INNOVATIONS IN MACHINE INTELLIGENCE
NATURAL PRINCIPLES AND ARTIFICIAL MODELS

Innovations in Machine Intelligence (IMI), 2025, 1–14

doi: 10.54854/imi2025.01

RESEARCH

Vegetables, Enemies, and Emergence: A Playable Game of Life on Nintendo for Educational Purposes

Bruno Senzio-Savino Barzellato [1, 2] and Faramarz Alsharif [3, *]

[1]SenzioTek S. de R.L. , Fray Junipero Serra 149, Padua 88 , Paseo San Junipero, Queretaro, Queretaro , 76146 , https://www.senzio-tek.com
[2]Graduate School of Science and Engineering, Yamagata University, Japan (Invited Researcher)
[3] Graduate School of Science and Engineering, Department of Informatics and Electronics , Electrical and Electronics Communication Engineering , Yamagata University, 4-3-16, Jonan, Yonezawa City, Yamagata Prefecture 992-8510

[*]Corresponding author: asharif@yz.yamagata-u.ac.jp

Abstract

Conway's Game of Life is a foundational model in cellular automata and artificial life research, with implementations spanning mainframes, personal computers, calculators, and modern interactive platforms. However, only a handful of attempts have been made to bring real-time Life simulation to the Nintendo Entertainment System (NES) or the Famicom, primarily as non-interactive technical demonstrations constrained by the console's limited memory, lack of a frame buffer, and strict PPU (Picture Processing Unit) timing. No prior work has integrated a full cellular automaton into a playable, narrative-driven game under authentic 8-bit hardware conditions. This article presents a novel, fully interactive implementation of the Game of Life embedded within a two-phase NES/Famicom game. Developed through a hybrid workflow combining NESmaker for high-level scene management and CA65 assembly for low-level, cycle-accurate logic, the system enables a real-time Life simulation on a 5×5 grid executed entirely within the console's VBlank (Vertical Blank Interval) constraints. Player actions in Phase 1 (a vegetable-collection scroller with AI-driven enemies, NPC interactions, and variable risk-reward dynamics) directly determine the initial state of the automaton in Phase 2, where optimized palette transitions, serpentine grid traversal, and compact state encodings make real-time updates feasible despite 2 KB RAM (Random Access Memory) and tile-based rendering limitations. The result is a console-native artificial-life game: a system in which emergent behavior, deterministic rules, and interactive gameplay coexist on hardware never designed for such computational tasks. Beyond demonstrating technical feasibility, this work shows how constrained retro platforms can serve as powerful educational tools for teaching discrete mathematics, complexity, and A-Life (Artificial Life) concepts through embodied, exploratory play.

Key words: cellular automata; Conway's Game of Life; Nintendo Entertainment System; Famicom; retro game development; game-based learning

Introduction

Cellular automata have long provided a compelling framework for understanding emergent behavior in discrete dynamical systems. Conway's Game of Life, introduced by Gardner in 1970 [1], remains the canonical example of how simple local rules can generate complex, large-scale structures. Since then, cellular automata have been used extensively to model computation, biological growth, and artificial life (A-Life) [2, 3, 4]. These systems are particularly valued for demonstrating how algorithmic simplicity can lead to unexpected and interpretable emergent behavior.

Artificial Life (A-Life). The Game of Life occupies a central position in A-Life research as an archetype of bottom-up complexity. Classic work in the field emphasizes that emergent structure arises from local interactions rather than top-down control [3, 4]. Embedding such a system inside a playable game environment enables a hybrid form of computational interaction in which player decisions influence the initial state of an autonomous dynamical process.

Received: 15 November 2025, Accepted: 27 January 2025, Published online: 21 March 2026.

Human–Computer Interaction on Retro Hardware. Constraints in older systems such as the Famicom/NES introduce unique HCI characteristics. Because the NES lacks a frame buffer, imposes strict timing windows for VRAM (Video RAM) access, and limits sprite rendering to eight sprites per scanline [5, 6], developers must design interfaces that respect the architecture's deterministic timing. These constraints create an environment where algorithmic expressiveness and player interaction must be carefully co-designed [7].

Retrocomputing and Historical Preservation. Retro hardware has experienced renewed academic attention. Researchers argue that older systems represent historically significant computational models whose technical limitations can reveal fundamental insights about algorithmic design [8]. The contemporary homebrew movement has transformed retro platforms into experimental environments for artistic expression, pedagogy, and unconventional computing [9].

Game Design and Emergence. Unlike classical arcade games—which typically emphasize deterministic reward structures and fixed outcome patterns—emergence-based game design leverages non-linear behaviors that arise from the interaction of simple rules and player actions within a system [10]. By linking vegetable collection (Phase 1) with the initialization of a Game of Life grid (Phase 2), this project creates a dependency between action-based gameplay and emergent simulation.

Educational Applications of Retro Games. Studies have shown that retro-style interfaces can stimulate curiosity and learning in ways that differ from modern high-fidelity environments [11]. The NES, with its iconic aesthetic and tactile simplicity, is particularly effective for introducing computational ideas such as discrete mathematics, cellular automata, and basic AI behaviors to younger learners.

In light of these intersections, this work presents a complete, fully playable implementation of Conway's Game of Life as a core game mechanic on the Famicom/NES. Unlike previous attempts that treated Life as a static technical demo, the system presented here embeds the automaton into a multi-phase gameplay loop where the player's strategic choices materially shape emergent outcomes. The result is a hybrid of A-Life simulation, retrocomputing practice, and computational learning through interactive play.

The NES/Famicom Hardware Constraints

The Nintendo Entertainment System (NES) and its Japanese counterpart, the Famicom, operate under severe architectural limitations that strongly influence all aspects of rendering, memory management, and algorithmic behavior. These constraints have been extensively documented in platform studies and hardware analyses [8, 12, 5]. Understanding these limitations is essential when embedding real-time cellular automata into a playable game loop.

The most relevant constraints include:

- **2 KB of system RAM**, shared between gameplay state, stack operations, temporary buffers, and pseudo-random seeds.
- **No frame buffer**: the PPU renders directly from name tables and CHR (Character ROM/RAM) memory, allowing background changes only during the restricted VBlank window [5].
- **Sprite limits**: 64 total sprites and a strict maximum of 8 sprites per scanline, producing flicker if exceeded [8].
- **Palette constraints**: four background palettes and four sprite palettes, each providing three visible colors plus transparency [9].
- **Fixed tile sizes**: 8x8 or 8x16 pixel tiles, with visual composition determined entirely by CHR pattern layout.

Hybrid Development Workflow: NESmaker + CA65

This project was developed using a hybrid workflow that combines:

Figure 1. Homebrew NES cartridge with game manual and box art

- **NESmaker** [13] as the high-level framework responsible for managing screen definitions, collision maps, basic asset pipeline, and event hooks; and
- **custom CA65 assembly routines** integrated into NESmaker's backend to implement low-level logic, optimized PPU updates, and the entire Game of Life simulation kernel.

NESmaker provides a structured environment for defining screens, entities, and tilesets, enabling rapid iteration and data-driven organization. However, its scripting layer and built-in virtual machine are insufficient for real-time cellular automata or cycle-counted PPU operations. Therefore, this work extends the NESmaker engine with:

- hand-authored 6502 routines for neighbor counting, grid traversal, and state transitions,
- custom VRAM update logic inserted into the NMI handler,
- serpentine mapping–based traversal to minimize cursor movement and VRAM access,
- palette-encoded state changes to avoid full tile rewrites,
- optimized CHR usage to keep attribute updates within VBlank limits.

Implications for Real-Time Cellular Automata

Running a cellular automaton such as Conway's Game of Life on NES hardware normally exceeds timing limits, as naive implementations require:

- double buffering or temporary storage for two grid generations,
- full-screen background updates each iteration,
- OAM edits for visible cursors and interactive elements.

By combining NESmaker's structured scene management with CA65-level optimizations, the system presented here achieves real-time Game of Life evolution on authentic hardware without exceeding VBlank constraints [14]. This hybrid model demonstrates that high-level tooling and low-level programming can work together to expand the expressive capabilities of 8-bit consoles far beyond their originally intended design.

