Scientific Research Publishing

International Journal of Intelligence Science, 2026, 16(1), 70-82
https://www.scirp.org/journal/ijis
ISSN Online: 2163-0356
ISSN Print: 2163-0283

Visual Scaffolding in Control Structures: An Educational Proposal for Introductory Programming Courses

Ana Lilia Laureano-Cruces[1], Orlando Martín Pérez-Hernández[2], Ismael Martínez-Bonilla[3], Lourdes Sánchez-Guerrero[1]

[1]Departamento de Sistemas, Universidad Autónoma Metropolitana-Azcapotzalco, CDMX, México
[2]Licenciatura en Ingeniería en Computación, Universidad Autónoma Metropolitana-Azcapotzalco, CDMX, México
[3]Laboratorio de Educación y Evaluación Digital, Facultad de Estudios Superiores Iztacala, UNAM, CDMX, México
Email: clc@azc.uam.mx, orlpema@gmail.com, ismael.m.bonilla@iztacala.unam.mx, lsg@azc.uam.mx

How to cite this paper: Laureano-Cruces, A.L., Pérez-Hernández, O.M., Martínez-Bonilla, I. and Sánchez-Guerrero, L. (2026) Visual Scaffolding in Control Structures: An Educational Proposal for Introductory Programming Courses. *International Journal of Intelligence Science*, **16**, 70-82.
https://doi.org/10.4236/ijis.2026.161004

Received: November 6, 2025
Accepted: December 16, 2025
Published: December 19, 2025

Abstract

Introductory programming courses involve the development of skills related to analysis, reasoning, and comprehension. Therefore, making execution visible (which instruction runs, which branch is taken, and how variables change) can facilitate early understanding. This work presents a proposal based on Information Visualization to help understand how control structures work. The objective of this study was to design a microworld that allows step-by-step observation of the functioning of control structures (sequence, simple and multiple selection, arithmetic progression, while, and repeat-until). These structures are part of Structured Programming. An application was developed in Java with four modules: 1) microworld environment, 2) visual interface for structures, 3) parameter input, and 4) visualization. The microworld was modeled as a 15 × 15 grid (30 × 30 px) mapped to a matrix, with a GUI (Swing) that captures conditions, selectors, and initial values. Animation was carried out through timers using metaphorical actions (walking, running, hitting) and energy/life variables. As a result, a prototype was developed that visualizes each structure with its specific semantics and allows users to set parameters and observe effects in real time. The modular architecture and timed rendering provide fluidity and scalability. It is concluded that the microworld translates code logic into reproducible visual cues, reduces costly inferences, and supports stable mental models for flow control.

Keywords

Structured Programming, Information Visualization, Microworld, Control Structures, Mental Models

1. Introduction

In the early years of systems and computer science programs, programming is the pillar of computational thinking and solution design because it requires abstracting, modeling, and automating processes with logical rigor—for example, linking syntax and semantics, tracing program state [variables and conditions], mastering control flow [conditionals and loops], understanding scope and typing, and understanding debugging and error reading [1]. Learning these skills, together with others such as translating problems into algorithms, understanding that the computer follows literal instructions, and reconciling a new syntax with the logic of flow, can be challenging [2].

In this sense, variables, control structures, functions/methods, input/output, and initial debugging form the foundation of computing education, and the way these topics are taught [activities, assessments, and resources] shapes what students will be able to learn later [1]. Thus, seeing execution and tracing how variables change at each step is especially useful for understanding control transitions and, at the same time, for facilitating problem solving [3].

Students often form incomplete mental models of these concepts [what values variables hold and when they change], which leads to typical errors in conditionals and loops. In addition, feedback in some cases tends to arrive as cryptic compiler messages, increasing frustration [2]. Hence, the need to use support that makes the flow and the step-by-step execution of what they are solving visible. For this reason, it is important to employ scaffolds and visualizations that make execution visible and help build correct mental models from the first years of the program [3] [4].

A mental model is understood as the internal, functional representation that the student constructs of the "machine" that executes a program, that is, how the instructions are evaluated, how control flows, and how the variables change. A correct mental model is consistent with the semantics of the language and makes it possible to predict and explain the state of the program step by step in new cases. In contrast, an incomplete or incorrect mental model contains gaps or faulty rules. In this sense, students learn better when their models are explicitly aligned with a clear structure and when recurrent misconceptions are corrected [4] [5].

Building on this foundation, we propose an Information Visualization project to observe what happens during a program's execution—that is, which instructions run, which path the flow takes, how variables change, and how to turn those execution data into useful information and, ultimately, knowledge [6] [7]. Visualization requires representing and presenting data in diverse forms to leverage visual perception and generate knowledge, and sets as a goal the design of an educational tool that incorporates these principles to improve the learning of structured programming [8].

