**Introduction**

I *love* [Arkham Horror; The Card Game](https://www.fantasyflightgames.com/en/products/arkham-horror-the-card-game/). I love it more than I really should; it’s *ridiculously* fun. It’s a cooperative card game where you build a deck representing a character in the [Cthulhu mythos](https://en.wikipedia.org/wiki/Cthulhu_Mythos) universe, and with that deck you play scenarios in a narrative campaign[1](https://ntguardian.wordpress.com/2018/08/13/evaluating-olive-mcbride-arkham-horror-lcg-chaos-bag-simulator-r/#fn-3304-1) where you grapple with the horrors of the mythos. Your actions in (and between) scenarios have repercussions for how the campaign plays out, changing the story, and you use experience points accumulated in the scenarios to upgrade your deck with better cards.

The game is hard. *Really* hard. And I don’t mean it’s hard because it has a learning curve (although if you’re not used to deep games like this, it probably does have a learning curve). This is a Cthulhu game, which means that the mythos is going to do everything to beat you down. Your best-laid plans are going to get derailed. The worst thing that could possibly happen to you *will* happen. This is why this game is unreasonably fun: it’s a game for masochists.

And it makes a damn ton of good stories. I often want to tell the tale of how amazingly awful the events played out, or how I just snatched victory out of the jaws of defeat. Bloomberg recently published [an article about the rise of board games in companies](https://www.bloomberg.com/news/articles/2018-04-26/wall-street-s-latest-trend-in-networking-board-game-nights). If you’re in one of these companies where employees often play board games together after hours, you should consider adding Arkham Horror (the card game; I’ve never played the board game and I’ve heard mixed reviews) to the mix. It’s a great team builder.

Two elements make Arkham Horror so damn hard and unpredictable: the encounter deck and the chaos bag. The encounter deck is a deck that all players draw from every turn that spawns monsters and awful events. The chaos bag is used for skill test you make for advancing your board state. You draw tokens from the chaos bag, add the modifier of the revealed token to your skill value for that skill test (most of them are negative numbers), check the new value against the difficulty of the test, and if your modified skill value is at least as great as the difficulty of the test, you pass; otherwise, you fail. (You can pitch cards from your hand to increase your skill value *before* you reveal a token.)

  
\*(Image source: [Ars Technica](https://arstechnica.com/gaming/2017/04/the-arkham-horror-card-game-you-got-some-rpg-in-my-ccg/))

This is the chaos bag in a nutshell, but the chaos bag does more. The elder sign token represents good luck and often helps your board state; it’s considered a great success. But most icon tokens in the bag are trying to hurt you. There are four icon tokens: the skulls, the cultist, the tablets, and the elder things. These often not only apply a modifier but do something bad to you (which changes between scenarios).

And of course there’s this little bastard.



This is the auto-fail token. You just fail. Thanks for playing.

**Simulating the Chaos Bag**

Bad things happen in Arkham Horror. Sometimes, though, they feel like they happen a lot. For example, you can draw two auto-fails in a row. Or three. I cannot understate how devastating that can be in a game. Sometimes it feels like this happens a lot. Sometimes it feels like this particular game went unusually poorly, and unusually poor games seem to happen frequently.

That’s a contradiction, of course. I think that this sense of bad games happening often emerges from humans fundamentally poor understanding of probability and how it actually works. I’m not just referring to the mathematics; people’s intuition of what should happen according to probability does not match what probability says happens. This phenomenon is well-documented (see, for example, the book [*Irrationality*, by Stuart Sutherland](https://www.amazon.com/Irrationality-Enemy-Within-Stuart-Sutherland/dp/1780660251)) and is one of the explanations for why people seemed to underestimate Donald Trump’s chances of winning the 2016 election (in short, unlikely events occur more frequently than people perceive; see also [Nate Silver’s series of articles](https://fivethirtyeight.com/features/the-real-story-of-2016/)). In fact, you are more likely to find *any* pattern than no pattern at all (and in Arkham Horror, patterns are usually bad for you.)