Development Tools for NES Homebrewing

Modern NES homebrew development relies on a diverse ecosystem of cross-platform toolchains, assemblers, ROM (Read-Only Memory) builders, and debugging environments. These tools allow developers to work within the tight constraints of the NES architecture while enabling behaviors originally thought impractical for 8-bit hardware.

- `ca65` assembler from the CC65 suite [15], used for 6502 assembly programming and linker-based ROM construction.
- NES Screen Tool [16], widely employed for generating CHR graphics, palettes, name tables, and tile-based backgrounds.
- FamiTracker [17], a tracker-style audio tool used to produce NSF music compatible with the NES APU.
- Emulators such as FCEUX and Mesen [18], providing PPU viewers, OAM inspectors, breakpoint debugging, and trace logging essential for cycle-accurate optimization.
- NESmaker [13], a graphical ROM-building environment enabling non-programmers to create NES games through data-driven scripting and asset pipelines.
- GameMaker Studio [19], often used for prototyping NES-style logic and gameplay before translating the mechanics into cycle-accurate 6502 assembly.

Together, these tools form a modern development workflow that bridges accessible design environments with low-level hardware programming. In this project, a fully custom `ca65` codebase was created, including hand-optimized PPU update routines, cycle-counted logic for the Game of Life kernel, and bespoke memory-bank layouts. NESmaker and GameMaker were used only for preliminary prototyping and asset exploration, whereas the final implementation is written entirely in cycle-accurate assembly for authentic hardware execution. Figure 1 shows the physical NES homebrew cartridge with it's box and game manual for educational purposes utilizing a diversified pedagogical approach.

Game of Life Overview

Conway's Game of Life, introduced in 1970 in Martin Gardner's "Mathematical Games" column [1], is a two-dimensional cellular automaton defined on an infinite orthogonal grid. Each cell exists in one of two possible states—alive or dead—and the global evolution of the system emerges from the synchronous application of four simple rules. Despite this simplicity, the system is capable of producing complex behaviors, long-range interactions, self-replication, and computational universality [20, 2, 21].

The rules governing the update of each cell can be formally expressed in terms of the Moore neighborhood (the eight surrounding cells):

For populated (alive) cells:

- A cell with 0–1 neighbors dies of underpopulation.
- A cell with 4 or more neighbors dies of overpopulation.
- A cell with 2–3 neighbors survives to the next generation.

Figure 2 presents these rules in a graphical fashion.

For empty (dead) cells:

- A cell with exactly 3 neighbors becomes alive (reproduction).

Together, these transitions produce a rich taxonomy of emergent phenomena:

1. Still Lifes

Stable formations that remain unchanged across generations (e.g., *block*, *beehive*). These represent equilibrium solutions of the automaton's update function.

2. Oscillators

Patterns that repeat in cycles of period $p \geq 2$, such as *blinker*, *toad*, and *pulsar*. Oscillators illustrate periodic attractors in the system's phase space.

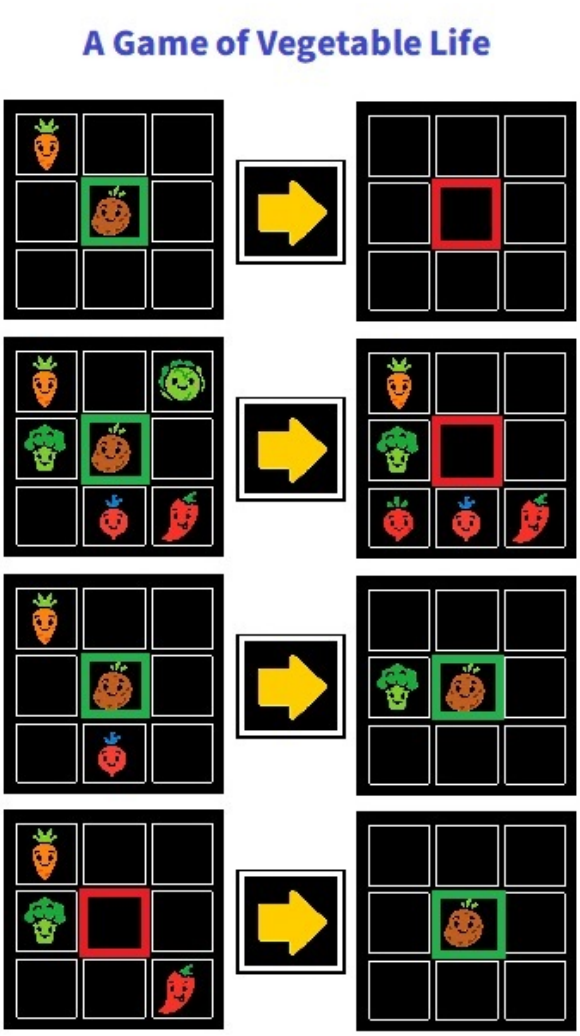

Figure 2. Example representation of conventional Conway's Game of Life patterns

3. Spaceships

Translating patterns such as the *glider* and *lightweight spaceship (LWSS)*, which propagate across the grid. Spaceships demonstrate non-local information transfer and momentum-like behavior.

4. Chaotic and Long-Lived Structures

Configurations capable of extremely long transients, pseudo-random behavior, or computational processes. Examples include *breeders*, *guns*, and *rakes* [22].

5. Computation and Universality

Life is Turing-complete: it can simulate arbitrary computation through the composition of logic gates built from glider streams and stable reflectors [21]. This makes it one of the earliest and most accessible examples of universal computation emerging from minimalist rules.

Pedagogical Relevance

The system's mix of simplicity and emergent complexity makes it ideal for demonstrating:

- discrete dynamical systems,
- local vs. global interactions,
- complexity and self-organization,

- algorithmic determinism vs. unpredictability,
- fundamental principles of artificial life (A-Life).

In the context of this project, the Game of Life is transformed into a playable simulation where player-collected vegetables initialize the grid. This bridges intuitive game mechanics with formal computational concepts, allowing learners—especially younger audiences—to interact directly with emergent phenomena through a retro hardware interface.

Through repeated simulation attempts and guided stochastic initialization, players are exposed to the inherent difficulty of predicting long-term behavior in the Game of Life. Even when starting from similar conditions, patterns may rapidly stabilize, oscillate, or vanish entirely, while others persist unexpectedly. This experiential interaction provides an intuitive understanding of undecidability-like behavior and long-term unpredictability in cellular automata, without requiring formal mathematical framing. The cooking metaphor reinforces this concept: players may naturally seek to "cook a successful dish" in which patterns persist, rather than watching all ingredients disappear, motivating exploration of persistence, extinction, and emergent structure through play.

Game of Life on Other Systems and NES/Famicom

Since its publication in 1970, Conway's Game of Life has been implemented across a wide spectrum of computational platforms, each leveraging different architectural affordances [23, 24]. Early implementations on mainframes and personal computers benefited from character-based displays, frame buffers, and comparatively abundant memory, enabling large grids and continuous visualization.

Beyond traditional computers, Life has also been implemented on constrained or specialized platforms, including:

- **Scientific calculators**, such as the TI-83/84 series and HP programmable calculators, where severe memory and display limitations required extreme optimization [25].
- **Graphing terminals and oscilloscopes**, using pixel-intensity modulation to visualize cellular evolution.
- **Web-based simulations**, which remain the most widespread today and support high-resolution grids, GPU acceleration, and interactive editing [26].
- **Mobile devices**, where touchscreen interfaces enable direct manipulation and multi-scale exploration.
- **Gaming consoles**, though far more rarely, including:
  - a Sega Genesis technical demonstration using large-tile transitions [27];
  - a Game Boy homebrew implementation relying on monochrome palette cycling [28];
  - experimental shader-based implementations on PlayStation and PSP development kits.

Despite this diversity, Game of Life implementations on the NES/Famicom remain exceedingly rare. The platform's lack of a frame buffer, strict VBlank timing constraints, and tile-based rendering pipeline severely restrict feasible grid sizes and update rates [5, 8].

