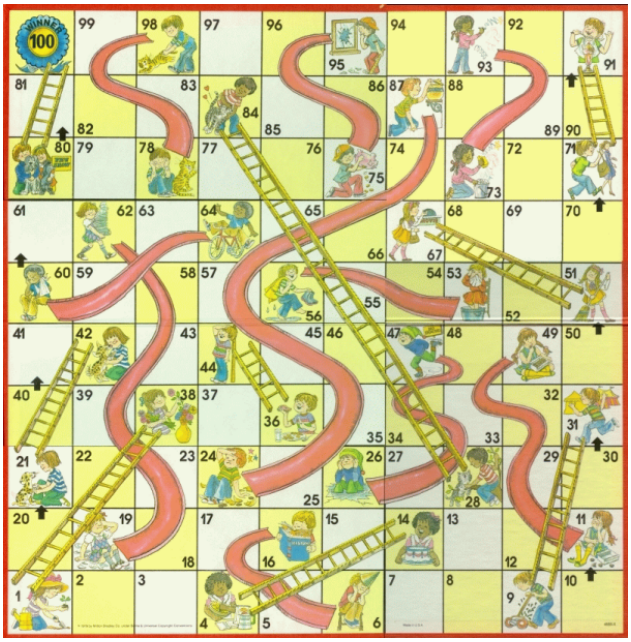
Chutes & Ladders Markov Chain Modeling

Colton Barger

4/2/2021

Chutes and Ladders (or for any Spongebob Squarepants fans, I guess you can call it Eels and Escalators) is a game all about luck. The goal of the game is to get to the end of the board by rolling a 6-sided die. If you happen to land on the base of a ladder, you move up to wherever the top of the ladder is. If you land on the top of a chute, you move back to the bottom of the chute. A picture of the game board is shown below. The first person to get to 100 (or 80) wins.

```
library(knitr)     # For knitting document and include_graphics function
img1_path <- 'chutes.png'
include_graphics(img1_path)
```



The playing of the game can be simulated via Markov chains through transition matrices. The main property of Markov chains is that any future outcome is entirely dependent on your present state. In other words, the past doesn't matter. Well, it matters a little bit... but the possible squares that you are able to land on in any given turn only depend on where you are currently at.

For example, let's say you're on square 26 on the above game board. You can roll a 6-sided die and get any numbers between 1 and 6 (inclusive). That mean you can land on squares 27, 28, 29, 30, 31, or 32; however, if you land on square 28, you venture up to square 84 because you landed at the base of a tall ladder. Thus if you start a turn on square 26, you can only end up on squares 27, 84, 29, 30, 31, or 32. How you got to square 26 does not matter at all at this point in the process.

To simulate the game by Markov chains, we need three different transition matrices. In the transition matrix P I've defined in the R code below, the number in the ith row and jth column represents the probability

of moving from square i to square j. The same holds true for any of the other matrices seen in the code chunk–D, C, or L. The matrix D represents the probability of landing on square j if you're standing on square i only by the roll of a 6-sided die. The matrices C and L represent the probabilities of moving from square i to square j through chutes and ladders, respectively. The numbers in the matrices C and L can only be either 0 or 1 because landing on the base of a ladder guarantees you'll climb to the top. Similarly, landing on the top of a slide guarantees you slide to the bottom of the slide.

The transition matrix P is obtained by multiplying together the three transition matrices D, C, and L. The order does not matter.

```
library(expm)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'expm'
```

```
## The following object is masked from 'package:Matrix':
##
##     expm
```

```
D <- matrix(0L, nrow=101, ncol=101)

for (i in 1:100){
  for (j in (i+1):(i+6)){
    if (j <= 100){
      D[i,j]=(1/6)
    }
  }
}
for (i in 1:6){
  D[101,i] = (1/6)
}
D[100,100] = 1
D[99,99]=5/6
D[98,98]=4/6
D[97,97]=3/6
D[96,96]=2/6
D[95,95]=1/6


C <- matrix(0L, nrow=101, ncol=101)
C[16,6] = 1
C[47,26] = 1
C[49,11] = 1
C[56,53] = 1
C[62,19] = 1
C[64,60] = 1
C[87,24] = 1
C[93,73] = 1
C[95,75] = 1
C[98,78] = 1
for (i in 1:101) {
  if (sum(C[i,]) == 0) {
    C[i,i] = 1
  }
}
```

```
L <- matrix(0L, nrow=101, ncol=101)
L[1,38] = 1
L[4,14] = 1
L[9,31] = 1
L[21,42] = 1
L[28,84] = 1
L[36,44] = 1
L[51,67] = 1
L[71,91] = 1
L[80,100] = 1
#only 9 ladders compared to 10 chutes, kind of mean
for (i in 1:101) {
  if (sum(L[i,]) == 0) {
    L[i,i] = 1
  }
}


P = D %*% C %*% L
```

