Math IA - A Mathematical Exploration

As a child I spent countless hours playing multiple board games, amongst these was Snakes and Ladders. I was always curious about how long a single game could take; sometimes games would finish in a few turns, sometimes they would take long periods of time. I realized that an infinitely long game was also possible. I wondered questions such as, how long does an average game last, whats the probability of a game lasting a certain amount of turns, etc. This investigation intends to model a game of Snakes and Ladders, so that certain probabilities and statistics can be ascertained. I chose Snakes and Ladders rather than another board game, because player interaction is limited, which simplifies the game, and I also personally enjoyed the game as a child.

Snakes and Ladders is usually played on a gameboard which consists of a 10 x 10 grid. The grid is numbered from 1 to 100, from the bottom left corner to the top left corner in a zig-zag fashion. At many different locations on the board there are snakes and ladders. These snakes and ladders connect certain pairs of squares together. Landing on a snake or ladder would transport players to a smaller or larger square, respectively. Visuals below show the board, as well as the locations of the aforementioned snakes and ladders. The positions of snakes and ladders are based on the Milton Bradley version of the game.

Game board

100	99	98	97	96	95	94	93	92	91
81	82	83	84	85	86	87	88	89	90
80	79	78	77	76	75	74	73	72	71
61	62	63	64	65	66	67	68	69	70
60	59	58	57	56	55	54	53	52	51
41	42	43	44	45	46	47	48	49	50
40	39	38	37	36	35	34	33	32	31
21	22	23	24	25	26	27	28	29	30
20	19	18	17	16	15	14	13	12	11
1	2	3	4	5	6	7	8	9	10

Snakes (From, to)

98	78
95	75
93	73
87	24
64	60
62	19
56	53
49	11
48	26
16	6

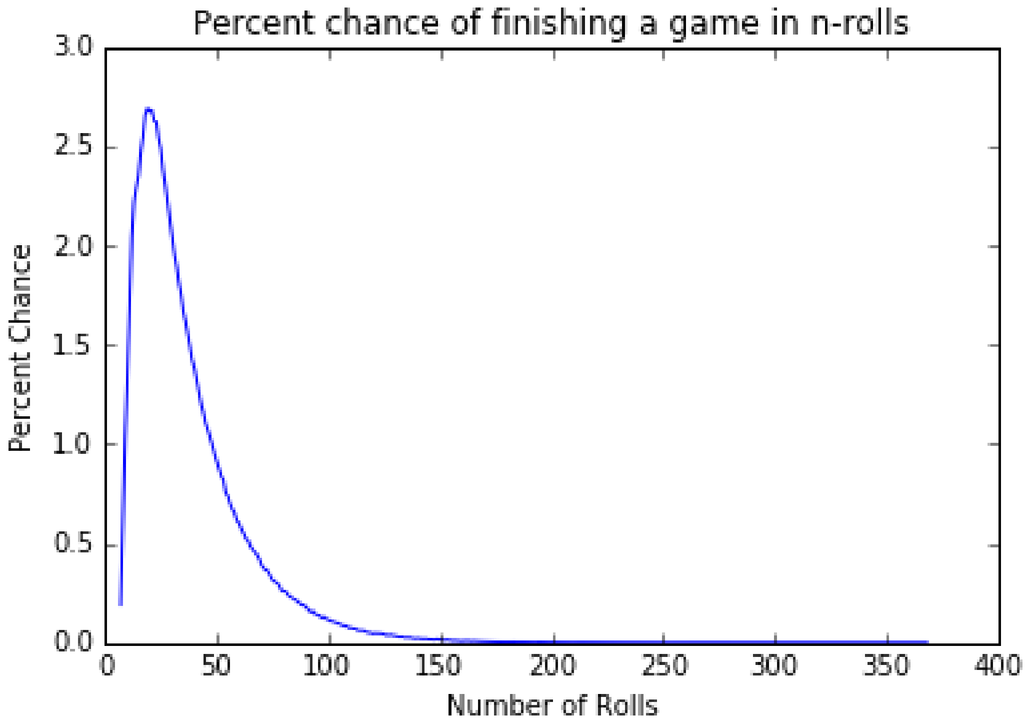
Ladders (From, to)

1	38
4	14
9	31
21	42
28	84
36	44
51	67
71	91
80	100

When starting the game, all players start from off of the board, which can also be said to be "square-0". They then roll a die, which is numbered 1-6, and move some spaces accordingly. If a player lands on a snake, then they are slided down the snake to a square with a lower value. Whereas if a player lands on a ladder, they climb up the ladder and are transported to a square with a higher value. Landing at the head of a ladder or the tail of a snake yields no change in position or other results. The winner of the game is the first player to reach square-100, hence a game is over when the position of a player is square-100.

