International Journal Software Engineering and Computer Science (IJSECS)
6 (1), 2026, 268-278
Published Online April 2026 in IJSECS (http://www.journal.lembagakita.org/index.php/ijsecs)
P-ISSN: 2776-4869, E-ISSN: 2776-3242. DOI: https://doi.org/10.35870/ijsecs.v6i1.7158.

RESEARCH ARTICLE Open Access

Application of the Linear Congruential Generator Method in a Number-Guessing Game

Efani Desi [1*]

[1*] Department of Information System, Faculty of Enginnering and Computer Science, Potensi Utama University, Medan City, North Sumatra Province, Indonesia.

*Corresponding author: efanidesi88@gmail.com.

Received: April 8, 2026; Accepted: April 15, 2026; Published: April 20, 2026.

Abstract: Number guessing games occupy a deceptively simple position in the landscape of interactive software — easy to play, yet dependent on a mechanism that most players never think about: the quality of the number being generated. If that number is predictable, the game collapses. If it is genuinely unpredictable, the experience holds. This study applies the Linear Congruential Generator (LCG) method as the core algorithm for random number generation in a number-guessing game operating within the range of 1 to 100. LCG was selected not because it is the most statistically sophisticated option available, but because its computational simplicity and deterministic structure make it well-suited to lightweight game applications where speed and transparency of implementation matter more than cryptographic strength. Four parameters govern the algorithm's behavior — modulus, multiplier, increment, and seed — and each was configured deliberately to produce a distribution that remains acceptably uniform across repeated sessions. The seed value, generated dynamically at runtime, ensures that no two sessions begin from the same starting point, which is the primary mechanism for maintaining unpredictability from the player's perspective. Test results indicate that the application performs adequately: number distribution is reasonably balanced, the interface responds correctly to valid and invalid inputs, and the overall gameplay experience is fair. A more critical reading of the results, however, reveals that LCG carries inherent trade-offs — its period is finite, and under certain parameter configurations, detectable patterns can emerge in the generated sequence. For a single-player guessing game, these limitations are largely inconsequential. For applications requiring stronger statistical guarantees, they would not be. The application was built using Python with a graphical interface constructed through the tkinter library, making it accessible to users without any technical background. This study is intended as a practical reference for developers building number-based games and as a modest contribution to the applied literature on pseudo-random number generation in educational software.

Keywords: Random Number Generator; LCG; Number Guessing; Game; Python.

1. Introduction

Number guessing games represent one of the most accessible forms of interactive entertainment, requiring no specialized hardware, no prior technical knowledge, and no steep learning curve — yet they remain genuinely engaging for players across age groups (Riis, 2026). Their appeal is not purely recreational. For children in particular, these games serve a functional educational purpose: they encourage numerical reasoning, reinforce basic arithmetic intuition, and have been associated with a measurable increase in mathematical interest when used in structured learning contexts (Nathan & Mamza, 2020). The game's premise is simple enough — a player attempts to identify a number that the system has selected at random within a defined range, typically from 1 to 100 — but the simplicity of the premise should not obscure the technical requirement sitting beneath it (Perbawa & Diana, 2022). The system must select that number in a way that is genuinely unpredictable. If the selection mechanism is weak or patterned, the game loses its challenge, and with it, much of its value as both entertainment and educational tool (Safitri, 2021).

This is where random number generation becomes a design concern rather than a background detail. Across computing applications — games, simulations, statistical modeling, and cryptographic systems — the quality of random number generation directly determines the reliability and fairness of the system's output (Khan *et al.*, 2024; Hameedi *et al.*, 2022). In game development specifically, a random number generator that produces detectable patterns or repeating sequences will eventually be noticed by players, consciously or not, and the experience degrades accordingly (Avdović, 2023). The generator does not need to be cryptographically secure for a guessing game — but it does need to be unpredictable enough that no player can reasonably anticipate the output from one session to the next.

Among the available methods for pseudo-random number generation, the Linear Congruential Generator (LCG) occupies a well-established position in the literature. It is one of the oldest PRNG algorithms still in active use, and its continued relevance is not accidental. The LCG produces a sequence of numbers through a deterministic recurrence relation involving four parameters — modulus, multiplier, increment, and seed — and the simplicity of that structure is precisely what makes it attractive for lightweight applications (Faure *et al.*, 2022; Pandit, 2019). Implementation requires minimal computational resources, the algorithm is transparent and auditable, and the behavior of the output is well-understood. That said, LCG is not without limitations. Its period — the length of the sequence before it begins to repeat — is bounded by the modulus value, and under poorly chosen parameters, the generated sequence can exhibit structural patterns that undermine its apparent randomness (Fanani, 2021). These trade-offs are known and documented, and they inform the parameter selection decisions described later in this study.

