frontiers | Frontiers in Education

TYPE Original Research
PUBLISHED 16 June 2025
DOI 10.3389/feduc.2025.1585602

OPEN ACCESS

EDITED BY
Marko Hölbl,
University of Maribor, Slovenia

REVIEWED BY
Eila Burns,
JAMK University of Applied Sciences, Finland
Na Li,
Central China Normal University, China

*CORRESPONDENCE
Cristian Vidal-Silva
✉ cvidal@utalca.cl

†These authors have contributed equally to this work

RECEIVED 28 February 2025
ACCEPTED 14 May 2025
PUBLISHED 16 June 2025

CITATION
Castillo-Salvatierra L, Cárdenas-Cobo J, de la Fuente-Burdiles C and Vidal-Silva C (2025) Programming competencies in university students through game development. *Front. Educ.* 10:1585602. doi: 10.3389/feduc.2025.1585602

Programming competencies in university students through game development

Luis Castillo-Salvatierra[1,2†], Jesennia Cárdenas-Cobo[2†], Claudia de la Fuente-Burdiles[3] and Cristian Vidal-Silva[3*†]

[1]Departamento Tecnologías de la Información y la Comunicación, Universidad Estatal de Milagro, Milagro, Guayas, Ecuador, [2]Facultad Ciencias e Ingeniería, Universidad Estatal de Milagro, Milagro, Guayas, Ecuador, [3]Departamento de Visualización Interactiva y Realidad Virtual, Universidad de Talca, Talca, Chile

Introduction: The growing importance of programming in higher education requires innovative approaches to facilitate learning. Video games have emerged as an engaging tool that enhances problem-solving skills and logical thinking.

Methodology: This study examines the effectiveness of Unity Visual Scripting in fostering programming competencies among university students. A project-based methodology was employed in the course "Physics for Videogames" during the third semester of the Engineering in Video Game Development and Virtual Reality program at the Universidad de Talca, Chile.

Results: Through game development, students overcame challenges associated with text-based programming. The results indicate significant improvements in students' computational thinking, motivation, and collaboration.

Discussion: By reducing syntactical barriers, Visual Scripting promotes an accessible learning experience that supports the transition to advanced programming concepts. These findings suggest that integrating game development into university curricula enhances digital literacy and fosters an inclusive programming education environment.

KEYWORDS

video games, higher education, programming, unity, visual scripting

1 Introduction

The development of programming skills is a fundamental competence in higher education, fostering logical thinking, problem-solving, and creativity among students (Wing, 2006). However, programming education has faced challenges due to the complexity of text-based languages, leading to high dropout rates and decreased motivation among beginners (Guzdial, 2019).

Programming has become a crucial skill across various disciplines beyond computer science (Grover and Pea, 2013; Denning, 2017). Its applicability in fields such as biomedicine, economics, engineering, and social sciences highlights its transdisciplinary nature. In higher education, teaching programming not only enhances computational thinking but also strengthens analytical and creative abilities essential for problem-solving in diverse knowledge domains (Resnick et al., 2009). The incorporation of innovative approaches, such as block-based programming through Visual Scripting in Unity, represents an effective strategy to make these concepts more accessible and encourage students to explore programming's potential (Strodick and Schattkowsky, 2024).

The use of video games as an educational tool has proven effective in facilitating programming learning, particularly in environments that reduce the cognitive load of textual syntax (Resnick et al., 2009). Unity Visual Scripting allows programming through visual blocks, eliminating syntactical barriers and enabling students to focus on programming logic and game development (Strodick and Schattkowsky, 2024).

To evaluate the effectiveness of Visual Scripting in higher education, this study poses the following research questions: (1) To what extent does Visual Scripting improve programming skills in university students? (2) How does Visual Scripting influence student motivation and collaboration in programming education? (3) Can Visual Scripting serve as a bridge for students to transition to more advanced programming languages?

Figure 1 illustrates an example of a simple program in Unity Visual Scripting. Figure 1a shows the "Hello World" program implemented using a node-based structure, while part Figure 1b displays the corresponding output. This example highlights the accessibility of visual scripting, which reduces syntax-related errors and enhances beginner-friendly programming experiences (Miranda et al., 2021).

Table 1 compares traditional text-based programming and Unity Visual Scripting. The table highlights key differences, such as the reduced syntax complexity and debugging requirements in visual scripting, making it a more accessible approach for novice programmers (Zhang et al., 2022).

