Application of Combinatorics and Probability Theory in Modeling the Simulation of Enchantment Table in Minecraft

Diandra Aria Yufana - 13525113
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: diandra.aria@gmail.com , 13525113@std.stei.itb.ac.id

***Abstract*—This paper presents the application of combinatorics and probability theory in modeling and simulating the mechanics of the Enchantment Table item in Minecraft. The Minecraft's enchantment system is a random process that the resulting enchantments depend on several factors, including the player's experience level, the type of item being enchanted, enchantment weights, enchantment compatibility, and randomly generated enchantment levels. To model this system's process, a simulation based on the original Minecraft enchantment algorithm is developed using Python. Combinatorial anaylsis is used ot study the possible enchantment combinations, while probability theory is applied to analyze the chance of obtaining specific enchantments. In addition, this simulation design will use Monte Carlo simulation through repeated enchantment trials to observe the distribution of outcomes. This study demonstrates how concepts from discrete mathematics' subtopic, specifically combinatorics, can be applied to model and analyze a complex random system in a digital game.**

Keywords— Combinatorics, Probability Theory, Monte Carlo Simulation, Minecraft, Enchantment Table, Discrete Mathematics

I. Introduction

Randomized systems are widely used in modern digital games to create variety, uncertainty, and unique player experiences. Many game mechanics rely on random processes to determine rewards, item attributes, or character abilites. These systems can often be analyzed using mathematical concepts, particularly combinatorics and probability theory. Studying such systems provides an opportunity to explore practical application of discrete mathematics in digital environments.

One example of a randomized game mechanic can be found in Minecraft through the Enchantment Table item. This feature allows players to enhance tools, weapons, and armor with special enchantments that improve their functionality. The enchantments obtained are not chosen directly by the player, instead, they are generated through a process influenced by some factors such as the player's experience level, the type of item being enchanted, enchantment weights, enchantment compatibility, and randomly generated enchantment levels.

The enchantment system naturally involves combinatorial concepts because a single item can receive multiple enchantments from a set of available options. Different enchantments can be combined to produce a large number of possible outcomes on a single item, while some enchantments cannot appear together due to compatibility restrictions. In Minecraft, enchantments that cannot be applied to the same item together are called "mutually exclusive." As a result, combinatorics can be used to study the number of valid enchantment combinations that may occur during the enchanting process.

Probability theory also plays an important role in modeling the enchantment system. Since enchantments are selected through weighted random processes, some enchantments have higher chance to appear than others. By analyzing these probabilities, it's possible to estimate the chances of obtaining specific enchantment or enchantment combinations. Repeated simulations can be done with the Monte Carlo method to observe the distribution of enchantment outcomes and compare the results with theoretical expectations.

This paper presents a simulation of Minecraft's enchantment table mechanics based on combinatorial and probability principles. The simulation is implemented in Python and follows the original enchantment selection process as close as possible. Through combinatorial analysis, probability calculations, and Monte Carlo simulation, this study aims to demonstrate how concepts from discrete mathematics can be applied to model and analyze a complex randomized system in a digital game.

II. Theoritical Framework

A. Combinatorics

Combinatorics is a branch of mathematics that focused on counting the number of object arrangements without having to enumerate all possible configurations [1]. One of the most common concepts in combinatorics is combinations, which used when the order of selection does not matter. The number of ways to choose (r) objects from a set of (n) objects is given by:

$$C(n,r) = \frac{n!}{r!\,(n-r)!}$$

where (n!) is the factorial of (n).

In the context of Minecraft enchantments, combiatorics can be used to analyze the possible enchantment sets that may be applied to an item. Since multiple enchantments may appear on the same item, the total number of valid enchantment combinations depends on the number of available enchantments and their compatibility.

B. *Probability Theory*

Probability theory is a branch of mathematics concerned with the analysis of random phenomena [3]. Probability theory provides mathematical tools for analyzing uncertain events. The probability of an event (E) is defined as:

$$P(E) = \frac{Number\ of\ favorable\ outcomes}{Total\ number\ of\ possible\ outcomes}$$

where ($0 \leq P(E) \leq 1$).

