FiveThirtyEight's February 19, 2021 Riddler

Emma Knight

February 21, 2021

This week's riddler is a take on the game of Jenga:

Question 1. *In the game of Jenga, you build a tower and then remove its blocks, one at a time, until the tower collapses. But in Riddler Jenga, you start with one block and then place more blocks on top of it, one at a time.*

All the blocks have the same alignment (e.g., east-west). Importantly, whenever you place a block, its center is picked randomly along the block directly beneath it.

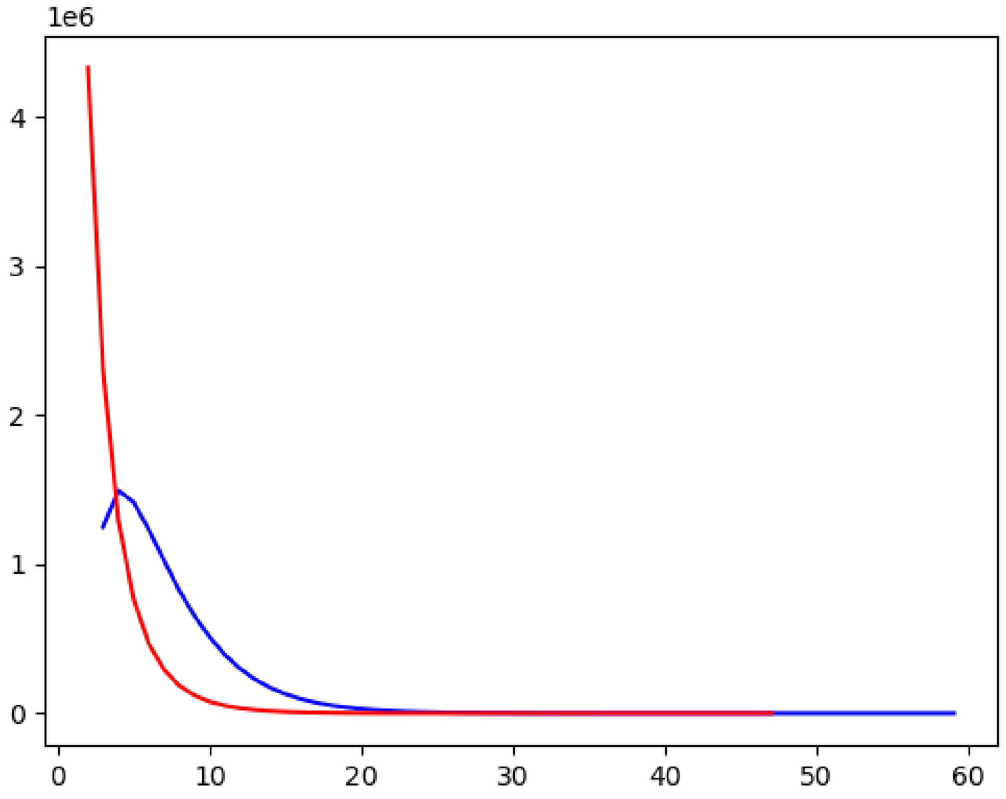
How many blocks, on average, are placed when the tower collapses?

For this problem, I will assume that the blocks have a width of 2. Let x_i be the distance between the center of the i^{th} block and the $(i+1)^{st}$ block, so that x_i is randomly and uniformly chosen in the interval $[-1,1]$ (this is the reason to have a width of 2).

In order for n blocks to fall off the k^{th} block, one needs that the average of the centers of masses of blocks $k+1$, $k+2$, ..., $k+n$ to be at least 1 unit away from the center of mass of block k. That is, one needs that $|x_{k+1}+(x_{k+1}+x_{k+2})+\cdots+(x_{k+1}+\cdots+x_{k+n})|>n$. After some rejiggering, this becomes $|\sum_{i=1}^{n}(n+1-i)x_{k+i}|>n$.

This immediately suggests a way to code this up: choose a sequence of numbers $x_i \in [-1,1]$ uniformly at random, and then, after x_i is chosen, check to see if the top block falls off, and then if the top two blocks fall off, and so on until you check to see if the all but one of the blocks fall off. If that happens at x_i, then your tower is $i+1$ units tall.

While doing the simulations, I kept track of how many blocks fell off on average because there was no reason not to. After 10000000 simulations, on average, the tower collapsed when it was 7.1123513 units high, and on average, 3.4799419 blocks fell off. Below is a graph of what the distributions on the height was and number of blocks to fall off (height is in blue and blocks falling off is in red):



One thing I found interesting is that it appears you are more likely to have the tower collapse at 4 blocks high than you are to have it collapse at 3 blocks high. It is also possible to compute that the odds that the tower collapses at height 3 is exactly $\frac{1}{8}$. There isn't much to say about the number of blocks to fall off; it is by far most likely that the top two blocks fall off than any larger number. This intuitively makes sense: in order for the top three blocks to fall of, one needs $x_n + 2x_{n-1} + 3x_{n-2}$ to be bigger than three but $x_{n-1} + 2x_{n-2}$ to be less than two and that seems harder to arrange by accident than just $x_n + 2x_{n-1}$ to be larger than 2.

Finally, here's some python code:

```
import random
import matplotlib.pyplot as plt

##This simulates one jenga tower being stacked and
##outputs how tall the tower was when it collapsed
##and how many blocks fell off.  Of note: it gives
##the largest number of blocks that fall off in
##case there's multiple stacks of blocks that become
##unstable.
```

```
def simulate():
    tower = []
    h = 1
    stop = False
    while(not stop):
        tower.append(random.uniform(-1, 1))
        h += 1
        for k in range(1, len(tower)+1):
            s = 0
            for i in range(k):
                s += (k-i)*tower[h-k+i-1]
            if (s > k) or (s < -k):
                stop = True
                b = k
    return([h, b])

##This simulates samples number of jenga towers.
##heights is a list of all the heights that arise,
##theight is the sum of everything in heights, and
##mheight is the largest element in heights.
##breaks, tbreak, and mbreak are the same thing
##but for how many blocks fall off.

samples = 10000000
heights = []
theight = 0
mheight = 0
breaks = []
tbreak = 0
mbreak = 0

##This is the main simulation loop.  At the end, it
##prints out the average height and the average break
##value.

for i in range(samples):
    l = simulate()
    heights.append(l[0])
    theight += l[0]
    mheight = max(mheight, l[0])
    breaks.append(l[1])
    tbreak += l[1]
    mbreak = max(mbreak, l[1])

print(theight/samples, tbreak/samples)

##This stuff is to make the pretty graph.
```

```
totheights = []
totbreaks = []

for i in range(3,mheight+1):
    totheights.append(0)

for i in range(2, mbreak+1):
    totbreaks.append(0)

for i in range(samples):
    totheights[heights[i]-3] += 1
    totbreaks[breaks[i]-2] += 1

plt.plot(range(3, mheight+1), totheights, 'b-')
plt.plot(range(2, mbreak+1), totbreaks, 'r-')
plt.show()
```