In practical terms, visualizing execution serves several functions that help teach students more effectively: explain [tell the "story" of the code step by step], simplify [remove distractors and make the essentials evident], compare [contrast ex-

ecution paths or parameter effects], and explore [try alternatives and see consequences instantly]. Empirical evidence indicates that visualizations and step-by-step traces have been associated with better understanding of basic programming concepts and with the construction of more stable mental models, especially in early stages. Moreover, interactive tools facilitate this, which helps explain why "seeing" the flow and state speeds the move from data to understanding [3] [6] [7].

Among the studies reporting how visualization helps improve learning is the work [9], which, in astrophysics, implemented an interactive interface based on qualitative reasoning to represent complex phenomena; the aim was for users to observe relationships and behaviors that are difficult to infer from formulas or text alone. In a different domain [10] implemented the visualization of an affective behavior through mythological narrative, exploring how a cognitive-affective model can be represented in a comprehensible way, placing the observer before internal processes that usually remain hidden.

On the other hand, research has examined how the design of visual elements and icons can strengthen the connection with people less familiar with technology [for example, older adults], thus functioning as a bridge between abstract concepts and concrete actions [11]. In parallel, information-visualization systems have been developed to present institutional academic data clearly to detect patterns and potential issues: a web system to analyze performance in Teaching-Learning Units and a project to present academic statistics with admissions criteria are examples of how graphical representation helps to understand complex datasets and make informed decisions [12] [13].

Closer to our objective, the work on virtual microworlds by Laureano-Cruces *et al.* [14] proposes visualizing abstract concepts of structured programming through scenarios inspired by everyday life and intended to be integrated into intelligent tutoring systems. This line is particularly relevant because it suggests that narrowing a pedagogical domain [objects, properties, and events] and dynamically representing control flow can help novice students build more stable mental models. This project situates itself in that tradition and extends it with an implementation focused on control structures [sequence, decision, and loops], incorporating a simulation interface that allows parameters to be configured and, step by step, shows which branch executes, how many iterations occur, and how the state changes—with the explicit aim of supporting early understanding in structured programming.

Building on this background, we propose a microworld implemented as a simulation interface that operationalizes these ideas into tasks students can manipulate. This microworld serves as support for first-year students in computer systems engineering, computer science, or related fields. Unlike prior work, here we explicitly bound the domain [objects, properties, and events] to control structures and aligned the visualization with clear instructional actions [set parameters, predict behavior, observe step-by-step execution, and verify the result].

This microworld integrates a "field" in which objects, properties, and events are

clearly defined; this field is deliberately limited to the functioning of control structures [sequence, simple and multiple selection, and loops], so that students interact only with the conceptual elements that matter for learning control flow and program state [14]. Finally, the "objects" act as a bridge between theory and execution, since they display behavior visibly and respond to the program's decisions and loops; thus, as the flow advances, the object moves and "shows" the consequence of the condition or the counter.

On this basis, the microworld presented covers the following control structures: Sequential—shows the ordered execution of actions "one after another," as the object travels across the map from bottom to top and visually activates each instruction when it is executed, making the order and the top-to-bottom flow of the code [start-end] evident; Simple selection—the student sets the logical condition [true/false] and the object takes the corresponding branch, with the option to adjust parameters and actions associated with each case, turning "if/else" into a step-by-step observable experience; Multiple selection [when there are more than two options]—the focus is on the selector [the value of an enumerated type], where the student defines that value and the case-specific actions, including the default path when there is no match; the branching and the fallback to the default become visible in the object's trajectory.

Unlike algorithm visualizers that show complete executions or data structures, and block-based environments that replace syntax with graphical pieces, this microworld intentionally narrows the domain to CS1 control structures [sequence, selection, and loops] and makes them observable and manipulable without textual programming, so that students can adjust to didactic parameters [condition, selector, counter] and observe step-by-step transitions through simple metaphors that externalize the state.

For loops, three behaviors are distinguished. In an arithmetic progression, the finite nature of the loop is emphasized: an N and a mode are set [increment/decrement with a constant step], and the object traces a cyclic movement [clockwise or counterclockwise, depending on the mode] that marks each iteration upon returning to the starting point. In the while-repetition structure, entry to the loop depends on evaluating a logical condition at the beginning of each pass; the object pauses to "ask" whether to continue or stop, so the student clearly sees when the condition allows continuation and when it does not. Finally, in repeat-until, at least one iteration is guaranteed before evaluation, keeping parameters and movement similar to the "while" but with the decision at the end of the cycle, which allows both repetition patterns to be contrasted in practice. In all cases, the object serves as a bridge between logic and execution: its movement, color changes, and evaluation pauses translate the abstract flow into clear signals about state and control transitions that students can follow at a glance.

