



# Pseudo-Random Number Generation in an Agent-based Gamified Cellular Automata Environment

**Ozan Can Acar**

(Software Engineering Dept., Atılım Univ., 06830 İncek, Gölbaşı, Ankara, Türkiye  
 <https://orcid.org/0000-0003-0781-0260>, acar.ozan@student.atilim.edu.tr)

**Hürevren Kılıç**

(Computer Engineering Dept., Atılım Univ., 06830 İncek, Gölbaşı, Ankara, Türkiye  
 <https://orcid.org/0000-0002-9058-0365>, hurevren.kilic@atilim.edu.tr)

**Abstract:** High-quality random number generation is a need for many engineering application areas grounded on different theoretical bases, including probability theory & statistics, game theory, information theory, etc. Our solution to the problem is a Pseudo-Random Number Generation (PRNG) framework constituted by a Two-Dimensional Linear Cellular Automata (2D LCA) model and the Monte Carlo Tree Search method within an agent-based gamified environment. The framework employs 2D LCA as the foundation, utilizing the temporal evolution histories of cells to examine its impact on random sequence generation enabled through an intelligent agent-based gamification approach. Experimental results and evaluations showed that under the proposed framework, both agent-to-agent and agent-to-environment goal-driven game competitions facilitate high-quality random sequence generation while passing the NIST Statistical Test Suite tests, with success rates surpassing most of the existing PRNGs in the literature.

**Keywords:** Pseudo-Random Number Generation, 2D Linear Cellular Automata, Agent-based Gamification, Monte Carlo Tree Search, NIST Statistical Test Suite

**Categories:** I.2.1, I.6.8, K.8.0, G.3, F.1.1

**DOI:** 10.3897/jucs.156553

## 1 Introduction

Random numbers are abstract concepts that we encounter in many areas of life and virtual environments, including modeling and simulation, engineering applications, cryptography, daily software systems applications, statistical research, hash functions, numerical analysis, artificial intelligence, and randomized algorithms of computer games. These fields require randomly generated numbers [Bhattacharjee and Das, 2022]. In particular, probability and statistics are two fundamental disciplines built over the abstract concept of random variables [L'Ecuyer, 2012]. They are widely utilized in many diverse fields, including probability theory, game theory, information theory, statistics, Monte Carlo simulations, cryptography, pattern recognition, and computer games. They are essential to virtual decision-making and game development and must be fulfilled on demand. For example, unpredictability, variability, and replayability are important gaming requirements that are satisfied basically via random number generators. Such requirements are important for any software development effort to mimic lifelike situations coupled with user/player interactions. Particularly in video

games, we need to introduce elements of randomness into various game aspects, including item drops, enemy encounters, critical hits, loot tables, and other gameplay mechanics [Quora, 2025]. However, we know that too much randomness may cause the player not to feel in control of the game. Given the randomness, the player must at least be able to estimate a percentage chance of success.

Although it is hard to describe the randomness, it can be characterized simply as the “lack of recognizable patterns.” The systems or algorithms that can generate a sequence of numbers with random properties are called random number generators (RNGs). In gaming lingo, RNG is used to generate events that seemingly have no pattern for the gamer. The RNG can be implemented through physical hardware devices or software programs. Accordingly, there are two types of random number generators: true random number generators (TRNG) and pseudorandom number generators (PRNG). TRNGs utilize a non-deterministic entropy source and a processing function to generate random sequences [Bassham III et al., 2010]. The entropy function generally consists of random events like thermal noise, user mouse movements or keystrokes, noise in an electric circuit, etc. Conversely, PRNGs utilize a deterministic algorithm and a set of inputs to generate random sequences. Inputs to the PRNG systems are called seed values. Since the algorithm is deterministic in the PRNG case, the same seed values produce the same results. A random sequence should be unpredictable in nature, so a PRNG, by default, should be coupled with a TRNG [Bassham III et al., 2010]. Despite this shortcoming, the fact that the same seed gives the same results is quite helpful in some areas, such as gaming. Whether TRNG or PRNG is used to generate a random number, a number in the generated random sequence should not provide any information about the following number. However, generating random numbers through TRNGs with true physical sources coupled with computers is time-consuming and expensive [Gaeini et al., 2015]. PRNGs, faster generators, are attracting more attention in gaming and security literature.

We know that RNGs are essential and inevitable components of computer games. Nevertheless, from another perspective, take the question: “How can we use agent-based gamification to achieve high-quality RNG design?”. The question can be studied under a general title: Gamification for Games. To the best of the authors’ knowledge, gamification as a tool in random number generation is very rare in both RNG and gaming literature. For example, the deterministic chaos game, a method of generating a fractal, was used by [Ayubi et al., 2020] for PRNG design and implementation for cryptographic purposes. In [Mariot, 2023], the author investigated the dynamics of an introduced game where the two orthogonal Latin squares are defined by Cellular Automata (CA). The game's dynamics are related to the problem of enumerating maximal cycles generated by orthogonal cellular automata for better PRNG design. However, in both studies, gamification is not realized via an autonomous intelligent entity (i.e., agent). However, it is used as a method/tool to realize/explain the high-quality PRNG generation procedure they introduced. In the previous literature, a notable gap in PRNG construction perspective is the treatment of the process as a classical monolithic single-body system design, rather than a gamified agent/environment. As part of our new original perspective and introduced model, the process can be significantly enhanced by leveraging single/multi-agent/environment interactions, which are grounded in the intelligent agents’ goal-driven competitions.

In this research, we propose a PRNG framework that integrates two complex systems: the cellular automata environment and agent(s), which are designed to play a

specific game for our purpose. Notice that although there is no widely accepted standard definition of complex systems/entities by the scientific research community, complex systems can be defined as systems in which many different components can work (e.g., cooperate or compete) with each other through multiple interactions [Ladyman et al., 2013]. Our gamified PRNG framework proposal consists of a linear rule 2D CA environment and a set of agents that can alter the state of the automata for their purposes. The introduced linear 2D CA environment is the basic random number generator with many neighborhood and state transition function variants. In addition to the CA environment, we use agent definitions that read from and take actions in the environment. To embed agent definitions into the environment, we gamified the cellular environment and assigned goals to the agents. While gamification is done through a number of alive cells in the environment, agent goals are defined as either maximizing the living (i.e., state value 1) or the dead (i.e., state value 0) cell counts. Agents take turns by playing in the environment, and at each turn, they are allowed to change the state(s) of a certain bounded number of environment cells according to their own goals. To realize intelligent decision-making mechanisms for the player agents, Monte Carlo Tree Search (MCTS) was chosen since it provides a high-quality decision-making performance even in cases where domain knowledge is limited [Browne et al., 2012]. In this research, our primary purpose is not to establish a benchmark gaming environment in which different algorithms can show and prove their quality, but to investigate whether competition between a single agent vs. the environment and/or two (or more) agents that act in an environment settings contribute to high-quality random number generation or not. To prevent possible bias effects in random number generation, all the agents play the introduced environment control game using the same MCTS-based algorithm with different goals. To test our proposed framework, we generated a large number of random bit streams using the agent-based gamification model across a range of parameters. Then, we tested those bitstreams regarding random number generation quality on the NIST test suite [Bassham III et al., 2010]. The obtained results showed that our proposed framework can produce random sequences of competitive high quality, some of which pass all the NIST test suite tests.

The manuscript is organized as follows: Section 2 introduces the related CA-based RNG and MCTS in games literature. Section 3 provides background information about the CA model. In Section 4, we give the details of the proposed PRNG framework. Section 5 includes the experimental setup used to evaluate the quality of the framework. Section 6 presents the results of NIST tests performed on the framework. Finally, Section 7 holds the conclusions.

## 2 Related Literature

### 2.1 CA-based RNG

CA-based RNG research has a long history over approximately the last 40 years. They were introduced in early pioneer work [Wolfram, 1986]. Various studies in the literature have utilized n-dimensional CA to generate random numbers. For example, [Tomassini et al., 2000] propose an evolutionary 2D CA rule to generate high-quality random numbers. In their research, the neighborhood information and update rules are represented as genomic sequences, and they utilize the genome representations in their

genetic algorithms to find better random generators. The sequence entropies are used as fitness function results. The genetic algorithm produces hybrid 2D CA rules tested on benchmarks and yields good results. [Kotoulas et al., 2006] utilizes a 1D CA to generate random numbers. The authors select real-time computer clock sequences as their seed values for a 1D CA. Values like day, month, year, hour, minute, and second are converted to a binary number to create the system's initial state. Their proposed generator is implemented as Field-Programmable Gate array (FPGA) hardware.

Shin et al. propose a PRNG based on the von Neumann neighborhood with a linear CA update rule [Shin et al., 2012]. The authors utilize two control elements combining AND, XOR, or OR logic functions to extend the expressive power of rule representations. This way, they still maintained randomness quality without increasing the linear rule count. [Hosseini et al., 2014] utilizes a hybrid solution based on two complex systems, 2D CA and Langton's ants, to generate high-quality RNGs. In their research, each cell is assigned an update rule determined by the virtual ant's position. Then, the CA makes an update based on cell rules and ants' movement in one step forward. That method results in high cycle lengths and good-quality random outputs. [Temiz et al., 2014] utilizes a hybrid update rule on 2D CA to generate random outputs. They have used two linear rules and a transition matrix to hybridize the update rule. [Sirakoulis, 2016] proposed a 1D hybrid autonomous DNA CA (HADCA) to generate random numbers. A nanomechanical DNA design capable of universal computation is modified to achieve HADCA simulations. The proposed HADCA can run different 1D CA rules in parallel and generate high-quality random sequences. [Bhattacharjee et al., 2017] proposes a 3-state CA to generate random numbers. The authors utilize a window-based approach to generate random numbers of any length. [Szaban, 2019] searches for rules on 1D totalistic CA that can generate good-quality random numbers and utilize entropy analysis to decide on the initial state of the 1D CA. The author performs the searches extended to 1, 2, 3, and 4 radius neighborhoods. [Mondal et al., 2019] propose an approach based on a secure image encryption schema for image communication and storage. Combining chaotic skew maps and elementary CAs, they obtained a fast PRNG with high key space. The encrypted image obtained due to their proposed scheme resists all known attacks.

