# Blog 3 - Quarto Simulation

Michael Ippolito

2022-11-24

## Contents



### Background

Quarto is a two-player strategy game akin to chess, but not as complicated, well known, or widely played. It is played on a four-by-four tiled board with sixteen pawns having different sizes and shapes. The object of the game is to beat your opponent by getting four matching pawns in a row, either horizontally, vertically, or diagonally.

Each pawn has characteristics from each of four categories:

1. Round or square
2. Solid or hollow
3. Tall or short
4. Dark or light

For example, the first pawn is round, solid, tall, dark (and handsome?). The second pawn is round, solid, tall, and light. And so on.

The more I played the game, the more I wondered if there is an advantage to going first versus second, and how often the game ends in a win for one player as opposed to the game ending in a draw (i.e. when no four pawns match in any direction). I got the idea to data-science the problem to figure out the answers. Concretely, my objectives were:

1. Find out if there is an advantage to going first versus second.
2. Find the number of times the game ends in a win versus a draw.
3. Find the number of moves needed to win.
4. See if this number follows a particular distribution and statistically evaluate goodness of fit for that distribution.

I'll make some assumptions for this analysis, namely:

1. Since the computer (rather than a human) will be selecting the game pieces to play and where to play them, I'll assume that the players are evenly matched.
2. The players are allowed to make mistakes. I.e., if a player places a pawn in a position where the other player can easily win, there are no backsies; the second player will be allowed to win. In a friendly game between two humans, this assumption wouldn't hold, but it makes the logic for this analysis more straightforward.
3. Since the players are evenly matched and are allowed to make mistakes, I'll assume that there should be no prior strategy by either player and, therefore, a randomly played game is just as likely to produce a victory as one played more strategically.

## Initialize Game

First, I'll initialize the game board by defining each pawn and setting up a blank set of squares.

```
## [1] "Pawns:"

## [1] "1 solid square dark short avail"
## [1] "2 solid square dark tall avail"
## [1] "3 solid square light short avail"
## [1] "4 solid square light tall avail"
## [1] "5 solid round dark short avail"
## [1] "6 solid round dark tall avail"
## [1] "7 solid round light short avail"
## [1] "8 solid round light tall avail"
## [1] "9 hollow square dark short avail"
## [1] "10 hollow square dark tall avail"
## [1] "11 hollow square light short avail"
## [1] "12 hollow square light tall avail"
## [1] "13 hollow round dark short avail"
## [1] "14 hollow round dark tall avail"
## [1] "15 hollow round light short avail"
## [1] "16 hollow round light tall avail"

## [1] ""
```

A number of functions will be needed, for example:

| function | purpose |
|---|---|
| clear_board | Empties the board of pawns |
| play_random_pawn | Plays a randomly selected pawn onto a randomly selected square of the game board |
| is_won | Evaluates the game board to determine whether there is a set of four pawns in a row in any direction |

## Game Play

Now for the actual game play, of which I'll iteratively run 1,000 times. For each game, a randomly selected pawn will be placed on a randomly selected empty square of the game board. After each pawn is played, the game board will be evaluated for whether one of the players has won or not. If the number of moves up until that point is odd, that indicates the player who went first won; conversely, if the number of moves is even, the second player is the winner.

```
# Iteratively play random games, talying the number of times the game is won and
# the number of wins by the first and second players
num_reps <- 1000  # the number of repetitions to perform
won_games <- c()  # the total number of games won
move_num <- c()  # the number of moves needed to win the game
set.seed(8)

# Iterate through reps
for (j in seq(1, num_reps)) {

    # Initialize game
    sqr <- init_squares()  # initialize game squares
    board <- clear_board()  # clear the game board
    pawns <- init_pawns()  # initialize the game pawns

    # Iterate through each square on the board
    for (i in seq(1, 16)) {
        board <- play_random_pawn()  # get index of pawn played
        game_is_won = F
        # Impossible for the game to be won unless there are at least 4 pawns on the board
        if (i > 3) {
            game_is_won <- is_won(board)  # see if board is won
        }
        if (game_is_won | i == 16) {
            # Add the number of won games to the tally
            won_games <- c(won_games, game_is_won)
            if (game_is_won) {
                # Add the number of moves required to win
                move_num <- c(move_num, i)
            } else {
                # If the game is a drawn, indicate this by setting move_num to zero
                move_num <- c(move_num, 0)
            }
            break
        }
    }
}
```