Finally, this tool is grounded in Vygotsky's pedagogical theory, in which the tool functions as a scaffold between the student's prior learning and new concepts. Scaffolding is understood as the set of temporary supports offered within the Zone

of Proximal Development [ZPD] so that the student can carry out tasks they are not yet able to solve independently and that, with appropriate guidance, are progressively internalized [15]. In design terms, our tool incorporates classical scaffolding principles by making the control flow and state visible as a sequence for predicting, observing, and explaining, and by allowing supports to fade as the student learns.

Accordingly, the central purpose is to design an educational tool to improve the learning experience in structured programming by making visible what is usually abstract during a program's execution. This tool is based on visualization to transform execution data into information and knowledge, and thereby facilitate the learning of basic structures [sequence, decision, loops] in introductory courses.

2. Methodology

2.1. Approach and Overall Architecture

The resource was implemented as an interactive microworld to make control flow and program state visible. The system was organized into four modules that structure the end-to-end user experience: 1) definition of the microworld environment, 2) a visual interface that represents control structures, 3) user parameter capture, and 4) execution visualization. This arrangement guided both development and integration testing.

2.2. Platform and Completion Criteria

The project was developed in Java, considering graphics libraries (LibGDX, JavaFX, LWJGL) for display and animation. Completion criteria were defined per module: 1) The environment had to exhibit the required appearance and functionality; 2) The interface had to faithfully represent the semantics of each structure; 3) Parameter capture had to validate and correctly input the data; 4) Integration had to allow the full execution to be observed without errors. These criteria were used as a checklist during integration.

2.3. Microworld Representation

To make flow/state observable, the microworld was modeled as a gridded mesh mapped to a two-dimensional array, where each pair of indices identifies a map cell. Visually, cells are represented with simple characters (e.g., "o"), and the environment has fixed dimensions of 15 × 15 cells, each 30 × 30 pixels. This decision balances visual clarity and precision for tracing (**Figure 1**).

This division simplifies the representation and avoids irrelevant details, reducing extraneous load and allowing attention to focus on control transitions (sequence and iterations). In parallel, metaphorical actions such as walking, running, and hitting (which will be described later) build on the student's prior schemata (with intuitive meanings and "actions") to immediately map the effects on the state (energy/life) without introducing additional technical vocabulary.

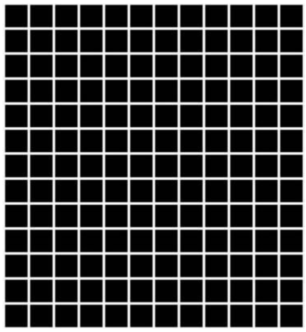
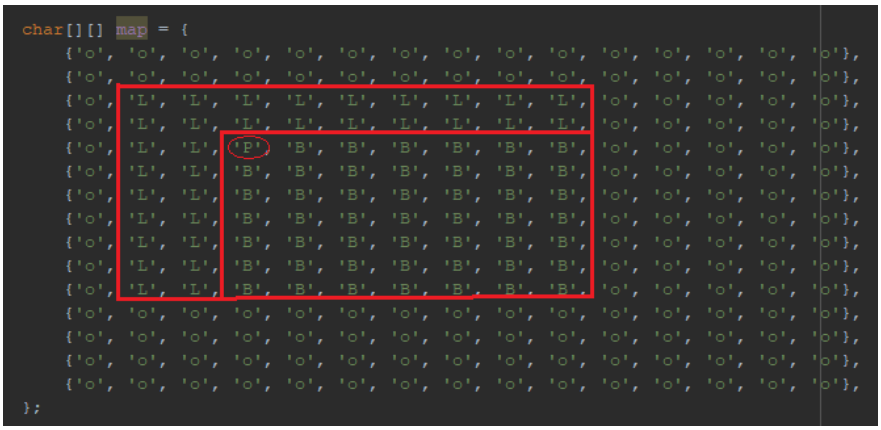


Figure 1. Structure of the microworld.

The agent's position and movement derive from that array (index → pixel conversion). For example, cell [3] [5] becomes (x = 150, y = 90)-useful for fine control of step-by-step advancement. Conversely, for inspection or collisions, a pixel → index conversion is performed. These transformations enable instrumenting different paths according to the control structure (Figure 2).



```
char[][] map = {
    {'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'L', 'L', 'P', 'B', 'B', 'B', 'B', 'B', 'B', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'L', 'L', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'L', 'L', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'L', 'L', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'L', 'L', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'L', 'L', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'L', 'L', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o'},
    {'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o', 'o'},
};
```

Figure 2. General structure of each representative map in the microworld.