Several NES/Famicom Game of Life implementations have been documented in community-driven technical archives and emulator-based experiments. For example, early homebrew ROMs discussed on the NESdev forums present Life as a background-tile visualization updated at reduced speed to accommodate VBlank limitations [29]. Additional demonstrations include Lua-scripted Game of Life simulations executed within the Mesen emulator, primarily intended to explore PPU timing behavior and VRAM update strategies rather than gameplay integration [30]. These examples typically emphasize rendering feasibility and timing constraints, offering limited or no player interaction.

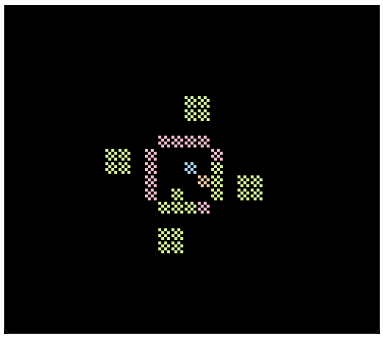

Figure 3. Example Life pattern rendered using NES tile palettes on a constrained grid [31].

Table 1. Representative Game of Life implementations on NES/Famicom.

Implementation	Grid size	Interactivity
Homebrew ROM (2004) [29]	32×30	Minimal (pause/reset)
Mesen Script Demo [30]	Configurable	Cell edit, zoom
Community ROM (2011) [29]	$\approx 16 \times 14$	No in-game input
This work	5×5	Full gameplay loop

Most known NES/Famicom Life projects originate from the homebrew community and are typically distributed as experimental ROMs or emulator scripts [29, 30]. Their primary purpose is generally to stress-test VRAM updates or demonstrate tile-based animation, rather than to serve as complete, interactive games.

Table 1 summarizes representative NES/Famicom Game of Life implementations reported in community resources. While not exhaustive, it illustrates the prevailing design space: relatively large grids, limited interaction, and an emphasis on visualization rather than gameplay integration.

In contrast to prior work, the implementation presented here integrates the cellular automaton directly into a complete gameplay loop. Player actions in the first phase of the game influence the initialization of the Life grid in the second phase, transforming the automaton from a passive visualization into a central, interactive artificial-life mechanic.

This positions the project not merely as another NES Life demonstration, but as a *console-native artificial-life game* in which emergent dynamics are structurally and mechanically central.

A-Life Veggie GoL (Game of Life) Implementation

Pedagogical Artifacts and Game Manual

To support the educational objectives of the game, a printed game manual accompanies the physical NES homebrew cartridge. Rather than serving solely as supplementary documentation, the manual is intentionally designed as a core pedagogical artifact that prepares players to engage with the Game of Life simulation. Figure 5 shows selected pages from the manual that are used as part of the learning scaffold.

The manual introduces the fundamental principles of Conway's Game of Life—including binary cell states, neighborhood-based update rules, emergence, persistence, and extinction—using diagrams and language accessible to non-technical audiences. By externalizing these concepts outside the gameplay loop, the manual enables players to form mental models before interacting with the simulation.

Two dedicated sections of the manual are particularly relevant (Figure 5). The first explains the mechanics of the "Cooking Stage,"

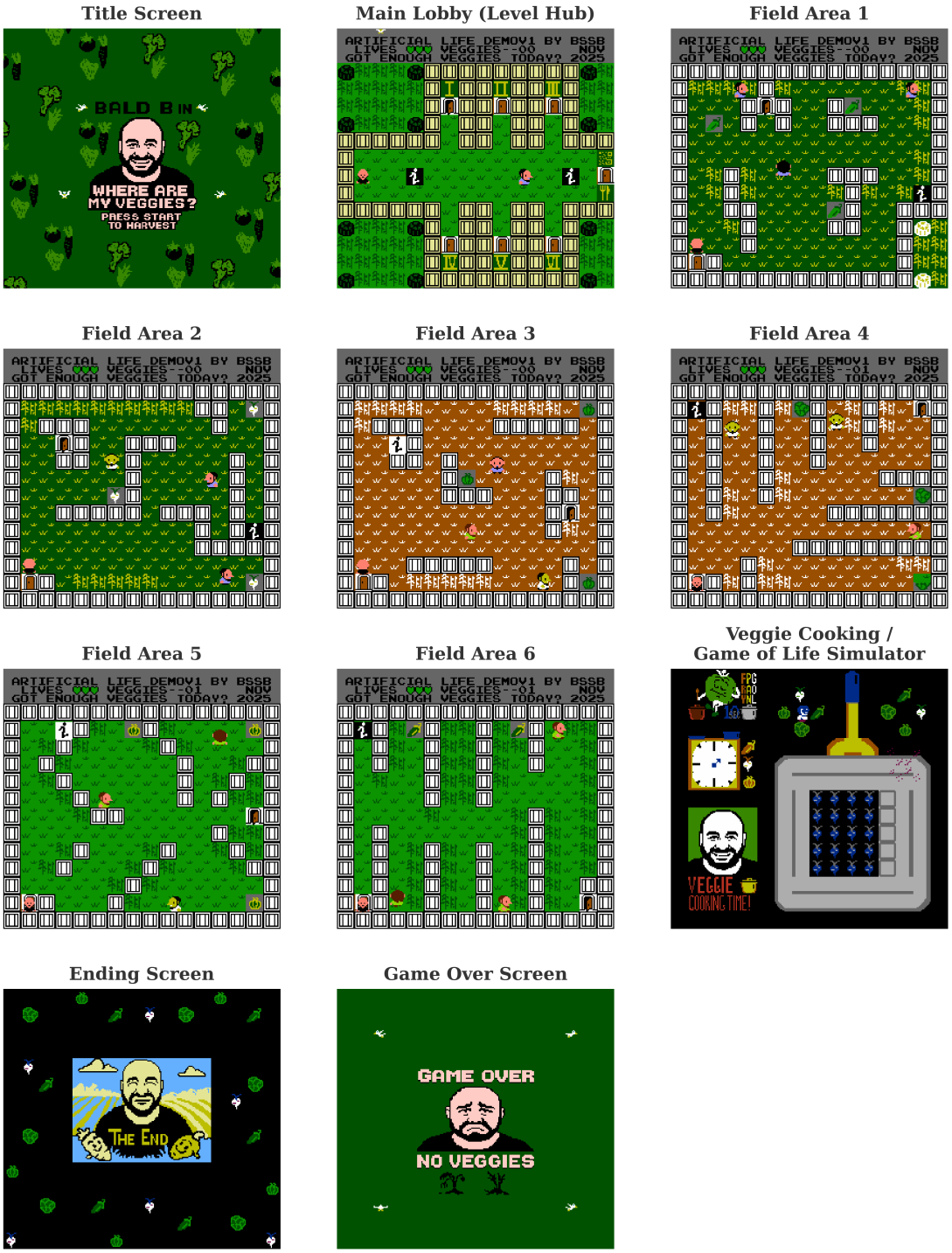

Figure 4. Game Screen Set and Menus

describing how the 5×5 grid evolves over time and how each iteration corresponds to a Life update. The second clarifies the mapping between vegetables collected during Phase 1 and active cells in Phase 2, emphasizing that vegetable diversity is a visual metaphor layered over a strictly binary cellular automaton.

The manual also explicitly documents the restart mechanism used in Phase 2 (Figure 5), noting that each restart generates a new stochastic initialization of the grid. This design encourages exploratory learning by allowing players to compare multiple emergent outcomes under similar initial conditions.

By combining interactive gameplay with a tangible instructional artifact, the system adopts a diversified pedagogical approach that integrates embodied play, visual observation, and reflective reading. This structure directly addresses concerns regarding clarity, learning objectives, and the connection between game mechanics and artificial life concepts.

Game Description

The game is built through a hybrid pipeline in which NESmaker manages the structural components—title screen, world map layout, entity placement, and collision environments—while the gameplay logic, AI routines, and Game of Life simulation are ex-

Figure 5. Selected pages from the printed game manual explaining the Game of Life mechanics, vegetable-to-cell mapping, and restart behavior as part of the pedagogical design.

ecuted using custom CA65 modules integrated into NESmaker's engine.

Entity behaviors such as enemy pursuit, projectile attacks, random-walk animals, interactive NPCs, and information kiosks are registered in NESmaker's scene editor but are executed by CA65 routines that override the default script functions. The frying-pan simulation environment in Phase 2 is defined visually in NESmaker but updated entirely through assembly routines that control the palette logic, serpentine indexing, and per-tick neighbor computation.

