

# Introduction

## True Randomness – Do You Really Want It?

### Introduction

It is easy to forget exactly what random numbers are. They are really, truly, random. When used with a DVD, what we may really want is not a random number, but something which has some of the properties of a random number, but is actually something else.

Consider this scenario: A random number generator is used to select one of 5 clips. On the first pass through the random number generator, clip 4 plays.

It sound good so far.

Then clip2 plays, followed by clip 4 again, then clip 5 twice, then clip4, then clip2. Something is obviously wrong with the random number generator. Clip 5 has played twice in a row, clip 4 has played twice (although not in a row), and clip1 has not played at all. By the laws of statistics, surely each clip should have played at least once!

Nothing in the data just presented indicates anything is wrong with the random number generator. In fact, it appears to be behaving quite randomly.

What the laws of statistics really say is that it is equally probable that, given five choices, any one of the five may turn up at any time. It is quite possible to have clip 5 play five times in a row, and not play again for some time.

What statistics do suggest is, if we wait long enough, all clips will play an equal number of times. How long "long enough" is may be longer than you wish to wait, particularly when a DVD is concerned.

### Some definitions:

I'm going to drive many of you to distraction if I don't define my terms before I use them, although the definitions will make more sense in context than they do here. In this discussion, I'm going to use the following terms:

**Sequence:** The entire set of pseudo-random numbers produced before they repeat, and a pattern is revealed. It will start with the number first produced, and end with the last number produced before the sequence repeats.

**Bounds:** The lowest and highest pseudo-random numbers generated. This is usually 1 through some number greater than 1000.

**Range:** The set of random numbers the user may actually want; frequently a subset of the bounds.

**Pseudo-random numbers:** numbers which appear random, but in fact have been artificially generated, and may eventually repeat. They may also have some less than obvious pattern.

**Random numbers:** The thing we normally think of as true randomness. The numbers produced have no sequence, and possibly no bounds.

**Skewed pseudo-random numbers:** Pseudo-random numbered which have been modified to meet some additional criteria.

**Clip:** Usually a movie or movie segment (video), but nothing in the pseudo-random algorithm precludes branching to a menu or other addressable entity.

I do not claim these to be the official, approved terms (if such exist), just the terms I'll use.

## Computers and random numbers

It is extremely difficult, if not impossible, to generate a true random number in a computer. The problem is that everything is computed, which in turn means it is not random. Some pattern exists; the most we can hope for is that the pattern does not repeat for a long interval, and that the numbers generated appear to be random.

That is,

- 1) It appears to be equally probable that any number in the range desired may be generated at any given time.
- 2) When examined over some long time interval, each number in the range occurs approximately the same number of times.
- 3) As stated previously, the time before the pattern repeats is long with respect to the time the numbers will be used.

Another problem occurs when trying to get the random number generator to start at a different point in the sequence each time it runs. There is nothing in the computer which is truly random to use as a starting point for the sequence. Frequently, things like time of day are used. This isn't random, but it has enough possible different values that, used as a seed, it may suffice.

That means the same pseudo-random sequence will be generated if started at the same time the next day. Since this is unlikely, it is frequently adequate.

## Real random numbers

Having said all this, are there any real random numbers? Certainly.

- 1) Radio static, digitized.
  - 2) Cards with numbers on them, shuffled and dealt (but not by an honest faro dealer!)
- are two which immediately come to mind. Random numbers are frequently the result of some random physical event, which has been turned into numeric form.

Having said all this, is it possible to create a true, quasi-random seeded pseudo-random number generator, given the limited software capability of a DVD player? The answer is emphatically yes, and is addressed here.

I would characterize this algorithm as being reasonably well proven. I've tested it on a cheapie DVD player, and using Markham's Tray player under Windows XP. It performs as designed in both of these environments. A number of others have also tested the algorithm. Problems were found initially, but it is more typical now for the problem to be elsewhere in their implementation.

## Skewed Pseudo-Randomness

However, earlier in this discussion, it was stated that you may not really want a pseudo-random number generator. What you may really want is something I'd call a skewed pseudo-random number generator. That is, the results of a pseudo-random generator have been changed to force them to meet some additional criteria.