From the base array, representative shapes were built for each structure, and simple markers were used to define the logical-visual "map" (example: L = colored section, B = walkable area, P = agent's starting position). With this coding, unique patterns were created per structure, differentiating key zones, movement areas, and spawn point.

2.4. User Interface and Parameter Capture

The GUI was implemented with Swing. Each structure view includes controls to set logical conditions, initial values (e.g., life/energy), iteration parameters (increment/decrement mode, N), and actions per branch/iteration; it also includes buttons to set, start, and reset execution. The values entered by the user are converted to the types required by the visualization engine.

2.5. Object/Agent and State Variables

The "object" used is a red circle that bridges logic and execution. It has two attrib-

utes: energy (affected by the action) and life (used in logical evaluations of conditional structures and loops). Three metaphorical actions were defined (walk, run, hit) that consume 2, 10, and 20 energy units, respectively. The life variable is incremented/decremented to alter the outcome of the condition when appropriate (for example, to continue/stop a loop).

2.6. Class Model Structure and Responsibilities

To represent the structures, a class model with well-defined responsibilities was established; this was implemented via a JPanel component, and the following classes were defined:

- Map: Generates the drawing of the environment from the array (includes shapes, walkable areas, and key points).
- Object: Manages the agent's state and movement on the map; each structure has its own variant of the move method.
- Panel: Contains map and object; coordinates rendering, parameter capture, and the execution cycle. This arrangement is replicated for each structure, and the full class diagram is included in the appendices.

3. Results

User Flow

The application starts with a main menu (Figure 3), from which one or several structures can be selected for concurrent visualization. This flow supports quick hypothesis testing ("if I change the selector/condition, what happens?") and comparison across structures.

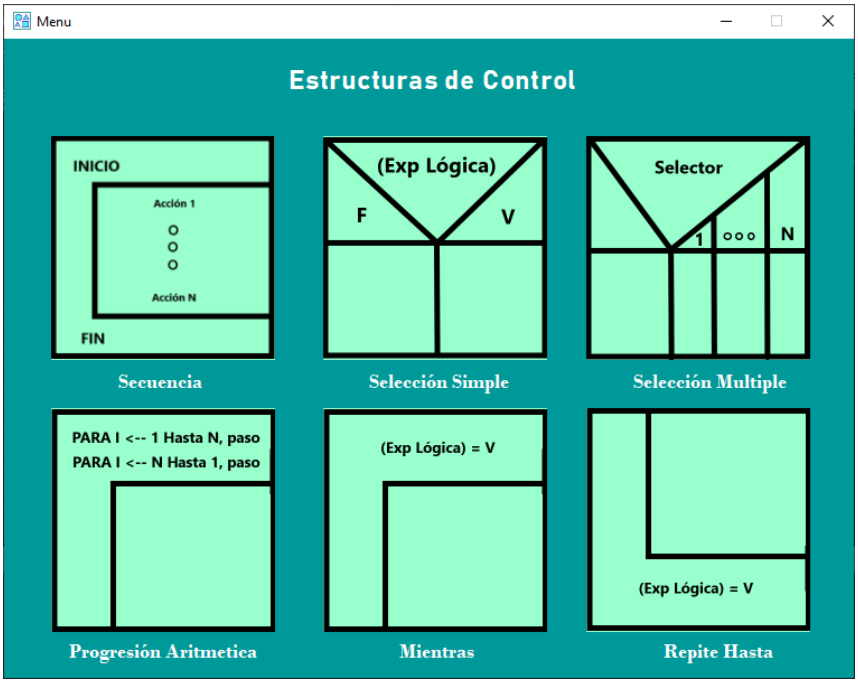


Figure 3. Screenshot of the application's main menu.

The sequence of the agent shows in the moves from bottom to top, activating the action associated with each step (Figure 4). Objective: to make the order and directionality of execution (start-end) evident.