2 Theoretical framework

The development of computational thinking and programming skills has gained increasing importance in higher education due to its relevance in various scientific and technical disciplines (Wing, 2006; Ogegbo and Ramnarain, 2021). Computational thinking fosters problem-solving abilities, algorithmic reasoning, and abstraction skills, essential in learning programming (Denning, 2017; Arslantaş, 2024).

However, teaching programming has traditionally faced significant challenges, primarily due to the complexity of text-based languages (Sun et al., 2024). High dropout rates among students in introductory programming courses have been attributed to syntax errors and the steep learning curve associated with traditional programming approaches (Guzdial, 2019; Raj and Sharma, 2023). In response, alternative methodologies such as block-based programming and visual scripting have been introduced to mitigate these challenges (Resnick et al., 2009; Vidal-Silva et al., 2024; Vinueza-Morales et al., 2025).

Visual scripting tools, such as Unity Visual Scripting, enable students to develop interactive applications without the need for extensive coding experience (Cárdenas-Cobo et al., 2024; Rojas-Valdés et al., 2022; Tupac-Yupanqui et al., 2022). Research suggests that visual programming environments reduce cognitive load and allow students to focus on understanding programming logic rather than syntax (Strodick and Schattkowsky, 2024). Additionally, these environments facilitate experimentation and foster creativity, making programming more accessible for beginners (Miranda et al., 2021).

Game-based learning has been identified as an effective strategy to enhance programming education (Zhao et al., 2022). Video game development provides a structured and engaging context where students apply computational concepts while improving problem-solving and teamwork skills (Martín-Hernández et al., 2021; Laakso et al., 2021). Furthermore, gamification techniques have increased student motivation and retention of programming concepts (Montes et al., 2021).

Studies have also highlighted the importance of integrating visual scripting tools into university curricula as a transitional approach toward text-based programming (Noone and Mooney, 2018). By lowering entry barriers, visual scripting encourages students to develop computational thinking and prepares them for more advanced programming paradigms (Weintrop and Wilensky, 2019).

Building upon these practical insights, this study anchors its approach in established educational theories that support the use of visual scripting in programming education. Papert's constructionism theory (Papert, 1980) emphasizes that meaningful learning occurs when learners actively construct tangible products, a process inherently supported by visual scripting environments. Furthermore, Vygotsky's (1978) theory of the Zone of Proximal Development (ZPD) highlights the role of scaffolding in enabling learners to engage with complex concepts progressively. Visual

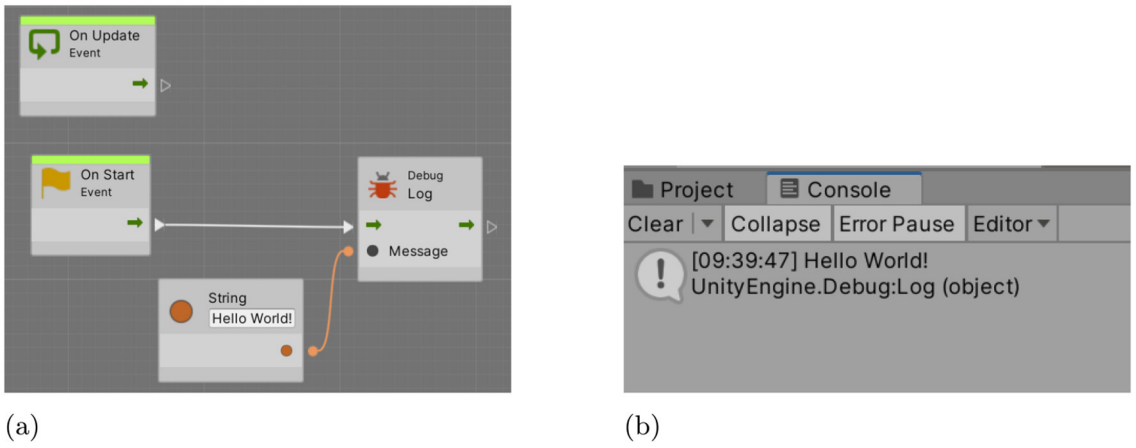

FIGURE 1
Example of "Hello World" using Unity Visual Scripting. **(a)** Code for "Hello World" in Unity Visual Scripting. **(b)** Execution and output of "Hello World" in Unity Visual Scripting.

scripting simplifies programming tasks, aligning with this scaffolding approach by allowing early access to computational concepts. Finally, as Wing (2006) posits, computational thinking is a fundamental competency in contemporary education. Visual scripting facilitates the development of computational thinking by enabling students to focus on logical problem-solving and algorithmic reasoning without being hindered by syntax complexities.

