

Modeling and Implementation of Weighted Random System in Game Design

Nalla Jayanth Chowdary

Department of Data Science

Sreenidhi Institute of Science and Technology

22311a6715@ds.sreenidhi.edu.in

Devarakonda Shashi Varun

Department of Data Science

Sreenidhi Institute of Science and Technology

22311a6748@ds.sreenidhi.edu.in

Gajji Ranjith

Department of Data Science

Sreenidhi Institute of Science and Technology

23315a6706@ds.sreenidhi.edu.in

Mr. K. Rajashekar

Department of Data Science

Sreenidhi Institute of Science and Technology

rajashekar.k@sreenidhi.edu.in

Abstract

Game designers these days are trying to get away from stuff that is just completely unpredictable. They want random elements that players can actually control or tweak a bit. I mean, the old way with uniform probabilities often feels off. It leads to experiences that are all over the place, and players do not get much chance to strategize or feel involved.

This project is about a weighted random system. It uses probabilities that are managed carefully, plus some rules to adjust things on the fly. That way, it boosts predictability without killing the surprise, and it makes interactions fairer for everyone. Traditional random stuff just sits there. You cannot really change it to fit the game situation or how far along someone is.

In interactive games, there has to be some balance. Players need agency mixed with a bit of chance so they stay hooked and can make choices that matter. But the usual methods do not adapt well. They ignore what players do, which might leave people feeling unsatisfied. It seems like that is a big issue, since human activities should influence outcomes somehow. Some might argue pure randomness keeps it exciting, but others say it just frustrates folks. This part gets a bit messy to explain. Anyway, the new approach tries to fix that gap.

Introduction

In general, games become much more entertaining if some random elements were implemented. However, an entirely random game may also have its drawbacks, especially when talking about rare items and events that occur either too frequently or not at all.

To me, weight-based systems seem to help solve this dilemma. As opposed to the fully random one, weighted system involves probabilities depending on the preferences of designers.

Random elements seem to play a critical role in designing games. They are vital in order to ensure unpredictability and, hence, attract users' attention. At the same time, it may look unfair to have something entirely random without any sense or logic.

As far as the relationship between the player's actions and the generated elements is concerned, I am not quite sure about the most optimal method to establish such correlation, yet I agree that it is crucial to achieve this goal.

Uniform distribution appears to be the simplest technique when developing the algorithm. Each possibility receives equal chances in this case; however, players cannot influence the result in any significant way. In other words, their actions do not lead to anything.

There are many opinions on whether random algorithms should be complex or rather simple. However, it is evident

that player engagement will decrease anyway. The purpose of such an approach is to overcome such difficulties through the use of a weighted random algorithm designed to be employed in gaming systems. The use of weights of predetermined probability enables the use of randomness in a controlled manner so that fairness is maximized while still retaining the element of uncertainty. All this comes packaged with the ability of the system to adapt to new situations by adjusting the probabilities based on rules. A session-based approach provides the foundation of the game's mechanics, enabling the player to pass several cycles with reduced chances of accomplishing the goals. During the development of this app, the designer incorporates extra strategic elements to create more complex conditions and encourage active involvement in the process and responsible decision-making. Such an integration of controlled uncertainty and advancement can be considered more balanced and rewarding for players. The Unity framework, which is the C game engine, helps implement the system. It utilizes the principle of modularity and has a modular structure that simplifies modifications and maintenance of the application.

In conclusion, the findings of this research highlight the possible benefits of integrating the proposed approaches within video games' design and implementation.

Literature Survey

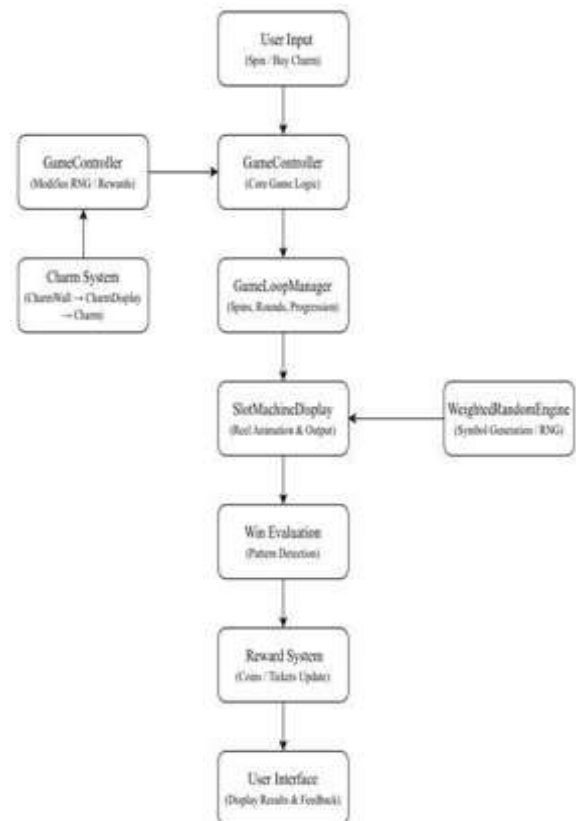
The most popular way to increase the appeal of the game by making it more appealing to play is by adding randomness. The traditional approach tends to rely on the use of simple mechanisms for generating random values, such as uniform probability distribution. Every possible result is seen as equally likely. Such techniques tend to be used in slot machine-based games due to their simplicity.