Probability is used to model the chance of obtaining particular enchantments during the enchanting process. Because enchantments are selected through random procedures, each enchantment has a certain probability of appearing. The probability distribution of enchantments can be analyzed theoretically and compared with simulation results.

C. *Weighted Random Selection*

Weighted random selection is an algoritmic technique used to pick an item from a list where some items have a higher chance of being chosen than others [4]. In many randomized systems, not all outcomes have the same probability of being selected. Instead, each outcome is assigned a "weight" that correspond to their probability of selection.

Suppose there are (n) possible outcomes with weights (w_1, w_2, …, w_n). The probability of selecting outcome (i) is:

$$P(i) = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

A larger weight corresponds to a higher probability of selection.

Minecraft uses weighted random selection when choosing enchantments. Different enchantment are assigned different weights, causing some enchantments to appear more frequently than others. Therefore, weighted probability is an essential component of the enchantment generation process.

D. *Monte Carlo Simulation*

Monte Carlo simulation is a mathematical modeling technique that estimates the probability of different outcomes by using repeated random sampling [5]. Instead of calculating probabilites analytically, the system is executed many times and the outcomes are recorded to get the behavior of a probabilistic system.

If an event (E) occurs (k) times during (N) simulation trials, its estimated probability can be calculated as:

$$\hat{P}(E) = \frac{k}{N}$$

As the number of trials increases, the estimated probability tends to approach the true probability of the event.

In this study, Monte Carlo simulation is used to repeatedly execute the enchantment algorithm and observe the frequency of different enchantments and enchantment combinations. The resulting distributions provide a statistical view of the enchantment system's behavior.

E. *Minecraft Enchantment Mechanics*

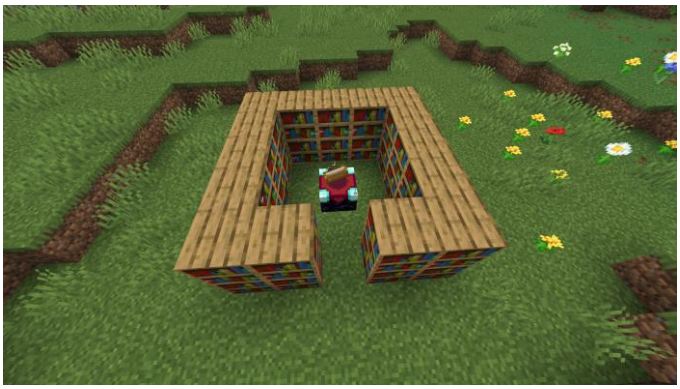


Fig 2.E.1 Minecraft Enchantment Setup.
(Source: Author's Archive)

The Enchantment Table is a game mechanic in Minecraft that allows player to apply enchantment to tools, weapons, armor, and other items. The enchantments generated depend on several factors, including the player's experience level, the enchantaibility of the selected item, the number of bookshelves surrounding the Enchantment Table, enchantment weights, and compatibility between enchantments.

The enchanting process begins by determining an effective enchantment level based on the player's selected enchantment option and item properties. Using this level, the game generates a list of eligible enchantments that satisfy specific level requirements. One enchantment is then selected using a weighted random process. Additional enchantments may be added through repeated random selections, while incompatible (or mutually exclusive) enchantments are removed from the candidate pool.

Fig 2.E.2 Minecraft Enchanting Process.

(Source: Author's Archive)

Because the enchantment system involves random selection, weighted probabilites, and multiple possible enchantment combinations, it provides an appropriate case study for applying combinatorics, probability theory, and Monte Carlo simulation.

III. Simulation Design and Implementation

A. Enchantment Data Collection

The simulation requires a dataset containing information about Minecraft enchantments and their properties. The data used in this study were collected from the Minecraft Wiki and organized into a structured JSON file for processing by the simulation program.

