Assignment 6

Christian Diangco

Math 381 A

Suppose two people (P1 and P2) play a game where they take turns rolling a fair six-sided die.

After a player rolls, they can choose to either add the roll to their current score or subtract the roll from their opponent’s score.

Player 1 always rolls first.

If a player’s score ever becomes negative, it is set to 0.

The first player to reach a score of 30 or greater wins.

What strategies could be used to win this game, and how do they compare to each other?

We can create a simulation of this game in order to answer this question.

**Strategy A, Strategy B**

The first strategies we will compare are randomly adding or subtracting (strategy A) vs. only adding (strategy B).

P1 will use A and P2 will use B.

We will run 10 simulations of 100,000 games.

For each simulation, we will keep track of how many games each player won (i.e. the estimated probability of them winning).

We will use the following Python code to accomplish this:

import random

import numpy as np

# P1's strategy will be to randomly add to their score or subtract from their opponent's

# P2's strategy will be to only add to their score.

# Number of simulations we will run

n\_sim = 10

# Number of games per simulation

n\_game = 100000

# The score players must reach to end the game

limit = 30

# These lists contain the calculated probabilities of each player winning for each simulation

p1\_results = np.empty(n\_sim)

p2\_results = np.empty(n\_sim)

# These are the probabilities that each player will win calculated after each game for the first 10 simulations.

# These will be plotted to show convergence.

p1\_converge = np.zeros((10, n\_game))

p2\_converge = np.zeros((10, n\_game))

# Runs n\_sim simulations with n\_game games each

for i in range(n\_sim):

# Tracks the number of games each player wins out of n\_game games

p1\_wins = 0

p2\_wins = 0

for j in range(n\_game):

p1\_score = 0

p2\_score = 0

# This loop plays a game. It ends once a player reaches limit.

while True:

# P1 rolls first

p1\_roll = random.randint(1,6)

# 0 = add to score, 1 = subtract from P2's score

decision = random.randint(0,1)

if decision > 0:

p2\_score -= p1\_roll

p2\_score = 0 if p2\_score < 0 else p2\_score

else:

p1\_score += p1\_roll

if p1\_score >= limit:

break

# Start of P2's turn

p2\_roll = random.randint(1,6)

p2\_score += p2\_roll

if p2\_score >= limit:

break

# Whoever reached limit first gets a win added to their total

if p1\_score > p2\_score:

p1\_wins += 1

else:

p2\_wins += 1

p1\_converge[i,j] = p1\_wins / (j + 1)

p2\_converge[i,j] = p2\_wins / (j + 1)

p1\_results[i] = p1\_wins / n\_game

p2\_results[i] = p2\_wins / n\_game

# Saving arrays for plotting

with open("randomVSadd.npy", "wb") as f:

np.save(f, p1\_results)

np.save(f, p2\_results)

np.save(f, p1\_converge)

np.save(f,p2\_converge)

If our simulations are long enough, we should see the estimated probabilities converge towards a single value as the number of games increases.

The code above also estimated the probability of winning after each game was played.

We can plot these estimates using the following Python code:

import numpy as np

import matplotlib.pyplot as plt

# Load data from simulations

with open("randomVSadd.npy", "rb") as f:

random\_results = np.load(f)

add\_results = np.load(f)

random\_converge = np.load(f)

add\_converge = np.load(f)

# This figure has the plots showing the convergence of the probabilities of each player winning.

convergence\_plots, c = plt.subplots(1,2)

convergence\_plots.suptitle("Random vs Only Adding Convergence")

n\_game = np.arange(1, len(random\_converge[0,:])+1)

for sim in random\_converge:

c[0].plot(n\_game, sim, color="black", linewidth=.25)

c[0].set\_xlabel("Number of games")

c[0].set\_ylabel("Estimated probability of winning")

