Roulette Simulation Blog

Christopher Joseph

Roulette Simulator

Solution Code

```
single_spin <- function(){
  possible_outcomes <- c(rep("red",18), rep("black",18), rep("green",2))
  sample(possible_outcomes, 1)
}

martingale_wager <- function(
    previous_wager
  , previous_outcome
  , max_wager
  , current_budget
){
  if(previous_outcome == "red") return(1)
  min(2*previous_wager, max_wager, current_budget)
}

one_play <- function(previous_ledger_entry, max_wager){
  # Create a copy of the input object that will become the output object
  out <- previous_ledger_entry
  out[1, "game_index"] <- previous_ledger_entry[1, "game_index"] + 1
  out[1, "starting_budget"] <- previous_ledger_entry[1, "ending_budget"]
  out[1, "wager"] <- martingale_wager(
    previous_wager = previous_ledger_entry[1, "wager"]
    , previous_outcome = previous_ledger_entry[1, "outcome"]
    , max_wager = max_wager
    , current_budget = out[1, "starting_budget"]
  )
  out[1, "outcome"] <- single_spin()
  out[1, "ending_budget"] <- out[1, "starting_budget"] +
    ifelse(out[1, "outcome"] == "red", +1, -1)*out[1, "wager"]
  out[1, "play_number"] <- previous_ledger_entry[1, "play_number"] + 1
  return(out)
}

one_series <- function(
  max_games, starting_budget, winning_threshold, max_wager
){
  # Initialize ledger
  ledger <- data.frame(
```

```
      game_index = 0:max_games
    , starting_budget = NA_integer_
    , wager = NA_integer_
    , outcome = NA_character_
    , ending_budget = NA_integer_
    , play_number = NA_integer_
  )
  ledger[1, "wager"] <- 1
  ledger[1, "outcome"] <- "red"
  ledger[1, "ending_budget"] <- starting_budget
  for(i in 2:nrow(ledger)){
    ledger[i, "play_number"] <- i - 1 # Replaced browser() to increment play number and count it.
    ledger[i,] <- one_play(ledger[i-1,], max_wager)
    if(stopping_rule(ledger[i,], winning_threshold)) break
  }
  # Return non-empty portion of ledger
  ledger[2:i, ]
}

stopping_rule <- function(
  ledger_entry
  , winning_threshold
){
  ending_budget <- ledger_entry[1, "ending_budget"]
  if(ending_budget <= 0) return(TRUE)
  if(ending_budget >= winning_threshold) return(TRUE)
  FALSE
}

profit <- function(ledger){
  n <- nrow(ledger)
  profit <- ledger[n, "ending_budget"] - ledger[1, "starting_budget"]
  return(profit)
}

require(magrittr)
```

```
## Loading required package: magrittr
```

```
svg(filename = "loser.svg", width=16, height =9)
par(cex.axis=2, cex.lab = 2, mar = c(8,8,2,2), bg = rgb(222, 235, 247, max = 255))
set.seed(1)
ledger <- one_series(200,200,300,500)
plot(ledger[,c(1,5)], type = "l", lwd = 5, xlab = "Game Index", ylab = "Budget")
dev.off()
```

```
## pdf
##   2
```

```
svg(filename = "winner.svg", width=16, height =9)
par(cex.axis=2, cex.lab = 2, mar = c(8,8,2,2), bg = rgb(222, 235, 247, max = 255))
set.seed(2)
```

```r
l2 <- one_series(200,200,300,500)
plot(l2[,c(1,5)], type = "l", lwd = 5, xlab = "Game Index", ylab = "Budget")
dev.off()
```

```
## pdf
##   2
```

Chris's Crazy Blog!

In this blog, I will write a blog post to explain how I used computer simulation to understand the operating characteristics of a strategy that I pasted within the code of this document. I will go over how I used simulation to calculate the average earnings of a gambler that uses the strategy I mentioned, and provide figures to demonstrate this. After that, I will demonstrate how changing a parameter can impact or not impact overall earnings for a specific play.