For each enchantment, several attributes were recorded, including the applicable item category, enchantment weight, maximum enchantment level, and enchantment compatibility. These attributes determine the probability of selection and the validity of enchantment combinations during the simulation process.

The collected data were stored in a JSON dataset to facilitate efficient access and manipulation with the Python implementation. This dataset serves as the primary source of information for all simulation trials.

```
{
  "item_type": "Sword",
  "enchantment": "Sharpness",
  "weight": 10,
  "max_level": 5,
  "mutual_exclusivity": "Smite,Bane of Arthropods"
},
```

Fig 3.A.1 JSON File Snippet.
(Source: Author's Archive)

B. Simulation Assumptions

To maintain a manageable scope while preserving the essential characteristics of Minecraft's enchanting system, this simulation will assume several things:

- First, the simulation focuses on enchantments obtainable through the Enchantment Table. The number of surrounding bookshelves is assumed to be sufficient to access the highest enchantment option available to players (level 30).
- Second, the simulation uses official enchantment weights and compatibility rules obtained from the Minecraft Wiki page. These values determine the probability of selecting enchantments and ensure that invalid enchantment combinations cannot occur.
- Third, the original Minecraft enchantment power calculation is not implemented. Instead, all enchantments applicable to the selected item are considered eligible candidates. Enchantment levels are generated randomly within its valid range defined by each enchantment's maximum level.

These assumptions simplify the implementation while preserving the probabilistic behavior necessary for combinatorial and probability analysis.

C. Enchantment Selection Algorithm

The simulation begins by selecting an item type, such as sword, bow, fishing rod, or tool. Based on the selected item, a candidate pool of valid enchantments is generated from the dataset.

Each enchantment has their weight value that determines its probability of selection. A weighted random selection process is then used to choose the first enchantment. Enchantments with higher weights have a higher probability or chance of being selected.

After the first enchantment is selected, an enchantment level is assigned randomly within the range from level 1 to the enchantment's maximum level. The selected enchantment is then added to the resulting enchantment set.

To mimic Minecraft's behavior of allowing multiple enchantments on a single item, the simulation may continue selecting additional enchantments. The probability of obtaining additional enchantments decreases after each successful selection, resulting in a smaller chances of obtaining larger enchantment sets.

D. Conflict Resolution

Minecraft has an enchantment compatibility rules that prevent certain enchantments from appearing together on the same item. These restrictions are incorporated into the simulation through a conflict resolution mechanism.

Whenever an enchantment is selected, all enchantments listed in its mutually exclusive group are removed from the candidate pool before the next selection occurs. This process guarantees that invalid enchantment combinations cannot be generated.

For example, the enchantments Sharpness, Smite, and Bane of Arthropods are mutually exclusive on a sword. If Sharpness is selected, both Smite and Bane of Arthropods will be removed from the candidate pool for the next selection.

The conflict resolution mechanism ensures that all generated enchantment sets remain consistent with Minecraft's official compatibility rules.

E. Python Implementation

The simulation is implemented using Python due to its extensive support for data processing and random number generation. The enchantment dataset is loaded from a JSON file and stored as a collection of dictionaries. Weighted random selection is performed using Python's built in random module, while conflict resolution is handled through candidate pool filtering.

To analyze the probabilistic behavior of the enchantment system, Monte Carlo simulation is used by repeatedly executing the enchantment algorithm for a large number of trials. During each trial, the resulting enchantments are recorded for statistical

analysis. The output of the simulation includes enchantment frequencies, enchantment probabilites, and the frequency of generated enchantment combinations. These results are exported as CSV files and later used in the analysis section.

