# Post 02 - Simulating Games of Chance in R

*Nicholas Lai*

*December 3, 2017*

```r
library(holdem)
```

RStudio version 1.0.153

R version 3.4.1

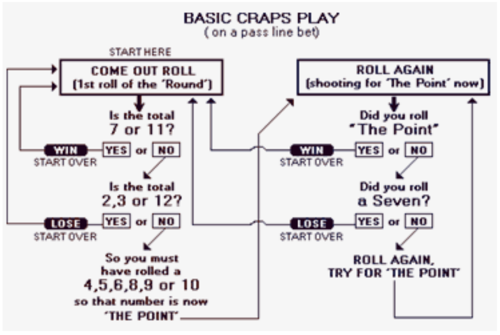
## Introduction

Many games of chance today have become so complicated that it is not always apparent how to solve them explicitly. Games can have hundreds of outcomes, thousands of inputs, and a whole bunch of other factors that make solving them generally a real pain.

Luckily, we don’t always have to. Games of chance can be simulated with computational tools like R, as setting up a game in R can be much simpler than trying to solve the game. Since in the long run, the results of the simulation will closely mirror the actual probabilites of game outcomes for a large number of simulations, the probabilities of those outcomes can approximately be known! This method of simulation is called “Monte Carlo”.

## Example: Craps

Craps is a game about rolling two dice. If the sum of the roll is a 7 or an 11, you win. If the sum of the roll is 2, 3, or 12, you lose. If the result is any other number, you roll the dice until you roll that number again (win) or you roll a seven (lose).



A reasonable thing to ask about this game is: how often do you win?

Craps is a simple example because the probability of winning is well-understood (and you can learn how to calculate it explicitly in Stat 134), but this is makes for a good example, as we can compare the observed and actual probabilities.

Our goal is to set up the game. We can do this using logical structures:

```r
#set a seed for the random number generator
set.seed(123456)
#vector of game results
games <- c()
#how namy games to simulate
n <- 10000

for (i in 1:n){
  #value of the first roll
  first_roll <- sample(1:6, 1) + sample (1:6, 1)
  #check if you lose on the first roll
  if (first_roll %in% c(2,3,12)){
    result <- F
  }
  #check if you win on the first roll
  else if (first_roll %in% c(7,11)){
    result <- T
  }
  #roll until either you hit 7 or the value of your first   roll
  else {
    roll <- sample(1:6, 1) + sample(1:6, 1)
    while (!(roll %in% c(7, first_roll))){
      roll <- sample(1:6, 1) + sample(1:6, 1)
    }
    if (roll == 7){
      result <- F
    }
    else{
      result <- T
    }
  }
  #add win/loss to result vector
  games[i] <- result
}
#Values in result vector
head(games)
```

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

```
#get proportion of victories (wins stored as 1, losses 0)
sum(games)/n
```

```
## [1] 0.4904
```

Through the above code, we get an experimental win percentage of `.4904` with 10000 trials. The actual probability of winning in a game of craps is `.4929` (https://www.youtube.com/watch?v=mzY-B2SQQaU), so our prediction is pretty close to the real probability. And we did so knowing very little about the underlying probability distribution, which may not be so useful for a game like craps, but can be useful for games with much more complex decision trees and other rules.