What and why? In the earlier example, clips played in the sequence 2, 4, 2, 4, 5, 4, 2. Often, it is desirable to have all 5 clips play before any clip plays again. This needs a skewed pseudo random generator. Or, if you prefer, cards dealt by an honest faro dealer.

In the example just cited, clip2 played first, and should not play until all four other clips play. What was probably desired was a play order something like this: 2,4,5,3,1, although it could obviously have varied slightly. The rules for this sequence would be something like this:

1. Pick a number in the desired range randomly.

2. Pick another number randomly, but exclude the number(s) previously picked.

3. Continue this process until all numbers are picked.

It is actually slightly harder to implement this than to implement the pseudo-random number generator, because the numbers picked must be remembered, and excluded. Two differing approaches are presented in these pages. The first generates real, honest-to-goodness pseudo-random numbers. The second generates skewed pseudo-random numbers. I strongly recommend you familiarize yourself with the basic pseudo-random generation, since the code to that routine is embedded in the skewed version.

I have created two skewed pseudo-random number generators, which appear in a separate tutorial. I strongly recommend you get the pseudo-random generator working, then add the seeding capability, then work with the skewing algorithms, if you are so inclined. I believe your troubleshooting will be far easier to perform if you follow this sequence.

# RANDU

RANDU was the name of a pseudo-random number generator distributed with mainframes and minicomputers in the 1960s and 1970s. I believe my random number generator uses a different mechanism than did those algorithms; the name was used for reasons of nostalgia.

DVD players, of course, contain a processor which emulates a standard instruction set computer. That instruction set reminds me a great deal of those earlier machines. This emulation contains no floating point instructions, and (IMHO) no good source of a random number.

Some might ask why a DVD player even needs a random number. On occasion, it is desirable to have one of several events to be chosen on a random basis. For example, one might want to vary the appearance of the DVD as it plays, selecting a different video at some point on a random basis.

Whatever the reason, the need for pseudo-random number generation has been a recurrent concern voiced by a number of different individuals. The problem is really in two parts:

- 1) Creating a real pseudo-random number generator which functions with an integer-only instruction set, and
- 2) Causing the pseudo-random number generator to start with a different number each time.

Item two is really harder to achieve than item one. When used in a player, a DVD is really a read-only device. The emulation software in the player appears to be read-only. If some location in the player memory comes up with a random value at power on, it has not, to my knowledge, been identified.

Thus, one is faced with causing some sort of user input to provide the starting point (or "seed") for the random number generator. The pseudo-random number generator described here is self-steering, and will begin the generation of pseudo-random numbers without a seed. However, a method of providing a seed is described.

The problem with integer-only pseudo-random number generation is that some pseudo-random number techniques have depended on the recurrent use of transcendental functions, making their implementation in an integer-only computer more difficult.

An integer-only random number generator must thus use an unconventional approach. The generator described here is based on prime numbers.

## Acknowledgments

I wish I could claim all the credit for this algorithm, but in fact much of it is based on earlier work. I first encountered this method of generating random numbers in 1969, when I implemented someone else's idea in hardware. That idea wasn't new then, and had been published for some years. My contribution is limited to the adaptation to DVD usage, and the use of the remainder to use a much larger pseudo-random number as a base for a small range of pseudo-random numbers.

I am indebted to Vapymid for suggestions which simplified and generally improved the final code. In addition, the task of transitioning from my spreadsheet and BASIC language models into code DVDLab could understand was made much easier by [The Unofficial DVD Specifications Guide](http://www.dvd-replica.com) (www.dvd-replica.com). I strongly recommend this document to anyone trying to write software for a DVD player to execute.

# REQUIREMENTS

This algorithm was designed around DVDLab Professional. The register naming conventions are those used by that program. These appear to be those used by most or possibly all DVD authoring programs. There is no reason why the algorithm should not work with other programs with similar capabilities.

You will need a copy of DVDLab Pro, or a similar program, to be able to use this code.

Two general purpose registers are required during the execution of the routine. One must be dedicated to this routine, and not used by any other routine on your DVD. The second may be used by other routines, but they must not depend on its contents being retained.

If you use the seed technique, you will need a third general purpose register to use as a counter, in addition to the two previously mentioned. This register will be released for reuse. However, any contents of the register will be lost.