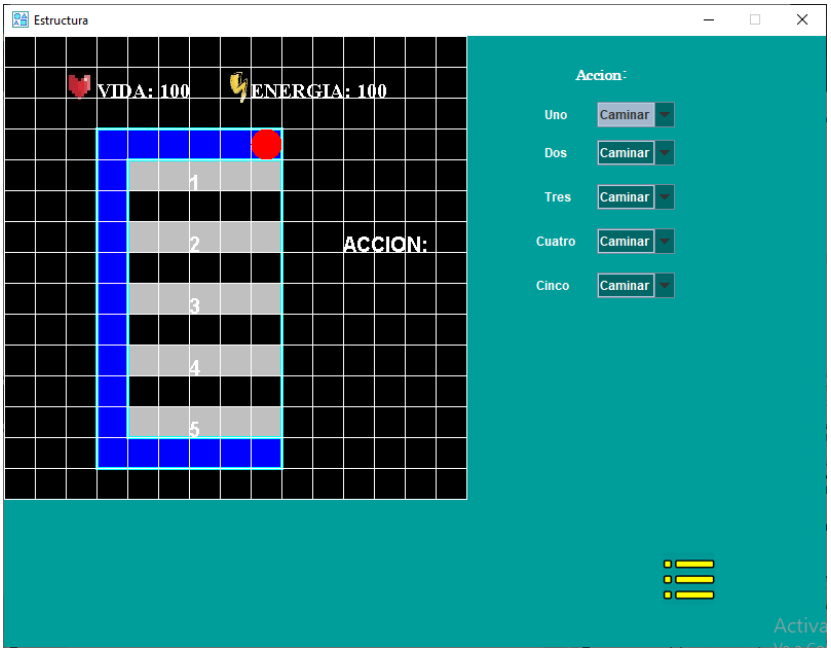


Figure 4. Visualization of the sequential structure.

Simple selection: at the branch, the system evaluates the condition and the agent takes the true (right) or false (left) path. Available parameters: life, energy, logical condition, and action per branch (Figure 5).

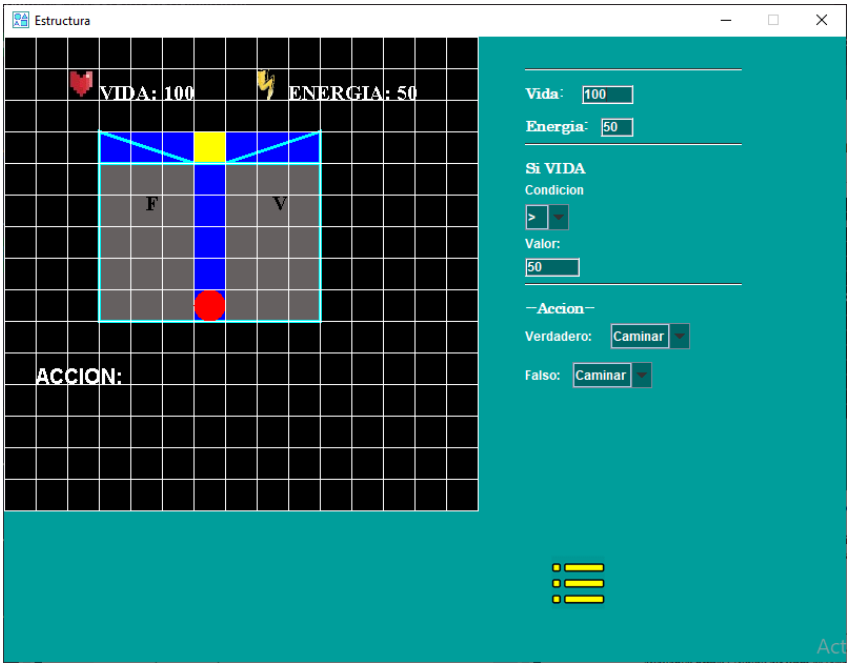


Figure 5. Visualization of the simple selection structure.

Multiple selection: the user sets the selector's value and the action for each case; if there is no match, the default route is executed (Figure 6).

