भारतीय प्रौद्योगिकी संस्थान
काशी हिन्दू विश्वविद्यालय
IIT INDIAN INSTITUTE OF TECHNOLOGY
BANARAS HINDU UNIVERSITY

Statistical Review

of Randomness in RummyCulture application using historical gameplay data

table of contents

Introduction: Randomness in Game Design

Randomness has emerged as a fundamental yet complex element in game design, serving both as a source of uncertainty and engagement. While games have historically incorporated elements of chance through dice rolls and card draws, modern digital games have elevated randomness into a sophisticated design tool that shapes player experience in profound ways. The purpose of this paper is to examine how randomness functions in games, exploring its implementation, benefits, and the growing need for true random number generation.

Games represent a unique medium where uncertainty is not only tolerated but often celebrated - a stark contrast to most other interactive systems where unpredictability is viewed as undesirable. As Costikyan (2013) notes, "Uncertainty is not, in most circumstances, a good thing. We do not wish to be uncertain about whether we can pay our bills, uncertain of the affections of the people who matter to us, uncertain about our health, or uncertain about our job prospects." Yet in games, randomness creates the very uncertainty that makes play engaging and meaningful. (Costikyan, 2013)

The discussion of randomness in games has become increasingly relevant as digital games have evolved. From procedurally generated worlds to loot box mechanics, random elements shape core gameplay experiences. However, implementing randomness effectively requires careful consideration - too much can lead to player frustration, while too little may result in predictable, unengaging experiences. Understanding how to properly balance and implement randomness has become a crucial skill for game designers.

In this introductory section we examine why randomness is essential to game design, explore various implementations across different genres, analyze the benefits it provides to player engagement, and investigate the growing importance of true random number generation testing for game integrity. Through this analysis, we aim to demonstrate that randomness, when thoughtfully applied, serves as a powerful tool for creating compelling and sustainable game experiences that are also responsible.

The essentiality of randomness in good game design

The incorporation of randomness in games serves several essential purposes that contribute to player engagement and game longevity. Understanding these fundamental reasons helps explain why randomness has become an integral part of game design across various genres.

First, randomness creates uncertainty, which is central to maintaining player interest. As Wang (2023) explains, games balance uncertainty with reward - the more certain an option, the lower the reward, while less certain options offer higher potential rewards (Randomness in Games, 2023). This risk-reward dynamic forces players to make meaningful decisions rather than simply following predetermined optimal strategies. Without uncertainty, games can become mechanical exercises in execution rather than engaging experiences that require strategic thinking and adaptation. Therefore, it can be concluded that some randomness in online games of skill such as rummy promotes strategic thinking.

Second, randomness enhances replayability by ensuring that each playthrough offers a unique experience. In games like *Rogue*likes or procedurally generated worlds, random elements create virtually infinite variations of content, extending the game's longevity far beyond what would be possible with purely designed content. This variability keeps players engaged by presenting new challenges and situations, even after multiple playthroughs.

Third, randomness can serve as an effective balancing mechanism. In multiplayer games, random elements can sometimes help level the playing field between players of different skill levels without completely negating the importance of skill. As demonstrated in games like Rummy and Poker, randomness creates situations where strategic thinking and adaptation become more important than pure mechanical execution. Thus, a skilled player may be confronted with a situation where they are required to display exemplary skill to over come an unfavourable hand of cards.

Fourth, randomness adds authenticity to simulations and creates more believable game worlds. In combat systems, for example, introducing an element of uncertainty through random damage variations or critical hits better reflects the unpredictable nature of real conflict. This enhanced realism contributes to player immersion and engagement.

Finally, randomness serves as a crucial tool for creating moments of surprise and excitement. The unpredictability of random events can generate memorable experiences that wouldn't be possible in a completely deterministic system. As Al-Hammadi and Abdelazim note in their research, randomness can enhance cognitive engagement and encourage players to think differently, particularly when implemented thoughtfully in educational games.

However, it's important to note that the implementation of randomness must be carefully balanced. Too much randomness can lead to player frustration if they feel their choices and skills don't meaningfully impact outcomes. Conversely, too little randomness can make games predictable and less engaging. The key lies in finding the right balance for each specific game design and target audience.

Some examples of uses of Randomness in Online Games

The implementation of randomness in games manifests in various sophisticated ways across different genres. By examining specific examples, we can better understand how randomness enhances gameplay and player experience.

Procedural Generation and World Creation Minecraft exemplifies the effective use of randomness in world generation. The game employs random number generators to create unique landscapes, cave systems, and resource distributions each time a new world is generated. This approach not only provides players with endless exploration possibilities but also creates unique challenges and opportunities in each playthrough. Similarly, No Man's Sky uses procedural generation on a massive scale to create entire planets and ecosystems, demonstrating how randomness can be used to generate vast amounts of content that would be impossible to design manually.

Combat Systems Many role-playing games utilize randomness in their combat mechanics to create tension and uncertainty. In games like Fallout: New Vegas, random number generation determines critical hits and damage variations, making each encounter unique and exciting. This implementation of randomness forces players to adapt their strategies based on the outcomes of these random elements, adding depth to the tactical decision-making process.

Loot Systems and Rewards Games like Diablo and World of Warcraft employ sophisticated random loot systems to maintain player engagement. These systems use carefully calibrated probability distributions to determine item drops, creating a compelling reward structure that keeps players invested. The randomness in these systems is typically weighted to ensure that while individual outcomes are unpredictable, the overall experience remains satisfying and progression feels meaningful.

Movement and AI Behavior In competitive games, randomness often plays a crucial role in making artificial intelligence opponents more challenging and unpredictable. As demonstrated in Player Unknown's Battlegrounds, random elements in weapon recoil patterns and AI behavior create more dynamic and engaging combat scenarios. This implementation of randomness prevents players from developing overly reliable strategies and maintains the game's challenge.

Card Games and Deck Building Digital card games like Hearthstone combine traditional card game randomness with digital-specific random elements. Beyond the inherent randomness of card drawing, these games often include cards with random effects that create unique situations each time they are played. This layered approach to randomness creates complex decision-making scenarios where players must constantly adapt their strategies. Games likes rummy and poker have an often-simpler implemntation of randomness.

Educational Applications As highlighted in Al-Hammadi and Abdelazim's research, randomness can be effectively used in educational games to enhance learning outcomes. Their study demonstrated how random elements in educational games can improve attention, memory, and cognitive processing, particularly when combined with structured learning objectives. (Al-Hammadi & Abdelazim, 2015)

Thus, randomness is an essential component to making games interesting and as Al-Hammadi et al have shown utilitarian as well. In this next section, we explore some of the benefits for the use of randomness. (Al-Hammadi & Abdelazim, 2015)

The Positive Benefits of Randomness in Games

The implementation of randomness in games offers several significant benefits that enhance both player experience and game design effectiveness. These advantages extend beyond simple unpredictability to create deeper, more engaging gameplay experiences.