The present study applies the LCG method to develop a functional number-guessing game application covering the range of 1 to 100. The work proceeds through several stages: system design using the Waterfall development model, selection and justification of LCG parameters, implementation in Python with a graphical user interface built using the tkinter library, and a functional evaluation of the completed application. The choice of Python as the implementation language reflects both its readability and its suitability for rapid application development — a consideration that matters when the goal is to produce a working, testable product rather than a performance-optimized system. The graphical interface was included specifically to make the application accessible to users without a technical background, addressing a gap identified in prior work where similar applications remained command-line dependent (Wardana & Sabrina, 2025).

Beyond the technical contribution, this study carries an educational motivation. Number-guessing games of this kind can function as entry points into mathematical thinking for children — low-stakes, engaging, and repeatable. The application developed here is intended to be usable in informal educational settings, and the research itself is offered as a reference for developers working on number-based games or exploring pseudo-random number generation in similarly constrained contexts. The broader aim is modest but clear: to show that a well-configured LCG, implemented carefully and wrapped in an accessible interface, is sufficient for producing a fair and engaging guessing game — and to document that process in enough detail that others can replicate or extend it.

2. Literature Review

2.1 State-of-the-Art Research

Safitri *et al.* (2021) examined the application of the LCG method within a dance-themed puzzle game designed as an educational tool for children. The game used randomization to vary puzzle configurations, with the specific aim of helping young players learn about and engage with Indonesian traditional dance. The LCG method proved effective in managing the game's randomization logic, producing sufficiently varied outputs to sustain gameplay across multiple sessions. This study is one of the earlier examples in the Indonesian literature of LCG being applied in an educational game context, and it established a useful baseline for understanding

how the algorithm behaves under game-specific constraints (Safitri, 2021). Wardana and Sabrina (2025) approached the problem from a different angle, conducting a case study on LCG implementation in a number-guessing game built using C++. Their findings suggest that even a structurally simple guessing application can serve as a meaningful vehicle for mathematical education, particularly for younger users. The LCG method, in their assessment, is a reasonable and practical choice for this type of application. The study is notable for its focus on the guessing game format specifically — a narrower and more directly relevant scope than puzzle-based implementations — though the application itself remained command-line dependent, with no graphical interface provided for end users (Wardana & Sabrina, 2025).

2.2 Comparison with Previous Studies

These two studies support the broader argument that LCG is a suitable algorithm for educational game development. Both demonstrate working implementations, and both report positive outcomes in terms of randomization quality and educational utility. The gaps, however, are identifiable. Safitri *et al.* (2021) worked within a puzzle game context, which differs from a number-guessing game in its interaction model, its randomization requirements, and its educational framing. Wardana and Sabrina (2025) did target the guessing game format directly, but their application ran through a command prompt — a delivery method that limits accessibility for non-technical users and makes the application impractical for use with children in informal educational settings. Neither study produced a graphical, standalone application built in a modern, readable programming language. The present study addresses both of these gaps simultaneously.

2.3 Positioning of This Research

The rationale for this study follows directly from the limitations identified above. The first prior study implemented LCG in an application but not in a number-guessing game context. The second targeted the correct game format but delivered it through a prompt-based interface that most end users would find inaccessible. This study sits at the intersection of both: a number-guessing game application, implemented in Python, with a full graphical user interface that requires no command-line interaction. Python was selected over C++ not merely for convenience, but because its readability and standard library support — particularly the tkinter module for GUI development — make it a more appropriate choice for educational software intended to be extended, modified, or studied by other developers.

2.4 Review of Technologies, Frameworks, and Algorithms

2.4.1 Games

Sumiah and Hakim (2021) describe games as a form of entertainment that actively fosters creativity and strategic thinking in players across all age groups. From a computer science perspective, game applications are among the more practically visible products of the discipline — systems that require the developer to think carefully about user interaction, feedback loops, and outcome unpredictability. The core cognitive demand of a game, as Sumiah and Hakim (2021) frame it, is the development of the brain's capacity to formulate strategies quickly and accurately under conditions of uncertainty. That framing applies directly to number-guessing games, where the uncertainty is generated algorithmically and the player's task is to resolve it through reasoning rather than reflex.