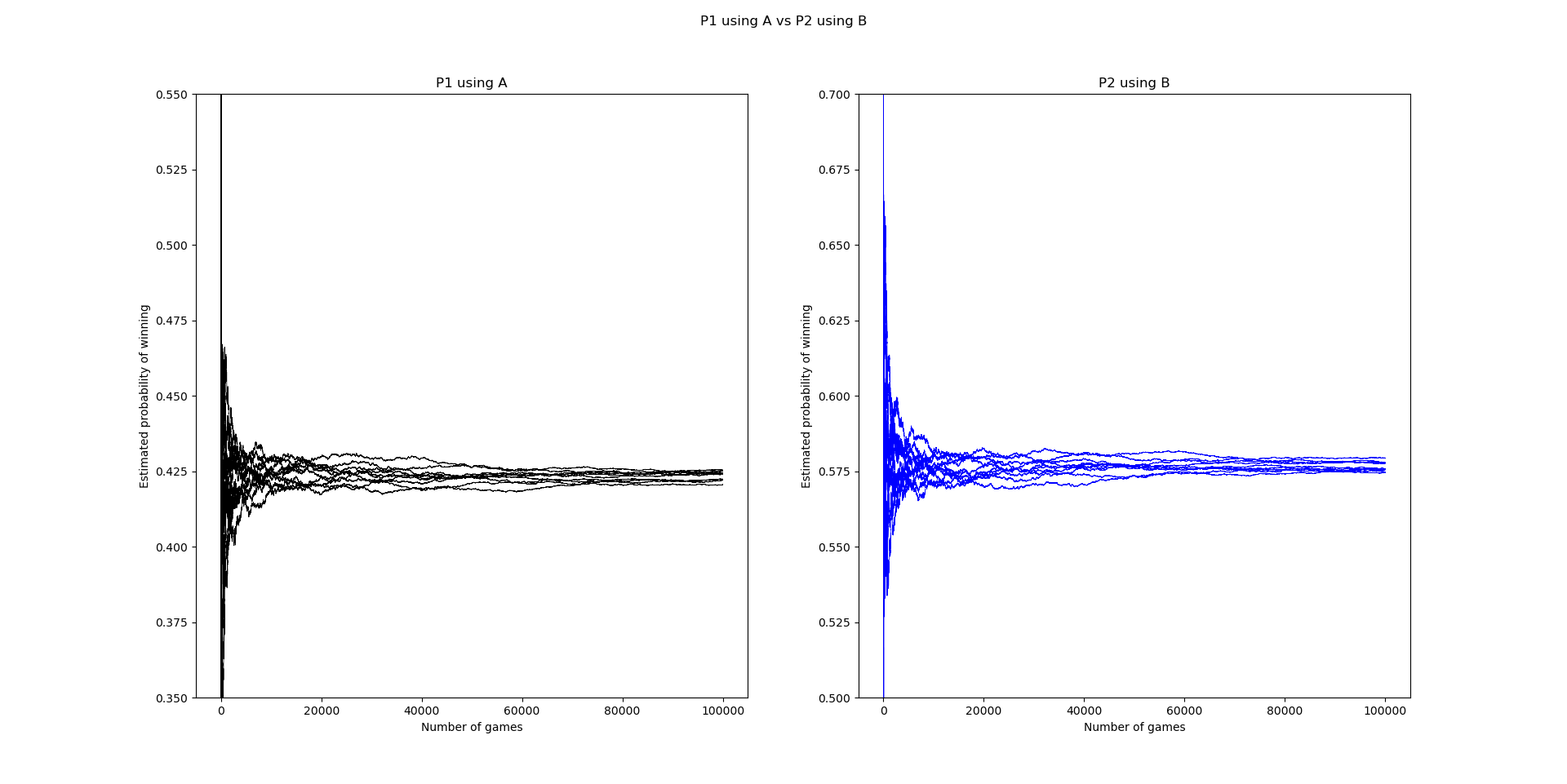
for sim in add\_converge:

c[1].plot(n\_game, sim, color="blue", linewidth=.25)

c[1].set\_xlabel("Number of games")

c[1].set\_ylabel("Estimated probability of winning")

plt.show()

This code generates these plots:

We can see that all 10 simulations converge to values that are very close to each other, but not exactly equal.