## Results

Now let's look at the results of the simulation.

```
## [1] "First 10 won games:"
```

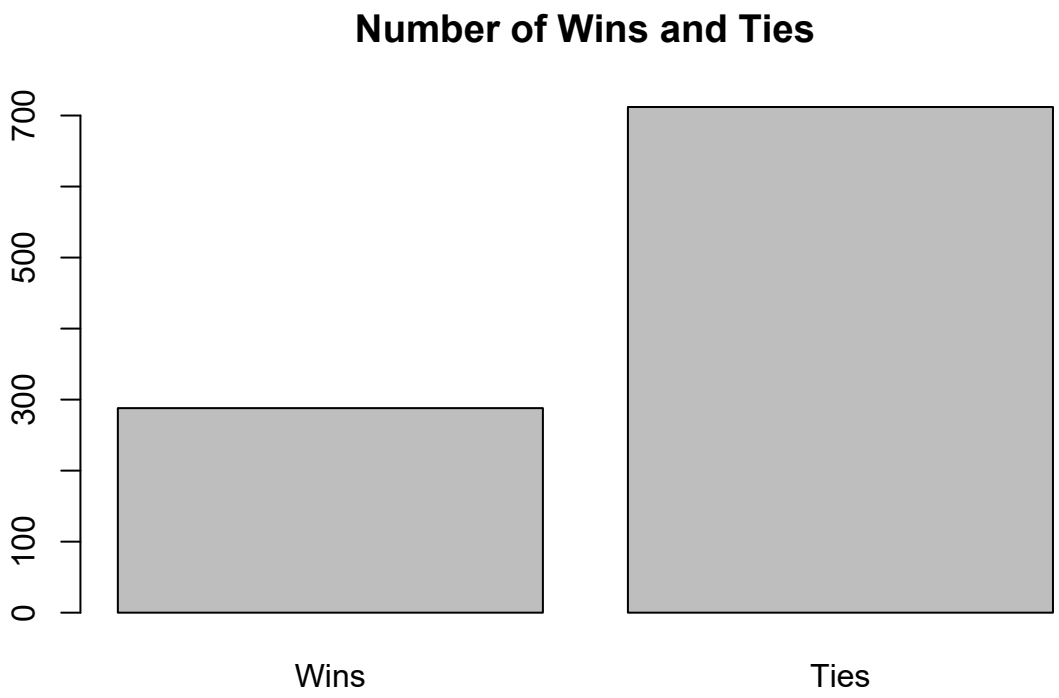
```
##  [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE
```

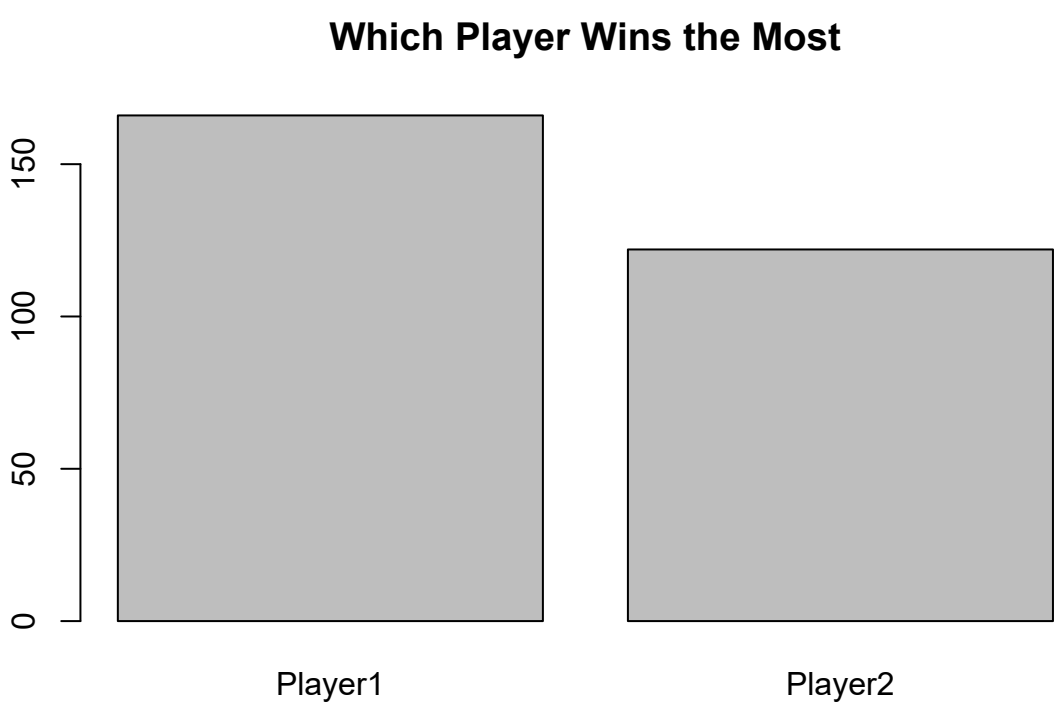
```
## [1] ""
```

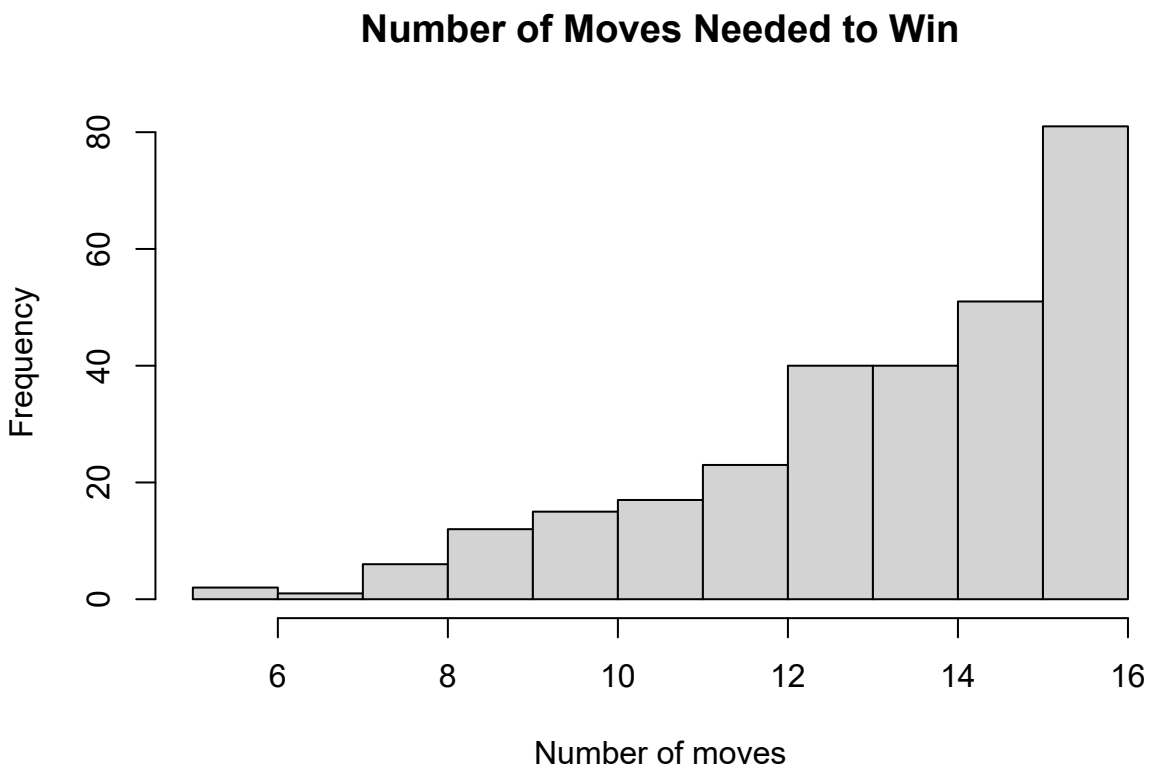
```
## [1] "First 10 number of moves:"
```

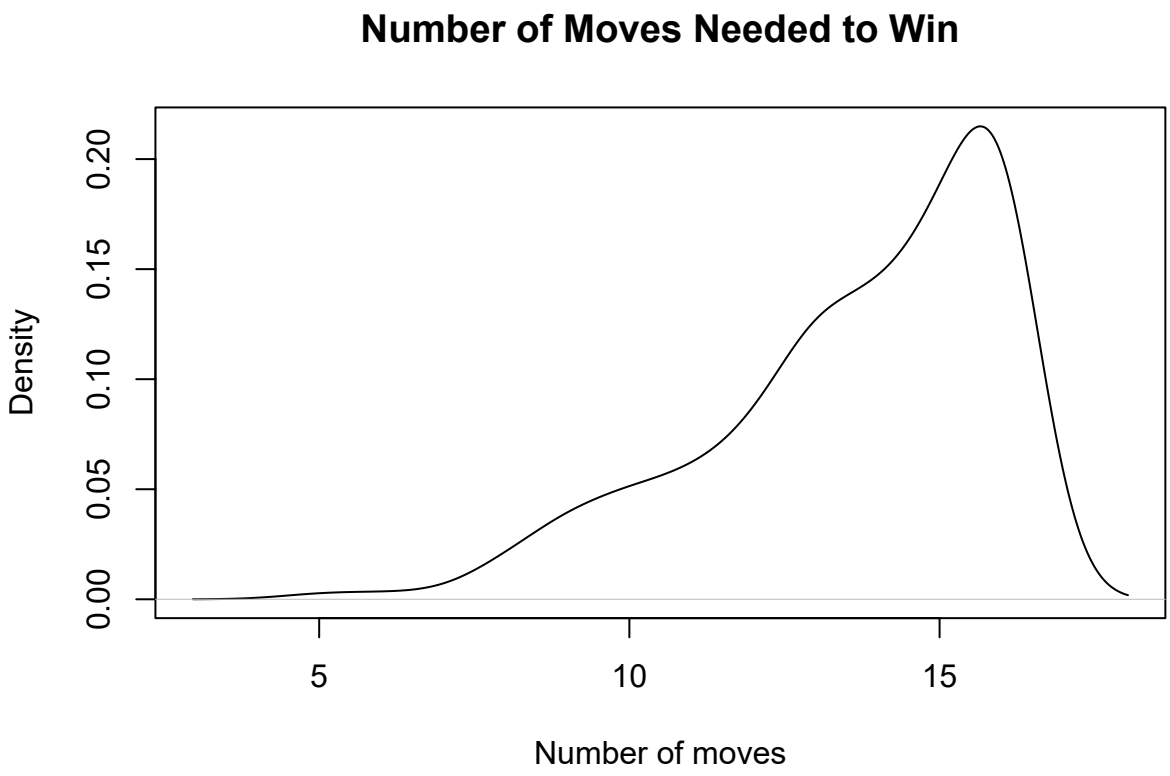
```
##  [1]  0  0 13  9 16 16  0  0 15  0
```

```
## [1] ""

## [1] "won 288 of 1000 (0.288)"

## [1] "First player won: 166 of 288 (0.576)"

## [1] "Second player won: 122 of 288 (0.424)"

## [1] "Average number of moves to win (when the game is won): 13.6"

## [1] "Median number of moves to win (when the game is won): 14"
```




Which Player Wins the Most
150
100
50
0
Player1
Player2


Number of Moves Needed to Win
Frequency
0
20
40
60
80
6
8
10
12
14
16
Number of moves

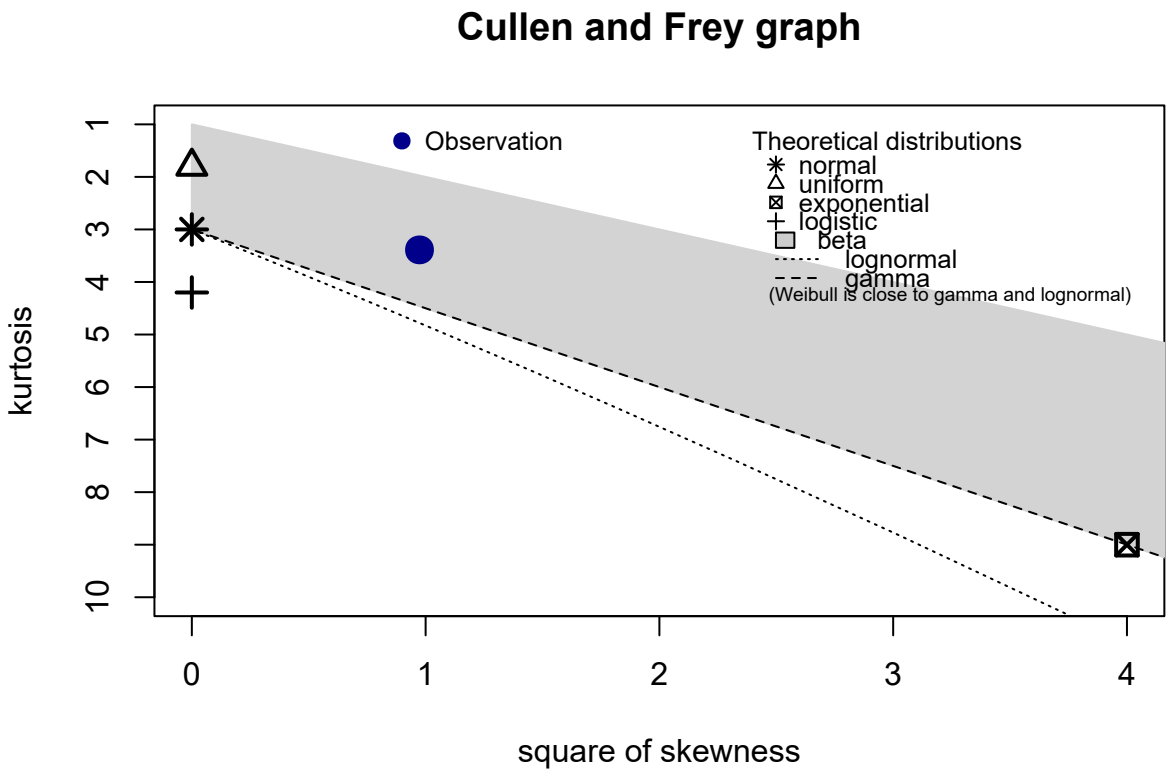



As shown, only a little more than a quarter (28.8%) of the games result in a win rather than a draw. And the first player has a clear advantage over the second, winning almost 8% more games. When the game results in a win, it takes an average of 13.6 moves to win it, with a median of 14.

## Distribution

Now we'll try to fit a distribution to the data. To do this, we'll first create a frequency table. Then we'll use the function "descdist" from the fitdistrplus package to look for a suitable distribution.

```
# Find distribution
descdist(move_num[move_num > 0], discrete=F)
```




```
## summary statistics
## ------
## min:  5   max:  16
## median:  14
## mean:  13.63889
## estimated sd:  2.340705
## estimated skewness:  -0.9868369
## estimated kurtosis:  3.389112
```

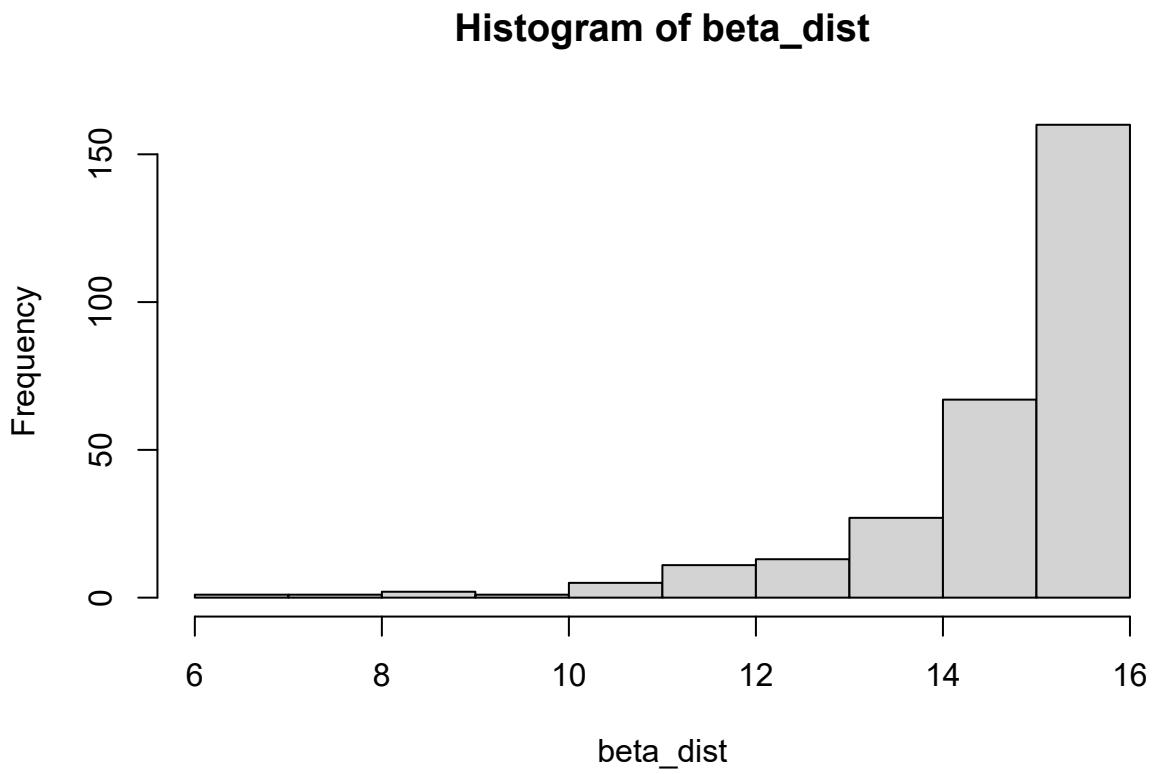
Based on the graph, a beta distribution seems like a good fit. The beta distribution takes two parameters: shape1 and shape2. We'll try to find these parameters by using the "fitdist" function from the same package.

```
# Fit beta distribution
(fd <- fitdist(df$p, 'beta'))
```

```
## Fitting of the distribution ' beta ' by maximum likelihood
## Parameters:
##         estimate Std. Error
## shape1 0.7195345  0.2511189
## shape2 8.0128068  3.7556951
```

