# Probabilistic Path Analysis of Genshin Impact's 5 Stars Character Pity System via Markov Chain Graph Modeling

Raya Medina Farrelin - 13525057
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: rayamedif1106@gmail.com, 13525057@std.stei.itb.ac.id

***Abstract*—The Gacha system mechanism on Genshin Impact relies heavily on progressive probability systems known as "Pity System" to regulate virtual asset distribution. This paper presents an analytical solution by framing the game's piecewise linear pity progression as an Absorbing Markov Chain mapped onto a directed weighted graph topology consisting of 91 distinct vertices. By using graph-theoretic path properties, we mathematically analyze the best-case and worst-case probability trajectories. Furthermore, a Monte Carlo simulation with 100,000 computational trials was executed to empirically validate the analytical model, yielding an empirical expected value of 62,939 pulls. The results prove a precise convergence between graph-theoretic bounds and empirical data, showing that the soft-pity zone acts as an aggressive stochastic filter that reduces the probability of reaching the absolute 90th hard-pity limit to near statistical zero.**

***Keywords—Markov chain; Graph Theory; Weighted graph; Genshin Impact; Monte Carlo Simulation***

## I. INTRODUCTION

Genshin Impact is an open-world adventure action role-playing game (RPG) that can be played on multiple platforms, including mobile devices, PCs, and gaming consoles, and developed by miHoYo (HoYoverse). This game has become one of the most successful mobile games ever. By the end of 2022, the game generated over US$4 billion in global revenue. The popularity of Genshin Impact is mainly driven by its free-to-play (F2P) AAA open-world model, combined with high-quality anime aesthetics, deep exploration, and regular expansive updates accessible across multiple platforms.

Genshin Impact has various features that make this game more interesting. These features include action combat, open-world exploration, endgame & dungeons, housing & crafting, and a Gacha System to get a character. Among its key features is a gacha mechanism known as the 'wish' or 'pity' system, which governs the probability of obtaining a 5-star character.

Genshin Impact's pity system doesn't always give players a 50/50 chance of getting a 5-star character. It operates across three probability thresholds: base rate pity, soft pity, and hard pity, alongside a separate 50/50 mechanism. In the base rate pity, each wish carries a 0.6% chance of getting a 5-star character. On the other hand, in soft pity, the probability of getting a 5-star character increases progressively beginning at the 74th wish. While in hard pity, if no 5-star character has been obtained by the 90th wish, one is guaranteed. Additionally, a separate 50/50 mechanism applies, upon obtaining a 5-star character, the player has a 50% chance of it being the featured character on the wish banner. If they lose the 50/50, the player's next 5-star character is guaranteed to be the featured one.

Although this system has been widely analyzed empirically by the player community, there has been no formal modeling that represents it as a mathematical graph structure. This paper proposes modeling the Genshin Impact pity system as a directed weighted graph using the Markov Chain approach, in order to analyze the probabilistic path of obtaining 5-star through a graph-theoretic framework.

## II. THEORETICAL FRAMEWORK

### *A. Pity System in Genshin Impact's Gacha*

Gacha in Genshin Impact uses Intertwined Fate as its currency, which can be obtained directly or purchased using Primogems, the game's official currency. Genshin Impact's gacha system employs a mechanism known as the pity system, which governs the probability of obtaining a 5-star character as a function of the number of consecutive wishes made without one, denoted as the pity counter $n$. This probability is not fixed, rather, it increases progressively across three distinct zones, as defined below.

TABLE I. Three Different Pity Zones

| Pity Zones | Wishes |
|---|---|
| Base rate zone | $0 \leq n \leq 73$ wishes |
| Soft Pity | $74 \leq n \leq 89$ wishes |
| Hard Pity | $n = 90$ wishes |

Each zone has a different probability, which increases progressively with each additional wish. At soft pity, the probability increases by a factor of the base rate multiplied by

10 every pull. Each zone's probability is defined in the piecewise function below.

$$p(n) = \begin{cases} 0.006 & 0 \le n \le 73 \\ 0.006 + 0.06\,(n - 73) & 74 \le n \le 89 \\ 1.0 & n = 90 \end{cases}$$

*Note: The exact pity formula is not officially disclosed by HoYoverse. The approximation used in this paper is derived from large scale community data analysis which author found from: https://library.keqingmains.com/evidence/general-mechanics/gacha*

