Random Number Generator in R (and Video Games)

Andrew Tunggal

October 31, 2017

Introduction:

If you've ever rage-quit because your opponent gets a critical hit on you, or lucked out by getting a clutch dodge when your opponent has a 95% hit rate, then you've experienced both the joy and the frustration that is a result of random number generators (RNG for short).

In the gif above, we see that we lucked out in the RNG with a critical hit, even though out critical rate is only 5%.

We will be looking into the basics behind this principle that is quite apparent in games such as Pokemon, Hearthstone, Fire Emblem, and even League of Legends and DOTA.

Motivation:

Many video games involve random number generators. There have been many a times where my character misses an attack at 97%, while my opponent hits a critical on me at 1% critical rate. This doesn't even include the bad level up boosts that are also due to RNG. Since I was younger, I've always been interested in how RNG works, and even wanted to look into how to manipulate RNG for some video games such as Pokemon (gotta find a way to get those rare and shiny Pokemans). However, that never worked because it turned out to be too much work… there are some who have go very far into the analysis of RNG in particular games, such as this RNG Manipulation Thread for Pokemon

Here, I'll just talk about a simple implementation of Random Number Generation.

Basics:

As quoted by Wikipedia:

"Random number generation is the generation of a sequence of numbers or symbols that cannot be reasonably predicted better than by a random chance, usually through a random-number generator (RNG)"

The basic principle behind RNG is that you have a set of numbers. For example, you have a set like below:

1 2 3 4 5

There is the option to sample with or without replacement. Random Number Generators sample with replacement, which would make it so that you have an equal likelihood of getting any number in the set at every simulation. In a random number generator using the above values, it is possible to get ten 1's in a row (very improbable, but still a chance)

This isn't just seen in video games, but seen in many typical examples used in stats classes such as dice rolling, coin flipping, shuffling straws, and even seen in instances such as slot machines.

HOWEVER, as one may notice, the numbers that we obtain from an RNG are not actually TRULY random. RNGs most of the time (if not all of them) generate "pseudorandom numbers". This is because there is a part called "the seed", which is basically your starting point for the generation of the random number.

In-Game Example:

Though there are different factors that can impact the RNG in a game (such as internal factors), we will look at an in-game example and base it purely off RNG.

So we have a Fire Emblem: Radiant Dawn character named Sothe. We see that his growth rates in this game are:

HP: 30% | Str: 60% | Mag: 20% | Skl: 80% |
Spd: 45% | Lck: 65% | Def: 20% | Res: 30% |

Below we see an example of a level up:

Using the principle of RNG, looking at this level up, we can determine that the first generated number is between 1-30 because there was an increase, a growth, in the particular stat (there is no 0 in the RNG in this game). This is the case for all the stats that were raised, with the randomly-generated number being between 1 and the value for their respective growth rates.

However, if you look at the stats for Speed and Luck, there was no increase. That means the fifth randomly generated number was between 46 - 100, and the sixth generated number (for luck) was between 66 - 100.

Uses in R:

Random number generating is a very useful tool to use, especially when trying to draw conclusions about the data from a population or sample. If you are given a sample, and want to bootstrap from the sample in order to draw a conclusion about the unknown population, one would randomly generate numbers/values from the given data set.

Two Functions in R: RUNIF and SAMPLE

There are two main functions in R for generating random numbers: the runif function, indicating random uniform distribution, and the sample function.

As indicated above, both are used to generate random numbers. However, when you run the two functions below, you will notice an important difference between the two:

```
# runif(n, x, y)
# n = number of generations
# x = lower bound, y = upper bound
runif(5, 0, 2)
```

```
## [1] 0.2241715 0.3280584 1.0173264 1.0241183 0.8943447
```

```
# sample(x:y, n)
# x:y = range for values to be sampled
# n = number of items to be obtained
sample(0:2, 5, replace = TRUE)
```