This perception of “unusually many auto-fails” (which, as a statistician, I know cannot be right no matter what my feelings say) prompted me to write an R function that generates a string of pulls from the chaos bag, each pull independent of the previous pull. Here’s the function:

chaos\_bag\_string\_sim <- function(dicebag = list('E' = 1, # Elder Sign

'P' = 1, # +1

'0' = 2,

'1' = 3, # -1

'2' = 2, # -2

'3' = 1,

'4' = 1,

'S' = 2, # Skull

'C' = 1, # Cultist

'T' = 1, # Tablet

'L' = 1, # Elder Thing

'A' = 1), # Autofail

n = 24, replace = TRUE) {

paste0(sample(names(dicebag), replace = replace,

prob = as.vector(unlist(dicebag)),

size = n), collapse = '')

}

Notice that the function takes a list, dicebag. This list uses one-character codes representing chaos bag tokens, and the actual contents of the list are numbers representing the frequency of that token in the bag. The bag that serves as a default is a fictitious chaos bag that I believe is representative of Arkham Horror on standard difficulty. Since most numbered tokens are negative, I denote the +1 token with 'P' and the negative tokens with their numbers.

How many pulls from the bag occur in a game? That's really hard to figure out; in fact, this will vary from scenario to scenario dramatically. So let's just make an educated guess to represent the "typical" game. A round consists of a mythos phase, then three actions per player, followed by a monsters phase and an upkeep phase. The mythos phase often prompts skill tests, and I would guess that the average number of skill tests made during each player's turn is two. We'll guess that about half of the draws from the encounter deck prompt skill tests. A game may last about 16 rounds. This adds up to about 40 skill tests per player per game. As a consequence, a four-player game (I mostly play multiplayer, and the more the merrier) would yield 160 skill tests.

Let's simulate a string of 160 skill tests, each independent of the other[2](https://ntguardian.wordpress.com/2018/08/13/evaluating-olive-mcbride-arkham-horror-lcg-chaos-bag-simulator-r/#fn-3304-2).

(game <- chaos\_bag\_string\_sim(n = 160))

[1] "311113E12121E2LS20A32A4ETT1SE3T3P0AT2S12PLSC033CP22A11CLL31L024P12ES24A1S0E

E3PSSS12C13T21224104L43PCT13TTCSSASE20SA1TT2SSC2CPC20012CC41S234PPEP101E0LS

1P1110TSS1"

I represented pulls from the chaos bag as a string because we can use [regular expressions](https://en.wikipedia.org/wiki/Regular_expression) to find patterns in the string. For instance, we can use the expression AA to find all occurances of double-auto-fails in the string.

grepl("AA", game)

[1] FALSE

The grepl() function returns a boolean (logical) value identifying whether the pattern appeared in the string; in this game, no two auto-fails appeared in a row.

What about two auto-fails appearing, separated by two other tests? This is matched by the expression A..A (remember, . matches any character).

grepl("A..A", game)

[1] TRUE

We can take this further and use our simulator to estimate the probability events occur. After all, in probability, there are two ways to find the probability of events:

1. Create a probability model and use mathematics to compute the probability an event of interest occurs; for example, use a model of a die roll to compute the probability of rolling a 6
2. Perform the experiment many times and count how many times the event occured in these experiments many times and count how often the event of interest occured; for example, roll a die 1000 times and count how many times it rolled a 6 (this is usually done on a computer)

While the latter approach produces estimates, these estimates are guaranteed to get better the more simulations are performed. In fact, if you want at least a 95% chance of your estimate being accurate up to two decimal places, you could perform the simulation 10,000 times.[3](https://ntguardian.wordpress.com/2018/08/13/evaluating-olive-mcbride-arkham-horror-lcg-chaos-bag-simulator-r/#fn-3304-3)

Using this, we can estimate the probability of seeing two auto-fails in a row.

mean(grepl("AA", replicate(10000, {chaos\_bag\_string\_sim(n = 160)})))

[1] 0.4019

There's about a 40% chance of seeing two auto-fail tokens in a single four-player game. That's pretty damn likely.

Let's ask a slightly different question: what is the average number of times we will see two autofails in a row in a game? For this we will want to dip into the **stringr** package, using the str\_count() function.

library(stringr)

str\_count(game, "AA")

[1] 0

str\_count(game, "A..A") # How many times did we see auto-fails separated by two

# tests?

[1] 1

Or we can ask how often we saw three "bad" tokens in a row. In Arkham Horror on standard difficulty, players often aim to pass a test when drawing a -2 or better. This means that "bad" tokens include -3, -4, auto-fail, and often the tablet and elder thing tokens, too. The regular expression pattern that can match "two bad things in a row" is [34TLA]{2} (translation: see either 3, 4, T, L, or A exactly two times).

str\_count(game, "[34TLA]{2}")

[1] 14

How could we estimate the average number of times this would occur in a four-player game? The simulation trick still works. Pick a large number of simulations to get an accurate estimate; the larger, the more accurate (but also more work for your computer).

mean(str\_count(replicate(10000, {chaos\_bag\_string\_sim(n = 160)}), "[34TLA]{2}"))

[1] 10.6717

You can imagine that this can keep going and perhaps there are queries that you would like to see answered. Below I leave you with an R script that you can use to do these types of experiments. This script is designed to be run from a Unix-flavored command line, though you could source() the script into an R session to use the chaos\_bag\_string\_sim() function interactively. Use regular expressions to define a pattern you are interested in (this is a good opportunity to learn regular expressions for those who want the exercise).