Enhanced Replayability and Longevity

Randomness significantly extends a game's lifespan by ensuring each playthrough offers a unique experience. This benefit is particularly evident in games utilizing procedural generation, where random elements create virtually infinite variations of content. As documented in Costikyan's research, this variability maintains player interest far beyond what would be possible with static, predetermined content (Costikyan, 2013). Players remain engaged because they cannot fully predict what will happen in subsequent playthroughs, encouraging multiple attempts and extended play sessions.

Balanced Challenge and Skill Development

Randomness serves as an effective mechanism for maintaining appropriate challenge levels across different player skill levels. By introducing elements of uncertainty, games can create situations where both novice and experienced players face meaningful challenges. The random elements force players to develop adaptive strategies rather than rely solely on memorized patterns or solutions. This dynamic is particularly valuable in educational games, where Al-Hammadi and Abdelazim's research demonstrates how randomness can enhance cognitive development and learning outcomes (Al-Hammadi & Abdelazim, 2015).

Emotional Engagement and Excitement

The unpredictable nature of random events creates powerful emotional responses in players. When properly implemented, randomness generates moments of surprise, excitement, and triumph that would be impossible in purely deterministic systems. These emotional peaks and valleys contribute significantly to player engagement and satisfaction. The anticipation of potential outcomes, whether in combat scenarios or reward systems, creates a compelling psychological hook that keeps players invested in the game.

Strategic Depth and Decision Making

Contrary to what might be expected, thoughtfully implemented randomness can actually deepen strategic gameplay. As Wang notes, when players must account for multiple possible outcomes, they are forced to develop more robust strategies that can adapt to different situations. This requirement for strategic flexibility creates more engaging decision-making processes than would be possible in completely predictable systems (Randomness in Games, 2023).

Social Interaction and Competition

In multiplayer environments, randomness serves as a social equalizer while maintaining competitive integrity. It creates shared experiences and talking points among players, fostering community engagement and discussion. Additionally, random elements can help prevent dominant strategies from emerging that might otherwise make competitive play stagnant or uninteresting.

Resource Management and Risk Assessment

Random elements create opportunities for meaningful resource management and risk assessment decisions. Players must weigh potential outcomes and decide how to allocate resources effectively, adding depth to gameplay systems. This aspect is particularly evident in strategy games and RPGs, where random events force players to maintain contingency plans and manage resources carefully.

These benefits demonstrate that randomness, when properly implemented, serves as a crucial tool for creating engaging and sustainable game experiences. However, realizing these benefits requires careful consideration of implementation methods and the appropriate balance of random elements within the overall game design.

Need for True Randomness

The growing sophistication of games and increasing concerns about fairness and security have heightened the importance of true random number generation (TRNG) in gaming applications. This section examines why true randomness has become crucial and the limitations of traditional pseudo-random number generators (PRNGs).

Security and Fair Play In modern gaming environments, particularly those involving competitive play or real-money transactions, the integrity of random number generation is paramount. Pseudo-random number generators, which rely on mathematical algorithms and seed values, can be potentially predicted or manipulated if their patterns are discovered. This vulnerability becomes particularly concerning in online gambling applications, competitive games, and systems involving valuable virtual items or cryptocurrencies.

The rise of competitive gaming and esports has further emphasized the need for genuine randomness. Professional players and tournament organizers require assurance that game outcomes are truly unpredictable and cannot be exploited through pattern recognition or mathematical analysis. True random number generation, derived from physical phenomena such as atmospheric noise or quantum events, provides this level of security and fairness.

Statistical Quality and Distribution True random number generators typically provide higher quality randomness compared to their pseudo-random counterparts. As noted in the technical literature, PRNGs can exhibit subtle patterns or biases over large sample sizes, potentially affecting game balance and fairness. True random number generation ensures a more uniform distribution of outcomes, which is crucial for maintaining game balance and player trust.

Certification and Regulatory Requirements The gaming industry increasingly faces regulatory scrutiny, particularly in areas involving monetary transactions or gambling mechanics. Many jurisdictions now require certified random number generators that meet specific statistical standards and security requirements. True random number generation systems more easily satisfy these regulatory requirements and provide the necessary documentation for compliance.

Player Trust and Game Integrity Player confidence in game fairness significantly impacts engagement and retention. When players believe that random elements are truly unpredictable and fair, they are more likely to invest time and resources in the game. True random number generation helps maintain this trust by providing verifiable, unbiased results that cannot be predicted or manipulated.

Technical Implementation Challenges Despite the clear advantages of true random number generation, implementing it in games presents several technical challenges. These include:

- Ensuring consistent availability of random numbers during gameplay
- Managing the computational overhead of accessing physical random number sources
- Maintaining game performance while using external random number services
- Balancing the need for true randomness with practical gaming requirements

The gaming industry continues to develop solutions to these challenges, often combining true and pseudo-random number generation systems to achieve optimal results. This hybrid approach allows games to maintain the benefits of true randomness while addressing practical implementation concerns.

The Role of Uncertainty and Randomness in Online Rummy

Uncertainty is a fundamental aspect of gaming, adding excitement, challenge, and replayability. In both traditional and digital games, the element of chance ensures that no two experiences are the same, forcing players to adapt and strategize rather than follow a fixed approach. This is particularly true for games like **rummy**, where randomness determines card distribution, directly influencing the fairness and unpredictability of each match.

The Role of Random Number Generators (RNGs) in Games

Dice rolls, shuffled cards, and coin flips introduce randomness, ensuring that outcomes cannot be predetermined. Whether it's a board game or an online card game, the presence of uncertainty keeps players engaged, as it prevents a single dominant strategy from guaranteeing victory. As Greg Costikyan (2013) explains, uncertainty is central to the appeal of games because it challenges players to master unpredictable situations while maintaining engagement through novelty and suspense. (Costikyan)

In modern digital games, **random number generators (RNGs)** simulate unpredictability. Since computers operate deterministically, they rely on **pseudorandom number generators (PRNGs)** to produce sequences that appear random. As Baglin (2017) explains, PRNGs use mathematical formulas or precomputed lists to generate numbers, mimicking randomness without truly achieving it. (Baglin) Unlike **true random number generators (TRNGs)**, which extract randomness from physical sources like radioactive decay or atmospheric noise, PRNGs follow an algorithmic structure, making them efficient but theoretically predictable.

However, randomness in games is often a point of scrutiny. Players expect fairness, especially in competitive environments where small advantages can significantly impact results. If a game's randomness is flawed—either due to poor algorithm design or intentional manipulation—it can create **bias** that favors certain players, leading to distrust and dissatisfaction.