The game begins at the Title Screen. When the player presses Start, the system enters the main lobby, which displays six barns representing the six levels. The player must navigate each barn-themed stage, collecting vegetables while avoiding enemy attacks. Each enemy collision removes collected vegetables, and three hits trigger a Game Over transition back to the Title Screen. Figure 4 provides the overall screen set and menus for the game and Figure 6 presents the game diagram in a nutshell.

Throughout the levels, information kiosks provide interactive explanations of how Game of Life works. These optional educational prompts help players understand the simulation stage that follows.

Upon completing the six stages, or collecting a sufficient number of vegetables, the player proceeds to the second gameplay phase: the "Cooking Stage," which visually represents a frying pan and serves as the Life simulation environment.

After the Life simulation concludes or is interrupted by the player, the game proceeds to an ending screen, after which pressing Start restarts the full cycle.

Fundamental Artificial Life Principles and Gameplay Mapping

The game is explicitly designed to expose players to fundamental principles of artificial life through direct interaction and exploratory play. Rather than presenting these concepts abstractly, each principle is embodied through specific gameplay mechanics distributed across the two main phases of the game and reinforced through the in-game manual.

Emergence. Emergence is demonstrated through the Game of Life simulation in Phase 2, where complex global patterns arise from simple local interaction rules. Players observe that no centralized control exists, yet coherent structures such as oscillators, stable patterns, or extinction events emerge over time.

Self-Organization. The autonomous reconfiguration of the grid during the cooking phase illustrates self-organization. Once initialized, the system evolves independently of the player, highlighting how structure can arise from decentralized, rule-based dynamics.

Persistence and Extinction. Players are exposed to the concept of persistence by observing which configurations survive across multiple generations and which collapse rapidly. The repeated ability to restart the simulation allows comparison between persistent, oscillatory, and vanishing configurations.

Unpredictability. Despite deterministic rules, the long-term behavior of the system is difficult to anticipate from initial conditions. This unpredictability is reinforced through stochastic initialization and repeated simulation attempts, encouraging exploratory learning rather than prescriptive control.

Non-trivial Temporal Evolution. Unlike static rule demonstrations, the Life simulation evolves over extended time horizons,

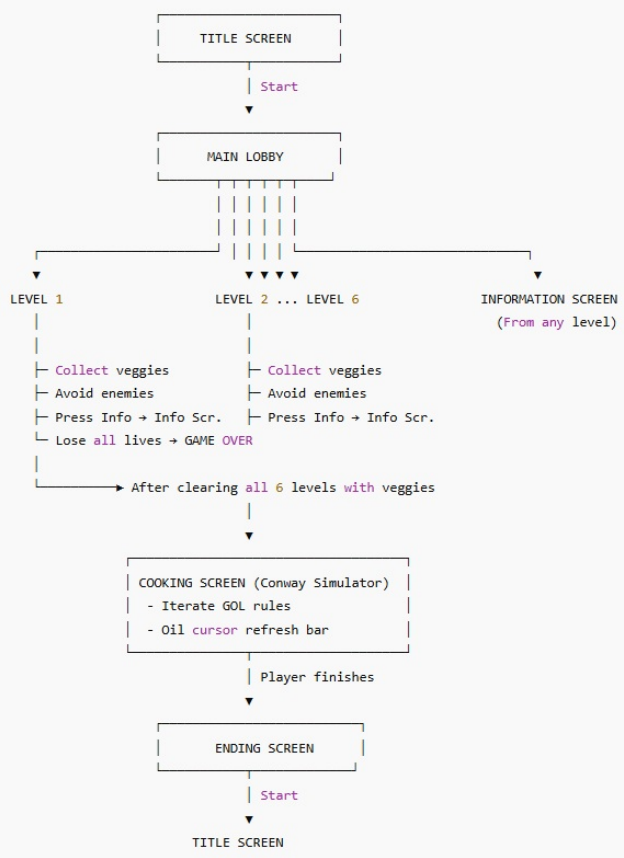

Figure 6. Overall game flow across Title Screen, Levels, and GoL Simulation.

allowing players to observe long transients, delayed stabilization, or sudden collapse. This reinforces the idea that meaningful computation and behavior can unfold over time in artificial life systems.

Phase 1 (vegetable collection) indirectly influences these principles by determining the initial population density and composition of the grid, while Phase 2 (the frying-pan simulation) exposes the emergent dynamics directly. The printed and digital game manual complements gameplay by explicitly explaining these principles in age-appropriate language, reinforcing the learning objectives introduced through interaction.

Relation to Gamification and Serious Games Literature

The pedagogical design of the proposed game aligns with established principles in the literature on gamification, serious games, and exploratory learning. Serious games emphasize learning through structured interaction rather than explicit instruction, allowing players to construct understanding through experimentation and feedback [32, 33].

Gamification research highlights the importance of motivation, challenge, and meaningful feedback in sustaining engagement and supporting learning outcomes [34, 35]. In the present work, risk–reward dynamics during vegetable collection, the possibility of failure and recovery, and the delayed reveal of emergent behavior in the Game of Life simulation serve as intrinsic motivators rather than external scoring mechanisms.

Exploratory and discovery-based learning frameworks further support the use of open-ended systems where learners are encouraged to test hypotheses, observe outcomes, and refine intuition over repeated interactions [36, 37]. The guided randomness employed in the initialization of the Life grid intentionally avoids prescriptive configuration, instead promoting experimentation and comparison across multiple simulation runs.

By embedding these principles within a constrained retro-

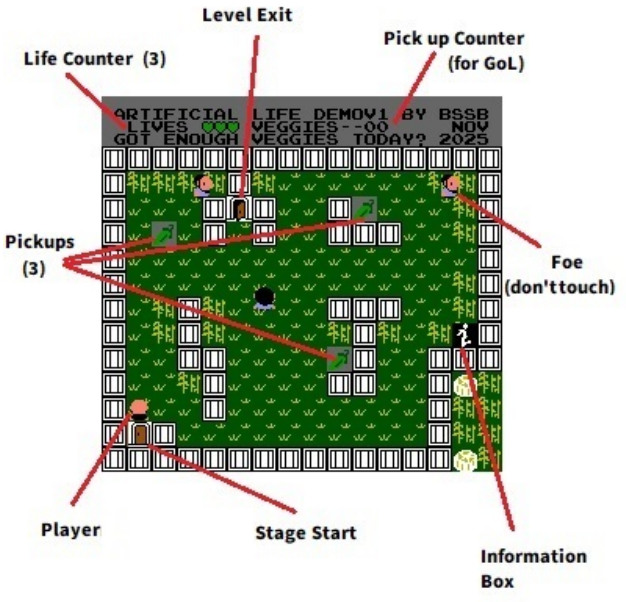

Figure 7. HUD and gameplay elements of the vegetable-collection phase.

hardware environment, the game functions as a serious game that prioritizes engagement, curiosity, and conceptual understanding over optimization or performance metrics. This positioning situates the work within existing pedagogical frameworks while highlighting the unique affordances of retro platforms for educational game design.

Phase 1: Veggie Collection

During the vegetable-collection stage, the player character traverses scrolling environments containing:

- autonomous enemies following predefined movement patterns
- collectible vegetable pickups
- informational kiosks accessible via a button press
- environmental obstacles with collision detection

The player may explore freely, collecting as many or as few vegetables as desired. However, each collision with an enemy reduces the vegetable inventory, and a total of three collisions result in a full game reset. This creates a trade-off between exploration, risk management, and preparation for the Game of Life simulation in the second phase. Figure 7 shows the typical screen HUD and flow in a maze-like genre game.

Phase 2: Veggie GoL Simulator

The second phase displays a stylized frying pan containing a 5×5 Life grid. Each collected vegetable acts as a living cell in the initial configuration. Additional hidden buffer cells support the movement of the "hot oil" cursor, which sweeps across the grid every five seconds to update all states. The frying-pan simulation screen is structured visually using NESmaker's tile and palette tools, while all cellular automaton updates are executed by CA65. NESmaker defines the static layout, but CA65 controls palette changes, serpentine traversal, neighbor evaluation, and timing. This hybrid design enables a visually intuitive environment combined with mathematically correct Game of Life evolution running in real time. The player may allow the simulation to run indefinitely, restart it, or trigger the ending screen.