#!/usr/bin/Rscript

#######################################

# AHChaosBagSimulator.R

#######################################

# Curtis Miller

# 2018-08-03

# A script for simulating Arkham Horror's chaos bag

#######################################

chaos\_bag\_string\_sim <- function(dicebag = list('E' = 1, # Elder Sign

'P' = 1, # +1

'0' = 2,

'1' = 3, # -1

'2' = 2, # -2

'3' = 1,

'4' = 1,

'S' = 2, # Skull

'C' = 1, # Cultist

'T' = 1, # Tablet

'L' = 1, # Elder Thing

'A' = 1), # Autofail

n = 24, replace = TRUE) {

paste0(sample(names(dicebag), replace = replace,

prob = as.vector(unlist(dicebag)),

size = n), collapse = '')

}

# optparse: A package for handling command line arguments

if (!suppressPackageStartupMessages(require("optparse"))) {

install.packages("optparse")

require("optparse")

}

main <- function(pattern, average = FALSE, pulls = 24, replications = 10000,

no\_replacement = FALSE, dicebag = "", help = FALSE) {

if (dicebag != "") {

dicebag\_df <- read.csv(dicebag, stringsAsFactors = FALSE)

dicebag\_df$token <- as.character(dicebag\_df$token)

if (!is.numeric(dicebag\_df$freq)) {stop("Dicebag freq must be integers.")}

dicebag <- as.list(dicebag\_df$freq)

names(dicebag) <- dicebag$token

} else {

dicebag = list('E' = 1, # Elder Sign

'P' = 1, # +1

'0' = 2,

'1' = 3, # -1

'2' = 2, # -2

'3' = 1,

'4' = 1,

'S' = 2, # Skull

'C' = 1, # Cultist

'T' = 1, # Tablet

'L' = 1, # Elder Thing

'A' = 1) # Autofail

}

games <- replicate(replications, {chaos\_bag\_string\_sim(dicebag = dicebag,

n = pulls,

replace = !no\_replacement)})

if (average) {

cat("Average occurance of pattern:",

mean(stringr::str\_count(games, pattern)), "\n")

} else {

cat("Probability of occurance of pattern:",

mean(grepl(pattern, games)), "\n")

}

quit()

}

if (sys.nframe() == 0) {

cl\_args <- parse\_args(OptionParser(

description = "Simulates the Arkham Horror LCG chaos bag.",

option\_list = list(

make\_option(c("--pattern", "-r"), type = "character",

help = "Pattern (regular expression) to match"),

make\_option(c("--average", "-a"), type = "logical",

action = "store\_true", default = FALSE,

help = "If set, computes average number of occurances"),

make\_option(c("--pulls", "-p"), type = "integer", default = 24,

help = "The number of pulls from the chaos bag"),

make\_option(c("--replications", "-N"), type = "integer",

default = 10000,

help = "Number of replications"),

make\_option(c("--no-replacement", "-n"), type = "logical",

action = "store\_true", default = FALSE,

help = "Draw tokens without replacement"),

make\_option(c("--dicebag", "-d"), type = "character", default = "",

help = "(Optional) dice bag distribution CSV file")

)

))

names(cl\_args)[which(names(cl\_args) == "no-replacement")] <- "no\_replacement"

do.call(main, cl\_args)

}

Below is an example of a CSV file specifying a chaos bag.

token,freq

E,1

P,1

0,2

1,3

2,2

3,1

4,1

S,2

C,1

T,1

L,1

A,1

Below is some example usage.

$ chmod +x AHChaosBagSimulator.R # Only need to do this once

$ ./AHChaosBagSimulator.R -r AA -p 160

Probability of occurance of pattern: 0.4074

$ ./AHChaosBagSimulator.R -r [34TLA]{2} -a -p 160

Average occurance of pattern: 10.6225

To see documentation, type ./AHChaosBagSimulator.R --help. Note that something must always be passed to -r; this is the pattern needed.

**Olive McBride**

This tool was initially written as a way to explore chaos bag probabilities. I didn't consider the tool to be very useful. It simply helped make the point that unlikely events seem to happen frequently in Arkham Horror. However, I found a way to put the tool to practical use.