2.4.2 Educational Games

Educational games occupy a specific and well-documented niche in learning theory. They are neither purely recreational nor purely instructional — they operate in the space between the two, using the engagement mechanics of play to deliver cognitive and developmental benefits that more formal learning environments sometimes struggle to produce (Perbawa & Diana, 2022). For children specifically, educational games have been associated with improvements in logical reasoning, numerical fluency, and sustained attention. Beyond cognition, they have been linked to physical development, personality formation, and the strengthening of caregiver-child relationships. The number-guessing game developed in this study fits within this category: it is simple enough to be immediately playable, but structured in a way that consistently exercises numerical estimation and probabilistic thinking.

2.4.3 Linear Congruential Generator

The LCG algorithm has accumulated a substantial body of definitional and analytical literature, and the characterizations offered across that literature are largely consistent. Gunadi (2023) describes LCG as an approach for generating numerical sequences through configurable mathematical operations, noting that the numbers it produces — while deterministic in origin — are classified as pseudo-random because they are statistically indistinguishable from truly random sequences under standard tests. Ramadan (2024) frames LCG more precisely as a recursive function that uses a linear congruential system to generate numbers that appear random without being genuinely so — a distinction that matters in cryptographic contexts but is largely

inconsequential in game applications. Bouras (2024) situates LCG within the broader PRNG category, emphasizing that it operates on the principle of linear recurrence and that its behavior is entirely determined by its four parameters: modulus, multiplier, increment, and seed. Fahrezi *et al.* (2023) add an important practical observation: the LCG sequence is periodic, meaning it will eventually return to its starting value after a finite number of generations. The length of that period is bounded by the modulus, and under well-chosen parameters, it can be made long enough to be functionally irrelevant for most game applications. The algorithm is also fast — computationally inexpensive enough to run without measurable latency even on modest hardware. Purwono *et al.* (2021) extend the discussion into security contexts, noting that LCG can reduce the risk of random key reuse when the multiplier, increment, and modulus are fixed appropriately, though they also acknowledge that LCG alone is insufficient for applications requiring strong cryptographic protection. For the purposes of this study — a single-player number-guessing game — the algorithm's speed, simplicity, and configurability are the properties that matter most.

2.4.4 Python

Python is a high-level, interpreted programming language with a well-established reputation for readability and developer efficiency (Crespo *et al.*, 2024). Unlike lower-level languages such as C++ or Java, Python allows complex logic to be expressed in significantly fewer lines of code, which reduces development time and lowers the barrier for developers who are new to the language (Soto-León *et al.*, 2023; Wardana & Sabrina, 2025). It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles within a single environment — a flexibility that makes it suitable for a wide range of application types (Bouras, 2024). For this study, Python's standard library was particularly relevant: the tkinter module provided the tools needed to build a graphical user interface without requiring any third-party dependencies, and the random module supplied the seed generation mechanism used to initialize the LCG algorithm at the start of each game session.

3. Methodology

Application development in this study follows the Waterfall model, which provides a structured, sequential framework for managing each stage of the development process — from initial design through to testing and deployment. The Waterfall model was selected because the scope of the application is well-defined and the requirements do not change between stages, making a linear development approach both practical and efficient. The stages of the development process are illustrated in Figure 1.

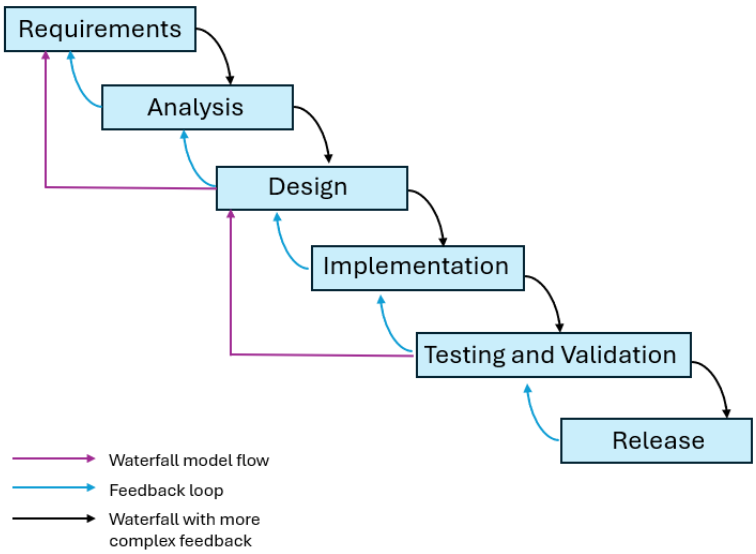

Figure 1. Waterfall Diagram

Following the methodology diagram, the next stage involves system design. A flowchart was constructed to map the operational logic of the application — that is, the sequence of steps the system executes from the moment the user interacts with the interface to the moment a result is returned. The flowchart is shown in Figure 2.