Chance and Fairness in Online Rummy

Online rummy blends skill with chance. Players must strategically discard and pick cards, but the initial distribution and subsequent draws are beyond their control. This makes randomness a core component of the game's integrity. A well-designed rummy game ensures that every player has an equal opportunity, relying purely on their skills to form valid sets and sequences. As Grabarczyk (2018) notes, randomness plays a dual role in games: it can enhance replayability by introducing variability while also creating ethical challenges if perceived as unfair or manipulated. (Grabarczyk)

Unlike physical rummy, where players manually shuffle the deck, online rummy platforms rely on RNGs to simulate shuffling. However, if the system is flawed, players might notice patterns, leading to potential exploitation or unfair outcomes. As Baglin (2017) explains, **poorly implemented RNGs can create predictable sequences, which players may recognize and use to gain an unfair advantage**. Worse, if the platform deliberately manipulates randomness to favor specific players or the house, it would destroy player confidence and damage the company's reputation.

For RummyCulture, ensuring fair randomness is **not just a technical requirement but a critical business necessity**. Players need to trust that their chances of winning are not influenced by hidden biases or external manipulation. In the competitive online gaming industry, platforms that fail to establish credibility often lose users to competitors who can guarantee fairness.

RNG Tests and Certification

The increasing importance of random number generation in games has led to the development of rigorous testing and certification procedures. These processes ensure that random number generators meet statistical requirements and maintain game integrity across different implementations.

Standard Testing Procedures Random number generators undergo comprehensive statistical testing to verify their quality and reliability. The primary test suites include NIST (National Institute of Standards and Technology) Statistical Test Suite and Diehard Tests, which evaluate various properties of random number sequences. These tests examine multiple characteristics including:

Distribution uniformity across the output range Independence between successive numbers Absence of detectable patterns or cycles Resistance to prediction based on previous outputs Statistical correlation analysis

Gaming Industry Certification The gaming industry has established specific certification requirements for random number generators, particularly for games involving monetary transactions. Organizations like eCOGRA (eCommerce Online Gaming Regulation and Assurance) and GLI (Gaming Laboratories International) provide independent testing and certification services. These certifications verify that random number generators meet industry standards and regulatory requirements.

For online gaming platforms, certification often requires continuous monitoring and periodic re-testing to ensure ongoing compliance. This process includes examining both the theoretical and actual results of the random number generator implementation, verifying that outcomes align with expected probabilities and maintain fairness over time.

Technical Requirements Modern certification standards address several key technical aspects:

- Implementation Security: iTech Certification

- Scalability Verification: iTech Certification
- Documentation Requirements: iTech Certification
- Regulatory Compliance: British & American Standards

Why RNGs Matter for Player Trust

Trust is the foundation of any online gaming platform, especially those involving real money. If players feel that a game is rigged or that certain outcomes occur more frequently than probability dictates, they are less likely to continue playing. Accusations of unfair play can spread quickly through reviews and online communities, causing lasting damage to a company's reputation.

Research has shown that **randomness in digital environments significantly affects user perception and decision-making**. Al-Hammadi & Abdelazim (2015) found that players who trust the randomness of a game are more engaged and willing to continue playing, while those who perceive bias in outcomes experience frustration and disengagement (Al-Hammadi and Abdelazim). This aligns with Costikyan's (2013) observation that uncertainty must be well-balanced in games to sustain player interest without creating frustration. (Costikyan)

To maintain transparency and fairness, many gaming platforms submit their RNGs for independent testing and certification. Regulatory bodies such as **NIST (National Institute of Standards and Technology)** and **British RTS (Remote Technical Standards)** define strict guidelines for randomness in gaming applications. Compliance with these standards ensures that the system does not exhibit bias, reassuring players that the game is fair.

This report focuses on testing the randomness of **RummyCulture's number generation system** using **dieharder tests**, a widely accepted method for evaluating statistical randomness. By analyzing the data provided by the company, we aim to determine whether the RNG used in their system produces unbiased, unpredictable results. This verification is crucial in ensuring that the game is fair for all players, reinforcing **trust, transparency, and credibility**.

Introduction to testing of randomness

NIST Standard

The **National Institute of Standards and Technology (NIST)** is a U.S. government agency that develops standards and guidelines for various industries, including **cryptography, cybersecurity, and randomness testing**. One of its most important contributions in the field of randomness evaluation is the **NIST Special Publication 800-22**, which provides a **Statistical Test Suite (STS)** to evaluate the quality of random number generators (RNGs).

NIST SP 800-22 provides **15 statistical tests** designed to detect **patterns, biases, and correlations** in an RNG's output. These tests assess whether a number generator produces **truly random and unpredictable sequences**

The British Remote Technical Standards (RTS) standard

According to the British RTS standards random number generation and game results must be 'acceptably random'. Acceptably random here means that it is possible to demonstrate to a high degree of confidence that the output of the RNG, game, lottery and virtual event outcomes are random through, for example, statistical analysis using generally accepted tests and methods of analysis. Adaptive behaviour (that is, a compensated game) is not permitted.

To abide by **Statistical Randomness and Fairness:** The RNG must produce truly random and unbiased sequences, ensuring that all card distributions follow expected probabilities.

Tests - Chi-Square, Kolmogorov-Smirnov, and NIST STS tests

The RNG must be **computationally infeasible** to predict, even with knowledge of past outputs.

The RNG must **not exhibit repeating cycles** over extended periods.

Tests -Berlekamp-Massey Algorithm (Linear Complexity Test), Spectral Test (Fourier Transform) – NIST DFT Test

Any **scaling techniques** used to map RNG outputs to card positions must **preserve randomness** and uniform distribution.

The system must not introduce **bias or compensation mechanisms** that alter game fairness.

Diehard Tests

Diehard is a well-known suite of statistical tests for random number generators (RNGs) developed by George Marsaglia. It consists of multiple tests designed to detect statistical anomalies in RNG outputs by examining different properties, such as uniformity, autocorrelation, and bit-level randomness. The tests include:

1. **Birthdays**
2. **Overlapping 5 Permutations**
3. **32x32 Binary Rank**
4. **6x8 Binary Rank**
5. **Bitstream**
6. **Overlapping Pairs Sparse Occupance (OPSO)**
7. **Overlapping Quadruples Sparse Occupance (OQSO).**
8. **DNA**
9. **Count the 1s (stream)**
10. **Count the 1s (byte)**
11. **Parking Lot**
12. **Minimum Distance (2D Spheres)**
13. **3D Spheres (minimum distance)**
14. **Squeeze**
15. **Sums**
16. **Runs**

Dieharder Tests

Dieharder is an enhanced version of Diehard that fixes several of its shortcomings while preserving its core functionality. The key improvements include:

Adjustable Parameters: Users can modify the number of test samples and repetitions to refine sensitivity.