Recently, Arkham Horror's designers have been releasing cards that seem to demand more math to fully understand. My favorite Arkham Horror card reviewer, [The Man from Leng](https://www.youtube.com/channel/UCddF9gpguoEMONRFkCF0BTA), has fretted about this in some of his recent reviews about [cards using the **seal** mechanic](https://arkhamdb.com/find?q=x%3Aseal&sort=name&view=list&decks=all&spoilers=hide), which change the composition of the chaos bag by removing tokens from it.

I can only imagine how he would feel about a card like Olive McBride (one of the cards appearing in the upcoming Mythos Pack, ["Heart of the Elders"](https://www.fantasyflightgames.com/en/news/2018/4/24/heart-of-the-elders/), to be released later this week.)



Olive McBride allows you to reveal three chaos tokens instead of one, and choose two of those tokens to resolve. This effect can be triggered any time you would reveal a chaos token.

I'll repeat that again: Olive can trigger any time a chaos token is revealed. Most of the time tokens are revealed during skill tests, but other events lead to dipping into the chaos bag too. Notably, *The Dunwich Legacy* leads to drawing tokens without taking skill tests, such as when gambling in "The House Always Wins", due to some encounter cards. This makes Olive McBride a ["master gambler"](https://www.reddit.com/r/arkhamhorrorlcg/comments/8el3ix/new_mythos_pack_announced_heart_of_the_elders/), since she can draw three tokens and pick the winning ones when gambling. (She almost breaks the scenario.) Additionally, Olive can be combined with cards like Ritual Candles and Grotesque Statue to further shift skill tests in your favor.





These are interesting situations but let's ignore combos like this for now. What does Olive do to your usual skill test? Specifically, what does she do to the modifiers?

Before going on, I need to address verbiage. When are about to perform a skill test, typically they will make a statement like "I'm at +2 for this test". This means that the skill value of the investigator (after applying all modifiers) is two higher than the difficulty of the test. Thus, if the investigator draws a -2 or better, the investigator will pass the test; if the investigator daws a -3 or worse, the investigator fails. My play group does not say this; we say "I'm at -2 for this test," meaning that if the investigator sees a -2 or better from the bag, the investigator will pass. This is more intuitive to me, and also I think it's more directly translated to math.

Presumably when doing a skill test with Olive, if all we care about is passing the test, we will pick the two tokens drawn from the bag that have the best modifiers. We add the modifiers of those tokens together to get the final modifier. Whether this improves your odds of passing the test or not isn't immediately clear.

I've written an R function that simulates skill tests with Olive. With this we can estimate Olive's effects.

olive\_sim <- function(translate = c("E" = 2, "P" = 1, "0" = 0, "1" = -1,

"2" = -2, "3" = -3, "4" = -4, "S" = -2,

"C" = -3, "T" = -4, "L" = -5, "A" = -Inf),

N = 1000, dicebag = NULL) {

dargs <- list(n = 3, replace = FALSE)

if (!is.null(dicebag)) {

dargs$dicebag <- dicebag

}

pulls <- replicate(N, {do.call(chaos\_bag\_string\_sim, dargs)})

vals <- sapply(pulls, function(p) {

vecp <- strsplit(p, "")[[1]]

vecnum <- translate[vecp]

max(apply(combn(3, 2), 2, function(i) {sum(vecnum[i])}))

})

vals[which(is.nan(vals))] <- Inf

return(vals)

}

The parameter translate gives a vector that translates code for chaos tokens to numerical modifiers. Notice that the auto-fail token is assigned -Inf since it will cause any test to fail. If we wanted, say, the elder sign token to auto-succeed (which is Mateo's elder sign effect), we could replace its translation with Inf. By default the function uses the dicebag provided with chaos\_bag\_string\_sim(), but this can be changed.

Here's a single final modifier from a pull using Olive.

olive\_sim(N = 1)

C01

-1

Interestingly, the function returned a named vector, and the name corresponds to what was pulled. In this case, a cultist, 0, and -1 token were pulled; the resulting best modifier is -1. The function is already built to do this many times.

olive\_sim(N = 5)

31S 2S1 LS1 120 C2E

-3 -3 -3 -1 0

Below is a function that can simulate a lot of normal skill tests.

test\_sim <- function(translate = c("E" = 2, "P" = 1, "0" = 0, "1" = -1,

"2" = -2, "3" = -3, "4" = -4, "S" = -2,

"C" = -3, "T" = -4, "L" = -5, "A" = -Inf),

N = 1000, dicebag = NULL) {

dargs <- list(n = N, replace = TRUE)

if (!is.null(dicebag)) {

dargs$dicebag <- dicebag

}

pulls <- do.call(chaos\_bag\_string\_sim, dargs)

vecp <- strsplit(pulls, "")[[1]]

vecnum <- translate[vecp]

return(vecnum)