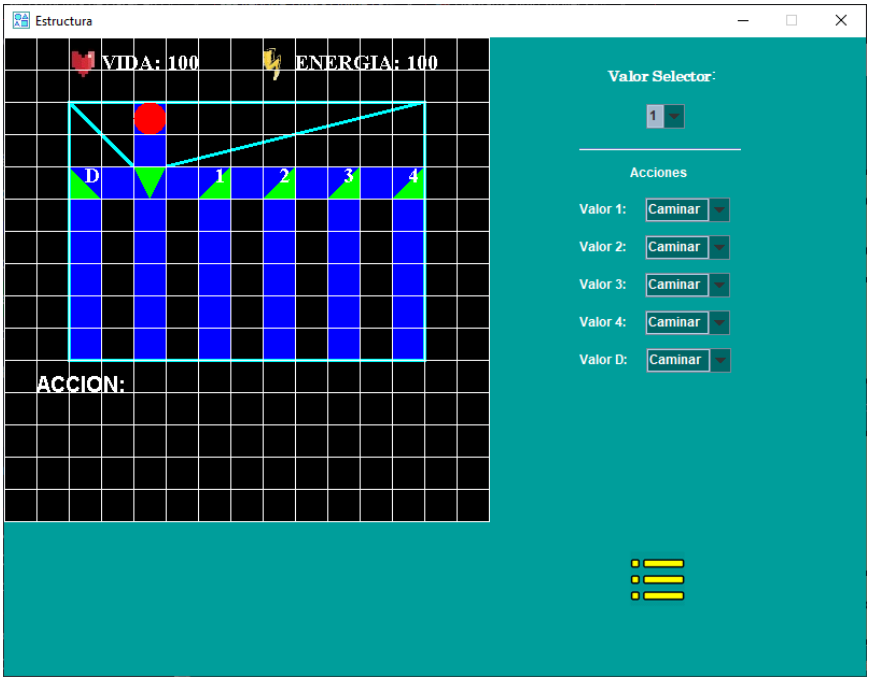


Figure 6. Visualization of the multiple selection structure.

Arithmetic progression: the agent traces a cyclic movement—rightward if incrementing and leftward if decrementing. One iteration is completed upon passing the starting point (Figure 7).

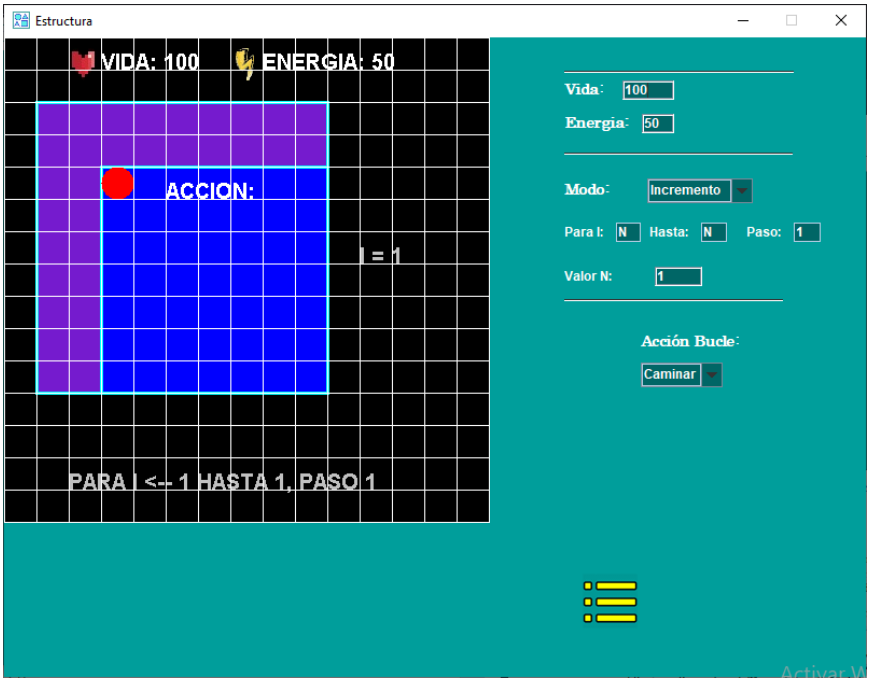


Figure 7. Visualization of the arithmetic progression structure.

While: at the start of each pass, the agent evaluates the condition; if true, it continues; if false, it stops (Figure 8). Parameters: life, condition, and action per iteration.

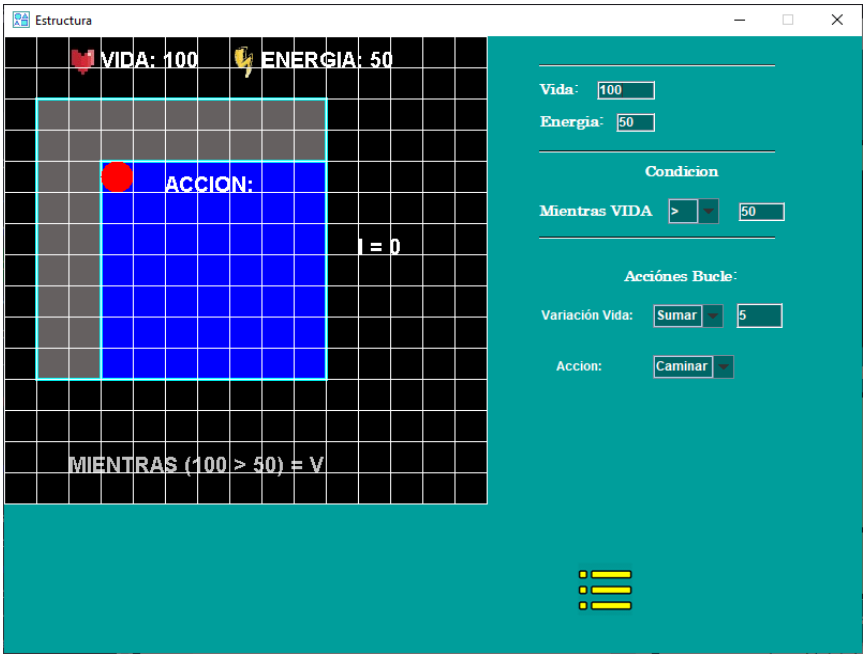


Figure 8. Visualization of the while structure.

