Slots Simulation

Avery Lindsey

January 11, 2024

[1]:
```
import numpy as np
import pandas as pd
```

The following is an attempt to simulate the financial position of a hypothetical casino filled with hypothetical slot machines. The goal of this analysis is to produce a set of probabilities that will give the casino an edge, and see how tweaking a few variables of the simulation can affect the casino's profit margin, as well as the amount of money players win/lose.

In the following code, I define the possible amounts that can be won from each bet in a machine. Next, I define a "win rate" or as I like to think of it, how much of each dollar played should be returned to the player. Lastly, I define a starting amount of money for each player, the number of times each player bets, the cost of each bet, and the number of trials for each simulation.

Note each player is starting with $2,500, there are 1,000 players, and each player is going to bet 1,000 times. This means each trial will test 1,000,000 bets and $2,500,000 will be paid in for each trial. This is important to keep in mind as you review the results.

[2]:
```
winAmounts = np.array([10000,5000,1000,500,100,50,10,1])

winRate = 0.9
startingAmount = 2500.0
bets = 1000
betCost = 1
trials = 1000
```

Next, I define the probability of each win as the reciprocal of the value of each win divided by the win rate. As you'll see in a moment, that turns out to be a terrible probability function.

[3]:
```
def probFunction(x):
    return (1/(x/winRate))

f = np.vectorize(probFunction)
```

[4]:
```
winProbs = f(winAmounts)
```

Summing up the probabilities shows they're currently above 1. Due to the way I have the following functions setup, this won't break the simulation, but it will prevent the Casino from being profitable. It also doesn't make a lot of sense in the context of probability but that's ok for now.

[5]:
```
print(np.sum(winProbs))
```

```
1.0199699999999998
```

[6]:
```python
import random
```

The first function I define is the "bet function". This will generate a random number between 0 and 1, then see if the generated number is greater than a running total of each probability. This essentially segments the number line between 0 and 1 into different outcomes and picks a random point on the line. Any remaining space not defined by our probabilites will return 0, a loss for the player. As you saw previously, my probabilities currently have a sum greater than 1. This means the player will never really lose, and the casino is probably not going to stay in business.

[7]:
```python
def betFunction():
    n = random.random()
    index = 0

    runningTotal = 0
    for probability in winProbs:
        runningTotal += probability
        if n > runningTotal:
            index += 1
        elif n < runningTotal:
            return winAmounts[index]

    return 0
```

The next function, "gamble" defines a gambling session. The player's money balance is set to the starting amount of money defined above. Each time a bet is entered, it costs the player money. A return is then generated and multiplied by the bet amount. Results of tweaking this variable are discussed further in the paper.

The final balance of the player's account is returned after all their bets. If they run out of money, the function returns 0.

[8]:
```python
def gamble():
    money = startingAmount
    for i in range(bets):
        money -= betCost
        money += betFunction() * betCost
        if money <= 0:
            return 0
    return money
```

Next, I define a function to setup multiple gambling trials and let each one go to town. The final balance of each player's account is added to an array and returned.

[9]:
```python
def runTrials(numTrials):
    trial = []
    for i in range(numTrials):
        trial.append(gamble())
    return np.array(trial)
```

Finally, I define a function to output the casino profit metrics in a clean manner.

[10]:
```
def countProfit(payOut):
    payIn = float(startingAmount * trials)
    payOut = float(payOut)
    profit = float(payIn - payOut)
    profitPercent = float(profit / payIn) * 100

    print('Total payin: ' + str(round(payIn,2)))
    print('Total payout: ' + str(round(payOut,2)))
    print('Profit: ' + str(round(profit,2)))
    print('Profit margin: ' + str(round(profitPercent,2)) + '%')
```

The first trial is run with descriptive statistics. An explanation of what each number means follows:
Count: The number of players in the trial
Mean: The average amount of money a player left with
Std: The standard deviation of player winnings
Min: The lowest amount of money someone left with
25%: 25% of players left with less than this amount of money
50%: 50% of players left with less than this amount of money
75%: 75% of players left with less than this amount of money
Max: The highest amount of money someone left with

[11]:
```
trial1 = pd.DataFrame(runTrials(trials))
trial1.describe()
```

[11]:
```
                  0
count   1000.000000
mean    8559.435000
std     3805.871811
min     3940.000000
25%     6220.750000
50%     7231.000000
75%     9272.250000
max    32343.000000
```

Now I'll tally up the casino's position. The following information shows the total "payin", or the total amount of money all players started with; followed by the total "payout", or what the casino paid in winnings. These numbers are subtracted to produce the casino's "profit" and profit margin on the amount paid in.