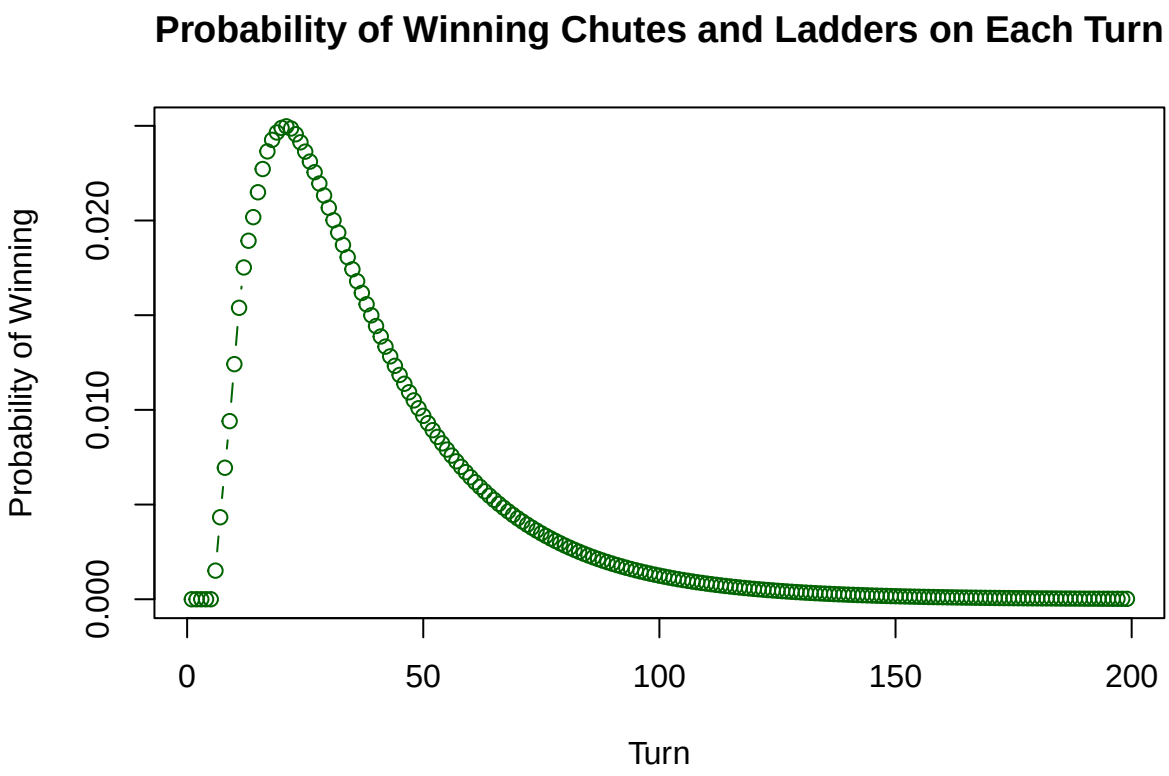
Something of interest might be to know the probability distribution of winning after each turn. This can be achieved by creating a starting vector of all zeros except for one 1 in the 101st index. We are using 101 as the index of the zeroth spot (i.e., you haven't done your first roll yet) so that each index corresponds to the correct number on the board.

To get the probability of getting to the end on turn i, we raise the transition matrix P to the ith power to artificially simulate i turns. The starting vector times this transition matrix after i turns will give us the vector telling us the probability of being in each square after i turns. This is done in the code below.

```
start <- rep(0L, 101)                #starting vector
start[101] = 1                       #starting off in the zeroth space
distribution <- rep(0L, 200)         #zero vector to store the probability vector in
for (i in 0:200) {
  P_n <- P %^% i                     #raising the transition matrix P to the ith power
  end_state <- start %*% P_n         #getting the probability of being in each square after i turns
  distribution[i] = end_state[100]   #grabbing the probability of being at the end

 prob_dist <- diff(distribution)
}

plot(prob_dist, main = 'Probability of Winning Chutes and Ladders on Each Turn',
     xlab = 'Turn',
     ylab = 'Probability of Winning',
     type = 'b',
     col='dark green')
```



Most Likely Game Length and Minimum Game Length

The most likely game length is given by the peak of the above plot. If we want to find this we simply need to find the index of the maximum value in the vector that I named prob_dist in the previous cell of code.

```
max_val = 0
for (i in 1:199){
  current_val = prob_dist[i]
  if(current_val > max_val){
    max_val = current_val
    most_likely = i
  }
}

most_likely
```

```
## [1] 21
```

```
prob_dist[most_likely]
```

```
## [1] 0.02497219
```

Thus the most likely game length is 22 rolls of the die, with probability 0.02497. I added one to the number 21 that the code output since the first "roll" isn't actually a roll, but rather the starting state.

