March 11, 2021

Do or Die

A Dice Game Simulation

Westley Brenner, Noelle Richards

ISYE 6644 – Simulation

Spring 2021 – Project 1

ABSTRACT:

We used R to do a simulation of a dice game based on the roll of a six-sided die and found that the game results most closely resembled a geometric distribution. We found that the expected number of cycles that the game will run is approximately 17.69.

BACKGROUND:

We simulated the following situation:

Suppose there are two people playing a dice game. They each begin with 4 coins and a pot of 2 coins. Each player also has one six-sided die. For each round of the game both players will roll their dice, which will determine the next action (shown below).

Roll Value	**Action Taken**
1	Nothing – Player bank and pot stay the same.
2	Player takes all coins in the pot. The pot now has 0 coins
3	Player takes half the coins in the pot (rounded down). Half remains in pot (rounded up).
4-6	Player puts one coin in the pot. Player bank decreases by one and pot increases by one coin.

The game ends when a player cannot complete an action. We were interested in the expected number of rounds before the game finishes and the distribution of this expected number.

In order to simulate, we used R and created two functions: pot_change (Appendix A) and run_sim (Appendix B). The first function simulates a round of the game, by generating a random number between 1 and 6 for each of Player A and Player B's dice rolls. It then applies the appropriate action to the player banks and the pot and outputs the new values. The second function run_sim, runs pot_change over and over until the game is finished, then outputs the number of rounds in the game. In order to find the distribution of the expected number of rounds per game, we would need to use run_sim repeatedly.

MAIN FINDINGS:

Given the parameters above, there is one specific scenario that would trigger the end of a game:

- The player balance is 0 before the dice roll and the player rolls a 4, 5, or 6. This would leave the player with a negative balance, and thus end the game because it is not possible.

The structure of the simulation is done in R. First step is to create the functions for the simulation, then run the simulation:

- Pot Change Function (Appendix A)
  - Create dice rolls for players A and B with the sample() function.
  - Create an empty data frame to store the end of cycle balances called run_history.
  - Change the pot and player balances based on the random dice rolls according to the rules for each dice outcome.

    - Return the player and pot balances for the next cycle.
- Run Simulation Function (Appendix B)
    - Initialize a cycle counter.
    - Run the Pot Change Function while end-of-cycle player balances are greater than or equal to 0.
    - Return the cycle counter when an end-of-game condition is met.

Now that the functions have been created, the game can be initialized with starting balances and started (Appendix C). We simulated this game 5,000 times and stored the cycle count as a list, and we then performed statistical tests on the final list. A histogram was created, various distributions were fit to the sample, and we compare the standard errors to pick the best distribution.

CONCLUSIONS:

We found that the distribution with the best fit was a geometric distribution as it had the smallest standard error. When graphed side by side, it seems that our conclusion was correct (Figure 1).

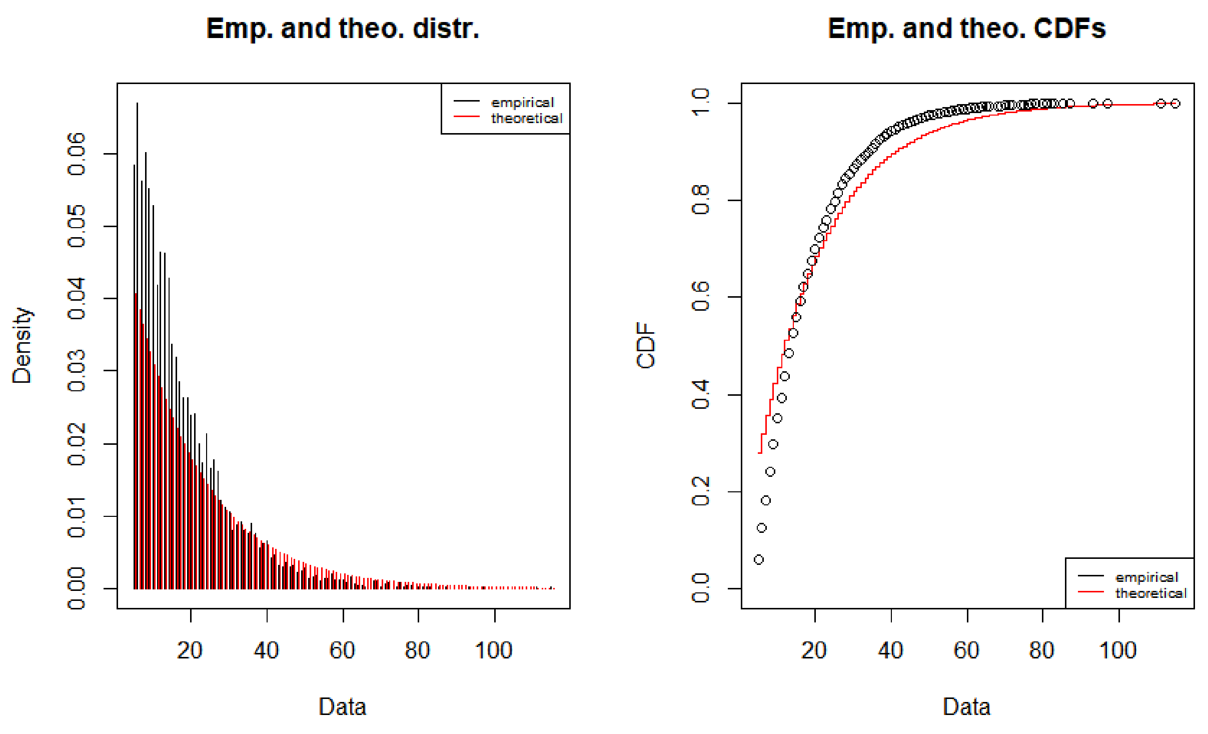


Figure 1

The expected value of this simulation is E[x] ≊ 17.69 and the best fit distribution is a geometric distribution. Because of the nature of the game and the setup, it is not possible for a game to end prior to the 5th cycle. Thus, the expected value of this simulation is E[x] ≊ 12.69 **from the first possible point of failure**.

APPENDIX A:

Function definition: pot_change

```
pot_change <- function(A,B,pot){
  a_die <- sample(1:6, 1, replace=TRUE)
  b_die <-sample(1:6, 1, replace=TRUE)
  run_history <- setNames(data.frame(matrix(ncol = 3, nrow = 0)), c("Player A", "Player B", "Pot"))

##Action for A
  if (a_die == 2) {
    A <- A+pot
    pot<- 0
  }
  if(a_die==3){
    A <- A+floor(pot/2)
    pot <- ceiling(pot/2)
  }
  if(a_die==4 | a_die==5 | a_die==6){
    A <- A-1
    pot <- pot+1
  }

##Action for B
  if (b_die == 2) {
    B <- B+pot
    pot<- 0
  }
  if(b_die==3){
    B <- B+floor(pot/2)
    pot <- ceiling(pot/2)
  }
  if(b_die==4 | b_die==5 | b_die==6){
    B <- B-1
    pot <- pot+1
  }
run_final<-c(A,B,pot)
run_history<-rbind(run_history,run_final)
return(run_history)
}
```

APPENDIX B:

Function definition: run_sim

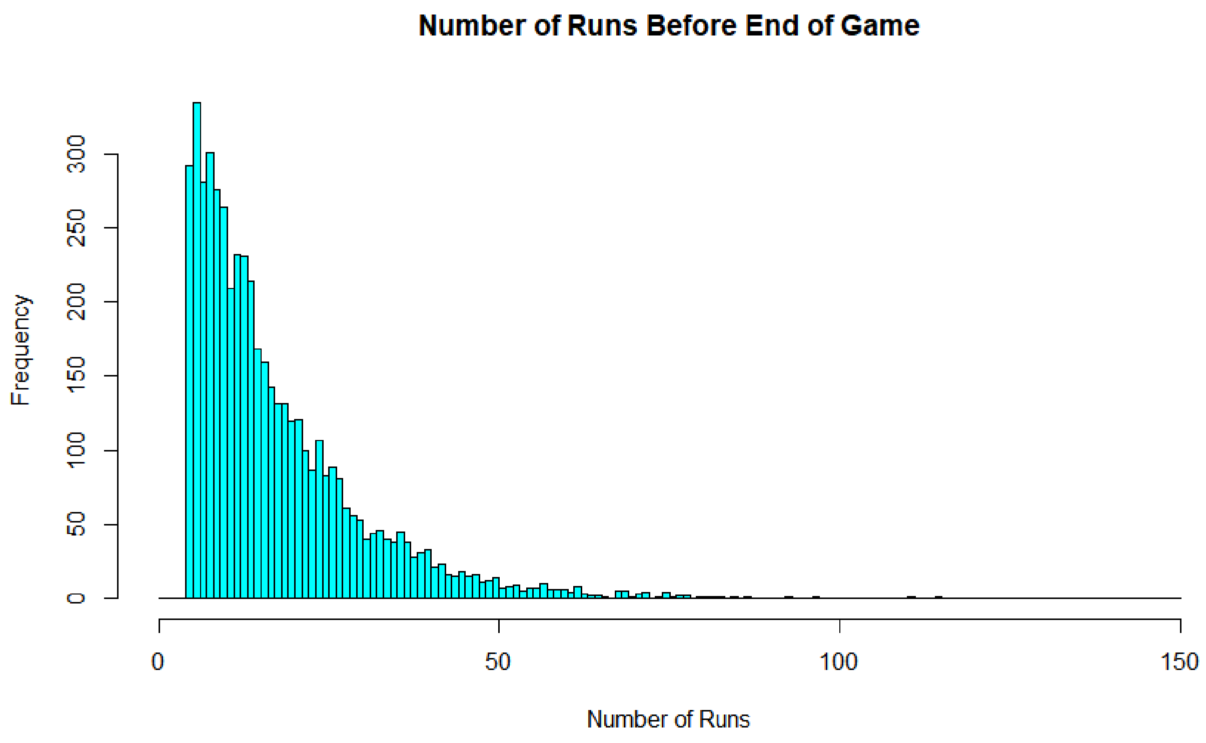
```
run_sim <-function(x){
  run_count <-0

  while(x[1]!=0 && x[2]!=0){
    y<-pot_change(x[1],x[2],x[3])
    x<-y
    run_count<- run_count+1
  }

  return(run_count)
}
```

APPENDIX C:
Simulation of Dice Game in R

```
rm(list = ls())
#install.packages("fitdistrplus")
library(ggplot2)
library(fitdistrplus)

#####
##### Initialize and Run

x<-c(4,4,2)
runs <- c()
set.seed(42)

for (i in 1:5000){
  current_run <- run_sim(x)
  runs<-append(runs,current_run)
}

hist(runs, breaks = 0:150, main = "Number of Runs Before End of Game", xlab = "Number of Runs", col =
"cyan")
summary(runs)

end_time <- Sys.time()
end_time - start_time

runs.g <- fitdist(runs, "gamma")
runs.ln <- fitdist(runs, "lnorm")
runs.w <- fitdist(runs, "weibull")
runs.geo <- fitdist(runs, "geom")

#plot.legend <- c("Gamma", "LogNorm", "Weibull")
#denscomp(list(runs.g, runs.ln, runs.w), legendtext = plot.legend, plotstyle = "ggplot", breaks = 1:150)

runs.g
runs.ln
runs.w
runs.geo

plot(runs.geo, breaks = 0:150)
#plot(runs.ln, breaks = 0:150)
```

APPENDIX D:

Histogram of Number of Runs


Number of Runs Before End of Game
Frequency
Number of Runs