A very popular casino game, roulette, is played by chancing a bet on either a red, green, or black pocket each with a 18/38, 2/38, and an 18/38 probability respectively. In this blog post, I will aim to calculate the average earnings of an individual that uses the "Martingale Strategy": a strategy where an individual doubles their bet after each loss with the goal of recovering previous losses while making a profit. I will provide two simulations, each with a different "randomness" seed to show how effective this strategy would be for two different scenarios. Imagine in Scenario 1 that a man named Dave walked into a casino to play roulette and used the Martingale strategy. Now imagine in Scenario 2 that a woman named Bella walked into a different casino to play roulette and used the same strategy. Both games outputs are different because their seeds are different, and that way we can see how variable this strategy is depending on how lucky a person is. The code for each simulation is shown below:

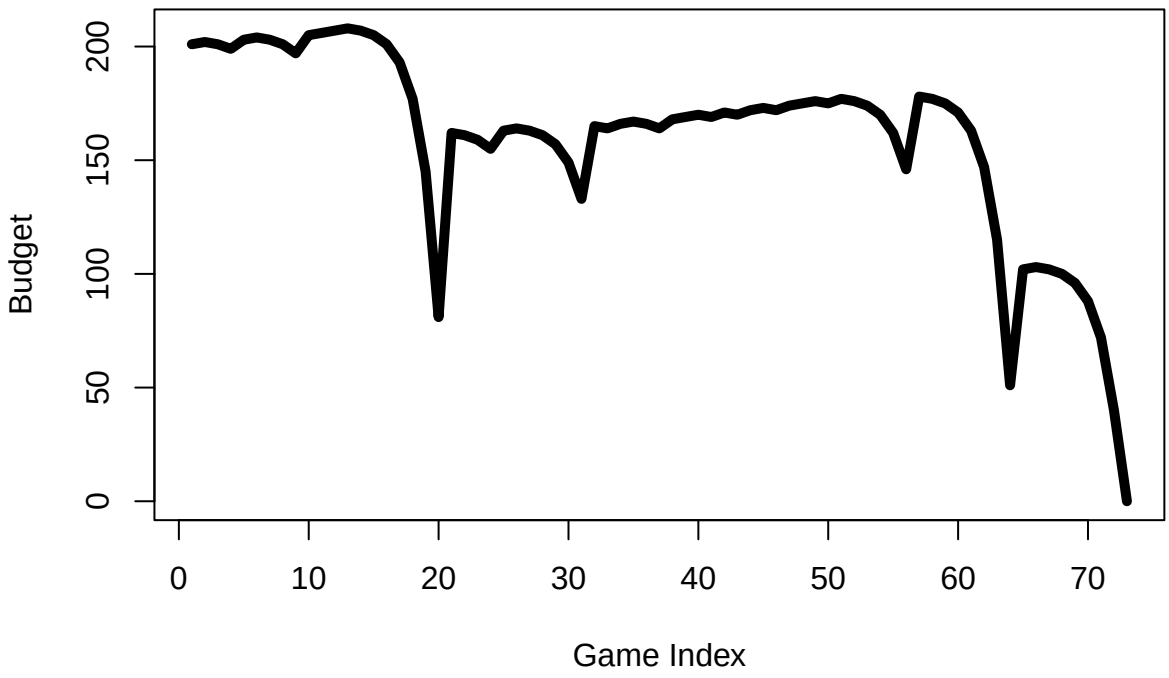
Simulation 1

```r
set.seed(1)

# Run the first simulation, where max_games = 200,
# starting_budget = 200, winning_threshold = 300,
# and max_wager = 500.
ledger1 <- one_series(200, 200, 300, 500)
average_earnings1 <- profit(ledger1) / nrow(ledger1)
cat("Average Earnings Per Play for Dave: $", average_earnings1, "\n")
```

```
## Average Earnings Per Play for Dave: $ -2.739726
```

```r
plot(ledger1[, c(1, 5)], type = "l", lwd = 5, xlab = "Game Index", ylab = "Budget")
```