The data presented in Figure 2 and Table 2 highlight the increasing adoption of visual programming tools in educational settings. The figure illustrates a steady growth in the usage of block-based and visual scripting tools, reflecting a shift in pedagogical approaches toward more intuitive and accessible programming environments. This trend aligns with recent studies emphasizing the importance of reducing syntactic barriers to enhance student engagement and learning outcomes (Montes et al., 2021). The comparative analysis in Table 2 further reinforces this notion, demonstrating that visual scripting tools strike a balance between syntax complexity and engagement. While traditional text-based programming remains essential for advanced development, visual scripting provides a valuable entry point for beginners, offering high engagement and retention rates. These findings support the argument for integrating visual scripting tools like Unity Visual Scripting into university curricula as a bridge toward more complex programming paradigms (Weintrop and Wilensky, 2019).

TABLE 1 Comparison of text-based programming and visual scripting.

Feature	Text-based programming	Visual scripting
Syntax complexity	High	Low
Learning curve	Steep	Moderate
Debugging requirement	High	Low
Accessibility for beginners	Low	High
Application scope	Advanced	Introductory/intermediate

3 Materials and methods

Understanding the impact of visual scripting in programming education requires a well-structured methodology. This section presents the study's design, participants, resources, experimental process, and evaluation techniques.

3.1 Experimental process

The study followed a quasi-experimental design over a six-week period. Participants were randomly assigned to two groups: an experimental group (using Unity Visual Scripting) and a control group (using traditional C# programming).

The experimental process involved the following stages:

- Pretest: all students completed an initial test assessing baseline programming knowledge and problem-solving skills.
- Intervention: both groups completed identical programming assignments (object creation, force application, collision simulation). The experimental group implemented these tasks using Unity Visual Scripting, while the control group developed their solutions using C#.
- Posttest: after the intervention period, students completed a final test evaluating their programming competencies.

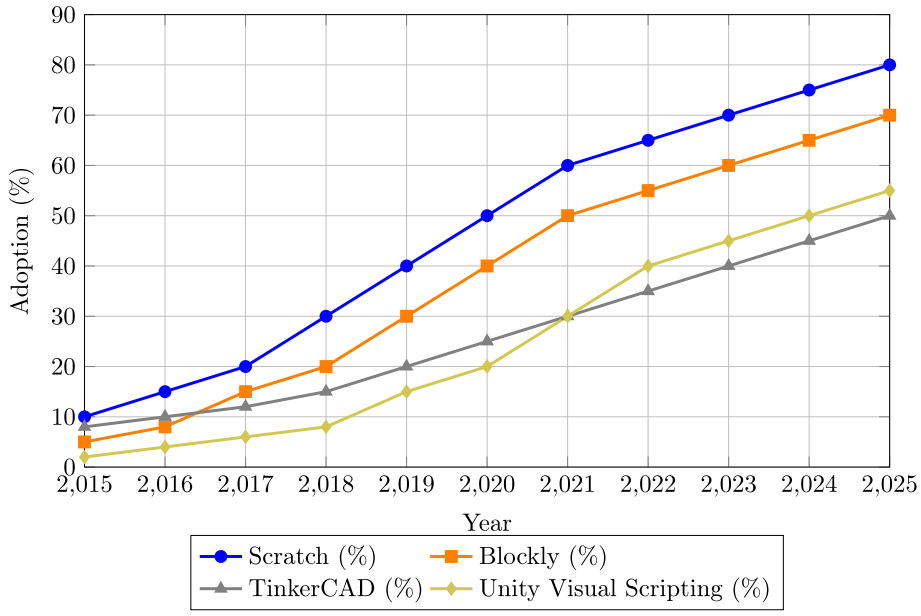

FIGURE 2
Trends in the adoption of visual programming tools from 2015 to 2025.