In [Ostapov, et.al. 2023], the authors developed two symmetric encryption algorithms based on cellular automata: (i) Block-cipher based on Advanced Encryption Standard (AES) and uses 3-D CA (ii) Stream-cipher, which uses a hardware-software entropy generation by tracking of keystrokes and mouse pointer movement, and also developed a hash function modified via CA transformations. Their research concludes that the proposed block and stream ciphers can successfully be used for cryptographic applications. In the context of asynchronous CA, conversely, [Cicuttin et al., 2023] propose an analytical approach to evaluate and select suitable CA rules for TRNG with physical implementations of the asynchronous CA networks having any number of inputs. Another research based on the asynchronous CA model [Roy, 2023] shows that a fully exposed cycle asynchronous CA can improve the randomness quality of a poor RNG. In [Bhattacharjee and Vikrant, 2023], CAs are represented by first-degree equations and investigated as sources of randomness. The authors identified a list of good first-degree CA parameters that remain an excellent source of randomness irrespective of a change in the number of states. A special three-neighborhood, two-state CA is introduced as the underlying model of their PRNG [Bhattacharjee et al., 2023]. The authors propose a new PRNG that works like the Mersenne Twister (the de

facto PRNG used in many computer programs) but uses a much smaller state space while giving better performance than the Mersenne Twister family. Research that aims to obtain good RNG by applying a structural modification on CA is introduced [Poornima et al., 2024]. In their hybrid approach, the modification is done by coupling Langton's Ant Model and Programmable Controllable Cellular Automata with a sand pile. Their results, benchmarked with the statistical NIST test, were satisfactory and attained high cycle lengths.

## 2.2 MCTS in Games

Monte Carlo Tree Search (MCTS) utilizes a best-first search method that does not require a positional evaluation function [Coulom, 2006]. Instead of an evaluation function, MCTS performs random simulations on a given problem space and builds up a search tree [Lorentz, 2016]. These random simulations allow MCTS to sample the solution space from an input state to a final state. Although random samples are used while searching, MCTS, like many other heuristic search methods, has a balance factor between exploration and exploitation [Kemmerling et al., 2024]. With these explorations and exploitations, MCTS gets successively better at estimating the promising actions that can be taken by a specific state [Winands et al., 2008]. In other words, MCTS is an anytime algorithm that can return a valid result even if it is interrupted prematurely. But the longer the search, the better the result. Because of the anytime nature of MCTS, it is generally set to operate within the limits of a predefined computation budget rather than searching the entire solution space. This computation budget can be time, memory, or iteration [Browne et al., 2012].

An efficient MCTS application often requires problem-dependent modification or integration with other techniques in complex games with a high branching factor or real-time ones. A study that surveys domain-specific modifications and hybrid approaches to MCTS can be found in [Swiechowski et al., 2023]. The use of MCTS in real-time strategy games that require handling substantially significant combinatorial decisions and state spaces is studied [Ouessai et al., 2022]. The authors propose using domain knowledge from expert traces or expert-authored scripts to steer the MCTS algorithm search and reduce decision space. Their solution can maintain a high performance against different opponents. In another research to solve the limited-simulation-time problem of MCTS, a deep learning-based surrogate-assisted modified MCTS for the real-time video games domain has been proposed [Kim et al., 2024]. Research that couples deep learning and ensemble learning as a weighted voting method applied to the selection step of sub-nodes in the MCTS process in a Go environment is introduced [Guo, 2024]. Besides their intelligent search improvements in the basic MCTS process, the authors have achieved considerable improvement due to the asynchronous parallelization of the MCTS algorithm.

The importance of integrating domain knowledge into the search process is a known fact. [Crippa et al., 2022] introduces a comparative study between an extended single-player MCTS (SP-MCTS) for Sokoban and a solver integrating Iterative Deepening A\* (IDA\*) search with several problem-specific heuristics. The authors conclude that IDA\* is seemingly the best solver for the Sokoban game because of its easy integration of domain knowledge. One way to introduce domain knowledge to MCTS is to adopt a hybrid approach [Baier and Winands, 2018]. The authors combine the strategic strength of MCTS and the tactical strength of minimax by embedding

shallow minimax searches into the MCTS framework. As an alternative to exploiting heuristic strategies in search, [Liu et al., 2024] propose an approach to model the crossword puzzle resolution problem as a Markov decision process and apply the MCTS to solve it. Their results are competitive, and they have introduced candidate generation; the reward function mechanisms are open to further improvements.

Although the environment is fully observable in our game setup, MCTS faces challenges in games with imperfect information (e.g., DouDiZhu, Big2, and Pokemon) due to unobserved data from other players. To cope with this challenge, [Whitehouse et al., 2011] introduced a technique called determinization for making decisions in games by sampling instances of the equivalent deterministic game of perfect information. A similar approach is developed by using information sets, subsets of the game tree that contain all possible outcomes of a player's decision [Ihara et al., 2018] for the Pokemon games. Their experimental results indicate that Information Set MCTS outperforms the conventional MCTS that uses determinization. The Deep Monte Carlo (DMC) method proposed by [Zha et al., 2021] enhances the MCTS-based solutions with deep neural networks, action encoding, and parallel actors, and effectively addresses the problems in the DouDiZhu. To address the challenges of resource usage and training time, DouDiZhu [Luo and Tan, 2024] developed two innovative methods: the Opponent Mode and Optimized Deep Monte-Carlo (ODMC). They can attain comparable performance with DMC, but in 25% in training time on the same hardware configuration. With game-specific intelligent agent design, current research shows that a robust method for achieving general intelligent game agent design implies MCTS coupled with reinforcement learning methods potentially serve as a general framework due to their theoretically higher optimization possibilities [Li et al., 2025].

### 3 Background

#### 3.1 Cellular Automata

Cellular automata are used for the mathematical modeling of systems that consist of many simple components, called cells, interacting with each other and creating complex behavioral patterns resulting from the interactions [Kılıç, 2024]. They are discrete deterministic dynamical system models in which the cells are arranged in a regular lattice structure and can be 1-Dimensional (1D) or Multi-Dimensional (2D, 3D, ...,  $nD$ ). In traditional CA, cells can be in only one of the  $s$  finite states at any given time. The state values express the status of cells in the environment at time  $t$ . The new state of each cell at time step  $t+1$  is determined by the state of the cell itself and its neighbors' state at time  $t$  on the lattice. State changes of a cell can be defined by the state transition function  $\phi$ . The state transition function takes the state information of itself and its neighbor cells as its input value. There are different neighborhood definitions in CA, defined by their neighborhood radius ( $r$ ) and patterns. Some commonly used neighborhood definitions for 2D CA are given in Figure 1.

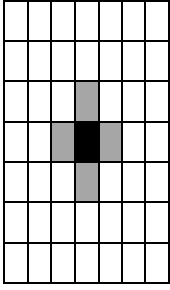
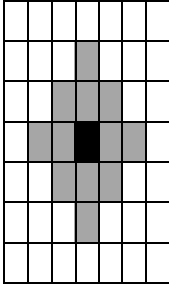
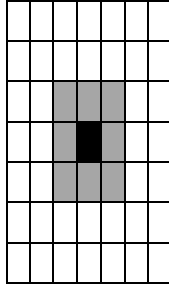
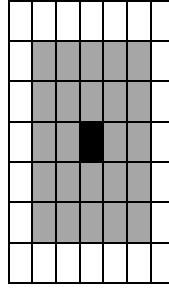
Von Neumann Neighborhood		Moore Neighborhood	
Radius, r = 1	Radius, r = 2	Radius, r = 1	Radius, r = 2
			

Figure 1: Some examples of neighborhood definitions are used in 2D CA

In Figure 1, the cell marked in black represents the cell for which the next state is to be found, and the cells marked in gray represent its neighbors. Note that the neighborhood definition includes the cell itself, as well. The neighbor cell states are utilized in the next state calculation of the current cell. At each discrete time step, state updates are performed on all cells to generate a new environmental state synchronously. For each cell  $x_{i,j}$  with Moore neighborhood having radius  $r = 1$ , the next state of the cell at time  $t + 1$  is calculated as:

$$x_{i,j}^{t+1} = \varphi(x_{i-1,j-1}^t, x_{i,j-1}^t, x_{i+1,j-1}^t, x_{i-1,j}^t, x_{i,j}^t, x_{i+1,j}^t, x_{i-1,j+1}^t, x_{i,j+1}^t, x_{i+1,j+1}^t) \quad (1)$$

Note that a 9-neighbor, s-state setup allows the definition of  $s^9$  different  $\varphi$  functions [Packard & Wolfram, 1985].

### 3.2 Linear Cellular Automata

In the  $s = 2$  and  $r = 1$  2D CA with Moore neighborhood setup, one can represent  $2^{512}$  possible  $\varphi$  functions. To reduce the size of the huge  $\varphi$  functions space and make it more investigable for our random number generation purpose, we worked on a subset of CA called Linear Cellular Automata (LCA) whose state transition function  $\varphi$  is limited to linear functions:

$$x_{i,j}^{t+1} = c_0 x_{i,j}^t + c_1 x_{i,j+1}^t + c_2 x_{i+1,j+1}^t + c_3 x_{i+1,j}^t + c_4 x_{i+1,j-1}^t + c_5 x_{i,j-1}^t + c_6 x_{i-1,j-1}^t + c_7 x_{i-1,j}^t + c_8 x_{i-1,j+1}^t \pmod{2} \quad (2)$$

where coefficient  $c_i \in \{0, 1\}$  and  $0 \leq i \leq 8$ . As a consequence, we need to consider  $2^9 = 512$  possible linear state transition functions based on the simplification provided by linearity. Figure 2 depicts our encoding template for the enumeration of each LCA.