}

Here's a demonstration.

test\_sim(N = 5)

2 S 3 1 0

-2 -2 -3 -1 0

Finally, below is a function that, when given a vector of results like those above, produce a table of estimated probabilities of success at skill tests when given a vector of values the tests must beat in order to pass the test (that is, using the "I'm at -2" type of language).

est\_success\_table\_gen <- function(res, values = (-10):3) {

r = v)})

names(r) <- values

return(rev(r))

}

Let's give it a test run.[4](https://ntguardian.wordpress.com/2018/08/13/evaluating-olive-mcbride-arkham-horror-lcg-chaos-bag-simulator-r/#fn-3304-4)

u <- test\_sim()

est\_success\_table\_gen(u)

3 2 1 0 -1 -2 -3 -4 -5 -6 -7 -8 -9

0.000 0.057 0.127 0.420 0.657 0.759 0.873 0.931 0.931 0.931 0.931 0.931 0.931

-10

0.931

as.matrix(est\_success\_table\_gen(u))

[,1]

3 0.000

2 0.057

1 0.127

0 0.250

-1 0.420

-2 0.657

-3 0.759

-4 0.873

-5 0.931

-6 0.931

-7 0.931

-8 0.931

-9 0.931

-10 0.931

This represents the base probability of success. Let's see what Olive does to this table.

y <- olive\_sim()

as.matrix(est\_success\_table\_gen(y))

[,1]

3 0.024

2 0.063

1 0.140

0 0.238

-1 0.423

-2 0.531

-3 0.709

-4 0.829

-5 0.911

-6 0.960

-7 0.983

-8 0.996

-9 1.000

-10 1.000

Perhaps I can make the relationship more clear with a plot.

plot(stepfun((-10):3, rev(c(0, est\_success\_table\_gen(u)))), verticals = FALSE,

pch = 20, main = "Probability of Success", ylim = c(0, 1))

lines(stepfun((-10):3, rev(c(0, est\_success\_table\_gen(y)))), verticals = FALSE,

pch = 20, col = "blue")

The black line is the estimated probability of success without Olive, and the blue line the same with Olive. (I tried reversing the x-axis, but for some reason I could not get good results.) What we see is:

* Olive improves the chances a "hail Mary" will succeed. If you need +1, +2, or more to succeed, Olive can help make that happen (though the odds still aren't great)
* Olive can help you guarantee a skill test will succeed. If you boost your skill value to very high numbers, Olive can effectively neuter the auto-fail token. That's a good feeling.
* Otherwise, Olive hurts your chances of success. Being at -2 is particularly worse with Olive than without. However, most of the time she changes the probability of success too little to notice.

For most investigators, then, Olive doesn't do much to make her worth your while. But I think Olive makes a huge difference for two investigators: Father Mateo and Jim Culver.



Both of these investigators like some chaos bag tokens a lot. Father Mateo really likes the elder sign since it is an auto-success (in addition to other very good effects), while Jim Culver likes skulls since they are always 0.

What does Olive do for Father Mateo?

translate <- c("E" = 2, "P" = 1, "0" = 0, "1" = -1, "2" = -2, "3" = -3,

"4" = -4, "S" = -2, "C" = -3, "T" = -4, "L" = -5, "A" = -Inf)

# Mateo

mateo\_translate <- translate; mateo\_translate["E"] <- Inf

mateo\_no\_olive <- test\_sim(translate = mateo\_translate)

mateo\_olive <- olive\_sim(translate = mateo\_translate)

plot(stepfun((-10):3,

rev(c(0, est\_success\_table\_gen(mateo\_no\_olive)))),

verticals = FALSE,

pch = 20, main = "Mateo's Probabilities", ylim = c(0, 1))

lines(stepfun((-10):3,

rev(c(0, est\_success\_table\_gen(mateo\_olive)))),

verticals = FALSE, col = "blue", pch = 20)

as.matrix(est\_success\_table\_gen(mateo\_no\_olive))

[,1]

3 0.049

2 0.049

1 0.101

0 0.226

-1 0.393

-2 0.627

-3 0.745

-4 0.862

-5 0.928

-6 0.928

-7 0.928

-8 0.928

-9 0.928

-10 0.928

as.matrix(est\_success\_table\_gen(mateo\_olive))

[,1]

3 0.177

2 0.177

1 0.218

0 0.278

-1 0.444

-2 0.554

-3 0.746

-4 0.842

-5 0.924

-6 0.972

-7 0.989

-8 0.997

-9 1.000

-10 1.000

Next up is Jim Culver.