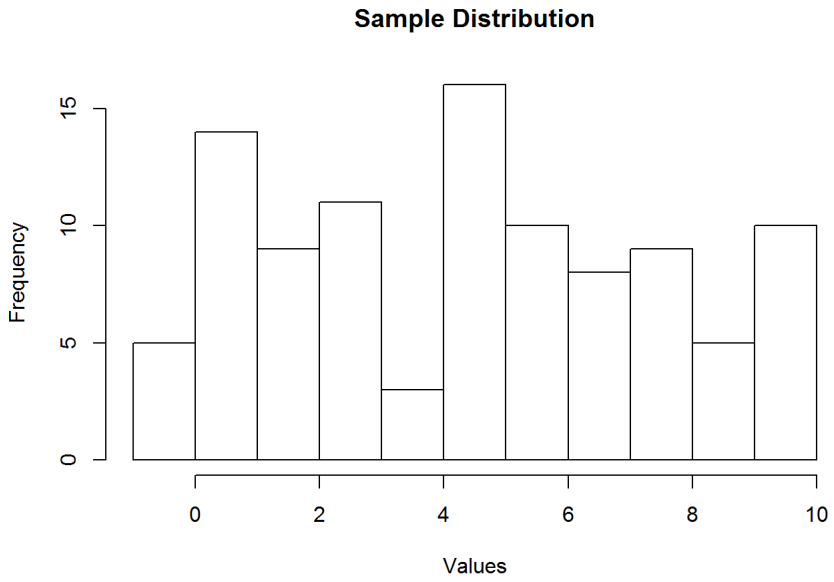
```
## [1] 0 0 0 2 1
```

RUNIF is used to generate any random real number between the indicated range. Thus, you are able to get numbers with multiple decimal points. Meanwhile, using the sample function, you are obtaining samples of size n within a range x:y. However, if you want there to be replacements of values (which is often the case in random number generators), you need to add in a variable indicating that “replace = TRUE”

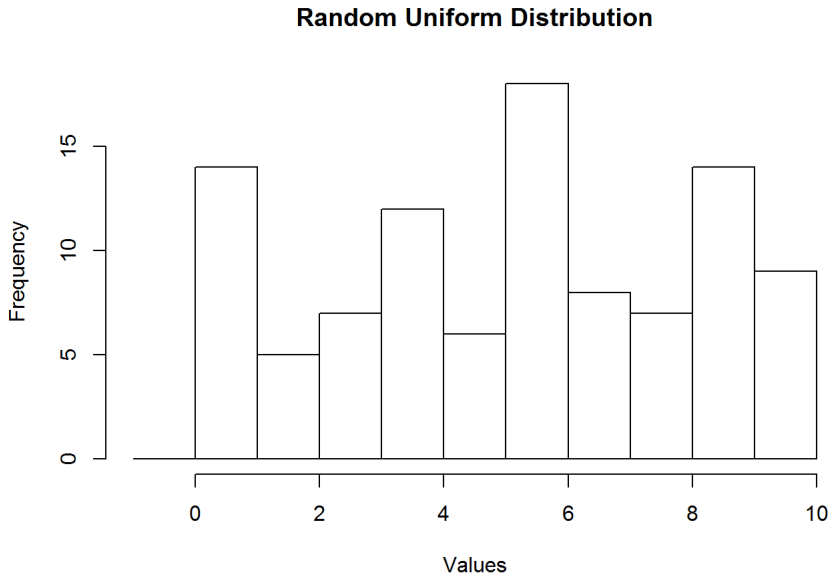
If we make a histogram showing the values of the random uniform distribution and the sample:

```
# Variables to represent the two different functions
samp <- sample(0:10, 100, replace = TRUE)
run <- runif(100, 0, 10)

# A histogram of the values obtained using the sample function
hist(samp, breaks = seq(-1, 10, 1), main = "Sample Distribution", xlab = "Values")
```