Due to the way the snakes and the ladders are distributed over the game board, there is no limit to how long a game could theoretically last. A player could repeatedly land on snakes, which would send them backwards repeatedly. However, most of the games that I played ended fairly quickly. I am interested in knowing how many turns an average game consists of. To find the length of an average game, I created a Monte Carlo simulation by writing a loop in Python. The Python code for the simulation can be found in the Appendix. A Monte Carlo simulation is when we model a system and then test it with random inputs to determine some value, as well as see the distribution of the outcomes.

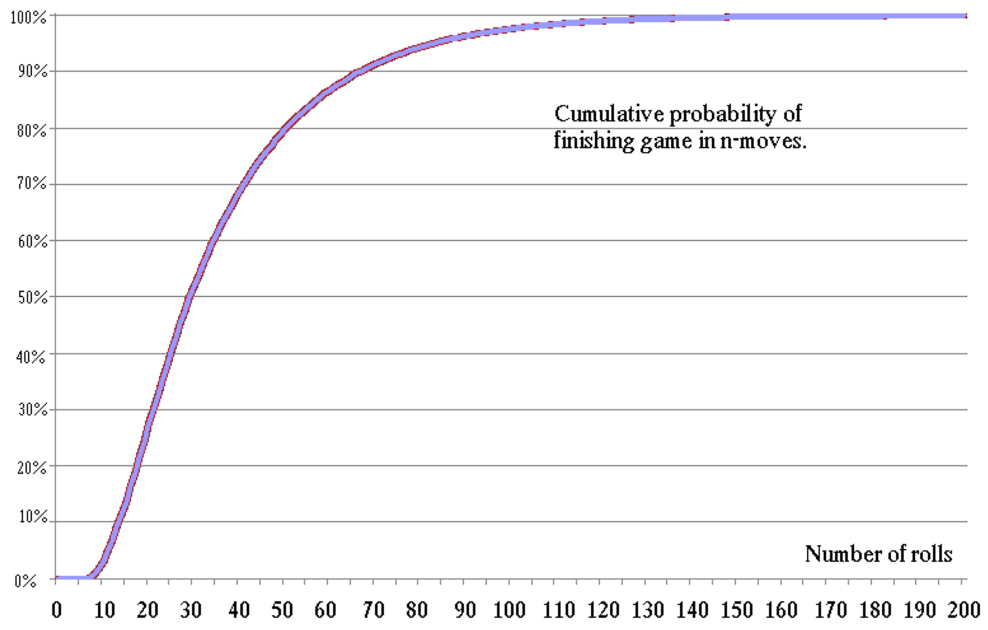
To create the Monte Carlo simulation for Snakes and Ladders, I created a loop that I could repeat multiple times. The gist of it is that a die is rolled by generating a random number between 1 and 6. A variable (x) holds the randomly generated value, and then checks if there are any snakes or ladder for that value. If there is a snake or ladder, then the value of x is changed to the corrseponding value. Every turn the rolls of the die are added up, so that the position after two rolls would be the first roll plus the second roll. Another variable is used as a counter, so that everytime a die is rolled, the value of the counter increases by 1. As a part of defensive coding I set the limit for the maximum amount of turns at 10,000, so that I would not get stuck in an infinite loop. However the longest game only took 402 turns. The game is over when x reaches 100. The function returns how many turns a game takes to complete. The number of rolls, as well their corresponding frequency is stored in a dictionary. I repeated this loop 1,000,000 times and created a graph to represent the results.



In the graph shown above, the x-axis shows the number of turns, and the y-axis shows the percent of games that were completed in that many number of turns.

Using the graph above, I was able extract certain data. The least amount of turns needed to win was found, which was 7. In fact, the graph starts from 7 as there is no possible way to win in less than 7 turns. The probability that a game will be completed in 7 turns is 0.02 or 2%. The number of turns needed to complete a game that came up most frequently in the simulation was 20, which is the mode. We can see this is the peak on the graph above. The probability of a game lasting 20 turns is 0.02682. We can actually find the probability that a game will last t turns from this graph, where $n \in \mathbb{N}$. We can also see from the graph that as the number of rolls reach large values, the probabilities become asymptotically small. Thus we can see that very long games are highly unlikely.

I extracted the mode from this graph, as well as some probabilities, however I still wish to find the average game length, or the mode. To do this, I created a cumulative probability graph.