The minimum game length is the index of the first entry in the vector prob_dist that is non-zero.

```
first = 0
for (i in 1:200) {
```

```
  if (prob_dist[i] > 0){
    first = i
    break
  }
}
prob_dist[first]
```

```
## [1] 0.00151106
```

```
first
```

```
## [1] 6
```

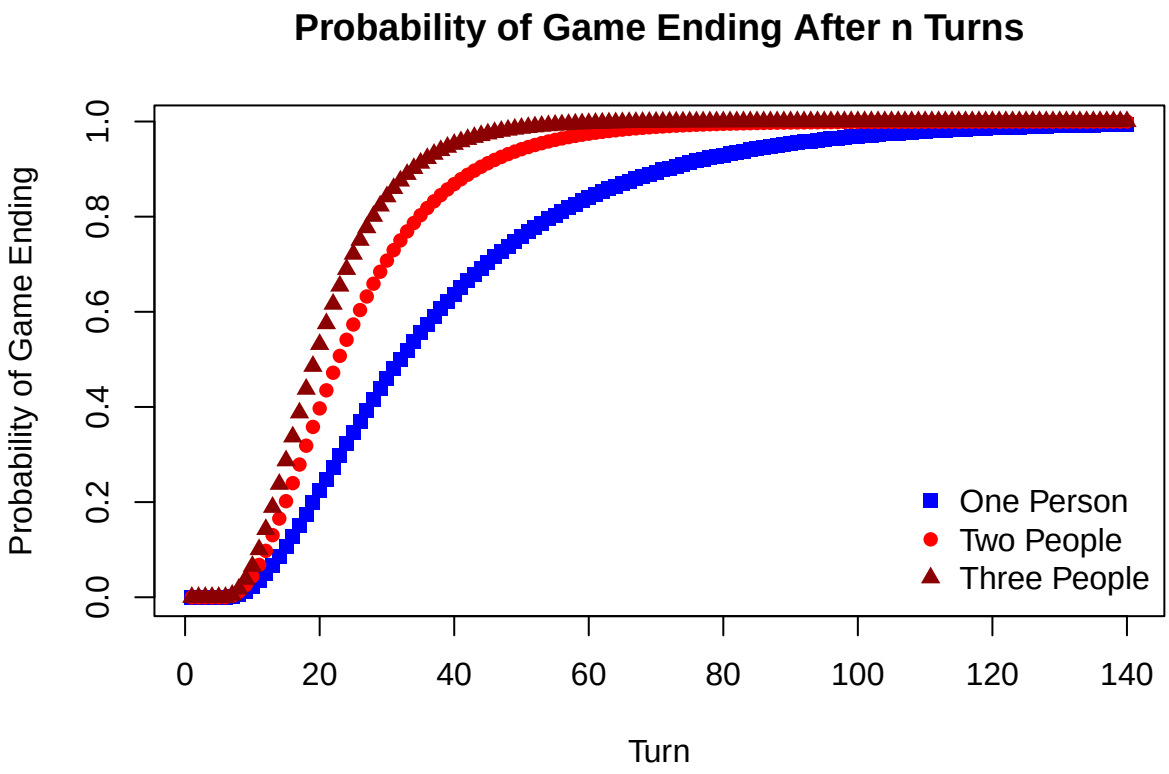
So according to the above code the minimum game length is achieved in 7 rolls of the die with a probability of 0.0015.

There are a few ways to achieve the shortest game, but one combination of rolls is (4, 6, 6, 2, 6, 6, 4). This puts you on square 100 exactly with the route 0->14->20->26->84->90->96->100

Probability of Game Ending After n Turns

Using independence, the probability that the game will end after n turns with 3 people is the probability that the game will end after at least one of the three people reaches the end. A well-known technique in probability is that the probability of "at least one" is equal to one minus the probability that none of the games end after n turns. The probability that none of the three games end is one minus the probability that each individual game will end, cubed. This can be seen in the code chunk below. Shown below is a graph of the probability of the game ending for 1, 2, and 3 people after n turns.