To make conclusions about the true probability, we can call upon the Central Limit Theorem.

It states that if we sample many values from an unknown distribution and average them, and do this multiple times, the averages will be approximately normally distributed around the mean of the distribution.

To show that this is true, we will create histograms of the estimated probabilities from 2500 simulations of 2500 games.

All we need to do is change n\_sim and n\_game in our simulation code and add the following code to our plotting code:

# This figure has the histograms of the computed probabilities of each player winning

histograms, h = plt.subplots(1,2)

histograms.suptitle("Random vs Only Adding Probability Estimates")

h[0].hist(random\_results, color="black", bins=30)

h[0].set\_title("Random")

h[0].set\_xlabel("Estimated probability of winning")

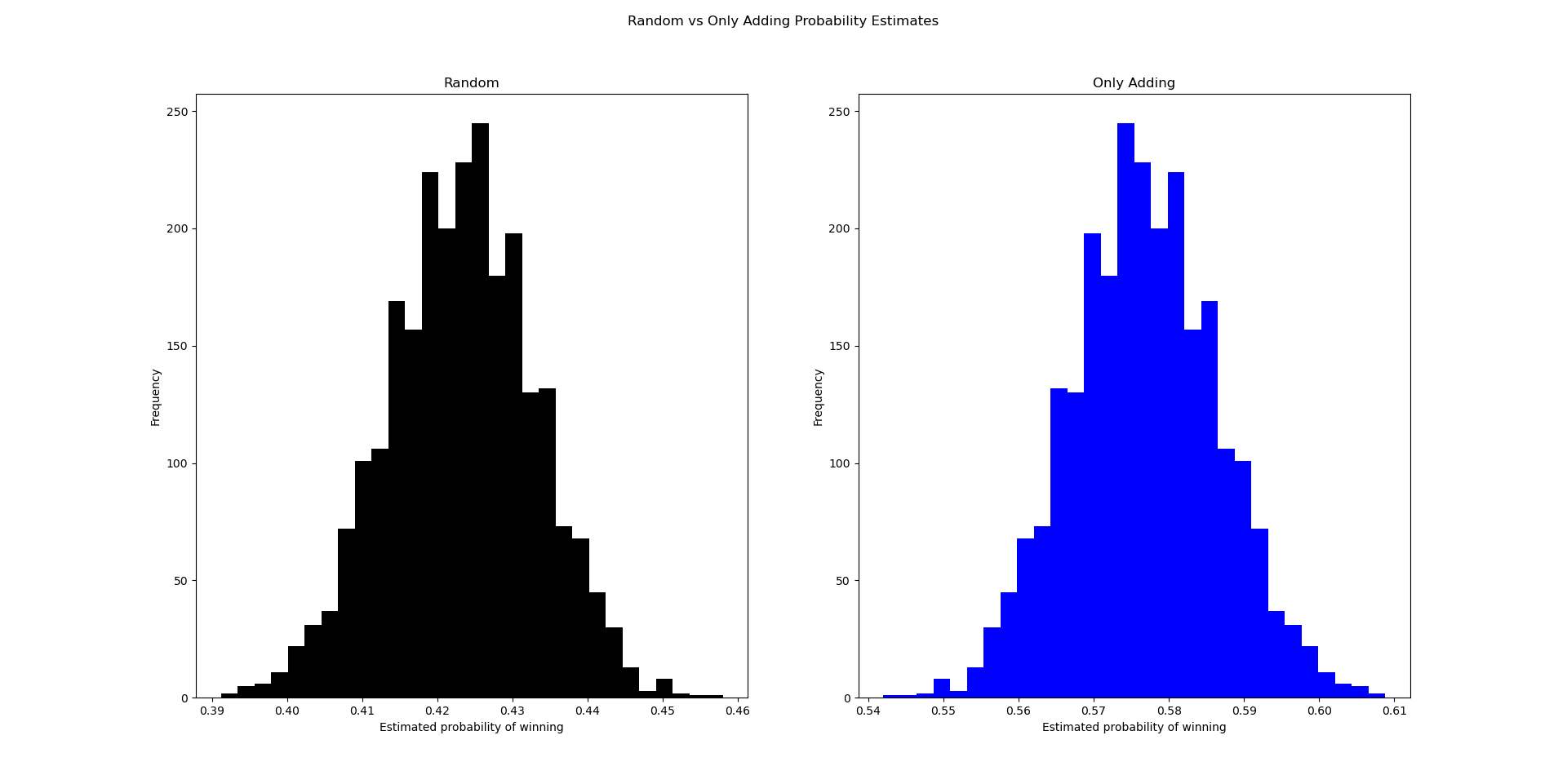
h[0].set\_ylabel("Frequency")