Kolmogorov-Smirnov (KS) Testing: Instead of relying on a single p-value, Dieharder aggregates multiple p-values using a KS test to improve reliability.

Integration with GNU Scientific Library (GSL): It leverages optimized GSL functions for increased accuracy and performance.

Support for File-Based and In-Memory RNGs: Unlike Diehard, which relied heavily on file-based input, Dieharder allows direct RNG integration, making it suitable for large-scale RNG testing.

Dieharder Tests

1. **Diehard Birthday Test** - Examines the probability of collisions (repeated values) in randomly chosen subsets.
2. **Diehard OPERM5 Test** - Tests random permutations of five elements for uniformity.
3. **Diehard 32x32 Binary Rank Test** - Evaluates the rank distribution of 32x32 binary matrices.
4. **Diehard 6x8 Binary Rank Test** - Similar to dieharder 32x32 binary rank test but for 6x8 binary matrices.
5. **Diehard Bitstream Test** - Analyzes overlapping sequences of bits for randomness.
6. **Diehard OPSO Test** - Checks overlapping pairs of sequences for randomness.
7. **Diehard OQSO Test** - Tests overlapping quadruples of sequences for randomness.
8. **Diehard DNA Test** - Simulates DNA sequences to evaluate randomness in overlapping patterns.
9. **Diehard Count the 1's (stream) Test** - Counts 1s in fixed-length bit streams and checks their distribution.
10. **Diehard Count the 1's Test (byte)** - Similar to the stream test but operates on bytes instead of bit streams.
11. **Diehard Parking Lot Test** - Simulates parking cars in a lot and tests spatial randomness.
12. **Diehard Minimum Distance (2d Circle) Test** - Measures minimum distances between random points in a 2D plane.
13. **Diehard 3d Sphere (Minimum Distance) Test** - Extends the minimum distance test to three dimensions.
14. **Diehard Squeeze Test** - Tests how well random numbers compress into smaller intervals.
15. **Diehard Sums Test** - Evaluates cumulative sums of random numbers for uniformity.
16. **Diehard Runs Test** - Analyzes lengths of runs (consecutive identical bits) in sequences.
17. **Diehard Craps Test** - Simulates craps games to assess randomness in outcomes.

18. **Marsaglia and Tsang GCD Test** - Uses greatest common divisors to test number distributions
19. **STS Monobit Test** - Checks if the number of 1s and 0s in a sequence is balanced.
20. **STS Runs Test** - Evaluates whether runs of identical bits are consistent with randomness.
21. **STS Serial Test (Generalized)** - Analyzes patterns of overlapping subsequences for uniformity.
22. **RGB Bit Distribution Test** - Examines bit-level distributions for uniformity.
23. **RGB Generalized Minimum Distance Test** - Similar to the Diehard minimum distance test but generalized for RGB data.
24. **RGB Permutations Test** - Tests the randomness of permutations in RGB data.
25. **RGB Lagged Sum Test** - Checks correlations between lagged sums of random numbers.
26. **RGB Kolmogorov-Smirnov Test** - Compares distributions using the Kolmogorov-Smirnov statistic.
27. **DAB DCT -** Evaluates randomness through frequency domain analysis.
28. **DAB Fill Tree Test** - Tests randomness by filling a binary tree with random numbers and analyzing structure.
29. **DAB Fill Tree 2 Test** - A variation of the fill tree test with different parameters.
30. **DAB Monobit 2 Test** - A variant of the monobit test, focusing on specific bit-level properties.

How does dieharder evaluate its tests

Dieharder utilizes p - value to evaluate the tests

- p-value close to 0 or 1: Suggests non-randomness (i.e., the RNG fails).
- p-value between 0.0005 and 0.9995: Considered normal behavior.
- p-value distribution uniformity: Dieharder applies a secondary KS test on p-values to verify randomness.

Methodology

About the Data

The RummyCulture team had shared two data files. The files were for a shuffle of 53 and 106 cards

To abide with user privacy and confidentiality the data consisted of 32-bit unsigned integers representing a shuffle of cards from various games. The conversion was made possible with the help of a hash function. Usage of hash function is mandatory and does not compromise randomness detection since randomness detection in itself will be a test for the correctness and functionality of hash function. No Personally Identifiable Information (PII) data or metadata was shared with the team.

Our working

We ran Dieharder battery of tests on the data provided to check if the card shuffling algorithm in online rummy games by RummyCulture acts according to the randomness standards set by NIST and British RTS.

We used the Dieharder test suite with the -a option to run all available randomness tests on the binary files hashed_values_53, hashed_values_106, treating it as raw input (-g 202). This ensured a comprehensive evaluation of the data's randomness.

Results

53 - card games

Type	Number	Percentage
WEAK	**3**	**2.66%**
PASSED	**110**	**97.34%**

P - value range	Observed percentage	Expected percentage
0.0 - 0.1	14.1	10
0.1 - 0.2	10.6	10
0.2 - 0.3	13.2	10
0.3 - 0.4	12.3	10

0.4 - 0.5	12.3	10
0.5 - 0.6	7.07	10
0.6 - 0.7	7.9	10
0.7 - 0.8	7.0	10
0.8 - 0.9	9.7	10
0.9 - 1.0	5.3	10