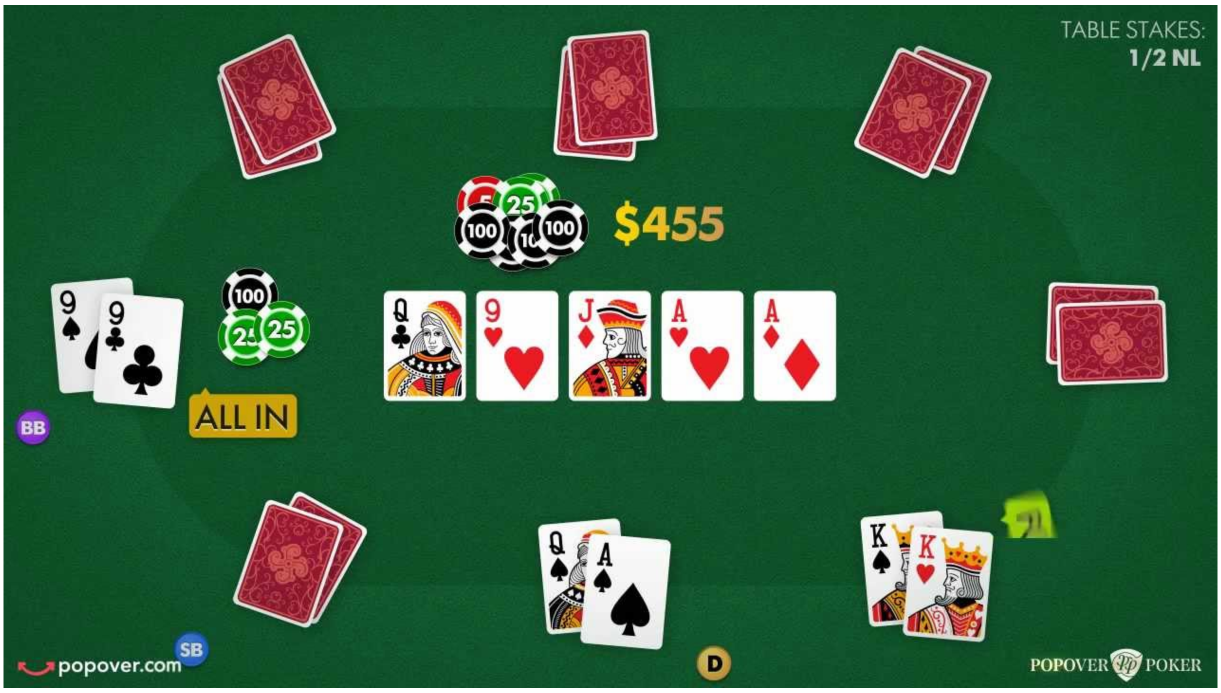
To reproduce this result, set your seed using `set.seed(123456)` and run the above code on your IDE.

Note: This is only one type of game you can play at a craps table. There are many other schemes that give you different odds of winning than this. Craps is famously the commonly run casino game that gives the player the best odds to win. If the best game to play still only has less than fair odds, then that should tell you something about gambling institutions.

## Example 2: Poker Hands and the holdem package

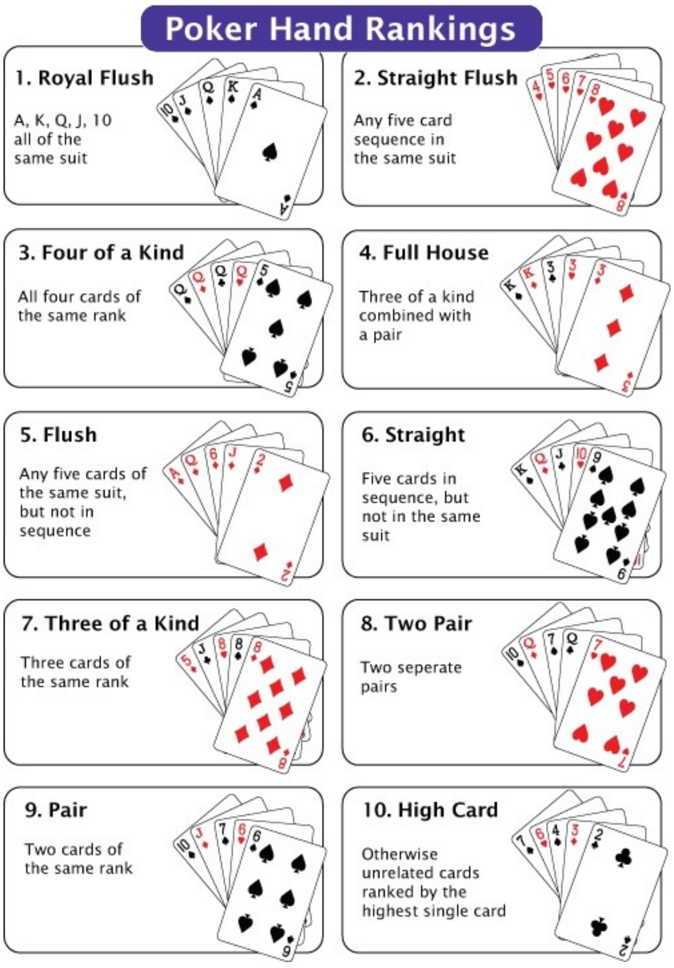
Some tools for simulating games of chance have useful packages already developed to lessen the burden of simulating work. One example is the `holdem` package for simulating poker games.

In a poker game, two cards are dealt to each player and five cards are on the board.



Typical Poker Board

Your hand is the best possible five-card hand that you can make with your two cards and the five on the board. The strength of the hands is as follows:


Poker Hand Rankings
1. Royal Flush
A, K, Q, J, 10 all of the same suit
2. Straight Flush
Any five card sequence in the same suit
3. Four of a Kind
All four cards of the same rank
4. Full House
Three of a kind combined with a pair
5. Flush
Any five cards of the same suit, but not in sequence
6. Straight
Five cards in sequence, but not in the same suit
7. Three of a Kind
Three cards of the same rank
8. Two Pair
Two seperate pairs
9. Pair
Two cards of the same rank
10. High Card
Otherwise unrelated cards ranked by the highest single card

Poker Hand Guide

Say you wanted to figure out the chance that both of two players in a poker game get a hand at least as powerful as a straight. Calculating this explicitly would be very annoying, but luckily we can simulate the game much more easily:

```r
# out of 10000 hands with 2 players, the proportion of times
# both players get a hand at least as powerful as a straight, if neither ever folds.
n = 10000

# vector of size n full of zeroes
result = rep(0,n)

# for loop to play 10000 hands
for(i in 1:n){
x1 = deal1(2) #deal hands to two players
b1 = handeval(c(x1$plnum1[1,],x1$brdnum1), c(x1$plsuit1[1,],x1$brdsuit1)) #values of the first player's hand
b2 = handeval(c(x1$plnum1[2,],x1$brdnum1), c(x1$plsuit1[2,],x1$brdsuit1)) #values of the second player's hand
if(min(b1,b2) > 4000000) result[i] = 1 #check if the strength of the hand is higher than a straight
}

#Calculate the proportion
sum(result>.5)/n
```

```
## [1] 0.021
```

As we can see, the chance of this happening is about two percent, experimentally.

To clarify the code above further, in the `holdem` package:

- the `deal1` function the number of players as an input, and deals a Texas hold'em poker game for that many players
- the `plnum1` object is the numerical values of the cards in the player's hands
- the `brdnum1` object is the numerical values of the cards on the board
- the `plsuit1` object is the suits of the cards in the player's hands
- the `brdsuit1` object is the suits of the cards on the board
- the `handeval` fuction takes in the above 4 values (with some fiddling) and return the power of the hand, which is a number in the millions. The higher the number, the stronger the hand in.

The `holdem` package defines a hand as least as powerful as a straight to have a `handeval` output higher than four million. For a more detailed explanation of the package, including the above objects, you can read the package documentation (https://cran.r-project.org/web/packages/holdem/holdem.pdf) which includes a similar example to the one above.

To reproduce the above code, load the `holdem` package into R using `library(holdem)` after installing the holdem package from CRAN, and run the above code using `set.seed(123456)`

## Conclusion and Takeaway

Often when dealing with complex games with lots of moving pieces, we will know how to set up and play the game on a computer, but we won't know the probabilities of certain events because they are hard to calculate using probabilities. Because of our above methods of simulating large numbers of trials of games, we can experimentally observe the chance of a certain event happening very easily.

Even though the above two examples (especially the first) may be simple enough to solve for explicitly, this method can be generalized for any game that you can get a computer to play (like monopoly: https://www.dropbox.com/s/f5xndyafuj2cmpn/monopoly-v1.py?dl=0), and is useful for quickly and easily obtaining chances.

## References

https://cran.r-project.org/web/packages/holdem/holdem.pdf - `holdem` package documentation

https://www.youtube.com/watch?v=mzY-B2SQQaU - Explicit solution for craps win probability

https://stackoverflow.com/questions/20624698/fixing-set-seed-for-an-entire-session

https://www.youtube.com/watch?v=BXRfDShFwXo - Craps simulation on R guide

https://stackoverflow.com/questions/13605271/reasons-for-using-the-set-seed-function

https://www.dropbox.com/s/f5xndyafuj2cmpn/monopoly-v1.py?dl=0 - Monopoly tile landing simulation

http://www.cardplayer.com/rules-of-poker/hand-rankings - Guide to poker hands