$C_6$ (64)	$C_7$ (128)	$C_8$ (256)
$C_5$ (32)	$C_0$ (1)	$C_1$ (2)
$C_4$ (16)	$C_3$ (8)	$C_2$ (4)

Figure 2: Some examples of neighborhood definitions are used in 2D CA

With the defined coefficient ordering, state transition rules can be encoded as:

$$\sum_{i=0}^8 c_i 2^i \tag{3}$$

For example, the state transition rule 171 can be encoded by coefficients:  $c_0 = 1, c_1 = 1, c_2 = 0, c_3 = 1, c_4 = 0, c_5 = 1, c_6 = 0, c_7 = 1, c_8 = 0$ .

#### 4 The Proposed PRNG Framework

According to [Bassham III et al., 2010], a perfect random bit sequence could be generated through the flips of an unbiased and fair coin with sides labeled 0 and 1. Each flip would have a probability of exactly 50% producing 0 or 1, while each flip is independent. This notion is visualized in Figure 3. A quality PRNG should break the correlation between its generated bits, and the values of zeroes and ones should be uniformly distributed. The next values from a quality PRNG should not be predicted from the previously generated values.

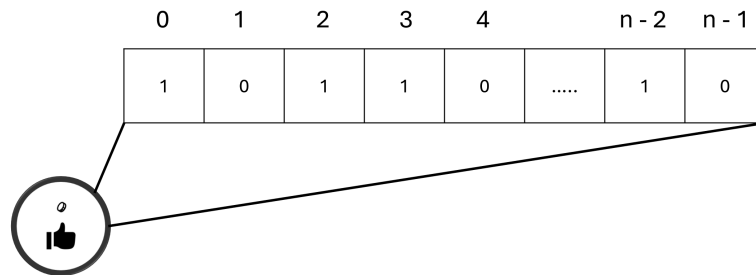


Figure 3: Perfect conceptual RNG

The CA environment component of our proposed PRNG framework is established on a previous work reported by [Kılıç, 2024]. However, in our study, the 2D LCA environment is gamified to develop a higher-quality PRNG. As pointed out in Section 2.1, related literature, CA-based RNGs are not new, but the way the random number sequences are constructed directly affects the performance of the designed RNG.

#### 4.1 Construction of Random Number Sequences from CA State Histories

For the elementary 1D CA with  $n$  cells with some arbitrary initial configuration, a random bit sequence of length  $z * n$  can be obtained by placing the first  $z$  temporal configurations one after the other [Rubio et al., 2004]. Such a procedure is not secure since if a third party obtains some of the output produced and the automata that produce the output are known, it will be easy to predict subsequent outputs. A better approach would be to use the states of a particular cell. In other words, a bit sequence can be generated through the temporal evolution of a specific cell. In our work, we used the temporal evolution history of a 2D LCA with  $N * N$  grid size with periodic boundary conditions to generate random numbers. The evolution history of states are stacked on top of each other to create a  $m * N * N$  3D cube where  $m$  represents the stack length. The stacked cube is depicted in Figure 4.

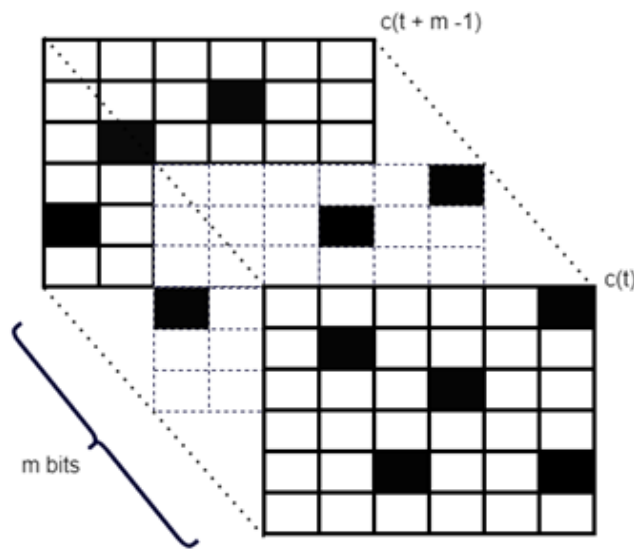


Figure 4: Temporal depth  $m$  state transition history of a  $N * N$  size 2D LCA represented as patterns of 0 (white) and 1 (black) cells

In the setup, each cell has  $m$  many historic values consisting of ones and zeroes. So, after  $m$  updates, each cell produces  $m$ -bit random sequences. In a 2D LCA environment with the properties given above, after  $m$  updates,  $N * N$  number of  $m$ -bit sequences can be obtained. As in [Kılıç, 2024], the bit sequences were added one after the other from left to right and top to bottom to cover the entire 2D pattern area. The bit order is arranged so that the oldest state corresponds to the most significant bit (see Figure 5).

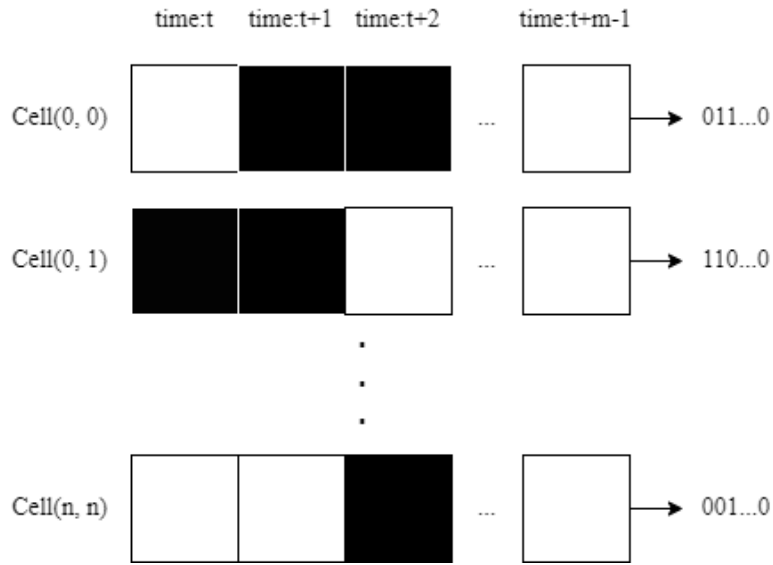


Figure 5: Creation of a random bit sequence from an indexed cell

Intelligent agents control the defined CA environment to achieve their goals for high-quality random number generation. The agents must act on discrete cell positions on the 2D grid by the end of their decision-making stage. The resulting  $m$ -bit sequences can be transformed into different output types like integer, character, etc. To generate an integer number having a value up to  $u$ ,  $m$  needs to be selected to satisfy the constraint  $2^m \geq u$ . For example, if a number up to  $u = 400$  is requested, the depth  $m$  should be at least 9 since 9 bits represent up to value 512. To avoid biasing on either side of the range, normalization (or remapping) is used. Unlike the modulus operator, remapping ensures that one range is evenly distributed over the other. Remapping is done through Equation (4).

$$u_i' = \frac{u_i - \min(u)}{\max(u) - \min(u)} * (\max'(u) - \min'(u)) + \min'(u) \tag{4}$$

where  $u_i$  is the number to be remapped,  $u_i'$  is the remapped number,  $\min(u)$  and  $\max(u)$  are the minimum and maximum values that  $u_i$  can take,  $\min'(u)$  and  $\max'(u)$  are the minimum and maximum values that  $u_i'$  can take, respectively.

#### 4.2 The Considered Transition Rule Set for 2D LCA

The linear 2D CA environment is considered to generate random bit streams. As mentioned in Section 3.2, when using a  $s = 2, r = 1$  2D LCA with Moore neighborhood,  $2^{512}$  possible  $\varphi$  functions can be defined. With linearization of the  $\varphi$  through Equation (2), the number is reduced to 512. In [Kılıç, 2024], an extensive experimental analysis of 2D CA is performed on linear rules and their sensitivity to initial fullness (i.e., the number of 1 values) ratios to find the best-performing linear 2D

CA random number generators. The 512 linear rules are tested with varying fill rates with a 5% step size on the NIST Statistical Test Suite. According to the author's findings, the initial configuration of a linear 2D CA environment affects the random number generation. The best-performing linear 2D CA random generators obtained (together with the initial fullness ratios) by the author are given in Table 1.

Rule Number	Fullness Ratio
29	%50
95	%25
350	%55
351	%45
382	%30
471	%70
475	%60

Table 1: The best-performing linear 2D CA rules and their fullness ratios (refer to [Kılıç, 2024] for details)

In this study, the generator environment utilizes the rules given in Table 1 as its transition rule  $\varphi$ , which defines the game environment. Also, to be consistent with the literature, the coefficient ordering given in Figure 2 and periodic boundary conditions assumptions with Moore neighborhood of radius 1 are made.

### 4.3 Random Number Generation Through Gamified 2D LCA Framework

The research aims to obtain high-quality PRNG via gamification to achieve the design and implementation of a self-sufficient game environment. To achieve this goal, we need to introduce an external source of complexity to the existing 2D LCA-driven PRNG process: Game-playing agent(s). A similar idea that couples a complex system with a 2D CA environment to investigate the effect of learning a forward model on the performance of a statistical forward planning agent is studied in [Lucas et al., 2019]. The authors transform Conway's Game of Life simulations into a single-player game to either preserve as much life as possible or extinguish all life as quickly as possible. The environment was gamified from the agent's perspective to integrate the player agents into the environment.

Although there is no consensus about its definition in the literature, gamification can be described as using game elements in non-game contexts [Seaborn and Fels, 2015]. According to [McGonigal, 2011], four elements make up a game: goal, rules, feedback system, and voluntary participation. Our proposal fulfils the first three elements in the 2D LCA environment. Since programmed entities are used, the voluntary participation element is discarded.