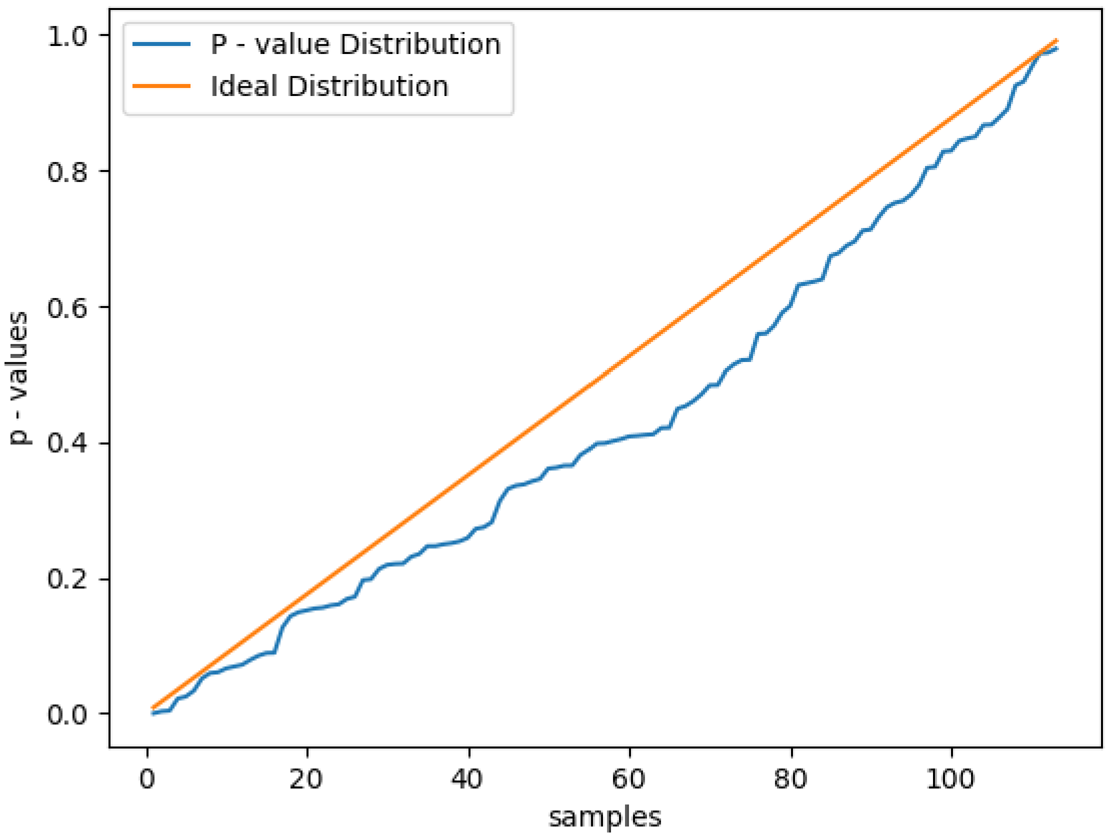

Fig 1: Dieharder Results on 53 card game data

106 - card games

Type	Number	Percentage
WEAK	2	1.76%
PASSED	112	98.24%

P - value range	Observed percentage	Expected percentage
0.0 - 0.1	**6.1**	**10**
0.1 - 0.2	**3.5**	**10**
0.2 - 0.3	**10.5**	**10**
0.3 - 0.4	**14.9**	**10**
0.4 - 0.5	**7.0**	**10**
0.5 - 0.6	**10.5**	**10**
0.6 - 0.7	**14.03**	**10**
0.7 - 0.8	**12.2**	**10**
0.8 - 0.9	**7.8**	**10**
0.9 - 1.0	**13.15**	**10**

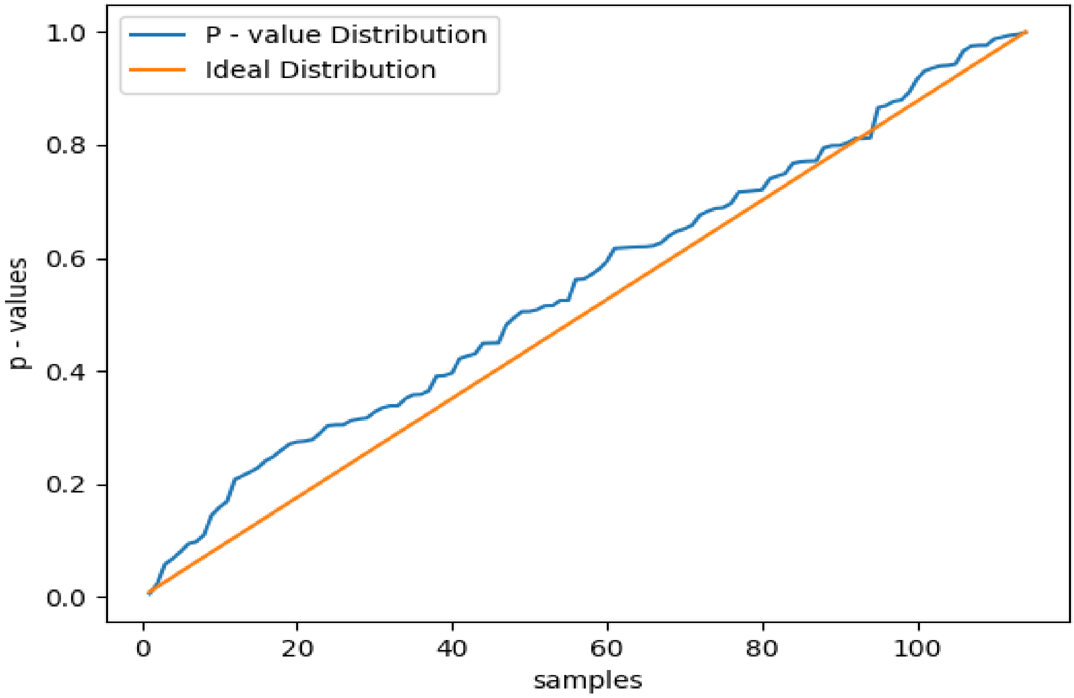

Fig 2: Dieharder Results on 103 Card Game

Conclusion

The Dieharder test results for RummyCulture's random number generator (RNG) demonstrate a high level of compliance with established randomness standards, ensuring fairness and unpredictability in card shuffling for online rummy games.

For 53-card games, the RNG passed **97.34%** of the tests, with only 2.66% classified as "WEAK." The p-value distribution was close to the expected uniform range, though some deviations in specific ranges (e.g., 0.0–0.1 and 0.9–1.0) were observed.

For 106-card games, the RNG performed even better, passing **98.24%** of the tests, with only 1.76% marked as "WEAK." The p-value distribution was more consistent with expectations, indicating strong randomness properties.

The few weak results observed are statistically acceptable given the large number of tests conducted and do not indicate systemic flaws in the RNG. Overall, the RNG meets the randomness requirements outlined by NIST and British RTS standards, ensuring fair gameplay and fostering trust among players.

These results affirm that RummyCulture's RNG implementation is robust and suitable for ensuring fairness in competitive online rummy games. Regular testing and monitoring are recommended to maintain these high standards over time.