The independent variable was the type of programming environment (Visual Scripting vs. Text-Based). Dependent variables included task completion time, project performance, and survey-based measures of engagement and motivation.

The assignments were designed to be equivalent in complexity across both groups to ensure a fair comparison. Weekly progress tracking and post-intervention surveys were used to evaluate outcomes.

3.2 Study design

A quasi-experimental design was implemented in the *Physics for Video Games* course at Universidad de Talca, Chile (Ingeniería en Desarrollo de Videojuegos y Realidad Virtual, 2025). The study divided students into two groups: an experimental group using Unity Visual Scripting and a control group using traditional text-based programming with C#. Both groups received identical assignments, ensuring direct comparison of their learning progress.

3.3 Participants

The study involved 40 undergraduate students in the early semesters of their Engineering in Video Game Development and Virtual Reality program. The division of students into groups ensured that prior programming knowledge did not influence the results, maintaining a fair evaluation of visual scripting as an entry-level tool.

TABLE 2 Comparison of different programming learning approaches.

Learning approach	Syntax complexity	Engagement	Retention rate
Text-based programming	High	Moderate	Low
Block-based programming	Low	High	Moderate
Visual scripting (unity)	Moderate	Very high	High

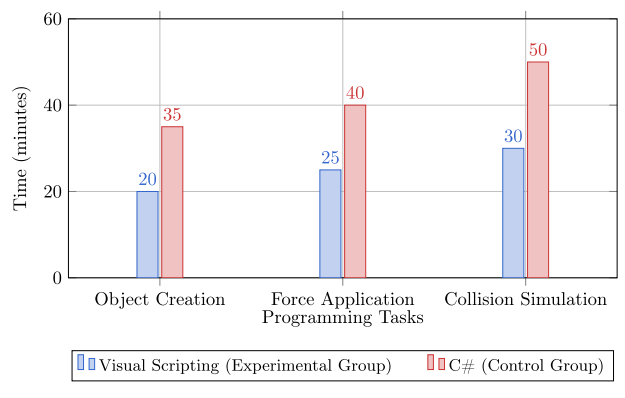

FIGURE 3
Average time to complete programming tasks by group.

3.4 Software and hardware resources

The experimental group worked with Unity Visual Scripting (Unity - Discussions, 2025), an intuitive, node-based development tool. The control group used C# with MonoGame, requiring direct text-based coding. Both groups utilized similar computer specifications: Intel i5 processors, 8GB RAM, and dedicated graphics cards, ensuring uniform performance conditions.

3.5 Evaluation metrics

Three key evaluation metrics were used:

- Knowledge improvement: measured via pre-tests and post-tests covering programming logic and problem-solving.
- Project performance: assessed based on the complexity, correctness, and functionality of student-developed game projects.
- Engagement and perception: evaluated using surveys assessing students' motivation, perceived difficulty, and confidence.

Figure 3 highlights the benefits of Unity Visual Scripting in beginner-level programming education. The experimental group showed greater improvement in task efficiency and higher engagement levels compared to the control group, reinforcing the effectiveness of visual scripting as an entry point into programming education (Umezawa et al., 2021; Marín-Lora and Chover, 2025).

4 Results

This section presents the results of the study, comparing the effectiveness of Unity Visual Scripting against traditional text-based programming methods. The findings include student performance analysis, engagement levels, and trends in learning progress.

4.1 Performance analysis

The pre-test and post-test assessments demonstrated significant learning improvements in both groups. Students in the experimental group (Unity Visual Scripting) achieved an average score improvement of 35%, while those in the control group (text-based C# programming) improved by 20%.

Notably, the relative improvement of the experimental group was ~75% higher compared to the control group. This suggests that visual scripting substantially enhanced students' ability to understand and apply programming concepts within a shorter learning curve. The reduction of syntax complexity likely contributed to faster mastery of core computational skills, aligning with the scaffolding principles of Vygotsky's Zone of Proximal Development.

Moreover, 85% of the experimental group students successfully completed all assigned programming projects, compared to 70% in the control group. Detailed task analysis revealed that the assignments involving force application and collision simulation exhibited the greatest differences in completion times, favoring

TABLE 3 Comparison of pre-test and post-test scores between experimental and control groups.