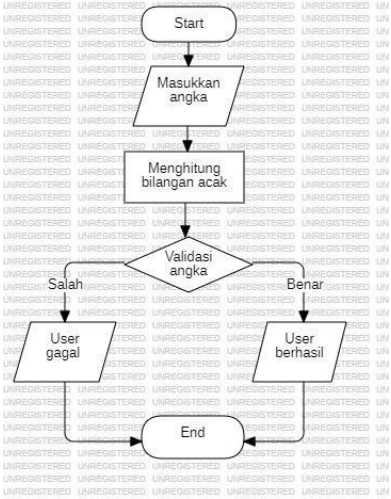

Figure 2. Flowchart Diagram

The flowchart traces the following sequence. The process begins at the "Start" node, after which the user enters a number into the input field. The application then initiates the random number generation process by computing the LCG formula using pre-configured parameters. The generated number is subsequently compared against the user's input through a conditional validation step. If the two values match, the system returns a correct-guess response. If they differ, the system returns an incorrect-guess response and the game session ends. This logic is straightforward, but its correctness depends entirely on the quality of the number generated by the LCG algorithm — which is why parameter selection, discussed later in this section, is treated as a deliberate design decision rather than an arbitrary one.

3.1 Library Setup and Interface Design

The application was written in Python using two standard libraries: random and tkinter. The random library is used exclusively to generate the seed value (X_0) at the start of each game session, ensuring that the LCG sequence begins from a different starting point each time the game is played. The tkinter library handles the graphical user interface. Python offers several alternative UI libraries — including PyAutoGUI and pyo — but tkinter was selected for its inclusion in the Python standard library, its stability, and its relatively low implementation overhead. For an application of this scope, it is the most practical choice.

```
import tkinter as tk
from tkinter import messagebox
import random
```

The interface was built using three core tkinter widgets. The Label widget displays static and dynamic text on the interface, including the application title and the result message that updates after each guess. The Entry widget provides the numeric input field through which the user submits their guess. The Button widget creates the interactive buttons — one to submit a guess for validation, and one to restart the game. The interface is initialized using the Tk() function, assigned to the root variable, and kept running through the mainloop() method, which holds the interface open and responsive until the user closes the application window.

3.2 LCG Algorithm Implementation

The Linear Congruential Generator operates through the following recurrence relation:

$$Xn = (aXn{-}1 + \mathrm{b})\ mod\ \mathrm{m} \qquad (1)$$

Where:
X_n = the n-th random number in the sequence
X_{n-1} = the previous random number
a = multiplier
b = increment
m = modulus

The seed value X_0does not carry a fixed value in the implementation. It functions as the generator key — the starting point from which the entire sequence is derived. Because the seed is generated dynamically at runtime using random.randint(), each game session begins from a different position in the sequence, which is the

primary mechanism for maintaining output unpredictability from the player's perspective. The remaining parameters — multiplier, increment, and modulus — are fixed values declared outside the LCG function, then passed as arguments when the function is called. To illustrate the algorithm's behavior before implementation, a manual calculation was performed using the following parameters: $a = 1$, $b = 19$, $m = 30$, and seed $= 1$, with a sequence length of ten. The results are as follows:

$X_1 = (1 \times 1 + 19) \text{mod } 30 = 20$
$X_2 = (1 \times 20 + 19) \text{mod } 30 = 9$
$X_3 = (1 \times 9 + 19) \text{mod } 30 = 28$
$X_4 = (1 \times 28 + 19) \text{mod } 30 = 17$
$X_5 = (1 \times 17 + 19) \text{mod } 30 = 6$
$X_6 = (1 \times 6 + 19) \text{mod } 30 = 25$
$X_7 = (1 \times 25 + 19) \text{mod } 30 = 14$
$X_8 = (1 \times 14 + 19) \text{mod } 30 = 3$
$X_9 = (1 \times 3 + 19) \text{mod } 30 = 22$
$X_{10} = (1 \times 22 + 19) \text{mod } 30 = 11$

The resulting sequence — [20, 9, 28, 17, 6, 25, 14, 3, 22, 11] — is summarized in Table 1. The values are distributed across the range without immediate repetition, which confirms that the chosen parameters produce an acceptable sequence for demonstration purposes.