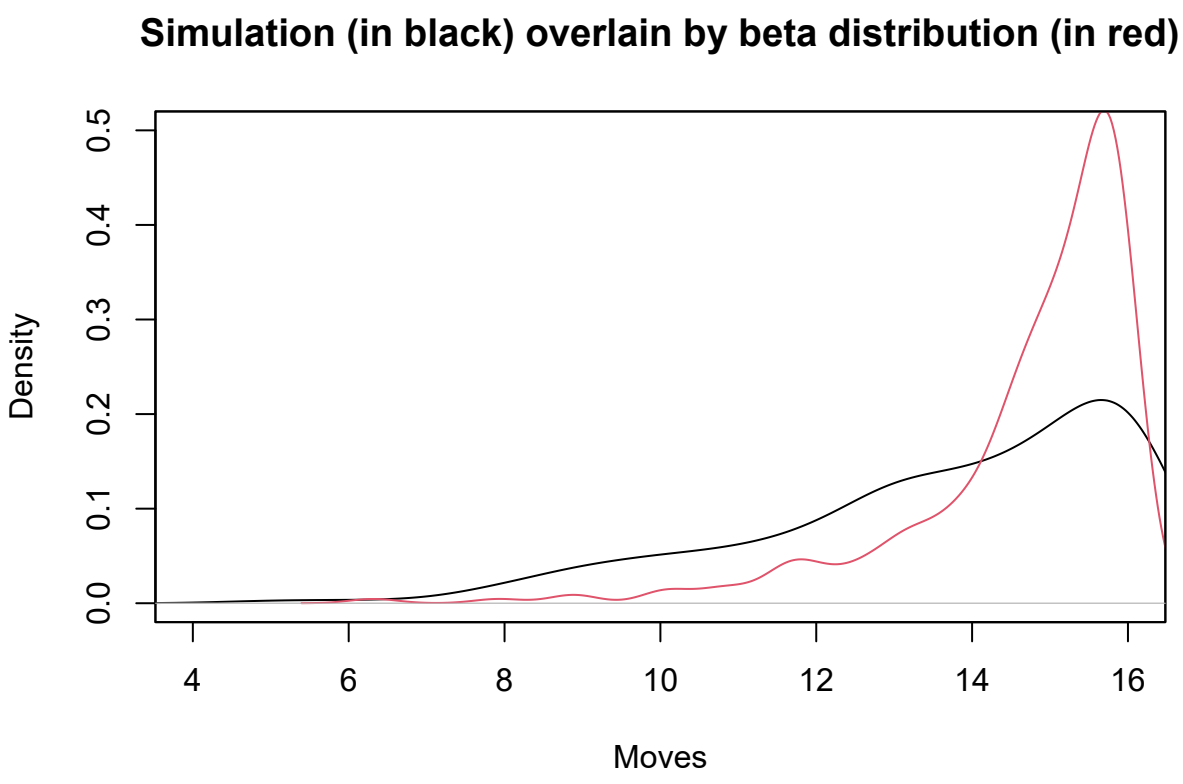
Now we'll generate a beta sequence using the rbeta function so we can compare it to the sequence generated by the simulation.

```r
# Generate a sequence following the beta distribution using the specified estimates
beta_dist <- rbeta(n=total_won, shape1=fd$estimate[2], shape2=fd$estimate[1])
beta_dist <- beta_dist * 16
hist(beta_dist)
```




Overlaying the density plots of the two distributions, we see that it's not a bad match.

```r
# Overlay plots
plot(density(move_num[move_num > 0]),
     main='Simulation (in black) overlain by beta distribution (in red)',
     xlab='', ylab='', xaxt='n', yaxt='n', xlim=c(4, 16), ylim=c(0, 0.5))
par(new=T)
plot(density(beta_dist), main='', col=2,
     xlab='Moves', ylab='Density', xlim=c(4, 16), ylim=c(0, 0.5))
```