For simplicity's sake, I have used GPRM0, GPRM1, and GPRM2. However, any general purpose register may be used, provided they meet the requirement listed above.

I recommend you do initial testing with Markham's Tray player, as I have used it, and know it to be capable of executing the code for both the unseeded and seeded pseudo-random generators. I had little luck with other computer based players. You should expect a certain amount of difficulty while integrating the code into your DVD; it will be easier to troubleshoot it in your computer.

# IMPLEMENTATION

## Initialization

Some software requires an initialization section to set up values that will be used later, and a continuation section which does the actual processing. RANDU can have an initialization section to establish a pseudo-random number seed. That section looks like this:

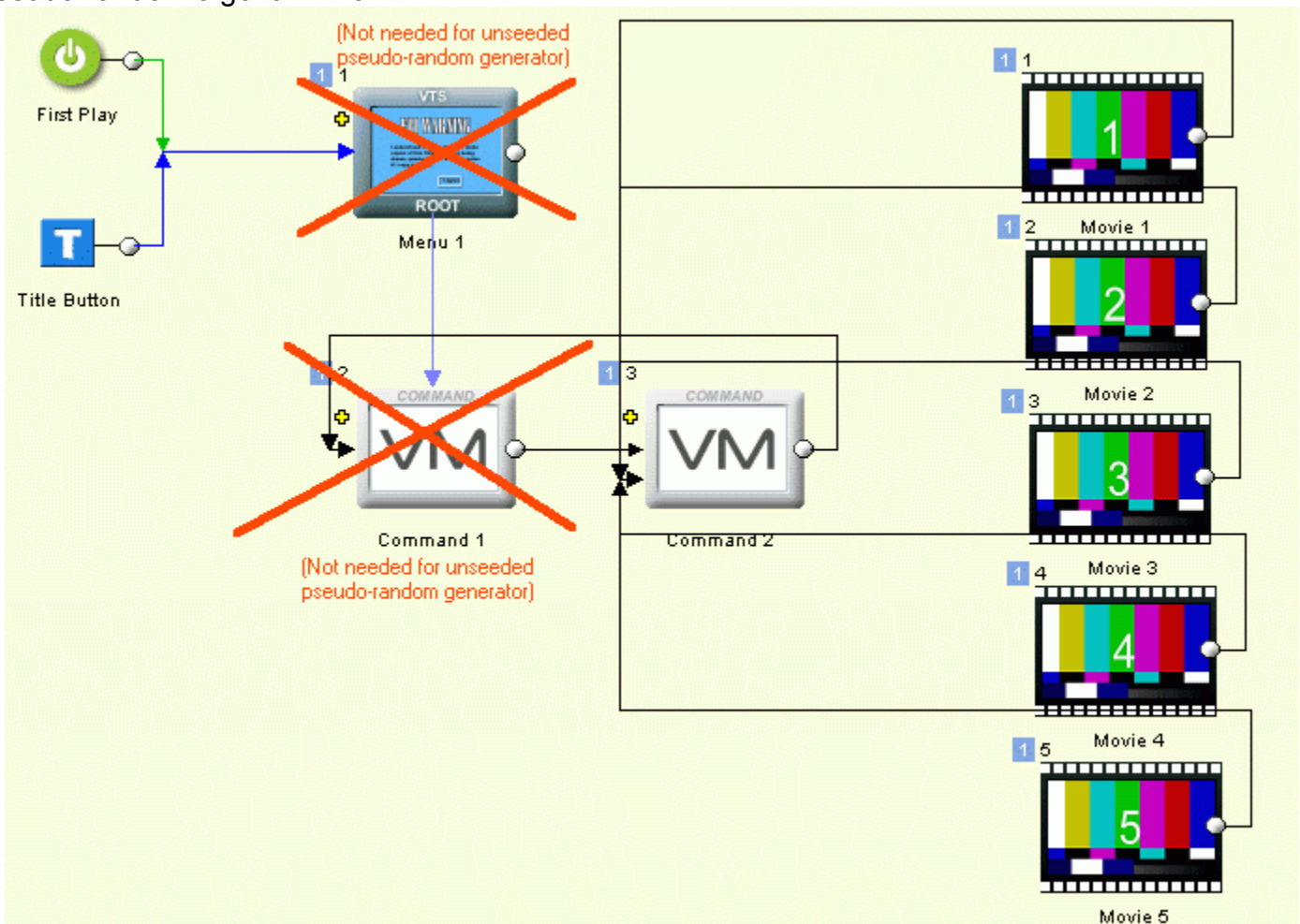
### GPRM0 = initial random value

The "initial random value" above just means that you snagged an initial pseudo-random value from somewhere else. If you don't have a source for an initial pseudo-random value, then don't worry about it. The good news is that RANDU will just keep churning out numbers, even if you don't give it a seed. This seed will be ignored for now, but be aware that it can be added later.

## Connections

The connections to implement RANDU look like this. You don't need either the menu or the functional block labeled "Command 1" yet. However, if you plan to implement the "seed" portion of the algorithm later, it will be easier if you include the menu segment and Command 1 at this time.

The menu can be almost anything, as long as it has a single selection box that points towards Command 1. Command 1 and command 2 are containers for the commands that will make the pseudo-random algorithm work.



## Continuation

The continuation section looks like this. For the moment, just look at the part in black. We'll discuss the part in red later. As discussed in the theory section, the pseudo-random number being generated is actually 11 bits long. The method used to generate the 11 bits disallows the bits from being all one or all zero. The first two lines of the code do nothing but insure that the number never gets in these states.

Another problem which could occur is that the number might accidentally grow to be greater than 11 bits. If this were to happen, the pseudo-random number generator would likely cease to function correctly. Line 3 does nothing but force the number to be only 11 bits long.