One of the early branches of probability theory helped establish a method for random selection based on a uniform distribution. Yet, the concept of weighted randomness was introduced since there was a need to create non-uniform probability distributions for situations existing in real life. The process of weighted random selection depends on the weights that are given to objects. This concept is widely used in games, simulations, and decision-making systems. One of the simplest forms uses CDF or cumulative distribution function.

Both Walker and Vose have made important contributions in the domain of weighted random systems, but the Alias Method seems to stand out. The complexity of preprocessing is $O(n)$. The sampling complexity of this method is $O(1)$. In this method, apart from the alias table, a probability table is also used. This method makes it possible to sample in constant time from a discrete

probability distribution. As a result, this method can be used very effectively for real time applications such as games. The newer versions of alias methods include the following: MMORPGs with a well-developed loot system are ideal for this.

Methodology



System architecture.

In the creation of a game that will incorporate the weighted random approach, it is important to have a list of outcomes, each one assigned weights. Every item that will be selected will be given a weight that represents how likely it is to choose the item. Rarity determines the weight of the loot items while spawn rate determines the weight of the enemy generation system. The weight total will then be determined by adding all the individual weights.

The basic way of implementing the algorithm entails generating a random number that ranges from 0 to the overall weight of the objects. The process continues to cycle through the objects and monitors their overall weights. Once a specified amount, generated randomly, is reached, the relevant object will be chosen. This method, being simple and easy to implement, makes it suitable for use in low-level programming or games.

The cumulative distribution function (CDF), a special type of array, is used to hold the computed cumulative weight, which can contribute to increased efficiency in a scenario with a large number of elements. Rather than iterating through all the elements, a search by means of a divide-and-conquer algorithm will be conducted to identify the appropriate interval in which the random number lies.

The reason that many high-quality games employ the alias scheme is the possibility to provide constant-time sampling. Two additional tables are produced from the distribution table through the alias technique – one representing the alias table, and the other one being the probability table. To decide whether to choose the actual index or its alias index is done randomly on the fly by selecting a random index first. This makes the sampling algorithm extremely fast despite the extra cost and memory needed for pretreatment.

It is common that game weights are changed dynamically based on events in the game or players' actions. For instance, if there was an additional "Luck" feature to improve the chance of finding rare loot in some game, higher weights would be assigned. In other cases, weights can be re-calculated on the go before the choice is made. It adds processing complexity, yet provides much more flexibility.

Another important aspect of implementation is the conditional weighted selection approach, whose result depends on particular conditions. In other words, the system considers the current status of the game, filters out all possible solutions, and performs the weighted random selection. This leads to filtering out wrong solutions, making games more consistent logically.

This type of weighted random selection can be implemented when rewards are given separately in games like cards and others. Whenever an object is selected and removed from the list, then the process of updating weights comes into play. Repetition needs to be avoided and variety is ensured.

A complex game algorithm usually employs a multilayer weighted algorithm. For instance, a game may use preassigned weights for categorizing certain items as "common," "rare," or "epic." This is followed by the employment of the same weights for a second weighted choice within the category identified. Through this layering process, one can manipulate the balance of probabilities in the game.

From the viewpoint of system design, the weighted random generator may be considered a part of the game engine and, as such, is highly interconnected with other

modules, among which the ones related to the input, output, and game state processing. In order to ensure proper performance, memory management and caching of precomputed values is important.

Usability vs. Performance Trade-off When Using Weighted Randomization Techniques

More complicated techniques have to be applied in case when randomization is used in real time and with large volumes of data. Simpler techniques can suffice in other cases. The choice depends on particular demands of the project: performance limitations, the ability to scale and adapt dynamically.

Modules

Data Entry Unit

The weighted random system requires all data to be gathered through the input module. The input module includes everything that is necessary, such as the items or events being selected, their weights, and other data. These include data such as the characteristics of the players, the difficulty of the game, and the conditions of the environment. Data are always checked for validity and proper format, without any errors, such as negative weights or datasets that are empty. In dynamic systems, the input module continuously receives updates from the game engine.

Weight Assignment

The probability weight of each item is set and regulated by the weight allocation module. The initial weights are set according to the rules of the game, which include how often treasure appears or how frequently enemies spawn. In order to ensure that the distribution of weights is even, the module may also perform normalization processes. To make the algorithm more intelligent and dynamic, advanced algorithms may employ a technique where the weight is adjusted dynamically based on specific variables.