Remember, there are 1,000 players and each player started with $2,500. This means $2,500,000 is being brought into the casino. In this case, $8,559,435 is leaving the casino, which leaves the casino with a net loss of $6,059,435.

[12]:
```
countProfit(trial1.sum())
```

```
Total payin: 2500000.0
Total payout: 8559435.0
```

```
Profit: -6059435.0
Profit margin: -242.38%
```

The probabilites were clearly horrible here. The average player walked out with quite a bit of extra cash, and the casino is hemorrhaging money. The whole "win rate" thing didn't work out how I imagined. Let's try some manually entered probabilites.

[13]:
```
winProbs = np.array([1/100000,1/50000,1/10000,1/5000,1/1000,1/800,1/400,1/4])
print(np.sum(winProbs))
```

```
0.25508
```

The sum of these probabilies are now well below 1. This leaves plenty of room for the players to lose and give money back to the casino. Let's see how it performs:

[14]:
```
trial2 = pd.DataFrame(runTrials(trials))
trial2.describe()
```

[14]:
```
                 0
count   1000.00000
mean    2321.40000
std     1220.18474
min     1734.00000
25%     1871.75000
50%     1968.00000
75%     2313.25000
max    12311.00000
```

[15]:
```
countProfit(trial2.sum())
```

```
Total payin: 2500000.0
Total payout: 2321400.0
Profit: 178600.0
Profit margin: 7.14%
```

That's better! (for the casino…) Now I'll try salvaging the original probability function:

[16]:
```
def probFunction(x):
    lossRate = 1-winRate
    return (1/(x/lossRate))

f = np.vectorize(probFunction)
winProbs = f(winAmounts)
print(np.sum(winProbs))
```

```
0.11332999999999997
```

[17]:
```
trial3 = pd.DataFrame(runTrials(trials))
trial3.describe()
```

```
[17]:                  0
      count   1000.000000
      mean    2255.544000
      std     1179.126809
      min     1626.000000
      25%     1835.750000
      50%     1954.500000
      75%     2270.500000
      max    12910.000000
```

```
[18]: countProfit(trial3.sum())
```

```
Total payin: 2500000.0
Total payout: 2255544.0
Profit: 244456.0
Profit margin: 9.78%
```

A little bit better than before. Now I'll try some manual adjustment of the probability function instead of using the "win rate".

```
[19]: def probFunction(x):
          return (1/(x/0.08))

      f = np.vectorize(probFunction)
      winProbs = f(winAmounts)

      trial4 = pd.DataFrame(runTrials(trials))
      trial4.describe()
```

```
[19]:                  0
      count   1000.000000
      mean    2127.364000
      std     1137.653213
      min     1601.000000
      25%     1753.750000
      50%     1840.500000
      75%     2018.250000
      max    12624.000000
```

```
[20]: countProfit(trial4.sum())
```

```
Total payin: 2500000.0
Total payout: 2127364.0
Profit: 372636.0
Profit margin: 14.91%
```

Now that's pretty profitable! If you look closely though, you'll notice 75% of players lost at least $482. That's a pretty sore day. In the real world, I imagine these variables would be tweaked to change the mechanics of how people lose money to maintain certain levels of player retention.

Let's try upping the bet a couple times and see what happens. I'm also going to switch back to the hard coded probabilities to make things easier to see what the odds are. First, we'll do another $1 bet for comparison. Pay attention to the casino profit margin.

```
[21]: winProbs = np.array([1/100000,1/50000,1/10000,1/5000,1/1000,1/800,1/400,1/4])
      betCost = 1

      trial5 = pd.DataFrame(runTrials(trials))
      trial5.describe()
```

```
[21]:                  0
       count   1000.000000
       mean    2335.119000
       std     1367.884761
       min     1715.000000
       25%     1872.000000
       50%     1977.000000
       75%     2310.750000
       max    13200.000000
```

```
[22]: countProfit(trial5.sum())
```

```
Total payin: 2500000.0
Total payout: 2335119.0
Profit: 164881.0
Profit margin: 6.6%
```

Now a $2 bet:

```
[23]: betCost = 2

      trial6 = pd.DataFrame(runTrials(trials))
      trial6.describe()
```

```
[23]:                  0
       count   1000.000000
       mean    2176.216000
       std     2684.234133
       min      964.000000
       25%     1261.500000
       50%     1473.000000
       75%     2199.500000
       max    23856.000000
```

```
[24]: countProfit(trial6.sum())
```

```
Total payin: 2500000.0
Total payout: 2176216.0
Profit: 323784.0
```

```
Profit margin: 12.95%
```

Finally a $4 bet:

[25]:
```
betCost = 4

trial7 = pd.DataFrame(runTrials(trials))
trial7.describe()
```

[25]:
```
                  0
count   1000.000000
mean    1804.592000
std     4994.735384
min        0.000000
25%        0.000000
50%      410.000000
75%     1586.000000
max    43952.000000
```

[26]:
```
countProfit(trial7.sum())
```

```
Total payin: 2500000.0
Total payout: 1804592.0
Profit: 695408.0
Profit margin: 27.82%
```

Upping the bet just magnifies the result. The biggest winners go home with more, but most people lose a lot more as well. At the $4 bet, the entire bottom quartile is wiped clean. 50% of players lost almost $2,000. The increased bet increases your "risk/reward" but doesn't change the probabilities at all. You're still expected to lose, but with a higher bet you're expected to lose more.

Now I'll change the number of gambles and see if that changes anything. Each player will now bet 10 times instead of 1,000. I'm also going to change the bet back to $1.

[27]:
```
bets = 10
betCost = 1

trial8 = pd.DataFrame(runTrials(trials))
trial8.describe()
```

[27]:
```
                0
count  1000.00000
mean   2506.87800
std     228.48442
min    2490.00000
25%    2492.00000
50%    2492.50000
75%    2494.00000
max    7492.00000
```

[28]:
```
countProfit(trial8.sum())
```

```
Total payin: 2500000.0
Total payout: 2506878.0
Profit: -6878.0
Profit margin: -0.28%
```

Wow. Bringing down the bets low enough causes the casino to lose it's edge (with these probabilities at least). The biggest loser only lost $10. Let's see what happens when we up the bets to 5,000.

[29]:
```
bets = 5000

trial9 = pd.DataFrame(runTrials(trials))
trial9.describe()
```

[29]:
```
                   0
count    1000.000000
mean     1822.816000
std      2965.548946
min         0.000000
25%         0.000000
50%       695.000000
75%      1785.250000
max     20042.000000
```

[30]:
```
countProfit(trial9.sum())
```

```
Total payin: 2500000.0
Total payout: 1822816.0
Profit: 677184.0
Profit margin: 27.09%
```

Even with the more "modest" probabilities I defined earlier, players are getting wiped out, and the casino is raking it in. Half of all players lost almost $2,000. What have we learned so far? The casino rigs the numbers in their favor, betting more money only amplifies the casino's advantage, and placing more bets also amplifies the casino's advantage.

This model clearly doesn't addres the entire psychological aspect of gambling. In the model, gamlbers stop at exactly 1000 bets, or when they run out of money, whichever comes first. In real life, people are influenced by all kinds of things and casino's employee thousands of people to adjust those influences.

The point of this isn't to make fun of gamblers or tell you what to do. Vices are a part of life and the more you understand their risks, the better equipped you are to maintain a healthy relationship with them. Alcohol is rigged in favor of liver cirrhosis; cigarettes are rigged in favor of lung cancer; and of course, gambling is rigged in favor of the casino.