Alongside the pity system, Genshin Impact employs a mechanism known as the 50/50 system, which determines whether the 5-star character obtained is the featured character or a standard character. Upon obtaining a 5-star character without a guarantee, the player has a probability $q = 0.5$ of receiving the featured character, and a probability of $1\text{-}q = 0.5$ of receiving a standard character instead. Regardless of the outcome, obtaining any 5-star character immediately resets the pity counter to $n = 0$. However, if the player loses the 50/50, which the player receives as a standard character (some players define this condition as 'rate on' conditions), the player enters a guaranteed state, which the next 5-star character obtained is certain to be the featured character.

## B. Graph Theory

A graph, denoted as $G = (V, E)$, consists of a non-empty set of vertices $V$ and a set of edges $E$ that establish connections between pairs of vertices. Depending on its structural properties and characteristics, graphs can be classified into several distinct types, including:

1. Simple & Non-simple Graph

   A simple graph is defined as a graph that contains neither loops (edges connecting a vertex to itself) nor multiple edges between the same pair of vertices. In this type of graph, each edge uniquely connects two distinct nodes. On the other hand, a non-simple graph is defined as a graph that contains loops or multiple edges between the same pair of vertices or both of them.

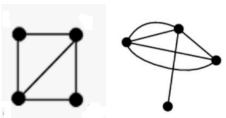

Figure 1. A simple graph (left) and a non-simple graph (right)
Source:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf

2. Directed & Undirected Graph

   A directed graph is defined as a graph where every edge consists of a directional orientation. Contrary, an undirected graph is defined as a graph where every edge has no directional orientation.

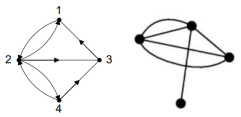


Figure 2. A directed graph (left) and an undirected graph (right)
Source:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf

3. Weighted & Unweighted Graph

   A weighted graph is defined as a graph where every edge consists of assigned value. Contrary, an unweighted graph is defined as a graph where every edge has no assigned value.

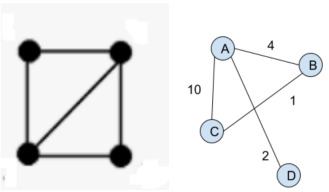


Figure 3. A unweighted graph (left) and a weighted graph (right)
Source:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf

4. Complete Graph

   A complete graph is defined as a simple graph in which every vertex has all edges other nodes.

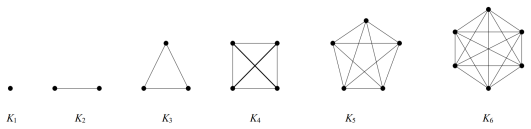


Figure 4. Complete graph
Source:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf

## C. Markov Chain

Markov chains were first introduced by the Russian mathematician Andrey Andreyevich Markov (1856–1922) in 1906. A Markov chain is a stochastic process that undergoes transitions from one state to another within a defined state space. The defining characteristic of a Markov chain is the Markov Property (memorylessness), which states that the probability of the system transitioning into a particular state at time $n$, given its entire history, depends only on its most recent state at time $n-1$, and is independent of all earlier states.. Formally, this property is expressed as:

$$P(X_n = s_n \mid X_0 = s_0, X_1 = s_1, \ldots, X_{n-1} = s_{n-1}) = P(X_n = s_n \mid X_{n-1} = s_{n-1})$$

Where $X_n$ denotes the random variable representing the system's state at time step $n$, and $s_n \in S$ is a specific value in the state space S.

A Markov chain can be naturally represented as a directed weighted graph G = (V, E, W). This graphical representation provides an intuitive framework for analyzing the structural path of stochastic processes. The set of vertices (V) in the graph corresponds directly to the state space of the Markov

chain, where $V = S = \{s_0, s_1, s_2, \ldots, s_n\}$. While a directed edge $e = (s_i, s_j) \in E$ exists iff there is a non-zero probability of transitioning from state $s_i$ to $s_j$ state in a single time step. While an edge weights (W) represent where each directed edge is assigned a weight that is exactly equal to the transition probability $P_{ij}$ from the transition matrix $P$.