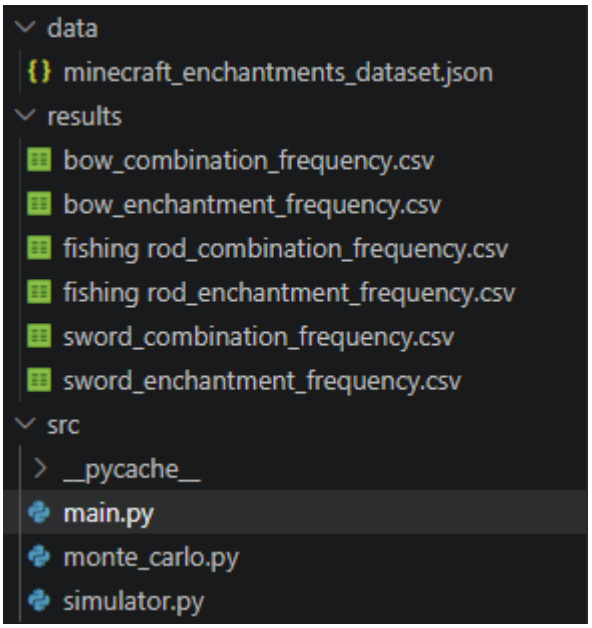


Fig 3.E.1 Project Structure.

(Source: Author's Archive)

The complete source code, datasets used in this study, and the results are available in the GitHub repository provided in the Appendix section.

IV. Results and Analysis

This section presents the results obtained from 100,000 Monte Carlo simulation trials for each selected item category. This analysis focuses on three representative Minecraft items: Bow, Fishing Rod, and Sword. These items were selected because they contain different enchantment sets, probability distributions, and compatibility rules. The simulation results are analyzed based on enchantment frequency, probability distribution, and enchantment combination frequency.

A. Bow

Table 4.A.1 Top 5 Bow Enchantments.
(Source: Author's Archive)

Enchantment	Frequency	Probability
Power	66945	0.66945
Unbreaking	44503	0.44503
Flame	20976	0.20976
Punch	20959	0.20959
Infinity	10946	0.10946

The simulation results show that Power is the most frequently generated enchantment, appearing in 66.95% of all trials. This outcome is consistent with the dataset, as Power enchantment has the highest weight among all Bow enchantments. Similarly, Unbreaking appeared as the second most common enchantment with a probability of 44.50%, while Flame and Punch have nearly identical probabilities due to having equal weights in the dataset.

The least frequent enchantment is Infinity, appearing in only 10.95% of the trials. This result is expected because Infinity has the lowest weight value among the available Bow enchantments. Overall, the probability distribution closely follows the weighted random selection mechanism implemented in the simulation. Enchantments with higher weights consistently appeared more frequent than enchantments with lower weight.

Table 4.A.2 Top 5 Bow Enchantment Combinations.
(Source: Author's Archive)

Combination	Frequency	Probability
Power	24770	0.24770
Power, Unbreaking	15788	0.15788
Unbreaking	12605	0.12605
Power, Punch	5894	0.05894
Flame, Power	5720	0.05720

The most common Bow enchantment combination is Power alone, with the probability of 24.77%. This result indicates that many simulation trials terminated after selecting the first enchantment, which is consistent with the decreasing probability mechanism used for generating additional enchantments.The combinations of Power plus Unbreaking and Unbreaking alone also appeared frequently due to the relatively their high enchantment weights.

As the number of enchantments within a combination increased, the observed frequency decreased significantly. For example, combinations of four or five enchantments occurred much less frequent than a single enchantment outcomes. This behavior is expected because the probability of selecting additional enchantments decreases after each successful selection.

B. Fishing Rod

Table 4.B.1 Top 3 Fishing Rod Enchantments.
(Source: Author's Archive)

Enchantment	Frequency	Probability
Unbreaking	73015	0.73015
Lure	44763	0.44763
Luck of the Sea	44524	0.44524

The simulation results indicate that Unbreaking is the most frequently generated Fishing Rod enchantment, appearing in 73% of all trials. This outcome shows its relatively high weight withit the dataset. The enchantments Lure and Luck of the Sea have very similar probabilites, appearing in 44.76% and 44.52% of the simulations. The relatively balanced probabilities of Lure and Luck of the Sea suggest that the weighted random selection process distributed enchantments according to their weights value. The results shows that the simulation model is able to reproduce expected probabilistic beavior across multiple trials.

