**Alice vs. Bob**

**Expected Outcome**

|  |  |
| --- | --- |
| ALICE | BOB |
| Skip | Found 1 |
| Skip | Found 2 |
| Found 3 | Skip |
| Skip | Found 4 |
| Skip | Found 5 |
| Skip | Found 6 |
| Skip | Found 7 |
| Found 8 | Skip |
| Skip | Found 9 |
| Found 10 | Skip |
| Found 11 | Skip |

Running Alice.py and Bob.py “Simultaneously” will produce this outcome.

**Program output**

Bob finds block 1 first so Alice skips.

A screenshot of a computer

Description automatically generated

A computer screen shot of a code

Description automatically generated

Bob finds block 2 first so Alice skips.

A black background with white text

Description automatically generated

A computer screen shot of a number and text

Description automatically generated

Alice finds block 3 first so Bob skips.

A black background with white text

Description automatically generated

A computer screen shot of a number and text

Description automatically generated

Bob finds block 4 first so Alice skips.

A black background with white text

Description automatically generated

A computer screen shot of numbers and symbols

Description automatically generated

Bob finds block 5 first so Alice skips.

A black background with white text

Description automatically generated

A computer screen shot of a number

Description automatically generated

Bob finds block 6 so Alice skips.

A black background with white text

Description automatically generated

A computer screen shot of numbers and letters

Description automatically generated

Bob finds block 7 so Alice skips

A black background with white text

Description automatically generated

A computer screen shot of a code

Description automatically generated

Alice finds block 8 first so Bob skips.

A purple background with white letters

Description automatically generated

A computer screen shot of a number

Description automatically generated

Bob finds block 9 first so Alice skips

A black background with white letters

Description automatically generated

A computer screen shot of numbers and letters

Description automatically generated

Alice finds block 10 first so bob skips.

A black background with white text

Description automatically generated

A computer screen shot of a number

Description automatically generated

Alice finds block 11 first so Bob skips.

A black background with white text

Description automatically generated

A computer screen shot of a number and text

Description automatically generated

**Code snippets (issues faced and resolutions)**

I decided to try the multi-threading way since it was something new to learn. The first thing I had to do was make sure which loops do I wanted running. (Which threads do I need)

Thread 1 = mining thread  
Thread 2 = message handling thread

These two threads run concurrently. Thread 1 finding the nonce and thread 2 handling messages from the other player (Bob and Alice)

The mining thread was simple as the mining function was basically the same as the lab. Just an addition of code that sends the successful transaction over to the other player.

A computer screen shot of text

Description automatically generated

The message handler is in a while true loop and is constantly listening for incoming messages.

A screen shot of a computer

Description automatically generated

Technically the message handler is handled by MySubscribeCallback() class. So I added an instance variable so that I can store the incoming message in that variable so that my message\_handler() function can access the stored message.

Once the message is received. It calls on the verification function which is the same as the one in Lab 4. And verifies if the current block hash value matches the hash of the previous block. If it is true, it sets a threading event called stop\_mining\_event. This event is to break the mining loop for Alice since Bob has already mined the block and vice versa. This is how they are able to mine , send messages and stop mining all concurrently.

A computer code on a white background

Description automatically generated

**ISSUE 1**

First issue I faced was not really a coding issue, but the performance issue of the program. While almost negligible, it takes time for the transaction message to be sent over to Bob (or Alice), and because of this delay it caused my program to either be out of sync, or not be able to process the message fast enough before another message comes in.

In this case, Bob is successful in mining block 4. So, he sends the block 4 information over to Alice. But because the nonce value of Bob’s block 5 is so small, he sends the block 5 information to Alice before Alice can finish processing and stopping her mining for block 4. This causes the block 4 data to be overridden by block 5 data, effectively causing 1 block to be “missing” .

A screenshot of a computer

Description automatically generated

Nonce value of 29,947 (almost instantaneously mined)

**Resolution of issue #1**

After…hours of trouble shooting, I found this problem and I managed to solve it by adding a delay in between each block mined. So that both Alice and Bob’s programs have time to do their processing before another message comes in. (time.sleep)



**ISSUE 2**

Issue 2 came about because of the resolution of issue 1. The way I run both programs is to open 2 terminal side by side and try to run both programs at the same time. (im sure there is a better way to run 2 scripts at the same time)

A screenshot of a computer

Description automatically generated

While the time.sleep helps with synchronization AFTER block 1 is found. The program is still prone to Bob and Alice finding a block at the “exact” same time. ( the nonce for block 7 for Alice was 800,000ish and the nonce for block 7 for Bob was about 750000ish, so if the program was delayed by even 1 second, they would’ve mined it simultaneously which was the problem in my case)

And if they do, both programs will send over a message to each other. This effectively causes

1. The wrong hash value for the remaining blocks (due to overridden .json files)
2. Mining stop even when it is not supposed to be stopped (therefore not creating the .json file)

**Resolution of issue #2**

What I realized is that while the message takes some time to be sent over to the other player, the creation of the local file is almost instantaneous. ( or at least faster then the sending and processing of the message )

Therefore I added an extra condition to my if statement in the message handler. This if statement was originally just supposed to tell the player to stop mining the block if the block is valid. But now it also checks if the .json file for the block already exists. If it exists it means that the other player technically mined it first so it will send set the event and stop the mining of the current block. If the file doesn’t exist yet, it carries on and creates the .json file. Problem solved.

A computer code with black and blue text

Description automatically generated

Thank you for coming to my ted talk. Fun assignment