AI Investigation: Using Binary Output Game with No Dominant Strategy to experiment using Confidence Intervals to determine inconclusive victories

AI Investigation Part 2

Goal: test using Confidence Intervals to determine inconclusive victories

Creator: `Lang (Ron) Chen 2022`

In [ ]:
```
import random
import math as m
```

Define Game

Basically a game with no optimal strategy. The chance of winning is 50% regardless of what strategy is chosen.

(The game is similar to Rock Paper Scissors, except deliberately avoided the chance of draws)

In [ ]:
```
CHOICE1 = ['A', 'B']
CHOICE2 = ['C', 'D']
```

In [ ]:
```
def winloss(player1, player2):
    """ If player 1 plays A and player 2 plays C, then player 1 wins; if player 2 plays D
    and player 2 plays C, then player 1 loses, if player 2 plays D then player 1 wins """

    if player1 == 'A':
        if player2 == 'C':
            return True
        else:
            return False
    else:
        if player2 == 'D':
            return True
        else:
            return False
```

Simulation

In [4]:
```
RUNS = 100000

sample = list()
victory = list()
for i in range(RUNS):
    obs1 = random.sample(CHOICE1, 1)[0]
    obs2 = random.sample(CHOICE2, 1)[0]
    sample.append(obs1)
    victory.append(winloss(obs1[0], obs2[0]))
```

Manipulation *Sidenote: In future basic data collection should be done WITHIN Simulation to save time*

In [5]:
```
A_Score = 0
```

```
B_Score = 0

A_win = 0
B_win = 0

A_Count = 0
B_Count = 0

for i in range(RUNS):
    if sample[i] == 'A':
        A_Count += 1
        if victory[i]:
            A_Score += 1
            A_win += 1
        else:
            A_Score -= 1
    else:
        B_Count += 1
        if victory[i]:
            B_Score += 1
            B_win += 1
        else:
            B_Score -= 1

ScoreDict = ({'A': A_Score, 'B': B_Score})
```

Can get an estimate of p by:

```
E(X) = 2p-1 => p_hat = (x_bar + 1)/2
```

In [6]:

```
A_Mean = A_Score/A_Count
B_Mean = B_Score/B_Count

P_A = (A_Mean + 1)/2
P_B = (B_Mean + 1)/2
```

Constructing confidence interval using

```
Mean = 2p-1

Var = 4(p_hat)(1-(p_hat))
```

(here we use p_hat as an approximation for p)

In [7]:

```
A_99CI = ((A_Mean/2) + 0.5 + (-1) * 2.57 * m.sqrt(4*P_A*(1-P_A))/(2*(m.sqrt(A_Count))),
          (A_Mean/2) + 0.5 + (1) * 2.57 * m.sqrt(4*P_A*(1-P_A))/(2*(m.sqrt(A_Count))))
```

In [8]:

```
print(f'A_Score: {A_Score}')
print(f'B_Score: {B_Score}')
print(f'A_Count: {A_Count}')
print(f'B_Count: {B_Count}')
print(f'A_Mean: {A_Mean}')
print(f'B_Mean: {B_Mean}')
print(f'P_A: {P_A}')
print(f'P_B: {P_B}')
print(f'99% Confidence Interval: {A_99CI}')
```

```
A_Score: 111
B_Score: -55
A_Count: 50083
B_Count: 49917
```

```
A_Mean: 0.0022163209072938923
B_Mean: -0.0011018290362000092
P_A: 0.5011081604536469
P_B: 0.4994490854819
99% Confidence Interval: (0.49536624368035564, 0.5068500772269382)
```

In [9]:

```
print(f'P_A: {A_win/A_Count}')
print(f'P_B: {B_win/B_Count}')
```

```
P_A: 0.5011081604536469
P_B: 0.4994490854819
```

Algorithm

In [10]:

```
def final_choice(ScoreDict, Scores, Counts):
    first = sorted(list(ScoreDict.items()), key = lambda x:x[1], reverse = True)[0][0]
    second = sorted(list(ScoreDict.items()), key = lambda x:x[1], reverse = True)[1][0]

    Means = {'A': Scores['A']/Counts['A'], 'B': Scores['B']/Counts['B']}

    Ps = {'A': (A_Mean + 1)/2, 'B': (B_Mean + 1)/2}

    # Construct 99% Confidence interval. If P of 'second' belongs in the 99% CI of first,
    # dominant strategy
    CI99 = ((Means[first]/2) + 0.5 + (-1) * 2.57 * m.sqrt(4*Ps[first]*(1-Ps[first]))/(2*(n
           (Means[first]/2) + 0.5 + (1) * 2.57 * m.sqrt(4*Ps[first]*(1-Ps[first]))/(2*(m.sc

    print(f'P_A: {Ps["A"]}')
    print(f'P_B: {Ps["B"]}')
    print(f'99% Confidence Interval: {CI99}')

    if Ps[second] > CI99[0] and Ps[second] < CI99[1]:
        return "No dominant strategy"

    return first
```

In [11]:

```
Scores = {'A': A_Score, 'B': B_Score}
Counts = {'A': A_Count, 'B': B_Count}
```

In [12]:

```
final_choice(ScoreDict, Scores, Counts)
```

```
P_A: 0.5011081604536469
P_B: 0.4994490854819
99% Confidence Interval: (0.49536624368035564, 0.5068500772269382)
```

Out[12]:

```
'No dominant strategy'
```

Final Note

Whilst this is a success, in reality changing the data into proportions is the same as using the mean to construct confidence intervals. In fact using the mean is more beneficial for games where the output are not binary (i.e. contain draws)