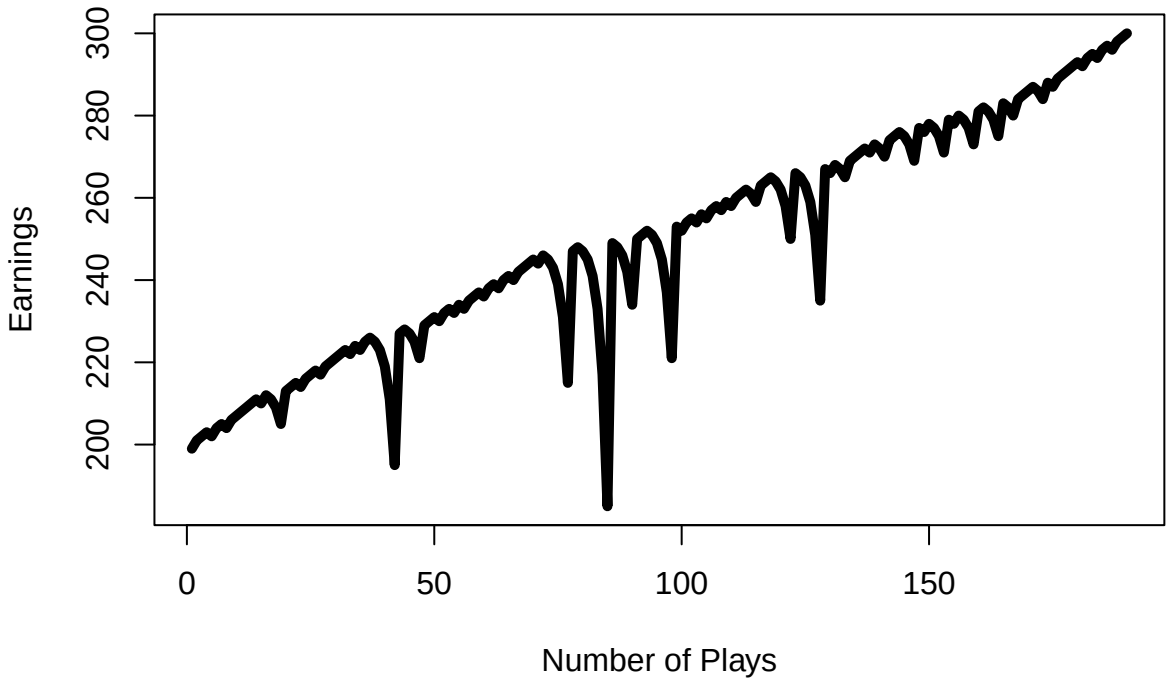
Simulation 2

```
set.seed(2)

# Run the second simulation, where max_games = 200,
# starting_budget = 200, winning_threshold = 300,
# and max_wager = 500.
ledger2 <- one_series(200, 200, 300, 500)
average_earnings2 <- profit(ledger2) / nrow(ledger2)
cat("Average Earnings Per Play for Bella: $", average_earnings2, "\n")
```

```
## Average Earnings Per Play for Bella: $ 0.5263158
```

```
plot(ledger2[, c(1, 5)], type = "l", lwd = 5, xlab = "Number of Plays", ylab = "Earnings")
```



Here we can see that Dave and Bella have drastically different outcomes despite using the same exact strategy. You could say that Dave is unlucky, as his overall earnings is -200, meaning he lost money, while Bella earned a total of $100. The average earnings per play for Dave and Bella are respectively -$2.74 and $0.53. I calculated the averages by using the overall "profit" function that the original code provided, and dividing it by the number of plays that there were, thus giving an "average number per play" for each gambler.

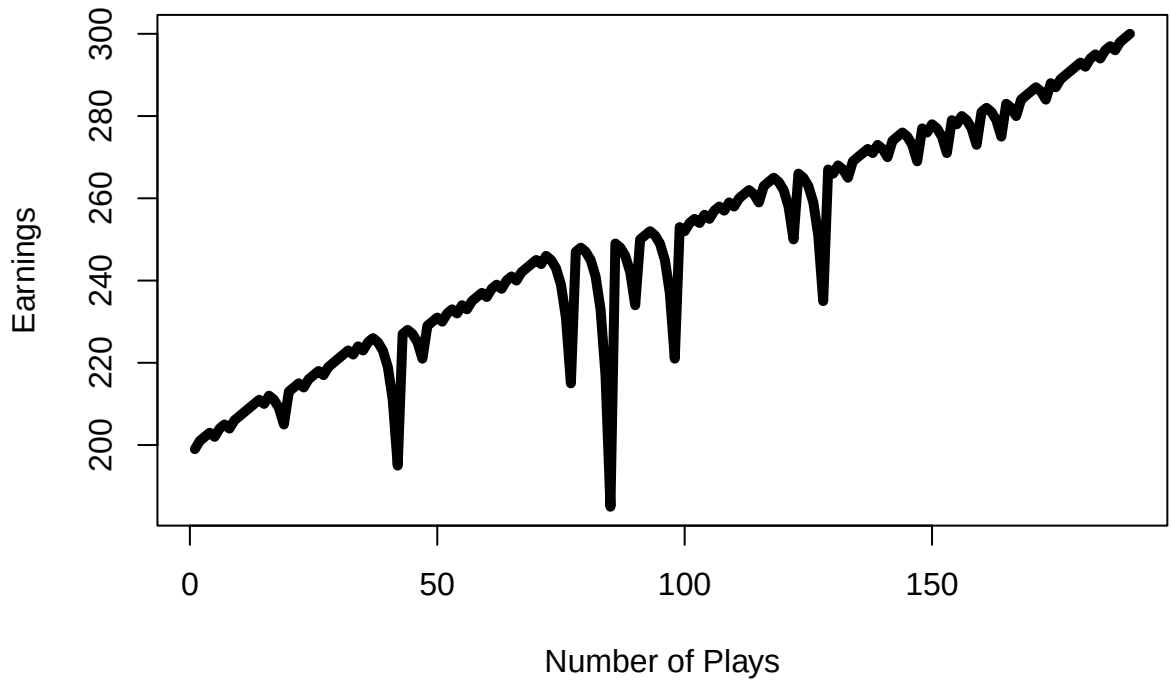
While these metrics are significant in showing the disparity in earnings between two scenarios, perhaps that could be attributed to a flaw in how the player is betting. Remember that we have several different parameters to take into account when playing roulette, detailed below in the table:

Summary of parameters

Parameter	Description
`starting_budget`	The Starting budget of our gambler
`winning_threshold`	The threshold of earnings that will satisfy our gambler and make them stop playing
`max_games`	The amount of games our gambler is willing to play before stopping
`max_wager`	The Casino's maximum wager amount

Let us observe how each of the parameters may affect Bella's earnings. Remember that the first figure represents Bella's earnings when her paramaters were:

Parameter	Starting value
`starting_budget`	$200
`winning_threshold`	$300 (Starting budget + $100 winnings)
`max_games`	200 plays
`max_wager`	$500

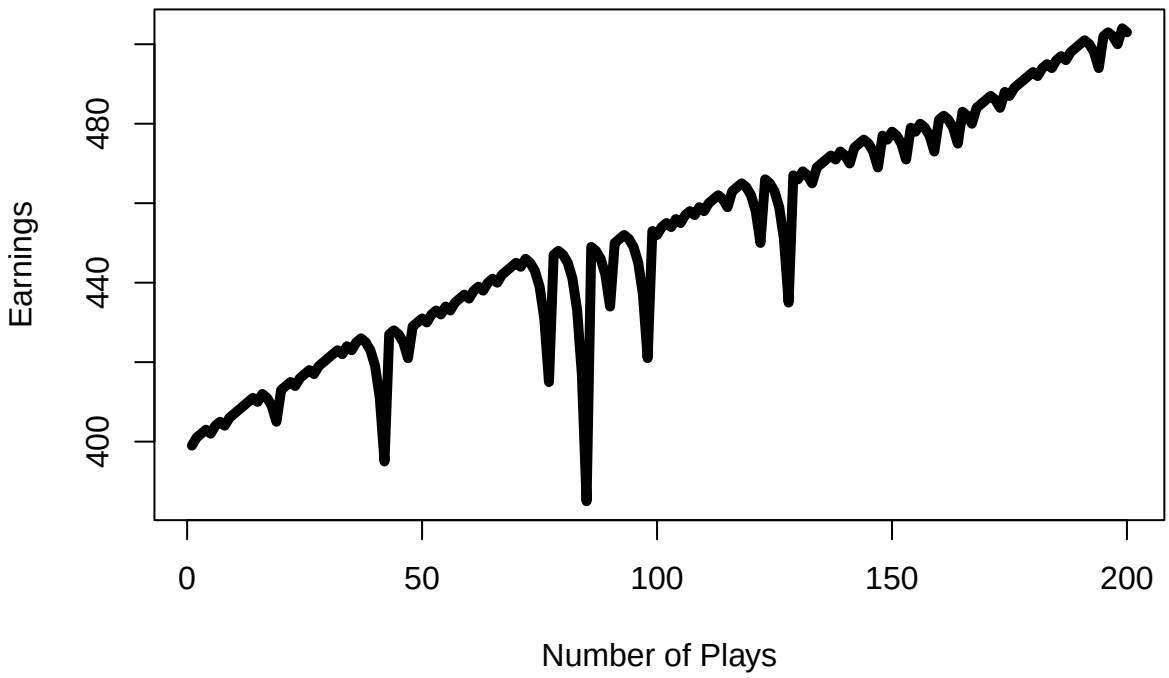


Now, let's say that Bella has more money to spend this time, and decides to start with double her original amount. Because of this, let's say her winning threshold also doubles. Her parameters now look like:

