SNAKES AND LADDERS

ADAM SIERAKOWSKI

Abstract. This document discusses following claim on Wikipedia about Snakes and Ladders:

"A player will need an average of 39.2 spins to move from the starting point, which is off the board, to square 100."

1. Rules of the game

The game is played on a 10×10 board with 100 squares with numbers 1 to 100. Starting of board (in state 0), players in turn roll a six-faces dice to determine how many squares each player moves forward. The player that first reached exactly 100 wins. If a player lands on $16, 47, 49, 56, 62, 64, 87, 93, 95$ or 98 respectively he/she is moved back to $6, 26, 11, 53, 19, 60, 24, 73, 75$ or 78 respectively (along a snake). If a player lands on $1, 4, 9, 21, 28, 36, 51, 71$ or 80 respectively he/she is moved forward to $38, 14, 31, 42, 84,\ 44, 67, 91$ and 100 respectively (along a ladder). If a move takes a player beyond the last square 100, the players stays put and the turn is lost. For example, a player rolling 16 times a six and then a four would reach 100 in 17 turns via squares $6, 12, \ldots, 90, 96, 100$.

2. A bit history

The most widely known edition of snakes and ladders was called Chutes and Ladders and was released by Milton Bradley in 1943. This is also the version described above and illustrated on Figure 1.

3. Maths behind the game

Theorem 1. *On average a game with a single player would take approximately 39.2 turns. Moreover, the proof indicates how this average can be computed exactly (as a suitable integer fraction).*

Proof. Let us call all the positions a player can be in before position 100 for *states*, so state 0 to 99 corresponding to the position of bord to the square 99 on the bord. Let Q be the 100×100 matrix, where $Q_{i,j}$ $(i, j = 0, \ldots, 99)$ is the probability to go state i to state j in exactly one step. For example from state 0 we can reach state $2, 3, 5, 6, 14, 38$ with probability $\frac{1}{6}$ each (as 1 and 4 is a ladder), so

$$Q_{0,j} = \frac{1}{6} \text{ for } j = 2, 3, 5, 6, 14, 38, \quad \text{and } Q_{0,j} = 0 \text{ otherwise,}$$

FIGURE 1. The game released by Milton Bradley in 1943.

and from state 96 we can reach state $78, 96, 97, 99$ with probability $\frac{1}{6}, \frac{2}{6}, \frac{1}{6}, \frac{1}{6}$ respectively (as rolling 5 or 6 keeps the player on 96), so

$$Q_{96,j} = \frac{1}{6} \text{ for } j = 97, 78, 99, \quad Q_{96,96} = \frac{2}{6}, \quad \text{and } Q_{96,j} = 0 \text{ otherwise.}$$

Let Q^k be the matrix Q multiplied by itself k times.

Fact 1: The probability of going from state i to state j in precisely k steps is $(Q^k)_{i,j}$, i.e., the i,j-th entry of Q^k.

Proof of Fact 1: The probability to go from state i to state j in exactly $k = 2$ steps is the sum of the probability $Q_{i,0}Q_{0,j}$ of going from state i to state 0, then from state 0 to state j, the probability $Q_{i,1}Q_{1,j}$ of going from state i to state 1, then from state 1 to state j, and so on. But this sum $\sum_{l=0}^{99} Q_{i,l}Q_{l,j}$ is exactly the i,j-th entry of Q^2. A similar argument applies for $k > 2$, for example $Q_{i,n}Q_{n,m}Q_{m,j}$ is the probability to go from state i to j via state n and then state m, so

$$\sum_{n,m=0}^{99} Q_{i,n}Q_{n,m}Q_{n,i} = (Q^3)_{i,j}$$

is the probability to go from state i to state j in 3 steps.

Fact 2: The matrix entries of Q^k uniformly tend to 0 as k goes to infinity, formally $\lim_{k\to\infty} \max_{i,j}(Q^k)_{i,j} = 0$.

Proof of Fact 2: Recall that one can finish the game in 17 steps via squares $6, 12, \ldots, 90, 96, 100$. Let $N = 18$. Now, starting in any state i there is a way to finish (reach 100) in N or less steps. With p_i being

the probability of *not* finishing in N steps, starting in state i, we get $p_i < 1$. Let $p = \max_{i=0,\dots,99} p_i$. By construction (and Fact 1),

- $p < 1$,
- the probability of not finishing (start at i) in N steps is $= p_i$,
- the probability of not finishing (any start) in N steps is $\leq p$,
- the probability of not finishing (any start) in $2N$ steps is $\leq p^2$,
- and this translates to: For any start i, $\sum_{j=0}^{99}(Q^{2N})_{i,j} \leq p^2$.

We finally obtain $\max_{i,j}(Q^{2N})_{i,j} \leq p^2$, $\max_{i,j}(Q^{3N})_{i,j} \leq p^3$ and so on. It now follows that $\lim_{k\to\infty} \max_{i,j}(Q^k)_{i,j} = 0$ as claimed.

Fact 3: The matrix $N := (I - Q)^{-1}$ exists.

