**Code explaination on how it relates to ‘Majority is not Enough: Bitcoin mining is Vulnerable’**

def pool (alpha, gamma, simulations):  
 chain = Mine (0, 0, 0, 0, 0, 0)

This sets all variables initalised in the initalization function to 0.

Scenario 1 – (Random = 0.2, selfish\_chain = 0, honest\_chain = 0)

With this, going forward lets assume the following:

* Selfish\_chain == **0**
* Honest\_chain == **0**

Lets first anaylise how this behaves with the **first 2** if statements.

for i in range(1, simulations):  
 #for loop with be iterated through n number of simulations, which is defined in the main  
 if random.uniform(0, 1) <= alpha:  
 # if random int value between 0,1 is less than equal to alpha  
  
 if chain.selfish\_chain > 0 and chain.selfish\_chain == chain.honest\_chain:  
 chain.selfish\_chain += 1 # state is 1  
 chain.selfish\_block += chain.selfish\_chain # selfish block mined is added onto the selfish chain  
 chain.honest\_orphan += chain.honest\_chain # honest block is added to the orphan as that isn't the longest chain  
 chain.honest\_chain = 0  
 chain.selfish\_chain = 0  
 # in this case we assume that the selfish miner doesnt publish the block  
  
 else:  
 chain.selfish\_chain += 1  
 # selfish miner doesnt publish the block

Now lets enter the initial for loop:

for i in range(1, simulations):

This is just the number of “Simulations” to be performed. This number is defined in the main. Basically just the number of times this for loop will run.

if random.uniform(0, 1) <= alpha:

This generates a random number between 0 and 1. Checks if its less than or equal to value alpha. Alpha has been defined in the main as 0.33.

Lets assume the following:

* Random in this case == **0.2**

if chain.selfish\_chain > 0 and chain.selfish\_chain == chain.honest\_chain:

Now we make our first checks. This first if statement is essentially the frontier, stating “If the selfish chain is greater than 0 and selfishchain and honestchain are equal”.

The check to see if selfish chain is greater than 0 is to allow us to assume that honest blocks would be orphaned since they are no longer the longest chain, or “frontier”. This is done in the body of the if statement.

However for our example lets make the check:

if chain.selfish\_chain > 0

Our chain is not greater than 0, so we immediately jump to the else statement.

else:  
 chain.selfish\_chain += 1

**In the Markov chain principle as discussed in the Selfish-Mine strategy, this if statement reflects State 0 prime.**

**“The selfish miner lead is 0 at initiation, and so are at equality between selifsh mine and honest mine branches. In this case alpha will increment the selfish chain, 1-alpha wont.”**

Scenario 2 – (Random = 0.2, selfish\_chain = 1, honest\_chain = 0)

For this next instance, we have the following:

* Selfish\_chain == **1**
* Honest\_chain == **0**

Lets assume the following:

* Random in this case == **0.2**

Lets go back into the loop with these conditions.

if chain.selfish\_chain > 0 and chain.selfish\_chain == chain.honest\_chain:

Now in this case, Selfish Chain is greater than 1, so it passes this check. However it won’t meet the condition to check if the Selfish Chain is the same as the Honest Chain; we have checks 1 == 0.

This is because the body of this if statement intends to orphan the Honest Chains winnings if they are at a frontier scenario, where random is less than alpha. I.e the Selfish Miner has won, here is what will happen now..

So again, we enter the else statement

else:  
 chain.selfish\_chain += 1

This results in the following:

* Selfish\_chain == **2**
* Honest\_chain == **0**

This is how the first two if statements will behave, this will continue to loop until the random number generated is greater than alpha (0.33).

**In the Markov chain principle as discussed in the Selfish-Mine strategy, this if statement reflects State 1.**

**“When the state is 1, honest miners can either catch up with their delay, closing the gap to 1 or selfish miners increase their lead to 2.”**

**“This can also head to state > 2, selfish miners are comfortably ahead, and keep on mining anonmously, releasing one block at a time, when honest miners find a block”**

Scenario 3 – (Random = 0.2, selfish\_chain = 1, honest\_chain = 1)

For this next instance, we have the following:

* Selfish\_chain == **1**
* Honest\_chain == **1**

The scenario where the Honest Chain can achieve a score of 1 is detailed in Scenario 5.