# Culver

culver\_translate <- translate; culver\_translate["S"] <- 0

culver\_no\_olive <- test\_sim(translate = culver\_translate)

culver\_olive <- olive\_sim(translate = culver\_translate)

plot(stepfun((-10):3,

rev(c(0, est\_success\_table\_gen(culver\_no\_olive)))),

verticals = FALSE,

pch = 20, main = "Culver's Probabilities", ylim = c(0, 1))

lines(stepfun((-10):3,

rev(c(0, est\_success\_table\_gen(culver\_olive)))),

verticals = FALSE, col = "blue", pch = 20)

as.matrix(est\_success\_table\_gen(culver\_no\_olive))

[,1]

3 0.000

2 0.065

1 0.112

0 0.333

-1 0.530

-2 0.638

-3 0.761

-4 0.874

-5 0.936

-6 0.936

-7 0.936

-8 0.936

-9 0.936

-10 0.936

as.matrix(est\_success\_table\_gen(culver\_olive))

[,1]

3 0.020

2 0.095

1 0.217

0 0.362

-1 0.564

-2 0.666

-3 0.803

-4 0.888

-5 0.951

-6 0.972

-7 0.992

-8 0.999

-9 1.000

-10 1.000

Olive helps these investigators succeed at skill tests more easily, especially Mateo. We haven't even taken account of the fact that good things happen when certain tokens appear for these investigators! Sealing tokens could also have a big impact on the distribution of the bag when combined with Olive McBride.

Again, there's a lot that could be touched on that I won't here, so I will share with you a script allowing you to do some of these analyses yourself.

#!/usr/bin/Rscript

#######################################

# AHOliveDistributionEstimator.R

#######################################

# Curtis Miller

# 2018-08-03

# Simulates the chaos bag distribution when applying Olive McBride

#######################################

# optparse: A package for handling command line arguments

if (!suppressPackageStartupMessages(require("optparse"))) {

install.packages("optparse")

require("optparse")

}

olive\_sim <- function(translate = c("E" = 2, "P" = 1, "0" = 0, "1" = -1,

"2" = -2, "3" = -3, "4" = -4, "S" = -2,

"C" = -3, "T" = -4, "L" = -5, "A" = -Inf),

N = 1000, dicebag = NULL) {

dargs <- list(n = 3, replace = FALSE)

if (!is.null(dicebag)) {

dargs$dicebag <- dicebag

}

pulls <- replicate(N, {do.call(chaos\_bag\_string\_sim, dargs)})

vals <- sapply(pulls, function(p) {

vecp <- strsplit(p, "")[[1]]

vecnum <- translate[vecp]

max(apply(combn(3, 2), 2, function(i) {sum(vecnum[i])}))

})

vals[which(is.nan(vals))] <- Inf

return(vals)

}

test\_sim <- function(translate = c("E" = 2, "P" = 1, "0" = 0, "1" = -1,

"2" = -2, "3" = -3, "4" = -4, "S" = -2,

"C" = -3, "T" = -4, "L" = -5, "A" = -Inf),

N = 1000, dicebag = NULL) {

dargs <- list(n = N, replace = TRUE)

if (!is.null(dicebag)) {

dargs$dicebag <- dicebag

}

pulls <- do.call(chaos\_bag\_string\_sim, dargs)

vecp <- strsplit(pulls, "")[[1]]

vecnum <- translate[vecp]

return(vecnum)

}

est\_success\_table\_gen <- function(res, values = (-10):3) {

r <- sapply(values, function(v) {mean(res >= v)})

names(r) <- values

return(rev(r))

}

# Main function

# See definition of cl\_args for functionality (help does nothing)

main <- function(dicebag = "", translate = "", replications = 10000,

plotfile = "", width = 6, height = 4, basic = FALSE,

oliveless = FALSE, lowest = -10, highest = 3,

chaos\_bag\_script = "AHChaosBagSimulator.R",

title = "Probability of Success",

pos = "topright", help = FALSE) {

source(chaos\_bag\_script)

if (dicebag != "") {

dicebag\_df <- read.csv(dicebag, stringsAsFactors = FALSE)

dicebag <- as.numeric(dicebag\_df$freq)

names(dicebag) <- dicebag\_df$token

} else {

dicebag <- NULL

}

if (translate != "") {

translate\_df <- read.csv(translate, stringsAsFactors = FALSE)

translate <- as.numeric(translate\_df$mod)

names(translate) <- translate\_df$token

} else {

translate <- c("E" = 2, "P" = 1, "0" = 0, "1" = -1,

"2" = -2, "3" = -3, "4" = -4, "S" = -2,

"C" = -3, "T" = -4, "L" = -5, "A" = -Inf)