Each restart of the game triggers a new stochastic initializa-

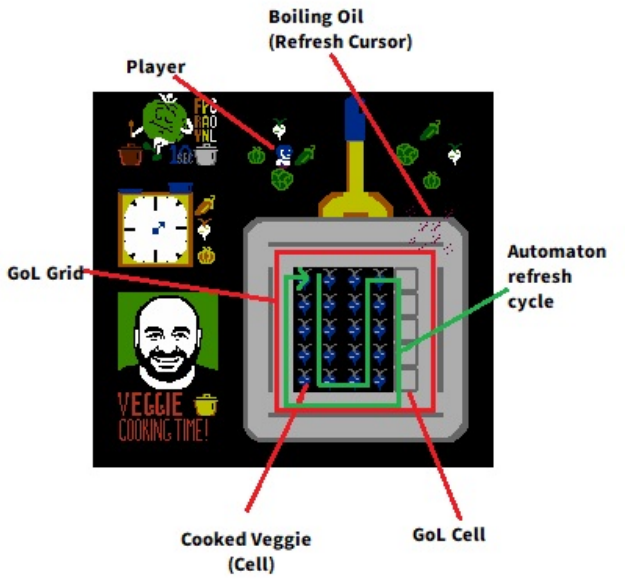

Figure 8. The 5 × 5 frying-pan Game of Life simulation.

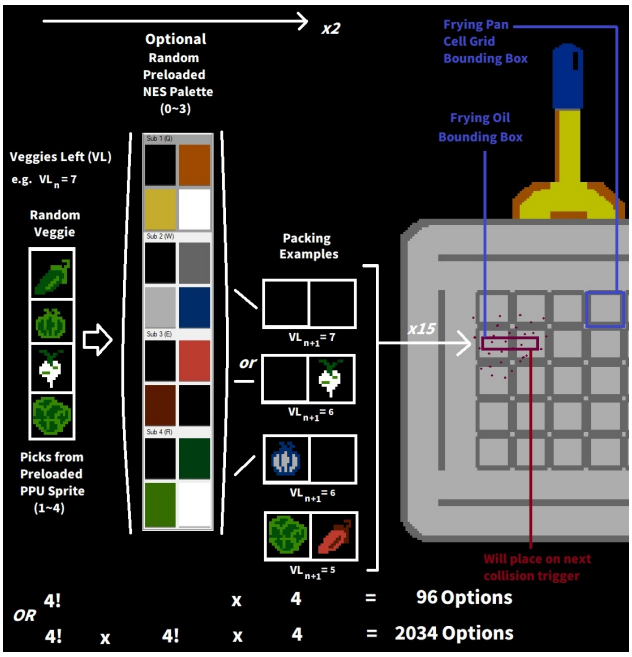

Figure 9. Sorting and placement pipeline for vegetable-based cell initialization.

tion of the 5 × 5 grid, generating a different distribution of active (vegetable) cells based on the same collection constraints. This allows players to explore, compare, and reflect on multiple emergent outcomes arising from identical or similar gameplay conditions.

The simplicity of the grid supports concise visualization of emergent patterns, while palette changes provide rapid feedback under strict PPU constraints. Figure 8 presents the GoL screen and flow of simulation. Although multiple vegetable sprites are used in the visual representation of the Game of Life grid, the underlying cellular automaton remains strictly binary. Each cell exists in one of two logical states: active (state = 1) or inactive (state = 0). Active cells are rendered using a vegetable sprite, while inactive cells are displayed as empty grid positions.

Sorting Out the Veggies

The transition from Phase 1 to the Game of Life simulation requires translating the player's vegetable collection into a valid initial configuration for the 5 × 5 frying-pan grid. This step is performed through a lightweight initialization algorithm designed specifically for the constraints of the NES Picture Processing Unit (PPU). The algorithm produces a structured sequence of cell states based on: (1) vegetable quantity, (2) vegetable type, and (3) palette assignment (Figure 9).

The vegetable-collection phase is constructed using NESmaker's level editor, which defines scrolling tile maps, collision zones, and entity spawn points. However, enemy AI behavior—including pursuit heuristics, random-walk generators, projectile timers, and autonomous animal movement—is implemented through CA65 extensions injected into NESmaker's event handlers. This allows the enemies to exhibit complex behavior while maintaining cycle accuracy.

The NES provides no hardware random-number generator, but its cycling registers and frame-based timing cues can be repurposed to generate pseudorandom values. These values seed the selection of:

- one of four available vegetable sprite archetypes
- one of four background palette entries
- one of two possible binary cell states (occupied or empty)
- a 1x2 vegetable arrangement used as the atomic placement unit

This 1x2 block structure is not arbitrary; it reflects the underlying memory and sprite limitations of the PPU. Using a compact two-tile pattern reduces the total number of attribute updates and permits local reuse of CHR patterns without incurring excessive tile-swapping overhead.

Frying the Veggies

At the outset, the frying pan grid is empty. After a five-second initialization, vegetables are distributed randomly based on the results from Phase 1. If no vegetables were collected, the simulation remains static as the cursor traverses empty cells.

For populated grids, each full sweep of the oil cursor performs a complete Life iteration. Cells outside the 5x5 boundary are discarded to maintain spatial coherence. The player may continue iterations indefinitely, restart the simulation, or exit to the ending sequence.

1. Serpentine Mapping Strategy

To efficiently populate the 5×5 grid, we employ a *serpentine mapping* traversal. In this pattern, the algorithm fills the first row from up to down, the second from down to up, and so on. This alternating pattern continues until all 25 cells are assigned.

This approach offers several advantages in the constrained NES environment:

- **Minimal cursor movement:** Serpentine traversal reduces the Manhattan distance between successive cell updates. On hardware without a frame buffer, every unnecessary VRAM update increases the risk of overrun during VBlank.
- **Efficient PPU attribute usage:** Because each serpentine row shares adjacency with the next in vertical order, palette attribute table writes can be batched and require fewer updates per scanline.
- **Reduced sprite burden:** The frying-pan grid relies on background tiles rather than sprites; however, sprite-based indicators (such as the hot-oil cursor) must move between cells. The serpentine route minimizes sprite displacement, reducing OAM update load.
- **Deterministic timing:** Knowing the exact spatial order of updates allows pre-counted cycles in CA65, ensuring the system never exceeds VBlank constraints.