#### The Goal:

A player agent has two possible goals: (i) To maximize the alive cell count in the environment and (ii) To minimize the alive cell count in the environment. There could be many other goal definitions, but only those two are for our binary sequence generation purpose. Based on the goal definitions, we have three player-agent strategies:

- Maximalist agent, an agent that aims to maintain the highest number of alive cells in the environment.
- Minimalist agent, an agent that aims to maintain the lowest number of alive cells in the environment.
- Random agent, an agent that takes random actions in the environment without any specific goal.

Since the described 2D LCA game does not have a clear terminal state, we define a pseudo-terminal state to mimic the end of the game event. Each agent with an actual goal believes the game will end after a certain playout and runs their search algorithms accordingly. The pseudo-terminal game event end state condition is ‘look ahead amount’ since the agents must look at a certain number of moves ahead during their search phases.

#### The Rules:

The following items define the rules of the game:

1. The game can be played with many player agents with a minimum player agent count of one. If there is only one player agent, it plays against the environment.
2. Player agents take their turns in order.
3. Each player is allowed to change the state of a certain  $k$  number of cells in the environment. The number of alterable cells can be less than  $k$  but never exceeds it.
4. The 2D LCA environment is updated after each player's action. The other player performs its actions in the updated environment.
5. Environment update rules and state transition functions known to all agents participating in the game.

#### Feedback System

The environment is fully observable to all participating player agents. All agents can evaluate the number of alive or dead cells and act based on their goals. From the rules defined above and the state observability, the 2D LCA game environment is fully observable, sequential, static, discrete, single/multi-agent, and deterministic.

#### Agent Description

Player agents are the key elements of the setup. In our context, the term player agent defines an abstraction of an encapsulated logic determined by the agent's goal defined above. To describe the player agent's behavior, we refer to its Performance, Environment, Actuator, and Sensor (PEAS) description [Russell and Norvig, 2021]:

- Performance: (Total number of cells in the task environment whose status is alive (or dead) according to an agent's purpose) /  $N^2$ .
- Environment: A game environment consisting of  $N * N$  cells whose state dynamics are defined by the living (1) or dead (0) states and a linear 2D CA state transition rule.
- Actuators: The ability of the agent to change the states of at most  $k$  cells that it chooses to set into either living (1) or dead (0) states, or not act even if its turn.
- Sensors: Total  $N^2$  number of sensors reporting the status of all cells at a given time step (Full-observability of the environment).

All three agent types defined above must show a behavior in the environment. On the other hand, the maximalist and minimalist agents need a decision-making mechanism in line with the rules stated above and based on their own goals. While maximalist and minimalist agents search for moves that meet their goals, the random agent generates random moves without any goal-directed strategy. However, as one might expect, all agents need a random number generator in their decision-making. We use our generators in the agents' decision-making mechanisms to prevent interference from other generators. Each agent utilizes the environment-only setup as their basic random number generator defined by the linear 2D CA rules. When an agent requires a random number, the environment-only generator provides one, as described in Section 4.1.

Any search algorithm can be used as a decision-making mechanism for the agents with a goal (maximalist and minimalist agents), and all participants know the update rule. This is one of the strengths and the flexibility of the proposed generator. Due to its high generalizability, the MCTS algorithm is chosen as the agent search algorithm. Interim studies have shown that the default MCTS backpropagation policy performs poorly and yields suboptimal results. The default policy does not consider the actions of other players that may be present in the environment. Although the agents do not communicate and are unaware of each other, the environment acts as an agent under certain constraints. Interim studies also showed that the environment, with the selected transition rule set, keeps the alive cell count in a particular range after each update. Every agent, regardless of whether they are minimalists or maximalists, must compete against the environment. Therefore, the default policy is replaced with the backup Negamax policy, which is analogous to the *Negamax* variant of the *MiniMax* search algorithm [Browne et al., 2012].

During the MCTS simulation stage, an action is evaluated through random playouts. Those playouts are simulated until the evaluated state reaches a terminal state. On the other hand, as mentioned before, in our environment, there is no actual terminal state; there is only a pseudo one. Therefore, state heuristics are utilized instead of a terminal state evaluation. State heuristics can evaluate an incomplete state and assign a score (win/lose). We know that the state heuristics can reduce the simulation times drastically. Alternative heuristics have been tried, but it has been found that making a high-quality move requires a tremendous amount of search time. This is not surprising considering the size of the problem at hand. Since the study aims neither to maximize nor to minimize the alive cell counts nor to search for a Garden of Eden configuration [Sutner, 1989], the following simple heuristic is used.

$$\begin{aligned}
 h_{max}(c) & \begin{cases} 0 & c \leq \frac{N^2}{2} \\ 1 & c > \frac{N^2}{2} \end{cases} \\
 h_{min}(c) & \begin{cases} 1 & c \leq \frac{N^2}{2} \\ 0 & c > \frac{N^2}{2} \end{cases}
 \end{aligned} \tag{5}$$

where  $N$  represents the cell count in a single dimension of the 2D LCA game environment,  $c$  represents the total alive cell count,  $h_{max}(c)$  represents the heuristic for

maximizer agent,  $h_{min}(c)$  represents the heuristic for minimizer agent. The two heuristics are inverse of each other and do not cause any bias in the random number generation behavior of the system. They are fast and good enough to allow agents with two different strategies to make different decisions in the same state.

The structured temporal flowchart representation of the framework that generates random numbers in a two-player agent and environment setup is depicted in Figure 6. At time  $t_0$ , all related parameters, including the pseudo-terminal game end state (i.e., the agent's look-ahead amount) and the initial states of Agent-1, Agent-2, and the Environment, are set. Also, the agent cycles are started. At time step  $t_i$ , the environment state is read by Agent-1. Based on the agent's objective and the result of the intelligent search driven by the MCTS algorithm, the agent decides its moves, and the Environment state is updated accordingly. The following update is done by executing the state transition function  $\varphi$ , which describes the behavior of the Environment. The updated state is appended to the state history. At time step  $t_{i+1}$ , Agent-2 takes control and decides its moves as Agent-1 does but based on its individual competing goal. The Environment state is updated accordingly, and so on. If a sufficient number of random sequences is generated, the Environment ends the process, and the game is over at time step  $t_{n-1}$ . Otherwise, the agents and the Environment continue to play the game.

There can be as many agents as introduced to the environment, where each one of them may require a random number up to  $u$  where  $0 \leq u \leq k$ , and  $k$  indicates how many cell states can be altered in each turn by an agent. The modular and flexible structure of the framework allows the introduction of new agents with different goals supported by alternative search algorithms for agent implementation.

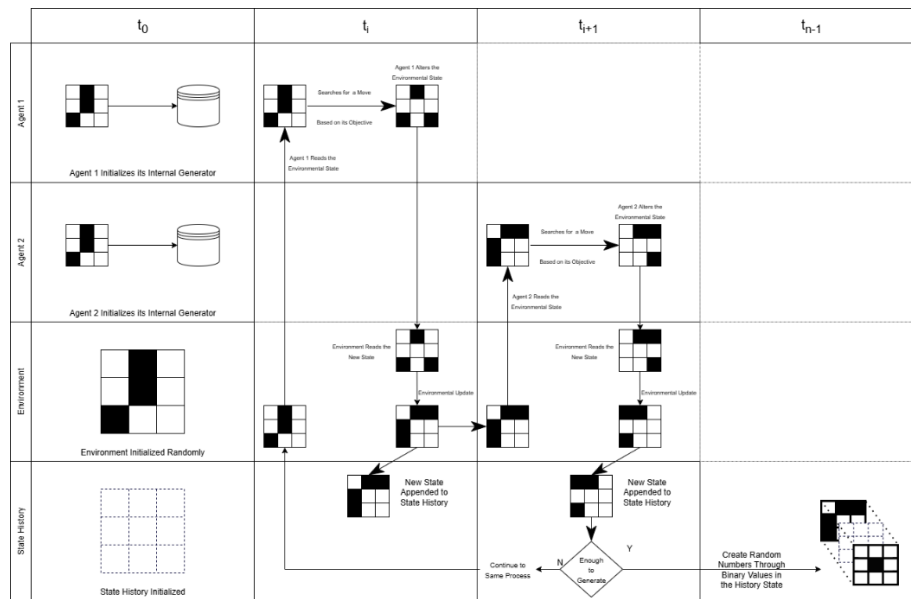


Figure 6: The structured temporal flowchart representation of the framework that defines the random number generation process for two MCTS agents

## 5 Experimental Setup and Methodology

To test the random number generation quality of the framework, files containing the generated random numbers are produced by using the following alternative parameter values:

- *Cellular Automata Grid Size (N)*: The outputs are generated for 20x20, 15x15, and 10x10 grid size configurations, which are three alternative setups in total.
- *Player Agent Setups (P)*: The outputs are generated for environment-only, environment and 1 random agent, environment and 1 maximizer MCTS agent, and environment and 2 MCTS agents (one maximizer and one minimizer), for a total of four alternative setups.
- *Player Agent Action Count (K)*: The maximum number of actions an agent can take during its turn. The outputs are generated for values  $k = 10, 20$  and 40, with three alternative setups.
- *Linear CA Rules*: The seven rules identified in [Kılıç, 2024] are used: Rule29, Rule95, Rule350, Rule351, Rule382, Rule471, and Rule475. Seven alternative setups in total. Figure 7 depicts a tile-based representation of the seven 2D LCA rules together with initial density values.
- MCTS-related parameters:
  - *Iteration Value (I)*: Number of search steps that MCTS agents perform when searching for a single action. It is the computation limit for the MCTS algorithm. Files were generated only for the value 400, which was big enough to attain satisfactory results.
  - *Child Amount (C)*: In the search step, MCTS creates new child nodes using the set of actions that can be taken from a state. Since our set of possible actions does not change at any level, the number of children in a node is set to 3.
  - *Look Ahead Amount (LA)*: Each MCTS agent performs random playouts until it reaches a pseudo-terminal state. A fixed value of 10 was used for LA.