The cumulative probability graph shown above shows the probability of a game ending in n-turns or less. Using the cumulative probability graph I found the median and the mode. The median number of turns in a game is 29. This can be found by looking at the corresponding x-value when y is 50%. This indicates that there are as many games that are longer than 29 turns as there are games that are less than 29 turns. The mean or average length of a game is the sum of all the rolls divided by the number of games played. I simulated 1 million games, and the die was rolled 37,034,085 times. $\frac{37034085}{1000000} = 37.03$. Since the die was rolled 37.03 times on average every game, that is the mean length of a game.

I obtained some values, such as the mean, median, and mode by using a Monte Carlo simulation. However, I believe that these values possess some uncertainty to them, there is too much randomness involved in this procedure. I think this because I obtain slightly different results when I run my code again. Also, I am not entirely confident in the randomness of the die within the simulation. There must be a better and more accurate method of modelling a game of Snakes and Ladders, and hence finding the length of an average game. Creating a Monte Carlo simulation was a very useful objective approach. However, I now wish to explore Snakes and Ladders from a subjective or Bayesian approach.

Using a Markov Chain to analyse a game of Snakes and Ladders would form a solid subjective approach to Snakes and Ladders. Markov Chains are used to model situations with discrete and different states with defined probabilities of moving between individual states. At any point in the game the probabilities of future events are entirely independent of past events. If a player is on square 32, then the probability of the next move is independent of how the player reached square 32. This is exactly what makes Snakes and Ladders suitable for analysis through the use of Markov Chains.

Given that a player has reached square N, there are 6 different outcomes that could occur. All of these outcomes possess equal probabilities of $\frac{1}{6}$. Obviously the probability of going to N is 0, given that there are no snakes which can take the player back to N.

N	N+1	N+2	N+3	N+4	N+5	N+6

The probabilities can also be displayed in a matrix. The first row shows the probabilites of moving to other spaces from square 6, and the second row shows the probabilities for square 7.

$$\begin{array}{ccccccccc} 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ \end{array}$$
$$\begin{bmatrix} 0 & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 \\ 0 & 0 & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{bmatrix}$$

To create a better model we can add in snakes and ladders to the matrix, which is shown below. If a player lands on a space that contains a snake or a ladder, they will be transported to another square. For example, if a player is on square 18, there are 6 different outcomes, which all still have the probability of $\frac{1}{6}$.

19	20	42	22	23	24

As we can see there is a ladder on square 21 which takes the player to square 42.

We can create another matrix, this time displaying the probabilties of the entire board, which would also account for snakes and ladders. However, there are some things that we must first account for. Firstly, there can be more than one way to reach a square. We can see that there are 2 ways to reach square 53. This is because a player could either roll a 3 and travel to square 53, or they could roll a 6 and they would end up at 53 via a snake. Therefore the probability of going to square 53 when a player is at square 50 is $\frac{2}{6}$, and not $\frac{1}{6}$. The matrix below shows the different outcomes and corresponding probabilites when a player is on square 50. This matrix is only one row of a large matrix that I will create later. We can call this part of the matrix row 50.

$$\begin{array}{cccccccccccccccccccc}
48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 & 64 & 65 & 66 & 67 \\
\left[0\right. & 0 & 0 & 0 & \frac{1}{6} & \frac{2}{6} & \frac{1}{6} & \frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \left.\frac{1}{6}\right]
\end{array}$$

There is also something else that we must account for, the winning condition. To win, a player must reach square 100. However, a player can be on square 99 and roll a 5 and they would win. It is not neccesary to roll a 1 on square 99 to win. An exact roll is not required, therefore there are multiple ways to reach square 100.

Now that we have accounted for these things, we can create a matrix to represent the probabilites and outcomes for the entire board. We should end up with a 101 x 101 matrix that would be mostly filled with values of zero. Without the addition of snakes and ladders, there would simply be six consecutive values of $\frac{1}{6}$ in every row. It would be 101 x 101, and not 100 x 100 since we would also include square 0. Square 0, as mentioned previously, is where players start off from. However, we don't actually need a 101 x 101 matrix. We can represent the entire board with only an 82 x 82 matrix. This is because the heads of snakes and bottoms of ladders don't count as squares. There are 10 snakes and 9 ladders, which is why the matrix is 82 x 82.