GPRM0 gets copied to GPRM1 in line four. GPRM1 will be used to determine the next number to appear in GPRM0. However, we don't want to change the value of GPRM0 until that calculation is performed.

The algorithm works by comparing the values of two bits, making a calculation, and manipulating the random number, based on the result. Lines six and seven check these two bits. If the two bits do not have the same value, 2048 is added to GPRM0, and the result is divided by two in line 8. Hardware minded readers may recognize this as shifting a "one" into the left side of an eleven bit register.

The result is a new pseudo-random number in GPRM0, which is then copied into GPRM1. But we normally want to chose between just a few items, not 2047 items. Something must be done to allow us to transform this random value into the range we desire. That will be discussed below.

```
1 if (GPRM0 == 0) GPRM0 = 2046 - zero is an illegal state in the shift register
2 if (GPRM0 == 2047) GPRM0 = 2046 - another illegal state
3 GPRM0 &= 2047 - get rid of all but lower 11 bits
4 GPRM1 = GPRM0
5 GPRM1 &= 17 - GPRM1 is now the result of the test for both bits 11 and 7.
6 if (GPRM1 == 1) GPRM0 |= 2048 - set bit 12 if bits 0 and 4 are different (1 0)
7 if (GPRM1 == 16) GPRM0 |= 2048 - set bit 12 if bits 0 and 4 are different (0 1)
8 GPRM0 /= 2 - shift random number right one bit GPRM1 %= 5 - bound the random number by
the allowable values
9 GPRM1 = GPRM0
10 GPRM1 %= 5 - bound the random number by the allowable values
11 if (GPRM1 < 1) Goto 16
12 if (GPRM1 < 2) Goto 17
13 if (GPRM1 < 3) Goto 18
14 if (GPRM1 < 4) Goto 19
15 JumpVTS_TT 5
16 JumpVTS_TT 1
17 JumpVTS_TT 2
18 JumpVTS_TT 3
19 JumpVTS_TT 4
```

That magic is performed by the lines ten through nineteen of the code, shown above with a red background. Line ten makes use of the fact that, so long as you divide a series of pseudo-random numbers by the same constant, the resulting series is still pseudo-random. Further, the remainders are pseudo-random.

The "5" in GPRM1 %5 is the number of choices to be made. The pseudo-random number in GPRM1 will be divided by the number of choices (5), and the remainder left from the division left in GPRM1.

There may be no remainder (a value of zero), or as much as four.

The branches following this use this remainder to link to one of five clips. If line fourteen seems confusing, consider that if the number wasn't less than 1,2,3, or 4, then (in this case) it must have been 5. However, on the off-chance, however unlikely, that some number bigger than 5 is in the register, this avoids branching off to some unknown or possibly undefined area.

