HW2

Garrett Pinkston

Problem 1 - Dice Game

a) Build the following versions

- Version 1: Implement this game using a loop.

```
loopDiceGame <- function(nrolls){
  wallet <- 0
  for (i in 1:nrolls) {
    roll <- sample(1:6,1)
    if ((roll == 3) || (roll == 5)) {
    wallet <- wallet + (2 * roll)
    }
    wallet <- wallet - 2
  }
  return(wallet)
}

set.seed(123)
loopDiceGame(3000)
```

```
[1] 2174
```

- Version 2: Implement this game using built-in R vectorized functions.

```
vectorizedDiceGame <- function(nrolls){
  wallet <- 0

  allRolls <- sample(1:6,nrolls,replace=TRUE)

  wallet <- (allRolls == 3 | allRolls == 5) * 2 *allRolls
```

```r
  wallet <- sum(wallet) - (2*nrolls)

  return(wallet)
}

set.seed(123)
vectorizedDiceGame(3000)
```

```
[1] 2174
```

- Version 3: Implement this by rolling all the dice into one and collapsing the die rolls into a single `table()`. (Hint: Be careful indexing the table - what happens if you make a table of a single dice roll? You may need to look to other resources for how to solve this.)

```r
tableDiceGame <- function(nrolls){
  wallet <- 0

  allRolls <- sample(1:6,nrolls,replace=TRUE)
  allRolls <- factor(allRolls,levels=1:6) #need to ensure all counts appear in table (even i

  rollsTable <- table(allRolls)

  profitForThrees <- as.numeric(rollsTable["3"]) * 2 * 3 # numOccurences * payoff * numberPro
  profitForFives <- as.numeric(rollsTable["5"]) * 2 * 5 # numOccurences * payoff * numberPro

  wallet <- profitForThrees + profitForFives
  wallet <- wallet - (2 * nrolls)

  return(wallet)
}
set.seed(123)
tableDiceGame(3000)
```

```
[1] 2174
```

- Version 4: Implement this game by using one of the "`apply`" functions.

```r
applyDiceGame <- function(nrolls){
  wallet <- 0
```

```
  allRolls <- sample(1:6,nrolls,replace=TRUE)

  allProfits <- vapply(allRolls,function(individualRoll){
      if ((individualRoll == 3) || (individualRoll == 5)) {
        return(as.integer(2 * individualRoll))
      } else{
        return(0L)
      }
  },integer(1))

  wallet <- sum(allProfits)
  wallet <- wallet - (2 * nrolls)
  return(wallet)
}
set.seed(123)
applyDiceGame(3000)
```

```
[1] 2174
```

b) Demonstrate that all versions work. Do so by running each a few times, once with an input a 3, and once with an input of 3,000.

```
loopDiceGame(3)
```

```
[1] -6
```

```
loopDiceGame(3000)
```

```
[1] 1418
```

```
vectorizedDiceGame(3)
```

```
[1] -6
```

```
vectorizedDiceGame(3000)
```

```
[1] 2306
```

```r
tableDiceGame(3)
```

```
[1] 0
```

```r
tableDiceGame(3000)
```

```
[1] 2084
```

```r
applyDiceGame(3)
```

```
[1] 10
```

```r
applyDiceGame(3000)
```

```
[1] 1972
```

as we can observe, all functions work and provide output. 8 functions were tested and 8 reasonable pieces of output were returned.

c) Demonstrate that the four versions give the same result. Test with inputs 3 and 3,000. (You will need to add a way to control the randomization.)

```r
set.seed(123)
loopDiceGame(3)
```

```
[1] 6
```

```r
set.seed(123)
vectorizedDiceGame(3)
```

```
[1] 6
```

```r
set.seed(123)
tableDiceGame(3)
```

```
[1] 6
```

```r
set.seed(123)
applyDiceGame(3)
```

```
[1] 6
```

```r
set.seed(123)
loopDiceGame(3000)
```

```
[1] 2174
```

```r
set.seed(123)
vectorizedDiceGame(3000)
```

```
[1] 2174
```

```r
set.seed(123)
tableDiceGame(3000)
```

```
[1] 2174
```

```r
set.seed(123)
applyDiceGame(3000)
```

```
[1] 2174
```

d) Use the microbenchmark package to clearly demonstrate the speed of the implementations. Compare performance with a low input (1,000) and a large input (100,000). Discuss the results

```r
library(microbenchmark)

smallBenchmark <- microbenchmark(
  loop = loopDiceGame(1000),
  vectorized = vectorizedDiceGame(1000),
  table = tableDiceGame(1000),
  vapply = applyDiceGame(1000),
  times = 100
)
```

```
Warning in microbenchmark(loop = loopDiceGame(1000), vectorized =
vectorizedDiceGame(1000), : less accurate nanosecond times to avoid potential
integer overflows
```

```r
print(summary(smallBenchmark))
```

```
        expr      min       lq       mean    median        uq      max neval
1       loop 1740.983 1769.745 1855.41646 1792.8275 1845.9020 2610.265   100
2 vectorized   25.502   26.814   27.82711   27.4905   28.1055   41.615   100
3      table   55.432   56.826   61.34912   58.0150   62.2585   99.302   100
4     vapply  235.299  242.351  300.54886  244.9135  252.0680 2814.158   100
```

```r
largeBenchmark <- microbenchmark(
  loop = loopDiceGame(100000),
  vectorized = vectorizedDiceGame(100000),
  table = tableDiceGame(100000),
  vapply = applyDiceGame(100000),
  times = 10
)

print(summary(largeBenchmark))
```

```
        expr        min         lq       mean     median         uq        max
1       loop 179.432605 180.959035 183.923831 181.934548 184.685238 198.788213
2 vectorized   2.453153   2.503788   2.539548   2.550774   2.563320   2.609978
3      table   3.657651   3.695863   3.743140   3.739466   3.793853   3.829072
4     vapply  23.535640  23.679222  24.752840  25.133471  25.426109  25.578014
  neval
1    10
2    10
3    10
4    10
```

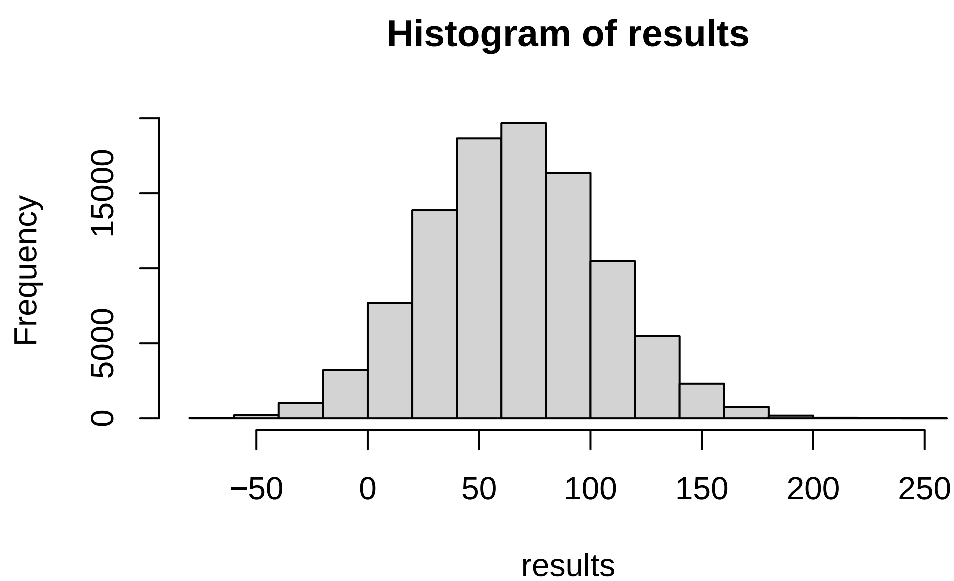
e) Do you think this is a fair game? Defend your decision with evidence based upon a Monte Carlo simulation.

```r
monteCarloDice <- function(nrolls,nsims){
  results <- vector()

  for(i in 1:nsims){
```

```r
    results[i] <- vectorizedDiceGame(nrolls)
  }
  return(results)
}

results <- monteCarloDice(100,100000)
hist(results)
```




```r
summary(results)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 -76.00   40.00   66.00   66.87   94.00  246.00 
```

You can add options to executable code like this

```r
2 * 2
```

```
[1] 4
```

The `echo: false` option disables the printing of code (only output is displayed).