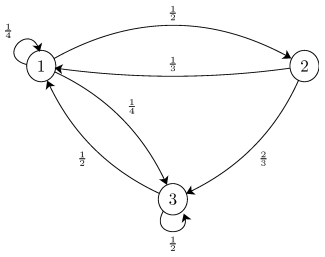


Figure 5. Markov Chain represented as directed weighted graph
Source:https://www.probabilitycourse.com/chapter11/11_2_2_state_transition_matrix_and_diagram.php

Since a graph can be represented in matrix form, a Markov chain, the Markov chain model can also be visualized using a matrix representation. While a standard (unweighted) adjacency matrix only indicates the presence or absence of an edge using binary values, and an incidence matrix relates vertices to edges, neither captures the probabilistic nature required to represent a Markov chain. Instead, a specialized form of the weighted adjacency matrix, known as the transition matrix, is used. The transition probabilities in Markov chain can be organized into an $n \times n$ transition matrix $P$. For a state space with $n$ possible configurations, each entry $P_{ij}$ in the matrix represents the probability of moving from state $s_i$ to $s_j$ state in a single time step. The matrix $P$ must satisfy the stochastic property, where every entry is non-negative $(0 \leq P_{ij} \leq 1)$ and the sum of probabilities in each row is exactly equal to 1.

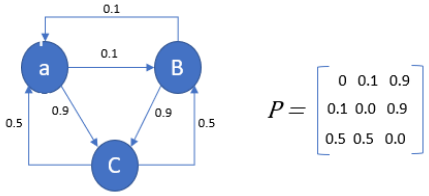


Figure 6. Markov Chain represented as transition matrix
Source:https://medium.com/data-science/markov-chain-analysis-and-simulation-using-python-4507cee0b06e

Beyond single-step matrix structures, a Markov chain can be utilized to evaluate chronological sequences of transitions. Let $X_0, X_1, X_2, X_3, \ldots$ be a markov chain with state space S. The probability distribution of the initial state $X_0$ is known as the initial distribution of the stochastic chain. A trajectory or path represents a finite sequence of sequential states visited by the process over time. Let the finite sequence $s_0, s_1, s_2, \ldots$ denote a structured path of finite length, where each $s_j$ explicitly represents the specific value occupied by the random variable $X_j$. By the Markov property, the probability of this path is:

$$P(X_0 = s_0, X_1 = s_1, X_2 = s_2, \ldots, X_n = s_n) = P(X_0 = s_0)P(X_1 = s_1 | X_0 = s_0) \ldots P(X_n = s_n | X_{n-1} = s_{n-1})$$

The conditional probabilities within this product formulation define the core transition probabilities of the network. For any given state configurations $i$ and $j$, the conditional probability $P(X_{n+1} = j | X_n = i)$ formally specifies the one-step transition probability executed at time-step $n$. Given a fixed starting state, this allows the probability of any specific finite path to be computed directly, forming the mathematical foundation for the path analysis conducted in this paper.

A Markov chain can be classified into several types based on its state properties, one of which is an Absorbing Markov Chain. Absorbing Markov Chain is a Markov Chain which every state can reach an absorbing state. An absorbing state is a state that is impossible to leave once it is entered. While, transient state is a state that the system will eventually leave and never return to with absolute certainty over time. While, recurrent State is a state that can be left, but the system is guaranteed to return to it eventually.

### *D. Monte Carlo Simulation*

Monte Carlo simulation is a type of computational algorithm that uses repeated random sampling to obtain the probability of various possible outcomes. The foundation of Monte Carlo simulation is experiments on probabilistic elements using random sampling. If a system contains elements that exhibit chance in their behavior, then Monte Carlo simulation may be applicable.

## III. PROBABILITISTIC PATH MODELING

### *A. State Space Mapping*

The first thing required to model the probabilistic path of the gacha pity system in Genshin Impact is to define the state space. For this model, the states are categorized into three different types; transient base-rate states, transient soft pity states, and absorbing success states.

Transient base-rate states is a set of states or vertices *(V)* where the system maintains a constant baseline probability of success, spanning from the initial pull up to the safety threshold prior to the probability inflection zone. It starts where the pity *(n)* is between 0 - 73. While Transient soft pity states is a set of states or vertices *(V)* where the transition probability shifts dynamically, increasing linearly at each consecutive step to reflect the game's pseudo-random acceleration mechanic. It starts where the pity (n) is between 74 - 89. While an absorbing success state is a single state representing the successful event of obtaining a 5-star character, which serves as the absolute absorption point where the stochastic process effectively concludes and resets. It starts where the pity *(n)* is equal to 90.