Module for Preprocessing

The data is organized to enable fast access for random selection in the preprocessing phase. In this phase, the system generates additional data structures such as alias tables and the cumulative distribution array and determines the total weight. Probability and alias tables are generated by efficient systems, while the CDF is produced in binary search systems. This phase is important

for maximum efficiency in execution time because it minimizes computational work.

Module for the Random Number Generator

This process is dependent on random numbers created using this particular module. This process will most often use PRNGs, which are already integrated into computer programming languages and gaming systems. Since randomness directly impacts the fairness and unpredictability of the game, it is an important concept. To synchronize multiplayer games or for testing purposes, seed values are used in some of the implementations. More complicated versions might use improved algorithms to achieve better statistical results.

Module for the Selection Engine

The selection engine is the core part of the algorithm where the actual weighted random selection takes place. The module analyzes the generated random number and chooses the corresponding element using one of the three algorithms chosen for the selection process. The correctness of the probability distribution given is ensured during the selection process. The efficiency of this module directly influences the response time of the entire system, especially in real-time games.

Module for Condition Filtering

The object cannot be trusted in a game. Some of the considerations that the condition filter module looks at include level, state of the game, mission status, and environmental restrictions. This filter removes anything which is not considered before the beginning of selection. A situation in which higher level opponents show up too early in a game or items that do not exist are awarded is an unrealistic outcome which it will filter out.

Results



Unity Editor Screen



Slot Machine (Before Spin)



Shop Before Buying A Charm (GoldenCherry +2
BaseValue -2 tickets)



Charm Shop After The Charm Purchase.



Game Over Screen (Level 5)

Symbol	ExpectedProbability	ExpectedCount	ObservedCount	ObservedFrequency
Lemon	0.194	1455	1472	0.196
Cherry	0.194	1455	1438	0.191
Clover	0.149	1118	1105	0.147
Bell	0.149	1118	1132	0.151
Diamond	0.119	893	910	0.121
Treasure	0.119	893	875	0.117
Seven	0.075	563	568	0.076

System evaluation

Conclusion

The slot-based gaming system in this project features well-balanced and interesting interactive aspects including regulated randomness, patterns recognition for wins, and dynamic gaming modifiers. A modular structure of the gaming system helps understand, scale, and manage

it better. The random engine with its weighted algorithms generating symbols according to predetermined probability distributions works as the main idea of the project. Randomness is the key feature of any game, because it provides fairness. The analysis of experiment results shows high correlation between symbol frequencies and expected probabilities, thus showing how precise the randomization process is. Some of possible winning patterns include jackpot combinations, zigzags, diagonals, horizontals, and verticals. In addition, the algorithm manages to control stackings of patterns, allowing discovering multiple winning combinations in one round, thus making the game more interesting and improving overall gaming experience. Implementation of the reward system inside the GameController module allows distributing both cash and tickets depending on the discovered exact patterns. It guarantees stable and fair allocation of the rewards because of the capacity to maintain balance under various gaming conditions. Another interesting strategic aspect of the project includes the charm system, enabling players to influence probability distributions of symbols and winning prizes. Thus, this feature increases interest in gaming due to its combination of random and controlled processes of decision making. With regards to performance issues, the project can be described as rock-steady and fast-working. No lags, crashes or delays were noticed during spins, animations, and UI manipulations during several gaming sessions. The project has been successfully completed and has achieved all its objectives of Creating a responsive, interactive, and fair game-playing system. By combining reason with choice and risk, the system highlights the relevance of a controlled probabilistic system in developing interactive games.

References

- [1]. Unity Technologies. (n.d.). Unity User Manual and Scripting API. Retrieved from <https://docs.unity3d.com>
- [2]. Unity Technologies. (n.d.). Unity Random Class Documentation. Retrieved from <https://docs.unity3d.com/ScriptReference/Random.html>
- [3]. Microsoft. (n.d.). C# Programming Guide. Retrieved from <https://learn.microsoft.com>
- [4]. Knuth, D. E. (1997). The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley.(For random number generation concepts)
- [5]. Adams, E., & Rollings, A. (2007).

- Fundamentals of Game Design. Prentice Hall.
- [6]. Schell, J. (2008). The Art of Game Design: A Book of Lenses. CRC Press.
- [7]. Fullerton, T. (2014). Game Design Workshop: A Playcentric approach to Creating Innovative Games. CRC Press.
- [8]. Novak, J. (2012). Game Development Essentials: Game Mechanics. Delmar Cengage Learning.
- [9]. Unity Learn Platform. (n.d.). Game Development Tutorials. Retrieved from <https://learn.unity.com>
- [10]. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley