Pokémon Catch Rate Calculator

Mark Litak

5/15/2022

INTRODUCTION

What

In this project, a catch chance calculator for Pokémon Go Generation I Pokémon was created. In addition to calculating the probability for a catch on any given throw, the number of throws before the player has an 80% chance of catching the Pokémon is given, as well as the number of throws before the Pokémon has a 50% chance of fleeing.

Why

This topic was chosen because I am a Pokémon Go player, and when we were talking about geometric distributions, this automatically came to mind. I had seen Pokémon Catch Chance calculators before, and I wanted to try my hand at creating one. However, I have never seen a catch chance calculator that also determined the number of throws before a likely failure or success.

How

Formula

On the Pokémon wiki, a formula for the Probability of Catching a Pokémon is provided, as well as several data tables for use with the formula,

$$P = 1 - (1 - \frac{BCR}{2 \times CPM})^{Ball \times Berry \times Throw \times Curveball \times Medal \times Encounter}$$

where BCR is the **Base Catch Rate**, provided in a table, **CPM** is the CP multiplier, **Ball** is the type of ball used to attempt to catch the Pokémon, **Throw** corresponds to a specific throw's accuracy, **Curveball** is whether a player threw a curveball or not, **Medal** is whether the player has a medal in that specific Pokémon's type, and **Encounter**, corresponding to whether the player encountered the Pokémon in the Wild or through some other means, such as a Raid.

Within this project, several assumptions were made.

- The Pokémon was assumed to have been encountered in the Wild, as the Wild contains Pokémon with the highest possible levels; this corresponds to an exponent multiplier of 1.
- The Pokémon was assumed to be the highest level possible with such an encounter (level 35, corresponding to a CP multiplier of 0.7615638).

- The player was assumed to successfully throw a curveball, which is not particularly difficult with practice; this corresponds to a multiplier of 1.7.
- The player is assumed to not have a medal in the type, corresponding to a multiplier of 1.

With the above assumptions, the formula used became

$$P = 1 - (1 - \frac{BCR}{2 \times 1.52312768})^{1.7 \times Ball \times Berry \times Throw}$$

Ball, Berry, and Throw Modifiers

In the exponent are several variables which change the roll modifier.

There are three types of balls, each corresponding to different modifiers within the formula. **Pokéballs** are normal balls, and their modifier is 1; it does not change the probability. **Great balls** are rarer, and their corresponding modifier is 1.5. **Ultra balls** are the rarest and most difficult to acquire, and their corresponding modifier is 2.

Additionally, there are several types of berries, three of which change the modifier. **Razz Berries** give a modifier of 1.5. **Silver Pinap Berries**, or Silver berries for short, give a modifier of 1.8. **Golden Razz Berries** give a modifier of 1.5.

Finally, there are four types of throws. **Normal** throws do not change the probability, as their modifier is 1. The modifier for a **nice** throw ranges from 1 to 1.3, depending on how accurately a ball is thrown; the average of 1.15 was taken in this calculator. The modifier for a **great** throw ranges from 1.3 to 1.7, and thus 1.5 was the modifier chosen in this calculator. **Excellent** throws give between 1.7 and 2 as a modifier, and therefore 1.85 was chosen as their modifier.

Number of Throws Before Success or Failure

For the purposes of this program, **success** was defined as a catch, and **failure** was defined as the Pokémon running away.

To determine the number of throws before a likely success (or a likely failure), the R function `qgeom` was used. A "likely success" was determined to be an 80% chance of having succeeded. A "likely failure," on the other hand, was determined to be a 50% chance of failure.

Body

Introduction to Pokémon Go

Pokémon Go is an augmented reality game, played on mobile phones, which allows a player to experience the Pokémon games in their surroundings. Pokémon spawn across the world, and the player attempts to catch the Pokémon with Pokéballs, which they may obtain at PokeStops and Gyms, tied to real-life landmarks. Players also obtain Berries, objects that they may "feed" a Pokémon. Some berries make catching a Pokémon easier.

When playing Pokémon Go, a player needs to manage their Pokéballs and their Berries, and may sometimes be in danger of running out of them. Some Pokéballs are rarer and more difficult to acquire, though they provide a higher chance of catching a Pokémon. Additionally, catching a Pokémon becomes more difficult the higher its level is. A Pokémon displays its CP above its head when a player encounters it; this CP corresponds to that particular Pokémon's level.