h[1].hist(add\_results, color="blue", bins=30)

h[1].set\_title("Only Adding")

h[1].set\_xlabel("Estimated probability of winning")

h[1].set\_ylabel("Frequency")

Which generates the following histograms (note: P1 is using Random and P2 is using Only Adding):

These histograms show that what the Central Limit Theorem says is true, as we have two roughly normal distributions of probabilities.

With our values being approximately normally distributed, we can calculate a confidence interval for the true probability of winning for each player.

If is the mean of the estimated probabilities for a given player, and *s* is the standard deviation, we can be 95% confident that

where *p* is the true probability.

We can calculate our confidence intervals using the following Python code:

import numpy as np

with open("randomVSadd.npy", "rb") as f:

p1\_results = np.load(f)

p2\_results = np.load(f)

# Probability that the true probability is between the highest and lowest estimate (95% confidence)

p1\_lower = np.mean(p1\_results) - 2 \* np.std(p1\_results)

p1\_upper = np.mean(p1\_results) + 2 \* np.std(p1\_results)

p2\_lower = np.mean(p2\_results) - 2 \* np.std(p2\_results)

p2\_upper = np.mean(p2\_results) + 2 \* np.std(p2\_results)

print("P1:", p1\_lower, p1\_upper)

print("P2:", p2\_lower, p2\_upper)

Which gives us the following output:

P1: 0.4205 0.4265

P2: 0.5735 0.5795

For each player, the lower and upper bound for the confidence interval is outputted.

So for P1 using strategy A,

.

For P2 strategy B,

.

Clearly, the probability of P2 winning using strategy B is greater when against P1 using strategy A.

However, strategy A performed better than I expected.

**Strategy C*n***

Another strategy for the game is to always subtract when your opponent is ahead by *n* or more, otherwise add, where *n* is some number (strategy C*n*).

Also, for strategy C*n*, when the player’s score is equal to 0, they must add to their score to avoid an endless loop.

Let’s compare strategy B and strategy C*n* for various values of *n.*

For this matchup, P1 will use strategy B and P2 will use strategy C*n*.

Here is the code for P2 using strategy C6, as an example:

.

.

.

# Start of P2's turn

p2\_roll = random.randint(1,6)

if p1\_score - p2\_score >= 6 and p2\_score > 0:

p1\_score -= p2\_roll

p1\_score = 0 if p1\_score < 0 else p1\_score

else:

p2\_score += p2\_roll

if p2\_score >= limit:

break

.

.

.

Below is a table showing the confidence intervals for each player with different values of *n*.

| Value of *n*, for C*n* | Confidence interval for P1 using B | Confidence interval for P2 using C*n* |
| --- | --- | --- |
| 0 | [0.4984, 0.5055] | [0.4945, 0.5016] |
| 1 | [0.3972, 0.4034] | [0.6128, 0.6154] |
| 2 | [0.3831, 0.3869] | [0.6131, 0.6169] |
| 3 | [0.3909, 0.396] | [0.604, 0.6091] |
| 4 | [0.4118, 0.4195] | [0.5805, 0.5882] |
| 5 | [0.4438, 0.4487] | [0.5513, 0.5562] |
| 6 | [0.4771, 0.4841] | [0.5159, 0.5229] |
| 7 | [0.5107, 0.5171] | [0.4829, 0.4893] |
| 8 | [0.5397, 0.5442] | [0.4558, 0.4603] |
| 9 | [0.5608, 0.5669] | [0.4331, 0.4392] |
| 10 | [0.5754, 0.5827] | [0.4173, 0.4246] |
| 15 | [0.5955, 0.6002] | [0.3998, 0.4045] |
| 20 | [0.5951, 0.5988] | [0.4012, 0.4049] |

For 0 < *n ,* P2 using strategy C*n* has a greater probability of winning against P1 using strategy B, with C1 giving the greatest probability of winning.