Figure 7: The tile-based representations of the seven successful 2D LCA environments with their initial state densities, where caption xxx:y.yy indicates the tile for rule number xxx with initial density y.yy

For the one-agent and two-agent setup, the number of generated files is  $3 \times 2 \times 3 \times 7 = 126$ . For the environment-only setup, it is  $3 \times 7 = 21$  files. For the environment with a random agent setup, we have  $3 \times 3 \times 7 = 63$ , in which three alternative player action counts are considered. So, the total number of all possible combinations that configurations generate is  $126 + 21 + 63 = 210$  files. Each file contains 125 million bits, nearly equal to 122 MB. With 210 files, the total file size is around 25.6 GB. Remember that the framework supports two linear 2D CA environments; the agents use one to generate random numbers, and the other produces the main outputs of the study.

Even though these two environments diverge from each other after the first CA update, the initial configuration of the two environments is initialized in the same way. For initial configuration generation, the Mersenne Twister algorithm from the C++ std library is used according to the experimentally observed linear 2D CA with their initial density values. When the CA environment reaches the history size of 64, the random bit sequence is created as described in Section 4.1 and written to a file. The history size value is the file append depth ( $d$ ).

No matter how many tests are performed, proving that a given sequence of numbers is genuinely random is impossible. However, it can be shown that its randomness is insufficient. Statistical tests evaluate the pseudo-randomness of a given sequence of numbers. The NIST Statistical Test Suite, which contains different comprehensive statistical tests, is used. The NIST Statistical Test Suite contains 15 randomness tests, and their characteristics are summarized below, see [Bassham III et al., 2010]:

- Frequency Test: Focuses on the proportion of zeroes and ones for the entire sequence. It determines if the given sequence approximately has the same number of ones and zeroes.
- Frequency Test within a Block Test: Performs a Frequency Test over  $M$ -bit blocks.
- Runs Test: Focuses on the total number of runs in the given sequence. Runs can be identified as: A run of length  $k$  consists of exactly  $k$  identical bits and is bounded before and after with a small quantity of the opposite value. In short, this test analyzes the oscillation between zeroes and ones. It can capture whether the oscillation is too fast or too slow.
- Longest Run of Ones in a Block Test: Analyzes the longest run of ones within  $M$ -bit blocks.
- Binary Matrix Rank Test: Analyzes the linear dependence among fixed-length substrings of the original sequence.
- Discrete Fourier Transform (Spectral) Test: Analyzes periodic features (repetitive patterns) in the given sequence. Periodicity implies non-randomness.
- Non-overlapping Template Matching Test: Analyze the number of occurrences of pre-specified target strings, which are provided with the benchmark itself. A sliding window is used over the bit stream. The application of this test requires 148 independent applications against internal parameters.
- Overlapping Template Matching Test: This test's objective is the same as the Non-Overlapping Template Matching Test. The difference is in the sliding window movement. This test moves the window one bit when a pattern is found instead of moving it through the entire pattern.
- Universal Statistical Test: Tries to compress the given sequence. If a sequence can be compressed significantly, it implies a non-random sequence.
- Linear Complexity Test: Determines whether the sequence is complex enough to be considered random. Analyzes the linear feedback shift register length to achieve its aim.
- Serial Test: Random sequences have uniformity features: Every  $m$ -bit pattern has the same chance of appearing as every other  $m$ -bit pattern. This performs a frequency check on the entire sequence by utilizing all combinations of an  $m$ -

bit sequence. This test generates two different  $p$  values, each of which is assumed to be an independent test.

- Approximate Entropy Test: Like the *Serial Test*, analyzes the frequency of all possible overlapping  $m$ -bit patterns across the entire sequence. It compares the frequency of overlapping blocks of two adjacent ( $m$  and  $m + 1$ ) against the expected result for a random sequence.
- Cumulative Sums (Cusum) Test: Converts the 0 in the sequence to  $-1$  and performs a cumulative sum on the new sequence. Then, performs a random walk on the sequence to analyze deviations from zero. Significant deviations from zero mean non-random data. This test generates two different  $p$  values, each of which is assumed to be an independent test.
- Random Excursions Test: Analyzes the number of cycles after performing  $K$  visits in a cumulative sum random walk. Determines if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence. The application of this test requires 8 independent applications against different internal parameters.
- Random Excursions Variant Test: Detects deviations from the expected number of visits to various states in the random walk. The application of this test requires 18 independent applications against different internal parameters.

In the suite, the tests are independent; some are performed more than once with different internal parameter values. So, together with 148, 8, and 18 independent test performances for *Non-overlapping Template Matching*, *Random Excursions*, and *Random Excursions Variant Tests*, respectively, and 2 times independent performances for *Serial* and *Cumulative Sums Tests*,  $148 + 8 + 18 + 2 + 2 + 10 \times 1 = 188$  test are performed in total.

After each test, the NIST Test Suite calculates a  $p$ -value based on the provided bit stream. The  $p$ -value represents the probability that a true random generator produces the given bitstream [Bassham III et al., 2010], [Hosseini et al., 2014]. A test considered

as passed if the calculated  $p$ -value lies within a confidence interval  $\hat{p} \pm 3 \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$ ,

where  $\hat{p}$  is  $\hat{p} = 1 - \alpha$ ,  $\alpha$  is the significance level and  $n$  is the sample size. NIST test suite uses 0.01 for the  $\alpha$  value. If a test result falls outside of this range, then there is evidence that the data is non-random. For example, with a sample size of  $n = 100$  and

$\hat{p} = 1 - \alpha = 1 - 0.01 = 0.99$  the confidence interval is  $0.99 \pm 3 \sqrt{\frac{0.99(1-0.99)}{100}} =$

$0.96015 \approx 0.96$ . In our experiments, we use a sample size of 100. Based on this size, the test is considered successful if a test's  $p$ -value is greater than 0.96. *Random Excursions* and *Random Excursions Variant* tests evaluate the inputs in terms of their applicability and perform the final tests on these applicable ones. Those tests are considered successful if at least 96% of the applicable tests are successful.

In [Kılıç, 2024], to prevent generating data with an insufficient number of cycles for the *Random Excursions* and *Random Excursions Variant* tests, the file append depth ( $d$ ) is taken as  $d = 64$ , not  $d = 4$  as for the other tests. In our experiments, file append depth  $d$  is fixed to 64 for all tests. To evaluate the system, the total number of tests that passed ( $p$ -values or percentages higher than 0,96) out of 188 tests is used as a success metric, namely Total Passed Tests (TPT). It is computed as  $TPT = \sum_{i=1}^{188} P_i =$

$\{0, 1\}$  indicates whether the system passed (i.e. score 1) from the  $i^{th}$  test or not (i.e. score 0). Consequently, the maximum score a generator can obtain from the NIST Test Suite application is 188.

## 6 Experimental Results and Discussions

In the experiments, we consider the 7 environment-only results and the results obtained for 11 known PRNGs, both of which are presented in [Kılıç, 2024] for our comparative study purpose. Table 2 shows the results obtained for 18 PRNGs from the literature.

Generator	$\sum_{i=1}^{188} P_i$	Generator	$\sum_{i=1}^{188} P_i$
Rule 29:0.50*	186	Mersenne Twister***	187
Rule 95:0.25*	187	Blum-Blum Shub***	186
Rule 350:0.55*	185	Micali-Schnor***	185
Rule 351:0.45*	185	Modular Exponent.***	185
Rule 382:0.30*	185	Cubic Cong.***	185
Rule 471:0.70*	187	Quadratic Cong.-2***	184
Rule 475:0.60*	185	Quadratic Cong.-1***	183
CA & Langton's Ant**	188	G Using SHA-1***	183
Linear Cong.***	188	XOR***	26

\*[Kılıç, 2024]; \*\*[Hosseini et.al., 2014]; \*\*\* [NIST Test Suit]

Table 2: NIST-based TPT success scores of 18 different PRNGs from the literature

Table 3 presents the overall performance results obtained for 3 alternative Grid Sizes, 3 different Player Agent Setups, 3 different Player Agent Action Counts, and 7 different 2D LCA Rules with initial densities. The green cells indicate the existence of improvement from the corresponding environment-only result from the literature given in Table 2. For example, the top-left cell shows the attained score of 188 as the result of environment setup based on Rule 29: 0.50 with 1 agent that can act on  $K = 10$  cells of the  $20 \times 20$  grid. The attained value indicates an improvement from the obtained corresponding environment-only Rule 29: 0.50 result 186 from the literature (see Table 2). The orange cells imply equal performance cases, while the red cells point to lower performances.

Relatively poor performances are obtained for the random agent setup, whose random behavior is generated by the environment-only PRNG setup. However, as the grid size increases, it is possible to attain better performances. This is mainly due to keeping the number of control points ( $K$ ) fixed against increasing grid sizes. For all different 2D LCA setups, the relatively small number of random control points results in an improved PRNG quality. Besides quality, the random number generator's computational performance (i.e., speed) is also essential. There is a known trade-off between the quality and speed of PRNGs. The trade-off is also observed among our 1-

agent, 2-agent, and random-agent setups. Clearly, 1-agent and 2-agent, which play maximizer/minimizer roles in MCTS setups, demand more resources during their algorithmic deliberation, resulting in higher-quality random number generation. On the one hand, the random-agent setup shows a generation speed quantified by only  $k$  number of the environment-driven random number generation for the control point assignments without any algorithmic deliberation. On the other hand, random agents generate relatively low-quality random numbers.

For the 1 maximizer MCTS agent and maximizer vs. minimizer MCTS agents setup, we obtain the best (i.e., 188/188) performances for almost all considered environment types, varying control points, and grid sizes. In particular, the best possible performances are attained for the  $K = 40$  control points case for both agent/environment setups. So, introducing one or two intelligent MCTS agents to the environment, with increased action/control points, improves PRNG quality. There are two 187/188 success cases in which 1 maximizer MCTS agent setup does not outperform but performs only equally well as the environment-only case: i) Environment 95: 0.25, Action Points Count  $K = 10$  and Grid Size  $20 \times 20$  in which the failed test is *Random Excursion Variant*, and ii) Environment 95: 0.25, Action Points Count  $K = 20$  and Grid Size  $15 \times 15$  in which the failed test is *Universal Statistical Test*. Similarly, for the maximizer vs. minimizer MCTS agents in Environment 29: 0.50, Action Points Count  $K = 20$  and Grid Size  $10 \times 10$  setups, both failed tests are *Random Excursion Variant*, and the setup performed equally well as an environment-only setup where their success scores are both 186/188.