Group	Pre-test score (%)	Post-test score (%)
Visual scripting (experimental)	45%	80%
Text-based programming (control)	50%	70%

the experimental group. For instance, the experimental group completed collision simulation tasks in 30 min on average, while the control group required 50 min. Table 3 summarizes the pre-test and post-test scores obtained by both groups. These results demonstrate the relative advantage of Unity Visual Scripting in both conceptual understanding and execution efficiency.

These performance differences indicate that students using Unity Visual Scripting not only achieved better learning outcomes but also demonstrated higher efficiency in task execution.

4.2 Student engagement and perception

Survey data collected at the end of the intervention showed that students using Visual Scripting reported higher levels of engagement, motivation, and perceived ease of learning compared to their counterparts. Specifically, 90% of the experimental group expressed satisfaction with the learning approach, while only 65% of the control group did. As shown in Table 4, students using Unity Visual Scripting reported significantly higher levels of satisfaction, motivation, and confidence compared to those in the control group, reinforcing the perceived benefits of a visual learning environment.

Furthermore, students in the experimental group indicated a higher level of confidence in their programming abilities post-intervention. These perceptions are consistent with constructionist learning theories, where active creation and reduction of cognitive load enhance learner engagement and motivation.

4.3 Trend analysis of learning progress

Figure 4 illustrates the trend analysis of performance improvement over the six-week study period. The experimental group exhibited a more consistent and steeper growth trend in programming competency compared to the control group.

In the experimental group, performance improvement rose steadily each week, reaching 35% by the end of the intervention. In contrast, the control group's improvement plateaued around 20%. This indicates that visual scripting supported more rapid and sustained learning progression.

The steady weekly improvements in the experimental group underscore the potential of visual scripting as an effective transitional pedagogical tool, preparing students for subsequent engagement with text-based programming environments.

Overall, these findings reinforce the educational value of visual scripting tools in enhancing programming skills, fostering higher engagement, and facilitating smoother learning curves in early programming education.

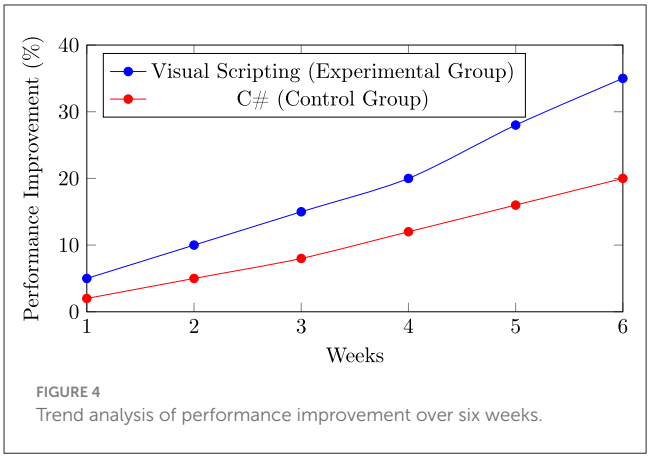

FIGURE 4
Trend analysis of performance improvement over six weeks.

TABLE 4 Comparison of student engagement and perception.

Aspect	Experimental group (% positive)	Control group (% positive)
Satisfaction	90%	65%
Motivation	88%	62%
Confidence in programming	85%	60%

Table 5 summarizes the key learning outcomes and student perceptions observed in both groups. The data consistently show that the Visual Scripting group outperformed the Text-Based Programming group across all measured dimensions, reinforcing the effectiveness of visual scripting in enhancing programming education.

5 Discussion

The findings of this study highlight the advantages of using Unity Visual Scripting in introductory programming education. This section critically discusses the implications of the results, situating them within the broader theoretical and empirical context, and identifies key takeaways and limitations for future curriculum development.

The improvements in test scores and project completion rates observed in Table 5 suggest that Unity Visual Scripting effectively lowers entry barriers for programming students. By reducing syntactic complexity, students could focus more on problem-solving and algorithmic reasoning, key components of computational thinking as defined by Wing (2006). This aligns with Papert's constructionism theory (Papert, 1980), which posits that active creation of artifacts enhances learning, and Vygotsky's (1978) Zone of Proximal Development, emphasizing the importance of scaffolding in acquiring complex skills.

The trend analysis depicted in Figure 4 further reinforces that visual scripting supports sustained, progressive skill acquisition over time. Students' steady performance improvements align with previous findings on the benefits of scaffolded, incremental learning environments (Marín-Lora and Chover, 2025; Umezawa et al., 2021).