$101 - (10 + 9) = 82$

It is impossible for a player to actually stop on a snake or ladder. Including them in the matrix would be redundant and useless (Their probabilites would just be 0). In the matrix, the entries in row-i and column-j represent the probability that a player on square-i will move to square-j in the next turn. Hence, the matrix represents the probability of moving from any square to any other square.

$$\begin{bmatrix} 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \cdots \\ 0 & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \cdots \\ 0 & 0 & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Shown above is a snipper of the 82 x 82 matrix which I shall refer to as M. The complete matrix M is shown in the Appendix.

I created another matrix, which I shall call L. L is simply a column vector which shows the position of the player after some amount of turns. We can say that L represents the state of a player. The position of the player when the game starts is off the board, and therefore the probability that the player will be in square 0 is 1.0. Hence, the column vector L, which will also be referred to as our State vector, will have a value of zero for every row, except row-0.(There are 82 rows to represent every possible position.) L is shown below

$$\begin{bmatrix} 1.0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

I then multiplied the column vector L with the matrix M. I did this by using some Python code, which is shown in the Appendix. The vector produced from the matrix multiplication is the probability distribution of the player's position at the end of the first turn. The value in each row of the resulting vector is the probability that the player will be on that square.

After one turn, there will be values of $\frac{1}{6}$ in rows 2, 3, 5, 6, 14, and 38. The rest of the column vectors will still be filled with zero. The values in the rows represent a probability distribution of where a player will be after one turn.

$$\text{Thus } M \times L = \begin{bmatrix} 0 \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \vdots \\ 0 \end{bmatrix}$$

Recall that certain squares or rows were removed, since they contained snakes and ladders. Adding them to the matrix M would have been redundant. Due to this, the first value in the column vector L represents the probability of the player being at square 0, which is why the probability is 0.

We have found the probability distribution after the first turn. However, we can also easily find the probability distribution after any n-turns. The output of the first turn would be the input of the second turn. We would simply take the product of L and M, and multiply it by our matrix M again. This would give us the probability distribution for the second turn. The probability distribution of the position of a player can said to be it's current state.

Thus, $M \times L \times M = State_2$

$M^2 \times L = State_2$

Furthermore, we can generalize, and create a general equation to give us the state of a player after n-turns.

$M^n \times L = State_n$

I have successfully created a Markov Chain that can accurately model a game of Snakes and Ladders. Now I can finally find the average length of a game.

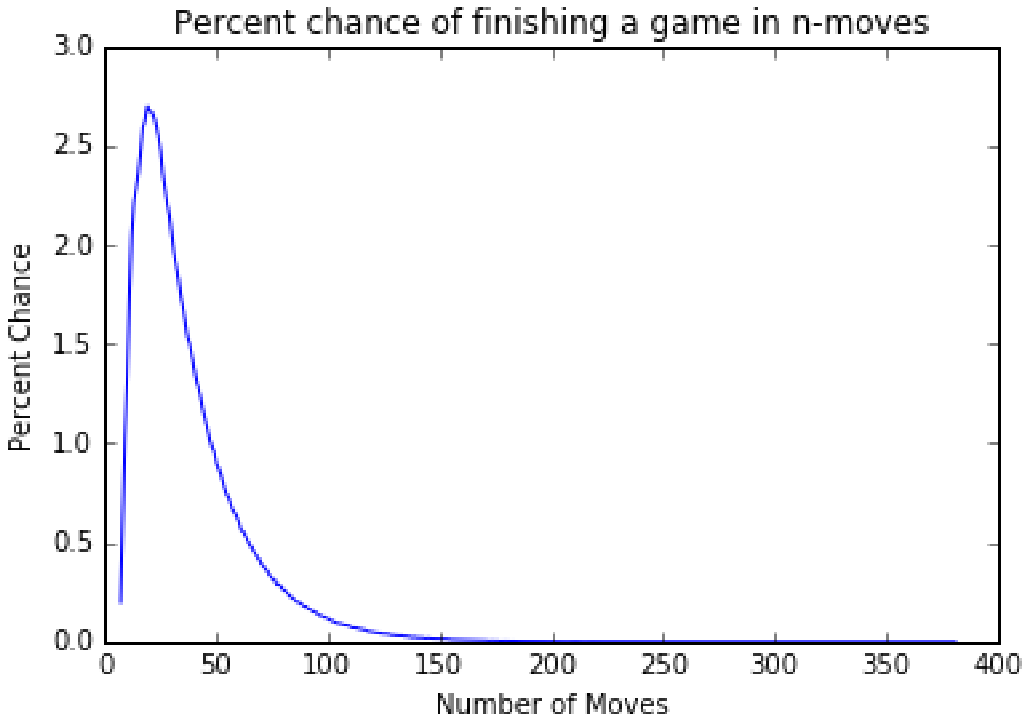
If we look at the State vector of a player after 7 turns, we can see that is the first time the probability of the player being on square 100 is not 0, there is a value in row-100 in our column vector. This value is the probability that a game will end in 7 turns.