Lets assume the following:

* Random in this case == **0.2**

We make the same check as the previous two Scenarios:

if chain.selfish\_chain > 0 and chain.selfish\_chain == chain.honest\_chain:

In this instance, Selfish Chain is greater than 0 and both chains hold the same value as 1.

So we enter the body of the if statement.

Due to random (0.2) being less than alpha (0.33) a Selfish Block is mined, and this is added onto the Selfish Chain.

chain.selfish\_chain += 1 # state is 1  
chain.selfish\_block += chain.selfish\_chain # selfish block mined is added onto the selfish chain

Because the selfish chain is longer, this is now the "frontier", this makes the Honest Chain now Honest Orphans.

chain.honest\_orphan += chain.honest\_chain # honest block is added to the orphan as that isn't the longest chain  
chain.honest\_chain, chain.selfish\_chain = 0, 0

**In the Markov chain principle as discussed in the Selfish-Mine strategy, this if statement reflects State 0 prime.**

**“The selfish miner lead is 0 at initiation, and so are at equality between selifsh mine and honest mine branches. In this case alpha will increment the selfish chain, 1-alpha wont.”**

Scenario 4 – (Random = 0.2, selfish\_chain = 0, honest\_chain = 0)

Next I’ll run through the next else condition. For this we assume a number greater than alpha (0.33).

else: # in this instance the honest miners get a block  
  
 if chain.selfish\_chain == 0:  
 chain.honest\_chain += 1  
 chain.honest\_block += chain.honest\_chain  
 chain.honest\_chain = 0  
  
 elif chain.selfish\_chain == chain.honest\_chain and random.uniform(0, 1) < float(gamma):  
 #  
 # where gamma = ratio of honest miners who mine on the selfish chain  
 chain.selfish\_block += chain.selfish\_chain  
 chain.honest\_orphan += chain.honest\_chain  
 chain.honest\_block += 1  
 chain.selfish\_chain = 0  
 chain.honest\_chain = 0  
 #elif chain.selfish\_mine\_chain == chain.honest\_mine\_chain and random.uniform(0, 1) >= float(gamma):  
  
  
 else:  
 #append block onto the selfish chain  
 chain.honest\_chain += 1

if chain.selfish\_chain == chain.honest\_chain + 1:  
 #honest miners gains are orphaned since selfish miner chain is the longest  
 chain.selfish\_block += chain.selfish\_chain  
 chain.honest\_orphan += chain.honest\_chain  
 chain.selfish\_chain, chain.honest\_chain = 0, 0  
  
  
if chain.honest\_chain > chain.selfish\_chain:  
 #selfish miners gains are orphaned since honest miner chain is the longest  
 chain.honest\_block += chain.honest\_chain  
 chain.selfish\_orphan += chain.selfish\_chain  
 chain.honest\_chain, chain.selfish\_chain = 0, 0

With this, going forward lets assume the following:

* Selfish\_chain == **0**
* Honest\_chain == **0**

We will iterate through the code on pages 1 and 2, however in this instance we change the following:

Lets assume the following:

* Random in this case == **0.6**

So in this scenario, the Honest Miners have “won” a block.

Lets make the first initial check:

if chain.selfish\_chain == 0:

Our Selfish Chain is assumed to be 0 in this instance, so it passes this check.

The value of the Honest Chain is then added to the Honest Block, so increment by 1

Then Honest Chain is reset back to 0.

chain.honest\_chain += 1  
chain.honest\_block += chain.honest\_chain  
chain.honest\_chain = 0

**In the Markov chain principle as discussed in the Selfish-Mine strategy, this if statement reflects State 0 prime.**

**“The selfish miner lead is 0 at initiation, and so are at equality between selifsh mine and honest mine branches. In this case alpha will increment the selfish chain, 1-alpha wont.”**

**In this instance honest miners won at initation.**

Scenario 5 – (Random = 0.6, selfish\_chain = 1, honest\_chain = 0)

For this next instance, we have the following:

* Selfish\_chain == **1**
* Honest\_chain == **0**

Lets assume the following:

* Random in this case == **0.6**

So in this scenario, the Honest Miners have “won” a block again.

First initial check:

if chain.selfish\_chain == 0:

This is failed because the Selfish Chain == 1

So we make our second check against the elif statement:

elif chain.selfish\_chain == chain.honest\_chain and random.uniform(0, 1) < float(gamma):

In this case, Selifish Chain doesn’t equal Honest Chain due to Selfish Chain having value 1, while Honest Chain has value 0.