Proof of Fact 3: Suppose $(I - Q)x = 0$ has a solution vector x. Then $x = Qx$ and $x = Q^k x$ for all $k = 1, 2, \dots$. But, by Fact 2, we know the entries of Q^k gets smaller and smaller as k goest to infinity. Hence the right hand side of $x_i = (Q^k x)_i = \sum_{l=1}^{99}(Q^k)_{i,l}x_l$ can be made arbitrary small by taking k sufficiently large. Thus, the only solution is $x = 0$. It follows $I - Q$ is invertible.

Fact 4: With the convention $Q^0 := I$ we have $N = \sum_{k=0}^{\infty} Q^k$.

Proof of Fact 4: For any positive integer n we have

$$\begin{aligned} I + Q + \cdots + Q^n &= (I - Q)^{-1}(I - Q)(I + Q + \cdots + Q^n) \\ &= (I - Q)^{-1}(I - Q^{n+1}) \end{aligned}$$

But since the matrix entries of Q^{n+1} uniformly tend to 0 as n goes to infinity, see Fact 2, we get $I + Q + Q^2 + \cdots = (I - Q)^{-1}(I - 0) = N$.

Fact 5: Let $N_{i,j}$ denote the i, j-th entry of N. Then $N_{i,j}$ is the expected number of times a game starting in state i visits state j.

Proof of Fact 5: Fix i, j. Let P be the expected number of times a game starting in state i visits state j. Let P_k be the expected number of times a game starting in state i visits state j after exactly k steps. Indexing over k, $P = \sum_{k=0}^{\infty} P_k$. Comparing

$P_1 =$ the expected number of times a game starting in state i visits state j after exactly 1 step

$Q_{i,j} =$ the probability to go state i to state j in exactly 1 step

we observe $P_1 = Q_{i,j}$. Similarly (again $Q^0 = I$),

$$P_0 = I_{i,j} = (Q^0)_{i,j}, \quad \text{and} \quad P_k = (Q^k)_{i,j}, \quad \text{for } k > 0.$$

If follows that $P = \sum_{k=0}^{\infty}(Q^k)_{i,j} = (\sum_{k=0}^{\infty} Q^k)_{i,j} = N_{i,j}$.

Fact 6: On average a game with a single player would take approximately 39.2 turns.

Proof of Fact 6: $N_{0,j}$ is the expected number of times a game (starting of board) visits state j. This contribution, giving how many times we get to state j, counts exactly how many times (on average) we must roll the dice to end in state j. Counting how many times we need to roll

(on average) to get to all the states, i.e., $\sum_{j=0}^{99} N_{0,j}$, gives the average game length. Since the inverse of an invertible integer matrix can be computed exactly in terms of (integer) fractions, the average game length can also be computed exactly as a certain (integer) fraction. Using a bit of coding this number can be computed to be approximately 39.2 (the code is in Listing 1). □

4. The Code

Here I have included Python code (approximately) computing the length of the game using the proof of Theorem 1. Recall,

$$\text{the average length of one game} = \sum_{j=0}^{99} N_{0,j}$$

This is precisely what is computed in method 2, the sum of all the entries in the first row of matrix N. This sum the first entry of the 100×1 vector $x := N\mathbf{1}$, where $\mathbf{1}$ denotes the 100×1 vector having all entries equal 1 (and N is as above). Rearranging, we get x can be found by solving $(Q - I)x = \mathbf{1}$, this is precisely what is computed in method 1. Finally, recall that $N = \sum_{k=0}^{\infty} Q^k$, so a good approximation of the game length the sum of all the entries in the first row of the matrix $N_{\text{apprx}} = \sum_{k=0}^{999} Q^k$, this is precisely what is computed in method 3.

Listing 1. Code for getting the 39.2.

```
from numpy.linalg import matrix_power
import numpy as np


def snakes_and_ladders_matrix():
    n = 101
    T = np.zeros((n, n))
    for i in range(n - 1):
        for j in range(i + 1, np.min([n, i + 7])):
            T[i, j] = 1 / 6
    for i in range(n - 6, n):
        T[i, i] = (i - n + 7) / 6

    snakes_and_ladders = np.array([
        [98, 78], [95, 75], [93, 73], [87, 24], [64, 60],
        [62, 19], [56, 53], [49, 11], [47, 26], [16, 6],
        [1, 38], [4, 14], [9, 31], [21, 42], [28, 84],
        [36, 44], [51, 67], [71, 91], [80, 100]])

    for src, dst in snakes_and_ladders:
        b = T[:, src].copy()
        T[:, dst] += b
        T[:, src] = 0
    return T
```

```
def main():
    T = snakes_and_ladders_matrix()
    Q = T[:-1, :-1]
    I = np.identity(100)
    o = np.ones((100, 1))

    # method 1
    result = np.linalg.solve(I - Q, o)
    print("via solve", result[0, 0])

    # method 2
    N = np.linalg.inv(I - Q)
    sum = np.sum(N, axis=1)
    print("via inver", sum[0])

    # method 3
    Napprx = np.zeros((100, 100))
    for k in range(1000):
        Napprx += matrix_power(Q, k)
    sumapprx = np.sum(Napprx, axis=1)
    print("via apprx", sumapprx[0])


main()
```

OUTPUT:
via solve 39.225122308234916
via inver 39.22512230823491
via apprx 39.225122308234866