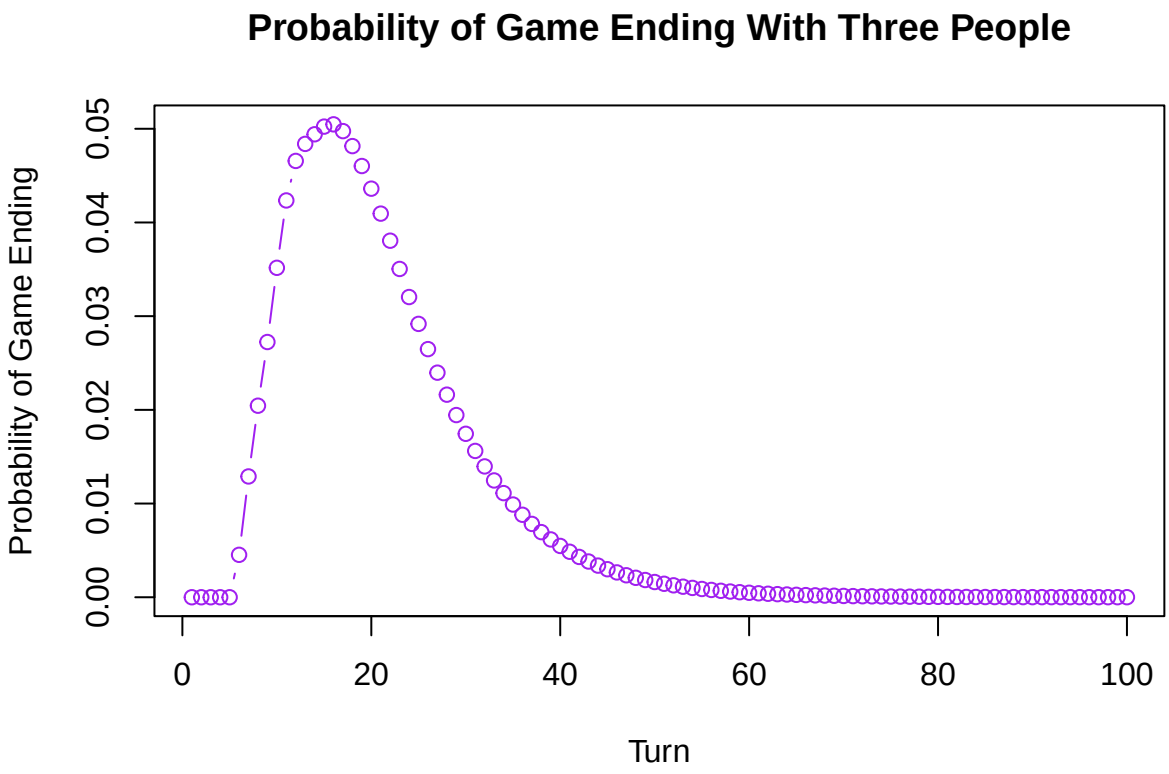
```
distribution2 <- 1-(1-distribution)^2
distribution3 <- 1-(1-distribution)^3
plot(distribution[1:140], pch=15, col="blue",
     xlab = 'Turn',
     ylab = 'Probability of Game Ending',
     main = 'Probability of Game Ending After n Turns')
points(distribution2[1:140], pch=16, col="red")
points(distribution3[1:140], pch=17, col="dark red")
legend('bottomright',
       legend=c('One Person', 'Two People', 'Three People'),
       col = c('blue', 'red', 'dark red'),
       pch = c(15,16,17),
       bty="n")
```



As expected, the game should end more quickly the more people we add to the game. Note that we're talking about number of turns here and not time. The amount of tiem in which you finish this game is completely dependent on the group of friends you are playing with. If it were my group of friends, we would add crazy conditions such as "play a game of jenga for every time you land on a chute or ladder." I'm getting off-topic, but you get the point.

The distribution of the game's length with three people is given by the plot below:

```
prob_dist3 <- diff(distribution3)
plot(prob_dist3[1:100],
     xlab = 'Turn',
     ylab = 'Probability of Game Ending',
     main = 'Probability of Game Ending With Three People',
     col = 'purple',
     type = 'b')
```



The most likely length of the game shifts closer to 0 (or should I say 7).

The most likely overall game length is calculated in a way similar to the game with one person.

```
max_val = 0
for (i in 1:199){
  current_val = prob_dist3[i]
  if(current_val > max_val){
    max_val = current_val
    most_likely = i
  }
}

most_likely
```

```
## [1] 16
```

```
prob_dist[most_likely]
```

```
## [1] 0.02271835
```

So the game will most likely end after 17 turns with a probability of 0.0227, or 2.27%