TABLE 5 Summary of learning outcomes and student perceptions by group.

Measured aspect	Visual scripting group	Text-based programming group
Pre-test average score	45%	50%
Post-test average score	80%	70%
Improvement in test scores	35%	20%
Average task completion time (min)	25 min	41.7 min
Project completion rate	85%	70%
Satisfaction (survey)	90%	65%
Motivation (survey)	88%	62%
Confidence in programming (survey)	85%	60%

Survey results indicated higher motivation, satisfaction, and confidence among students in the experimental group. These findings are consistent with research on the motivational effects of reducing cognitive load and promoting active learning environments (Montes et al., 2021). Students reported that the visual scripting interface made programming more accessible and engaging, which corroborates prior studies on gamified and block-based learning approaches (Resnick et al., 2009; Zhao et al., 2022).

5.1 Comparison with previous studies

The effectiveness of block-based programming environments has been widely acknowledged. Tools such as Scratch and Blockly have demonstrated positive impacts on early programming education (Montes et al., 2021). The present findings extend these insights to higher education contexts, demonstrating that Unity Visual Scripting can play a similar role in fostering computational thinking and programming competencies.

Moreover, the significant reduction in task completion times observed in the experimental group reflects findings by Miranda et al. (2021), who noted that intuitive, low-syntax interfaces facilitate faster task execution and deeper conceptual understanding.

5.2 Limitations

While the findings are promising, several limitations must be acknowledged. First, the sample size was limited to a single university cohort, potentially constraining the generalizability of the results. Future research should replicate this study across diverse educational settings and larger sample sizes.

Second, the study focused primarily on short-term learning outcomes. Longitudinal research is needed to assess the persistence of skills acquired through visual scripting, particularly as students transition to more advanced, text-based programming languages.

Third, although performance and engagement metrics were evaluated, qualitative insights into students' learning experiences were not deeply explored. Future studies could incorporate interviews or open-ended surveys to capture richer perspectives.

5.3 Implications for curriculum design

The results emphasize the importance of adopting progressive instructional strategies in programming education. Visual scripting can serve as an effective bridge to more complex programming paradigms by initially reducing cognitive barriers and fostering confidence.

Institutions should consider:

- Hybrid learning approaches: integrating both visual and text-based programming progressively within the curriculum.
- Bridging strategies: designing transition modules that gradually introduce textual elements to students comfortable with visual representations.
- Gamified learning environments: leveraging game development frameworks to sustain student engagement and promote deeper computational thinking.

Overall, Unity Visual Scripting represents a promising pedagogical tool for enhancing digital literacy, computational competencies, and inclusive programming education environments.

6 Conclusions

This study provides compelling evidence that Unity Visual Scripting serves as an effective pedagogical tool for introducing programming concepts in higher education. By reducing syntactic barriers and promoting logical reasoning, visual scripting environments facilitate deeper engagement with problem-solving and computational thinking, aligning with key educational theories such as Papert's constructionism and Vygotsky's scaffolding principles.

The results demonstrated that students using Unity Visual Scripting achieved a significantly higher improvement in test scores (35%) and a greater project completion rate (85%) compared to students in the text-based programming group (20% improvement and 70% project completion rate, respectively). Furthermore, students in the experimental group reported higher levels of satisfaction, motivation, and confidence in their programming abilities.

These findings confirm that visual scripting tools effectively lower initial learning barriers, foster engagement, and promote the development of computational competencies. They reinforce prior research on the benefits of block-based and gamified programming environments (Montes et al., 2021; Resnick et al., 2009; Zhao et al., 2022) and demonstrate that visual scripting can act as a transitional stage toward advanced programming paradigms (Weintrop and Wilensky, 2019).

Despite these advantages, several challenges remain. The transition from visual scripting to text-based programming languages remains a critical hurdle, with some students initially struggling to adapt to the increased syntactic complexity. Future research should investigate structured transition models that progressively integrate textual coding within visual environments (Marín-Lora and Chover, 2025).

The study suggests several key implications for curriculum design:

- Hybrid learning approaches: integrating visual and text-based programming within the curriculum to support a progressive learning curve.
- Bridging strategies: developing transition modules that ease the shift from visual to textual programming paradigms.
- Game-based learning integration: leveraging interactive and gamified environments to sustain student motivation and engagement.