## Application Comments

1) If you do not have 5 choices, then line 10 is modified to reflect the number of choices you have. For example, if you wanted to randomly choose between two videos, only lines 10,11, and 12 would be present. They would read:

```
GPRM1 % 2  
if (GPRM1 <1) JumpVTS_TT 1  
JumpVTS_TT 2
```

But I emphasize that it could just as easily have been LinkPGCNx, linking to another menu.

2) You must check the clip numbers of the video segments to which you wish to link. There is no guarantee that they will be numbered one through five, or whatever. The random numbers produced will always have the range 0 through one less than the total number of clips you designated. I recommend you build yourself a table like the one below in a spreadsheet or somewhere, so that you know which pseudo-random number should branch you where.

Pseudo Random Number	Links to
1	Movie 1
2	Movie 2
3	Command 4
4	Movie 3
5	Movie 5
6	Movie 6

3) Be sensitive to the number of random selections you make. Although movies can be numbered above 99, only 99 movies can be present at any one time. All of this logic is also going to take processing time within the DVD player, and may cause you frozen images or flickering on less expensive players.

4) If you branch to a chapter of a movie, you will not return to the pseudo-random generator before the rest of the movie plays.

My original implementation had much of the code embedded within movie clips. I knew there had to be a way to get the code to function within the VM menus; it just took a little time and a bit more understanding to get there.

# THEORY

## RANDOM NUMBER DISCUSSION

Before digging into how the pseudo-random number generator actually works, some further discussion of random numbers is probably appropriate. Rather than try to provide some sort of obtuse definition, I will try to discuss their characteristics.

First, a real random number could be any number, positive or negative. It would be just as likely to be several billion as to be, say, 5. That doesn't do us much good when we want to randomly choose between which of a relatively few video clips we wish to play. For this reason, the pseudo-random numbers generated here are bounded by 1 and 2046, and are further scaled to have values in the specific range needed, such as 0 to 4.

Second, when a number is random, it is just as likely to occur again immediately as it is to not occur for some time. However, if the numbers are generated for a long enough time, all numbers within the bounds will be generated an equal number of times. If one ran the algorithm presented here 2045 times, one would find that each number between 1 and 2046 occurs exactly once, but in an apparently random fashion. When the number is scaled into the range desired, it will be found that each scaled number appears an approximately equal number of times.

For example, with a range of 5, and a run of 2045, zero would appear approximately 409 times, one would appear 409 times, and so on. You may see zero appear four times in a row, or once, then not again for the next twenty tries. That's just the nature of randomness: it really is random! But in the 2045 cycles, each number from 0 to 4 will appear approximately 409 times. Of course, if you used a different range, then each number would appear  $(2045/\text{range value})$  times.

Having said all this, how can we be sure that the numbers in the range we chose are still truly random?

One of the properties of a pseudo-random number is that, if all the random numbers in a sequence are divided by the same constant, the results are themselves pseudo-random numbers. Perhaps nicer still, if we are dealing with integer pseudo-random numbers and the divisor is an integer:

- 1) The remainders are themselves pseudo-random numbers, and
- 2) The remainders have the values 0 through range. That is, if the range is 5, the remainders will be a pseudo-random sequence of the numbers 0,1,2,3, and 4. That's exactly what we want!

## GENERATION METHOD

Figures 1 and 2 below demonstrate how the pseudo-random sequence is generated. First, we start with a seed. This is some number in the bounds of the random numbers to be generated, and must be greater than zero, and less than 2047. Those two numbers "lock up" the generator used in this algorithm. Since all integers between 0 and 2047 will be generated once and only once, and always in the same sequence, by selecting a seed, we have selected where in the sequence we will start.

This seed is the first pseudo-random number. It will be used to generate the next pseudo-random number, which will be used to generate the next pseudo-random number, and so on. As shown in figure 1, bits 10 through 1 are all copied one bit to the right in the next pseudo random number. In the code, that's the place where everything is divided by 2. That's the same as a shift right, or a copy of each bit one bit to the right.

A seed is not necessary to get this pseudo-random generator to run. It will start a default sequence without being initialized. However, it will create the same sequence each time it is started.