Annexure: Complete Dieharder Test Results

53 - card games

Name	n-tup	tsamples	psamples	p-values	Assessment
diehard_birthdays	0	100	100	0.71127	PASSED
diehard operm5	0	1000000	100	0.60023	PASSED
diehard_rank_32x32	0	40000	100	0.27456	PASSED
diehard_rank_6x8	0	100000	100	0.48380	PASSED
diehard_bitstream	0	2097152	100	0.77849	PASSED
diehard_ospo	0	2097152	100	0.15597	PASSED
diehard_osqo	0	2097152	100	0.71305	PASSED
diehard_dna	0	256000	100	0.36559	PASSED
diehard_count_1s_byt	0	256000	100	0.23514	PASSED
diehard_parking_lot	0	12000	100	0.36104	PASSED
diehard_2dsphere	2	8000	100	0.89066	PASSED
diehard_3dsphere	3	4000	100	0.82909	PASSED
diehard_squeeze	0	100000	100	0.97947	PASSED
diehard_sums	0	100	100	0.00302	WEAK
diehard_runs	0	100000	100	0.15923	PASSED
diehard_runs	0	100000	100	0.63089	PASSED

diehard_craps	**0**	**200000**	**100**	**0.40848**	**PASSED**
diehard_craps	**0**	**200000**	**100**	**0.97370**	**PASSED**
marsaglia_tsang_gcd	**0**	**10000000**	**100**	**0.67816**	**PASSED**
marsaglia_tsang_gcd	**0**	**10000000**	**100**	**0.22098**	**PASSED**
sts_monobit	**1**	**100000**	**100**	**0.25880**	**PASSED**
sts_runs	**2**	**100000**	**100**	**0.34585**	**PASSED**
sts_serial	**1**	**100000**	**100**	**0.51415**	**PASSED**
sts_serial	**2**	**100000**	**100**	**0.50445**	**PASSED**
sts_serial	**3**	**100000**	**100**	**0.59022**	**PASSED**
sts_serial	**3**	**100000**	**100**	**0.38102**	**PASSED**
sts_serial	**4**	**100000**	**100**	**0.33593**	**PASSED**
sts_serial	**4**	**100000**	**100**	**0.63975**	**PASSED**
sts_serial	**5**	**100000**	**100**	**0.48422**	**PASSED**
sts_serial	**5**	**100000**	**100**	**0.40454**	**PASSED**
sts_serial	**6**	**100000**	**100**	**0.84723**	**PASSED**
sts_serial	**6**	**100000**	**100**	**0.73129**	**PASSED**
sts_serial	**7**	**100000**	**100**	**0.24664**	**PASSED**
sts_serial	**7**	**100000**	**100**	**0.24673**	**PASSED**

sts_serial	**8**	**100000**	**100**	**0.75521**	**PASSED**
sts_serial	**8**	**100000**	**100**	**0.08495**	**PASSED**
sts_serial	**9**	**100000**	**100**	**0.63356**	**PASSED**
sts_serial	**9**	**100000**	**100**	**0.93122**	**PASSED**
sts_serial	**10**	**100000**	**100**	**0.52039**	**PASSED**
sts_serial	**10**	**100000**	**100**	**0.07879**	**PASSED**
sts_serial	**11**	**100000**	**100**	**0.84374**	**PASSED**
sts_serial	**11**	**100000**	**100**	**0.76422**	**PASSED**
sts_serial	**12**	**100000**	**100**	**0.87863**	**PASSED**
sts_serial	**12**	**100000**	**100**	**0.95302**	**PASSED**
sts_serial	**13**	**100000**	**100**	**0.19788**	**PASSED**
sts_serial	**13**	**100000**	**100**	**0.52094**	**PASSED**
sts_serial	**14**	**100000**	**100**	**0.05145**	**PASSED**
sts_serial	**14**	**100000**	**100**	**0.00008**	**WEAK**
sts_serial	**15**	**100000**	**100**	**0.12666**	**PASSED**
sts_serial	**15**	**100000**	**100**	**0.84990**	**PASSED**
sts_serial	**16**	**100000**	**100**	**0.39848**	**PASSED**

sts_serial	**16**	**100000**	**100**	**0.41074**	**PASSED**
rgb_bitdist	**1**	**100000**	**100**	**0.40164**	**PASSED**
rgb_bitdist	**2**	**100000**	**100**	**0.16872**	**PASSED**
rgb_bitdist	**3**	**100000**	**100**	**0.24940**	**PASSED**
rgb_bitdist	**4**	**100000**	**100**	**0.36544**	**PASSED**
rgb_bitdist	**5**	**100000**	**100**	**0.57108**	**PASSED**
rgb_bitdist	**6**	**100000**	**100**	**0.86695**	**PASSED**
rgb_bitdist	**7**	**100000**	**100**	**0.36248**	**PASSED**
rgb_bitdist	**8**	**100000**	**100**	**0.42133**	**PASSED**
rgb_bitdist	**9**	**100000**	**100**	**0.06069**	**PASSED**
rgb_bitdist	**10**	**100000**	**100**	**0.80548**	**PASSED**
rgb_bitdist	**11**	**100000**	**100**	**0.86764**	**PASSED**
rgb_bitdist	**12**	**100000**	**100**	**0.55887**	**PASSED**
rgb_minimum_distance	**2**	**10000**	**1000**	**0.15473**	**PASSED**
rgb_minimum_distance	**3**	**10000**	**1000**	**0.16087**	**PASSED**
rgb_minimum_distance	**4**	**10000**	**1000**	**0.02476**	**PASSED**
rgb_minimum_distance	**5**	**10000**	**1000**	**0.44925**	**PASSED**

rgb_permutations	**2**	**100000**	**100**	**0.92555**	**PASSED**
rgb_permutations	**3**	**100000**	**100**	**0.97284**	**PASSED**
rgb_permutations	**4**	**100000**	**100**	**0.33102**	**PASSED**
rgb_permutations	**5**	**100000**	**100**	**0.42092**	**PASSED**
rgb_lagged_sum	**0**	**1000000**	**100**	**0.14922**	**PASSED**
rgb_lagged_sum	**1**	**1000000**	**100**	**0.05937**	**PASSED**
rgb_lagged_sum	**2**	**1000000**	**100**	**0.38869**	**PASSED**
rgb_lagged_sum	**3**	**1000000**	**100**	**0.40944**	**PASSED**
rgb_lagged_sum	**4**	**1000000**	**100**	**0.46079**	**PASSED**
rgb_lagged_sum	**5**	**1000000**	**100**	**0.06908**	**PASSED**
rgb_lagged_sum	**6**	**1000000**	**100**	**0.27213**	**PASSED**
rgb_lagged_sum	**7**	**1000000**	**100**	**0.17200**	**PASSED**
rgb_lagged_sum	**8**	**1000000**	**100**	**0.08938**	**PASSED**
rgb_lagged_sum	**9**	**1000000**	**100**	**0.74570**	**PASSED**
rgb_lagged_sum	**10**	**1000000**	**100**	**0.02180**	**PASSED**
rgb_lagged_sum	**11**	**1000000**	**100**	**0.34233**	**PASSED**
rgb_lagged_sum	**12**	**1000000**	**100**	**0.23106**	**PASSED**

