The Differences of Pseudorandom and Truly Random Numbers

To what extent can random numbers be used in place of pseudorandom numbers?

Computer Science

3844 words

Contents

Introduction:

Random numbers are essential to modern life and have a wide range of applications. There are two categories of randomness: truly random and pseudorandom. A truly random number (TRN) is a random number created from random events. A pseudorandom number (PRN) is a random number created algorithmically. Each type of random number is more suited for some applications of random numbers than others, some of which are essential to modern life.

Since PRNs are created using algorithms, they are very easy for computers to create. Computers create PRNs using algorithms known as PRN generators. The numbers produced by these algorithms are not truly random since TRN cannot be predicted and algorithms are perfectly predictable. Despite this, PRN generators attempt to create numbers that appear to be truly random.

TRN, on the other hand, require observing quantum events or chaotic systems. Quantum events are events between quantum particles and are probabilistic. Chaotic systems are systems that are very sensitive to changes and initial conditions. Under the current understanding of physics, quantum events and chaotic systems cannot be predicted and can, therefore, be used to create TRNs.

This investigation will explore what randomness is and what it means for numbers to be random. It will also briefly explain why random numbers can be extracted from cyanotic systems and radioactive decay. The limitations of PRNs will be shown using the Middle Square Method. A more suitable PRN generator, the Messene Twister, will also be analyzed to highlight the efficiency and practicality of PRN generators. Three sources of TRNs will also be considered, two from radioactive decay and one from a chaotic system. One of the

TRN generators that use radioactive decay was experimentally created to show how TRNs can be generated in practice. The several random number sources will also be analyzed based on the several characteristics of random numbers. PRN and TRN generators will be compared to highlight each's place in modern society, with reference to three applications: encryption, simulations and video games.

Randomness:

Random numbers have a few essential characteristics. Firstly, they are independent of each other, rolling a six on a fair die will not affect the next roll. Secondly, random numbers in this context should be equally probable just like when you flip a fair coin. To be truly random, they should also not be predictable. PRNs cannot satisfy this expectation of random numbers but should satisfy the other two conditions.

Before random number generators can be properly analyzed, a way to decide whether a set of numbers meets the requirements of independence and equal probability. Furthermore, a way to test the practicality of a set of random numbers is also necessary. Randomness, unfortunately, cannot ever truly be tested because "It is impossible to prove that a given sequence is random" (Dewdney 53). However, the ability to prove a series of numbers is random is not necessary, compelling evidence can still be acquired in favor of a series of numbers being random.

To test if each possible number is occurring at the correct rate the average and Shannon's entropy (entropy) can be used. The average of a series of random numbers is expected to be about equal to the average of all the possible numbers. Entropy is the measure of how much information each number in a series of numbers contains and is given by:

$$H = -\sum_{i=0}^{m} p_i log_2(p_i)$$

Equation 1 (Borda 11)

Where m is how many different numbers are possible, p_i is how often the i^{th} possible number occurs, and H is the number of bits of information each number has, or the entropy of each number. "Entropy is maximized when the [numbers] have equal probabilities" (Borda 12).

Since this is expected of random number generators, H should be about the number of bits in each random number. For example, imagine two different random number generators. Both produce 2-bit random numbers between zero and three. Generator 1 produces good random numbers and Generator 2 produces poor random numbers. A set of sixteen random numbers is produced by each. Set 1 and Set 2 are generated by Generator 1 and Generator 2 respectively. Table 1 shows how often each number occurs in each set.

1	3	2	0	0	0	2	3	0	1	3	2	1	1	2	3

Set 1

0	3	3	3	0	0	0	2	3	1	1	0	1	2	3	3

Set 2

Number	Set 1 Frequency	Set 2 Frequency
0	25%	31.25%
1	25%	18.75%
2	25%	12.5%
3	25%	37.5%

Table 1

Using the information from Table 1, the entropy of Set 1 and Set 2 can be calculated. The entropy of Set 1 is 2.00 bits while and the entropy for Set 2 is only about 1.89 bits. Set 1 appears to be random because its entropy is equal to the number of bits in each random number. Set 2 does not appear to be random because its entropy is less than the number of bits in each random number.

To test whether random numbers in a series are independent of each other the serial correlation of the series can be analyzed. Serial correlation "measures the extent to which each byte in the file depends upon the previous byte" (Walker, "A Pseudorandom Number Sequence

Test Program."). The serial correlation of a set of random numbers is expected to be about zero since random numbers should have no influence over each other.

Lastly, to test the practicality of a series of random numbers a Monte-Carlo simulation can be run. Monte-Carlo simulations are "numerical experimentation technique[s] to obtain the statistics of the output variables of a system computational model" (Cruse 123). These types of simulations should produce accurate results if it is run using random numbers. Therefore, accurate results from Monte Carlo simulations suggest randomness.

To collect these statistics and run a Monte Carlo simulation on the random number samples produced by each source a utility named Ent (Walker, *Ent*) was used. Ent calculates the average, entropy, and serial correlation of a series of random numbers and runs a Monte-Carlo simulation to approximate Pi. Ent treats each sample of random numbers as a source of eight-bit unsigned random numbers ranging from zero to 255. Each metric and its expected value are listed in Table 2.

Statistic	Expected Value
Shannon's Entropy	8.0
Serial Correlation	0.0
Average	127.5
Monte Carlo Pi	3.1416

Table 2

These tests are not proof but evidence and can produce false positives and negatives. However, the probability that a set of random numbers would not satisfy these conditions is so small that for pragmatic purposes, the possibility of these tests being incorrect can be safely ignored, and for the remainder of this investigation will be. For all but the experimentally produced random numbers, the analysis was conducted with four KiB of random numbers.