Otherwise, P1 has a greater probability of winning.

It also appears that in general, the smaller the value of *n*, the greater the probability of winning is for P2 (with the exception of C0).

But what if both players used strategy C*n*?

How do different values of *n* compare to other values of *n*?

We will not be looking at large values for *n*, as they seem to make strategy C*n* less effective.

Below is a table comparing the confidence intervals for different matchups of various values for *n*.

P1’s *n* values are in the leftmost column, and P2’s *n* values are in the top row.

P1 on left, P2 on top

| Value of *n*, for C*n* | 1 | 2 | 3 | 4 | 6 |
| --- | --- | --- | --- | --- | --- |
| 1 | P1: [0.5283, 0.5351]  P2: [0.4649, 0.4717] | P1: [0.4256, 0.431]  P2: [0.569, 0.5744] | P1: [0.4124, 0.4172]  P2: [0.5828, 0.5876] | P1: [0.4534, 0.4591]  P2: [0.5409, 0.5466] | P1: [0.5405, 0.5475]  P2: [0.4525, 0.4595] |
| 2 | P1: [0.6309, 0.6354]  P2: [0.3646, 0.3691] | P1: [0.5308, 0.5353]  P2: [0.4647, 0.4692] | P1: [0.5098, 0.5152]  P2: [0.4848, 0.4902] | P1: [0.5218, 0.5276]  P2: [0.4724, 0.4782] | P1: [0.5799, 0.5863]  P2: [0.4137, 0.4201] |
| 3 | P1: [0.6433, 0.6483]  P2: [0.3517, 0.3567] | P1: [0.551, 0.5559]  P2: [0.4441, 0.449] | P1: [0.5323, 0.5359]  P2: [0.4641, 0.4677] | P1: [0.5374, 0.5412]  P2: [0.4588, 0.4626] | P1: [0.5913, 0.5936]  P2: [0.4064, 0.4087] |
| 4 | P1: [0.6041, 0.6079]  P2: [0.3921, 0.3959] | P1: [0.5388, 0.5463]  P2: [0.4537, 0.4612] | P1: [0.524, 0.5314]  P2: [0.4686, 0.476] | P1: [0.5311, 0.5357]  P2: [0.4643, 0.4689] | P1: [0.5781, 0.5851]  P2: [0.4149, 0.4219] |
| 6 | P1: [0.5236, 0.5297]  P2: [0.4703, 0.4764] | P1: [0.4855, 0.4916]  P2: [0.5084, 0.5145] | P1: [0.4775, 0.4825]  P2: [0.5175, 0.5225] | P1: [0.4865, 0.4915]  P2: [0.5085, 0.5135] | P1: [0.5349, 0.5436]  P2: [0.4564, 0.4651] |

It appears that when both players use strategy C*n*, the player who rolls first (P1) is more likely to win for most combinations of *n*.

Even when both players use the same *n*, P1 has a greater probability of winning.

C3 is interesting because for the values tested above, it gave both players their highest probability of winning.

**Strategy D*m***

Another strategy that could be used is to always subtract when your opponent is *m* points from winning and your current roll doesn’t win you the game, otherwise add (strategy D*m*).

To be clear, if a player has 25 points and rolls a 5, even if their opponent is points from winning, the player will add to their own score to win the game.

Let’s first see how this strategy compares to the only adding strategy.

P1 will use strategy B and P2 will use strategy D*m*.

Here is the code for P2:

.

.

.