Seed (0<seed,2048) => 721											Random Number Generated	Value to use for branching
Range =====> 5												
1	2	3	4	5	6	7	8	9	10	11		
Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
0	1	0	1	1	0	1	0	0	0	1	721	1
0	0	1	0	1	1	0	1	0	0	0	360	0
0	0	0	1	0	1	1	0	1	0	0	180	0
1	0	0	0	1	0	1	1	0	1	0	1114	4
1	1	0	0	0	1	0	1	1	0	1	1581	1
1	1	1	0	0	0	1	0	1	1	0	1814	4
1	1	1	1	0	0	0	1	0	1	1	1931	1
1	1	1	1	1	0	0	0	1	0	1	1989	4

The next step is to determine the value of bit 10. As shown in figure 2, bit 10 is determined by examining bits 0 and 4. If these bits have the same value (both 0 or both 1), bit 10 is set to 0. If bits 0 and 4 have differing values (0 1 or 1 0), bit 10 is set to 1. Hardware minded readers may recognize this as an exclusive OR (XOR) of bits 0 and 4.

Seed (0<seed,2048) => 721											Random Number Generated	Value to use for branching
Range =====> 5												
1	2	3	4	5	6	7	8	9	10	11		
Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
0	1	0	1	1	0	1	0	0	0	1	721	1
0	0	1	0	1	1	0	1	0	0	0	360	0
0	0	0	1	0	1	1	0	1	0	0	180	0
1	0	0	0	1	0	1	1	0	1	0	1114	4
1	1	0	0	0	0	0	1	1	0	1	1581	1
1	1	1	0	0	0	1	0	1	1	0	1814	4
1	1	1	1	0	0	0	1	0	1	1	1931	1
1	1	1	1	1	1	1	0	0	0	1	1989	4
1	1	1	1	1	1	1	0	0	0	1	2018	3
0	1	1	1	1	1	1	1	0	0	1	1009	4
0	0	1	1	1	1	1	1	0	0	0	504	4
1	0	0	1	1	1	1	1	1	1	0	1276	1
1	1	0	0	1	1	1	1	1	1	0	1662	2
1	1	1	0	0	1	1	1	1	1	1	1855	0
0	1	1	1	0	0	1	1	1	1	1	927	2
0	0	1	1	1	0	0	1	1	1	1	463	3
1	0	0	1	1	1	0	0	1	1	1	1255	0
1	1	0	0	1	1	1	0	0	1	1	1651	1
0	1	1	0	0	1	1	1	0	0	1	825	0
0	0	1	1	0	0	1	1	1	0	0	412	2
1	0	0	1	1	0	0	1	1	1	0	1230	0
0	1	0	0	1	1	0	0	1	1	1	615	0
1	0	1	0	0	1	1	0	0	1	1	1331	1
0	1	0	1	0	0	1	1	0	0	1	665	0

That's it! The next pseudo-random state has been generated. The pseudo-random number

generated is shown in base 10, in the next column to the right of the binary value. The remainder, after dividing by the range, is shown in the next column. If you were to perform statistics on those remainders, you would find that each number from 0 to (range-1) is approximately equally distributed.

Why does it work? It may not be obvious, but this is based on the properties of prime numbers. If you start counting from the left, bit 0 is the eleventh bit; bit four is the seventh bit. Any binary word whose length is a prime number, whose low order bit is compared to a bit a prime number of steps from the left in the manner shown here, can be used to create a pseudo-random number sequence. Its length will be  $2^{(n-1)}$ , where n is the length of the binary sequence.

Selection of different binary word lengths and the choice of different prime points thus impact the total sequence length. It also drives the number of sequential 1s or 0s in the pseudo-random binary word generated. I chose 7 and 11 based on the length of sequence needed, and the need to avoid long strings of 1's or 0's in the words generated.

That could have had adverse effects on the remainders. While the remainders might have still been technically pseudo-random, a given number might have occurred more times in a row than desirable in this application.

If I have any brother or sister physicist is reading this, they will understand my great joy at making a relatively useless area of math to do something useful. I will probably be blacklisted in math departments for my part in this, which is the highest accolade a physicist could hope for. :)



Latest revision: 20050321