Repeat-Until: guarantees at least one iteration and evaluates the condition at the end of the cycle, keeping parameters and movement analogous to "while" (Figure 9).

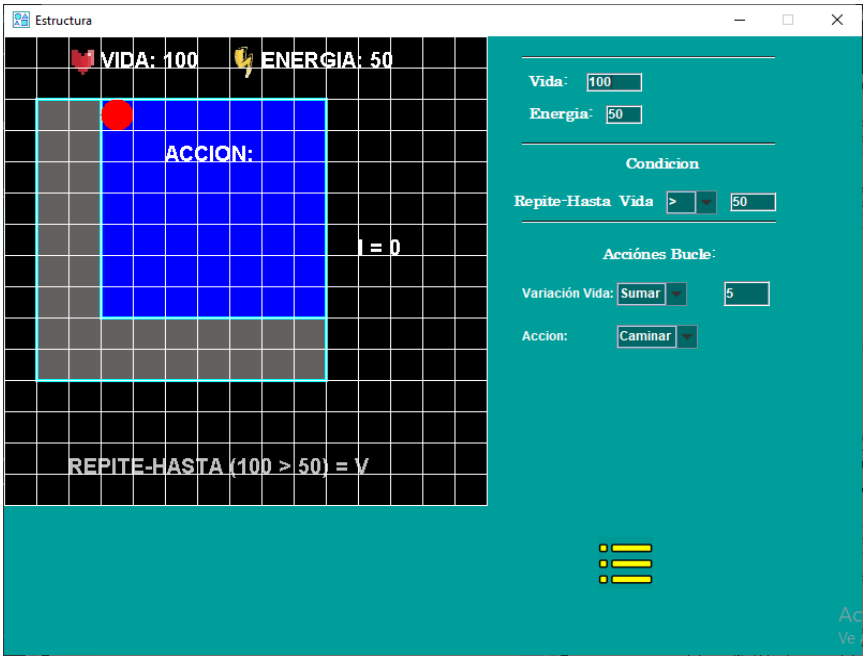


Figure 9. Visualization of the repeat-until structure.

An important point is that encapsulating repainting inside a continuous while loop produced an "instant" visualization of the traversal, with no observable intermediate steps, which negated the tool's didactic value. The solution was to adopt timers that schedule position updates and repainting at millisecond intervals, thus achieving incremental, controlled execution animation. This decision also allowed the inclusion of supportive effects (e.g., color changes) that reinforce the reading of the flow without saturating the scene.

Finally, each phase was declared complete upon meeting the predefined criteria: 1) environment with the expected appearance/functionality, 2) interface faithful to the semantics, 3) valid/effective parameter capture, and 4) stable integration that lets users observe the full execution and the effects of each parameter on the agent's behavior.

4. Discussion

The microworld developed in this article demonstrates that it is possible to visually represent traditionally abstract concepts in structured programming, provided that elements and techniques that stimulate user perception are combined (simple shapes/colors, object trajectories, recognizable actions). This strategy not only makes the semantics of control structures visible but also enables real-time exploration and verification practices.

The modular architecture, grid-based array, and timed rendering loop constitute a "technical scaffold" that enables step-by-step observation of execution and student manipulation of parameters. These decisions translate code logic into clear, reproducible visual cues, fostering early understanding and confidence when entering the world of structured programming.

From a pedagogical perspective, information visualization acts as a mediator between perception and reasoning, as it reduces the inferences students must make and offers a concrete basis for thinking. When visual elements maintain a stable correspondence among shape, color, position, and movement, the semantic anchoring of abstract concepts is strengthened. Likewise, integrating common actions such as walking, running, and hitting adds a layer of meaning close to everyday experience, making it easier to remember rules and supporting the construction of mental models.

Another point worth highlighting is that Java—a high-level language applied to didactic, low-complexity scenarios—together with the combination of timers, a clear class model, and a well-defined internal structure, is sufficient to achieve a smooth and formative visual experience.