# Start of P2's turn

p2\_roll = random.randint(1,6)

if p1\_score + 6 >= limit and p2\_score + p2\_roll < limit:

p1\_score -= p2\_roll

p1\_score = 0 if p1\_score < 0 else p1\_score

else:

p2\_score += p2\_roll

if p2\_score >= limit:

break

.

.

.

Below is a table showing the confidence intervals for each player using various values of *m*:

| Value of *m*, for D*m* | Confidence interval for P1 using B | Confidence interval for P2 using D*m* |
| --- | --- | --- |
| 0 | [0.5952, 0.6006] | [0.3994, 0.4048] |
| 1 | [0.5798, 0.5877] | [0.4123, 0.4202] |
| 2 | [0.5645, 0.5693] | [0.4307, 0.4355] |
| 3 | [0.5437, 0.5496] | [0.4504, 0.4563] |
| 4 | [0.5202, 0.5261] | [0.4739, 0.4798] |
| 5 | [0.4927, 0.5] | [0.5, 0.5073] |
| 6 | [0.4727, 0.4778] | [0.5222, 0.5273] |
| 7 | [0.4749, 0.4797] | [0.5203, 0.5251] |
| 8 | [0.4764, 0.484] | [0.516, 0.5236] |
| 9 | [0.4798, 0.487] | [0.513, 0.5202] |
| 10 | [0.4872, 0.4928] | [0.5072, 0.5128] |
| 11 | [0.4954, 0.499] | [0.501, 0.5046] |
| 12 | [0.5024, 0.5091] | [0.4909, 0.4976] |
| 13 | [0.5101, 0.5152] | [0.4848, 0.4899] |
| 14 | [0.5199, 0.5232] | [0.4768, 0.4801] |
| 15 | [0.5241, 0.5318] | [0.4682, 0.4759] |

For , P2 using strategy D*m* has a greater probability of winning against P1 using strategy B, with D6 giving P2 the greatest probability of winning.

Otherwise, P1 has a greater probability of winning.

Interestingly, D6 corresponds to the opponent being possibly one roll from victory (i.e. they roll a 6 with a score of 24).

Let’s now take a look at how this strategy does against itself for various values of *m*.

Below is a table comparing the confidence intervals for different matchups of various values for *m*.

P1 on left, P2 on top

| Values of *m*, for D*m* | 5 | 6 | 7 | 8 | 10 |
| --- | --- | --- | --- | --- | --- |
| 5 | P1: [0.5732, 0.5792]  P2: [0.4208, 0.4268] | P1: [0.5362, 0.5431]  P2: [0.4569, 0.4638] | P1: [0.5374, 0.5451]  P2: [0.4549, 0.4626] | P1: [0.5419, 0.5481]  P2: [0.4519, 0.4581] | P1: [0.5474, 0.5545]  P2: [0.4455, 0.4526] |
| 6 | P1: [0.6048, 0.612]  P2: [0.388, 0.3952] | P1: [0.5678, 0.5736]  P2: [0.4264, 0.4322] | P1: [0.5766, 0.5832]  P2: [0.4168, 0.4234] | P1: [0.5842, 0.5918]  P2: [0.4082, 0.4158] | P1: [0.5993, 0.6054]  P2: [0.3946, 0.4007] |
| 7 | P1: [0.5982, 0.6032]  P2: [0.3968, 0.4018] | P1: [0.554, 0.5596]  P2: [0.4404, 0.446] | P1: [0.5637, 0.5699]  P2: [0.4301, 0.4363] | P1: [0.5744, 0.5782]  P2: [0.4218, 0.4256] | P1: [0.5876, 0.5937]  P2: [0.4063, 0.4124] |
| 8 | P1: [0.5933, 0.598]  P2: [0.402, 0.4067] | P1: [0.541, 0.5466]  P2: [0.4534, 0.459] | P1: [0.5494, 0.5563]  P2: [0.4437, 0.4506] | P1: [0.5591, 0.5647]  P2: [0.4353, 0.4409] | P1: [0.5771, 0.5826]  P2: [0.4174, 0.4229] |
| 10 | P1: [0.5766, 0.5809]  P2: [0.4191, 0.4234] | P1: [0.5153, 0.5206]  P2: [0.4794, 0.4847] | P1: [0.5253, 0.5305]  P2: [0.4695, 0.4747] | P1: [0.5355, 0.5406]  P2: [0.4594, 0.4645] | P1: [0.5529, 0.558]  P2: [0.442, 0.4471] |

Again, as seen for C*n*, the player who rolls first (P1) appears to be more likely to win, even when both players use the same *m*.