By summing these defined states, the graph yields a clean, linear topology consisting of |V| = 90 transient states + 1

absorbing states = 91 total vertices. This structured environment provides a simplified yet mathematically rigorous framework for conducting path probability propagation and analytical calculations.

### B. *Directed Edges and Probability Weights*

Once the linear state space is defined, the probabilistic paths of the gacha system are established by introducing directed edges $E$ to connect the vertices. Each directed edges $e = (s_i, s_j) \in E$ is assigned with a specific probability weight, which dictates the stochastic movement of a player upon making a single wish.

For every state $n \in \{0, 1, 2, .., 89\}$, has two outgoing edges. Here we define the absorbing success rate as W for win . This corresponds to the two possible outcomes of wish, success or failure. A directed edge $(n,W)$ connects state $n$ to the absorbing win state W, with weight equal to $p(n)$. $p(n)$ defined as a piecewise function to calculate the probability:

$$p(n) = \begin{cases} 0.006 & 0 \le n \le 73 \\ 0.006 + 0.06\,(n - 73) & 74 \le n \le 89 \\ 1.0 & n = 90 \end{cases}$$

Conversely, a directed edge $(n,n+1)$ connects state $n$ to the next state, with weight equal to 1 - $p(n)$, representing failing to obtain a 5-star character and advancing the pity counter by one. So, the complete formula is:

$$w(n, W) = p(nP,\ w(n, n + 1) = 1 - p(n),\ \forall n\{0, 1, \ldots 89$$

Then, when reaching the state W, the pity counter (n) resets, and the chain transitions back to state 0 with weight 1, completing one full wish cycle, $w(W, 0) = 1$. This edge structure ensures that the stochastic propensity of the transition matrix is preserved, since the outgoing weights from every state sum to exactly one :

$$w(n, W) + w(n, n + 1) = p(n) + (1 - p(n)) = 1$$

### C. *Graph Visualization and Python Simulation*

To visualize the Markov chain topology, a Python-based implementation is utilized. However, because the state space is large (|V| = 91), rendering every node simultaneously would cause visual clutter. To maintain clarity, the graph visualization selectively displays only key pivotal states,such as the base rate, the soft-pity state, and the hard-pity state (n=90), alongside the absorbing success node. This simplified presentation ensures that the macro-flow and structural connectivity of the directed graph remain clear and interpretable.

To implement the mathematical abstractions of the 91 transient states and the single absorbing state formulated in the previous sections, the system model is computationally initialized using Python's networkx library.

To maintain the clarity of the graph visualization, the author only used some of the pity nodes, which is pity 0, pity 1, and pity 73 as transient base-rate states, pity 74, and pity 89, as transient soft pity states, and Win as an absorbing success state. While the other pity is represented by 'NextbasePity' for pity between 2 - 72 and 'NextsoftPity' for pity between 75-88.