$M^7 \times L = State_7$

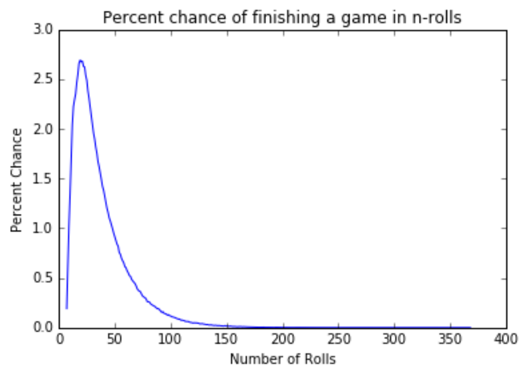
The value in row-100 of this State vector is 0.02, this is the same value that we obtained from the Monte Carlo simulation.

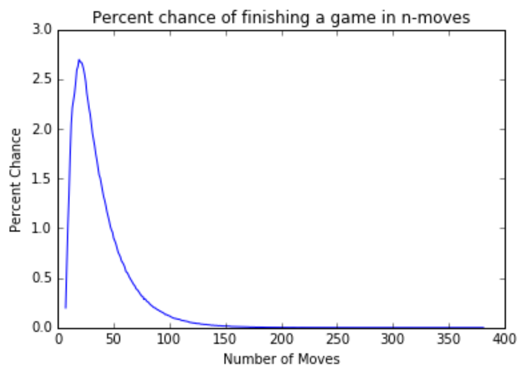
The probability of a game ending in n-turns is the value in row-100 of the State vector, which is the product of $M^n \times L$. If we graph the value of row-100 as we increase n, then we can obtain a probability distribtion. The value in row-100 would be the probability that a game has ended in n-moves.

Here is a graph of the probability distribution:



The probability distrubtion graph obtained through Markov Chains looks nearly identical to the graph that I produced from the Monte Carlo simulation. I have displayed them both below so that they can be compared.





We can see that the two graphs are nearly identical, hence they would give us the same values for median, mode, and mean that we obtained through the Monte Carlo Simulation. The table below displays these statistic values.

Mean	37.03
Mode	20
Median	29

Also, due to the similarity between the two graphs, we can conclude that the two approaches to the modelling of Snakes and Ladders were both correct. Using a subjective method and an objective method both produced the same results in the analysis of the probabilties and statistics of Snakes and Ladders.

Bibiography

Snakes and Ladders. Digital image. Gioco Dell'Oca. N.p., n.d. Web. 25 Oct. 2016. ¡http://www.giochidelloca.it/images/s/snakes1115a.jpg¿.

Appendix

Python Code:

```
#Umar Ahmed - Math IA - Python 3.5.2 -Written in Anaconda
import random
import collections
import matplotlib
z = [ ]


snakes = {98:78,95:75,93:73,87:24,64:60,62:10,56:53,49:11,48:26,16:6}
ladders = {80:100, 71:91,28:84, 51:67, 21:42,36:44,9:31,4:14, 1:38}


def rollDice(): #This rolls a dice for the player
        roll = random.randint(1,6)
        return roll



def checkSL(n): #Checks if a certain space has a S/L
        if n in ladders:
        n = ladders[n]
        elif n in snakes:
        n = snakes[n]
        return n


def play():
        x = 0
        y = 0
        a = 0
        while x<100 and y<1000:
                x += rollDice()
                a = a +1
                x = checkSL(x)
                y = y+1
        z.append(y)
        b.append(a)

b = []

while len(z)<2000000:
        play()


z.sort()
counter = collections.Counter(z)
# Counter({n-rolls:freq})

for key in counter:
        counter[key] /= 20000.0

print(counter)
print(sum(b))
w = counter.items()
import matplotlib.pyplot as plt
plt.title('Percent chance of finishing a game in n-moves')
plt.xlabel('Number of Moves')
plt.ylabel('Percent Chance')
plt.plot(*zip(*w))
plt.show() \newline



#Matrix Multiplication - Umar Ahmed - Math IA - Anaconda
import numpy as np
M = matrix.np("0,0.1666,0.1666,0.1666,0.1666,0,0,0,0,0,0,0,0,0,0,0,0,0,
L = matrix.np("1:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:
c = A*L
print(c)
```