on the client, a sandboxed execution layer running in a Web Worker, and a cloud layer for saving progress and telemetry.

A. Client and cross-platform delivery

Heavy 3D engines often produce WebGL builds larger than 40 MB, which is a problem on weak connections. We therefore chose a 2D WebGL/Canvas pipeline based on Phaser.js, which keeps the uncompressed asset bundle under 6.5 MB. For Android and iOS, the same HTML5 source is wrapped with the Capacitor runtime. Capacitor gives the JavaScript code access to haptics, low-latency audio and SQLite storage without changes to the game engine, so there is no separate mobile codebase.

B. Viewport and touch handling

Phones vary from 16:9 to 20:9 and 21:9 screens. To avoid clipped interfaces and mis-sized touch targets, the game draws into a fixed virtual space of 1920×1080 units and maps screen touches into that space with a transform:

$$\begin{bmatrix} X_{\text{game}} \\ Y_{\text{game}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1/S_x & 0 & -O_x \\ 0 & 1/S_y & -O_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_{\text{screen}} \\ Y_{\text{screen}} \\ 1 \end{bmatrix} \quad (1)$$

where S_x and S_y are scaling factors and O_x and O_y are letterbox offsets. An input layer converts mouse drags and touch gestures into one common pointer format, so play feels the same with a mouse or a finger.

C. Sandboxed execution

Running code written by novices is risky. One faulty loop can freeze the interface or crash a phone browser. AlgoQuest therefore runs all student logic in a separate Web Worker. The learner's visual block program is parsed into a small abstract syntax tree (AST), and the worker interprets it step by step under an instruction budget:

$$I_{\text{budget}} \leq 10,000 \text{ execution steps} \quad (2)$$

If a program exceeds the budget without finishing, the worker stops it, produces a structured "budget exceeded" message and sends it to the interface. The main rendering loop is unaffected.

D. Backend and offline behaviour

The backend is a serverless Node.js and PostgreSQL service behind an API gateway. Clients send progress, level times, error logs and hint use over HTTPS REST endpoints, authenticated with JSON Web Tokens. When the network drops, the client stores transactions locally in IndexedDB or SQLite and synchronizes when the connection returns. Every session is logged per user, in the same spirit as the per-student tracking and assessment reporting recommended in [8] and the individual login used in [18]. This logging is also what let us collect the timing and error data reported below.

IV. GAME DESIGN AND MECHANICS

A. Narrative and core loop

The player is a Systems Architect repairing a damaged distributed cloud network. Each stage is a failed routing node that needs a specific algorithm to restore data flow. A stage has four steps:

1. **Diagnostic briefing.** The game states the problem, the data constraints, the expected output and the complexity limit (for example, "sort this packet stream in $O(n \log n)$ comparisons using at most two auxiliary registers").
2. **Visual algorithm synthesis.** The learner arranges logic blocks, registers, conditional splitters and pointer operators on a circuit canvas.
3. **Stepped simulation.** The learner runs the design and watches data packets move between registers at an adjustable speed of 1–10 Hz, with variable values, comparisons and memory overwrites highlighted.
4. **Automated testing.** The routine is checked against hidden test cases, including edge cases such as null pointers, duplicate keys and reversed input. The level is complete only if every test passes within the complexity limit.

B. Dynamic Difficulty Adjustment

Difficulty D ranges from 1.0 to 5.0 and is updated after each stage k . For that stage we record:

- T_{actual} , the time from presentation to a valid solution (seconds);
- T_{expected} , the calibrated baseline time for the stage;
- E , the number of syntactic, structural or logical failures;
- H , the number of hints requested (0 to 3). The normalized performance metric is

$$P_m = w_1 \cdot (T_{\text{expected}} / \max(T_{\text{actual}}, 0.5 \cdot T_{\text{expected}})) - w_2 \cdot E - w_3 \cdot H \quad (3)$$

with weights $w_1 = 0.50$, $w_2 = 0.30$ and $w_3 = 0.20$. These weights were set empirically during pilot testing and were not otherwise tuned. Difficulty for the next stage is then

$$D_{k+1} = \text{clamp}(D_k + \alpha \cdot \tanh(P_m - \theta), D_{\min}, D_{\max}) \quad (4)$$

where $\alpha = 0.40$ is the adaptation rate, $\theta = 0.0$ is the equilibrium point, and $D_{\min} = 1.0$, $D_{\max} = 5.0$. The tanh function keeps each adjustment bounded, so a single very good or very bad attempt cannot swing difficulty too far.