Therefore:

else:  
 #append block onto the selfish chain  
 chain.honest\_chain += 1

Honest Chain is incremented by 1.

So the chains now hold values:

* Honest\_chain = **1**
* Selfish\_chain = **1**

if chain.selfish\_chain == chain.honest\_chain + 1:  
 #honest miners gains are orphaned since selfish miner chain is the longest  
 chain.selfish\_block += chain.selfish\_chain  
 chain.honest\_orphan += chain.honest\_chain  
 chain.selfish\_chain, chain.honest\_chain = 0, 0

Due to the Honest Chain being incremented in the previous else statement, we enter this if statement.

Because the honest miner has just caught up to the selfish miner, the selfish miner publishes their chain.

This adds the current value of the Selfish Chain, in this case 1.

The Honest Chain is now disregarded, since the Selfish Chain has been published. This means Honest Orphans is incremented by 1.

This is the end of this iteration, so both chains are initalised back to 0.

**In the Markov chain principle as discussed in the Selfish-Mine strategy, this if statement reflects State 2.**

**“When the state is 2, honest miners can either catch up, closing the gap to 1 or selfish miners increase their lead. In the former case, selfish miners will publish their chain to ensure computational resources arent wasted.”**

Scenario 6 – (Random = 0.6, selfish\_chain = 1, honest\_chain = 1)

For this next instance, we have the following:

* Selfish\_chain == **1**
* Honest\_chain == **1**

Lets assume the following:

* Random in this case == **0.6**

So in this scenario, the Honest Miners have “won” a block again.

First initial check:

if chain.selfish\_chain == 0:

This is failed because the Selfish Chain == 1

So we make our second check against the elif statement:

elif chain.selfish\_chain == chain.honest\_chain and random.uniform(0, 1) < float(gamma):

In this case, lets assume random is 0.5, which is greater than gamma. So we move to the next statement.

Therefore:

else:  
 #append block onto the selfish chain  
 chain.honest\_chain += 1

Honest Chain is incremented by 1.

So the chains now hold values:

* honest\_chain = **2**
* selfish\_chain = **1**

We then go into the next block of if statements.

if chain.honest\_chain > chain.selfish\_chain:  
 #selfish miners gains are orphaned since honest miner chain is the longest  
 chain.honest\_block += chain.honest\_chain  
 chain.selfish\_orphan += chain.selfish\_chain  
 chain.honest\_chain, chain.selfish\_chain = 0, 0

We make a check to see if the Honest Chain is larger than the Selfish Chain, which in this instance is true.

Therefore, Honest Chain increments the Honest Block value, in this case by 2.

The Selfish Chain value is added onto the Selfish Orphans, in this case by 1.

The chains are reset back to 0, 0 as this is the end of the iteration.

**In the Markov chain principle as discussed in the Selfish-Mine strategy, this if statement reflects State 0.**

**“When the state is 0, selfish miners adopt the public chain, as honest miners mined one or more blocks in advance over the selfish chain, creating the new public frontier”.**

Scenario 7 – (Random = 0.6, selfish\_chain = 1, honest\_chain = 1)

For this next instance, we have the following:

* Selfish\_chain == **1**
* Honest\_chain == **1**

Lets assume the following:

* Random in this case == **0.6**

So in this scenario, the Honest Miners have “won” a block again.

First check made:

if chain.selfish\_chain == 0:

This is failed because the Selfish Chain == 1

So we make our second check against the elif statement:

elif chain.selfish\_chain == chain.honest\_chain and random.uniform(0, 1) < float(gamma):

In this case, lets the Selfish Chain and Honest Chain are both equal

Lets assume the following:

* Random in this case == **0.2**

Therefore random is less than gamma (**0.3**). The intention of this is to simulate the probability of Honest Miners mining on the Selfish Chain.

If random is less than gamma, we assume that this is the case, resulting in the following:

chain.selfish\_block += chain.selfish\_chain  
chain.honest\_orphan += chain.honest\_chain  
chain.honest\_block += 1  
chain.selfish\_chain, chain.honest\_chain = 0, 0

Selfish Miners gain another block, as helped by the Honest Miners. The chains are then initalised back to 0.