rgb_lagged_sum	13	1000000	100	0.39795	PASSED
rgb_lagged_sum	14	1000000	100	0.47070	PASSED
rgb_lagged_sum	15	1000000	100	0.19648	PASSED
rgb_lagged_sum	16	1000000	100	0.63627	PASSED
rgb_lagged_sum	17	1000000	100	0.25105	PASSED
rgb_lagged_sum	18	1000000	100	0.21899	PASSED
rgb_lagged_sum	19	1000000	100	0.22046	PASSED
rgb_lagged_sum	20	1000000	100	0.31329	PASSED
rgb_lagged_sum	21	1000000	100	0.28185	PASSED
rgb_lagged_sum	22	1000000	100	0.06628	PASSED
rgb_lagged_sum	23	1000000	100	0.68905	PASSED
rgb_lagged_sum	24	1000000	100	0.00400	WEAK
rgb_lagged_sum	25	1000000	100	0.07173	PASSED
rgb_lagged_sum	26	1000000	100	0.14340	PASSED
rgb_lagged_sum	27	1000000	100	0.69529	PASSED
rgb_lagged_sum	28	1000000	100	0.41170	PASSED
rgb_lagged_sum	29	1000000	100	0.21303	PASSED

rgb_lagged_sum	**30**	**1000000**	**100**	**0.15190**	**PASSED**
rgb_lagged_sum	**31**	**1000000**	**100**	**0.67393**	**PASSED**
rgb_lagged_sum	**32**	**1000000**	**100**	**0.45333**	**PASSED**
rgb_kstest_test	**0**	**10000**	**1000**	**0.75220**	**PASSED**
dab_bytedistrib	**0**	**51200000**	**1**	**0.25378**	**PASSED**
dab_dct	**256**	**50000**	**1**	**0.33775**	**PASSED**
dab_filltree	**32**	**15000000**	**1**	**0.55969**	**PASSED**
dab_filltree	**32**	**15000000**	**1**	**0.80384**	**PASSED**
dab_filltree2	**0**	**5000000**	**1**	**0.03337**	**PASSED**
dab_filltree2	**1**	**5000000**	**1**	**0.82797**	**PASSED**
dab_monobit2	**12**	**65000000**	**1**	**0.08894**	**PASSED**

106 - card games

Name	**n-tup**	**tsamples**	**psamples**	**p-values**	**Assessment**
diehard_birthdays	**0**	**100**	**100**	**0.73991**	**PASSED**
diehard operm5	**0**	**1000000**	**100**	**0.14403**	**PASSED**
diehard_rank_32x32	**0**	**40000**	**100**	**0.58028**	**PASSED**

diehard_rank_6x8	**0**	**100000**	**100**	**0.51518**	**PASSED**
diehard_bitstream	**0**	**2097152**	**100**	**0.57012**	**PASSED**
diehard_opso	**0**	**2097152**	**100**	**0.97628**	**PASSED**
diehard_oqso	**0**	**2097152**	**100**	**0.21412**	**PASSED**
diehard_dna	**0**	**256000**	**100**	**0.79883**	**PASSED**
diehard_count_1s_str	**0**	**256000**	**100**	**0.67576**	**PASSED**
diehard_count_1s_byt	**0**	**256000**	**100**	**0.25944**	**PASSED**
diehard_parking_lot	**0**	**12000**	**100**	**0.50512**	**PASSED**
diehard_2dsphere	**2**	**8000**	**100**	**0.30280**	**PASSED**
diehard_3dsphere	**3**	**4000**	**100**	**0.91685**	**PASSED**
diehard_squeeze	**0**	**100000**	**100**	**0.74921**	**PASSED**
diehard_sums	**0**	**100**	**100**	**0.27521**	**PASSED**
diehard_runs	**0**	**100000**	**100**	**0.31206**	**PASSED**
diehard_runs	**0**	**100000**	**100**	**0.69633**	**PASSED**
diehard_craps	**0**	**200000**	**100**	**0.77031**	**PASSED**
diehard_craps	**0**	**200000**	**100**	**0.99590**	**WEAK**
marsaglia_tsang_gcd	**0**	**10000000**	**100**	**0.00538**	**PASSED**

marsaglia_tsang_gcd	**0**	**10000000**	**100**	**0.15840**	**PASSED**
sts_monobit	**1**	**100000**	**100**	**0.81131**	**PASSED**
sts_runs	**2**	**100000**	**100**	**0.35062**	**PASSED**
sts_serial	**1**	**100000**	**100**	**0.87704**	**PASSED**
sts_serial	**2**	**100000**	**100**	**0.94324**	**PASSED**
sts_serial	**3**	**100000**	**100**	**0.68757**	**PASSED**
sts_serial	**3**	**100000**	**100**	**0.61976**	**PASSED**
sts_serial	**4**	**100000**	**100**	**0.97492**	**PASSED**
sts_serial	**4**	**100000**	**100**	**0.72040**	**PASSED**
sts_serial	**5**	**100000**	**100**	**0.81247**	**PASSED**
sts_serial	**5**	**100000**	**100**	**0.86958**	**PASSED**
sts_serial	**6**	**100000**	**100**	**0.62149**	**PASSED**
sts_serial	**6**	**100000**	**100**	**0.61784**	**PASSED**
sts_serial	**7**	**100000**	**100**	**0.65129**	**PASSED**
sts_serial	**7**	**100000**	**100**	**0.49377**	**PASSED**
sts_serial	**8**	**100000**	**100**	**0.87959**	**PASSED**
sts_serial	**8**	**100000**	**100**	**0.81134**	**PASSED**

sts_serial	**9**	**100000**	**100**	**0.98787**	**PASSED**
sts_serial	**9**	**100000**	**100**	**0.31654**	**PASSED**
sts_serial	**10**	**100000**	**100**	**0.77163**	**PASSED**
sts_serial	**10**	**100000**	**100**	**0.42149**	**PASSED**
sts_serial	**11**	**100000**	**100**	**0.68218**	**PASSED**
sts_serial	**11**	**100000**	**100**	**0.24101**	**PASSED**
sts_serial	**12**	**100000**	**100**	**0.93082**	**PASSED**
sts_serial	**12**	**100000**	**100**	**0.35812**	**PASSED**
sts_serial	**13**	**100000**	**100**	**0.32683**	**PASSED**
sts_serial	**13**	**100000**	**100**	**0.39066**	**PASSED**
sts_serial	**14**	**100000**	**100**	**0.99079**	**PASSED**
sts_serial	**14**	**100000**	**100**	**0.59396**	**PASSED**
sts_serial	**15**	**100000**	**100**	**0.27392**	**PASSED**
sts_serial	**15**	**100000**	**100**	**0.10949**	**PASSED**
sts_serial	**16**	**100000**	**100**	**0.99922**	**WEAK**
sts_serial	**16**	**100000**	**100**	**0.31470**	**PASSED**
rgb_bitdist	**1**	**100000**	**100**	**0.06763**	**PASSED**