True Randomness:

TRNs must be extracted from random events, for example flipping a coin. However, flipping a bunch of coins every time you need a random number is far too tedious and takes far too long to be practical. For this reason, physical processes are exploited. One physical process that can be used is radioactive decay. Radioactive decay in a random physical event that occurs in unstable atomic nuclei and is determined by quantum mechanics ("Radioactivity"). Radioactive substances tend to obey a half-life meaning that half of the remaining radioactive atoms usually decay every fixed length of time (Wolke). Though we can calculate how long it should take before each particle is emitted from a radioactive substance using its half-life, each atomic nucleus is independent of the others and has a certain probability of decaying. Because of this, the emission of radioactive particles cannot be predicted. The randomness of when radiation partials will be emitted from radioactive substances can be used to create TRNs. TRNs can also be generated by measuring changes in a chaotic system such as atmospheric noise. Both radioactive decay and atmospheric noise as sources of random numbers will be explored in this investigation. Websites that provide TRN samples used in this investigation will be assumed not to have tampered with the random number to ensure good statistical analysis.

Pseudo-Randomness:

PRN generators attempt to mimic truly random sources algorithmically. PRN generators use states to create random numbers. States are sets of values for all the variables in a PRN generator (PRNG). Each state can produce one or many random numbers depending on which algorithm is being used. If more PRNs are needed and the state has produced all its PRNs, a function is applied on the current state to generate the next. PRNGs need a seed to start so that they can create an initial state an being producing random numbers. The same seed and algorithm will always produce the same set of random numbers because computers are deterministic. The seed of a PRNG is usually chosen based on the time but can determine in other ways. However, PRNs have one major flaw: they always enter a loop. Looping occurs because the current state determines the next state. Since there are a finite number of possible states for a PRNG to be in, it will eventually return to a previous state. This results in indefinite looping and can be visualized in Figure 1.

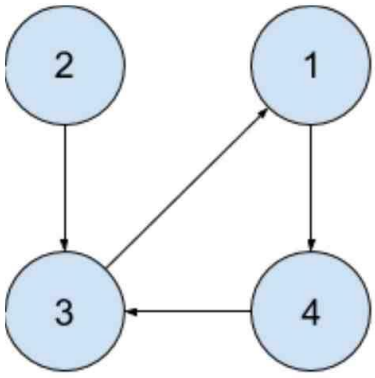

Figure 1

Starting at any number and following the arrows will result in looping between one, four, and three. With finitely many possible states and drawing only one arrow from each state, there is no way to avoid looping. The quantity of random numbers produced by a PRNG before it loops

is called the period of that PRNG. The longer the period is the more random numbers it can generate before looping. After a PRNG beings to loop, it should no longer be used. However pseudorandom number generators are much faster than TRN generators because computers are fast at performing calculations.

The Middle Square Method:

The Middle Square Method is a very simple PRNG that highlights the looping problem in PRNGs. In the Middle Square Method, each state is also a random number. The next state and random number is calculated by squaring the current state, removing extra digits or adding zeros on the left until it is uniform length, and then extracting the middle digits. In this algorithm zero is followed by itself, meaning that if the algorithm produces a zero, it will only produce zeros thereafter. This problem is worsened by the fact that zeros are a frequent output of this PRNG. When a current state is a small number or power of ten, the next state is usually a zero. As a result, this algorithm often enters a loop of zeros. Because of this tendency, collecting four KiB of random numbers without collecting almost exclusively zeros is not possible without modifying the algorithm. In order to analyze the Middle Square Method best, whenever it entered a loop it was reset and given a new seed. The statistics of a four KiB sample produced using the modified Middle Square Method can be seen in Table 3.

Statistic	Value	Expected Value	Error
Bytes	16384	n/a	n/a
Shannon's Entropy	7.4045	8.0000	7.4437%
Serial Correlation	0.0259	0.0000	n/a
Average	125.1362	127.5000	1.8539%
Monte Carlo Pi	3.0139	3.1416	4.0640%

Table 3

There is not strong evidence of randomness for this algorithm. Due to the large sample size, smaller errors would be expected of random numbers. The entropy of the random numbers produced by this algorithm is particularly poor suggesting some numbers occurred too often and some numbers did not occur often enough. This problem can also be observed in Figure 2.

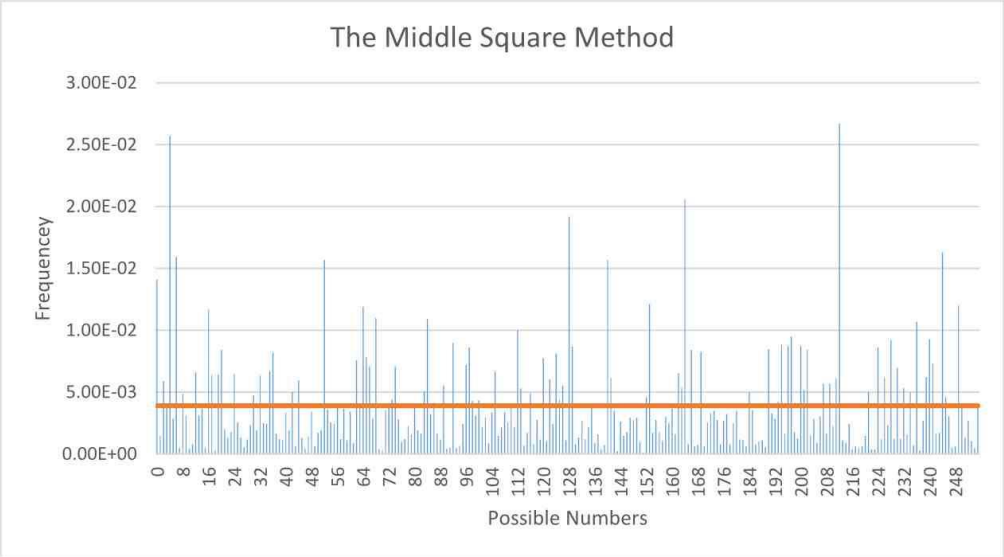

Figure 2

Certain numbers occurred at much higher frequencies than expected while others occurred at far lower frequencies than expected. Because of this, there is no evidence that this sample of numbers is random. The Middle Square Method is an excellent example of how not to generate a random number and clearly outlines the key problems with PRN. While other PRNGs can correct the statistical problems, no pseudorandom number can avoid looping.