Parameter	Starting value
`starting_budget`	$400 (Double the previous)
`winning_threshold`	$600 (Double the previous)
`max_games`	200 plays
`max_wager`	$500

Let's see how this will affect her earnings:

```
## Average Earnings Per Play for Bella: $ 0.515
```

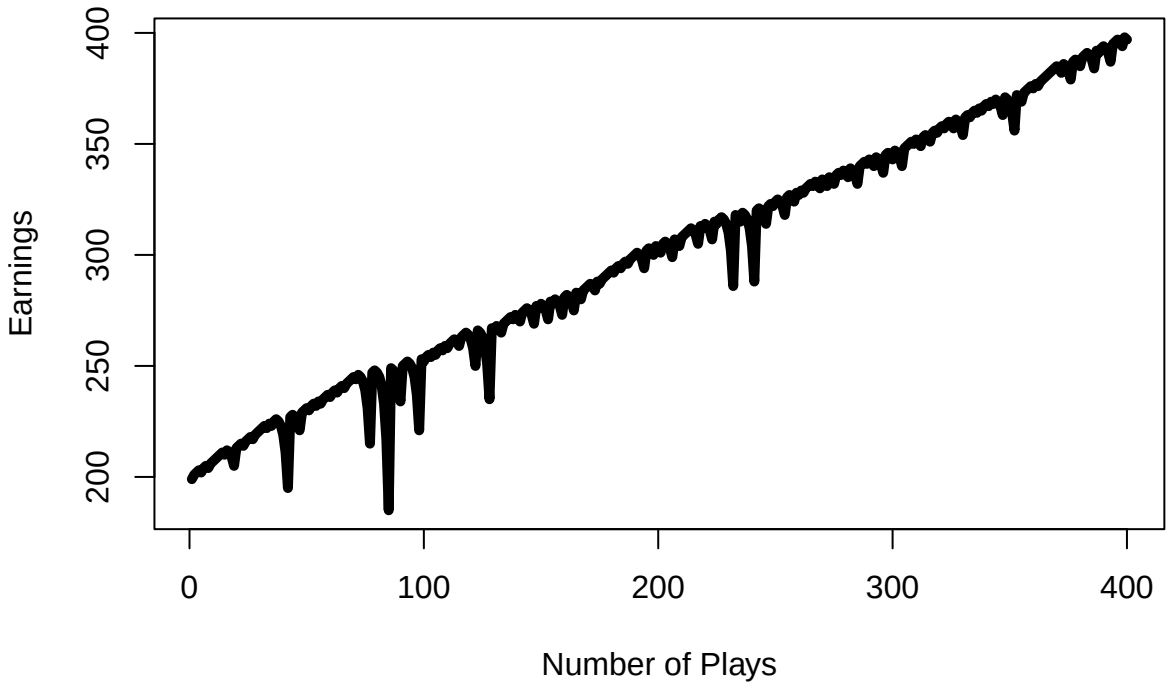


Overall, the trend in the figure is still the same as before. This makes sense considering that we only changed our starting and ending amounts. The bets are still the same. The average earnings per play is also pretty consistent with our previous figure.

Let's consider reverting back to our original starting budget, and instead focus on doubling our max games. What would happen if Bella played double the amount of plays, and had double the winning threshold than the original? Her parameter table would look like:

Parameter	Starting value
`starting_budget`	$200
`winning_threshold`	$600 (Double the original)
`max_games`	400 plays (Double the original)
`max_wager`	$500

```
## Average Earnings Per Play for Bella: $ 0.4925
```