}

if (any(names(translate) != names(dicebag))) {

stop("Name mismatch between translate and dicebag; check the token columns")

}

if (!oliveless) {

olive\_res <- olive\_sim(translate = translate, dicebag = dicebag,

N = replications)

cat("Table of success rate with Olive when at lest X is needed:\n")

print(as.matrix(est\_success\_table\_gen(olive\_res, values = lowest:highest)))

cat("\n\n")

}

if (basic) {

basic\_res <- test\_sim(translate = translate, dicebag = dicebag,

N = replications)

cat("Table of success rate for basic test when at lest X is needed:\n")

print(as.matrix(est\_success\_table\_gen(basic\_res, values = lowest:highest)))

cat("\n\n")

}

if (plotfile != "") {

png(plotfile, width = width, height = height, units = "in", res = 300)

if (basic) {

plot(stepfun(lowest:highest, rev(c(0, est\_success\_table\_gen(basic\_res)))),

verticals = FALSE, pch = 20, main = title,

ylim = c(0, 1))

if (!oliveless) {

lines(stepfun(lowest:highest,

rev(c(0, est\_success\_table\_gen(olive\_res)))),

verticals = FALSE, pch = 20, col = "blue")

legend(pos, legend = c("No Olive", "Olive"), col = c("black", "blue"),

lty = 1, pch = 20)

}

}

dev.off()

}

quit()

}

if (sys.nframe() == 0) {

cl\_args <- parse\_args(OptionParser(

description = "Estimate the chaos bag distribution with Olive McBride.",

option\_list = list(

make\_option(c("--dicebag", "-d"), type = "character", default = "",

help = "(Optional) dice bag distribution CSV file"),

make\_option(c("--translate", "-t"), type = "character", default = "",

help = "(Optional) symbol numeric translation CSV file"),

make\_option(c("--replications", "-r"), type = "integer",

default = 10000,

help = "Number of replications to perform"),

make\_option(c("--plotfile", "-p"), type = "character", default = "",

help = paste("Where to save plot (if not set, no plot",

"made; -b/--basic must be set)")),

make\_option(c("--width", "-w"), type = "integer", default = 6,

help = "Width of plot (inches)"),

make\_option(c("--height", "-H"), type = "integer", default = 4,

help = "Height of plot (inches)"),

make\_option(c("--basic", "-b"), type = "logical",

action = "store\_true", default = FALSE,

help = "Include results for test without Olive McBride"),

make\_option(c("--oliveless", "-o"), type = "logical",

action = "store\_true", default = FALSE,

help = "Don't include results using Olive McBride"),

make\_option(c("--lowest", "-l"), type = "integer", default = -10,

help = "Lowest value to check"),

make\_option(c("--highest", "-i"), type = "integer", default = 3,

help = "Highest value to check"),

make\_option(c("--pos", "-s"), type = "character",

default = "topright", help = "Position of legend"),

make\_option(c("--title", "-m"), type = "character",

default = "Probability of Success",

help = "Title of plot"),

make\_option(c("--chaos-bag-script", "-c"), type = "character",

default = "AHChaosBagSimulator.R",

help = paste("Location of the R file containing",

"chaos\_bag\_string\_sim() definition; by",

"default, assumed to be in the working",

"directory in AHChaosBagSimulator.R"))

)

))

names(cl\_args)[which(

names(cl\_args) == "chaos-bag-script")] = "chaos\_bag\_script"

do.call(main, cl\_args)

}

You've already seen an example CSV file for defining the dice bag; here's a file for defining what each token is worth.

token,mod

E,2

P,1

0,0

1,-1

2,-2

3,-3

4,-4

S,-2

C,-3

T,-4

L,-5

A,-Inf

Make the script executable and get help like so:

$ chmod +x AHOliveDistributionEstimator.R

$ ./AHOliveDistributionEstimator.R -h

**Conclusion**

If you've never heard of this game I love, hopefully you've heard of it now. Give the game a look! And if you have heard of this game before, I hope you learned something from this post. If you're a reviewer, perhaps I've given you some tools you could use to help evaluate some of Arkham Horror's more intractable cards.