Random.org:

Random.org is a service that measures changes in the amplitude of atmospheric noise to produce random bits. The random numbers produced by random.org are based on chaos, not quantum mechanics. Despite this, it is still a source of TRNs since chaotic systems are unpredictable without perfect knowledge, which is impossible in this case. Perfect information for atmospheric noise includes the location and speed of every air particle (Haahr). The Heisenberg Uncertainty Principle states that this is impossible to know both the speed and location of a particle, making atmospheric noise unpredictable. However, unpredictability alone is not enough, each bit should have an equal chance of being a one or zero. To test this, the relevant metrics of a four KiB sample of random numbers from Random.org can be seen in Table 4.

Statistic	Value	Expected Value	Error
Bytes	16384	n/a	n/a
Shannon's Entropy	7.9899	8.0000	0.1258%
Serial Correlation	0.0032	0.0000	n/a
Average	127.6027	127.5000	0.0805%
Monte Carlo Pi	3.1121	3.1416	0.9392%

Table 4

There is clear evidence of randomness in the numbers produced by Random.org. The errors are very low as expected due to the large sample size. The serial correlation is also very low implying that the random numbers do not affect each other. The high entropy of the sample implies that each number occurred at about the same rate. This is further supported by Figure 3.

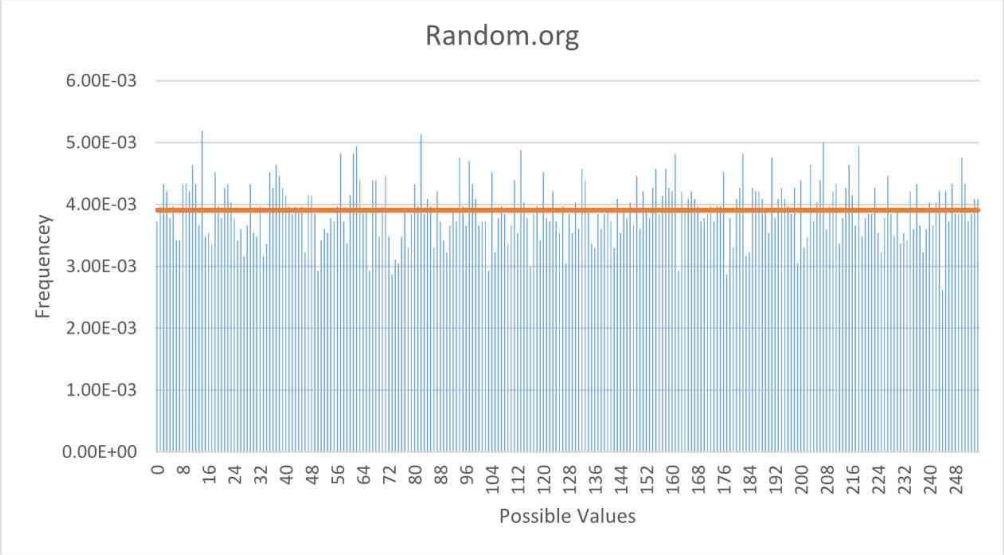

Figure 3

All possible numbers occur about as often as each other. There are no excessively high or low frequencies as suggested by the low value of entropy. There also does not appear to be any patterns on the frequency chart. Therefore, there is evidence that the sample is truly random. However, this truly random number generator (TRNG) cannot produce random numbers as quickly as PRNGs because physical measurements must be made to produce each bit.

Mersenne Twister:

The Mersenne Twister is a PRNG that is more complex, and resource intensive than the Middle Square Method. It is currently the default source of random numbers in many programming languages, for example, Ruby (Britt). In the Mersenne Twister, a signal state produces many random numbers. Like the Middle Square Method, if the Mersenne Twister ever returns to a previous state then it enters a loop. However, it takes much longer for the Mersenne Twister to enter a loop. It can also safely provide zeros as random numbers. The statistics of a four KiB sample produced using the Mersenne Twister can be seen in Table 5.

Statistic	Value	Expected Value	Error
Bytes	16384	n/a	n/a
Shannon's Entropy	7.9897	8.0000	0.1284%
Serial Correlation	-0.0017	0.0000	n/a
Average	128.2832	127.5000	0.6143%
Monte Carlo Pi	3.0711	3.1416	2.2351%

Table 5

There is some evidence of randomness for the numbers produced by the Messene Twister. They are high in entropy and has a low serial correlation. However, their average differs more from the expected value than the numbers produced by Random.org. Furthermore, the results of the Monte-Carlo simulation show that these numbers are not as suitable for simulations as the numbers produced by Random.org.

