Section 1: Multithreaded Loop Architecture

- This section was probably the easiest section for me to complete. I am currently learning threads in 246, so I was able to use and understand the examples we learned in class and the one provided in this class to make my version of multithreading. At first, I used threads to get delta time and to check my inputs. Later throughout my design, I realized I wanted to handle input in the main loop so I could see if the current window was in focus (I had an issue where clients would move regardless of what window I was in). After completing section 3, I knew I needed to use threading to continuously search for new clients or new client input. I would later use this in my final implementation of this homework. I chose to use threading in my client though for handling subscriber messages and parsing and for also handling delta time.

Section 2: Measuring and Representing Time

- I started this section first since it would be the easiest to complete before other sections of the assignment. I took a lot of inspiration from lecture 5 and lecture 6. I had already been researching delta time before this assignment because, although not required, I wanted to add a jump mechanic for my player. This was difficult because I was using the SFML libraries for a while and running into bizarre errors. Most of the function implementation for the suggested Timeline class was pretty straightforward once I realized I could get the current time from "`std::chrono::duration_cast<std::chrono::milliseconds>(std::chrono::system_clock::now().time_since_epoch()).count()`" which helped way more than using the SFML libraries for getting time.
- My biggest issue, was when I implemented my pause onto my player. If I paused my player, it would build up velocity and randomly teleport. For a while, I poked around my Timeline class to see how I could be getting an absurdly high frame delta. I initially thought there was something wrong with my getTime() function since it was doing most of the heavy lifting in the Timeline class. I made sure that my Timeline class was performing the equations properly. I would scale my delta time so I don't move like 10 units if I'm really aiming to move one or two. This still works well for my pause, since I just multiply my velocities by my time. I also don't even update the character if paused, so it keeps its momentum after unpaused. My characters will now multiply their velocity by a scalar of delta time.

Section 3: Networking Basics

- This section was relatively quick, but still took longer than the first two. I'm pretty new to networking, so I did a lot of reading to understand how ZMQ works. I found two example projects that really helped me understand direct messaging and PUB-SUBs. I had my client send a "-1" to the server to indicate it is a new client. The server would read this in a thread and give the client an ID. I was initially confused as to how I could print my iterations and also wait for a new client to join if needed, but it became apparent that threads are going to be very useful in this project. I used a map to keep track of client IDs and iterations to easily reference them later in my loop (this idea would be useful in my engine for storing client IDs and strings of my objects). The server would publish all iterations to every client so they could get updated at once. I probably would not have thought to use a PUB-SUB if it were not suggested to me by some of my friends. I later used the two files I created for reference in part four.

Section 4: Putting it all Together

- This section was the absolute bane of my existence for about two weeks (can I say that in the reflection?). I initially started this section with a server-heavy design, which, in retrospect, was a horrible idea. I wanted to receive input from the clients and handle all the input, collisions, and time on the server. I would then simply send the positions to the client and draw each object. I soon realized that this was incredibly slow even for one client and had trouble with more than one input at a time. I was still very hung up on managing more in my server than I should; I would often try to debug zmq logic using a server window.
- I had to completely restructure my project once I realized that this was supposed to be a client-heavy project. I first started moving everything I was calculating in the server over to the client. I realized the server is mainly there to publish the entirety of every client string to each other, but should not handle input, collisions, or other calculations. Like part 3, I do have a thread that checks for new clients in the server. My client had trouble for a while since I forgot to set the socket, which caused my client to wait indefinitely. It was sort of simple to set the client up the way the server way, but getting clients to communicate with each other was very hard. One issue was that my parse string function was not creating new characters for older clients.
- Creating a synchronized environment was also not easy to do. I went through several ideas of whether I should have a server compute an environment or maybe have a client do it somehow. I ended up hosting the environment on my server, where I performed a basic operation. All my clients will get the position on this platform and just set the position as if it were another client. I had to refactor my parse function to accept the position of the platform which took more time than I had hoped. I added a tag to tell my function that the platform position was coming up.
- What I'm most disappointed about that I will work on for the next homework is how the program can get slow sometimes and clients will keep their velocity and not update to normal for some reason when another client joins while it's moving. You must stay still at the moment another client joins. I'm currently working on resolving this issue, but otherwise, everything still works as it should.