rgb_bitdist	**2**	**100000**	**100**	**0.56152**	**PASSED**
rgb_bitdist	**3**	**100000**	**100**	**0.05762**	**PASSED**
rgb_bitdist	**4**	**100000**	**100**	**0.44904**	**PASSED**
rgb_bitdist	**5**	**100000**	**100**	**0.44852**	**PASSED**
rgb_bitdist	**6**	**100000**	**100**	**0.99408**	**PASSED**
rgb_bitdist	**7**	**100000**	**100**	**0.39604**	**PASSED**
rgb_bitdist	**8**	**100000**	**100**	**0.22858**	**PASSED**
rgb_bitdist	**9**	**100000**	**100**	**0.36418**	**PASSED**
rgb_bitdist	**10**	**100000**	**100**	**0.96669**	**PASSED**
rgb_bitdist	**11**	**100000**	**100**	**0.50487**	**PASSED**
rgb_bitdist	**12**	**100000**	**100**	**0.09387**	**PASSED**
rgb_minimum_distance	**2**	**10000**	**1000**	**0.76720**	**PASSED**
rgb_minimum_distance	**3**	**10000**	**1000**	**0.65762**	**PASSED**
rgb_minimum_distance	**4**	**10000**	**1000**	**0.42636**	**PASSED**
rgb_minimum_distance	**5**	**10000**	**1000**	**0.50826**	**PASSED**
rgb_permutations	**2**	**100000**	**100**	**0.56268**	**PASSED**
rgb_permutations	**3**	**100000**	**100**	**0.93608**	**PASSED**

rgb_permutations	4	100000	100	0.61956	PASSED
rgb_permutations	5	100000	100	0.43000	PASSED
rgb_lagged_sum	0	1000000	100	0.80340	PASSED
rgb_lagged_sum	1	1000000	100	0.86621	PASSED
rgb_lagged_sum	2	1000000	100	0.35738	PASSED
rgb_lagged_sum	3	1000000	100	0.93974	PASSED
rgb_lagged_sum	4	1000000	100	0.71650	PASSED
rgb_lagged_sum	5	1000000	100	0.30435	PASSED
rgb_lagged_sum	6	1000000	100	0.24827	PASSED
rgb_lagged_sum	7	1000000	100	0.02362	PASSED
rgb_lagged_sum	8	1000000	100	0.97646	PASSED
rgb_lagged_sum	9	1000000	100	0.26997	PASSED
rgb_lagged_sum	10	1000000	100	0.22116	PASSED
rgb_lagged_sum	11	1000000	100	0.79494	PASSED
rgb_lagged_sum	12	1000000	100	0.94079	PASSED
rgb_lagged_sum	13	1000000	100	0.51567	PASSED
rgb_lagged_sum	14	1000000	100	0.52459	PASSED

rgb_lagged_sum	**15**	**1000000**	**100**	**0.52459**	**PASSED**
rgb_lagged_sum	**16**	**1000000**	**100**	**0.16835**	**PASSED**
rgb_lagged_sum	**17**	**1000000**	**100**	**0.61698**	**PASSED**
rgb_lagged_sum	**18**	**1000000**	**100**	**0.71766**	**PASSED**
rgb_lagged_sum	**19**	**1000000**	**100**	**0.64670**	**PASSED**
rgb_lagged_sum	**20**	**1000000**	**100**	**0.28996**	**PASSED**
rgb_lagged_sum	**21**	**1000000**	**100**	**0.48077**	**PASSED**
rgb_lagged_sum	**22**	**1000000**	**100**	**0.68865**	**PASSED**
rgb_lagged_sum	**23**	**1000000**	**100**	**0.62678**	**PASSED**
rgb_lagged_sum	**24**	**1000000**	**100**	**0.27797**	**PASSED**
rgb_lagged_sum	**25**	**1000000**	**100**	**0.71916**	**PASSED**
rgb_lagged_sum	**26**	**1000000**	**100**	**0.30443**	**PASSED**
rgb_lagged_sum	**27**	**1000000**	**100**	**0.89297**	**PASSED**
rgb_lagged_sum	**28**	**1000000**	**100**	**0.44933**	**PASSED**
rgb_lagged_sum	**29**	**1000000**	**100**	**0.33797**	**PASSED**
rgb_lagged_sum	**30**	**1000000**	**100**	**0.33797**	**PASSED**
rgb_lagged_sum	**31**	**1000000**	**100**	**0.79908**	**PASSED**

rgb_lagged_sum	**32**	**1000000**	**100**	**0.61884**	**PASSED**
rgb_kstest_test	**0**	**10000**	**1000**	**0.07958**	**PASSED**
dab_bytedistrib	**0**	**51200000**	**1**	**0.33436**	**PASSED**
dab_dct	**256**	**50000**	**1**	**0.74471**	**PASSED**
dab_filltree	**32**	**15000000**	**1**	**0.09707**	**PASSED**
dab_filltree	**32**	**15000000**	**1**	**0.77122**	**PASSED**
dab_filltree2	**0**	**5000000**	**1**	**0.20788**	**PASSED**
dab_filltree2	**1**	**5000000**	**1**	**0.63890**	**PASSED**
dab_monobit2	**12**	**65000000**	**1**	**0.39152**	**PASSED**

References

1. Al-Hammadi, M., & Abdelazim, A. (2015). Randomness impact in digital game-based learning. In *Engineering Education Towards Excellence and Innovation* (p. 6). IEEE.

2. Baglin, S. (2017). *Random Numbers and Gaming.* Retrieved Feb 06, 2025, from ScholarWorks @ SJSU: https://scholarworks.sjsu.edu/art108/7/

3. British RTS. (n.d.). British RTS aim 7.

4. Costikyan, G. (2013). *Uncertainty in Games.* MIT Press.

5. Grabarczyk, P. (2018). From Rogue to Lootboxes: Two Faces of Randomness in Computer Games. In *Philosophy of Computer Games Conference* (p. 10). IT University of Copenhagen. Retrieved from https://www.semanticscholar.org/paper/From-Rogue-to-lootboxes-%3A-two-faces-of-randomness-Grabarczyk/8030eb2c7d660a0c66512e63f308eb0e3b9f880f

6. Randomness in Games. (2023). In W. Wang, *The Structure of Game Design* (p. 10). Springer International Publishing. Retrieved February 6, 2025

For any queries please reach:
Prof Bhaskar Biswas
Indian Institute of Technology (Banaras Hindu University), Varanasi, BHU Campus, Varanasi, Uttar Pradesh, 221005
Phone: 0542-6701808, 2367780, 2368428

भारतीय प्रौद्योगिकी संस्थान
काशी हिन्दू विश्वविद्यालय
INDIAN INSTITUTE OF TECHNOLOGY
BANARAS HINDU UNIVERSITY