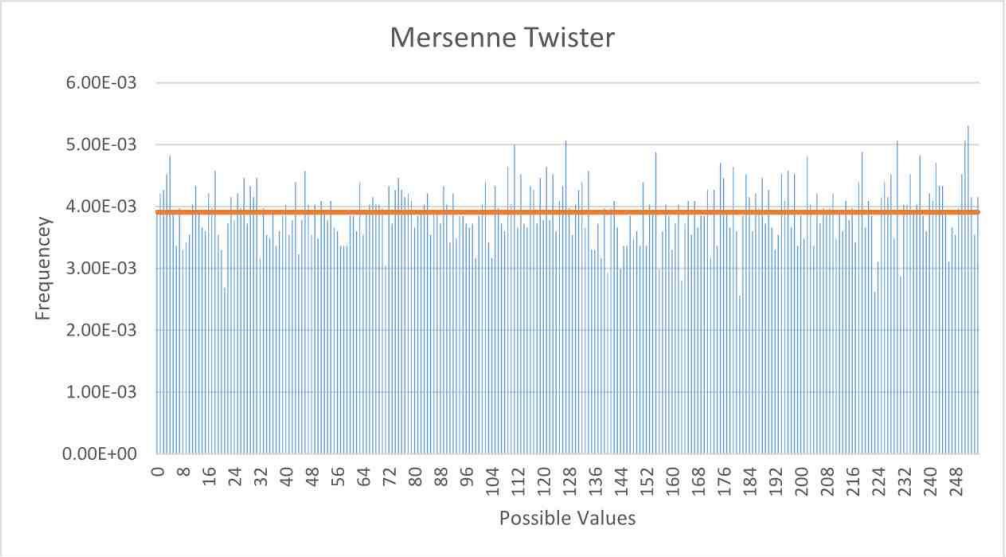

Figure 4

Figure 4 shows that there are no large spikes and that there is appropriate variation in frequency similar to that of Random.org. The Messene Twister appears provides good quality random numbers more efficient than TRNG, though it does not produce accurate simulations.

HotBits:

HotBits like Random.org is a service that provides random numbers over the internet. HotBits however, uses radioactive decay to generate random numbers. They use Cæsium-137 as a radioactive source (Walker, "Tunnelling to Freedom"). HotBits uses four consecutive emissions to create one random bit. The time between the first two emissions and the second two emissions are compared to each other. Depending on which time span is longer, a one or a zero is recorded. HotBits alternate between which time span needs to be larger to prevent a bias in their random numbers since the second-time span is expected to take slightly longer because there are two fewer atoms capable of decaying. The statistics of a four KiB sample from HotBits can be seen in Table 6.

Statistic	Value	Expected Value	Error
Bytes	16384	n/a	n/a
Shannon's Entropy	7.9896	8.0000	0.1305%
Serial Correlation	-0.0008	0.0000	n/a
Average	127.6028	127.5000	0.0806%
Monte Carlo Pi	3.1399	3.1416	0.0530%

Table 6

There is strong evidence of randomness from this source. The entropy is very high, the serial correlation is very low, the average is close to the expected value and the approximation of Pi is very accurate. The statistics calculated for this sample are very similar to random.org. The low entropy is reflected in Figure 5.

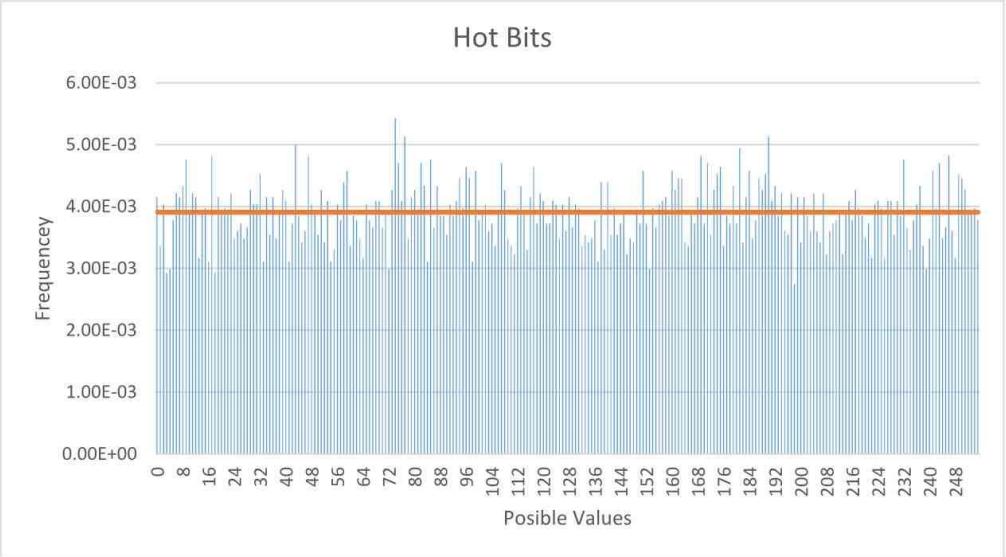

Figure 5

There are no frequencies that are significantly higher or lower than the expected frequency. There is clear evidence of randomness given the statistics and frequency graph. The similarities between the statistics and graphs of the samples produced by HotBits and Random.org suggest that they both produced the same type of numbers: TRNs.

The Experiment:

To test random numbers generation in practice, radioactive decay was used to produce random bits. The method used was similar to that of HotBits with a few small changes. Firstly, instead of using four emissions to produce a random bit, three were used for the first bit and two for all successive bits. This efficiency boost is possible since HotBits does not use the time span between the second and third emissions or the time spans between each set of four emissions. Secondly, a Geiger counter was used along with an Arduino to collect the time stamps for later processing instead of being processed as the emissions were detected. Lastly, due to a limitation in the Geiger counter, only 9999 emissions could be recorded without human intervention. Figure 6 shows the arrangement of the different devices in the recording apparatus.