Table 1. LCG Result Table

X_n	Result
X_1	20
X_2	9
X_3	28
X_4	17
X_5	6
X_6	25
X_7	14
X_8	3
X_9	22
X_{10}	11

3.3 Code Implementation

The LCG function is implemented as follows. The seed value x is accepted as a parameter, the recurrence relation is applied, and the result is mapped to the range 1–100 using a modulo operation before being returned.

```
def lcg(modulus, multiplier, increment, seed):
    x = seed
    x = (multiplier * x + increment) % modulus
    return x % 100 + 1
```

The start_game function initializes each game session. It generates a new seed value using random.randint(), calls the LCG function with the configured parameters, and stores the result as target_number — the value the player must identify. The input field is cleared and the result label is reset at the start of each session.

```
def start_game():
    global target_number
    seed = random.randint(0, modulus - 1)
    target_number = lcg(modulus, multiplier, increment, seed)
    entry_guess.delete(0, tk.END)
    label_result.config(text="Enter your guess!")
```

The validation function, check_guess, handles user input processing. It reads the value entered in the Entry widget, checks that it falls within the valid range of 1 to 100, and compares it against target_number. A matching value triggers a success response; a non-matching value ends the session with an incorrect-guess notification. Input outside the valid range raises a ValueError and prompts an error dialog without advancing the game state.

```
def check_guess():
    try:
```

```
        user_guess = int(entry_guess.get())
        if user_guess < 1 or user_guess > 100:
            raise ValueError("Guess must be between 1 and 100.")
    except ValueError as e:
        messagebox.showerror("Input Error", str(e))
        return
    if user_guess == target_number:
        label_result.config(text="Correct! Your guess is right.", fg="green")
        messagebox.showinfo("Success", "You guessed the number!")
    else:
        label_result.config(text="Wrong guess. Game over.", fg="red")
        messagebox.showwarning("Failed", "Incorrect guess. Game over.")
        print(target_number)
```

This application does not use iterative number generation. Since the game requires a single target value per session, only X_1— the first element of the LCG sequence — is computed and used as the target. This design choice keeps the implementation simple and the session logic clean.

3.4 Testing and Compilation

Once the code was complete, the application was compiled and tested for syntax errors and runtime exceptions. All interface components — the input field, the check button, and the restart button — were verified to function as intended under both valid and invalid input conditions. Following successful testing, the application was packaged as a standalone .exe file using a Python-to-executable conversion tool, allowing end users to run the application directly on Windows without requiring a Python interpreter or any additional setup.

4. Result and Discussion

4.1 Results

The development process followed all stages defined in the Waterfall model — from system design and interface layout through to LCG implementation, functional testing, and final packaging. The application was built using Python 3.10, a version selected for its updated standard library, improved syntax clarity, and broader community support relative to earlier releases. Once development was complete, the application was converted from a .py source file into a standalone .exe executable using a Python-to-executable packaging tool. This conversion means that end users can run the application directly on any compatible Windows system without installing Python, opening a command prompt, or interacting with a compiler in any form — a deliberate design decision aimed at making the application accessible to non-technical users, including children. The completed application presents a graphical user interface built with the tkinter library. The interface is intentionally minimal, containing three primary elements: a numeric input field (Entry widget), two action buttons, and a dynamic result label. The input field accepts the user's guess. The first button — labeled "Check Guess" — submits that guess for validation against the LCG-generated target number. The second button — labeled "Play Again" — resets the session, generates a new target number, and clears the input field, allowing the user to start a new round without restarting the application. The result label sits below the buttons and updates dynamically to reflect the outcome of each guess attempt. The application interface is shown in Figure 3.

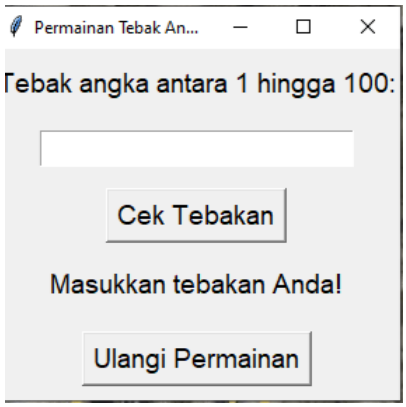

Figure 3. App Interface

To begin a game session, the user types a number between 1 and 100 into the input field and clicks the "Check Guess" button. The application enforces the valid input range through a conditional check in the check_guess function. If the user enters a value outside the range of 1 to 100 — whether too large, too small, or non-numeric — the application immediately returns an error dialog notifying the user that the input must fall within

the specified range. The game state does not advance until a valid input is submitted. This behavior is shown in Figure 4.