* Honest\_chain = **0**
* Selfish\_chain = **0**

**In the Markov chain principle as discussed in the Selfish-Mine strategy, this if statement reflects State 0 prime.**

**“The selfish miner lead is 0 at initiation, and so are at equality between selifsh mine and honest mine branches. In this case gamma will increment the selfish chain, 1-gamma wont.”**

**Documenting the Main**

def main():  
 alpha = 0.33  
 gamma = 0.3

Alpha variable defined and value assigned. Alpha is the value used to represent the hashing power of Selfish Miners.

Gamma variable defined and value assigned. Gamma is the value used to represent the number of Honest Miners that accidently mine on the Selfish Miners chain.

simulations = 200000

Number of times the method “pool” is ran. In this instance, 200,000 “iterations” were ran. This means the sum of Selfish Blocks, Honest Blocks, Selfish Orphans and Honest Orphans will equal the number of simulations ran.

results = [simulations, alpha, gamma, ((alpha \* (1 - alpha) \*\* 2 \* (4 \* alpha + gamma \* (1 - 2 \* alpha)) - alpha \*\* 3) / (1 - alpha \* (1 + (2 - alpha) \* alpha))), chain.selfish\_block / float(chain.selfish\_block + chain.honest\_block),  
 chain.selfish\_block, chain.honest\_block, chain.selfish\_orphan, chain.honest\_orphan, 100\*round(chain.selfish\_block / (chain.selfish\_block + chain.honest\_block),3)]  
with open('results.csv', 'a', encoding='utf-8') as lines:  
 lines.write(', '.join([str(x) for x in results]) + '\n')

<Should probably be split into a separate function, I will refactor this once confirmed that everything is working as intented>

This stores number of simulations, alpha, gamma, and all subsequent outputs into a variable called “results”.

An empty CSV which is saved in the same folder directory is then opened, and then variable “results” then writes all of the outputs to the CSV. Each time the program is run, a different set of results is generated, which is appended to the line below the previous set of results.

print("\n Number of Simulations | %d \n α (Alpha) | %f (Selfish miner hash power) \n γ (Gamma) | %f (Proportion of honest miners which mine on the selfish pool)" % (simulations, alpha, gamma))

Prints number of simulations, alpha, and gamma values in the following format:

Number of Simulations | 200000

α (Alpha) | 0.330000 (Selfish miner hash power)

γ (Gamma) | 0.300000 (Proportion of honest miners which mine on the selfish pool)

print("Theoretical probability of selfish miner hashing power | ", ((alpha \* (1 - alpha) \*\* 2 \* (4 \* alpha + gamma \* (1 - 2 \* alpha)) - alpha \*\* 3) / (1 - alpha \* (1 + (2 - alpha) \* alpha))))

Prints the theoretical probability of the Selfish Miners hashing power in the following format:

Theoretical probability of selfish miner hashing power | 0.35791962912051334

This value was taken from ‘Majority is not Enough’ – section 4.2 Profitability.

“Rpool = rpool rpool + rothers = · · · = α(1 − α) 2 (4α + γ(1 − 2α)) − α 3 1 − α(1 + (2 − α)α) .”

print("Simulated probability of selfish miner hashing power | ", chain.selfish\_block / float(chain.selfish\_block + chain.honest\_block))

Prints the calculated probability of the Seflish Miners hashing power in the following format:

Simulated probability of Selfish Miner hashing power | 0.35688267683185093

This is calculated through dividing the sum of the selfish blocks by the total blocks mined.

print("Number of Selfish Blocks Mined | ", chain.selfish\_block)

Prints the total number of “selfish blocks” mined in the following format:

Number of Selfish Blocks Mined | 54769

print("Number of Honest Blocks Mined | ", chain.honest\_block)

Prints the total number of “honest blocks” mined in the following format:

Number of Honest Blocks Mined | 98696

print("Number of Selfish Orphan Blocks | ", chain.selfish\_orphan)

Prints the total number of “selfish blocks orphaned” in the following format:

Number of Selfish Orphan Blocks | 11063

print("Number of Honest Orphan Blocks | ", chain.honest\_orphan)

Prints the total number of “honest blocks orphaned” in the following format:

Number of Honest Orphan Blocks | 35471

print("Profitability | ", 100\*round(chain.selfish\_block / (chain.selfish\_block + chain.honest\_block),3))

Prints the calculated profitability of the selfish miner by dividing the number of selfish mined blocks by the total number of mined blocks. 100\*round then rounds this number to 3 decimal places.

This is outputted in the following format:

Profitability | 36.0