



Republic of the Philippines
NUEVA VIZCAYA STATE UNIVERSITY
Bayombong, Nueva Vizcaya



COLLEGE OF ARTS AND SCIENCES
BACHELOR OF SCIENCE IN COMPUTER SCIENCE

THESIS

**A COMPARATIVE STUDY ON PSEUDO-RANDOM AND
UNIFORM DISTRIBUTION GENERATOR IN
GAMES**

CHINO JOSHUA G. ABBAGU
GRANEL JAYLORD A. CABUENA

JULY 2022

	QUALITY FORM	FR-INS-07.Rev00
	THESIS PROJECT ACCEPTANCE SHEET	TAS No.: _____

This thesis project entitled “*A COMPARATIVE STUDY ON PSEUDO-RANDOM AND UNIFORM DISTRIBUTION RANDOM GENERATOR IN GAMES*”, prepared and submitted by **Granel Jaylord A. Cabuena** and **Chino Joshua G. Abbagu**, in partial fulfillment of the requirements for the degree **BACHELOR OF SCIENCE IN COMPUTER SCIENCE (BSCS)** is hereby recommended for acceptance.

THESIS PROJECT COMMITTEE

EMMANUEL M. DANGUILAN, MBA, RN
Adviser

Date

MICHAEL R. LACAR, MIT
Member, Panel of Examiners

Date

JOAN HAZEL V. TIONGSON, DIT
Member, Panel of Examiners

Date

CARMELO ALEJO D. BISQUERA, DIT
Chair, Panel of Examiners

Date

VILCHOR G. PERDIDO, MIT
Chairman, CS Department

Date

ALVIN B. FELIX
Research Coordinator

Date

EMILIA T. DANGILAN
Language Editor

Date

Accepted and approved in partial fulfillment of the requirements for the degree of **Bachelor of Science in Computer Science (BSCS)**.

FILEMON A. PAMITTAN, PhD.
Dean, College of Arts and Sciences

Research Contribution No.: _____

Date: _____

RAMON P. BANGAO, JR.
Research MIS Coordinator

Date

ACKNOWLEDGEMENT

First and foremost, we thank God Almighty for the wisdom He has bestowed upon us, as well as the strength, peace of mind, and good health needed to complete this research.

The completion of this study would not have been possible without the participation and cooperation of a large number of people whose names may not be listed here. We, the researchers would like to convey our heartfelt gratitude to the following persons for their assistance in making this study possible in some way:

We would like to extend our thanks to the Dean of the College of Arts and Science, Professor Filemon A. Pamittan, for his consideration, patience, and untiring support for the completion of this study.

We would like to give our deepest gratitude to our panelist Dr. Carmelo Alejo D. Bisquera, Dr. Joan Hazel V. Tiongson, Sir. Michael L. Lacar, who are the keys in the realization of this study. Thank you for their time suggestion and encouragement. Allowing us to get the necessary information we need to make this thesis possible.

To Ma'am Nilda Dela Cruz, thesis professor, for providing us invaluable supervision, support, and tutelage during the course of our research study.

To Sir Emmanuel Danguilan, research adviser, for providing us with ideas for conducting this research and for taking the time to guide and share his knowledge in preparing for this research.

Our thanks and appreciation also go to our respondents for their willingness and time to participate.

We consider it an honor to work with my colleagues, and the other group, for the memories we had shared and the teamwork in every task and activity were in, we have never failed to cooperate with one another especially in tough and badtimes.

Last but not the least, to our beloved parents and siblings for the moral and financial support in conducting this research.

DEDICATION

This work is a fruit of countless and arduous sacrifices. Through the researchers' effort, this work is heartily and proudly dedicated to the people who serve as an inspiration. From parents and guardians, to classmates and circle of friends whom extended their help in the midst of problems while doing this work and provide their moral, spiritual, emotional, and financial support.

And lastly, we dedicate this book to the Almighty God, thank you for the guidance, strength, power of mind, protection and kills and for giving us a healthy life. All of these we offer to you.

TABLE OF CONTENTS

Title	Page
Title Page	i
Capstone Project Acceptance Sheet	ii
Acknowledgement	iii
Dedication	v
Table of Contents	vi
List of Tables	ix
List of Figures	x
Abstract	xi
Chapter 1. THE PROBLEM AND REVIEW OF RELATED LITERATURE	1
Introduction	1
Review of Related Literature	2
Conceptual Framework	12
Paradigm of the Study	13
Objectives of the Study	14
Significance of the Study	14
Scope and Delimitations of the Study	15
Definition of Terms	16
Chapter 2. METHODOLOGY	20
Research Design	20
Participants of the Study	22
Instrumentation	23

Data Gathering Procedures	24
Data Analysis	25
Chapter 3. RESULTS AND DISCUSSION	
Results and Discussion	27
Chapter 4. SUMMARY OF FINDINGS, CONCLUSION & RECOMMENDATIONS	
Summary of Findings	37
Conclusion	37
Recommendations	38
REFERENCES	39
APPENDICES	
Appendix A (Participant’s Data)	42
Appendix B (Game Survey Table)	44
Appendix C (Game Design with GDScript)	46
Appendix D (Integrating C++ in Godot)	52
Ways to Integrate C++ into Godot	53
Integrating C++: Setup	54
Integrating C++: Game – Pseudo	56
Integrating C++: Game – Uniform	60
Appendix E (Game Flowchart: Dise)	62
Appendix F (Game System Compatibility)	64
Appendix G (Compiler Source)	66
Appendix H (Random Walk Code)	69
Appendix I (Pseudo-Random Dise Testing Player Mechanics)	73
Appendix H (Uniform Distribution Dise Testing Player Mechanics)	76

CURRICULUM VITAE	79
-------------------------------	-----------

LIST OF TABLES

Title	Page
1. Gacha Concept	8
2. Table of Participants	23
3. Recommendation Percentage	26
3.2 Participant's Data	44
3.3 Pseudo-Random Cho-han Testing Player Mechanics	75
3.4 Uniform Distribution Cho-han Testing Player Mechanics	78

LIST OF FIGURES

Title	Page
1. Conceptual Framework of the Study	12
2. Research Paradigm	13
3. SDLC (RAD Model)	21
4. Population and Year	23

ABSTRACT

Random Number Generators (or RNG) are a part or an especially important feature in Video Games as it can be used to produce unpredictable events and random outputs. This study analyzes what and will be the appropriate RNG for Games. To fulfill and to analyze deeper, we take the two RNGs work in their different algorithms, test, and create a game while comparing them with how fast they respond and how frequent the repetition is. We used Dev C++ as our Integrated Development Environment (IDE) for the tests, Godot as our game engine, and Visual Studio as to create a C++ library for the game. The data results show that Uniform Distribution produce more equally random outputs than Pseudo-Random. Correct choice of RNG is a good choice now that every game uses RNG.

Keywords: *Random Number Generators (RNG), Algorithms, Uniform Distribution, Pseudo-Random, Godot*

Chapter 1

THE PROBLEM AND REVIEW OF RELATED LITERATURE

Introduction

As we live in this world, games are evolving overtime and so many new things have been improved. That one thing is the algorithm of randomness. We all have at some point played video games once and sometimes wonder how every instance spawn, objects move by itself, some instance where currency is based on simple math. We all know how those things do but not how those things work. Video games like technology evolves overtime as developers invent new innovations to their work: Physics, Engine, and many aspects of gaming that are based in real life. But all of these, we found that randomness is remarkably interesting. Many research studies are based on randomness such as predictability, probability and statistics, social perception and many more. We, researchers, only focus on the aspect of Randomness in games or as we all call it “Random Number Generators”.

Random Number Generators are a part or an special important feature in Video games as it can be used to produce random outputs, this includes; random events or occurrences, chances of having a doubled product on a single input, being lucky. During this study, we will be acknowledging Random Number Generators as RNG/s. There is much more when we say just RNG use for gaming purposes. Gacha is more the proper word or at least the gaming community knows, basically it has RNG elements with a given probability; users see this as a bad and a good thing. While it is a sad thing, players gamble their real-world currency for “digital/virtual loot” but that is not guaranteed you will get but just a chance. In a good thing, players can be

satisfied by getting that “loot” and being lucky with it: not everybody can be lucky, that is why it is mixed with good and bad reactions.

The aim of this study is to make a game implementing both Pseudo-Random Number Generator and Uniform Distribution Random Number Generator. The participants will play the game that the researchers made and to test which Random Number Generator best suits for the game. Furthermore, the researchers aim to compare the Pseudo-Random Number Generator and Uniform Distribution Random Number Generator and describe the benefits of this Random Generators through the game and through tests using data outputs.

The researchers game design for the First Testing Phase is called “Cho-han”, a game that originated from Japan. Though it is basically a game of two dice summing up then the player will guess if the sum of the hidden two dice is even or odd, if the player guessed right then he gets a point. We will do exactly two games of Cho-han with the use of Pseudo-Random Number Generator and Uniform Distribution Random Number Generator. The programming language C++ is mainly used in the development and a game engine called “Godot”.

Review of Related Literature

The review of the related literature contains topics that are relevant to this study where in their literature will also become a guide for both researchers and the development of the game.

Random Numbers

A lot of games in this time use random numbers which are a part of a system called Random Number Generator. When we say random, it can be defined as “made or

chosen without method”. It is difficult to consider how this concept could be implemented in a video game. An example of a system that computes random numbers is through Pseudo-Random Number Generator or PRNG for short. PRNGs generate random numbers by using a mathematical formula or a precalculated list, which corresponds to someone rolling a die many times and taking note of the results (Haahr, 2019). PRNGs are important especially to games, as almost random numbers are generated using this system. Random numbers exist in almost all of the games out there, but what exactly does these elements do?

This is just a simple example of a PRNG:

```
x = seed(); \\ Variable x holds a seed method where the formula is based on the  
developer.
```

A seed acts like a holder for the initial state, it changes when variable x produces another random number.

where the formula is: **x = (5 * x) % 9;** \\ % is modulo. Now the given numbers can be any numbers, the important part is the variable x.

Assume the x starts with 8 therefore:

```
x = (5 * 8) % 9; \\ 4 is the output of x.
```

and then

```
return x; \\ Returns the computed output and onto the next seed where it tries to  
compute again with the same formula or until it reveals the initial value for the  
first seed.
```

Pseudo-Random

The most known and used Random Number Generator in software right now. The example of just a method of how to use this is the *rand()* function (See Appendix C). What does it do exactly? Well, it returns a random positive integer at a given range. The numbers are generated by a mathematical algorithm which when given a starting number (called the "*seed*"), generates the same sequence of numbers. Since the same sequence is generated each time the seed remains the same, the *rand()* function generates a pseudo-random sequence. Below is an example code on how to use this function:

```
#include<stdlib.h> \ \ Use for rand and srand.
```

```
int main(){ srand(time(0)); \ \ A seed that holds the rand function for every  
execution, it randoms.
```

```
for(int c=1;c<=10;c++){ \ \ Loop up to 10 times.
```

```
cout<<rand()%25+1;} \ \ Display random outputs between 1 and 25.
```

Note: New seed = New set of random outputs.

Pseudo-Random Numbers

Pseudo-Random Numbers is considered a set of statistically random values or elements that is derived from a specified beginning point and is often repeated over and over. It is also satisfying to generate random numbers from a process that is well to establish understanding of physics is truly random and a mathematical model would match a computing procedure. A pseudorandom number generator is a function that takes a short random seed and outputs a longer bit sequence that “appears random” (Wright

2003). To be cryptographically secure, the output of a pseudorandom number generator should be computationally indistinguishable from a random string. Given a short prefix of the sequence, it should be computationally infeasible to predict the rest of the sequence without knowing the seed. Many so-called random number generators, such as those based on linear feedback shift registers (LFSR) or linear congruences, are not cryptographically secure, as it is possible to predict the sequence from a short prefix of the sequence.

Creating a Seed

A Computer gives a set of random values which is called “seeds”. These so-called seeds produce random output whose values are fixed according to the input. So let us say that seeds are a part of the system RNG. Thus, the output from this RNG is completely determined by the input. Although, if the seeds’ variables are limited, then the number is also limited to those outcomes and cannot exceed the given variable. Suppose that RNG returns a selection of six different numbers from the set $\{1, 2, 3, \dots, 49\}$, that is, it is like a lottery ticket. In addition, if a CPU (Central Processing Unit) clock register is 16 bits. There will be 65, 536 possible random seed values, and thus the RNG can only choose that many of the outcomes 49 is the overall set over 6 random numbers $(49/6)$ with 14 million or less outcomes (13,983,816 lottery tickets).

Assume a RNG seed is used for playing cards, a machine that plays a simultaneous hand of poker that has a chance of getting straight flushes. Since an ordinary single-seed RNG can only provide a relatively tiny subset of the possible

shuffles of a single deck ($52! > 10^{67}$), such as social application of RNG's calls for RNG's with many seeds (Marsaglia, 2003).

Uniform Distribution

Uniform Distribution is considered as the simplest probability/statistical distribution. It derives from the concept of the foundation of statistical analysis and probability theory. Let us assume that you are distributing waivers to any person who walks by, every person would have an equal chance of receiving a waiver, that is if you have a condition like you prefer women to men, then that is considered as not uniform distribution. Now when we say Uniform Distribution, there is two with different outcomes:

A Discrete Uniform Distribution is where the probability has equal chances with a finite value. An example is rolling a die with six possible outcomes with these outcomes having an equal chance of appearing. Therefore, each side has a chance of $1/6$.

A simple representation with the above scenario:

$P(x) = 1/6$; \ \ Assume variable x is a random variable. 1 for every output over 6 possible outcomes. Thus, all outcomes have the same equal probability.

This part is the computation of the mean and variance; not necessarily relevant to the random but this part reveals the possible outcomes.

$E(x) = (n+1)/2$; $E(x) = 3.5$; \ \ The mean measures of location which finds the center of the set. n is the parameter in which case, 6.