Total Passed Tests (TPT) = $\sum_{i=1}^{188} P_i$										
20x20 Grid	K = 10			K = 20			K = 40			
	Linear Rule	1 Agent & Env.	2 Agent & Env.	Random Agent & Env.	1 Agent & Env.	2 Agent & Env.	Random Agent & Env.	1 Agent & Env.	2 Agent & Env.	Random Agent & Env.
29:0.50	188	188	188	188	188	188	188	188	188	188
95:0.25	187	188	188	188	188	187	188	188	188	188
350:0.55	187	188	188	188	188	188	188	188	188	188
351:0.45	188	188	187	188	188	188	188	188	188	188
382:0.30	188	188	188	188	188	188	188	188	187	188
471:0.70	188	188	188	188	188	188	188	188	188	188
475:0.60	188	188	188	188	188	188	188	188	188	188
15x15 Grid										
29:0.50	188	188	186	188	188	188	188	188	188	186
95:0.25	188	188	186	187	188	184	188	188	188	186
350:0.55	187	188	187	188	188	187	187	188	188	186
351:0.45	187	188	187	188	188	187	188	188	188	185

382:0.30	188	188	188	188	188	187	188	188	186
471:0.70	188	188	188	188	188	188	188	188	183
475:0.60	188	188	188	188	187	187	187	188	185
<b>10x10 Grid</b>									
29:0.50	188	188	160	188	186	187	188	188	161
95:0.25	188	188	187	188	188	187	188	188	187
350:0.55	188	188	188	188	188	188	188	188	186
351:0.45	188	188	186	188	188	188	188	188	187
382:0.30	187	188	187	188	188	184	188	188	177
<b>Color Legend:</b>	Better Performance		Equal Performance			Lower Performance			
471:0.70	188	188	185	188	188	186	188	188	186
475:0.60	188	188	184	188	188	184	188	188	187

Table 3: Proposed generator's result with different experimental setups

To identify the factors that most strongly influence the score, statistical tests were conducted. Specifically, ANOVA was selected for its ability to assess whether there are statistically significant differences in mean scores across multiple groups. Tukey's HSD test was performed as a post-hoc procedure to determine which specific group pairs differ significantly. To perform these tests, categorical variables need to be identified against the score metric. In the experimental setting, the following categories were selected: three alternative grid sizes, three different agent configurations, three distinct action counts, and seven unique two-dimensional LCA rules. One-way Analysis of Variance (ANOVA) was conducted for each category, and the p-values of each category are given in Table 4.

Category	P-Value
Agent Category (1 Agent, 2 Agent, Random Agent)	0.000078
K Value Category (10, 20, 40)	0.459012
Grid Size Category (100, 225, 400)	0.018427
Linear Rule Category (29, 95, 350, 351, 382, 471, 475)	0.197286

Table 4: ANOVA statistical test results of each category

From these results, statistically significant differences in mean scores were detected for Agent Category and grid size ( $p < 0.05$ ), whereas K Value and Linear Rule did not show significance. As ANOVA is designed to detect whether at least one group mean differs from the others [Keppel and Wickens, 2004], the findings suggested that differences existed, without specifying which group pairs were responsible. To overcome this limitation, Tukey's Honest Significant Difference (HSD) was performed as the next step. Tukey's HSD was selected due to its ability to perform all possible

pairwise comparisons between group means. The following tables give the Tukey's test results.

Group 1	Group 2	Mean Diff	P-adjusted	Reject
1 Agent	2 Agent	0,0635	0,9	False
1 Agent	Random Agent	-1,9524	0,001	True
2 Agents	Random Agent	-2,0159	0,001	True

Table 5: Tukey's HSD results for Agent Category

The results presented in Table 5 indicate no statistically significant difference between the 1-agent and 2-agent environments. However, both environments demonstrate a statistically significant improvement compared to the random agent environment. These findings support the article's hypothesis that environments incorporating agents with objectives perform better compared to environments incorporating agents without objectives.

Group 1	Group 2	Mean Diff	P-adjusted	Reject
10	20	0,4127	0,7058	False
10	40	-0,254	0,8745	False
20	40	-0,6667	0,4336	False

Table 6: Tukey's HSD results for Action Count (K Value) Category

The results obtained from Tukey's test for the K value category, as shown in Table 6, indicate that there is no significant difference between the various K values.

Group 1	Group 2	Mean Diff	P-adjusted	Reject
100	225	0,9524	0,1724	False
100	400	1,4921	0,0146	True
225	400	0,5397	0,5609	False

Table 7: Tukey's HSD results for Grid Size Category

The Tukey's HSD test results for the grid size category, presented in Table 7, reveal a statistically significant difference between the grid sizes 10x10 and 20x20. In contrast, no significant differences were found between the 10x10 and 15x15 grid sizes, nor between the 15x15 and 20x20 grid sizes. This suggests that increasing the grid size from 10x10 to 20x20 has a notable impact.

Group 1	Group 2	Mean Diff	P-adjusted	Reject
29	95	1,7778	0,3111	False
29	350	1,963	0,2011	False
29	351	1,9259	0,2208	False
29	382	1,5185	0,5062	False
29	471	1,8519	0,2635	False
29	475	1,7407	0,3366	False
95	350	0,1852	0,9	False
95	351	0,1481	0,9	False
95	382	-0,2593	0,9	False
95	471	0,0741	0,9	False
95	475	-0,037	0,9	False
350	351	-0,037	0,9	False
350	382	-0,4444	0,9	False
350	471	-0,1111	0,9	False
350	475	-0,2222	0,9	False
351	382	-0,4074	0,9	False
351	471	-0,0741	0,9	False
351	475	-0,1852	0,9	False
382	471	0,3333	0,9	False
382	475	0,2222	0,9	False
471	475	-0,1111	0,9	False

Table 8: Tukey's HSD results for Linear Rule Category

The Tukey's HSD test results for the Linear Rule category, as presented in Table 8, indicate that there are no statistically significant differences between the linear rules. From the results obtained, it is evident that the most significant category is the Agent Category. Therefore, a confidence interval was calculated for this category at an alpha level of 0.05, and the results are given in Table 9.

Agent Category	Lower CI	Mean	Upper CI
1-Agent	187.788489	187.873016	187.957543
2-Agent	187.859838	187.936508	188.013178
Random Agent	184.668488	185.920635	187.172782

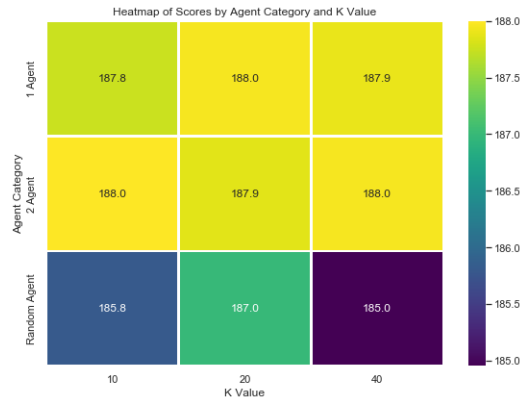
Table 9: Confidence Intervals for Agent Category

The resulting intervals are consistent with the results obtained from the ANOVA and Tukey’s HSD tests. Based on these experiments and results, generators with 2 agents can be considered the most successful ones. This is observable from the upward-sloping curve towards the 2-agent environment in Figure 8. However, the results of these two environments are too close to demonstrate a statistically significant difference. It should be noted that even if the number of agents is 1, this agent has a purpose that conflicts with the environment itself. The environment itself can be considered as a separate agent.



Figure 8 Confidence Intervals for Each Agent Category

For visualization, and to depict the interaction between Agent Category and {K Value, Grid Size} categories, heatmaps were produced and given in Figure 9.



(a)



Figure 9: (a) Agent Category / K Value and (b) Agent Category / Grid Size Heatmaps

The heatmaps indicate that certain combinations achieve the maximum possible score of 188, while other combinations yield values close to this upper bound. The performance score of the environment containing the random agent is significantly lower compared to the other two environments. Specifically, the 2-agent environment consistently attains a score of 188 across all settings where  $K=10, 40$ , whereas the 1-agent environment achieves the maximum score for all settings with  $K=20$ . Similarly, the 2-agent environment reaches the score of 188 for all settings with grid sizes of  $15 \times 15$  and  $20 \times 20$ , while the 1-agent environment attains this score for all settings with a grid size of  $10 \times 10$ . When these results are compared with those presented in Table 2, they demonstrate strong performance. Moreover, these findings support the article's hypothesis that the presence of competition enhances the quality of randomness. Next, failed cases are examined individually. A failed case is defined as tests yielding overall imperfect results (i.e., scores of  $186/188$  or  $187/188$ ) obtained by the MCTS agent-driven setups; random agent environment results were not taken into consideration. Each failed case, along with its corresponding experimental configuration, is presented in Table 10.

MCTS Agent Cnt.	Environ.	Grid Size	Action Point Cnt.	Score	Failed Test
1	95:0.25	20x20	10	187	Random Excursion Variant
1	350:0.55	20x20	10	187	Random Excursion Variant
1	350:0.55	15x15	10	187	Random Excursion Variant
1	351:0.45	15x15	10	187	Overlapping Template
1	382:0.30	10x10	10	187	Linear Complexity
1	95:0.25	15x15	20	187	Universal Statistical Test
1	350:0.55	15x15	40	187	Overlapping Template

1	475:0.60	15x15	40	187	Overlapping Template
2	475:0.60	15x15	20	187	Approximate Entropy
2	29:0.50	10x10	20	186	Random Excursion (2 Times)
2	382:0.30	20x20	40	187	Random Excursion Variant

Table 10: Failed test summaries for 1 and 2 MCTS agent environment setups

The failure cases are primarily for 1 MCTS agent setups and in Random Excursion (or its Variant) Tests. Based on the obtained failure types, it would be wrong to conclude and generalize that Random Excursion (or its Variant) Tests are the most challenging tests for any 2D LCA environments inhabited by MCTS agents. Remember that in Section 5, we pointed out that to prevent generating data with an insufficient number of cycles required by the Random Excursions and Random Excursions Variant tests, different from [Kılıç, 2024], the file append depth ( $d$ ) is taken as  $d = 64$ . The failure cases in the two tests can be attributed to the insufficient number of generated cycles resulting from the introduced gamification process and the limited applicability of the tests. One may expect still better-quality results for alternative file append depth ( $d$ ) values.