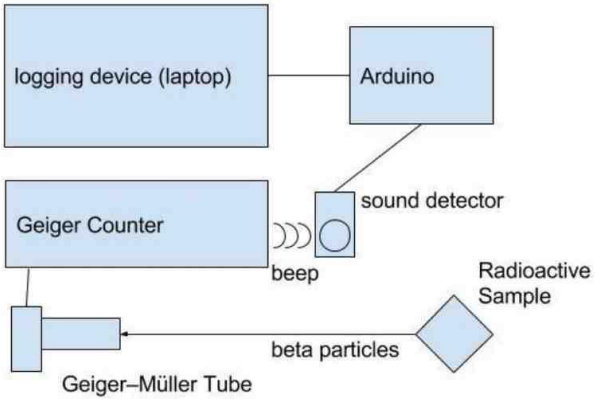

Figure 6

For safety reasons, this experiment could not be conducted during the day when the classroom was in use. These limitations meant that 4 KiB of random numbers could not be produced, only 0.63 KiB. This has serious consequences on the statistics shown in Table 7. In order to determine if these values are acceptable, the Ent was recursively run on the sample from

HotBits to determine the value of each metric at all possible sample size. Figures 8 through 11 show the effect of sample size on the errors in each metric using the HotBits sample. In Figures 8 through 11 the measurements made on the experimentally generated sample are also plotted. Figure 7 shows a graph of the frequencies of each number in the experimentally generated sample.

Statistic	Value	Expected Value	Error
Bytes	648	n/a	n/a
Shannon's Entropy	7.6993	8.0	3.7581%
Serial Correlation	-0.0173	0.0	n/a
Average	132.3534	127.5	3.8066%
Monte Carlo Pi	3.1852	3.1416	1.3876%