A rough estimate of catch chance is given by a colored circle when catching a Pokémon, but this is a rough estimate. A calculator for a precise catch chance, as well as the number of throws before an 80% chance of success, was created in order to facilitate Pokéball and Berry management. Additionally, as Pokémon may also flee before they are caught, a figure for the number of throws before the Pokémon flees is also provided, to give the player a good idea of how many throws they can truly risk before failure.

Preliminary Challenges: Base Functionality

A number of challenges arose in creating this Catch Chance Calculator. Formatting the data for Base Catch Rates proved difficult, as did determining which variables to include in the program, and which to keep constant.

The first challenge in creating this calculator was acquiring the data for Base Catch Rates, and formatting it properly for R to interpret. I ended up needing to manually enter this data into Excel from a Pokémon Wiki page, as attempting to copy paste from a table proved strangely difficult; images of the Pokémon were unable to be removed when copy-pasting. After manual entry, I converted the Base Catch Rate data into a CSV and opened it in R.

Upon considering the Catch Rate formula, I realized that several variables were present. Some of these variables I discarded outright for simplicity, such as Encounter type and whether or not the player had a Medal; I assumed that the most difficult option would be the case. As learning how to throw curveballs may take only a few attempts, as this is simple enough, I felt that I could safely assume that the player would be throwing a curveball.

However, this still left me with five variables. I knew that the most popular calculator made a table, of the type of ball to be used and the accuracy of a throw, with checkboxes for berries, and a slider for the Pokémon's level. I knew that I wanted to keep a rudimentary User Interface, with an option for the chosen Pokémon. I considered whether I wanted to keep the CPM a variable, or if I wanted to keep it at a constant level, and whether I wanted to keep the Berry a constant variable as well, or if I wanted to assume that the player was using no berry (or a Razz berry, which provides a modifier of 1.5, the lowest of the available berries).

After a short period of consideration, I decided to keep the Berry choice, as this would add only a comparatively low amount of complexity. I debated whether I should leave this as an option for the user to choose initially, along with the Pokémon's number, or if I wanted to create a tensor, or 3D table, with the data immediately visible slice by slice. Ultimately, I decided that choosing a berry to use initially would be the simplest approach to take; however, taking a more complex approach may be possible with more work to be done.

The question of CPM was both easier and more difficult. Initially, I very much wanted to have an option for the user that would allow them to input the CP of the Pokémon, have the program calculate the level of the Pokémon and therefore its CPM, and then determine the catch rate based on this. However, upon further research, I discovered that the CP of a Pokémon and its level vary tremendously, and listings of this data were poorly formatted for copy-pasting; therefore, having some sort of table or data frame with that information would require a large amount of manual input. As a result, I opted to choose the most difficult possible level for a Wild encounter, Level 35.

Programming in R

To create the widget with displays for catch rate, an R script becomes necessary, and therefore learning the language. A few requirements were needed in order to make such a script work:

- Create a Catch/Flee Rate CSV file
- Reading catch rate and flee rate data, as well as the modifier, from the CSV files
- Get input from the user to select a Pokémon and select the berry to be used

- Create functions to calculate the catch rate, the number of throws before a likely catch, and the number of throws before the Pokémon likely flees
- Create tables to display the data
- Display the data

As an example Pokémon for the purposes of this file, I will be using Bulbasaur, or Pokémon number 1, and using no berries.

Create Necessary File

As a preliminary step, catch/flee rate and modifier files were necessary. Obtaining the catch rate and flee rate data simply required visiting the Pokémon wiki Bulbapedia and manually typing out the information given. (Copy-pasting the information proved to be strangely difficult, as for whatever reason, the images could not be deleted.)

Read Data

The first thing that the script itself needs to do is to read the Catch Rate data from a CSV file.

Reading a file containing data and creating a vector for it simply requires one line:

```
var <- read.csv('filename.csv', stringsAsFactors = T/F, head = T/F, row.names = 1)
```

Therefore, the lines reading data from the two files is as follows:

```
Pokémon <- read.csv('BaseCatchRate.csv', stringsAsFactors = T,head = T, row.names = 1)
```

Modifier data for berries was manually created as a list, as this information does not change and is relatively small.

```
berr <- c(1, 1.5, 1.8, 2.5)
```

Get Input From User

In order to make the widget work, the user must input the Pokémon whose catch rate they wish to evaluate, and the berry they are attempting to use.