$V(x) = n^2-1/12$; $V(x) = 35/12$; 17

A simple code for the scenario:

```
default_random_engine gen; \ A random generator usable.
```

```
uniform_int_distribution<int> distribution (1,6); \ A code that distributes  
random numbers between 1 and 6.
```

```
for(int x=0; x<10; x++){cout<<distribution(gen);} \ Displaying every evenly  
distributed number based on above.
```

Gacha

A word those gamers are familiar with. Gacha is used to define a system where players gamble with luck and probability with real world currency. Whenever we talk about this with other fellow gamers, they answer differently as this could be a sad thing or a good thing. Also called a “loot box” in the same concept with Gacha. This word is a highly controversial topic among the gaming world, with mixed reactions. This so-called Loot Boxes and similar mechanism constitute an “implementation of random procedures used for selection and delivery of rewards in video games” (Nielsen & Grabarezyk, 2018). According to this team of researchers (Brückner et.al, 2019), they conducted a study by analyzing random reward system mechanisms by using these methods: Analyze the relation between the rarity management and game content and Study how random reward systems affect the diversity and complexity of games. They found out that distinctive characteristics of each game lies in the management of duplicate rewards and how they benefit players, i.e., whether their benefit is limited to the same item or applicable. From which they divided it into 4 parts of principles with 4 parts of principle with 4 parts of genres:

1. *Multiple Loot Boxes Principle*, controls how rare it is for diverse types of items. Suitable with *Console-based RPGs* and *Complex games combining single and multiplayer elements*.
2. *Dedicated Benefit Principle*, duplicated rewards that benefits itself. Suitable with *Console-based RPGs* and *Character Oriented RPGs*.
3. *General Benefit Principle*, duplicate rewards that provide general benefits. Suitable with *Complex games combining single and multiplayer elements* and *TCGs*.
4. *Single Loot Box Principle*, single random reward system in different types of items. Suitable with *Character Oriented RPGs* and *TCGs*.

Table 1. *Gacha Concept*

Principles	Multiple Loot Boxes Principle	Single Loot Box Principle
General Benefit Principle	Complex games combining single and multiplayer elements	Complex games such as TCGs, for users are encouraged to have a stable collection
Dedicated Benefit Principle	Conventional console-based RPGs, for it offers rarity management systems	Simple RPGs and such character oriented gacha games.

Monte Carlo

Monte Carlo methods, or Monte Carlo experiments, are a type of computational process that uses repeated random sampling to produce numerical results. The basic idea

is to employ randomness to solve problems that are in principle deterministic. A Monte Carlo Simulation also uses a probability distribution, such as a uniform or normal distribution, to create a model of probable outcomes for any variable with inherent uncertainty and they are commonly used for long-term forecasting. The number of forecasts grows in lockstep with the quantity of inputs, allowing you to anticipate events further out in time with greater precision.

According to IBM Cloud Education, one simple example of a Monte Carlo Simulation is to consider calculating the probability of rolling two standard dice. There are 36 combinations of dice rolls. Based on this, you can manually compute the probability of a particular outcome. Using a Monte Carlo Simulation, you can simulate rolling the dice 10,000 times or more to achieve more accurate predictions. For Monte Carlo applications, it is necessary to use a high-quality random number generator with a long enough period. The theoretical principles underlying the generator and its quality should be published (Owen, 2013).

Godot

Godot is a game engine oriented towards both 2D and 3D game development, the game engine focuses on providing a well-rounded set of tools for development – including a built-in code editor, a graphics rendering engine, audio playback tools, animation tools, and more (Schardon, 2022). As stated by Mr. Lindsay Schardon, the game engine was released in 2014 by Juan Linietsky and Ariel Manzur, Godot has recently risen in popularity among game developers and is quickly becoming a favorite of

many. He said that many developers, on the other hand, have never heard of Godot or have no idea why they should use it over other popular engines like Unity or Unreal.

According to him, Godot has a lot of features that are required for 3D graphics such as lighting systems, physics systems, material support reflection, refraction, and other post-processing effects are all included, as are tools for post-processing effects. As reported by Mr. Lindsay Schardon additional complex visual capabilities, including as shaders and particles, are also supported by the engine, providing developers a complete range of tools to modify their games. He also said that Godot is also a real 2D engine, unlike other engines that attain 2D by just flattening a single axis, this means the engine can efficiently execute 2D backend calculations while still dealing with pixel-based units.

Pseudo-random difference

According to a group of researchers (Vattuilaenen et al., 1993), they conducted research about the different generators from Pseudo-Random Number Generators. This study consists of different programs and tests each one about their efficiency. These generators are concluded in this study: *G05FAF*, *RANMAR*, *RAN3*, *RCARRY*, and *R250*. While *GGL* is acknowledged as a Uniform Random Number Generator, it is heavily based on Pseudo Random Generator's linear congruential method. Every one of these has complex methods for computing random outputs.

Testing RNGs

There are a lot of series of tests that can be use to define how the RNG can be a success. We already know the purpose of RNG is and that is to produce "random"

outputs, that is why testing your RNG is important especially when you want to look for patterns.

According to (Unitychain, 2019), that there are two phases to test the RNG process. [1] is the need of source of entropy; this means that it needs a basis to produce something. And [2] is an algorithm.

These are the basic tests that they listed includes: Skew Correction Algorithm, simulating visual tests, Probability tests, Monobits tests, Runs tests, Binary Matrix Rank tests, and more.

Synthesis

So, what does these literatures have in common? First, we know that these are integral parts of what we call a Random Number Generation. Every part of it has different usability from a seed to producing random outputs. Upon researching these two random number generators, we have known that both of these have the capability to produce random outputs with differing methods. Random Number Generators are a very important aspect of games especially in the Gacha genre as it relies heavily on Pseudo-Random, Uniform Distribution, or True Random. Simulating visual tests are interesting since looking for patterns with numbers is just too frustrating so random walk should be enough for the visual test.

Conceptual Framework

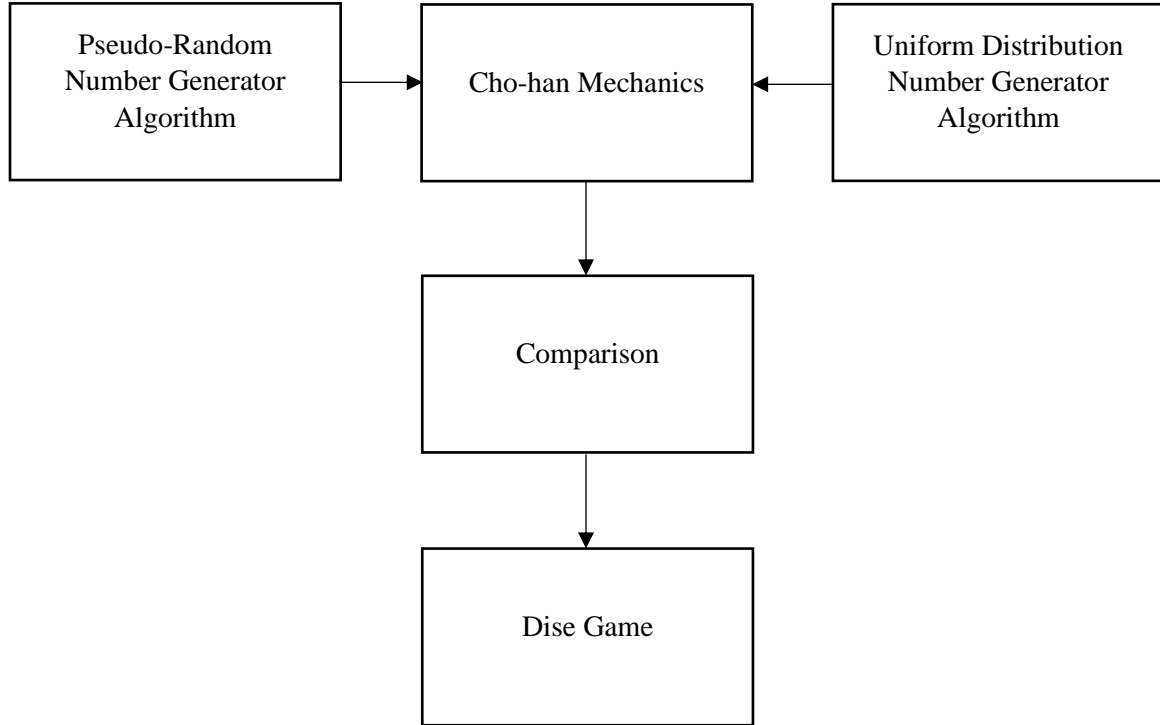


Figure 1. *Conceptual Framework of the Study*

Figure 1 shows the comparability as an independent variable pertains to an efficient instructional method and the similarities between the Pseudo-Random Number Generator and the Uniform Distribution Random Number Generator in games. The second independent variable is the Cho-han Mechanics, which the researchers how does the real-life game works with each of the RNGs. Similarities and differences between the comparability of the Pseudo-Random Number Generator and the Uniform Distribution Random Number Generator by using series of tests and developed games. The efficiency of randomness will be determined after analyzing the comparability of the Pseudo-Random Number Generator and the Uniform Number Random Number Generator.

Paradigm of the Study

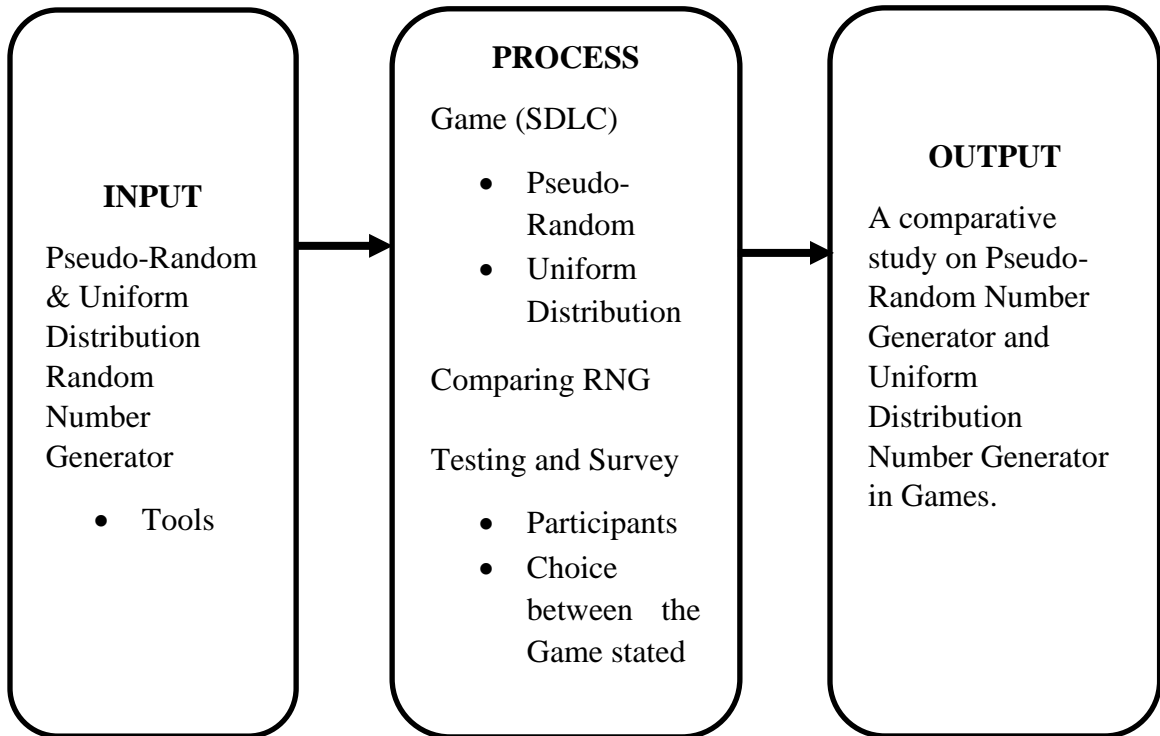


Figure 2. *Research Paradigm*

The Input Process Output of the study shows in Figure 2. chart of our research. Input consists of Analysis with finding the difference, similarities, and the initial results of efficiency of these two RNGs based on other's studies. Process consists of the majority of our research. We will conduct and create two game apps for each of our variables with the same language, after that we will conduct testing called "First Testing Phase" and coordination from our participants; choosing what they prefer between the two. While in the First Testing Phase, maximizing our time, studying and making speculations for the game. The Output comes at last is, A comparative study on Pseudo-Random Number Generator and Uniform Distribution Number Generator in Games.

Objectives of the Study

The main objective of the study is to create a game app using Pseudo Random and Uniform Distribution Number Generator algorithm, particularly the researchers aim

1. To create a game application by using the Cho-han mechanics applying Pseudo-Random Number Generator.
2. To create a game application by using the Cho-han mechanics applying Uniform Distribution Number Generator.
3. To test the efficiency of Pseudo Random Number Generator and Uniform Distribution Random Number Generator algorithms by using the “Dise Game”.

Significance of the Study

The generalization of this study would be a great contribution to the vast knowledge in relation to the aspect of game algorithms.

The findings of this study will redound to the benefit of the following:

Game Developers or Programmers. The information presented will allow them to make their games less determined and thus more difficult for the player to beat, any value that developers' want to initialize in a game object is a candidate for possible randomization. Randomness will fuzz up a game, making it less deterministic and therefore harder to defeat with simple patterns and it is more replay-able. Developers code some possibilities

and let chance determine what happens when the player reaches a certain point in the map.

Video Game Players. Data given will provide the Gamers with information on how randomness works and affects the games. The duration of effort a player's skills can affect opponents is determined by randomness. Without such use of random number generators, video games would be much less entertaining.

Gaming Industries. The results of the study will help the Casino Gaming Industries to find better randomness. Since they must assure randomness in all games so that each user gets a unique experience when playing games such as roulette or slot machines, they must supply randomization in all games.

Future Researchers. The ideas presented may be used as reference data in conducting new research or in testing the validity of other related findings. This study will also serve as their cross-reference that will give them a background or an overview about the aspect of random algorithms in games.

Scope and Delimitations of the Study

This study is focused in examining the efficiency of randomness between Pseudo-Random Number Generator and Uniform Distribution Random Number Generator. Focusing on the important aspects of the two RNGs. The researchers must be proficient to create two game applications, in difference with their algorithms. Not just this but the researchers need to be considerable and open about what must be improved on our Games. In this study, the researchers will focus on the randomness in gaming industries which can be considered as a delimitation. The researchers intend to produce a game

based on both the pseudo-random number generator and the uniform distribution random number generator, as well as enhance their knowledge in preparation for the game application that researchers will develop to meet the goals of this research.

Definition of Terms

To ensure clarity of the words that has been used and to derive a clear concept of the study, the following terms are defined:

C++. C++ is an object-oriented computer language as part of the evolution of the C family of languages. It was developed as a cross-platform improvement of C to provide developers with a higher degree of control over memory and system resources. Main language to use in tests and game “Dise”.

CMD. A command is a specific action assigned to a program to perform a specific task. It commonly refers to a specific word or phrase that tells the computer what to do through a command line interface or shell, depending on what kind of system is being used. To call scones to extract libraries for plugins to work for godot gdnative.