Table 7

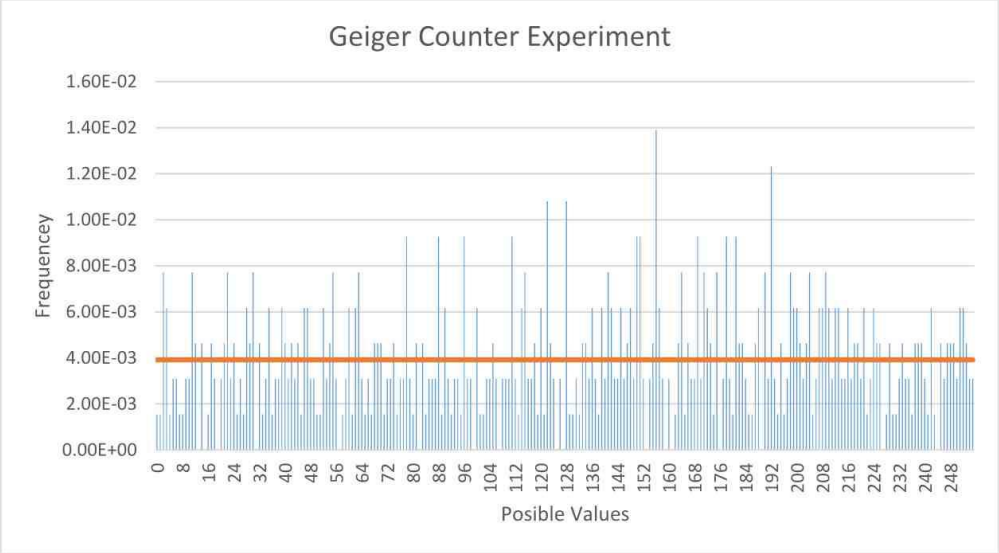

Figure 7

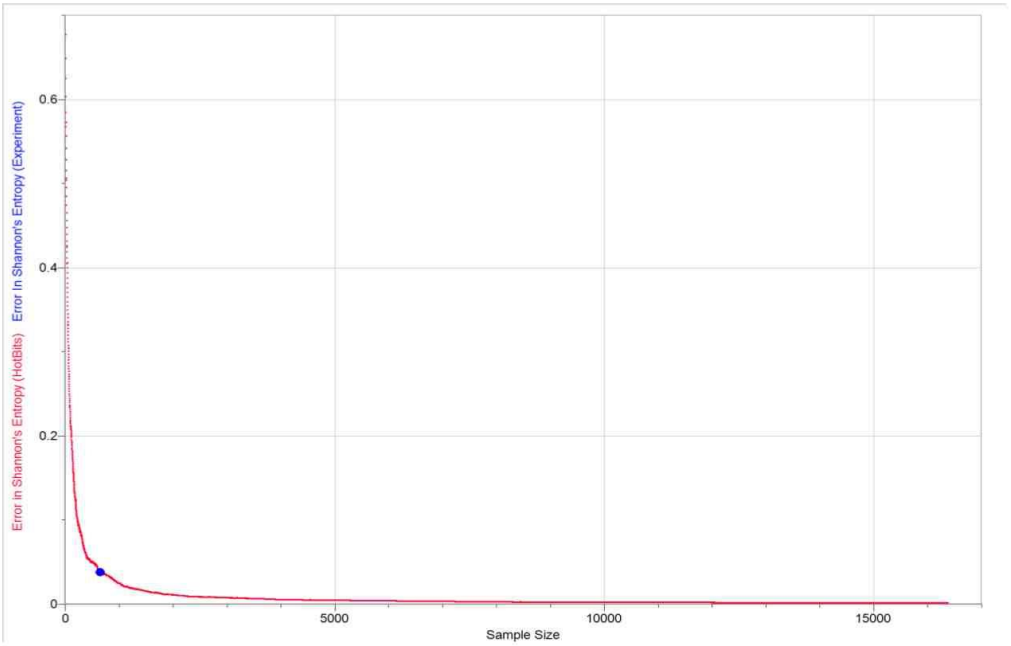

Figure 8

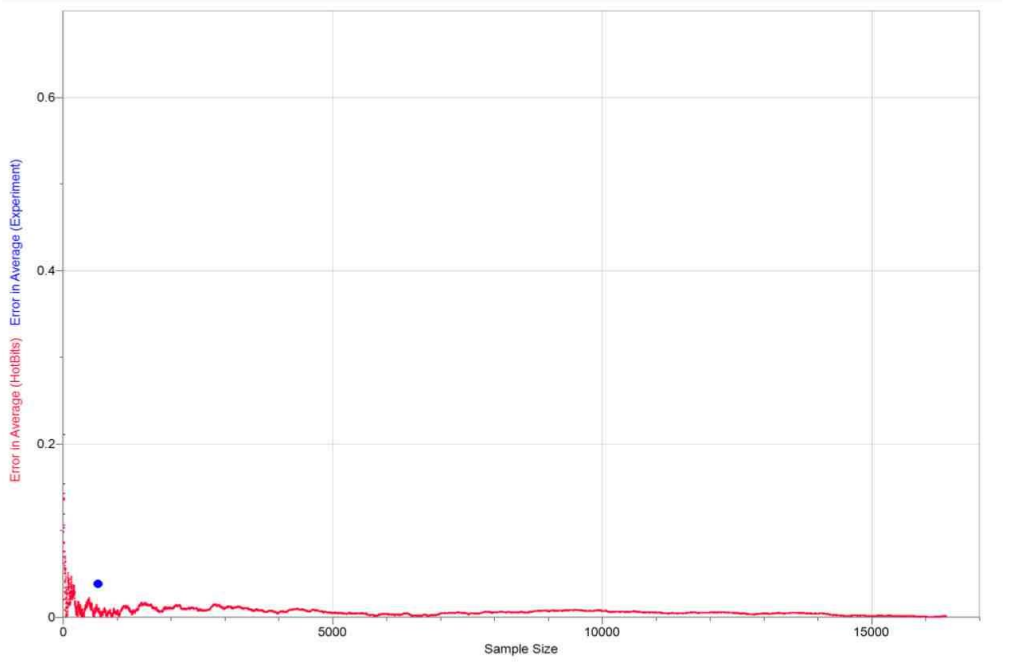

Figure 9

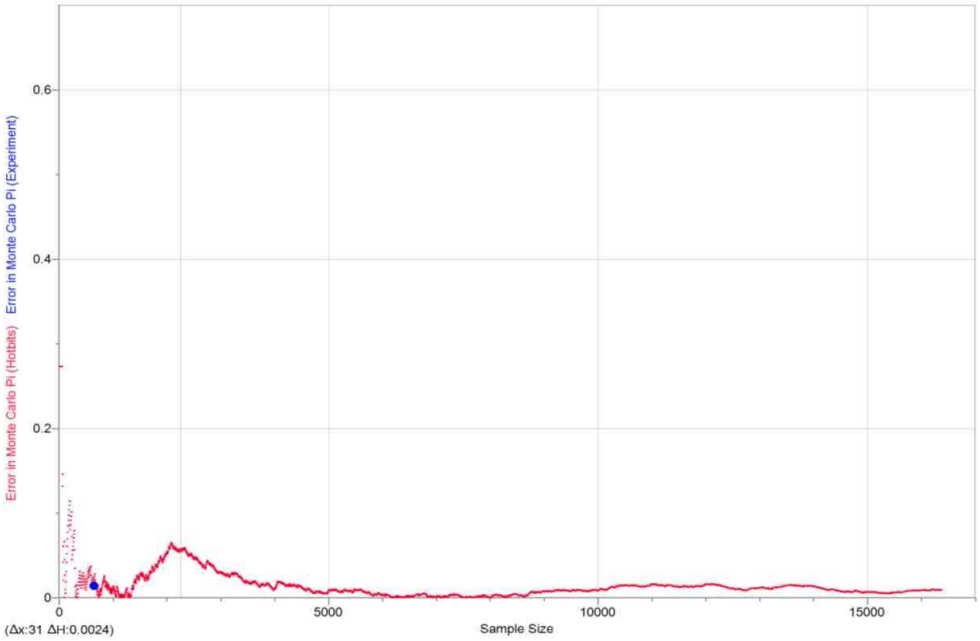

Figure 10

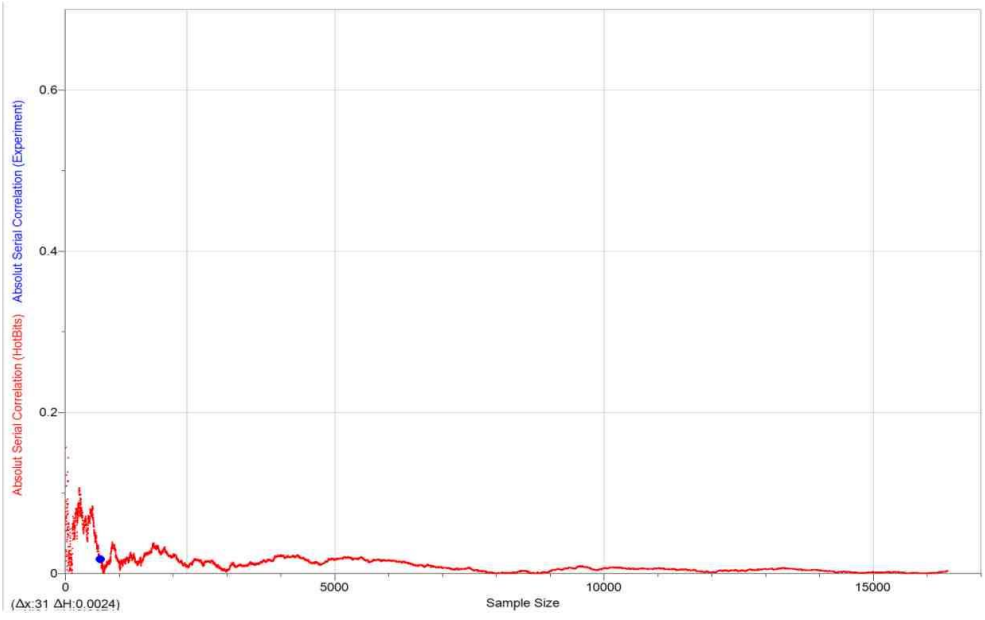

Figure 11

There is some evidence for the sample produced experimentally being random. Every metric except the average is acceptable when compared to HotBits with the same sample size. The average is unusually high and can be visualized in Figure 9. However, there are not enough numbers to determine if there is serious bias or if it just occurred by chance. Due to the nature of random numbers, it is impossible to know. However, there is some evidence in favor of randomness that should not be ignored. A larger sample size would be needed to solidify a compelling case for or against randomness.

Analysis:

There is evidence that TRNGs are statistically superior to PRNGs. However, PRNs can be generated much more efficiently than TRNs. Also, PRNs, unlike TRNs, can be re-generated using the same seed and algorithm. Depending on the application this can be a feature or a major flaw. Deciding when to use which type of random numbers to use in applications across the world is very important to our daily lives.

Encryption plays a huge role all over the world, without it people could not authenticate themselves, protect important information, or issue and validate digital signatures. Encryption, however, only works when each set of keys is unique to those using it. To ensure someone else does not guess a key, random numbers are essential. PRNs introduce weaknesses into encryption since their behavior is easy to predict. For this reason, cryptographically secure PRNG is used for generating encryption keys. However, this still leaves one weakness, seeds are usually just sixty-four bits and encryption keys are usually much larger. It is therefore usually easier to guess the seed of a cryptographically secure random number generator than it is to guess the key itself. Since the strength of encryption lies in the difficulty of guessing the key, PRNGs weaken encryption. For most applications, this weakness is acceptable since large-scale attacks are not expected, and the seed is still difficult to guess. However, for critical applications or applications expected to be heavily attacked, truly random numbers should be used to maximize security.

Simulations are another application of random numbers. Atmospheric simulations, for example, require vast quantities of random numbers. PRNs are a seemingly better choice given their efficiency, and the large number needed. However, the looping problem with PRNGs can be problematic. Furthermore, the PRNGs produced significantly worse results from the Monte Carlo simulation. TRNs appear to produce more accurate simulations. On the other hand, using

PRNs allows simulations to be repeated easily by recording just the seed instead of all of the random numbers outputted by a TRNG. A PRNG with a sufficiently long period may be more favorable depending on whether repeating the simulation will be necessary, however, it is important to ensure that the PRNG has a long period. If a PRNG is used for simulations and the period is too short, then the simulation will not provide good results. The use of TRNs, however, could make simulations more accurate. Therefore, deciding which type of random numbers to use in simulations should be based on whether speed or accuracy is more important.