Future research should focus on evaluating the long-term retention of skills acquired through visual scripting, exploring its scalability across diverse educational contexts, and investigating its integration with other instructional methodologies to optimize programming education outcomes (Hu et al., 2021; Umezawa et al., 2021).

In conclusion, Unity Visual Scripting represents a promising and innovative approach to programming education. Its implementation in university courses can significantly enhance digital literacy, foster inclusive learning environments, and support the development of computational thinking skills essential for success in an increasingly digital world.

Data availability statement

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Author contributions

LC-S: Writing – original draft, Writing – review & editing. JC-C: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. CF-B: Writing – original draft, Writing – review & editing. CV-S: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Funding

The author(s) declare that no financial support was received for the research and/or publication of this article.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Gen AI was used in the creation of this manuscript.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Arslantaş, T. K. (2024). "Theoretical framework for integrating computational thinking in education," in *Integrating Computational Thinking Through Design-Based Learning*, eds. M. Saritepeci, and H. Yildiz Durak (Singapore: Springer), 15–31. doi: 10.1007/978-981-96-0853-9_2

Cárdenas-Cobo, J., Vidal-Silva, C., Arévalo, L., and Torres, M. (2024). Applying recommendation system for developing programming competencies in children from a non-weird context. *Educ. Inf. Technol.* 29, 9355–9386. doi: 10.1007/s10639-023-12156-y

Denning, P. J. (2017). Computational thinking in science. *Commun. ACM* 60, 33–35. doi: 10.1515/9780691188720-006

Grover, S., and Pea, R. (2013). Computational thinking in k-12: a review of the state of the field. *Educ. Res.* 42, 38–43. doi: 10.3102/0013189X12463051

Guzdial, M. (2019). *Learner-Centered Design of Computing Education*. San Rafael, CA: Morgan & Claypool.

Hu, M., Assadi, T., and Mahroeian, H. (2021). "Teaching visualization-first for novices to understand programming," in *2021 IEEE International Conference on Engineering, Technology & Education (TALE)* (Wuhan), 654–660. doi: 10.1109/TALE52509.2021.9678922

Ingeniería en Desarrollo de Videojuegos y Realidad Virtual (2025). *IDVRV - Ingeniería en Desarrollo de Videojuegos y Realidad Virtual*. Available online at: http://videojuegos.utalca (accessed February 20, 2025).

Laakso, N. L., Korhonen, T. S., and Hakkarainen, K. P. (2021). Developing students' digital competences through collaborative game design. *Comput. Educ.* 174:104308. doi: 10.1016/j.compedu.2021.104308

Marín-Lora, C., and Chover, M. (2025). GameScript: a simplified scripting language for video game development. *Multimed. Syst.* 31:70. doi: 10.1007/s00530-024-01574-8

Martín-Hernández, P., Gil-Lacruz, M., Gil-Lacruz, A. I., Azkue-Beteta, J. L., Lira, E. M., and Cantarero, L. (2021). Fostering university students' engagement in teamwork and innovation behaviors through game-based learning (GBL). *Sustainability* 13:13573. doi: 10.3390/su132413573

Miranda, J., Navarrete, C., Noguez, J., Molina-Espinosa, J.-M., Ramírez-Montoya, M.-S., Navarro-Tuch, S. A., et al. (2021). The core components of education 4.0 in higher education: Three case studies in engineering education. *Comput. Electr. Eng.* 93. doi: 10.1016/j.compeleceng.2021.107278

Montes, H., Hijón-Neira, R., Pérez-Marìn, D., and Montes, S. (2021). Using an online serious game to teach basic programming concepts and facilitate gameful experiences for high school students. *IEEE Access* 9, 12567–12578. doi: 10.1109/ACCESS.2021.3049690

Noone, M., and Mooney, A. (2018). Visual and textual programming languages: a systematic review of the literature. *J. Comput. Educ.* 5, 149–174. doi: 10.1007/s40692-018-0101-5

Ogegbo, A. A., and Ramnarain, U. (2021). A systematic review of computational thinking in science classrooms. *Stud. Sci. Educ.* 58, 203–230. doi: 10.1080/03057267.2021.1963580

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas, Volume 14 of Harvester Studies in Cognitive Science*. Hassocks: Harvester Press.