C. Scaffolding tiers

The same metric selects one of three support levels for the next challenge:

$$S = \begin{cases} 2 \text{ (structural skeleton)} & \text{if } P_m < -0.60 \\ 1 \text{ (conceptual hints)} & \text{if } -0.60 \leq P_m < 0.00 \\ 0 \text{ (unassisted)} & \text{if } P_m \geq 0.00 \end{cases} \quad (5)$$

At $S = 2$ the canvas is pre-filled with key structure, such as loop initialization, so the learner is never completely stuck but still has to work out the core logic. This is the mechanism meant to help learners who need more guidance, the group that Cahyana et al. found was poorly served by a mobile game without adaptation [19]. Algorithm 1 gives the full procedure.

Algorithm 1. Real-time difficulty adjustment and scaffolding

```

Input:  $D_k, T_{\text{actual}}, T_{\text{expected}}, E, H$ 
Output:  $D_{\text{next}}, S$ 
1:  $w_1 \leftarrow 0.50, w_2 \leftarrow 0.30, w_3 \leftarrow 0.20, \alpha \leftarrow 0.40$ 
2:  $D_{\min} \leftarrow 1.0, D_{\max} \leftarrow 5.0$ 
3:  $T_{\text{clamped}} \leftarrow \max(T_{\text{actual}}, 0.5 \cdot T_{\text{expected}})$ 
4:  $T_{\text{ratio}} \leftarrow T_{\text{expected}} / T_{\text{clamped}}$ 
5:  $P_m \leftarrow w_1 \cdot T_{\text{ratio}} - w_2 \cdot E - w_3 \cdot H$ 
6:  $\delta \leftarrow \alpha \cdot \tanh(P_m)$ 
7:  $D_{\text{raw}} \leftarrow D_k + \delta$ 
8:  $D_{\text{next}} \leftarrow \max(D_{\min}, \min(D_{\max}, D_{\text{raw}}))$ 
9: if  $P_m < -0.60$  then  $S \leftarrow 2$ 
10: else if  $P_m < 0.00$  then  $S \leftarrow 1$ 
11: else  $S \leftarrow 0$ 
12: return  $D_{\text{next}}, S$ 

```

V. EXPERIMENTAL METHOD

A. Research questions

- **RQ1.** Does learning with AlgoQuest lead to a significantly larger gain in algorithmic knowledge than conventional lectures plus a browser-based text IDE?
- **RQ2.** How does real-time DDA affect learners' perceived workload, particularly mental demand, time pressure and frustration?
- **RQ3.** Does the cross-platform architecture meet reasonable performance targets on desktop and mobile devices?

B. Participants and design

The four-week study took place in the MCA department at SSBT's College of Engineering and Technology, Jalgaon. Eighty enrolled students took part, all with basic experience in procedural C/C++. They were randomly assigned to two groups of 40:

- **Control group (n = 40):** the standard lectures plus the same weekly assignments, done in a browser-based text IDE with textbook exercises and console output.
- **Experimental group (n = 40):** the same lectures, with all practical work done in AlgoQuest. Students could move freely between their own phones and lab desktops.

C. Content and timeline

Following a design-and-development approach similar to [18] and [20], the four weeks were planned around four modules:

1. Week 1: arrays and in-place sorting (bubble sort, insertion sort, two-pointer scanning).
2. Week 2: divide and conquer (merge sort partitioning, quicksort pivot selection).
3. Week 3: stacks, queues and pointers (call-stack tracing, circular queues, linked-list reversal).
4. Week 4: search and recursion (binary search bounds, recursive backtracking, memoization).

D. Instruments

- **Knowledge test.** A 30-item test of multiple-choice and short-answer questions covering tracing, error detection, complexity analysis and pseudocode writing (Cronbach's $\alpha = 0.84$), given before and after the intervention.
- **Hake's normalized gain.** $g = (\%S_{\text{post}} - \%S_{\text{pre}}) / (100\% - \%S_{\text{pre}})$, which allows comparison between groups that start at different levels (6). Gains are read as high ($g \geq 0.70$), medium ($0.30 \leq g < 0.70$) or low ($g < 0.30$).
- **NASA-TLX.** Six-subscale workload questionnaire [10].
- **System Usability Scale (SUS).** Ten-item usability survey, completed by the experimental group only [5].
- **Telemetry.** Frame rate, load time, memory and synchronization latency recorded across 1,200 sessions.

VI. RESULTS

A. Baseline equivalence

Before the intervention the control group averaged 44.82% (SD = 8.21) and the experimental group 45.31% (SD = 7.94). The difference was not significant, $t(78) = 0.274$, $p = 0.785$, so the groups started from the same place.

