How did you approach the project?

I decided to use a python queue to manage waiting clients since it has built in support for locking. As soon as a teller becomes available they could just grab a client off the queue and start serving them.

How did you organize the project? Why?

I put everything in one file to keep things simple, because of how threading works in python it made the most sense to define a client function and a teller function and then launch them all from some for loops, this also makes it easy to tweak the numbers of clients/tellers/managers/vault users. Tellers run on a loop until there are no more clients waiting, and clients run through their actions before automatically exiting, using the semaphores to block while waiting for a teller to become available.

What problems did you encounter? How did you fix them?

Figuring out how to know when to close the tellers/bank drove me to use a python queue since my initial plan of just keeping track of which clients had been served led to two bankers trying to serve the same client on some occasions. This has the added benefit of keeping client acquisition in sequential order, which I know wasn't strictly required but it keeps the output a little more orderly like what an actual bank would operate like.

What did you learn doing this project?

I was familiar with the concept of locks/multi-threading before, but had never actually implemented a semaphore, usually I just use more advanced structures like context managers and ThreadPoolExecutors. After doing some of the research for this project I saw how semaphores can be used in conjunction with these more advanced structures and I might try to implement one for a personal project I have which has a computer constantly spitting out I/O on a separate thread.

If you did not complete some feature of the project, why not?

I had trouble getting inter-thread communication to work. I tried a few different methods but because of how much back and forth communication there is I couldn't make anything work smoothly. I believe the proper way to handle this would be to use events, but I'm not super familiar with how events work and had trouble establishing the connection between a client and teller and my solutions often led to multiple tellers trying to respond to one event triggered by the client.