Raj, R., and Sharma, N. (2023). "Understanding syntax errors in C sharp using game-based techniques at tertiary level," in *Big Data Intelligence and Computing. DataCom 2022, volume 13864 of Lecture Notes in Computer Science*, eds. C. Hsu, M. Xu, H. Cao, H. Baghban, and A. Shawkat Ali (Singapore: Springer Singapore), 435–450. doi: 10.1007/978-981-99-2233-8_31

Resnick, M., Maloney, J., Rusk, N., Silverman, B., and Eastmond, E. (2009). Scratch: programming for all. *Commun. ACM* 52, 60–67. doi: 10.1145/1592761.1592779

Rojas-Valdés, P., Vidal-Silva, C., and Fuente, C. L. (2022). "Successful development of problem-solving and computing programming competences in children using arduino," in *2022 International Symposium on Measurement and Control in Robotics (ISMCR)* (Houston, TX), 1–6. doi: 10.1109/ISMCR56534.2022.9950596

Strodick, K., and Schattkowsky, T. (2024). "Visual scripting in unity: a comparative analysis of existing frameworks," in *Proceedings of the ACM/IEEE 8th International Workshop on Games and Software Engineering* (New York, NY: ACM), 44–49. doi: 10.1145/3643658.3643921

Sun, D., Looi, C.-K., Li, Y., Zhu, C., Zhu, C., Cheng, M., et al. (2024). Block-based versus text-based programming: a comparison of learners' programming behaviors, computational thinking skills and attitudes toward programming. *Educ. Technol. Res. Dev*. 72, 1067–1089. doi: 10.1007/s11423-023-10328-8

Tupac-Yupanqui, M., Vidal-Silva, C., Pavesi-Farriol, L., Sánchez Ortiz, A., Cardenas-Cobo, J., Pereira, F., et al. (2022). Exploiting arduino features to develop programming competencies. *IEEE Access* 10, 20602–20615. doi: 10.1109/ACCESS.2022.3150101

Umezawa, K., Ishii, Y., Nakazawa, M., Nakano, M., Kobayashi, M., and Hirasawa, S. (2021). "Comparison experiment of learning state between visual programming language and text programming language," in *2021 IEEE International Conference on Engineering, Technology & Education (TALE)* (Wuhan), 1–5. doi: 10.1109/TALE52509.2021.9678608

Unity - Discussions (2025). *A space to Discuss All Things Unity*. Available online at: https://discussions.unity.com/ (accessed February 20, 2025).

Vidal-Silva, C., Cárdenas-Cobo, J., Tupac-Yupanqui, M., Serrano-Malebrán, J., and Sánchez Ortiz, A. (2024). Developing programming competencies in school-students with block-based tools in Chile, Ecuador, and Peru. *IEEE Access* 12, 118924–118936. doi: 10.1109/ACCESS.2024.3449228

Vinueza-Morales, M., Cárdenas-Cobo, J., Cabezas-Quinto, J., and Vidal-Silva, C. (2025). Applying the block-based programming language Alice for developing programming competencies in university students. *IEEE Access* 13, 21471–21485. doi: 10.1109/ACCESS.2025.3536279

Vygotsky, L. S. (1978). *Mind in Society: Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press. Available in cloth, paper, and electronic formats (ePub, Mobi, PDF).

Weintrop, D., and Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Comput. Edu.* 142:103646. doi: 10.1016/j.compedu.2019.103646

Wing, J. M. (2006). Computational thinking. *Commun. ACM*, 49, 33–35. doi: 10.1145/1118178.1118215

Zhang, Y., Liang, R., Li, Y., and Zhao, G. (2022). "Improving java learning outcome with interactive visual tools in higher education," in *Artificial Intelligence in Education: Emerging Technologies, Models and Applications. AIET 2021. Lecture Notes on Data Engineering and Communications Technologies, Vol. 104*, eds. E. C. K. Cheng, R. B. Koul, T. Wang, and X. Yu (Singapore: Springer). doi: 10.1007/978-981-16-7527-0_17

Zhao, D., Muntean, C. H., Chis, A. E., Rozinaj, G., and Muntean, G.-M. (2022). Game-based learning: Enhancing student experience, knowledge gain, and usability in higher education programming courses. *IEEE Trans. Educ.* 65, 502–513. doi: 10.1109/TE.2021.3136914