B. Learning gains (RQ1)

TABLE III

TEST SCORES AND NORMALIZED GAIN				
Group	Pre-test (SD) Absolute (SD)	Post-test gain	Hake's g	Category
Control (n = 40)	44.82% (8.21)	61.25% (9.12) +16.43%	0.297	Low
Experimental (n = 40)	45.31% (7.94)	77.10% (8.42) +31.79%	0.581	Medium

The control group gained 16.43 percentage points ($g = 0.297$, just under the medium threshold). The experimental group gained 31.79 points ($g = 0.581$, a medium gain). An independent-samples t-test on post-test scores showed a clear advantage for AlgoQuest: $t(78) = 8.085$, $p < 0.0001$, Cohen's $d = 1.81$, a very large effect.

For context, the normalized gains in the two Indonesian classroom studies that reported them were also in the medium range (0.566 and 0.573 in [20]), and the standardized effect size in [18] ($d = 1.64$) is similar to ours. Our result is therefore in line with earlier work, with the added benefit of a control group.

C. Workload and usability (RQ2)

TABLE IV

NASA-TLX SUBSCALES (0–100; LOWER IS BETTER EXCEPT PERFORMANCE)

Dimension	Control M Experimental M (SD)	(SD)	Changep
Mental demand	74.2 (11.4)	71.8 (10.6)	–3.2% 0.334
Temporal demand	68.5 (14.2)	54.1 (12.8)	–21.0% < 0.001
Performance	58.2 (13.5)	78.4 (11.2)	+34.7% < 0.0001
Effort	76.1 (12.3)	68.3 (11.9)	–10.2% 0.005
Frustration	50.8 (15.6)	31.4 (12.2)	–38.2% < 0.0001

Mental demand was about the same in both groups, so the game did not make the material easier. Frustration, however, was 38.2% lower in the experimental group. This fits the intended role of the DDA engine, which supplies hints or a skeleton after repeated failures. We did not run a version of the game without DDA, so we cannot say how much of the drop is due to DDA specifically and how much to the game format in general.

The experimental group rated usability at a mean SUS score of 81.42 (SD = 6.24). Under Bangor et al.'s benchmarks [15], a score above 80.3 corresponds to an "A" grade and an "excellent" rating.

D. Technical performance (RQ3)

TABLE V

PERFORMANCE ACROSS DEPLOYMENT TARGETS (1,200 SESSIONS)

Metric	Desktop web (Chrome)	Mobile app (Android)	Target
Cold-start load	1.84 s (±0.22)	1.42 s (±0.15)	< 3.0 s
Frame rate	59.8 FPS (±0.4)	58.4 FPS (±1.8)	≥ 55 FPS
Peak heap memory	148.4 MB	112.6 MB	< 250 MB
Cloud sync latency	124 ms	178 ms	< 300 ms
Offline cache size	8.2 MB	12.4 MB	< 50 MB

Desktop tests used an Intel Core i5 with 8 GB RAM and integrated graphics. Mobile tests used an Android 13 phone with a MediaTek Dimensity 7050 and 8 GB RAM. Every target was met on both platforms.

VII. DISCUSSION

Why the learning gains were larger. In a text IDE, novices have to run the code in their heads, and a compiler message says little about what memory looks like at that moment. In AlgoQuest the execution is visible. Students watched values move between array slots, and an off-by-one pointer error showed up on screen straight away. Immediate feedback of this kind is also what Asri et al. credit for the gains and time savings in their mathematics game [18]. We did not measure task completion time as they did, which is something we would add in future work.

What DDA seems to do. The workload data suggest that the game kept the intellectual demand high while lowering frustration. This matters because Cahyana et al. saw the opposite pattern with a non-adaptive mobile game: strong self-directed learners benefited and weaker ones fell behind [19]. Our results are consistent with adaptive support helping to avoid that, but we did not measure learning independence, so this remains an inference and not a tested claim.

Mobility. Telemetry showed that 46.2% of all sessions took place on phones, mostly between 19:00 and 23:00, outside lab hours. Students could start a challenge on a lab computer and continue on a phone on the way home, which added practice time without requiring extra lab access. This matches the point made in [19] that anywhere-anytime access is a main strength of mobile game media.

Integration and assessment. Moreno-Ger et al. recommend connecting games to a learning management system so that instructors can define which events to track and grade [8]. AlgoQuest records hint use, errors and timing for every level, but it does not yet connect to an LMS through standards such as SCORM or IMS. This is an obvious next step, and it would let instructors set assessment rules without editing the game.