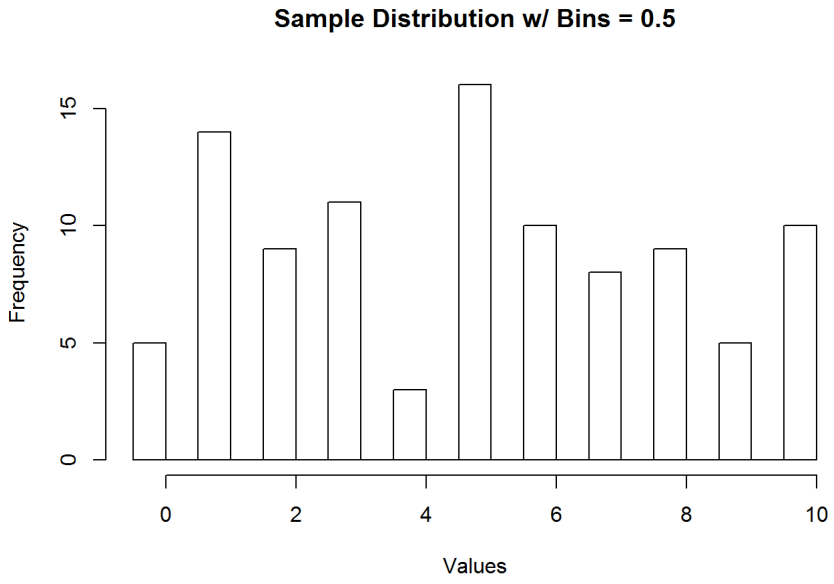



```
# A histogram of the values obtained using the runif function
hist(run, breaks = seq(-1, 10, 1), main = "Random Uniform Distribution", xlab = "Values")
```

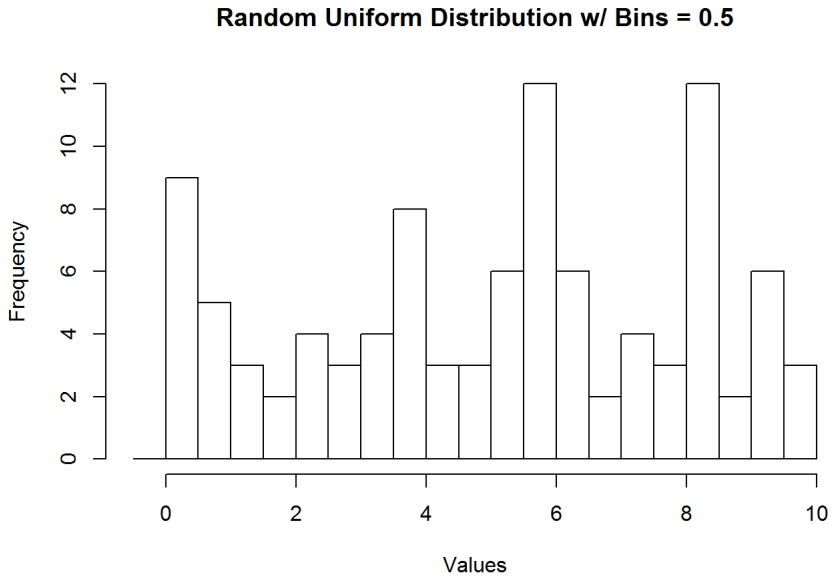


You will see above that they look quite similar at the moment. The only difference is that there is a very low chance of you getting a value that is exactly 0 in the random uniform distribution (compared to 1/11 chance for the sample). Thus, often times you will not find a value in the bin for zero, while for the sample distribution, because there is a high chance to get a zero, there is an extra bin added. However, if you decrease the size of the bins:

```
# A histogram of the sample function with bins = 0.5
hist(samp, breaks = seq(-0.5, 10, 0.5), main = "Sample Distribution w/ Bins = 0.5", xlab = "Values")
```




```r
# A histogram of the runif function with bins = 0.5
hist(run, breaks = seq(-0.5, 10, 0.5), main = "Random Uniform Distribution w/ Bins = 0.5", xlab = "Values")
```



You can see that while the histogram for the sample function has gaps, the histogram for the random uniform distribution showcases no gaps, with values filling in the gaps that would be there for the sample function.