Getting input from a user was something completely new to me. I had to look up how to do this myself. However, it turned out to not be so different from Python. The function `readline()` gives an input for the user, and `readline(prompt="")` gives a display for the user to be prompted.

```
v.p <- readline(prompt="Enter Pokémon number: ")
v.berry <- readline(prompt="Berry (None = 1, Razz = 2, Silver = 3, Golden = 4): ")
```

After the user inputs the Pokémon they wish to see, as well as the berry they wish to use, the program assigns variables as it needs. Initially, as all input is read as a string, the input numbers need to be converted to actual integers using `as.integer`:

```
v.p <- as.integer(v.p)
v.berry <- as.integer(v.berry)
```

Next, data for the specific Pokémon and berry need to be read.

```
berry <- berr[v.berry] # finds the Berry modifier from the inputted berry
BCR <- Pokémon[v.p,1] # finds the Base Catch Rate
flee_chance <- Pokémon[v.p,2] # finds the Flee Rate
```

For Bulbasaur, `BCR` is `.2`, `flee_chance` is `.1`, and not using a berry will give a modifier of 1:

```
berry <- 1
BCR <- .2
flee_chance <- .1
```

After this data is read, we are ready to move on to **creating a function**.

Creating Functions

Three functions needed to be created: the probability of catching a Pokémon on a given throw ($P(Catch)$), the number of throws before the Pokémon will have an 80% chance of being caught ($Throws$), and the number of throws before the Pokémon will have a 50% chance of running away ($FleeRate$).

The first, and most crucial, is the probability of catching a Pokémon. Recall that the formula for the Probability of Catching is

$$P = 1 - (1 - \frac{BCR}{2 \times 1.52312768})^{1.7 \times Ball \times Berry \times Throw}$$

Creating a function in R is similar to creating a function in Python. A function has the following information:

```
variable <- function(inputs) {
  whatevercalculationsandstuff
  return(something)
}
```

The calculation for $P(Catch)$ can be written in R as `p <- 1-(1-(BCR/1.52312768))^(1.7*ball*berry*throw)`; therefore, its function will be

```
probCatch <- function(BCR,ball,berry,throw) {
  p <- 1-(1-(BCR/1.52312768))^(1.7*ball*berry*throw)
  return(p)
}
```

Determining the number of throws before a Pokémon will be 80% likely to have been caught requires the utilization of the **geometric distribution**. The geometric distribution is a distribution that represents exactly this: the number of failures before a likely success, to a given percentile. In R, this can be calculated using the function `qgeom(percentile,probability)`, where `percentile` is the percent certainty to which we want to calculate, and `probability` is the probability of success on a given trial.

Therefore, the formula for $Throws$,

```
throwsCatch <- function(pcatch) {
  return(qgeom(.8, pcatch))
}
```

and for $FleeRate$,

```
throwsFlee <- function(p) {
  return(qgeom(.5, p))
}
```

Create Tables to Display Data

The next step to take is to create the necessary catch tables, which we can later use to display.

Using the R command `cbind`, a table can be created from a list of columns. `cbind` can bind together vectors, matrices, and dataframes. This was perfect for my uses, as I could create columns of data for each type of Pokéball used, with each row indicating the type of throw. I would then combine these vectors into a table.

As stated above, a Pokéball gives a modifier of 1, a Greatball gives a modifier of 1.5, and an Ultraball gives a modifier of 2. A normal throw gives no modifier, a Nice throw gives a modifier between 1 and 1.3 (averaged to 1.15 for simplicity of the program), a Great throw gives a modifier between 1.3 and 1.7 (averaged to 1.5 for simplicity), and an Excellent throw gives a modifier between 1.7 and 2.0 (averaged to 1.85 for simplicity).

Therefore, each vector had to vary along the Throw type, while keeping the Ball type the same. We create these vectors, filling each row with a calculated $P(Catch)$ using the `probCatch` function we created earlier.

```
# Pball's Ball modifier is 1, and the column varies by throw
Pball <- c(probCatch(BCR,1,berry,1),
           probCatch(BCR,1,berry,1.15),
           probCatch(BCR,1,berry,1.5),
           probCatch(BCR,1,berry,1.85))

# Gball's Ball modifier is 1.5
Gball <- c(probCatch(BCR,1.5,berry,1),
           probCatch(BCR,1.5,berry,1.15),
           probCatch(BCR,1.5,berry,1.5),
           probCatch(BCR,1.5,berry,1.85))

# Uball's Ball modifier is 2
Uball <- c(probCatch(BCR,2,berry,1),
           probCatch(BCR,2,berry,1.15),
           probCatch(BCR,2,berry,1.5),
           probCatch(BCR,2,berry,1.85))
```