Limitations. Several factors limit how far these results can be generalized.

1. *Novelty effect.* A new game can raise motivation for a while regardless of its design. Both [18] and our own results should be read with this in mind, and longer studies are needed.
2. *Single population.* All participants were MCA students at one institution. Results may differ for undergraduates or school pupils.
3. *No expert validation stage.* Unlike [20], we did not run a formal expert-validation round with media and subject experts before the trial. We relied on pilot testing and the SUS and NASA-TLX results.
4. *No DDA ablation and no learning-independence measure.* We compared AlgoQuest with a text IDE, not with a static-difficulty version of AlgoQuest, and we did not classify students by learning independence.
5. *Hardware.* Testing covered mid-range phones. Devices with less than 2 GB of RAM might show frame drops during complex recursive visualizations.

VIII. CONCLUSION AND FUTURE WORK

We presented AlgoQuest, a game for learning algorithmic thinking that runs in browsers and on Android and iOS from one codebase, and evaluated it in a four-week randomized study with 80 MCA students. The experimental group achieved a medium normalized gain ($g = 0.581$) against a low gain for the control group ($g = 0.297$), reported 38.2% lower frustration, and rated usability at 81.42 on the SUS. The game ran at about 58 FPS on a mid-range phone with cloud synchronization under 180 ms.

These findings suggest that a shared architecture with adaptive difficulty is a workable way to deliver algorithm practice to a mixed group of learners. Future work will:

1. run a longer study, with a static-difficulty comparison group and a measure of learning independence, to test whether DDA especially helps less self-directed students;
2. add an expert-validation stage and a learning-interest questionnaire, as in [18] and [20];
3. connect the game to an LMS so that assessment and adaptation rules can be set by instructors [8];
4. add collaborative debugging over WebRTC; and
5. replace the heuristic DDA equations with a Bayesian Knowledge Tracing model or a reinforcement-learning agent.

REFERENCES

- [1] J. McGonigal, *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. New York, NY, USA: Penguin Press, 2011.

- [2] M. Csíkszentmihályi, *Flow: The Psychology of Optimal Experience*. New York, NY, USA: Harper & Row, 1990.
- [3] R. Hunicke, M. LeBlanc, and R. Zubek, "MDA: A formal approach to game design and game research," in *Proc. AAAI Workshop on Challenges in Game AI*, San Jose, CA, USA, 2004, pp. 1–5.
- [4] R. R. Hake, "Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses," *Amer. J. Phys.*, vol. 66, no. 1, pp. 64–74, 1998.
- [5] J. Brooke, "SUS: A 'quick and dirty' usability scale," in *Usability Evaluation in Industry*, P. W. Jordan et al., Eds. London, U.K.: Taylor & Francis, 1996, pp. 189–194.
- [6] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining gamification," in *Proc. 15th Int. Academic MindTrek Conf.*, Tampere, Finland, 2011, pp. 9–15.
- [7] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cogn. Sci.*, vol. 12, no. 2, pp. 257–285, 1988.
- [8] P. Moreno-Ger, D. Burgos, I. Martínez-Ortiz, J. L. Sierra, and B. Fernández-Manjón, "Educational game design for online education," *Comput. Human Behav.*, vol. 24, no. 6, pp. 2530–2540, 2008.
- [9] K. Kiili, "Digital game-based learning: Towards an experiential gaming model," *Internet Higher Educ.*, vol. 8, no. 1, pp. 13–24, 2005.
- [10] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research," *Adv. Psychol.*, vol. 52, pp. 139–183, 1988.
- [11] T. W. Malone, "Toward a theory of intrinsically motivating instruction," *Cogn. Sci.*, vol. 5, no. 4, pp. 333–369, 1981.
- [12] V. Shute, "Focus on formative feedback," *Rev. Educ. Res.*, vol. 78, no. 1, pp. 153–189, 2008.
- [13] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY, USA: Basic Books, 1980.
- [14] L. S. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA, USA: Harvard Univ. Press, 1978.
- [15] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the System Usability Scale," *Int. J. Human-Comput. Interact.*, vol. 24, no. 6, pp. 574–594, 2008.
- [16] C. A. Steinkuehler and M. W. Duncan, "Scientific habits of mind in virtual worlds," *J. Sci. Educ. Technol.*, vol. 17, no. 6, pp. 530–543, 2008.
- [17] T. Connolly, E. A. Boyle, E. MacArthur, T. Hainey, and J. M. Boyle, "A systematic literature review of empirical evidence on computer games and serious games," *Comput. Educ.*, vol. 59, no. 2, pp. 661–686, 2012.