Unlike C*n*, however, P1 was more likely to win for all combinations of values tested.

D6 gave both players their highest probability of winning, which we also found when vs. strategy B.

It also appears that for both players, as *m* increases from 6, their probability of winning decreases.

Now that we’ve seen B vs. C*n*, B vs. *Dm*, and C*n* and D*m* vs. themselves, we will now analyze C*n* vs. D*m*.

We will look at the values of *n* and *m* that have shown to work best for C*n* and D*m* so far.

Below is a table showing the confidence intervals for P1 using C*n* and P2 using D*m.*

P1 using C*n*, P2 using D*m*

| Parameter Values | *m* = 6 | *m* = 7 | *m* = 8 |
| --- | --- | --- | --- |
| *n =* 1 | P1: [0.606, 0.6102]  P2: [0.3898, 0.394] | P1: [0.5919, 0.5981]  P2: [0.4019, 0.4081] | P1: [0.5816, 0.5864]  P2: [0.4136, 0.4184] |
| *n =* 2 | P1: [0.6155, 0.6224]  P2: [0.3776, 0.3845] | P1: [0.6162, 0.6217]  P2: [0.3783, 0.3838] | P1: [0.6059, 0.6097]  P2: [0.3903, 0.3941] |
| *n =* 3 | P1: [0.602, 0.6057]  P2: [0.3943, 0.398] | P1: [0.6126, 0.6186]  P2: [0.3814, 0.3874] | P1: [0.6162, 0.6215]  P2: [0.3785, 0.3838] |

P1 using C*n* has a much higher probability of winning against P2 using D*m* for the values tested.

But how much of this difference is due to P1 using C*n*?

We should also compare what happens when the two players switch strategies.

Below is a table showing just that:

P1 using D*m*, P2 using C*n*

| Parameter Values | *m* = 6 | *m* = 7 | *m* = 8 | *m =* 15 |
| --- | --- | --- | --- | --- |
| *n =* 1 | P1: [0.4744, 0.4787]  P2: [0.5213, 0.5256] | P1: [0.4846, 0.4892]  P2: [0.5108, 0.5154] | P1: [0.4926, 0.4991]  P2: [0.5009, 0.5074] | P1: [0.5362, 0.5409]  P2: [0.4591, 0.4638] |
| *n =* 2 | P1: [0.462, 0.4711]  P2: [0.5289, 0.538] | P1: [0.463, 0.4695]  P2: [0.5305, 0.537] | P1: [0.4717, 0.4798]  P2: [0.5202, 0.5283] | P1: [0.5138, 0.5192]  P2: [0.4808, 0.4862] |
| *n =* 3 | P1: [0.4819, 0.4856]  P2: [0.5144, 0.5181] | P1: [0.4689, 0.473]  P2: [0.527, 0.5311] | P1: [0.463, 0.4699]  P2: [0.5301, 0.537] | P1: [0.4998, 0.5101]  P2: [0.4899, 0.5002] |

We can see that even with the player using strategy D*m* rolling first, the player using strategy C*n* has a higher probability of winning for the variations tested.

However, I noticed that the probability of the D*m* player winning increased with *m* for some variations.

So I decided to test a higher value, D15, to see if this stayed true, which it appears to do so.

This is interesting because against itself and strategy B, generally, increasing *m* after 6 gave a lower probability of winning.

From these simulations, it appears that overall, the best strategy is C*n*.

It is also clear that rolling first gives a clear advantage.

So the optimal play would be to roll first and use maybe strategy C1, C2, or C3.

If you are P2 against P1 using these strategies, your best option would be to use a higher value version of D*m* (e.g. D15).