Video games, however, require far fewer random numbers. Since TRNs require special hardware and take longer to generate, PRNs are cheaper and more efficient making them the best candidate for video games. Furthermore, since the seed can be used to reproduce the series of random numbers, unique situations and levels can all be replicated by just using the seed. This allows the authors of the game to easily reproduce errors. TRNs, on the other hand, would take longer and could not be easily shared. Furthermore, unlike with encryption secrecy is often not an important feature. Algorithms like the Mersenne Twister are the best choice for video games.

There were several flaws in this investigation. The most significate flaw was the inability to collect four KiB of random numbers from the experimental source. The comparison of the error in HotBits at each sample size revealed that the error in average was higher than expected casting some doubt on to the quality of the random number produced. Furthermore, the need to detect the beeping sound from the Geiger Counter introduced the potential for missed beeps and therefore a bias in the random numbers. Another key flaw is that either HotBits or Random.org or both could have altered their random numbers to ensure they produced good random numbers. This could be the reason that each of the truly random sources had performed well on each statistic instead of the randomness of the source.

Conclusion:

Random numbers are essential to all modern forms of encryption, can be used to simulate complex systems, and are used in many applications such as video games. The issue of using TRNs or PRNs is one of global importance. While sometimes it is a matter of efficiency, practicality, or precision, other times it can be the difference between a secure transaction and bankruptcy.

The statistical analysis of the random samples from pseudorandom sources and truly random sources provides evidence that seemingly random behavior can be algorithmically generated. An insight into how they are generated however reveals that they should not be used in some applications due to their predictability and periodic nature. PRNs, however, are universal and do not require special hardware.

The statistical analysis of TRNs is evidence that chaotic system and quantum events are both good sources of TRNs. The experimentally produced random numbers sample was not large enough to make a compelling case for randomness, however, it does warrant further investigation with a larger sample size. In addition, it shows that an alternative method could potentially double the quantity of random bits produced by HotBits.

TRNs are inefficient, non-deterministic, and do not loop. PRNs, on the other hand, are efficient, deterministic, and always loop. Which type of random numbers is best depends on which features are most important to an application. Using the correct types of random numbers can allow games include introduce unpredictable behavior, make simulations either more accurate or quicker, and can even make encryption safer.

Works Cited

Borda, Monica. *Fundamentals in Information Theory and Coding*. Berlin, Springer, 2011.