The relatively small number of failure results obtained for 2 MCTS agents makes it perform better than a setup with 1 MCTS agent and even the best setup. This can be interpreted as the competition between two players with different goals positively affecting the random generator quality. The statistical findings presented above also support this interpretation, as the results of the ANOVA and subsequent Tukey's HSD test indicate a statistically significant performance difference between the 2-agent configuration and the other setups, with the confidence intervals further reinforcing the observed superiority of this configuration. As in every game, conflict or competition is expected to make the game environment more unpredictable and complex. Many of the results for the proposed 2D LCA MCTS agent-driven environment achieve the highest possible score of 188 or close to it. They give competitive results against high-quality PRNGs, including CA & Langton's Ant [Hosseini et al., 2014], Linear Congruential, Mersenne Twister, and Blum-Blum-Shub from the literature.

Lastly, to provide a more precise representation of the resource and time constraints associated with the proposed approach, memory usage and execution times are evaluated from sample runs. In this analysis, the number of agents is not a relevant factor, as the agents operate sequentially and share identical K Value and grid size constraints. The interaction sequence remains in a fixed {agent, environment, agent, environment} structure regardless of whether an agent acts independently or in opposition to another agent. Consequently, an XZ plane was constructed using the K Value and grid size as its dimensions. For each pair, the average score metric was computed, as the Y axis, and the obtained results are given in Figure 10.

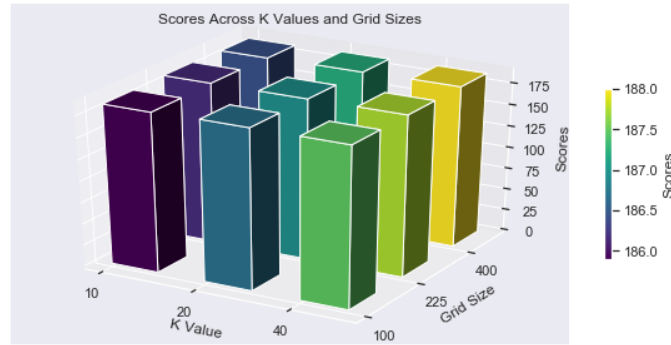


Figure 10: Score across  $\{K \text{ Value}, \text{Grid Size}\}$  configurations

As shown in Figure 10, larger K and grid size values are generally associated with higher scores. Subsequently, the runtime and memory consumption for each configuration were evaluated. To this end, 125,000 random bits were generated in each of 100 independent runs, and the average execution time and memory usage were recorded. The aggregated results are presented in Figure 11.

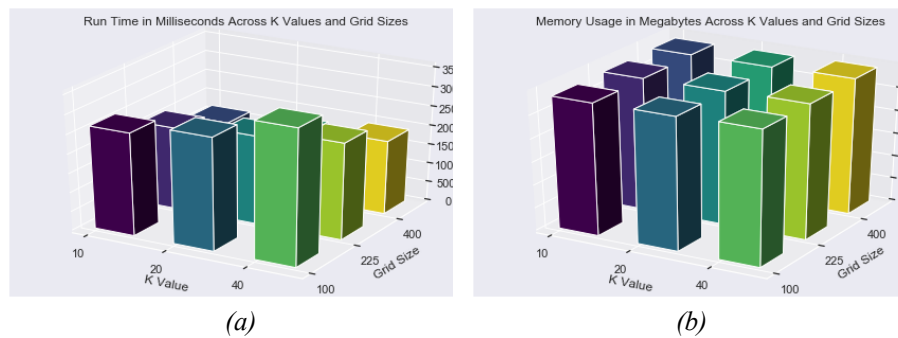


Figure 11: Impact of K Value and Grid Size on (a) Runtime and (b) Memory Usage

As the K Value increases, both memory consumption and runtime are observed to increase, which is expected since each agent must perform up to K actions. However, an unexpected trend is observed when the grid size is increased to  $20 \times 20$ : memory usage increases only slightly, while runtime decreases significantly. This phenomenon may be explained by how agents maintain internal generators with the same grid size configuration as the environment. When the grid size is smaller, these internal states must be updated more frequently, which can lead to less efficient memory access patterns and potentially higher cache utilization overhead. Consequently, larger grid sizes appear to improve runtime efficiency, suggesting a promising direction for optimizing the proposed approach.

However, based on the conducted tests, the proposed method does not appear to be suitable for real-time generation scenarios, as it exhibits a larger memory footprint and longer execution time compared to other established methods. Since optimization was

not the primary objective of this work, no in-depth effort was devoted to it. Nevertheless, the approach has substantial potential for performance improvement due to the inherently parallelizable nature of both Monte Carlo Tree Search (MCTS) and cellular automata.

## 7 Conclusions

As part of our investigation into whether single/multi-agent/environment interactions and goal-driven competitions among them enhance PRNG quality, we proposed a PRNG framework that combines two complex systems: 2D LCA and agents that can alter the CA environment. The environment was gamified to provide agents with specific goals. An original contribution of this study is to introduce a gamified 2D LCA environment and player agents to achieve high-quality PRNGs. To evaluate the randomness quality of the proposed generator, 210 setups were created by sub-permuting a set of parameters. All files are subjected to NIST Statistical Test Suite tests. The results showed that the introduced gamified PRNGs perform competitively or even better than some well-known PRNGs in the literature. It was observed that introducing intelligent MCTS agents competing in the environment improves the quality of the produced random outputs. Although environments with random agents provided acceptable random quality on large grids, this quality decreased significantly as the environment size decreased and the impact of the random agent's actions increased. This was evidence that agents with goals increase the complexity of the environment, hence the random quality, more consistently. Furthermore, configurations with two competing agents mostly yield better results than one-agent setups, suggesting that the complexity introduced by multi-agent interactions positively impacts the generated random quality. Therefore, the original contribution of this study to the literature is to reveal that competition among intelligent MCTS agents with conflicting goals promotes high-quality pseudo-random number generation in the gamified 2D LCA environment.

The proposed framework is flexible, allowing for different experimental trials with varying parameters, such as grid size, agent action/control count, and MCTS iterations, to generate random sequences. While the framework performed exceptionally well under most configurations, further investigations about scalability and optimization of computational resources could enhance its applicability to large-scale or real-time systems. It should be clear that although the quality of the generated random number sequences surpasses that of most state-of-the-art PRNGs, the proposed system falls short in terms of both sequence generation speed and memory requirements. Therefore, it is not possible to use it, for example, for cryptographic purposes that demand high-speed, minimal memory solutions. In the future, more complex agent behaviors, alternative CA configurations, and environmental alterations (like making the environment semi-observable, updating the environment via multiple rules, etc.) could be explored. Furthermore, one can develop a formalized mathematical approach that explores alternative ways of entropy injection via agents as a generalization to the introduced CA-based control game environment as part of future work. Finally, life is nothing but an entertainment and game, so we need to think about the next level, the true, after-life and better to play life neither randomly nor based on generated randomness.

## References

- [Ayubi et.al, 2020] Ayubi, P., Setayeshi, S. & Rahmani, A.M. (2020). Deterministic chaos game: A new fractal based pseudo-random number generator and its cryptographic application, *Journal of Information Security and Applications* 52, 102472, <https://doi.org/10.1016/j.jisa.2020.102472>
- [Bai and Guo, 2024] Bai, J., Guo, X. (2024). The application of chessboard game based on integrated learning and UCT algorithm in mental health and emotional regulation, *Entertainment Computing*, Volume 51, 100722, <https://doi.org/10.1016/j.entcom.2024.100722>
- [Baier and Winands, 2018] Baier, H., & Winands, M. H. M. (2018). MCTS-minimax hybrids with state evaluations, *Journal of Artificial Intelligence Research*, 62, 193–231. <http://dx.doi.org/10.1613/jair.1.11208>.
- [Bassham III et.al., 2010] Bassham III, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Barker, E. B., ... & Vo, S. (2010). Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications.
- [Bhattacharjee and Das, 2022] Bhattacharjee, K., & Das, S. (2022). A search for good pseudo-random number generators: Survey and empirical studies. *Computer Science Review*, 45, 100471.
- [Bhattacharjee et.al., 2023] Bhattacharjee, K., More, N., Singh, S.K., Verma, N., (2023). Cellular automaton-based emulation of the Mersenne twister, *Complex Systems*, 32(2), 139–169.
- [Bhattacharjee et.al., 2017] Bhattacharjee, K., Paul, D., & Das, S. (2017). Pseudo-random number generation using a 3-state cellular automaton. *International Journal of Modern Physics C*, 28(06), 1750078.
- [Bhattacharjee and Vikrant, 2023] Bhattacharjee, K. & Vikrant, V. (2023). Study of First Degree Cellular Automata for Randomness, *Journal of Cellular Automata*, 17(1-2), 47–78.
- [Browne et.al., 2012] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... & Colton, S. (2012). A survey of Monte-Carlo Tree Search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1–43.
- [Cicuttin, et.al., 2023] Cicuttin, A., Micco, L.D., Crespo, M.L., Antonelli, M., Garcia, L., Florian, S.M. & Silva, A. (2023). Looking for suitable rules for true random number generation with asynchronous cellular automata. *Nonlinear Dynamics*, 111, 2711-2722, <https://doi.org/10.1007/s11071-022-07957-8>
- [Coulom, 2006] Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, 72–83, Berlin, Heidelberg: Springer Berlin Heidelberg.
- [Crippa et.al., 2022] Crippa, M., Lanzi, P.L., Marocchi, F. (2022). An analysis of Single-Player Monte Carlo Tree Search performance in Sokoban, *Expert Systems with Applications*, Volume 192, 116224, <https://doi.org/10.1016/j.eswa.2021.116224>
- [Gaeini et.al, 2015] Gaeini, A., Mirghadri, A., & Jandaghi, G. (2015). A General Evaluation Pattern for Pseudo Random Number Generators. *Trends in Applied Sciences Research*, 10(5), 231–244. <https://scialert.net/abstract/?doi=tasr.2015.231.244>
- [Hosseini et.al., 2014] Hosseini, S. M., Karimi, H., & Jahan, M. V. (2014). Generating pseudo-random numbers by combining two systems with complex behaviors. *Journal of Information Security and Applications*, 19(2), 149–162. <https://doi.org/10.1016/j.jisa.2014.01.001>