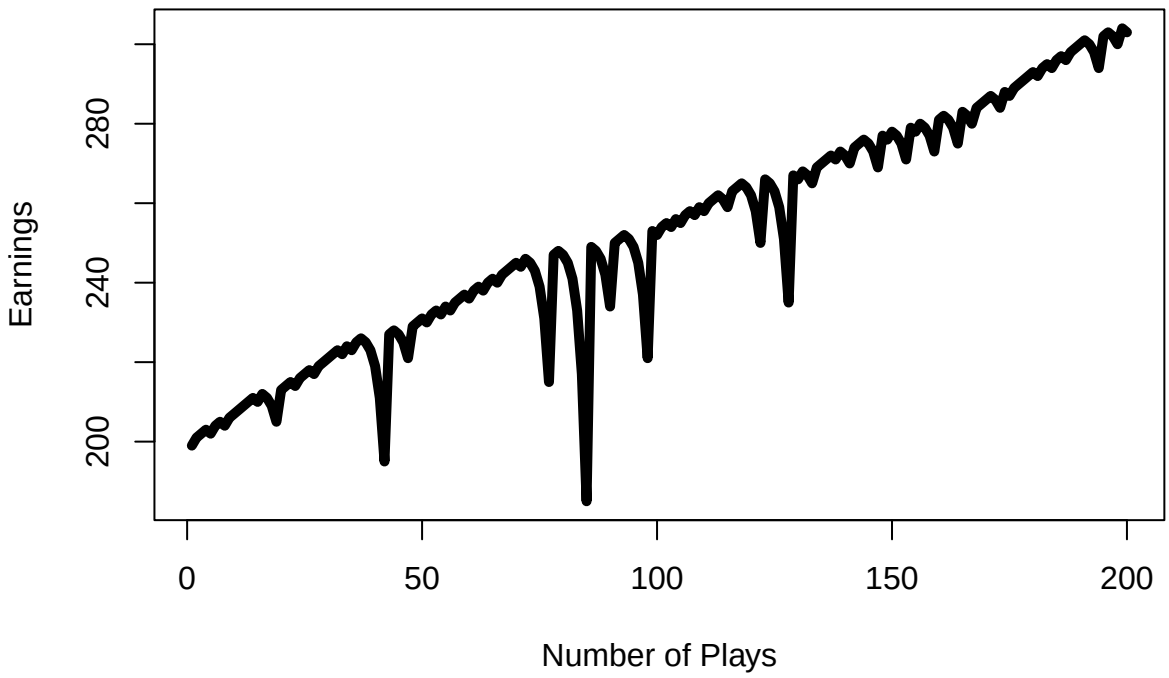


Interestingly enough, Bella still remains very lucky and her earnings are largely the same.

What if we now changed the max wager to double the original amount and reverted all other parameters back to their defaults?

Parameter	Starting value
`starting_budget`	$200
`winning_threshold`	$600 (Double the original)
`max_games`	200 plays (Double the original)
`max_wager`	$1000

```
## Average Earnings Per Play for Bella: $ 0.515
```



Now, there is a reason why these figures are not changing much. This is because inherently, our seed is still the same across all the figures, meaning that "Bella" will always have the same luck as she across all parameter variations of her strategy, which ultimately can affect how much she earns, but it will not change how the figures will trend. Hence, there is one aspect we should look at to really see if the Martingale strategy may be a good strategy: running multiple simulations at once.

First, I had to modify the original code to keep track of the play count so that when a simulation is over, we know how many plays it took for a stopping rule to apply. I did this by first adding a play_number column to the ledger, and then I made sure to replace the browser() line with a play counter so that the ledger can now return a play count.

In the code I have below, I defined a function "single_sim" that does everything the one_series function from the original code does, but with the addition of a parameter "seed" that allows us to set which seed we wish to use for the game. The reason I did this is so that when we want to run multiple simulations, like 100, we can then set 100 different seeds to get as much variability as possible with each game. In the code below, I have the seed parameter set as 0:99, which means that for each of the 100 simulations it will be a unique seed between 0 and 99. From then on, we can take the play counts of the 100 simulations and take the average to see how long the average game takes to reach a stopping rule.

```
# each row in the ledger is one play, we can get the number of plays using the number of rows

replicate(1000, one_series(1000, 200, 300, 100) |> nrow()) |> mean()
```

```
## [1] 209.422
```

Be sure to explain the limitations of the simulation; identify simplifications or other sources of uncertainty.

As with any simulation, there are many simplifications and assumptions that must be made in order to create a model. Becuase of this, this can cause limitations in the simulation itself. The simulations that I used, specifically with Bella are limited in the sense of looking at a specific seed for Bella's night at the casino. It may seem that Bella is very lucky, when in reality, each new spin of a roulette wheel is very variable. In this simulation, we had to assume first that the roulette wheel was not biased. We also have to assume the gambler only stops when a stopping rule is met, but there could be cases where a gambler has to leave before the rules are satisfied, maybe they have a phone call, or maybe they got kicked out of the casino. The simulation also fails to account for casino win limits, where an individual is unable to earn more than a designated amount set by a casino in the event that they win a lot. Overall, there is still uncertainty among these real world factors that prevent that model from being a direct representation of reality, but the simulation does well to mimic it.