My final thoughts on Olive: she's not going to displace [Arcane Initiate's](https://arkhamdb.com/card/01063) spot as the best Mystic ally, but she could do well in certain decks. Specifically, I think that Mateo decks and Jim Culver decks planning on running [Song of the Dead](https://arkhamdb.com/card/02112) will want to run her since there are specific chaos tokens those decks want to see; the benefits of Olive extend beyond changing the probability of success. Most of the time you will not want to make a hail Mary skill test and you won't have the cards to push your skill value to a point where anything but the auto-fail is a success, so most of the time Olive will hurt your chances rather than help you, if she has any effect at all. Thus a typical Mystic likely won't find Olive interesting… but some decks will love her.

By the way, if you are in the Salt Lake City area of Utah, I play Arkham Horror LCG at [Mind Games, LLC](http://mind-games.com/). While I haven't been to many other stores, Mind Games seems to have the best stocking of Arkham Horror (as well as two other Fantasy Flight LCGs, *A Game of Thrones* and *Legend of the Five Rings*). Every Tuesday night (the store's late night) a group of card game players come in to play; consider joining us! In addition, Mind Games likely has the best deal when it comes to buying the game; whenever you spend $50 on product, you get an additional $10 off (or, alternatively, you take $10 off for every $60 you spend). Thus Mind Games could be the cheapest way to get the game (without going second-hand or Internet deal hunting).

*(Big image credit:* [*Aurore Folney and FFG*](https://www.artstation.com/artwork/XKJQD)*.)*

**EDIT: WordPress.com does not like R code and garbled some of the code in the script for Olive simulation. It should be correct now. If anyone spots other errors, please notify me; I will fix them.**

I have created a video course published by [Packt Publishing](http://packtpub.com/) entitled [*Applications of Statistical Learning with Python*](https://ntguardian.wordpress.com/video-courses/applications-of-statistical-learning-with-python/), the fourth volume in a four-volume set of video courses entitled, *Taming Data with Python; Excelling as a Data Analyst*. This course discusses how to use Python for data science, emphasizing application. It starts with introducing natural language processing and computer vision. It ends with two case studies; in one, I train a classifier to detect spam e-mail, while in the other, I train a computer vision system to detect emotions in faces. Viewers get a hands-on experience using Python for machine learning. If you are starting out using Python for data analysis or know someone who is, please consider [buying my course](https://www.packtpub.com/big-data-and-business-intelligence/applications-statistical-learning-python-video) or at least spreading the word about it. You can buy the course directly or purchase a [subscription to Mapt](https://www.packtpub.com/mapt/) and watch it there.

If you like my blog and would like to support it, spread the word (if not get a copy yourself)! Also, stay tuned for future courses I publish with Packt at the Video Courses section of my site.

1. You can also play scenarios stand-alone, which isn’t nearly as fun as playing in a campaign, especially one of the eight-scenario campaigns. [](https://ntguardian.wordpress.com/2018/08/13/evaluating-olive-mcbride-arkham-horror-lcg-chaos-bag-simulator-r/#fnref-3304-1)
2. This is not right because sometims you redraw tokens from the bag without replacement; we'll ignore that case for now. [](https://ntguardian.wordpress.com/2018/08/13/evaluating-olive-mcbride-arkham-horror-lcg-chaos-bag-simulator-r/#fnref-3304-2)
3. Using the asymptotically appropriate confidence interval based on the Normal distribution (as described [here](https://ntguardian.wordpress.com/2018/08/13/evaluating-olive-mcbride-arkham-horror-lcg-chaos-bag-simulator-r/)), the margin of error error will not exceed \frac{1}{\sqrt{n}}since \sqrt{\hat{p}(1 - \hat{p})} \leq \frac{1}{2}. Thus, we have m \leq \frac{1}{\sqrt{n}}; solving this for nyields n \geq m^{-2}. So if we want a 95% chance that the margin of error will not exceed m = 0.01 = \frac{1}{100}, we need a dataset of size n \geq (0.01)^{-2} = 10,000. [](https://ntguardian.wordpress.com/2018/08/13/evaluating-olive-mcbride-arkham-horror-lcg-chaos-bag-simulator-r/#fnref-3304-3)
4. Note that the earlier calculations that argued the odds of being accurate up to two decimal places were high no longer hold, since many more numbers are being estimated, and one estimate is not independent of the other. That said, the general principle of more simulations leading to better accuracy still holds. [](https://ntguardian.wordpress.com/2018/08/13/evaluating-olive-mcbride-arkham-horror-lcg-chaos-bag-simulator-r/#fnref-3304-4)