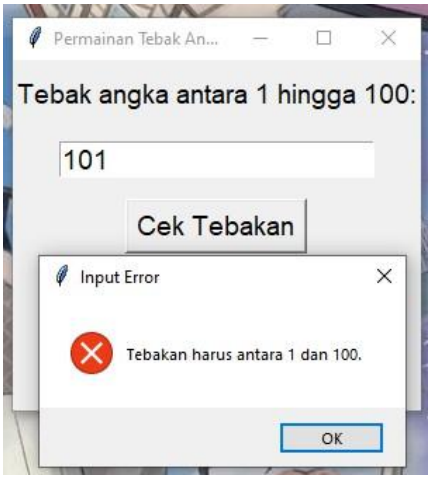

Figure 4. Error Message From Application

Once a valid number is submitted, the system executes the LCG algorithm using the pre-configured parameters and the dynamically generated seed value. The output of the LCG function — specifically X_1, the first element of the sequence — is stored as the target_number for that session. The validation function then compares the user's input against this value using a conditional statement. If the two values match, the result label updates to display a success message in green text, and a confirmation dialog appears notifying the user that the guess was correct. This outcome is shown in Figure 5.

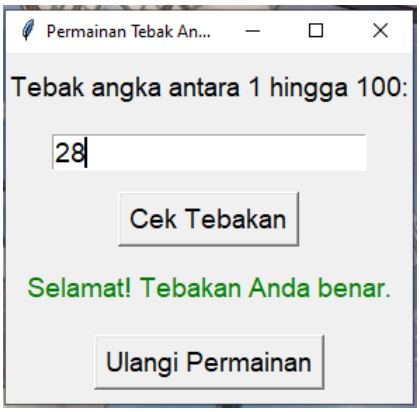

Figure 5. Message If The User Is Correct

If the submitted value does not match the target number, the result label updates to display an incorrect-guess message in red text, and a warning dialog informs the user that the game session has ended. The target number is also printed to the console at this point, which serves as a debugging aid during testing and can be removed in a production release. This outcome is shown in Figure 6.

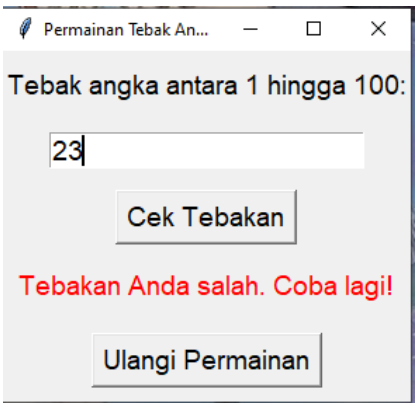

Figure 6. Message If The User Is Incorrect

Across all tested input scenarios — valid correct guesses, valid incorrect guesses, out-of-range inputs, and non-numeric inputs — the application responded as expected. No runtime errors were encountered during functional testing, and the LCG algorithm produced a different target number in each session due to the randomized seed, confirming that the dynamic seed generation mechanism functions correctly.

4.2 Discussion

The results confirm that the application performs as intended. The LCG algorithm generates a target number per session, the interface accepts and validates user input correctly, and the comparison logic produces the appropriate response in every tested case. What the results also show — and what deserves direct acknowledgment — is that the application's correctness depends not on the LCG algorithm alone, but on the combination of dynamic seed generation and appropriate parameter configuration. A fixed seed would produce the same target number every session, rendering the game trivially solvable after the first round. The use of random.randint() to generate the seed at runtime is therefore not a minor implementation detail; it is the mechanism that makes the game functionally unpredictable from the player's perspective. Compared to the two prior studies reviewed in Section 2, this study addresses identifiable gaps in both. Safitri *et al.* (2021) applied LCG in a puzzle game context — a different interaction model with different randomization demands. Wardana and Sabrina (2025) targeted the number-guessing format directly but delivered the application through a command-line interface, which limits accessibility for non-technical users and makes the application impractical for use with children in informal educational settings. The present application resolves both issues: it targets the guessing game format and delivers it through a graphical interface that requires no technical background to operate.