The complete programmatic implementation used to construct, constrain, and render this specialized topology is detailed below:

```python
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGraph()

def get_weightprob(n):
    if 0 <= n <= 73:
        return 0.006
    elif 74 <= n <= 89:
        return 0.006 + 0.06 * (n-73)
    elif n == 90:
        return 1.0
    return 0.0

# karena tidak menggunakan semua pity states, diberikan lambang 'NextbasePity' yang menunjukkan pity 2-72
# dan 'NextsoftPity' untuk pity 75-88
pity_nodes = ["Pity 0", "Pity 1","NextbasePity", "Pity 73", "Pity 74", "NextsoftPity", "Pity 89", "Pity 90", "Win" ]

G.add_nodes_from(pity_nodes)

failure_edgeweight = [("Pity 0", "Pity 1", f"{1 - get_weightprob(0):.3f}"),("Pity 1", "NextbasePity", f"{1 - get_weightprob(1):.3f}"),
                      ("NextbasePity","Pity 73", "1-P(n)"), ("Pity 73", "Pity 74", f"{1 - get_weightprob(73):.3f}"),
                      ("Pity 74", "NextsoftPity", f"{1 - get_weightprob(74):.3f}"),("NextsoftPity","Pity 89", "1-P(n)"),
                      ("Pity 89", "Pity 90", f"{1 - get_weightprob(89):.3f}")]

success_edgeweight = [
    ("Pity 0", "Win", f"{get_weightprob(0):.3f}"),
    ("Pity 1", "Win", f"{get_weightprob(1):.3f}"),
    ("NextbasePity", "Win", "P(n)"),
    ("Pity 73", "Win", f"{get_weightprob(73):.3f}"),
    ("Pity 74", "Win", f"{get_weightprob(74):.3f}"),
    ("NextsoftPity", "Win", "P(n)"),
    ("Pity 89", "Win", f"{get_weightprob(89):.3f}"),
    ("Pity 90", "Win", f"{get_weightprob(90):.3f}")
]

for u,v,w in failure_edgeweight:
    G.add_edge(u,v, label=w)
```

```python
for u,v,w in success_edgeweight:
    G.add_edge(u,v, label=w)

# Pengaturan posisi
position = {
    "Pity 0":  (0, 2),
    "Pity 1":  (1, 2),
    "NextbasePity":     (2, 2),
    "Pity 73": (3, 2),
    "Pity 74": (4, 2),
    "NextsoftPity":     (5, 2),
    "Pity 89": (6, 2),
    "Pity 90": (7, 2),
    "Win": (3, 0)
}

plt.figure(figsize=(12,6))

nx.draw_networkx_nodes(G, position, nodelist=["Win"], node_color="gold", node_size=1400, node_shape="s")
nx.draw_networkx_nodes(G, position, nodelist=[n for n in pity_nodes if n != "Win"], node_color="skyblue", node_size=1000)
nx.draw_networkx_labels(G, position, font_size=9, font_weight="bold")

nx.draw_networkx_edges(
    G, position, edgelist=[(u,v) for u,v,_ in success_edgeweight],
    edge_color="forestgreen", style="dashed", arrowsize=15,
    connectionstyle="arc3,rad=0.08"
)

linear_labels = {(u, v): d['label'] for u, v, d in G.edges(data=True) if v != "Win"}
nx.draw_networkx_edge_labels(G, position, edge_labels=linear_labels, font_size=8, font_color="black", label_pos=0.5)

success_labels = {(u, v): d['label'] for u, v, d in G.edges(data=True) if v == "Win"}
nx.draw_networkx_edge_labels(G, position, edge_labels=success_labels, font_size=8, font_color="darkgreen", label_pos=0.3)

plt.title("Simplified Markov Chain Graph Topology with Probability Weights", fontsize=12, fontweight="bold")
plt.axis("off")
plt.tight_layout()

plt.savefig("gacha_graph_with_weights.png", dpi=300)
plt.show()
```

Figure 6,7. Python code to visualize Pity Graph
Source:Author

The output of the code shows an integrated graphical architecture that structurally transforms the abstract pity system mechanics into a visible Markovian chain.

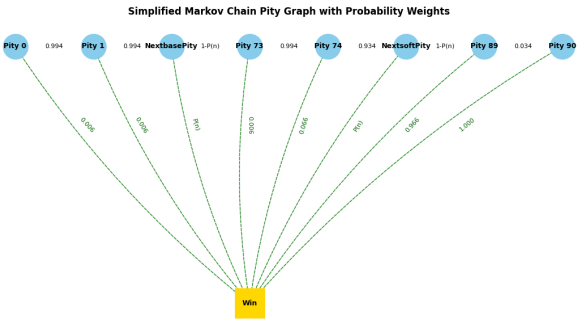


Figure 8. Markov Chain Pity Graph output of the python code
Source:Author

As represented in the compiled graph output, the "Win" node serves as an absorbing state. To ensure numerical precision without cluttering the structural edges, empirical probabilities are formatted strictly to three decimal places (:.3f).

## IV. RESULTS AND DISCUSSION

### *A. Manual Analysis for Best and Worst Case Scenarios for Gacha*

After conducting a visualization of the pity probability with a markov graph, we can calculate the best and worst case probabilities to get a 5-star character on Genshin Impact's Gacha. The calculation will be divided into three categories.