Similarly, the function evaluating $Throws$ varied according to Throw type and ball, using the `throwsCatch` function we had created earlier, using $P(Catch)$ as the input:

```
# Pball's Ball modifier is 1, and the column varies by throw
throwsP <- c(throwsCatch(probCatch(BCR,1,berry,1)),
             throwsCatch(probCatch(BCR,1,berry,1.15)),
             throwsCatch(probCatch(BCR,1,berry,1.5)),
             throwsCatch(probCatch(BCR,1,berry,1.85)))

# Gball's Ball modifier is 1.5
throwsG <- c(throwsCatch(probCatch(BCR,1.5,berry,1)),
             throwsCatch(probCatch(BCR,1.5,berry,1.15)),
             throwsCatch(probCatch(BCR,1.5,berry,1.5)),
             throwsCatch(probCatch(BCR,1.5,berry,1.85)))
```

```
# Uball's Ball modifier is 2
throwsU <- c(throwsCatch(probCatch(BCR,2,berry,1)),
             throwsCatch(probCatch(BCR,2,berry,1.15)),
             throwsCatch(probCatch(BCR,2,berry,1.5)),
             throwsCatch(probCatch(BCR,2,berry,1.85)))
```

Once these vectors are created, we can create their associated matrices. First, to create a table we need to use `cbind` to bind columns together:

```
catchTab <- cbind(Pball,Gball,Uball) # column binds pcatch
throwsTab <- cbind(throwsP, throwsG, throwsU) # column binds throwscatch
```

Next, we need to name each column and row for each table using `colnames` and `rownames`:

```
colnames(catchTab) <- c('POKEBALL', 'GREATBALL', 'ULTRABALL')
rownames(catchTab) <- c('NORMAL', 'NICE', 'GREAT', 'EXCELLENT')
colnames(throwsTab) <- c('POKEBALL', 'GREATBALL1', 'ULTRABALL')
rownames(throwsTab) <- c('NORMAL', 'NICE', 'GREAT', 'EXCELLENT')
```

Finally, we need to convert each matrix to a table:

```
catchTab <- as.table(catchTab)
throwsTab <- as.table(throwsTab)
```

And, last of all, we need to determine the number of throws before a Pokémon has at least a 50% chance of fleeing using the function `throwsFlee`:

```
flee_num <- throwsFlee(flee_chance)
```

Display Data

Last of all, we need to display the tables that we had created. Displaying data was also something that I needed to learn; luckily, it was exactly the same as the Python `print` function:

```
print("String to display")
print(object_to_display)
```

Thus, I was able to use the output function to display the Pokémon's name, table names, and dividers:

```
print(rownames(Pokémon)[v.p])
# Prints the name of the Pokémon from the CSV file
```

```
print("Bulbasaur")
```

```
## [1] "Bulbasaur"
```

```
print("CATCH PROBABILITY ON A GIVEN THROW")
```

```
## [1] "CATCH PROBABILITY ON A GIVEN THROW"
```

```r
print(catchTab)
```

```
##            POKEBALL GREATBALL ULTRABALL
## NORMAL    0.2128251 0.3015957 0.3803556
## NICE      0.2405802 0.3382059 0.4232815
## GREAT     0.3015957 0.4163395 0.5122315
## EXCELLENT 0.3577090 0.4852484 0.5874623
```

```r
print('--------------------------------------------------------------------')
```

```
## [1] "--------------------------------------------------------------------"
```

```r
print("NUMBER OF THROWS BEFORE 80% CHANCE OF CATCHING")
```

```
## [1] "NUMBER OF THROWS BEFORE 80% CHANCE OF CATCHING"
```

```r
print(throwsTab)
```

```
##           POKEBALL GREATBALL1 ULTRABALL
## NORMAL           6          4         3
## NICE             5          3         2
## GREAT            4          2         2
## EXCELLENT        3          2         1
```

```r
print('--------------------------------------------------------------------')
```

```
## [1] "--------------------------------------------------------------------"
```

```r
print("NUMBER OF THROWS BEFORE 50% CHANCE OF FLEEING")
```

```
## [1] "NUMBER OF THROWS BEFORE 50% CHANCE OF FLEEING"
```

```r
print(flee_num)
```

```
## [1] 6
```

Potential Future Improvements

As mentioned above, one point of improvement that I would like to make in the future is rather than asking the player to input their berry *before* a calculation is made, the data would be provided to give the player a chance to compare how many throws they can risk with the number of throws until a likely failure. This would be done by making several tables, which can be made with a `for` loop. The display of these tables in a legible and easily-comparable manner may prove to be another challenge, as four Throwing tables would need to be displayed, in addition to a Probability table and a Failure figure. A longer-term goal would be to grant the player the ability to manually input a Pokémon's CP, and allow the program to vary accordingly.