Let's Simulate Sothe's Level-Ups:

Now we can try to simulate the example to see what kinds of level ups we'd get. For these intents and purposes, we will be using the sample() function. Because he has already leveled up to Level 2, he will level up 18 more times before getting to level 20 (which is the max level before class change). We can simulate the kinds of level ups he will get:

```r
# Creating a data frame that has the randomly generated values for level-ups
Sothe <- data.frame(matrix(ncol = 8, nrow = 18))
for (i in c(1:18)) {
  Sothe[i,] <- sample(1:100, 8, replace = TRUE)
}
names(Sothe) <- c("HP", "Str", "Mag", "Skill", "Sp", "Lck", "Def", "Res")
Sothe
```

```
##      HP Str Mag Skill  Sp Lck Def Res
## 1    13  69  56    48  26  92  73  66
## 2    33  63  57    96  29  80  11  32
## 3    92  57  85    44  61  94  59  12
## 4    96   5  14    23  34  11  42  44
## 5    60  43  93    18  64  22  43  21
## 6     9  70  21    58  25  68  46  39
## 7    50  41  13    52   7  27  20   2
## 8    16  26  78    60  60  29  74  44
## 9    61  70   9    10  88  25  46  93
## 10   56  13  63    25 100  43  76  89
## 11   68  74  34    10  15  68   9  29
## 12   65  18  93     7  55  21  95   8
## 13   85  39  95    40   5  28  89  79
## 14  100   9   4    45  83  98  66   2
## 15   41  78  86    58  98  56  63  48
## 16   12   1  52    54   2  57  49  69
## 17   72  37   7    91  15  74  16  69
## 18   19  32  71    28  63  93  85  19
```

This will give you a simulation of his next 18 level ups.

You can take these values and simulate to obtain a guess-timate of what you might expect Sothe's stats at level 20 to be.

```
# Obtaining the number of times each value would result in a stat increase, given Sothe's growth rates
hp <- nrow(Sothe[Sothe$HP<= 30, ])
str <- nrow(Sothe[Sothe$Str<= 60, ])
mag <- nrow(Sothe[Sothe$Mag<= 20, ])
skill <- nrow(Sothe[Sothe$Skill<= 80, ])
sp <- nrow(Sothe[Sothe$Sp<= 45, ])
lck <- nrow(Sothe[Sothe$Lck<= 65, ])
def <- nrow(Sothe[Sothe$Def<= 20, ])
res <- nrow(Sothe[Sothe$Res<= 30, ])

# A data-frame with Sothe's final stats at Level 20
final_stats <- data.frame(matrix(ncol = 8, nrow = 1))

# Adding the randomly-generated values for stat boost to his current stats at Level 2 (based off the picture above
)
final_stats[1, 1] <- hp + 36
final_stats[1, 2] <- str + 19
final_stats[1, 3] <- mag + 5
final_stats[1, 4] <- skill + 21
final_stats[1, 5] <- sp + 20
final_stats[1, 6] <- lck + 15
final_stats[1, 7] <- def + 15
final_stats[1, 8] <- res + 10

names(final_stats) <- c("HP", "Str", "Mag", "Skill", "Sp", "Lck", "Def", "Res")

final_stats
```

```
##   HP Str Mag Skill Sp Lck Def Res
## 1 41  31  10    37 29  25  19  17
```

Conclusion:

RNG is vital to the mechanics of so many games that people know and play.

Though we have explored its relevance and use in the context of video games, one mustn't forget that the usefulness and application of RNG extends far beyond merely video games. Randomly generating values is an important tool in data analysis, such as making predictions about a population paramter when one is only given a sample (bootstrapping).

Now I'm gonna go play Fire Emblem (because making this post made me want to play…)

If you want to read more about the uses of RNG in video games, you can visit this website.

References:

https://lparchive.org/Fire-Emblem-Radiant-Dawn/Update%2011/50-RFEP01-83.jpg

http://www.makeuseof.com/tag/lesson-gamers-rng/

http://fireemblem.wikia.com/wiki/Sothe

https://www.reddit.com/r/pokemonrng/

https://lparchive.org/Fire-Emblem-Blazing-Sword/Update%2022/35-piratemage.gif

https://en.wikipedia.org/wiki/Random_number_generation

https://www.thoughtco.com/what-is-bootstrapping-in-statistics-3126172

https://softwareengineering.stackexchange.com/questions/109724/how-do-random-number-generators-work