5. Conclusions

As a future line of work, it is advisable to apply the project to a group of students in order to obtain empirical support for its functionality; this would increase its reliability and validity as a scaffolding tool for novice programming students. Likewise, it is important to consider that this tool addresses a specific set of struc-

tures (sequence, simple/multiple selection, and loops) without nesting or combined cases, and it does not cover other key aspects of execution (e.g., memory, scope, method calls/recursion, error handling, or concurrency), so it cannot be generalized to other contexts.

On the other hand, the selected visual metaphors and actions (walking, running, hitting), together with the 15 × 15 grid and the timed rendering, prioritize clarity over realism; however, they may not be universal across different cultural contexts nor fully accessible (e.g., color blindness, screen readers). Finally, this tool, in accordance with learning styles, can be integrated as part of an Intelligent Tutoring System, supporting the teaching-learning process.

Acknowledgements

This work represents the research conducted by Orlando Martín Pérez-Hernández to obtain a Bachelor's degree in Computer Engineering from Universidad Autónoma Metropolitana-Azcapotzalco. It is also part of the divisional project, "Design of Intelligent Interfaces for Simulating the Behavior of Living or Animate Organisms," from the same University.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Laureano-Cruces, A.L. and Barceló-Aspeitia, A.A. (2003) Formal Verification of Multi-Agent Systems Behaviour Emerging from Cognitive Task Analysis. *Journal of Experimental & Theoretical Artificial Intelligence*, **15**, 407-431. https://doi.org/10.1080/0952813031000119719

[2] Yáñez-Castillo, J.A., Laureano-Cruces, A.L. and Garmendia-Ramírez, F. (2017) Diseño emocional de elementos interactivos visuales: Una perspectiva para mejorar la interacción de los inmigrantes digitales con la computadora personal. *Revista Tecnología & Diseño*, No. 8, 37-55.

[3] Mladenović, M., Žanko, Ž. and Aglić Čuvić, M. (2020) The Impact of Using Program Visualization Techniques on Learning Basic Programming Concepts at the K-12 Level. *Computer Applications in Engineering Education*, **29**, 145-159. https://doi.org/10.1002/cae.22315

[4] Qian, Y. and Lehman, J. (2017) Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Transactions on Computing Education*, **18**, 1-24. https://doi.org/10.1145/3077618

[5] Sorva, J. (2013) Notional Machines and Introductory Programming Education. *ACM Transactions on Computing Education*, **13**, 1-31. https://doi.org/10.1145/2483710.2483713

[6] Vuckovic, M. and Schmidt, J. (2022) On Sense Making and the Generation of Knowledge in Visual Analytics. *Analytics*, **1**, 98-116. https://doi.org/10.3390/analytics1020008

[7] Wall, E., Agnihotri, M., Matzen, L., Divis, K., Haass, M., Endert, A., *et al.* (2019) A Heuristic Approach to Value-Driven Evaluation of Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, **25**, 491-500. https://doi.org/10.1109/tvcg.2018.2865146

[8] Viana, V.C. (2020) Sistema web para la obtención y visualización de información sobre Unidades de Enseñanza y Aprendizaje. Proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, Mexico.

[9] Sánchez, B.R. (2020) Presentación de estadísticas académicas con base en criterios de ingreso utilizando técnicas de visualización de información. Proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, Mexico.

[10] Omer, U., Farooq, M.S. and Abid, A. (2021) Introductory Programming Course: Review and Future Implications. *PeerJ Computer Science*, **7**, e647. https://doi.org/10.7717/peerj-cs.647

[11] Yáñez-Castillo, J.A. (2016) De lo efectivo a lo afectivo. Master's Thesis, Universidad Autónoma Metropolitana.

[12] Torres-Velasco, E.O. (2021) Visualización a través del razonamiento cualitativo: un fenómeno de astrofísica. Master's Thesis, Universidad Autónoma Metropolitana.

[13] Laureano-Cruces, A.L., Serrano-Muñoz, R., Castillo-Bernal, A.S., Sanchez-Guerrero, L. and Ramírez-Rodríguez, J. (2025) Empathy through the Visualization of Emotions: Story of a Toltec Prince. Association for the Advancement of Computing in Education (AACE), 634-643. https://www.learntechlib.org/primary/p/211137/

[14] Laureano-Cruces, A.L., Yáñez-Castillo, A., Sánchez-Guerrero, L., Ramí-rez-Rodríguez, J. and Mora-Torres, M. (2018) Examples for a Virtual Micro World: Visualization of Abstract Concepts. *World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, Las Vegas, 15-18 October 2018, 1432-1437.

[15] Vygotsky, L.S. (1978) Mind in Society: The Development of Higher Psychological Processes. Harvard University Press.