- [Ihara et.al., 2018] Ihara H., Imai S., Oyama S., Kurihara M. (2018). Implementation and evaluation of information set Monte Carlo tree search for Pokémon IEEE Int. Conf. Syst., Man, Cybern., 2182-2187, 10.1109/SMC.2018.00375
- [Kemmerling et.al, 2024] Kemmerling, M., Lütticke, D. & Schmitt, R. H. (2024). Beyond games: a systematic review of neural Monte Carlo tree search applications. Appl. Intell. 54, 1020–1046 (2024). <https://doi.org/10.1007/s10489-023-05240-w>
- [Keppel and Wickens, 2004] Keppel, G. & Wickens, T. D. (2004). *Design and Analysis: A Researcher's Handbook* (4th ed.). Pearson Prentice Hall
- [Kılıç, 2024] Kılıç, H. (2024). Performance analysis of pseudo-random number generations of two-dimensional linear uniform cellular automata that considers initial state densities, Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi, 39(2), 693–707. <https://doi.org/10.17341/gazimmfd.989265>. (In Turkish)
- [Kim et.al., 2024] Kim, M.J., Lee, D., Kim, J.S. & Ahn, C.W. (2024). Surrogate-assisted Monte Carlo Tree Search for real-time video games, Engineering Applications of Artificial Intelligence, 133, 108152, <https://doi.org/10.1016/j.engappai.2024.108152>
- [Kotoulas et.al., 2006] Kotoulas, L., Tsarouchis, D., Sirakoulis, G. C., & Andreadis, I. (2006). 1-d cellular automaton for pseudorandom number generation and its reconfigurable hardware implementation. In 2006 IEEE International Symposium on Circuits and Systems (ISCAS), 4.
- [Ladyman et.al., 2013] Ladyman, J., Lambert, J., & Wiesner, K. (2013). What is a complex system?. European Journal for Philosophy of Science, 3, 33-67.
- [Li et.al., 2025] Li, H., Pang, X., Sun, B., & Liu, K. (2025). A concise review of intelligent game agent, Entertainment Computing, 52, 100894, <https://doi.org/10.1016/j.entcom.2024.100894>
- [Liu et.al., 2024] Liu, J., Chen, L., Jiang S., Wang, C., Zhang, S., Liang, J., Xiao, Y., Song, R. (2024). A crossword solving system based on Monte Carlo tree search, Artificial Intelligence, Volume 335, 104192, <https://doi.org/10.1016/j.artint.2024.104192>
- [Lorentz, 2016] Lorentz, R. (2016). Using evaluation functions in Monte-Carlo tree search. Theoretical computer science, 644, 106-113.
- [Mariot, 2023] Mariot, L. (2023). Enumeration of maximal cycles generated by orthogonal cellular automata: L. Mariot. Natural Computing, 22(3), 477-491.
- [Lucas et.al., 2019] Lucas, S. M., Dockhorn, A., Volz, V., Bamford, C., Gaina, R. D., Bravi, I., ... & Kruse, R. (2019). A local approach to forward model learning: Results on the game of life game. In 2019 IEEE Conference on Games (CoG), 1-8.
- [Luo and Tan, 2024] Luo, Q., Tan, T.P. (2024). Improved learning efficiency of deep Monte-Carlo for complex imperfect-information card games, Applied Soft Computing, Volume 158, 111545, <https://doi.org/10.1016/j.asoc.2024.111545>.
- [L'Ecuyer, 2012] L'Ecuyer, P. (2012). Random number generation, 35-71. Springer Berlin Heidelberg.
- [McGonigal, 2011] McGonigal, J. (2011). Reality is broken: Why games make us better and how they can change the world. Penguin Press, New York, USA.
- [Mondal et.al., 2019] Mondal, B., Singh, S., & Kumar, P. (2019). A secure image encryption scheme based on cellular automata and chaotic skew tent map. Journal of information security and applications, 45, 117-130.

- [Ostapov et.al., 2023] Ostapov, S., Diakonenko, B., Fylypiuk, M., Hazdiuk, K., Shumyliak, L. & Tarnovetska, O. (2023). Symmetrical Cryptosystems based on Cellular Automata, *International Journal of Computing*, 22(1), 15-20. <https://doi.org/10.47839/ijc.22.1.2874>
- [Ouessai, et.al., 2022] Ouessai, A., Salem, M., & Mora, A.M. (2022). Evolving action pre-selection parameters for MCTS in real-time strategy games, *Entertainment Computing*, 42, 100493, <https://doi.org/10.1016/j.entcom.2022.100493>
- [Packard and Wolfram, 1985] Packard, N. H., & Wolfram, S. (1985). Two-dimensional cellular automata. *Journal of Statistical Physics*, 38(5), 901-946.
- [Poornima et.al., 2024] Poornima, I.G.A., Yogaraja, C.A., Venkatesh, R., Swarna Sudha, M. & Vijayalakshmi, B. (2024). Pseudo Random Number Generator Based on Cellular Automata with Self Organized Criticality, *SN Computer Science* vol. 5, Iss. 5, Article Number 454.
- [Quora, 2025] Quora Link (2025). <https://www.quora.com/What-is-the-purpose-of-the-RNG-in-games-Is-it-simply-a-random-number-generator> (Last visited on 02.01.2025)
- [Roy, 2023] Roy, S. (2023). Fully Asynchronous Cellular Automata as a 'good' Randomness Enhancer, *Journal of Cellular Automata*, 17(1-2), 79-97.
- [Rubio et.al., 2004] Rubio, C. F., Encinas, L. H., White, S. H., del Rey, A. M., & Sánchez, G. R. (2004). The Use of Linear Hybrid Cellular Automata as Pseudo Random Bit Generators in Cryptography. *Neural Parallel Sci. Comput.*, 12(2), 175-192.
- [Russell and Norvig, 2021] Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach*. 4<sup>th</sup> Edition, Pearson.
- [Seaborn and Fels, 2015] Seaborn, K., & Fels, D. I. (2015). Gamification in theory and action: A survey. *International Journal of human-computer studies*, 74, 14–31.
- [Shin et.al., 2012] Shin, S. H., Kim, D. S., & Yoo, K. Y. (2012). A 2-dimensional cellular automata pseudorandom number generator with non-linear neighborhood relationship. In *Networked Digital Technologies: 4th International Conference, NDT 2012, Dubai, UAE, April 24-26, 2012. Proceedings, Part I 4*, 355–368. Springer Berlin Heidelberg.
- [Sirakoulis, 2016] Sirakoulis, G. C. (2016). Parallel Application of Hybrid DNA Cellular Automata for Pseudorandom Number Generation. *Journal of Cellular Automata*, 11(1), 63–89.
- [Sutner, 1989] Sutner, K. (1989). Linear cellular automata and the Garden-of-Eden. *The Mathematical Intelligencer*, 11(2), 49–53.
- [Swiechowski et.al., 2023] Swiechowski M., Godlewski K., Sawicki B., Mańdziuk J. (2023). Monte Carlo tree search: A review of recent modifications and applications, *Artif. Intell. Rev.*, 56(3), 2497-2562, 10.1007/s10462-022-10228-y.
- [Szaban, 2019] Szaban, M. (2019). Pseudorandom number generator based on totalistic cellular automaton. In *Parallel Computing Technologies: 15th International Conference, PaCT 2019, Almaty, Kazakhstan, August 19–23, 2019, Proceedings 15*, 360–370. Springer International Publishing.
- [Temiz et.al., 2014] Temiz, F., Siap, İ., & Akin, H. (2014). On Pseudo Random Bit Generators via Two-Dimensional Hybrid Cellular Automata. *Acta Physica Polonica A*, 125(2).
- [Tomassini et.al., 2000] Tomassini, M., Sipper, M., & Perrenoud, M. (2000). On the generation of high-quality random numbers by two-dimensional cellular automata. *IEEE Transactions on computers*, 49(10), 1146–1151.

[Whitehouse et.al., 2011] Whitehouse D., Powley E.J., Cowling P.I. (2011). Determinization and information set Monte Carlo tree search for the card game Doudizhu, IEEE Conference on Computational Intelligence and Games, 87–94, 10.1109/CIG.2011.6031993.

[Winands et.al., 2008] Winands, M. H., Björnsson, Y., & Saito, J. T. (2008). Monte-Carlo tree search solver. In *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29-October 1, 2008, Proceedings 6*, 25–36. Springer Berlin Heidelberg.

[Wolfram, 1986] Wolfram, S. (1986). Cryptography with cellular automata. In *Advances in Cryptology—CRYPTO’85, Proceedings 5*, 429–432. Springer Berlin Heidelberg.

[Zha et.al., 2021] Zha, D. Xie, J., Ma, W., Zhang, S., Lian, X., Hu, X. & Liu, J. (2021). DouZero: Mastering DouDizhu with Self-Play Deep Reinforcement Learning, *Proceedings of the 38th International Conference on Machine Learning*, 139, 12333–12344.