Table 4.B.2 Top 7 Fishing Rod Enchantment Combinations.
(Source: Author's Archive)

Combination	Frequency	Probability
Unbreaking	27891	0.27891
Lure, Unbreaking	16396	0.16396
Luck of the Sea, Unbreaking	16259	0.16259
Luck of the Sea, Lure, Unbreaking	12469	0.12469
Lure	11189	0.11189
Luck of the Sea	11087	0.11087
Luck of the Sea, Lure	4709	0.04709

The most frequently observed combination is Unbreaking alone, appearing in 27.89% of all trials. The combinations of Lure plus Unbreaking and Luck of the Sea plus Unbreaking followed closely, indicating that Unbreaking is commonly selected among other Fishing Rod enchantments.

Interestingly, the combination of Luck of the Sea, Lure, and Unbreaking appeared in more than 12% of the simulations. This relatively high frequency can be explained by the small number of available Fishing Rod enchantments and the absence of mutually exclusive enchantment pairs.

Unlike the Sword, no mutually exclusive enchantments were defined for Fishing Rods. For the consequence, all valid enchantment combinations are permitted during the simulation process, resulting in a broader range of compatible enchantment outcomes.

C. Sword

Table 4.C.1 Top 8 Sword Enchantmetns.
(Source: Author's Archive)

Enchantment	Frequency	Probability
Sharpness	35324	0.35324
Unbreaking	28515	0.28515
Knockback	28159	0.28159
Bane of Arthropods	17660	0.17660
Smite	17656	0.17656
Sweeping Edge	12346	0.12346
Fire Aspect	12312	0.12312
Looting	12262	0.12262

Among all Sword enchantments, Sharpness appeared most frequently, with a probability of 35.32%. This result is consistent with the dataset because Sharpness has the highest weight value among the available Sword enchantments. The enchantments Unbreaking and Knockback followed with the probabilites of 28.52% and 28.16%.

The enchantment Smite and Bane of Arthropods have nearly identical probabilites, reflecting their equal weights in the dataset. Even though their weight value are the same as Unbreaking's, but their frequency are not as high as Unbreaking's frequency because both Smite and Bane of Arthropods enchantments are mutually exclusive with Sharpness, which has a higher weight value.

Similarly, Looting, Fire Aspect, and Sweeping Edge appeared less frequently because they have lower weight values. Overall, the results demonstrate a strong relationship between enchantment weight and observed probability. Higher weight enchantments consistently appeared more often.

Table 4.C.2 Top 7 Sword Enchantment Combinations
(Source: Author's Archive)

Combination	Frequency	Probability
Sharpness	13731	0.13731
Unbreaking	7018	0.07018
Bane of Arthropods	6957	0.06957
Knockback	6873	0.06873
Smite	6851	0.06851
Sharpness, Unbreaking	4983	0.04983
Knockback, Sharpness	4854	0.04854

The most common Sword enchantment combination was Sharpness alone, occuring in 13.73% of all simulation trials. It is followed by Umbreaking, Bane of Arthropods, Knockback, and Smite as a single enchantment outcomes. Several two enchantment combinations also appeared frequently, including Sharpness plus Unbreaking and Sharpness plus Knockback. These combinations benefited from the relatively high weights value of their constituent enchantments and fully compatible each others.

An important observation is that no combinations containing Sharpness and Smite, Sharpness and Bane Arthropods, or Smite and Bane of Arthropods were generated together on the same item during the simulation. These enchantments are mutually exclusive in the dataset and therefore cannot appear together on the same item.