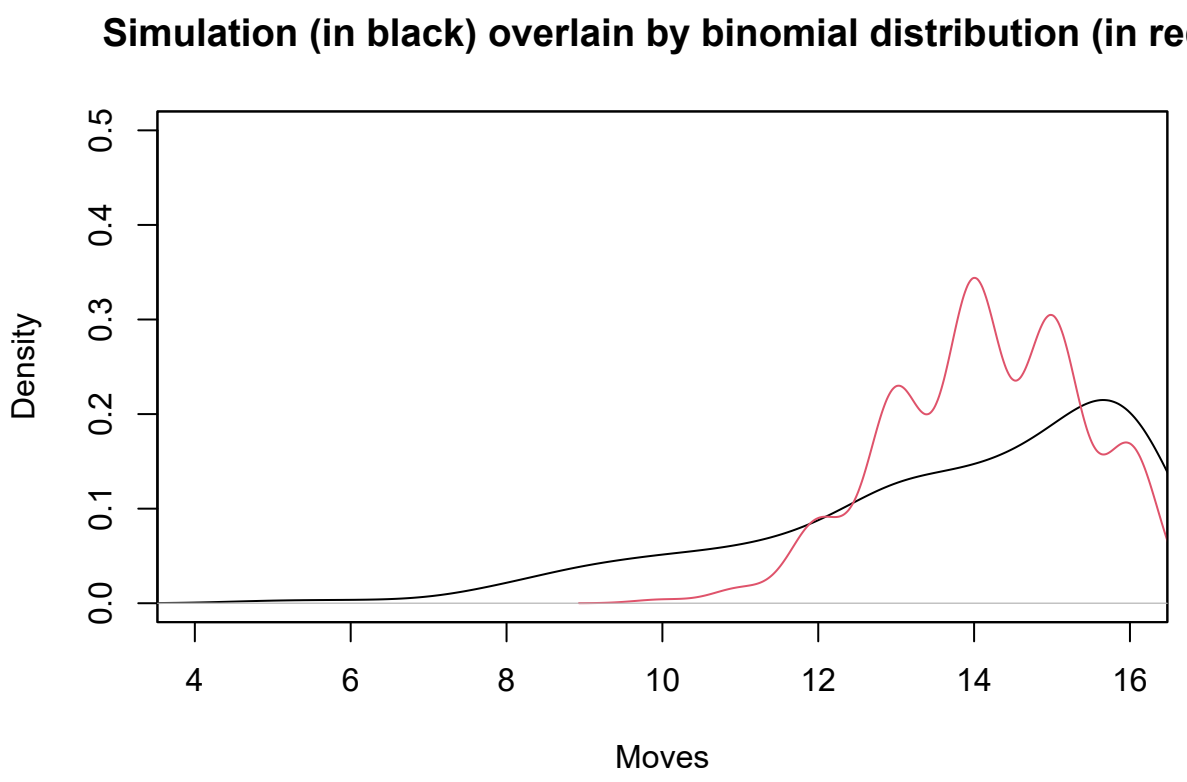



Now we'll make the same comparison using a binomial distribution.

```
# Generate a sequence following the binomial distribution using the specified estimates
binom_dist <- rbinom(n=total_won,size=12,prob=1-mean_p)
binom_dist <- binom_dist + 4
hist(binom_dist)
```


Again, we'll overlay the plots to see how well they match.

```
# Overlay plots
plot(density(move_num[move_num > 0]),
     main='Simulation (in black) overlain by binomial distribution (in red)',
     xlab='', ylab='', xaxt='n', yaxt='n', xlim=c(4, 16), ylim=c(0, 0.5))
par(new=T)
plot(density(binom_dist), main='', col=2,
     xlab='Moves', ylab='Density', xlim=c(4, 16), ylim=c(0, 0.5))
```




To quantitatively evaluate how well the two distributions fit the data, we'll perform chi-squared tests.

```
# Chi-squared test comparing distributions
chisq.test(x=df$Moves, y=dfbeta$Moves)
```

```
## Warning in chisq.test(x = df$Moves, y = dfbeta$Moves): Chi-squared approximation
## may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$Moves and dfbeta$Moves
## X-squared = 240, df = 225, p-value = 0.2348
```

```
chisq.test(x=df$Moves, y=dfbinom$Moves)
```

```
## Warning in chisq.test(x = df$Moves, y = dfbinom$Moves): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$Moves and dfbinom$Moves
## X-squared = 240, df = 225, p-value = 0.2348
```

## Conclusions

As shown, the beta and binomial distributions form a relatively good fit to the distribution of wins geneated by the random game simulations. This is confirmed by chi-squared tests, which yielded p-values above 0.05. In addition, the objectives were met with the following conclusions:

1. There is a clear advantage to going first versus second, with the first player winning 57.6% of the time.
2. The number of times the game ends in a win versus a draw is roughly 28.8% of games.
3. On average, it takes 13.6 moves to win a game (when the game doesn't end in a draw), with a median of 14.
4. The distribution of won games roughly follows the beta and binomial distributions with statistical evidence to support the fit.