1. Base Rate Calculation
   In the base rate conditions ($0 \leq n \leq 73$ wishes), the probability to get a 5-star character will always be 0.6% (0.006). So the best case scenario for this condition is when a player gets a 5-star character on the first pull. Define formulated :
   $$P(BestCase) = P(1) = 0.006$$
   On the other hand, the worst case scenario for this condition is when a player doesn't get a 5-star character until the 73rd pull. Manually calculated as :
   $$P(WorstCase) = 0.994 \times \ldots \times 0.994 = 0.994^{73}$$

2. Soft Pity Calculation
   In the soft pity conditions ($74 \leq n \leq 89$ wishes), the probability to get a 5-star character increases as the pull increases. So the best case scenario for this condition is when a player gets a 5-star character on the 74th pull (right after entering soft pity zone), Manually calculates as:
   $$P(SBest) = P(74) = 0.006 + 0.06(74 - 73) = 0.066$$
   On the other hand, the worst case scenario for this condition is when a player doesn't get a 5-star character until the 89th pull. Because the probability is different we need to calculate each of the probability first. Manually formulated and calculated as :
   $$P(SWorst) = \prod_{n=74}^{89} (1 - p(n)) \approx 1.8949 \times 10^{-8}$$
   *Note: p(n) refers to probability calculation for soft pity defined in chapter II.*

3. Hard Pity (Pity = 90) Calculation
   In the hard pity condition, a player is guaranteed to get a 5-star character. So the probability to get a 5-star character in a hard pity condition is 1.

After we got all the calculations, now we can combine them into best and worst case scenarios. For the best scenarios is when a player gets a 5-star character on the first pull.
$$P(BestCase) = P(1) = 0.006$$
On the other hand, the worst case scenario is when a player gets a 5-star character on the 90th pull.
$$P(Worst) = 0.994^{73} \times 1.8949 \times 10^{-8} \approx 1.2195 \times 10^{-8}$$

Expressed in a percentage format, this probability is equal to approximately 0.000000012195%. This extremely small percentage shows that although the 90th pull is the maximum boundary of the system, the chance of a player being that unlucky is practically almost zero. This mathematical finding explains why players rarely reach the 90th pull in real life.

### *B. Monte Carlo Simulation Experiment Program*

To prove the manual analysis for probability path on best and worst case for gacha at Genshin Impact, the author created a simulation experiment on python. In this experiment a Monte Carlo simulation is executed with 100,000 trials to programmatically test the gacha system and empirically validate the theoretical path probabilities derived from the Markov chain graph model.

```python
import random

def probability(n):
    #mendapatkan probabilitas dapat 5-star character pada pull n
    if 0 <= n <= 73:
        return 0.006
    elif 74 <= n <= 89:
        return 0.006 + 0.06 * (n-73)
    elif n == 90:
        return 1.0

def simulate():
    #mensimulasikan cycle pulls sampai dapat 5-star character
    n = 0 #pity start dari 0
    pulls = 0
    while True:
        pulls +=1
        if random.random() < probability(n):
            return pulls
        n +=1

def run_simulation(trials=100000):
    results = [simulate() for i in range(trials)]
    expected_value = sum(results)/len(results)
    worst_case = max(results)
    best_case = min(results)
    return expected_value, worst_case, best_case, results

random.seed(42)
E, worst, best, results = run_simulation(100000)
print(f"Expected value (rata-rata pull): {E:.3f}")
print(f"Worst case (maksimum pull): {worst}")
print(f"Best case (minimum pull): {best}")

for k in [50, 60, 70, 74, 80, 90]:
    count = sum(1 for r in results if r <= k)
    print(f"P(win dalam <= {k} pull) = {count/len(results)*100:.2f}%")
```

Figure 9. Monte Carlo Pulls Simulation in 100.000 trials
Source:Author

To execute the empirical validation, the Monte Carlo simulation code is structured into three main computational phases directly corresponding to the program's functions:

1. Function Probability()

   This part of the code shows the probability to get a 5-star character for every pity condition ($n$) as defined in the piecewise function on chapter II.

2. Function Simulate()

   This part of the code shows the pull simulation. It initializes the player's current pity counter n at 0 and a total pulls counter at 0. A while True loop continuously increments the pull counter by 1. In each iteration, a pseudo-random floating-point number between 0.0 and 1.0 generated via random.random() is compared against the threshold value from *probability(n)*. If the random float falls below the returned probability, the loop terminates and returns the total pulls required to reach the absorbing state; otherwise, the pity state n increments by 1, and the cycle continues.