Additionally, combinations that contains four or more enchantments are consider less common than a single or two enchantment outcomes. The rarest combinations appeared only a few times among 100,000 simulation trials. It demonstrates how rapidly the porbability of obtaining large enchantment sets decreases as additional selections are performed, illustrating the combinatorial complexity of the enchantment generation process.

V. Summary and Recommendations

The results of the Monte Carlo simulation demonstrate that the proposed enchantment model successfully reproduces the probabilistic behavior of Minecraft's Enchantment Table system. Across all three analyzed items, Bow, Fishing Rod, and Sword, enchantments with higher weight values consistently appeared more frequently than enchantments with lower weights. This finding confirms that the weighted random selection mechanism effectively shows the probability distribution defined in the enchantment dataset.

The simulation also highlights the combinatorial nature of the enchantment generation process. Single enchantment

outcomes are generally more common than combinations that contains multiple enchantments, while larger enchantment combinations appeared significantly less frequently. This behavior is expected because the probability of obtaining additional enchantments decreases after each successful selection by the mechanism. As a result, the simulation demonstrates how combinatorial principles influence the distribution of possible enchantment outcomes.

The conflict resolution mechanism also successfully enforced the compatibility rules between enchantments. No mutually exclusive combinations, such as Sharpness with Smite, Sharpness with Bane of Arthropods, and Smite with Bane of Arthropods, were generated during the simulation. This indicates that the implemented algorithm correctly prevented incompatible enchantments from being selected simultaneously on the same item.

Based on these findings, Monte Carlo simulation provides a practical application of combinatorics, probability theory, weighted random selection, and conflict resolution in modeling a game-based random system. The simulation can serve as an educational example of how mathematical concepts can be applied to analyze and predict outcomes in comlex combinatorics and probability environments.

Future work to improve the accuracy of the model by involving additional Minecraft enchantment mechanics that were not considered in this study. These include enchantability values, enchantment power requirements (minimum and maximum power levels), bookshelf effects, player level variations, and the complete enchantment generation algorithm used in Minecraft Java Edition. Implementing these features would produce a more realistic simulation and allow for a better analysis of enchantment probabilities and combinations.

VI. Appendix

The complete source code, dataset, simulation results, and supporting files used in this study are available in the following GitHub repository: https://github.com/aria1811/matdis-minecraft-enchantment-simulation.git

Acknowledgment

The author would like to express sincere gratitude to Allah Swt. for His blessings, guidance, and strength throughout the completion of this paper. Without His grace, this work would not have been possible. The author also extends sincere appreciation to Mr. Dr. Ir. Rinaldi Munir, M.T., the lecturer of the IF1220 Discrete Mathematics course, for providing valuable knowledge and support during the learning process.

References

[1] Munir, Rinaldi. 2025. *Kombinatorika (bagian 1).* https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/18-Kombinatorika-Bagian1-2026.pdf (accessed on 17 June 2026)

[2] Munir, Rinaldi. 2025. *Kombinatorika (bagian 2).* https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/19-Kombinatorika-Bagian2-2026.pdf (accessed on 17 June 2026)

[3] Siegmund, David O.. 2026. *probability theory.* https://www.britannica.com/science/probability-theory (accessed on 18 June 2026)

[4] TT, Jack. 2023. *Understanding the Weighted Random Algorithm.* https://dev.to/jacktt/understanding-the-weighted-random-algorithm-581p (accessed on 18 June 2026)

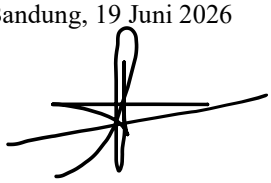
[5] Kenton, Will. 2026. *Monte Carlo Simulation: What It Is, How It Works, History, 4 Key Steps.* https://www.investopedia.com/terms/m/montecarlosimulation.asp (accessed on 18 June 2026)

[6] Minecraft Wiki. 2026. *Enchanting table mechanics.* https://minecraft.wiki/w/Enchanting_table_mechanics (accessed on 18 June 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



Diandra Aria Yufana, 13525113