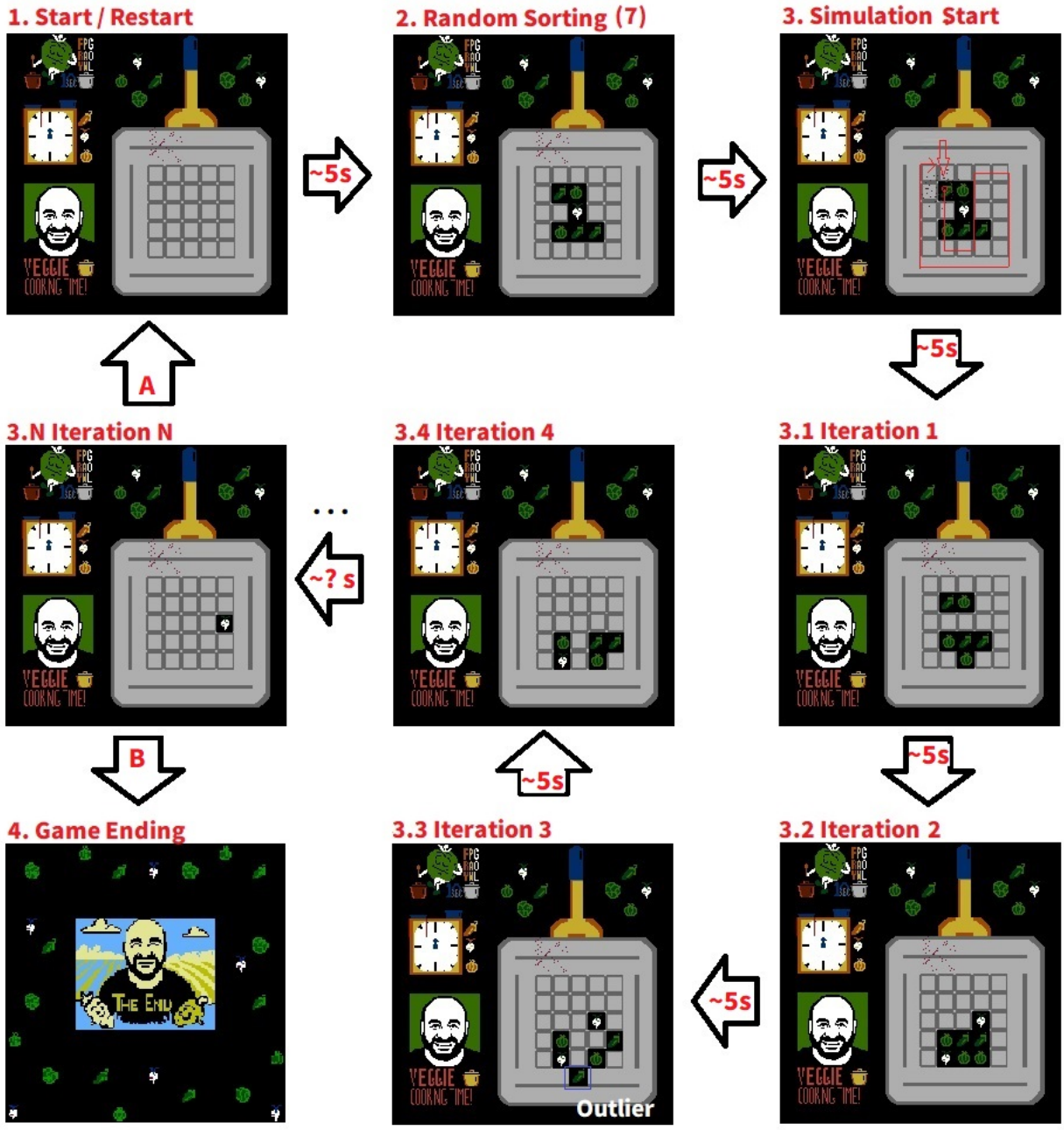

Figure 10. Life-iteration cycles driven by the oil cursor refresher.

In combination, these factors make serpentine mapping the optimal traversal method for grid initialization on NES hardware.

2. Interaction With NES PPU Constraints

The NES PPU imposes strict limitations that fundamentally shape the grid-update strategy. The system relies on:

- **Name Table memory (960 bytes)** for tile indices,
- **Attribute Tables (64 bytes)** for palette assignment per 16x16 block,
- **CHR ROM/CHR RAM** for tile patterns,
- **Object Attribute Memory (OAM)** for sprite handling.

Since VRAM is writable only during VBlank (about 20 scanlines), updating the frying pan requires a highly optimized sequence:

i. Precompute the serpentine indices during NMI setup.
ii. Write two consecutive tile indices (for the 1x2 vegetable block) in a single VRAM burst.
iii. Update attribute bytes only when the palette assignment differs from the previous cell.
iv. Move the oil cursor sprite minimally between adjacent cells.

This design ensures that no frame exceeds the available VBlank window, preserving gameplay smoothness and eliminating tearing or missed updates.

Figure 11. Player feedback and educational testing with children.

3. Integration With Gameplay Semantics

The serpentine mapping does more than improve performance: it also mirrors the conceptual metaphor of cooking. The oil cursor sweeps the frying pan in a visually coherent, back-and-forth pattern, reinforcing the analogy of a sizzling surface. Thus, the player experiences both computational logic and playful visual feedback, bridging A-Life theory and game design.

Overall, the sorting and mapping subsystem is not only a technical requirement for operating under 8-bit constraints, but also a deliberate aesthetic and pedagogical design choice that prepares the grid for meaningful Life evolution in Phase 2. Figure 10 summarizes graphically the algorithm implementations.

4. Unbounded Iteration and Player-Controlled Convergence

A distinctive feature of this implementation is that the Life simulation is not restricted to a predetermined number of generations. Instead, the system is engineered to operate indefinitely, enabling the player to observe emergent structures across arbitrarily long temporal horizons. This aligns with established treatments of cellular automata as open-ended dynamical systems [38, 39].

Technically, unbounded evolution is feasible because each generation update resides within the strict VBlank-time computational budget dictated by the NES PPU. No cumulative frame latency is introduced, and the constrained 5×5 topology ensures that the state space remains tractable—avoiding the exponential explosion associated with larger grids [40]. Thus, real-time responsiveness is preserved across long iteration sequences.

In effect, the simulation functions not only as a gameplay mechanic but also as an open-ended artificial-life laboratory, allowing players to inspect long-term emergent behaviors under strict hardware constraints while maintaining complete control over temporal exploration.

Results

The evaluation presented in this work is qualitative and exploratory in nature. The goal of the study is not to provide a formal assessment of learning outcomes, but rather to demonstrate the feasibility and educational potential of embedding an artificial-life simulation into a playable NES game under authentic hardware constraints.

Observations were gathered through informal play sessions with children and adult participants, focusing on engagement, curiosity, and conceptual understanding rather than quantitative performance metrics. Participants were encouraged to interact freely with both gameplay phases and to restart the simulation multiple times in order to observe differences in emergent behavior.

Based on these observations, players demonstrated increased curiosity about how the Game of Life operates(Figure 11), particularly regarding persistence, extinction, and unpredictability of patterns. These findings are reported as qualitative indicators of pedagogical potential rather than validated educational outcomes.

Further Work

Future work includes optimizing the Game of Life update kernel, expanding grid sizes within mapper constraints, and developing additional educational games for subjects such as orthodontics, biological growth models, or next-generation strategy games.

Moreover, we aim to integrate mixed-reality interfaces and EEG-based control systems with retro hardware, combining traditional console aesthetics with modern interactive modalities.

In addition to the extensions proposed above, future work may explore the intersection between artificial-life simulations on constrained hardware and concepts from renewable energy systems. Cellular automata have been used extensively to model diffusion dynamics, distributed control, and emergent optimization processes relevant to photovoltaic arrays, smart-grid behavior, and energy harvesting strategies [41, 42]. Because the present system demonstrates how rule-based emergent behavior can be executed efficiently under extreme computational limitations, an intriguing direction involves adapting similar lightweight simulation kernels to educational tools that explain renewable energy distribution, decentralized energy management, or the behavior of microgrids.

Furthermore, the hybrid NESmaker–CA65 architecture could be repurposed to create simplified, visually intuitive simulations that introduce younger learners to renewable energy concepts through retro-style interactive experiences. These might include grid-balancing games, energy-flow visualizations implemented as cellular automata, or low-power embedded systems demonstrating sustainability principles. Such extensions would align artificial-life research with pedagogical efforts in environmental awareness and renewable energy education, thereby broadening the societal impact of constrained-platform simulations.

A formal quantitative evaluation of learning outcomes—such as controlled studies, pre/post testing, or comparative analysis with alternative teaching tools—is beyond the scope of the present work and is identified as a clear direction for future research.

Discussion

The hybrid NESmaker–CA65 methodology used in this project demonstrates that high-level tooling and low-level hardware-oriented programming can coexist to produce emergent artificial-life behaviors on highly constrained 8-bit systems. NESmaker enables rapid prototyping, visual iteration, and structured scene organization, while CA65 provides the precision necessary to implement real-time cellular automata within strict VBlank and memory limitations.

This dual-tool approach reveals an important insight: educational game systems on retro hardware benefit from tooling that supports both conceptual accessibility and technical depth. NESmaker lowers the barrier for constructing visually consistent game worlds, whereas assembly-level logic encourages analytical reasoning about memory, timing, algorithmic efficiency, and deterministic state evolution.

The results of this project support the idea that hybrid workflows may play a significant role in computational education, retrocomputing research, and interactive A-Life demonstrations.

Potential Implications

Beyond its immediate technical contributions, the present work suggests several broader implications for research and education.

First, implementing cellular automata on retro hardware provides a novel approach to studying minimal computational substrates for artificial life. By reducing the system to strict hardware constraints, researchers can better isolate which properties of emergence persist under limited memory, fixed-tile graphics, and deterministic timing environments.