Console-based RPG. Also called Text-based Role-Playing Game, is a genre of game that uses a text-based user interface, that such as, the user interface employs a set of pixelated graphics.

Gacha. A word derived from the slang word “gotcha” which means having an urge to get something according to that event. A popular system where players rely on luck and probability based on a certain type of random number generator.

GDNative. GDNative is a Godot-specific technology that lets the engine interact with native shared libraries at run-time. You can use it to run native code without compiling it with the engine. *Note: GDNative is not a scripting language and has no*

relation to GDScript. A part of Godot engine to be able to developed games in c++ other than using its native language – “GDScript”.

GDScript. GDScript is a high-level, dynamically typed programming language. It uses a syntax similar to Python Its goal is to be optimized for and tightly integrated with Godot Engine, allowing great flexibility for content creation and integration. Godot engine main language.

Hack N’ Slash. Also known as hack and slay or slash 'em up, refers to a type of gameplay that emphasizes combat with melee-based weapons. They may also feature a few projectile-based weapons as well as secondary weapons. It is a subgenre of beat 'em up games, which focuses on melee combat usually with swords.

LFSR. Also called Linear-feedback Shifter Register, is a shift register that forms a feedback channel by combining some of its outputs in exclusive-OR combinations and it is widely used to generate a random number of 1s and 0s as pseudorandom pattern generators.

LOOT/s. Is an item where it can be a currency, spells, equipment and weapons. Loot is meant to reward the player for progressing in the game, and can be of superior quality to items that can be bought.

Pseudorandom. or Pseudo-Random. A sequence of numbers is one that appears to be statistically random, despite having been produced by a completely deterministic and repeatable process. One of the two RNG that will be compared.

Python. Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose

language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems.

Random Number. A number chosen from a pool of finite or infinite numbers that has no way of predicting patterns. RNG outputs.

Random Number Generator. Short for RNG. A system that can generate one or many random numbers within a defined scope. Random Number Generators can be hardware based or Pseudo- Random Number Generators.

RPG. Also called a Role-Playing Game, is a game in which players assume the roles of chosen characters in a fictional world. RPG comes with different genres which have their own unique playstyle.

Scons. SCons is an Open Source software construction tool. Think of SCons as an improved, cross-platform substitute for the classic Make utility with integrated functionality. In short, SCons is an easier, more reliable and faster way to build software.

SDL. Also known as Simple DirectMedia Layer is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D.

Seeds. Are random variables that specify the start point mostly when a computer generates a random number sequence. Deterministic which means it produces random numbers with a set of rules so it is predictable.

Sprites. Sprites are images that represent game assets. Player characters, enemies, projectiles, and other items are all called sprites (more on sprite types to come). Thus, sprites appear everywhere in games, including the title screen, within game levels, and even the game over screen.

TestU01. Is a software library in ANSI (American National Standard Institute) C language with a collection of utilities for the empirical statistical testing of uniform random number generators.

TCG. Also called Trading Card Game, is a type of card game that mixes strategic deck building elements and collecting cards by earning or interacting with different players.

TRNG. Also known as True Random Number Generators, is a device that generates random numbers using physical processes rather than algorithms.

Chapter 2

METHODOLOGY

This chapter presents the research design, participants, instrumentation process, data gathering procedures and data analysis tools employed in data treatment.

Research Design

The researcher used the developmental design method. The researchers aimed to understand the differences and similarities between Pseudo-Random Number Generator and Uniform Distribution Number Generator. The researchers made an application implementing both Pseudo-Random Number Generator and Uniform Distribution Number Generator. After finishing the important details in the application, the researchers conducted a survey for the participants to test which number generator they prefer to use. The researchers created two versions of game using two different number generators and ascertained each result in which random generator will have fewer patterns and more replayability in a game.

This study determined the differences between Pseudo-Random Number Generator and Uniform Distribution Number Generator. These two random generators affect the gameplay and which are the most successful random outputs between these two generators. Using both Pseudo-Random Number Generators and the Uniform Distribution Random Number Generator, the researchers searched for an application that they used in creating the game and the software application that they chose is the Godot which is a game engine oriented towards both 2D and 3D game development.

The game engine focuses on providing a well-rounded set of tools for development – including a built-in code editor, a graphics rendering engine, audio playback tools, animation tools, and more. After the previous tests are done, the participants find out what Random Number Generator are they comfortable to use.

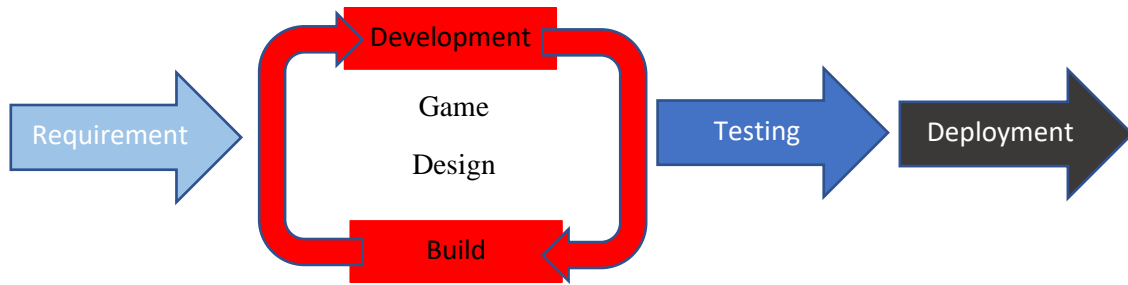


Figure 3. *RAD Model*

Requirement Stage is where the researchers used a personal computer in creating the game with the features — Intel i5 2.67ghz and 8GB RAM. Godot will be used as the game engine for the research to produce the game. The researchers created the Cho-han, the first design of the first set of games, in order to become familiar with the aforementioned Game Engine. The researchers have the knowledge that Godot speaks a language that is part of its system. Dev-C++ was used by the researchers as their code development environment, creating a code to test the algorithm of both Pseudo-Random and Uniform Distribution Generator, if the code will function. The researchers chosed Visual Studio in generating codes since it comes with built-in support for IntelliSense code completion, sophisticated semantic code understanding and navigation, and code refactoring, which they will employ for serious coding. The researchers used Simple Direct Media Layer to manage the audio and background music of the game. The next

stage is the Game Design, the researchers will start developing the game after attaining all of the necessary informations and requirements. When creating the game, the researchers decided that the major goal would be to generate random results. The next is the Testing Stage, during the testing phase, the selected individuals will play the game that the researchers created in order to determine which Random Number Generator works best for the game. The last stage is the Deployment Stage, along with comparing Pseudo-Random Number Generator and Uniform Distribution Random Number Generators after testing the game with the selected participants, the researchers also hope to demonstrate the advantages of these generators using a game and experiments using data outputs. After analyzing the outputs from the test, the results from the testing phase will be in the conclusion.

Participants of the Study

The participants of the study are the Computer Science students of the Nueva Vizcaya State University and Saint Mary's University ranging from first to fourth-year students. Students who took a Bachelor of Science in Computer Science fit this research because they have technical knowledge about technology and computer algorithms that involve a system that produces random outputs. The researchers considered someone who are at least have a knowledge about technology and computer algorithms which is close to Computer Science students, as the research subject.

Table 2. *Table of Participants*

PARTICIPANTS					
University	1st Year	2 nd Year	3 rd Year	4 th Year	BS in Computer Science
NVSU	_____	4	2	_____	6
SMU	_____	_____	_____	4	0

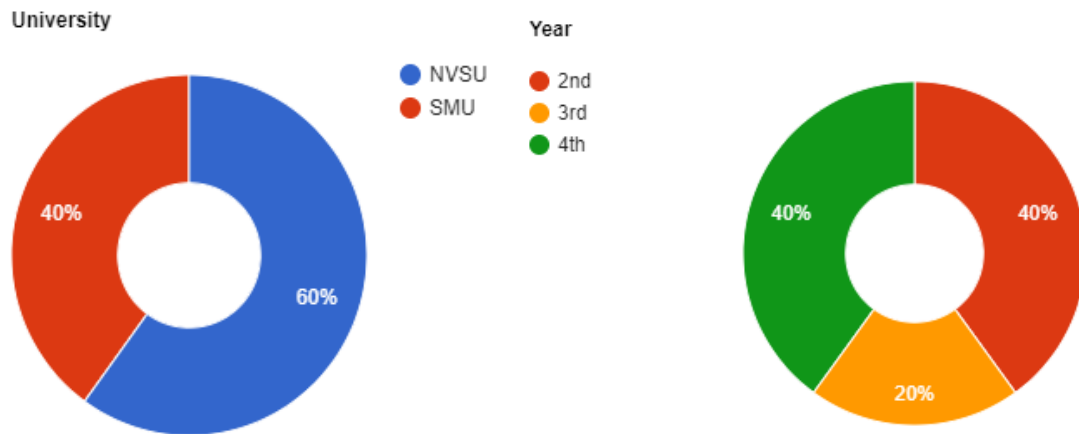


Figure 4. *Population and Year*

The total population was considered with a total of 10 first to fourth year students; where 20 percent of the participants are 3rd year, 40 percent are from 2nd year and 40 percent from 4th year. From the total population of actual respondents, 60 percent of the participants are from Nueva Vizcaya State University and 40 percent of the participants are from Saint Mary's University.

Instrumentation

The researchers conducted a survey where the participants are able to test which number generator they prefer to use. The survey is simple enough for people with knowledge around technology and gaming. Survey questionnaires are collected to determine and classify information so that the data gathered from respondents are analyzed and summarized. The researchers employed dichotomous type questions in the initial section of the questionnaire which contains questions with two possible responses. The researchers utilized this type of questions to decide the comparison between two RNGs that the respondents will choose.

After conducting the survey, the researchers analyzed and made a conclusion about which is better when it comes to outstanding results. The researchers created two different number generators and ascertained each result in which random generator will have fewer patterns and more replayability in a game. The preferred number generators of the respondents are requested to function as a moderator or intervening variable of the research. *See (Appendix B)* for lists of surveys.

Data Gathering Procedures

The researchers sent a request letter for the actual Computer Science students and requested permission to conduct the game. The researchers contacted the responders by sending a letter of consent to the game and surveys in order to gain their participation in the study. The researchers gathered the data for the game's algorithms. The researchers needed methods, basic, and sample source codes to be familiar with these RNGs. The researchers' sources are from the studies that focused on what algorithms they used and

results. Following the survey, the researchers studied the data and came to a judgment regarding which method is superior in terms of achieving spectacular results. Immediately after the end and analysis of the survey, the researchers called each and every one of the participants to test a new game, which is now the prototype, to check if the subjects still prefer their first choice Random Number Generator.

For additional data the researchers conducted tests on different participants. Data among the participants included their preferences, opinions, feedback, and what must be done to improve on the last game. The data gathered is discrete and must be closed to each and every participant for unbiased recommendation. The researchers did a little internet research about efficient algorithms and how code works.

Data Analysis

The starting phase of this research analyzed the difference of these RNGs in mind. Especially on how they differ in terms of algorithm, codes, and results. Additionally, the researchers compared as well, as this is a comparative study. A game is created for each of the RNGs stated in this research as: Pseudo-Random Number Generator and Uniform Distribution Random Number Generator. Once we gathered data from the test runs and from participants, the researchers made a percentage of recommendation and made a graph for every RNGs. The researchers focused on how the first two games and how the data played out and probably applied them both to the final prototype.

Table 2 shows the preferred RNG of the participants of the Nueva Vizcaya State University and Saint Mary's University. Percentage taken based on Table 3.2 "Preference". 10% percent per participant's preference.

Table 3. Recommendation Percentage*PRNG – Pseudo-Random Number Generator**UDRNG – Uniform Distribution Random Number Generator*

GAME RECOMMENDATION TABLE		
GAME DESIGN	MAJORITY	
Questions:	YES	NO
Do you like “Dise Game” user interface?	100%	0

GAMEPLAY	MAJORITY	
Preferences	PRNG	UDRNG
Questions:		
Which “Dise Game” is more appealing?	90%	10%
Which “Dise Game” is more repetitive? (ex. duplicate dices)	80%	20%
Which “Dise Game” has the faster response? (ex. hiccups when interacting with a button)	50%	50%
Which “Dise Game” is your preference?	40%	60%
Do you think that “Dise Game” design is confusing?	100%	0
Do you like “Dise Game” overall design?	100%	0

The researchers created a game that consists of two different random algorithms; Pseudo-Random and Uniform Distribution. The game was tested by 10 participants and each one chose their preferences (Pseudo or Uniform). The results from the participants preference are recorded for the researchers to come up with a code with a combined algorithm. Based on the table 40 percent of the participants chose Pseudo-Random. On the other hand, 60 percent of the participants chose Uniform Distribution Generators. The researchers calculated the number of participants that chose their preference and divide by the total number of participants times 100 $((\text{choice}/\text{total participants}) * 100)$ to get the percentage. The researchers have concentrated on the two games and the data collected

Chapter 3

RESULTS AND DISCUSSION

I. The avatier game application using Cho-han.

This part discusses the running code and the results pertaining to the comparative on pseudo-random and uniform distribution generators in games. Significant differences were also included considering the moderator variables used for this research.

Running Code

For Pseudo:

```
#include<iostream>

#include<chrono>

using namespace std;

int main()

{

    int seed = chrono::system_clock::now().time_since_epoch().count();

    int output = 0;

    srand(seed);

    for (int c = 0; c < 10; c++)

    {

        //will output from 1 to 6;

        output = rand()%6+1;

        cout << output<<endl;

    }

}
```

//int seed = value of the epoch unix time including nanoseconds; better than time null

For Uniform:

```
#include<iostream>
```

```
#include<random>
```

```
#include<chrono>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int seed = chrono::system_clock::now().time_since_epoch().count();
```

```
    int output=0;
```

```
    default_random_engine gen(seed);
```

```
//will output from 1 to 6
```

```
    uniform_int_distribution<>distribution(1, 6);
```

```
    for (int c = 0; c < 1000; c++)
```

```
    {
```

```
        output = distribution(gen);
```

```
        cout << output;
```

```
    }
```

```
}
```

//int seed = value of the epoch unix time including nanoseconds; better than time null

//default_random_engine = holds the value of the seed as the seed for distribution

This source will be our running code “or” based from. Different tests vary with different code but it is mostly similar.

