Homework 2 Reflection

When implementing the multithreaded loop architecture, I currently have it so that when looping through the logic() functions of my Objects (the things that happen every frame), I have a thread that is dedicated to looping through all the PhysicsAffected Objects (currently just Character Objects) that I have and one that loops through all the Generic Objects (currently just Platform Objects). The way I make sure no (important) race conditions are meet (in my case this means two objects trying to move and check collision at the same time) is first I created a static mutex object in my Object class so that it will reach all of my game Objects when threads loop through them and will be locked out of functionality when necessary. I then create unique_locks in both PhysicsAffected and Generic overridden move functions so that as the object tries to move and checks for collision, no other objects will be moving past the point of detection for our collisions. Then all of the threads join and the objects are drawn (or rather information is then sent to the client and drawn there in my Part 4 implementation).

When implementing the Timeline in my game engine, I created a Timeline class that holds the start_time, current_time, last_time (the current_time from the last iteration, used for delta_time), tic step, and other variables used in functions to calculate the previous variables. There are two constructors, the default constructor that holds purely the time gathered from the sf::Clock object that SFML provides. The other constructor takes in a pointer to an anchor Timeline (the one that is made using the default constructor, this could possibly be statically constructed and be used as default for all additional Timelines but as I'm not super familiar with that I decided against it, could be a minor change if it seems tedious later, though this could also seem tedious if constructed statically, so again, not sure). The timelines have a setTime() function that needs to be called every iteration (of something, each could be called separately but the default Timeline would need to be called for that iteration (again maybe better to have with static functionality). There is a getCurrentTime function that merely gets the current_time, not really utilized but why not. A deltaTime function that is used to determine how far an object moves (this is something that needs to be called by the user for calculations depending on whether they want them to be time bound. The function is implemented by calculating the average of 5 previous delta_times. Then I have a pause and unpause function that merely changes a boolean within the class. When this boolean is true (the game is paused) the current_time and last_time is still kept up with but the delta_times are not messed with, in the engine, there is a simple conditional that checks isPaused, which reports the paused boolean in Timeline, and if it (the desired Timeline) is paused, the main game loop is skipped. In order to implement the change in speed of time, I implemented a setTic function that changes the current tic to the new tic and sets all delta_times accordingly so that they don't still operate on the previous tic resulting in a large variance of values, and a cycleTic function that iterates through the three speeds: 1, 2, and 0.5 through the setTic function. (This is in the README but pause is "esc" and changing speed is "p", yeah that might be confusing but whatever). Probably should note how I access these individual timeLines across files. I created a TimeManager singleton class that will be able to find the individual Timeline classes as they are needed throughout my code. Just simply add Timeline and get Timelines functions implemented.

When figuring out ZMQ [1] I created a separate client and server file that is able to support any number of clients and sends a string in the form "Client X: Iteration Y". When connecting to the server, I have a loop that has a REP socket in the server listening for any new connections, the client (with a REQ socket) sends that it has connected and the server then replies with its client number and stores what iteration it arrived in. Then in the second part of the loop, I iterate through a vector of ints that represent the iteration each client started inside and I send each variation of "Client X: Iteration Y" through a PUB socket, which the client will receive its own string as it knows its client number and subscribes to the respective string "Client X" and then prints it on its end.

When incorporating ZMQ into my engine to allow for multiple clients to connect, I used a similar approach but used threads to handle the separate tasks that need to happen semi-simultaneously. The main part of the engine handles what it normally did with the original engine loop. Before that main loop is reached, I make a thread that handles client creation and within the loop, it looks similar to what I did for part 3 and have a loop with a REQ/REP exchange that listens for any new client connections and then creates another thread to handle the back and forth messaging between that one client. In this new thread, is another REQ/REP connection that is unique to that client, the client sends data of its inputs to the server and the server replies with the current positions of all the objects (this definitely is not the most efficient implementation of this functionality as there could be a push/pull socket to retrieve and differentiate the input from clients and then have a PUB/SUB to send all the position data once instead of for each client, this could be explored in HW 3). The input is put into a vector of strings that hold the inputs for each client and that vector is used to update the respective characters by parsing the string and performing the desired actions. The way this works is that the game engine will continuously handle input based on the last given set of inputs received from the client until the client sends an updated input in which the input handling will use that until that input has been updated. Object now has two new functions, one to create string data based on its position (for server use), and another to parse the input and update the position of the object accordingly (for client use). The client's loop takes the input and converts it to a string and sends it to the server, then receives each object's string separately and calls the function to be parsed, then it draws the objects. Whenever the vector of inputs is being handled, objects being updated, or object information is being sent, a mutex is locked as all of this data works together and without it some interesting data could be sent or errors in updating the objects could occur.

Right now it seems a bit laggy (I'm also on a Surface Pro 6 but idk if that'd be why) and a possible solution could be making a new socket for the engine to publish the information to the client instead of sending it to each client individually. Though I'd still need some unique socket (to my knowledge) in order to receive the input from the client, however that seems to take up much less computational time.

Appendix

[1]

Also this website is banger epic, just wanna give credit:
https://brettviren.github.io/cppzmq-tour/index.html