Second, this work may influence pedagogical practice. Retro consoles offer a unique entry point for teaching algorithmic thinking: students engage with a system that is both approachable and historically meaningful. The tactile and visual character of NES hardware can support interdisciplinary lessons combining computing, mathematics, and digital arts.

Third, the methodology demonstrated here—palette-based encoding of state changes, serpentine grid traversal, and PPU-timed refresh—may be applicable to other educational or simulation-based games on constrained platforms. While this is not intended as proof of broader importance, it highlights that constrained systems can serve as experimental testbeds for computational creativity.

Finally, in a personal opinion of the author, this type of project encourages a re-evaluation of the role of retro gaming hardware as a vehicle for contemporary research. Rather than obsolete systems, consoles like the Famicom can become modern laboratories for A-Life, discrete mathematics, and HCI (Human–Computer Interaction) studies when augmented with thoughtful game design.

Methods

This project employs a hybrid development methodology that integrates high-level tooling from NESmaker with low-level, cycle-accurate programming in CA65 assembly. This structure supports rapid asset iteration while enabling the implementation of real-time cellular automata, which are not feasible using NESmaker's scripting layer alone.

Development Workflow

NESmaker [13] serves as the organizational framework for screen construction, entity placement, collision maps, and general scene transitions. Its data structures provide a structured and modifiable environment for orchestrating the six overworld stages, the vegetable-collection mechanics, and the transition to the Game of Life simulation.

However, the kernel of the system—including the neighbor-counting routines, generation updates, serpentine grid traversal, palette manipulation, and VRAM write scheduling—is implemented exclusively in CA65 assembly. These custom routines are injected through NESmaker's project-level hooks, replacing or extending the tool's default virtual machine.

Low-Level Implementation

Because the NES lacks a frame buffer and only allows background modifications during VBlank, a fully hand-optimized approach was required. The methods include:

- **Cycle-counted VRAM Update Routines:** All Game of Life cell transitions are updated within the narrow VBlank period by batching tile writes and attribute updates through CA65-coded NMI (Non-Maskable Interrupt) handlers.
- **Serpentine Grid Traversal:** The Game of Life grid uses serpentine mapping to minimize cursor movement, reduce VRAM access frequency, and maintain deterministic timing at 60 Hz.
- **Palette-Encoded State Transitions:** To avoid tile rewriting, life/death states are encoded through background palette changes, reducing the VRAM footprint to a level compatible with NES timing constraints.
- **CHR Memory Optimization:** Vegetable sprites and frying-pan backgrounds are stored in CHR banks arranged to minimize bank-switching overhead.
- **Hybrid Data Encoding:** NESmaker's scene definitions (tile maps, object placement, NPC hooks) are combined with CA65-driven automata state variables stored in custom RAM layouts.

Testing and Debugging

Debugging was conducted primarily using Mesen and FCEUX [18], leveraging:

- the PPU viewer for verifying VRAM writes,
- OAM (Object Attribute Memory) inspection for cursor and enemy-sprite behavior,
- trace logging for cycle-count verification,
- breakpoints on NMI and VRAM write cycles.

This mixed-method approach ensures both structural clarity—thanks to NESmaker's scene tools—and hardware accuracy through assembly-level control.

Comparison and Evaluation

Two implementation strategies were compared:

- a sprite-based Life updater, and
- a palette-based background updater.

The sprite-based method exceeded PPU bandwidth limits and produced flicker, whereas the palette-buffered method remained stable under NTSC timing. Only the palette method was retained.

Reproducibility

To ensure reproducibility, all CA65 routines are provided in structured modules,and NESmaker's projectfile contains all screen definitions exactly as used. The workflow canbe reproduced by loading the NESmaker project and compiling with the inserted CA65 routines, requiring only the CC65 toolchain, NESmaker environment, and standard NESemulation software. Reproducibility is a central objective of this work. To ensure that the methods, algorithms, and results can be independently replicated, the following measures were taken:

Code Transparency.. All core update routines, including the Game of Life kernel, palette-based state transitions, grid traversal logic, and PPU synchronization code, were implemented in deterministic 6502 assembly using the `ca65` assembler. Each routine is cycle-counted to ensure correct behavior under NTSC timing. Modular code organization and commenting facilitate line-by-line inspection and reproduction.

Environment Specification.. The development and testing pipeline relies on a documented toolchain consisting of:

- `ca65/ld65` (CC65 toolchain),
- NES Screen Tool for CHR/tile generation,
- FamiTracker for audio assets,
- Mesen and FCEUX for cycle-exact emulation and PPU debugging,
- Original Famicom hardware with a flash cartridge for final verification.

This environment can be reproduced identically across systems.

Deterministic Behavior.. All aspects of the Game of Life simulation are deterministic, including:

- the seed-based vegetable sorting mechanism,
- the serpentine grid traversal,
- the oil cursor refresh timing,
- the Life rule application sequence.

This ensures that given identical player inputs, hardware states, and ROM images, the simulation will evolve identically on all systems.

Data and ROM Availability.. The final ROM file, source code, and accompanying documentation—including memory maps, tile tables, and grid evolution snapshots—will be deposited in a publicly accessible repository with a DOI in compliance with IMI and GigaScience policies. This repository will contain:

- full assembly source code
- build scripts
- emulator configuration files
- reproducible test cases and output logs

Overall, the combination of transparent code, deterministic logic and publicly accessible resources supports full reproducibility of the work and encourages reuse in both educational and research contexts.

Data, Code, and Material Availability

To ensure full reproducibility, all materials associated with this work are publicly available. The complete source code of the game—including all CA65 assembly routines, NESmaker project files, build scripts, and graphical assets—is hosted in a public GitHub repository:

https://anonymous.4open.science/r/wamg-5913

In addition, a fully playable version of the game is available through an online emulator, allowing immediate execution of the ROM without requiring local toolchain installation. Supplementary materials, including the printed game manual, box artwork, and educational documentation, are publicly accessible at:

https://theretroverse.com/product/bald-b-in-where-are-my-veggies/

These resources collectively provide full transparency of the implementation and enable independent verification, reproduction, and extension of the system.

Declarations

List of Abbreviations

A-Life Artificial Life. Field studying emergent behavior in artificial systems.
CA Cellular Automaton / Cellular Automata. Discrete dynamical systems governed by local rules.
CA65 6502 assembler from the CC65 toolchain, used for low-level NES/Famicom programming.
CHR Character ROM/RAM. Memory containing tile patterns for background and sprite graphics.
CPU Central Processing Unit of the NES (Ricoh 2A03/2A07).
HCI Human–Computer Interaction. Study of interaction between humans and computing systems.
NES Nintendo Entertainment System (North American version of the Famicom).
NESmaker Modern development tool for constructing NES homebrew games using a data-driven pipeline.
NMI Non-Maskable Interrupt. Vertical blank interrupt used for PPU updates on the NES.
OAM Object Attribute Memory. Sprite metadata storage for the NES PPU.
PPU Picture Processing Unit. NES video processor responsible for tile and sprite rendering.
PRG Program ROM. Memory that stores the game code and logic.
RAM Random Access Memory. NES working memory (2 KB total).
ROM Read-Only Memory. Cartridge-based memory for code, graphics, and game assets.
Serpentine Mapping Alternating left-to-right / right-to-left or Alternating down-to-up / up-to-down traversal pattern used to minimize VRAM updates and cursor movement.
VRAM Video RAM of the PPU used for name tables, attribute tables, and pattern table access.
VTBL Vector Table or jump table used in assembly-level dispatch.
VBlank Vertical Blank Interval. Period during which the PPU allows safe background updates.
GoL Conway's Game of Life. A binary-state cellular automaton defined by John Conway.
UI User Interface.

Consent for Publication

This manuscript does not contain any individual person's data, images, or identifiable information in any form. Therefore, consent for publication is not applicable. All figures included in the manuscript are original creations produced by the author for the purposes of scientific description and demonstration.

Competing Interests

The author declares that there are no financial or non-financial competing interests that could be perceived as influencing the work reported in this manuscript. No commercial entity, organization, or individual has provided funding, materials, or incentives that could affect the objectivity or integrity of the research. The development of the game system and all associated analyses were conducted independently by the author.