II. The two game-version embedding the two algorithms.

Difference in Algorithm

For Pseudo:

Pseudo-Random generates sequence of numbers through a formula. And it heavily relies on the seed to get the random output. The algorithm for getting the random output is very similar to the Linear Congruential Generator (LCG) algorithm.

Algorithm for the LCG:

let $x[i] = \text{seed}$ let $m = \text{modulo or the range of numbers}$

let $a = \text{multiplier}$ let $c = \text{increment}$

$x[i+1] = (a * x[i] + c) \% m$

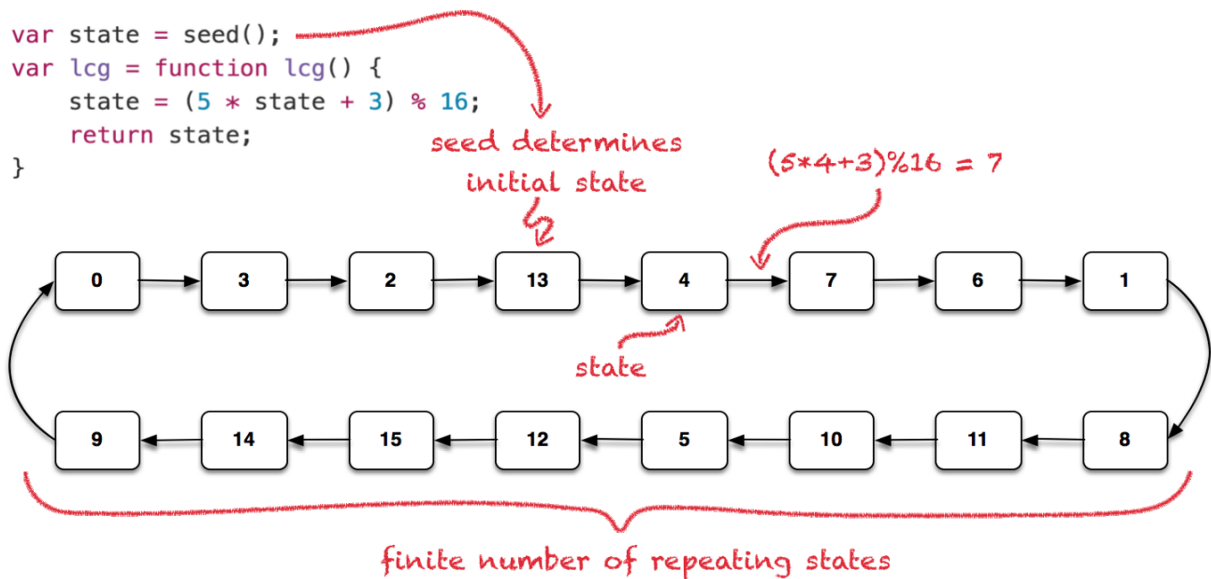
An example is shown below:

let $x[i] = 13$ // initial seed

$x[i+1] = (5 * x[i] + 3) \% 16$;

with $x[i]$ is equal to 4 and so on...

Refer to the image below for the representation of LCG:



Though it follows a formula the outputs can be looped if the seed is defined. We now have the idea on how LCG works, let us move on to Pseudo-Random.

Let us see the code ***For Pseudo*** and get the method rand(). This is the most known for using the fast and deterministic RNG.

Inside the rand() is an algorithm very similar to LCG:

//Initialization

int seed = 1; //we take out the original seed and change into 1

int holder = 0; //added a new variable that holds the new seed

//Method

*holder = (214013 * seed + 2531011); //holder then stores the value for the NEXT*

seed when looped // Every compiler has their own multiplier and increment. We are using the Microsoft Visual C++ source

seed = ((holder) >> 16) & 0x7fff; //we take the holder value and shift it to right by 16 after that we calculate the bits and 0x7fff (32767 or RAND_MAX) // Alternative to the shifting is dividing the holder to 65535

See (Appendix G) for the full list of source compiler.

When we run it through a code, this is how it displays:

Holder: 2745024

Random Output: 41

Seed = 2745024:

Holder: -937167229

Random Output: 18467

Same output when we call rand() function normally.

For Uniform:

Uniform does not follow an algorithm unlike Pseudo-Random or LCG that relies on the seed for producing random outputs. Uniform gets its random outputs through probabilities. However, a seed is called to not produce the same output as before.

Let us see the code ***For Uniform***

//Initialization

default_random_engine gen() //this is a engine to hold our seed in || error if empty

uniform_int_distribution<>distribution(1, 6) //this will initialize the range of numbers right now it is 1 to 6

//Method

distribution(gen); //we call distribution with the parameter of the gen() which then give the random output through 1 to 6

float pro = 1.0/(distribution.max() - distribution.min() +1); //this calculates the probabilities of the numbers that will be the output || assume that 1.0 is 100% and we divide it with the value of total possible numbers

When we run it through a code, this is how it is displays:

Output: 1

Total Possible Numbers: 6

Probabilities: 0.166667%

1 to 17:

Total Possible Numbers: 17

Probabilities: 0.0588235%

III. The Pseudo-Random Number Generator and Uniform Distribution Number Test.

Fast Test

A fast and responsive game is somewhat an advantage to people who likes to play games. Let us say that if someone push a button, the game will call that function and give response to the player. This is very important to when playing real-time with other players.

We conducted this test in 3 PC which they differ in hardware. Global running code is used.

where:

Start = From when the code starts End = From when the code ends

Elapsed Time = From when the code ends relative to Start (End – Start)

0.001 = 1ms

0.010 = 10ms

0.100 = 100ms

PC: A = Intel i5 2.67ghz and 8gb Ram B = AMD Ryzen 3.60ghz and 8gb Ram

 C = Intel (R) Celeron(R) 1.80GHz and 2.00GB RAM

For Pseudo: (A)

Looping ten times:

Start: 0.001s

End: 0.002s

Elapsed Time: 0.001s

Looping hundred times:

Start: 0s

End: 0.046s

Elapsed Time: 0.046s

Looping thousand times:

Start: 0s

End: 0.651s

Elapsed Time: 0.651s

(B)

Looping ten times:

Start: 0.001s

End: 0.002s

Elapsed Time: 0.001s

Looping hundred times:

Start: 0.041s

End: 0.107s

Elapsed Time: 0.066s

Looping thousand times:

Start: 0.04s

End: 0.737s

Elapsed Time: 0.697s

(C)

Looping ten times:

Start: 0.001s

End: 0.003s

Elapsed Time: 0.002s

Looping hundred times:

Start: 0.001s

End: 0.033s

Elapsed Time: 0.032s

Looping thousand times:

Start: 0.001s

End: 0.208s

Elapsed Time: 0.207s

For Uniform:

(A)

Looping ten times:

Start: 0s

End: 0.001s

Elapsed Time: 0.001s

Looping hundred times:

Start: 0.001s

End: 0.047s

Elapsed Time: 0.046s

Looping thousand times:

Start: 0.001s

End: 0.617s

Elapsed Time: 0.616s

(B)

Looping ten times:

Start: 0s

End: 0.002s

Elapsed Time: 0.002s

Looping hundred times:

Start: 0.038s

End: 0.094s

Elapsed Time: 0.056s

Looping thousand times:

Start: 0.001s

End: 0.648s

Elapsed Time: 0.647s

(C)

Looping ten times:

Start: 0.001s

End: 0.002s

Looping hundred times:

Start: 0.002s

End: 0.031s

Elapsed Time: 0.001s

Elapsed Time: 0.029s

Looping thousand times:

Start: 0.001s

End: 0.232s

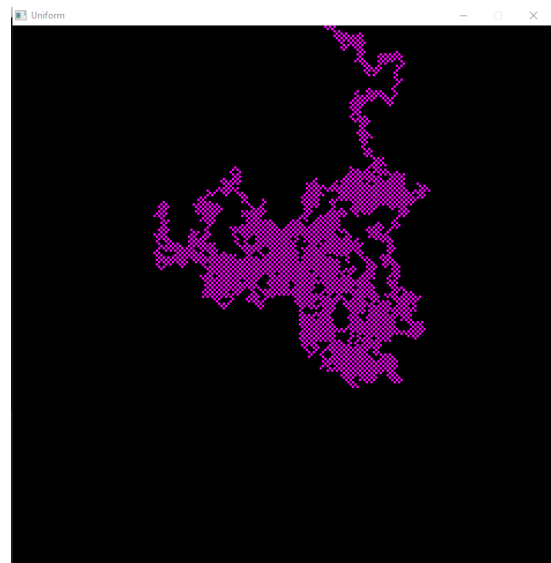
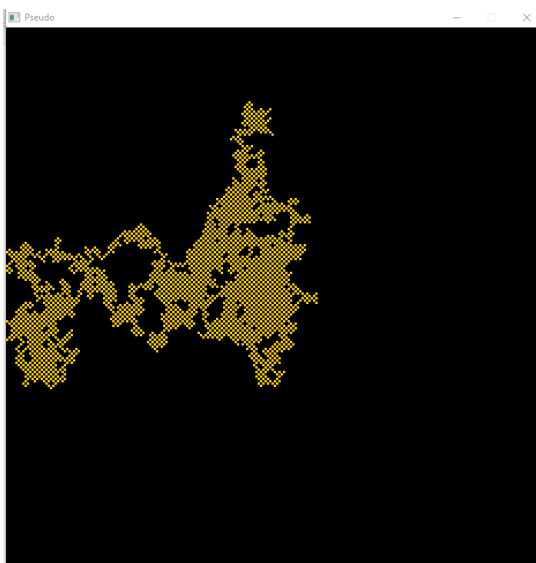
Elapsed Time: 0.231s

Random Walk

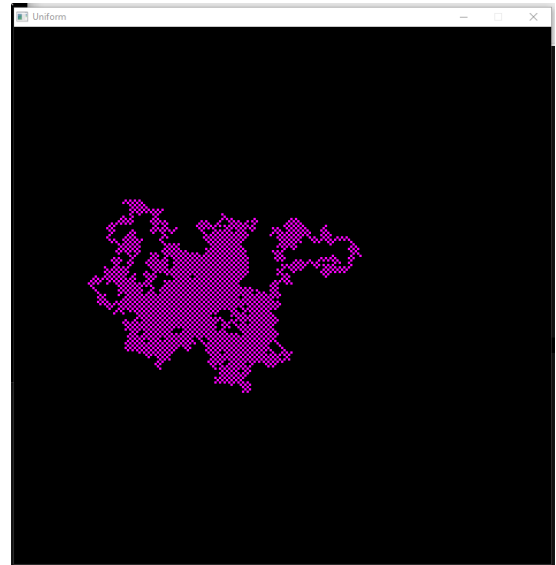
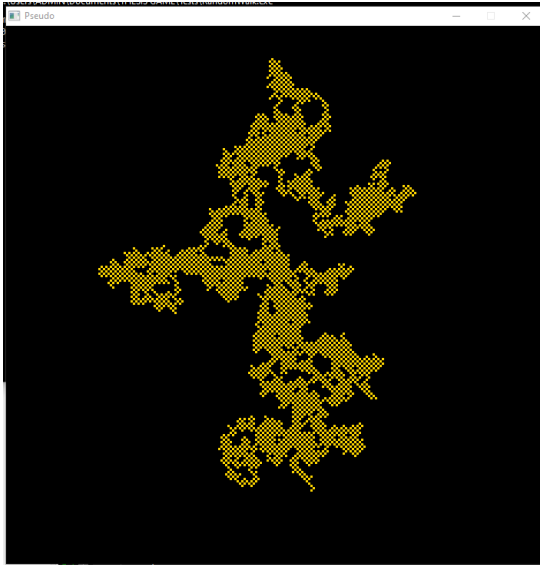
Instead of displaying the countless loop of numbers, we made a program that represents what they call a random walk. Basically, the RNG functions the same but in a different way of producing random outputs. This test will also show how they differ from each other. Each program runs are looped 10,000 times to see a pattern and each number 1-4 is correspondent to where the walk should go. If they go over the screen, the walk then will reset back to default place. *See (Appendix H)* for the code.

- Pseudo-Random
- Uniform Distribution

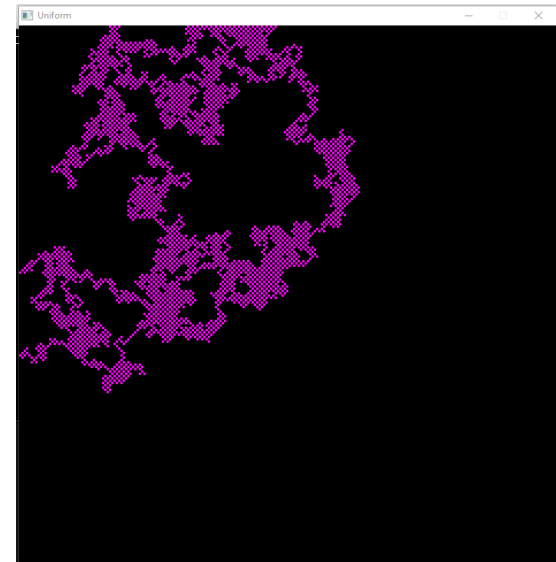
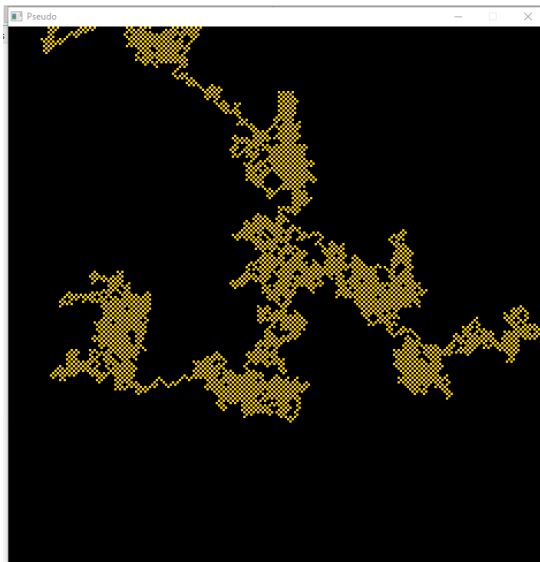
First Run:



Second Run:



Third Run:



The runs show that Pseudo-Random spreads everywhere than Uniform Distribution. This means that outputs have a possible correlation with the last output. Uniform Distribution on the other hand, spreads evenly, that is because of the numbers have the same probability of being the output regardless of the last output.

Chapter 4

SUMMARY OF FINDINGS, CONCLUSION AND RECOMMENDATIONS

This chapter summarizes the findings and presents the conclusions and recommendations drawn from the study.