Britt, James. "Class: Random (Ruby 1.9.3)." *Ruby-Doc*, Neurogami, ruby-doc.org/core-1.9.3/Random.html. Accessed 7 Sept. 2017.

Cruse, Thomas A. *Reliability-based Mechanical Design*. New York, M. Dekker, 1997.

Dewdney, Alexander. "Random Numbers." *The New Turning Omnibus*, 1989, pp. 49-55.

Haahr, Mads. "Introduction to Randomness and Random Numbers." *random.org*, www.random.org/randomness/. Accessed 7 Sept. 2017.

"Radioactivity." *Institute of Physics*, www.iop.org/resources/topic/archive/radioactivity/index.html. Accessed 10 Sept. 2017.

Walker, John. *Ent. Pseudorandom Number Sequence Test Program*, John Walker, 28 Jan. 2008, www.fourmilab.ch/random/. Accessed 5 Sept. 2017.

---. "A Pseudorandom Number Sequence Test Program." *Pseudorandom Number Sequence Test Program*, 28 Jan. 2008, www.fourmilab.ch/random/. Accessed 5 Sept. 2017.

---. "Tunnelling to Freedom." *HotBits*, www.fourmilab.ch/hotbits/how3.html. Accessed 10 Sept. 2017.

Wolke, Robert L. "Half-life." *The Gale Encyclopedia of Science*, edited by K. Lee Lerner and Brenda Wilmoth Lerner, 5th ed., Gale, 2014. *Science in Context*, link.galegroup.com/apps/doc/CV2644031061/SCIC?u=fis&xid=a1a9cb19. Accessed 23 Mar. 2017.

Works Consulted

Knuth, Donald E. "Chapter 3: Random Numbers." *Art of Computer Programming*, vol. 2, 1968. 5 vols.

"Nuclear Detection Devices." *World of Forensic Science*, edited by Sara Constantakis, 2nd ed., vol. 2, Gale, 2016, pp. 490-491. *Science in Context*, link.galegroup.com/apps/doc/CX3630600375/SCIC?u=fis&xid=939589bd. Accessed 23 Mar. 2017.

Osaci, Mihaela. "STUDIES ABOUT INFLUENCE OF THE PSEUDO-RANDOM NUMBER GENERATORS IN SIMULATION MODELS FOR MAGNETODIELECTRIC NANOCOMPOSITES." *Acta Technica Corviniensis - Bulletin of Engineering*, vol. 7, no. 3, 2014, pp. 103-107*, Research Library Prep*, https://search.proquest.com/docview/1547946933?accountid=44981.

Radioactive Decay. physics.bu.edu/py106/notes/RadioactiveDecay.html. Accessed 23 Mar. 2017.

"Radioactive Decay." *IB Physics Stuff*, ibphysicsstuff.wikidot.com/radioactive-decay. Accessed 15 Mar. 2017.

"Random Number Generation Basics." *PCG, a Better Random Number Generator*, 2015, www.pcg-random.org/rng-basics.html. Accessed 15 Mar. 2017.

Uner, Eric. "Generating Random Numbers." *Embedded.com*, 24 May 2004, www.embedded.com/design/configurable-systems/4024972/Generating-random-numbers. Accessed 15 Mar. 2017.

Walker John. "Genuine Random Numbers, Generated by Radioactive Decay." *HotBits*, Jan. 2017, www.fourmilab.ch/hotbits/. Accessed 15 Mar. 2017.

Ward, Steve. "Can a Computer Generate a Truly Random Number?" *MIT School of Engineering*, 1 Nov. 2011, engineering.mit.edu/ask/can-computer-generate-truly-random-number. Accessed 15 Mar. 2017.

Appendix

Appendix A - the first 256 bytes produced using the middle square method displayed in hexadecimal

```
44 49 7a 99 2c 37 ad a4 81 2a 94 04 f4 23 3c b1
02 66 e6 81 ea d1 e2 78 20 c5 cf ef 8c c4 80 c2
be 3e f9 c8 8c 54 04 a4 d4 34 04 77 2e 29 7b 54
83 80 b5 4e 3c 24 0c 01 00 00 b9 08 9a 71 42 08
3a dd 42 95 1e a9 7d 0c e4 a9 7a a6 13 f4 53 d4
11 62 99 06 70 4a 8d 24 f1 a2 69 ca 41 06 61 7c
44 14 f4 d4 34 04 a4 d4 3c b1 02 66 e6 81 ea d1
e2 78 20 c5 cf ef 8c c4 80 c2 be 3e f9 c8 8c 54
04 a4 d4 34 04 79 03 ac 44 86 b6 b0 77 9e 59 a6
e0 00 0c f6 61 06 d3 7e 28 2a ce 09 fa a3 f0 40
f5 18 05 00 00 a0 50 32 21 9e 6b ee f9 06 70 4a
8d 24 f1 a2 69 ca 41 06 61 7c 44 14 f4 d4 34 04
a4 d4 6c 31 d2 30 38 40 5c 43 7e 99 a6 e0 00 0c
f6 61 06 d3 7e 28 2a ce 09 fa a3 f0 40 f5 18 05
00 00 ed fd ae 59 8c 73 81 6a 1e f9 ef 98 4f 59
f0 80 5c 87 33 23 5c 71 18 50 d7 0e 2c 57 c5 22
```

Appendix B - Example output from Ent produced using the random numbers created using the middle square method

File-bytes	Entropy	Chi-square	Mean	Monte-Carlo-Pi	Serial-Correlation
16384	7.404508	17564.9375	125.13623	3.013919	0.025901
Value	Occurrences	Fraction			
0	231	0.014099			
1	24	0.001465			
2	97	0.00592			
3	65	0.003967			
4	421	0.025696			
5	47	0.002869			
…	…	…			