Acknowledgements

Special acknowledgement is extended to Shiru, whose contributions to the NES homebrew community—particularly through his FamiTracker compositions and software tools—have supported the audio component of this project.

Authors' Contributions

The authors were solely responsible for all components of this work. This includes the conceptualization of the study, the complete design and development of the game, and the full implementation of all algorithms and system behaviors in 6502 assembly. The contributions are as follows:

- **Conceptualization**
- **Formal Analysis**
- **Investigation**
- **Writing – Original Draft**
- **Visualization**

All design decisions, optimizations, graphical assets, gameplay logic, artificial intelligence routines, PPU update mechanisms, and

Game of Life implementation strategies were created exclusively by the authors without external coding assistance. The project, from initial concept to final assembly-verified ROM, represents the contribution in its entirety.

References

1. Gardner M. Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life". Scientific American 1970;223(4):120–123.
2. Wolfram S. Statistical Mechanics of Cellular Automata. Reviews of Modern Physics 1983;55(3):601–644.
3. Langton CG. Studying Artificial Life with Cellular Automata. In: Proceedings of the Fifth Annual Conference of the Center for Nonlinear Studies; 1986. .
4. Bedau MA. Artificial Life: Organization, Adaptation and Complexity from the Bottom Up. Trends in Cognitive Sciences 2003;7(11):505–512.
5. NESDev Wiki Contributors, PPU Rendering and Hardware Behavior; 2024. Accessed: 2025-02-28. https://www.nesdev.org/wiki/PPU.
6. VBlank and VRAM Access;. Accessed: 2025-12-30. https://www.nesdev.org/wiki/The_frame_and_NMIs.
7. Anthropy A, Clark N. A Game Design Vocabulary. Addison-Wesley; 2014.
8. Altice N. I AM ERROR: The Nintendo Family Computer / Entertainment System Platform. MIT Press; 2015.
9. Montfort N. The Future of Text. MIT Media Lab; 2020. Includes chapters on retrocomputing and constrained platforms.
10. Salen K, Zimmerman E. Rules of Play: Game Design Fundamentals. Cambridge, MA: MIT Press; 2004.
11. Yamada H, Retro Game Systems as Educational Tools for Computational Thinking; 2021. Online: https://ieeexplore.ieee.org. IEEE International Symposium on STEM Education.
12. NESDev Wiki Contributors, NES Architecture Documentation; 2024. Accessed: 2025-02-27. https://www.nesdev.org/wiki/NES_Architecture.
13. Heroes TNB, NESmaker: Build NES Games Without Coding; 2018. Accessed: 2025-02-27. https://www.thenew8bitheroes.com.
14. NESDev Wiki Contributors, PPU Frame Timing and VBlank Restrictions; 2024. Accessed: 2025-02-28. https://www.nesdev.org/wiki/PPU_frame_timing.
15. Spencer U, cc65: A C compiler for 6502-based systems; 2023. https://cc65.github.io.
16. Shiru, NES Screen Tool; 2011. Accessed: 2025-02-27. https://shiru.untergrund.net/software.shtml.
17. HertzDevil, jsr, FamiTracker: NES Music Tracker; 2010. Accessed: 2025-02-27. http://famitracker.com.
18. Sour, Mesen: Cycle-accurate NES and Famicom Emulator; 2024. Accessed: 2025-02-27. https://www.mesen.ca.
19. Games Y, GameMaker Studio; 2024. Accessed: 2025-02-27. https://gamemaker.io.
20. Berlekamp E, Conway J, Guy R. Winning Ways for Your Mathematical Plays. Academic Press; 1982.
21. Rendell P. Turing Machine Universality of the Game of Life. Collision-Based Computing 2016;p. 513–539.
22. Gardner M. Mathematical Games: On the construction of minimal Life patterns and the "glider gun". Scientific American 1971;224(2):112–117.
23. Bell G, Gray J, Szalay A. Petascale Computational Systems: Life After Moore's Law. Communications of the ACM 2008;51(11):58–66.
24. Levy S. Artificial Life: The Quest for a New Creation. Pantheon Books; 1992.
25. Programmers TC, Conway's Game of Life for TI-83/84 Calculators; 2004. Accessed: 2025-02-28. http://ticalc.org.
26. Contributors V, Web-based Conway's Game of Life Implementations; 2021. Accessed: 2025-02-28. https://copy.sh/life/.
27. Developers SH, Game of Life Tech Demo for Sega Genesis; 1995. Accessed: 2025-02-28. https://segadev.org.
28. Community GBH, Conway's Game of Life for Nintendo Game Boy; 2001. Accessed: 2025-02-28. https://gbdev.io.
29. NESDev Community, Homebrew Game of Life Experiments on NES/Famicom; 2004. Accessed: 2025-02-28. https://forums.nesdev.org.
30. Community M, Mesen Lua Game of Life Scripts; 2022. Accessed: 2025-02-28. https://www.mesen.ca/docs/.
31. Cason K, John Conway's Game of Life - NES; 2025. https://kennycason.com/posts/2025-03-22-game-of-life-nes.html, accessed: 2025-11-27. Online image.
32. Michael DR, Chen SL. Serious Games: Games That Educate, Train, and Inform. Boston, MA: Thomson Course Technology; 2006.
33. Abdelhamid M, Ben Abbes A, Gargouri F. Serious games for education: A systematic literature review. Education and Information Technologies 2019;24(4):2051–2076.
34. Deterding S, Dixon D, Khaled R, Nacke L. From Game Design Elements to Gamefulness: Defining "Gamification". In: Proceedings of the 15th International Academic MindTrek Conference ACM; 2011. p. 9–15.
35. Hamari J, Koivisto J, Sarsa H. Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. Proceedings of the 47th Hawaii International Conference on System Sciences 2014;p. 3025–3034.
36. Bruner JS. The Act of Discovery. Harvard Educational Review 1961;31(1):21–32.
37. Kirschner PA, Sweller J, Clark RE. Why Minimal Guidance During Instruction Does Not Work. Educational Psychologist 2006;41(2):75–86.
38. Wolfram S. A New Kind of Science. Champaign, IL: Wolfram Media; 2002.
39. Ilachinski A. Cellular Automata: A Discrete Universe. World Scientific; 2001.
40. Gardner M. Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life". Scientific American 1970;223(4):120–123.
41. Alvarado I, Peralta D. Cellular Automata Models for Energy Distribution in Renewable Microgrids. Renewable Energy 2020;158:1205–1217.
42. Zhang W, Li C. A Cellular Automata Framework for Optimizing Distributed Energy Systems. Energy and Buildings 2019;199:350–361.

Bruno Senzio-Savino Barzellato is the founder and principal developer at SenzioTek S. de R.L., where he leads the creation of innovative and sui generis embedded systems, retro-computing projects, and interactive game-based learning tools.

With a background in mechatronics engineering (B.S. Summa Cum Laude, National Autonomous University of Mexico) and graduate studies in Japan, his interdisciplinary work spans video game system development, brain–computer interfaces, and educational technology.

Faramarz Alsharif received his Bachelor's degree in Electrical and Electronics Engineering from the Faculty of Engineering, University of the Ryukyus, in 2009. He obtained his Master's degree from the Graduate School of Science and Engineering at the same university, and completed his Ph.D. in Interdisciplinary Intelligent Systems Engineering in 2014. From 2014 to 2016, he worked as a postdoctoral research fellow at University of the Ryukyus. During 2014–2018, he was also employed as a Development Engineer at Okinawa Sabani Technology Co. Ltd.. In 2018, he joined Kitami Institute of Technology as an Assistant Professor in the Faculty of Engineering, Department of Electrical and Electronic Engineering. He is currently an Associate Professor at Yamagata University, Graduate School of Science and Engineering, Faculty of Engineering, Department of Informatics and Electronics, Electrical and Electronics Communication Course. His research interests include power converters for renewable energy systems, control systems engineering, signal processing, distributed wireless measurement, analysis of small-scale wind turbine dynamics under strong wind conditions, integration of control and communication engineering, and stability analysis of remote control systems. He is a member of IEEE.