Summary of Findings

The following is a summary of the results of the study which was presented in detail in the previous chapter:

1. The researchers were able to create two games implementing the Cho-han mechanics.
2. The researchers were able to incorporate the two algorithms, Pseudo Random Number Generator and Uniform Distribution, separating in both games.
3. The researchers were able to test the two algorithms by their own and by using Fast Test and Random Walk.

Conclusion

In this part, the conclusions derived from the findings from the comparative study on Pseudo-Random and Uniform Distribution Generator in games were described. The conclusions were based on the purpose, research questions, and results of the study.

1. Cho-han mechanics was applied to the Dice Game to test the efficiency between Pseudo-Random Number Generator and Uniform Distribution Number Generator.
2. In response to the comparison between Pseudo-Random Number Generator and Uniform Distribution, Uniform Distribution Random Number Generator is faster after conducting the fast test in two personal computer with different hardware.

While in random walk, the results demonstrate that Pseudo-Random is more widely distributed than Uniform Distribution. This indicates that outputs may be related to the previous output. Uniform Distribution, on the other hand, spreads equally since all numbers have the same chance of being the next output, regardless of the previous result.

3. Sixty percent of the participants preferred Uniform Distribution Random Number Generator, on the other hand, forty percent of the participants favoured to Pseudo-Random Number Generator.

Upon exploring the situation from multiple perspectives, the overall results and findings pointed out Uniform Distribution Random Number Generator as preferred compared to Pseudo Random Number Generator.

Recommendations

Based on the findings and conclusions made, the researcher recommends the following:

1. Creation of a game where in it will have an own algorithm choosing Pseudo-Random Number Generator and Uniform Distribution Number Generator.
2. That other students will use this study to further improve the comparison of other gaming algorithms.

REFERENCES

- Baglin, S. (2017). Random Numbers and Gaming. Retrieved from: <http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1006&context=art108>
- Biebighauser, D. (2000). Testing Random Number Generators. REU Summer. Retrieved from: <http://facweb.cs.depaul.edu/sjost/csc433/documents/testing-rngs.pdf>
- Brückner, S., Kurabayashi, S., Sato, Y., & Waragai, I. (2019). Analyzing Random Reward System Mechanics and Social Perception. DiGRA. Retrieved from: http://www.digra.org/wp-content/uploads/digitalibrary/DiGRA_2019_paper_169.pdf
- Davis, Z. (1981-2022). Pseudo-random numbers. Retrieved from: <https://www.pcmag.com/encyclopedia/term/pseudo-random-numbers>
- Dunn, W. L., & Shultis, J. K. (2012). Pseudorandom Number Generators. In Exploring monte carlo methods 47–68. Academic Press. Retrieved from: <https://doi.org/10.1016/B978-0-444-51575-9.00003-8>
- Gupta, S. (2018). Random numbers in software-based games. Random Numbers In Software Based Games. Retrieved from: <https://medium.com/@exploke6/random-numbers-in-software-based-games-b0be384238e3>
- Haahr, M. (2019). RANDOM.ORG - Introduction to Randomness and Random Numbers. Random.org; RANDOM.ORG. <https://www.random.org/randomness/>
- Hong, S. L., & Liu, C. (2015). Sensor-Based Random Number Generator Seeding. IEEE Access. Retrieved from: <https://doi.org/10.1109/ACCESS.2015.2432140>
- L'Ecuyer, P. (2005). Uniform Random Number Generation. University of Montreal. Retrieved from: <https://www.iro.umontreal.ca/~lecuyer/myftp/papers/horms.pdf>
- Lawnik, M. (2017). Generation of Pseudo-Random Numbers with the Use of Inverse Chaotic Transformation. Open Mathematics. Retrieved from: <https://doi.org/10.1515/math-2018-0004>
- Marsaglia, G. (2003). Seeds For Random Number Generators. Communications of the ACM, 46(5), 90-93. Retrieved from: <https://doi.org/10.1145/769800.769827>
- Nielsen, R., & Grabarczyk, P. (2019, October 11). Are Loot Boxes Gambling? Random reward mechanisms in video games [Review of Are Loot Boxes Gambling? Random reward mechanisms in video games]. Retrieved from <https://www.semanticscholar.org/paper/Are-Loot-Boxes-Gambling-Random-reward-mechanisms-in-Nielsen-Grabarczyk/6046891c1c0c4ac6e707c6de8f5c68dc5674cf46>

- Owen, A. (2013). Uniform random numbers. Retrieved from: <https://artowen.su.domains/mc/Ch-unifrng.pdf>
- Rajan, R. (2019, February 2). What is the code for the rand() function which generates random numbers in C language? Retrieved from <https://www.quora.com/What-is-the-code-for-the-rand-function-which-generates-random-numbers-in-C-language>
- Rangineni, S. (2011). Cryptographic Analysis of Random Sequences. Oklahoma State University. Retrieved from: https://shareok.org/bitstream/handle/11244/8230/Rangineni_okstate_0664M_11591.pdf
- Sankaran, K. K., & Jayakumar, K. (2016). A New Extended Uniform Distribution. *International Journal of Statistical Distributions and Applications*, 2(3), 35. Retrieved from: <https://doi.org/10.11648/j.ijstd.20160203.12>
- Sharifian, S. (2016). Random Number Generation using Human Gameplay. University of Calgary, Calgary, AB. Retrieved from: <https://doi.org/10.11575/PRISM/27524>
- Schardon, L. (2021, September 15). What is godot? A guide to 2D & 3D game development. Retrieved from: <https://gamedevacademy.org/what-is-godot/>
- Selinger, P. (2007). The GLIBC random number generator. Peter Selinger: The GLIBC pseudo-random number generator. Retrieved from: <https://www.mathstat.dal.ca/~selinger/random/>
- Unitychain (Ed.). (2019). Provable randomness: How to test rngs. Provable Randomness: How to Test RNGs. Retrieved from: <https://medium.com/unitychain/provable-randomness-how-to-test-rngs-55ac6726c5a3>
- Vattulainen, I., Kankaala, K., Saarinen, J., & Ala-Nissila, T. (1993). A Comparative Study of Some Pseudorandom Number Generators. *Computer Physics Communications*. 86. 209-226. Retrieved from: [https://doi.org/10.1016/0010-4655\(95\)00015-8](https://doi.org/10.1016/0010-4655(95)00015-8)
- Wang, L., & Cheng, H. (2019). Pseudo-random number generator based on logistic chaotic system. *MDPI*. Retrieved from: <https://doi.org/10.3390/e21100960>
- Widynski, B. (2020). Squares: A Fast Counter-Based RNG. *ArXiv*, Retrieved from: <https://abs/2004.06278>
- Wikimedia Foundation. (2022). Linear Congruential Generator. Wikipedia. Retrieved from: https://en.wikipedia.org/wiki/Linear_congruential_generator

APPENDICES

Appendix A

Participant's

Data

Table below is each of our participants data including their courses, year, and University where they are currently studying.

Table 3.2 *Participant's Data*

Name	Preference	Age	Course	Year	School
Kevin Raña	Pseudo	19	BSCS	2 nd year	NVSU
Paul Padilla	Uniform	20	BSCS	2 nd year	NVSU
Winford Cabuena	Uniform	20	BSCS	3 rd year	NVSU
John Luis Cabauatan	Uniform	21	BSCS	3 rd year	NVSU
Ryan Christian Ramos	Uniform	20	BSCS	2 nd year	NVSU
Julian Serquiña	Pseudo	19	BSCS	2 nd year	NVSU
Franz Chua	Pseudo	22	BSOA	4 th year	SMU
Jester Mauricio	Uniform	22	BSA	4 th year	SMU
Latrell Gelacio	Uniform	22	BSN	4 th year	SMU
Reo Lance Villanueva	Pseudo	22	BSEE	4 th year	SMU

Appendix B

Game Survey Table

Dear Respondent,

We are 4th year BSCS students and currently enrolled in NVSU (Thesis 2). We would like to request opinions from you regarding the satisfaction you experience with two of our games – “Dise Game” with each of the RNG.

Researchers

-----oOo-----

Name: _____ Course: _____

School: _____

Instructions: Draw a CIRCLE or a CHECK MARK in the corresponding column of your answer. If neither of your answers are in the columns – just leave it blank.

PRNG – Pseudo-Random Number Generator

UDRN – Uniform Distribution Random Number Generator

Game Design	Yes	No
Do you like “Dise” user interface?		
Do you think that “Dise” design is confusing?		
Do you like “Dise” overall design?		
Gameplay	PRNG	UDRNG
Which “Dise” is more appealing?		
Which “Dise” is more repetitive? (ex. duplicate dices)		
Which “Dise” has the faster response? (ex. hiccups when interacting with a button)		
Preference		
Which “Dise” is your preference?		

Appendix C

Game Design with GDScript

As stated in our Introduction that we will be using Godot as our Game Engine to create a game. To be familiar with the said Game Engine, the researchers tried using the Engine by making the first design of our first set of games which is the Cho-han. One thing that we know is that Godot uses a kind of language that it is built in its system.

We settled the game design to be simple but creative for using to conduct the recommendation. Below is the design:

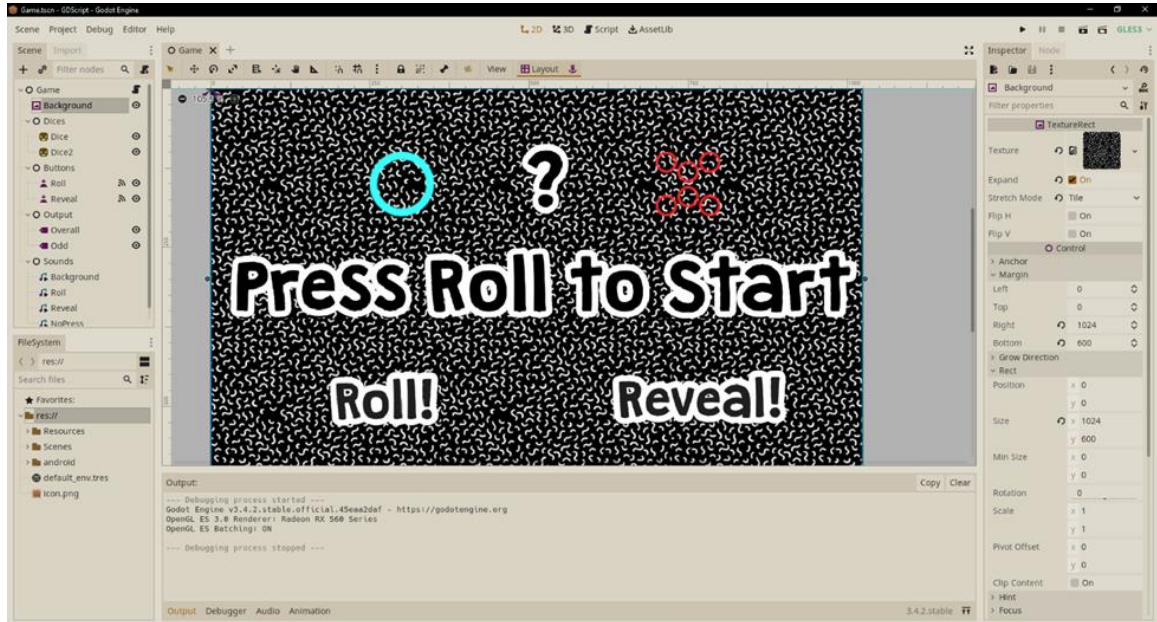


Figure 3.1 *Game Design in Godot*

The dashboard contains all of the scene design which consists of [1] two dices, [2] a label that shows the addition of the two dices rolled (?) [3] a label that shows what kind of number is: odd or even (Press Roll to Start) [4] two buttons that is called Roll and Reveal that have their own corresponding function.

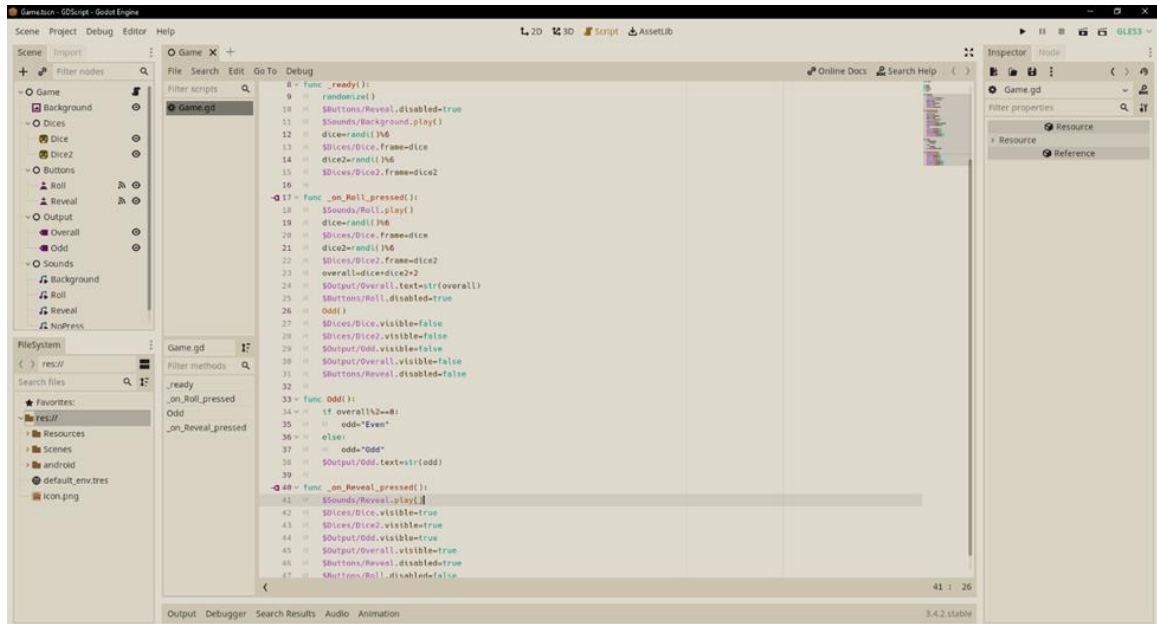


Figure 3.2 Code.

Note: GDScript is not what the language we will use. This is only a test and getting familiar with the engine.

The code starts with creating 4 variables. Function **_ready()**, it means that the scene is triggered when it is played. Inside it is the **randomize()** function that acts as a random initial seed for producing random numbers. A green code that starts with this (\$) sign means that you are calling one of the nodes list that can be seen on the left. The **randi()** function is the most important part of this code, it produces random output over six sided dice, thus $\text{randi()} \% 6$.

We go on the next function, the **_on_Roll_pressed()**, this means that it is binded to the Roll button and when it is pressed it does what is inside of the function. For this, we used the $\text{randi()} \% 6$ function again to display the dice in the game itself, so the color sided dice is based on the random number and frames defines what side of the side is ex. 1 is one red circle, 2 are the two green circles, etc. We also called the **Odd()** method that

contains a conditional statement that defines what kind of number is the variable **overall** with the formula of $\text{overall} \% 2$ and if it has a remainder it is Odd.

Same as the Roll Button, the `_on_Reveal_pressed()` reveals the hidden Sprites that when Roll is pressed it hides them for this is the mechanics of how the game Chohan is played. The dealer then roll the dices and at the right time he/she covers the dices with a cup. He/She then asks the player/s to guess if the combined dices is Even or Odd.



Figure 3.3 *Game Design in Fullscreen*

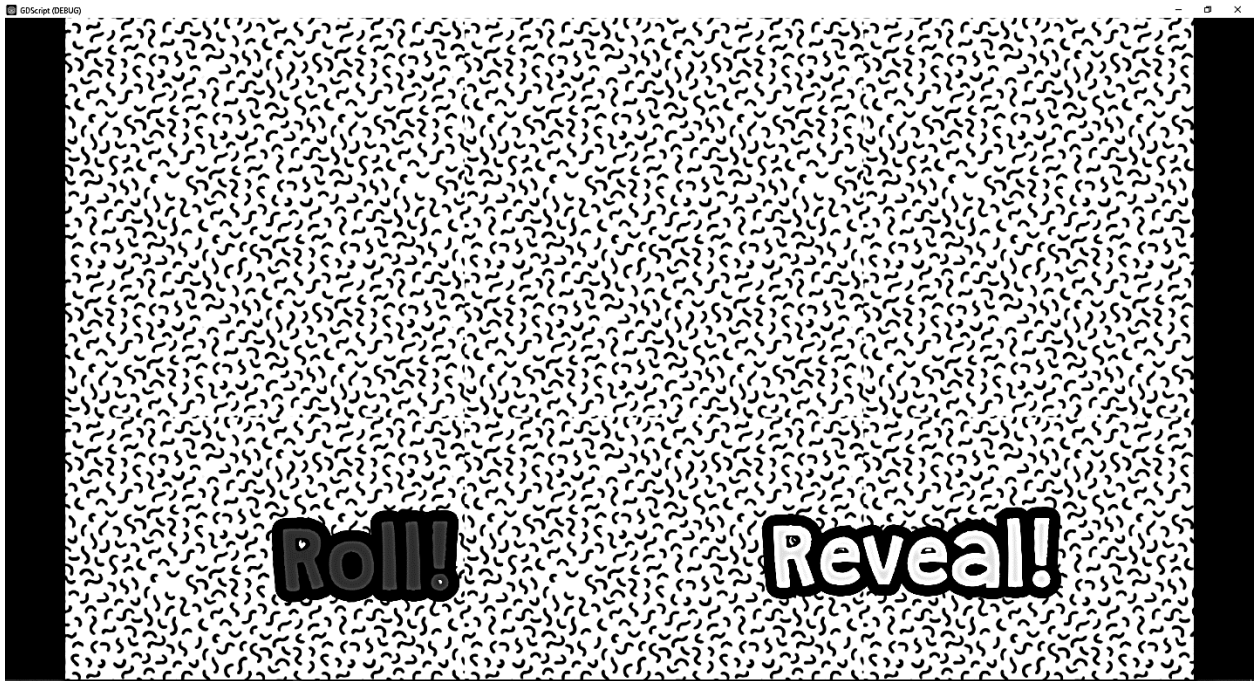


Figure 3.4 *Scenario – Clicked Roll*

Hides all the sprites and labels to ensure the mechanics on how to play Cho-han. After pressing Roll it disables the button itself until the Reveal button has been clicked. By this time the game already have the random numbers and it will be presented by the two dices

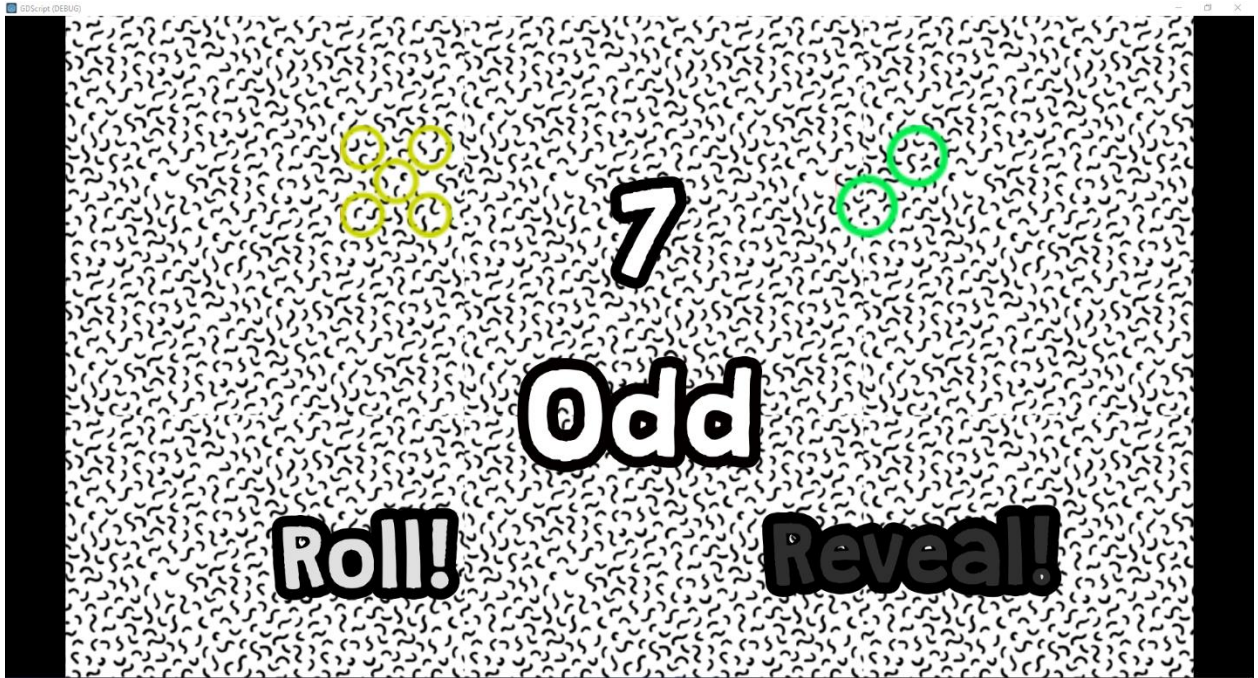


Figure 3.5 *Scenario – Reveal Pressed*

Appendix D

Integrating C++ in Godot

Ways to Integrate C++ into Godot

Integrating C++ into the game engine is not that straightforward easy but there is some ways on how to do it; [1] First is creating an external dynamic library (.dll) for Godot to initialize which they called **GDNative**, [2] or rebuilding the whole game engine with custom modules – modules which contains the code for the C++.

So, for the fast and reliable way we need to use [1]. It does not need to rebuild the whole engine and wait for a long time to just bake a new engine. Mistakes and errors in the code within the modules means rebuilding the game engine itself.

Integrating C++: Setup

GDNative is not limited to just C++ but it can also integrate C, D, Python, Rust and many more. To setup our integration, we need some plugins and libraries by using it we can create some C++ codes. Things we need: Python (latest version), any C++ compiler or IDE (we are using Visual Studio), Godot C++ bindings, and Godot headers.

1. Installing Scons.

We need python for this to work. With python installed, open up command prompt and write “*pip install scons*” – which will download scons – a software use to build other softwares (more details on Definition of Terms). If there is a prompt that a new version of pip means that you are not using the latest version of the Python. Go ahead and copy the command it gives and it will reinstall automatically.


```

Command Prompt
Microsoft Windows [Version 10.0.19042.1586]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>
C:\Users\ADMIN>python
Python 3.10.3 (tags/v3.10.3:a342a49, Mar 16 2022, 13:07:40) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

C:\Users\ADMIN>pip install scon
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scon in c:\users\admin\appdata\roaming\python\python310\site-packages (4.3.0)
Requirement already satisfied: setuptools in c:\program files\python310\lib\site-packages (from scon) (58.1.0)

C:\Users\ADMIN>scon

scon: *** No SConstruct file found.
File "C:\Users\ADMIN\AppData\Roaming\Python\Python310\site-packages\SCons\Script\Main.py", line 944, in _main

C:\Users\ADMIN>

```

Figure 4.1 *Installing Scons*

2. Building Godot Bindings

We need Godot C++ Bindings and Godot headers for this to work. After extracting these two, copy all of the contents of godot headers into the folder of godot cpp bindings – godot cpp is dependent on godot headers without it, it cannot build bindings. Assuming all of the above are done, open up command line within the file of godot cpp bindings itself or just put the directory of the file. Write “*scons platform=windows generate_bindings=yes*” we called scon for binding godot cpp bindings into one library. A bin folder was created during the process, containing the library named as **libgodot-cpp-windows.debug.default.lib**. This library contains files such as headers and cpp codes. This can now be linked and be used as a dependency on creating our C++ codes.

```

Select C:\Windows\System32\cmd.exe - scons platform=windows generate_bindings=yes
Microsoft Windows [Version 10.0.19042.1586]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ADMIN\Documents\THESES GAME\Game Test\godot C++\godot-cpp\scons platform=windows generate_bindings=yes.
scons: Reading SConscript files ...
src/gen: File exists
src/gen: File exists
scons: done reading SConscript files.
scons: Building targets ...
c1 /Fsrc/gen/AcceptDialog.obj /c src/gen/AcceptDialog.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AcceptDialog.cpp
c1 /Fsrc/gen/AnimatedSprite.obj /c src/gen/AnimatedSprite.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimatedSprite.cpp
c1 /Fsrc/gen/AnimatedSprite3D.obj /c src/gen/AnimatedSprite3D.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimatedSprite3D.cpp
c1 /Fsrc/gen/AnimatedTexture.obj /c src/gen/AnimatedTexture.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimatedTexture.cpp
c1 /Fsrc/gen/Animation.obj /c src/gen/Animation.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
Animation.cpp
c1 /Fsrc/gen/AnimationNode.obj /c src/gen/AnimationNode.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNode.cpp
c1 /Fsrc/gen/AnimationNodeAdd2.obj /c src/gen/AnimationNodeAdd2.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeAdd2.cpp
c1 /Fsrc/gen/AnimationNodeAdd3.obj /c src/gen/AnimationNodeAdd3.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeAdd3.cpp
c1 /Fsrc/gen/AnimationNodeAnimation.obj /c src/gen/AnimationNodeAnimation.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeAnimation.cpp
c1 /Fsrc/gen/AnimationNodeBlend2.obj /c src/gen/AnimationNodeBlend2.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeBlend2.cpp
c1 /Fsrc/gen/AnimationNodeBlend3.obj /c src/gen/AnimationNodeBlend3.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeBlend3.cpp
c1 /Fsrc/gen/AnimationNodeBlendSpace2D.obj /c src/gen/AnimationNodeBlendSpace2D.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeBlendSpace2D.cpp
c1 /Fsrc/gen/AnimationNodeBlendSpace3D.obj /c src/gen/AnimationNodeBlendSpace3D.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeBlendSpace3D.cpp
c1 /Fsrc/gen/AnimationNodeBlendTree.obj /c src/gen/AnimationNodeBlendTree.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeBlendTree.cpp
c1 /Fsrc/gen/AnimationNodeOneShot.obj /c src/gen/AnimationNodeOneShot.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeOneShot.cpp
c1 /Fsrc/gen/AnimationNodeOutput.obj /c src/gen/AnimationNodeOutput.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeOutput.cpp
c1 /Fsrc/gen/AnimationNodeStateMachine.obj /c src/gen/AnimationNodeStateMachine.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeStateMachine.cpp
c1 /Fsrc/gen/AnimationNodeStateMachinePlayback.obj /c src/gen/AnimationNodeStateMachinePlayback.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeStateMachinePlayback.cpp
c1 /Fsrc/gen/AnimationNodeStateMachineTransition.obj /c src/gen/AnimationNodeStateMachineTransition.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeStateMachineTransition.cpp
c1 /Fsrc/gen/AnimationNodeTimeScale.obj /c src/gen/AnimationNodeTimeScale.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeTimeScale.cpp
c1 /Fsrc/gen/AnimationNodeTimeSeek.obj /c src/gen/AnimationNodeTimeSeek.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeTimeSeek.cpp
c1 /Fsrc/gen/AnimationNodeTransition.obj /c src/gen/AnimationNodeTransition.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationNodeTransition.cpp
c1 /Fsrc/gen/AnimationPlayer.obj /c src/gen/AnimationPlayer.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationPlayer.cpp
c1 /Fsrc/gen/AnimationRootNode.obj /c src/gen/AnimationRootNode.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationRootNode.cpp
c1 /Fsrc/gen/AnimationTrackEditPlugin.obj /c src/gen/AnimationTrackEditPlugin.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationTrackEditPlugin.cpp
c1 /Fsrc/gen/AnimationTree.obj /c src/gen/AnimationTree.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationTree.cpp
c1 /Fsrc/gen/AnimationTreeFreePlayer.obj /c src/gen/AnimationTreeFreePlayer.cpp /TP /nologo /Z7 /O4 /EHsc /D_DEBUG /MDd /I. /Igodot-headers /Iinclude /Iinclude/gen /Iinclude/core
AnimationTreeFreePlayer.cpp

```

Figure 4.2 Bindings Process

3. Linking Libraries

Binding Process done all things fall into our Visual Studio. We recommend Visual Studio for easily linking up libraries. Basically, just put all the directories of godot headers, gen, core, and the folder which contains **libgodot-cpp-windows.debug.default.lib**. Also, it is important that your project settings is the same as the library for example is the name from above we see **debug** word, so it means change your project settings to debug or release if it named release.

We are now ready to create C++ codes by using the same project with the linked libraries from above.

Integrating C++: Game – Pseudo

After setting it all up from installing scons to linking libraries, it is time to create plugins or libraries.

We need to create a cpp code that will be recognized and start as a initial entry point to Godot. This cpp then can be reused for plugins you want to create. Every code function will be explained.

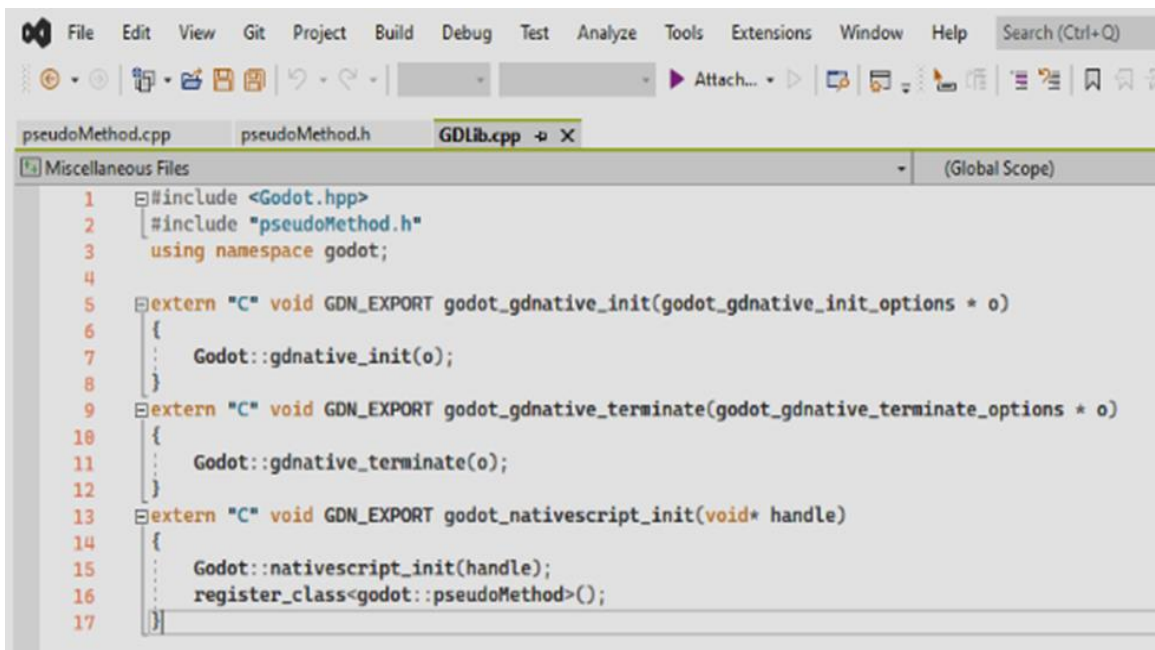


Figure 4.3 Registering Custom Classes

extern *"C"* *void* *GDN_EXPORT*
godot_gdnative_init(godot_gdnative_init_options **o)*
{Godot::gdnative_init(o);} \\ this first function is called to initialized our plugin
and calls into the bindings library and also setups the communication with Godot.

```
extern          "C"          void          GDN_EXPORT
godot_gdnative_terminate(godot_gdnative_init_options      *o)
{Godot::gdnative_terminate(o);}
```

\\ this second function is called when the plugin is unloaded and ensures the binding library cleans itself.

```
extern  "C"  void  GDN_EXPORT  godot_gdnative_init(void*  handle)
{Godot::nativescript_init(handle);
register_class<godot::pseudoMethod>();}
```

\\ this third function is important, this function includes our nativescript which is the class we are going to use for the plugin. For this we named it *pseudoMethod*.

Remember this is important for registering plugins for GDNative.

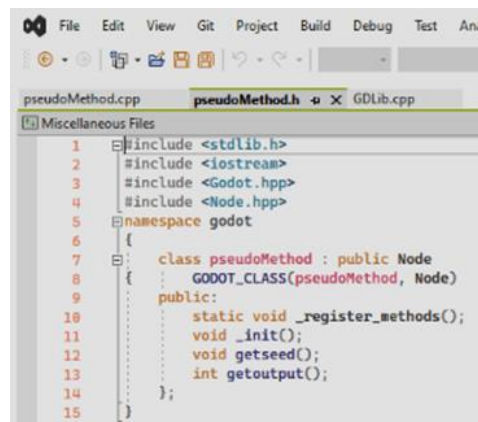


Figure 4.4 *Pseudo Header*

This header file contains all of the declaration of other header files from the previous linked library. It includes the **Node.hpp** – this meaning that this code will subclass Node.

class pseudoMethod : public Node \\ this is the name of our class we will register in the Node class.

GODOT_CLASS(pseudoMethod, Node) this will recognize that we just created a class with the name **pseudoMethod** in the Node class.

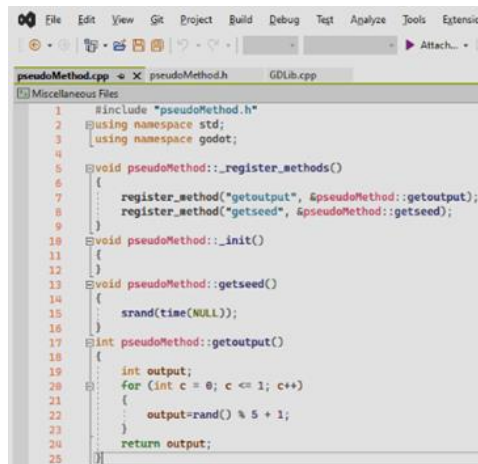
static void _register_methods(); \\ this method will find every properties and methods registered.

void _init(); \\ this method will initialized the class we are about to register.

void getseed(); \\

int getoutput(); \\ this two methods are the ones we created.

Passing declared headers are possible as long as the cpp has declared the header file that contains the declared headers.



```
1 #include "pseudoMethod.h"
2 #using namespace std;
3 #using namespace godot;
4
5 void pseudoMethod::_register_methods()
6 {
7     register_method("getoutput", &pseudoMethod::getoutput);
8     register_method("getseed", &pseudoMethod::getseed);
9 }
10 void pseudoMethod::_init()
11 {
12 }
13 void pseudoMethod::getseed()
14 {
15     srand(time(NULL));
16 }
17 int pseudoMethod::getoutput()
18 {
19     int output;
20     for (int c = 0; c <= 1; c++)
21     {
22         output=rand() % 5 + 1;
23     }
24     return output;
25 }
```

Figure 4.5 Pseudo CPP

This cpp has only one header declared which is **pseudoMethod.h**. We will not declare anymore headers because we included all headers that we are using in that header file.

*register_method(“*name of the method”), &pseudoMethod::*method that we are making);* \\ in this instance the name of our methods are **getseed** and **getouput**. This will let Godot know that we have a method it can call.

Make sure that all lines are correct and no errors, as long as you do not see any errors it is good. Now that we finished registering our methods in GDNative, we build the whole project file as a dynamic library (dll) this can now be used in Godot. Make a new godot project and create a new gdnativelibrary and choose our dll. And now we can start declaring the methods we created in Godot.

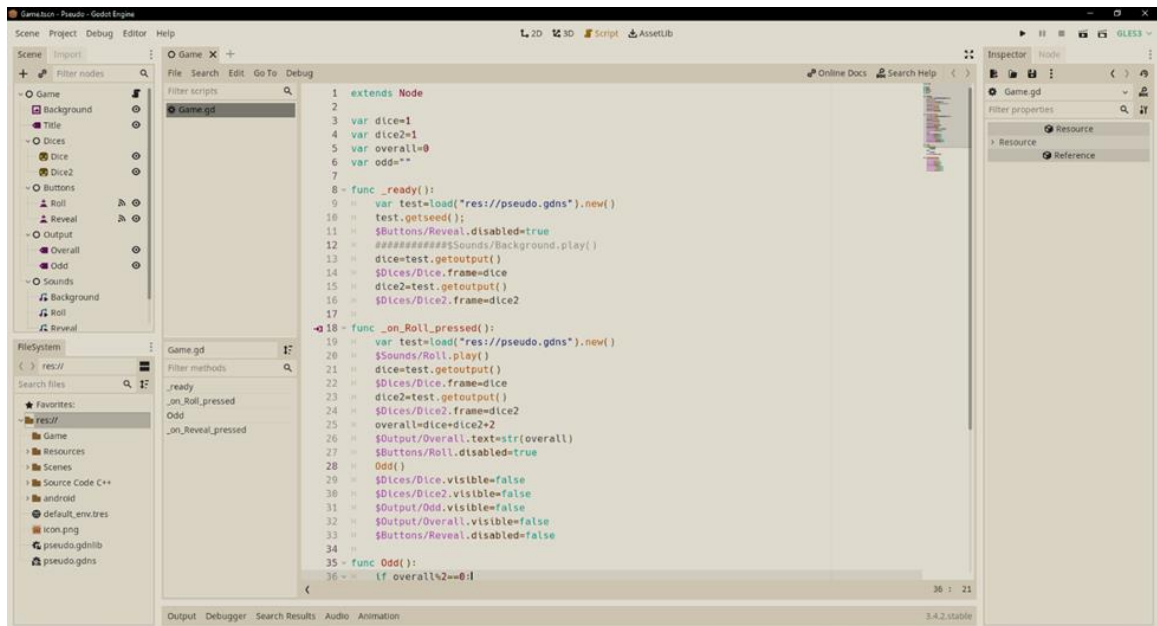


Figure 4.6 *Pseudo Game Code*

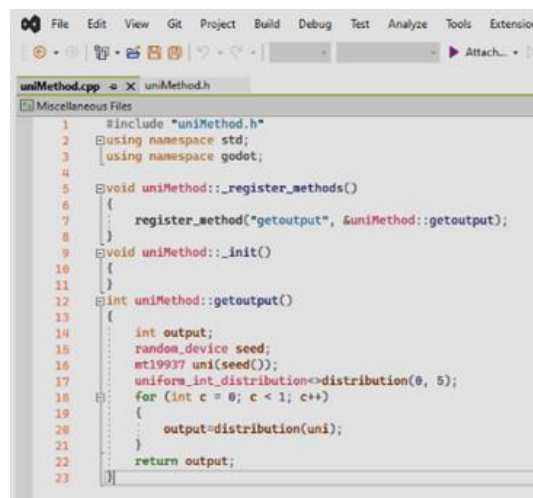
Any change is minimal prior to the Game Design with GDScript. We can just see the methods that we created in the plugin and call it to Godot.

```
var test=load("res://pseudo.gdns).new() \\ gdns is the name of our saved  
gdnativescript and must contain the dll plugin.
```

```
dice=test.getoutput() \\ We called getoutput that contains the running code for our  
Pseudo-Random Number Generator.
```

Integrating C++: Game – Uniform

We did the same way the Pseudo-Random was registered. The only thing that changed is that we remove the method **getseed** and changed the random function which classifies as a Uniform Distribution. The figure below as follows.



```
1 #include "uniMethod.h"  
2 using namespace std;  
3 using namespace godot;  
4  
5 void uniMethod::_register_methods()  
6 {  
7     register_method("getoutput", &uniMethod::getoutput);  
8 }  
9 void uniMethod::_init()  
10 {  
11 }  
12 int uniMethod::getoutput()  
13 {  
14     int output;  
15     random_device seed;  
16     mt19937 uni(seed());  
17     uniform_int_distribution<>distribution(0, 5);  
18     for (int c = 0; c < 1; c++)  
19     {  
20         output=distribution(uni);  
21     }  
22     return output;  
23 }
```

Figure 4.7 Uniform CPP

We can say that the game code is the same as in Figure 6. The code is different and only Uniform Distribution function. Changed gdnativescript to appropriate plugin.

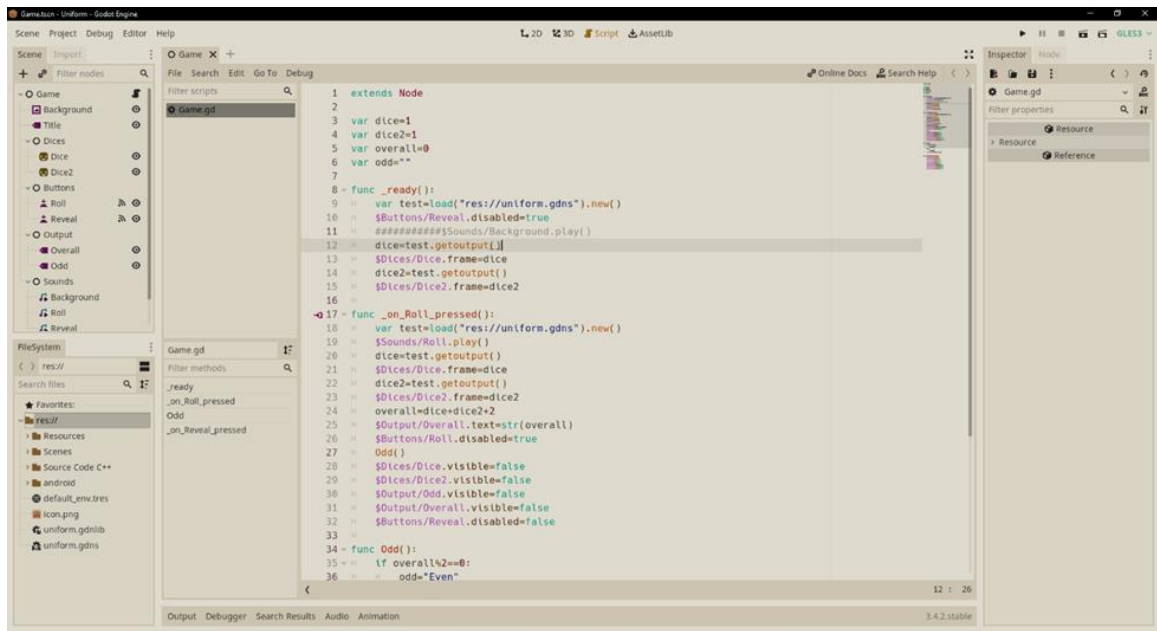
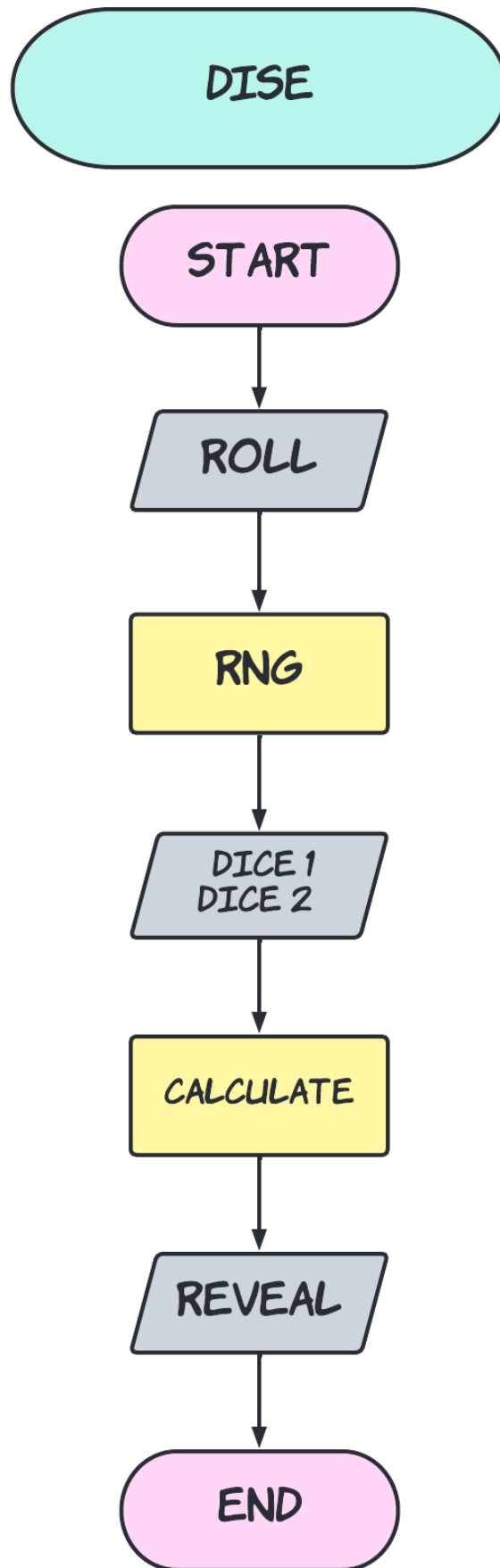


Figure 4.8 *Uniform Game Code*

As of v1.4 of the game **Untitled**; Pseudo and Uniform were added to the User Interface.

Appendix E

Game Flowchart: Dise



Appendix F

Game System Compatibility

Untitled System Requirement

<i>OS</i>	Windows 7/8/10/11
<i>Processor</i>	at least 32-bit
<i>Memory</i>	250 MB usage
<i>Graphics</i>	OpenGL 2.0 minimum with 200 MB usage
<i>Display</i>	1280x720p minimum, 1920x1080p maximum
<i>Storage</i>	420 MB available space
<i>32-bit Support</i>	Yes, separated
<i>Android Support</i>	No

Appendix G

Compiler Source

Source	modulus m	multiplier a	increment c	output bits of seed in <i>rand()</i> or <i>Random(L)</i>
ZX81	$2^{16} + 1$	75	74	
<i>Numerical Recipes</i>	2^{32}	1664525	1013904223	
Borland C/C++	2^{32}	22695477	1	bits 30..16 in <i>rand()</i> , 30..0 in <i>lrand()</i>
glibc (used by GCC)	2^{31}	1103515245	12345	bits 30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++ C90, C99, C11: Suggestion in the ISO/IEC 9899, C17	2^{31}	1103515245	12345	bits 30..16
Borland Delphi, Virtual Pascal	2^{32}	134775813	1	bits 63..32 of $(seed \times L)$
Turbo Pascal	2^{32}	134775813 (8088405 ₁₆)	1	
Microsoft Visual/Quick C/C++	2^{32}	214013 (343FD ₁₆)	2531011 (269EC3 ₁₆)	bits 30..16
Microsoft Visual Basic (6 and earlier)	2^{24}	1140671485 (43FD43FD ₁₆)	12820163 (C39EC3 ₁₆)	
RtlUniform from Native API	$2^{31} - 1$	2147483629 (7FFFFFFD ₁₆)	2147483587 (7FFFFFFC3 ₁₆)	
Apple CarbonLib, C++11's <i>minstd_rand0</i>	$2^{31} - 1$	16807	0	
C++11's <i>minstd_rand</i>	$2^{31} - 1$	48271	0	

MMIX by Donald Knuth	2^{64}	6364136223846 793005	1442695040888 963407	
Newlib, Musl	2^{64}	6364136223846 793005	1	bits 63..32
VMS's MTH\$RANDOM , old versions of glibc	2^{32}	69069 (10DCD ₁₆)	1	
Java's java.util.Random, POSIX [ln]rand48, glibc [ln]rand48[_r]	2^{48}	25214903917 (5DEECE66D ₁₆)	11	bits 47..16
random0	13445 6 = $2^3 7^5$	8121	28411	
POSIX [jm]rand48, glibc [mj]rand48[_r]	2^{48}	25214903917 (5DEECE66D ₁₆)	11	bits 47..15
POSIX [de]rand48, glibc [de]rand48[_r]	2^{48}	25214903917 (5DEECE66D ₁₆)	11	bits 47..0
cc65	2^{23}	65793 (10101 ₁₆)	4282663 (415927 ₁₆)	bits 22..8
cc65	2^{32}	16843009 (1010101 ₁₆)	826366247 (31415927 ₁₆)	bits 31..16
<i>Formerly common:</i> RANDU	2^{31}	65539	0	

Appendix H

Random Walk Code


```

#include<SDL2/SDL.h>
#include<random>
#include<chrono>
#include<iostream>
using namespace std;

int main(int argv, char** args)
{
    SDL_Window* pseudowindow=NULL;
    SDL_Window* uniformwindow=NULL;
    SDL_Renderer* pseudorenderer=NULL;
    SDL_Renderer* uniformrenderer=NULL;
    int ux=115, px=115;
    int uy=100, py=100;
    int loop;

    int seed = chrono::system_clock::now().time_since_epoch().count();
    default_random_engine gen(seed);
    uniform_int_distribution<int>distribution(1, 4);
    srand(seed);

    cout<<"How many times should the loop be?"<<endl;
    cin>>loop;
    pseudowindow=SDL_CreateWindow("Pseudo",100,100,700,700,0);
    uniformwindow=SDL_CreateWindow("Uniform",1920-900,100,700,700,0);
    pseudorenderer=SDL_CreateRenderer(pseudowindow,0,0);
    uniformrenderer=SDL_CreateRenderer(uniformwindow,0,0);
    SDL_RenderSetScale(pseudorenderer, 3, 3);
    SDL_RenderSetScale(uniformrenderer, 3, 3);
    for(int u=0; u<loop;u++)

```

```

{
    int dir=rand()%4+1;
    switch(dir)
    {
        case 1:
            px+=1;
            py+=1;
            break;
        case 2:
            px-=1;
            py+=1;
            break;
        case 3:
            px-=1;
            py-=1;
            break;
        case 4:
            px+=1;
            py-=1;
            break;
    }
    SDL_SetRenderDrawColor(pseudorenderer, 255,215,0,255);
    SDL_RenderDrawPoint(pseudorenderer, px,py);
    SDL_RenderPresent(pseudorenderer);
    SDL_Delay(1);

    int dir2=distribution(gen);
    switch(dir2)
    {
        case 1:

```

```

        ux+=1;
        uy+=1;
        break;
    case 2:
        ux-=1;
        uy+=1;
        break;
    case 3:
        ux-=1;
        uy-=1;
        break;
    case 4:
        ux+=1;
        uy-=1;
        break;
}
SDL_SetRenderDrawColor(uniformrenderer, 255,0,255,255);
SDL_RenderDrawPoint(uniformrenderer, ux,uy);
SDL_RenderPresent(uniformrenderer);
SDL_Delay(1);
if (px>230||px<0||py>200||py<-100) px=115, py=100;
if (ux>230||ux<0||uy>200||uy<-100) ux=115, uy=100;
}
system("Pause");
}

```

Appendix I

Pseudo-Random Dise Testing Player Mechanics

Table 3.3 *Pseudo-Random Dice Testing Player Mechanics*

Pseudo Random Number Generator	R1	R2	R3	R4	R5	R6	Appearance					
							1	2	3	4	5	6
Ryan Ramos	5	4	5	3	2	3	0	1	2	1	2	0
	5	3	2	2	2	3	0	3	2	0	0	0
	EVEN	ODD	ODD	ODD	EVEN	EVEN						
John Luis Cabauatan	5	4	2	3	2	3	0	2	2	1	1	0
	4	2	4	4	4	2	0	2	0	4	0	0
	ODD	EVEN	EVEN	ODD	EVEN	ODD						
Paul Padilla	2	4	1	2	5	6	1	3	0	1	1	1
	6	4	4	4	3	5	0	0	1	2	2	1
	EVEN	EVEN	ODD	EVEN	EVEN	ODD						
Kevin Raña	2	6	6	5	2	6	0	2	0	0	1	3
	2	2	6	4	5	6	0	2	0	1	1	2
	EVEN	EVEN	EVEN	ODD	ODD	EVEN						
Julian Serquiña	5	2	4	3	5	6	0	1	1	1	2	1
	2	5	5	4	6	6	0	1	0	1	2	2
	ODD	ODD	ODD	ODD	ODD	EVEN						
Winford Cabuena	3	6	3	5	2	3	0	1	2	0	1	1

	4	2	5	2	3	3	0	2	2	1	1	0
	ODD	EVEN	EVEN	ODD	ODD	EVEN						
Jester Mauricio	3	6	6	3	5	2	0	1	2	0	1	2
	5	6	6	3	4	2	0	1	1	1	1	2
	EVEN	EVEN	EVEN	EVEN	ODD	EVEN						
Latrell Gelacio	3	2	2	5	2	3	0	3	2	0	1	0
	4	3	5	5	6	2	0	1	1	1	2	1
	ODD	ODD	ODD	EVEN	EVEN	ODD						
Franz Chua	2	5	3	5	2	3	0	2	2	0	2	0
	6	3	2	5	6	5	0	1	1	0	2	2
	EVEN	EVEN	ODD	EVEN	EVEN	EVEN						
Reo Lance Villanueva	6	2	4	3	4	4	0	1	1	3	0	1
	5	2	6	3	4	6	0	1	1	1	1	2
	ODD	EVEN	EVEN	EVEN	EVEN	EVEN						

Appendix J

Uniform

Distribution Dise

Testing Player

Mechanics

Table 3.4 *Uniform Distribution Dice Testing Player Mechanics*

Uniform Distribution Random Number Generator	R1	R2	R3	R4	R5	R6	Appearance					
							1	2	3	4	5	6
Ryan Ramos	1	3	5	4	5	4	1	0	1	2	2	0
	5	5	6	5	2	4	0	1	0	1	3	1
	ODD	EVEN	ODD	ODD	ODD	EVEN						
John Luis Cabauatan	3	5	1	6	5	2	1	1	1	0	2	1
	1	4	2	1	4	3	2	1	1	2	0	0
	EVEN	ODD	ODD	ODD	ODD	ODD						
Paul Padilla	3	5	1	2	4	5	1	1	1	1	2	0
	2	4	2	3	4	3	0	2	2	2	0	0
	ODD	ODD	ODD	ODD	EVEN	EVEN						
Kevin Raña	2	2	4	2	4	3	0	3	1	2	0	0
	6	3	6	4	6	1	1	0	1	1	0	3
	EVEN	ODD	EVEN	EVEN	EVEN	ODD						
Julian Serquiña	1	5	5	2	4	5	1	1	0	1	3	0
	4	1	2	5	2	4	1	2	0	2	1	0
	ODD	EVEN	ODD	ODD	EVEN	ODD						
Winford Cabuena	5	6	3	4	5	5	0	0	1	1	3	1

	6	6	2	5	6	2	0	2	0	0	1	3
	ODD	EVEN	ODD	ODD	ODD	ODD						
Jester Mauricio	6	6	1	6	4	4	1	0	0	2	0	3
	2	3	3	5	4	1	1	1	2	1	1	0
	EVEN	ODD	EVEN	ODD	EVEN	ODD						
Latrell Gelacio	2	2	1	4	6	5	1	2	0	1	1	1
	2	1	4	2	3	5	1	2	1	1	1	0
	EVEN	ODD	ODD	EVEN	ODD	EVEN						
Franz Chua	5	4	2	1	2	3	1	2	1	1	1	0
	1	2	2	5	1	2	1	3	0	0	1	0
	EVEN	EVEN	EVEN	EVEN	ODD	ODD						
Reo Lance Villanueva	4	5	5	4	6	5	0	0	0	2	3	1
	6	5	6	4	1	2	1	1	0	1	1	2
	EVEN	EVEN	ODD	EVEN	ODD	ODD						

CURRICULUM VITAE

Curriculum Vitae



Granel Jaylord A. Cabuena

#04 Saquing St. San Nicolas, Bayombong, Nueva Vizcaya 3700
cabuenagranel1@gmail.com

PERSONAL INFORMATION

Birthdate : July 2, 2000
Place of Birth : Bayombong, Nueva Vizcaya
Civil Status : Single
Religion : Roman Catholic

EDUCATIONAL ATTAINMENT

Baccalaureate :
Secondary : Saint Mary's University HS & SHS
Elementary : Bayombong West Elementary School

Membership to Organizations

Name of Organization

Date

Seminars Attended

Drive Online Sales with Google Ads	August 12, 2021
Introduction to Artificial Intelligence	August 14, 2021
Create an Eye Catching Website	August 18, 2021
AI (Artificial Intelligence) and ML (Machine	

Learning) as Software Test Automation Tool	August 19, 2021
Blockchain and the Cloud	August 26, 2021
Digital Technology Maximization	August 28, 2021
Introduction to DevOps Tools & Stages	September 11, 2021
Cyber Security: The Rise of Phishing Attacks	September 11, 2021

Curriculum Vitae



Chino Joshua G. Abbagu

#9A Saquing St. San Nicolas, Bayombong, Nueva Vizcaya 3700
cabbagu@gmail.com

PERSONAL INFORMATION

Birthdate : June 20, 2000
Place of Birth : Bayombong, Nueva Vizcaya
Civil Status : Single
Religion : Roman Catholic

EDUCATIONAL ATTAINMENT

Baccalaureate :
Secondary : Saint Mary's University HS & SHS
Elementary : Bayombong West Elementary School

Membership to Organizations

Name of Organization

Date

Seminars Attended

Drive Online Sales with Google Ads	August 12, 2021
Cyber Safety	August 14, 2021
Introduction to Artificial Intelligence	August 14, 2021
AI (Artificial Intelligence) and ML (Machine Learning) as Software Test Automation Tool	August 19, 2021
Digital Technology Maximization	August 28, 2021
Introduction to C Programming Language	September 6, 2021

Artificial Neural Network

September 9, 2021

Future of Cyber Security

September 11, 2021