The practical applicability of this application extends beyond the immediate research context. The primary target users are children and general members of the public who would benefit from a low-stakes, engaging tool for practicing numerical reasoning. The application requires no internet connection, no account registration, and no installation procedure beyond running the .exe file — a deployment profile that makes it suitable for use in schools, community centers, or home environments with minimal infrastructure. In terms of computational requirements, the application is extremely lightweight: the LCG algorithm performs a single arithmetic operation per session, and the tkinter interface imposes negligible processing overhead. There are no scalability concerns at the individual user level, and the application can be distributed freely at no cost.

That said, the study has clear limitations that should be stated plainly. The application is desktop-only, built for Windows, and has not been tested on macOS, Linux, or any mobile platform. The visual design is functional but basic — there is no main menu, no difficulty progression, no scoring system, no background audio, and no visual customization. These are not incidental omissions; they reflect the bounded scope of the research. The LCG algorithm itself also carries a known limitation: its period is finite, and under certain parameter configurations, the generated sequence can exhibit detectable patterns. For a single-player guessing game operating within a range of 1 to 100, this limitation is inconsequential in practice. For a more complex application — one involving multiple simultaneous users, longer sessions, or any security-sensitive context — a more statistically robust PRNG would be the appropriate choice. The broader educational aim of this study also deserves mention. Number-guessing games of this kind can function as entry points into mathematical thinking for children — low-stakes, immediately playable, and repeatable without becoming tedious. The application developed here is simple enough to be used without instruction, but structured in a way that consistently exercises numerical estimation and basic probabilistic reasoning. Whether that educational value is realized in practice depends on how the application is deployed and who uses it — questions that fall outside the scope of this study but that future research could address through user testing with target populations.

5. Conclusion and Recommendations

This study applied the Linear Congruential Generator method to develop a functional number-guessing game application operating within the range of 1 to 100. The LCG algorithm was configured with fixed parameters — modulus, multiplier, and increment — combined with a dynamically generated seed value at the start of each session, ensuring that the target number changes between rounds without requiring the player to do anything beyond clicking the restart button. Test results indicate that the algorithm produces a reasonably uniform distribution of output values across repeated sessions, which is the property most directly relevant to fairness in a guessing game context. No session produced a target number outside the valid range, and the application responded correctly to all tested input conditions, including valid guesses, out-of-range inputs, and non-numeric entries.

The LCG method, taken on its own terms, is a reasonable fit for this application. It is computationally inexpensive, straightforward to implement, and well-documented in the literature — qualities that matter when

the goal is to produce a working, maintainable application rather than a performance-optimized system. Its known limitations — a finite period and the potential for detectable sequence patterns under certain parameter configurations — do not pose meaningful problems in a single-player guessing game where the session is short and the stakes are low. That assessment changes, however, as application complexity increases. For systems involving multiple simultaneous users, longer number sequences, or any requirement for statistical unpredictability beyond what LCG can reliably provide, more robust alternatives — such as the Mersenne Twister or cryptographically secure PRNGs — would be the more defensible choice.

Several directions for future research follow naturally from the limitations of this study. The application is currently desktop-only and has not been tested on mobile platforms; extending the implementation to Android or iOS would substantially broaden its reach, particularly given that the primary target users — children and general members of the public — increasingly access software through mobile devices rather than desktop computers. The interface, while functional, is visually basic: a main menu, difficulty levels, a scoring system, background audio, and visual customization would all improve the player experience without requiring changes to the underlying algorithm. Future work might also conduct formal user testing with target populations to evaluate whether the application produces the educational outcomes — improved numerical reasoning, increased interest in mathematics — that motivated its development. A comparative study evaluating LCG against alternative PRNG algorithms within the same game context would provide a more rigorous basis for algorithm selection recommendations than the current study can offer. These are not minor additions — they represent the logical next stage of development for anyone who wants to take this work further.

References

Avdović, A. (2023). Extension of Linear Congruential Generator. *Scientific Publications of the State University of Novi Pazar Series A: Applied Mathematics, Informatics and Mechanics, 15*(2), 87–95. https://doi.org/10.46793/spsunp2302.087a

Bouras, A. (2024). *Integrating randomness in large language models: A Linear Congruential Generator approach for generating clinically relevant content* (pp. 1–13). arXiv. https://arxiv.org/abs/2407.03582

Bulolo, S. (2019). Implementasi metode Linear Congruent Method (LCM) pada simulasi ujian akhir sekolah menengah kejuruan Lolomatua. *Prosiding Seminar Nasional Teknologi Informasi, 2*(November), 60–64.

Fahrezi, G. R., Sakti, D. V. S. Y., & Muhardi, H. (2023). Implementasi Pseudo Random Number Generator (PRNG) dengan algoritma Linear Congruential Generator (LCG) pada permainan mengetik bertema Candi Prambanan. *Jurnal Edukasi dan Penelitian Informatika, 9*(2), 299. https://doi.org/10.26418/jp.v9i2.64713

Fanani, A. (2021). Pengacakan soal pada sistem Computer Based Test (CBT) dengan metode Linear Congruential Generator (LCG) di SMA Negeri Jogoroto. *Jurnal Ilmiah Teknologi Informasi dan Sains, 1*(2001), 50–56.

Faure, E., Fedorov, E., Myronets, I., & Sysoienko, S. (2022). Method for generating pseudorandom sequence of permutations based on Linear Congruential Generator. *CEUR Workshop Proceedings, 3137*, 175–185. https://doi.org/10.32782/cmis/3137-15

Gunadi, H. (2023). Aplikasi ujian online untuk SMA PKP JIS dengan metode Linear Congruential Generator (LCG) berbasis web. *Jurnal KLIK, 4*(2), 815–828. https://doi.org/10.30865/klik.v4i2.1175

Hameedi, B. A., Hattab, A. A., & Laftah, M. M. (2022). A pseudo-random number generator based on new hybrid LFSR and LCG algorithm. *Iraqi Journal of Science, 63*(5), 2230–2242. https://doi.org/10.24996/ijs.2022.63.5.35

Khan, S. S., Palmer, B., Edelmaier, C., & Aktulga, H. M. (2024). OpenRAND: A performance portable, reproducible random number generation library for parallel computations. *SoftwareX, 27*, 101773. https://doi.org/10.1016/j.softx.2024.101773

Luis Crespo, J., González-Villa, J., Gutiérrez, J., & Valle, A. (2024). Assessing the quality of random number generators through neural networks. *Machine Learning: Science and Technology, 5*(2). https://doi.org/10.1088/2632-2153/ad56fb

Nathan, N., & Mamza, S. A. (2020). *Mathematical Journal of Interdisciplinary Sciences, 8*(2), 61–68.

Pandit, K. (2019). Linear congruential generator and caesar cipher. *International Journal of Engineering and Advanced Technology, 9*(1), 2248–2250. https://doi.org/10.35940/ijeat.A9731.109119

Perbawa, K. A., & Diana, D. (2022). Application of Linear Congruential Generator (LCG) algorithm in Android based mathematics education game. *Jurnal Komputer, Informasi dan Teknologi, 2*(1), 47–56. https://doi.org/10.53697/jkomitek.v2i1.599

Ramadan, A. N. (2024). *Simulasi penyingkat URL menggunakan Linear Congruential Generator (LCG)*. Institut Teknologi Bandung.

Riis, S. (2026). *Term coding: An entropic framework for extremal combinatorics and the guessing-number sandwich theorem*. arXiv. https://arxiv.org/abs/2601.16614

Safitri, D. N. (2021). Implementasi metode Linear Congruential Generator pada game puzzle kesenian tari. *JATISI (Jurnal Teknik Informatika dan Sistem Informasi), 8*(1), 129–140. https://doi.org/10.35957/jatisi.v8i1.632

Soto-León, V., Arco, J. M., Martínez-Pérez, M. Á., Oliviero, A., & Aguilar, J. (2023). Effects of transcranial static magnetic field stimulation over the left dorsolateral prefrontal cortex on random number generation. *Clinical Neurophysiology, 149*, 18–24. https://doi.org/10.1016/j.clinph.2023.02.163

Sumiah, A., & Hakim, R. R. (2021). Implementasi metode Linear Congruential Generator pada game puzzle berbasis Android. *JEJARING (Jurnal Teknologi dan Manajemen Informatika), 8*(1), 129–140. https://doi.org/10.35957/jatisi.v8i1.632

Triwibowo, D. N., Purwono, Ashari, I. A., Sandi, A. S., & Rahman, Y. F. (2021). Enkripsi pesan menggunakan algoritma Linear Congruential Generator (LCG) dan konversi kode Morse. *Buletin Ilmiah Sarjana Teknik Elektro, 3*(3), 194–201. https://doi.org/10.12928/biste.v3i3.5546

Wardana, S. C. P., & Sabrina, S. S. J. N. (2025). Game tebak angka multipemain dengan penilaian skor otomatis menggunakan bahasa C++. *Jurnal Komputasi dan Pengembangan Aplikasi, 1*(3), 22–29. https://journals.arces.org/jukompak/.