3. Function run_simulation()

   This part of the code executes the batch experiment by leveraging list comprehension to gather the simulation results of 100,000 independent players into the results array.

Then the output of the code are:

```
Expected value (rata-rata pull): 62.939
Worst case (maksimum pull): 89
Best case (minimum pull): 1
P(win dalam <= 50 pull) = 26.06%
P(win dalam <= 60 pull) = 30.24%
P(win dalam <= 70 pull) = 34.22%
P(win dalam <= 74 pull) = 35.74%
P(win dalam <= 80 pull) = 85.84%
P(win dalam <= 90 pull) = 100.00%
```

Figure 10. Monte Carlo Pulls Simulation in 100.000 trials Output
Source:Author

From the output of the codes, we know that the manual analysis for the best and worst case scenarios can be applied. The output of the code shows that in the 100.000 trials (suppose that 100.000 trials means 100.000 different players) the best case scenario is 1, where a player gets 5-star character on the first pull. While the worst case scenario is 89, where a player gets a 5-star character on their 89th pull. This shows that the probability of a player getting their 5-star character on the 90th pull is very small, is true.

## *V. CONCLUSION*

This paper demonstrates the application of directed weighted graphs and Absorbing Markov Chains to evaluate the mathematical limits and probability path of Genshin Impact's pity system.

Through structured state-space mapping, the path calculations reveal that while a 90-pull hard-pity boundary is structurally guaranteed, the probability of a player hitting this exact limit is infinitesimally small ($\approx 0.000000012195\%$). This rarity is due to the aggressive filtering effect within the soft-pity zone ($74 \leq n \leq 89$).

This theoretical model is strongly validated by the Monte Carlo simulation of 100,000 trials, which yielded a clear statistical convergence with an empirical expected value of 62.939 pulls and a practical maximum pull (worst case) at 89 pulls. Ultimately, the integration of graph modeling and programmatic validation proves that complex game alignment structures can be rigorously quantified and evaluated using discrete mathematics.

## APPENDIX

Github Repository for Python code used for the experiment:

https://github.com/rmdifmedia/IF1220_Makalah-Matdis

## ACKNOWLEDGMENT

This author would like to express sincere gratitude to Allah SWT for His blessings, mercy, and guidance, which have enabled the completion of this paper. Without His endless grace, this work would not have been possible. The author would also like to express my deepest gratitude to Mr. Dr.

Rinaldi Munir, M.T., the lecturer of IF1220 Discrete Mathematics course, for providing valuable guidance, insightful lectures, and continuous encouragement throughout the semester. His expertise and dedication in teaching have profoundly inspired the author and greatly contributed to the development of this paper.

## REFERENCES

[1] R. Munir. 2026. "Graf (Bagian 1)." https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf. Accessed: Jun. 17, 2026.

[2] KeqingMains. 2026. "Genshin Impact General Mechanics: Gacha." https://library.keqingmains.com/evidence/general-mechanics/gacha. Accessed: Jun. 17, 2026.

[3] Game8 Genshin Team. 2026. "Genshin Impact Drop Rates and Pity System Explained." https://game8.co/games/Genshin-Impact/archives/305937. Accessed: Jun. 17, 2026.

[4] H. Pishro-Nik. 2026. "Introduction to Probabilities, Statistics, and Random Processes: State Transition Matrix and Diagram." https://www.probabilitycourse.com/chapter11/11_2_2_state_transition_matrix_and_diagram.php. Accessed: Jun. 18, 2026.

[5] Data140 Course Team. 2026. "Transitions and Markov Chains," in *Probability for Data Science*. https://data140.org/textbook/content/chapter-10/transitions/. Accessed: Jun. 18, 2026.

[6] HoYoverse. 2026. "Genshin Impact Wish System Mechanics and Official Pity Rates." https://www.hoyolab.com/article/169531. Accessed: Jun. 18, 2026.

[7] Binus Business School. 2017. "Simulasi Monte Carlo dalam Analisis Manajemen Risiko." https://bbs.binus.ac.id/management/2017/12/simulasi-monte-carlo/. Accessed: Jun. 19, 2026.

## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Raya Medina Farrelin
13525057