However, a catch chance calculator that allows its user to see the number of throws before a likely success, and compare it to the number of throws before a likely failure, gives an opportunity for the player to see how easily a Pokémon can run away, and how many attempts they are likely to get before a success or failure.

Topics From Class

RMarkdown

Creating files in RMarkdown was not significantly difficult to me. Through my previous degree in math, I have experience in writing Markdown files with LaTeX, as many homework assignments were required in such a format. Additionally, we had been using RMarkdown all semester, and so I really felt that I had gotten the hang of using it.

Github

On the other hand, Github is not something that I am very familiar with. I learned a lot in trying to create a Repository for this project, which I had only done once previously. I learned how to create a README file, and how to commit changes to Github. I also learned how to display LaTeX equations in Github and .md files, to display the necessary equations for understanding my project in the Github README.md file. (Also, to be completely honest, I initially simply uploaded files to the Github website instead of pushing to Git itself.)

Geometric Distribution

To determine both the number of throws before a likely success (probability of 80% or more) and the number of throws before Pokémon will likely run away (probability of 50% or more), a geometric distribution was necessary. A geometric distribution is a distribution corresponding to the probability of failure given some number of trials. This was discussed in class as a helpful tool in probability and statistics. To involve the geometric distribution in the calculations, the function `qgeom(X,Y)` was used. It corresponds to a sum of the values of the geometric distribution until reaching the desired percentile. It determines the number of failures before a probability of success `X` is found, given a probability of success per trial `Y`. This function was also discussed and used in class as a helpful tool in probability and statistics.

R: User Input

User input became necessary to involve in the creation of the widget. User input determines which Pokémon's catch rate to pull up, and which berry the user feeds the Pokémon to make the catch easier. I learned how to involve user input using `readline(prompt="Prompt"`, and user output using `print()`, which was very similar in syntax and usage to Python.

R: Creating Functions in R

In order to make the R script function, it was necessary to create functions. I created three: a function to replicate the P formula listed above, a function to calculate the number of throws necessary to likely catch a Pokémon (with a probability of .8), and a function to calculate how many throws before a Pokémon will likely run away (with a probabilty of .5). These functions were created in R, and the entire course was exploring concepts in probability and statistics using R.

Conclusion

This project was very interesting, and through it, I became more comfortable using the geometric distribution and its related function in R, `qgeom`. Additionally, I learned a lot about formatting in R, creating functions, and creating and formatting tables for the user's perusal. I also learned more about scripting in R, which

I had not been doing throughout class, preferring instead to use RMarkdown. This allowed me to make a rudimentary command line-based User Interface. In addition to learning about R topics, I learned a lot about the subject matter which I was analyzing. Through running several trials, I learned that, in most cases, even with the weakest ball and no berries used, the number of throws before a likely catch will still be higher than the number of throws before a Pokémon runs away. I also, of course, learned how the catch probability is calculated.

Had I had more time, instead of creating only a single table and requiring a user to input the type of berry used, if any, I would have instead displayed four tables with catch chances and the number of throws necessary to have a likely success. If the data had been easier to format, or I had a *lot* more time, I would have included a calculator for a Pokémon's CP and its corresponding level, and included that in the catch chance calculation.

Sources

Information about the probability formula, catch rates, and CP modifiers was taken from the Pokémon Wiki, Bulbapedia: https://bulbapedia.bulbagarden.net/wiki/Catch_rate_(GO)

I refreshed my memory on `qgeom`, c(), cbind(), and making tables was taken from Statology. https://www.statology.org/dgeom-pgeom-qgeom-rgeom-r/ https://www.statology.org/c-function-in-r/ https://www.statology.org/create-table-in-r/

I refreshed my memory on making functions using https://swcarpentry.github.io/r-novice-inflammation/02-func-R/

I learned how to take user input from https://www.datamentor.io/r-programming/examples/user-input/