HW2 Game Engine Foundations

Karan Rakesh (200316009)

Section 1 :

I built the timeline as per Dr Roberts' design that he had outlined in class. I implemented step size as a parameter that can be altered, but also experimented with using a GameTime pointer to attach the to the timeline with the required step size, but settled with using a function to double or halve the step size. I implemented pausing on click of a button, and achieved it by marking the point in time where the pause began and then at the end of the pause I pulled the timeline back to that point to ensure the moving platform remains in the same position with the same velocity once the game unpauses. I initially considered the idea of only counting the pausetime and then adjusting the position of the moving platform accordingly, but found no additional logical benefit in doing that. Also during the pause, no inputs are registered and so the players don't move. I have integrated pausing into my final solution. But since in the final solution I have handed moving platform control to the server, it will not show up in the solution. The changes can still be inspected in the myTime.h header file under start_pause() and stop_pause(). Changing the speed of the game has been implemented by modifying the tic size as mentioned earlier.

Section 2:

This part of the homework was the most time-consuming and arguable the hardest part since setting up ZMQ and getting it to work was not at all a straightforward task. The documentation was not updated for 4.3.1 and that also caused errors. It was especially hard (for me) to get the strings to send, as I was getting garbage values on using memcpy() and I eventually got it to work with sprintf(). It also took some careful debugging for me to find out that I was supposed to allocate one bit more than I was to prevent the null pointer from being excluded on the receivers side despite the ZMQ documentation insisting that they do not transmit the null character while the send msgs in their documentation.
On carefully reading the entire documentation, I was able to get REQ-REP working after many days of trial and error. Initially I was able to get the Hello World client-server model running and then to extend it to multiple clients receiving their own msgs was very straighforward. The challenge came in storing the msgs to be sent to all clients on the server side and sending it out to all clients. After experimenting with strings and vectors, I eventually settled on using maps to store the key (id) and value (iteration number) for all the clients whenever they sent a req and sending the entire updated map back to the client as a reply. This helped achieve the required functionality from part 4.

Section 3:

So with my first two sections I built a solid base of a game that had the required functionality and a server that can handle multiple client requests and also store client data to be sent to multiple clients. I only needed to modify this existing functionality to incorporate the game elements. My design goal for this section was simple, build a server that govern the position of the moving platform and broadcasts the position data of every client. The main design decision that was required to be made was whether to do all the client's motion processing on the client-side or server-side. One of the reasons I picked the client-side was because it made sense to minimize the load on the server by carrying out the processing on the client-side itself. Another reason that pushed me to develop it on the client-side was since the whole game so far was built as a client-side application, it required the least amount of code modifications, thus reducing the probability of errors creeping in. Another major decision to be taken was whether the exisiting REQ-REP would suffice or whether there was a necessity to go for a PUB-SUB. I decided to test it before making any drastic design changes and to my delight the REQ-REP worked perfectly and handled all the msgs with ease. Probably in a system with a very high number of clients REQ-REPLY might have been infeasible, but for the scale that we are operating, the CPU gives way much earlier than ZMQ message queues. Also to implement pausing across clients, I just blocked the sending and receiving of any msgs until the game is unpaused, and hence it resumes with the moving platform at the current position on server as well as the updated locations of all other players as well. One noticeable glitch that I have noticed is that since I halt all updates to character object when the screen goes out of focus is that the character may remain stationary on the platform level even though the platform moves out from under him. Since no updates are being processed or sent to server when the window goes out of focus, to ensure only keyboard inputs are registered by one client at any point.

Section 4:

This section suggested the use of multithreading which I tried experimenting with, but I showed Dr. Roberts my thread-less implementation that satisfied all the required criteria for the 4th section without having to leverage the functionality of threads. The major reason I felt using threads were unnecessary was because ZMQ is made to be lightweight and efficiently enable multiple connections over the same socket and handle them without any issues (especially at the small scale that we are developing for). Hence I tested my basic solution for the functionality in the section and my prior design decisions ensured minimal code changes to satisfy section 4. To ensure the situation of the slower speed clients also having a slower reading/writing, since I figured that having a slower messaging rate would directly affect the rate of refresh of the entire game, I accomplished the same functionality by sleeping on the client side for durations which were inverse of the step size. Getting the clients to message at exactly half the rate is nearly impossible due to the variability in the system configurations i.e. when it is plugged in vs on battery, device configuration, power modes etc. So, I implemented basic functionality to ensure different rates of sending and consuming based on the client's selected speed.

Note: I did not capture any screenshots while I was encountering the large number of errors that had as I was busy debugging them, so I don't have any appendix per se to add at the end. I

believe my design decisions paid off in terms of supporting existing as well as future requirements(HW3 arbitary no of connections at arbitary times). I also believe my comprehensive study of ZMQ helped me exploit the power of ZMQ to tackle the final section of the assignment without requiring multithreading and I’m glad that Dr Roberts allowed and appreciated it.