Project 2: Time and Networking Foundations

Overview

Your task for this assignment is to explore how to represent time and the basics of constructing a multithreaded network server. This assignment has team parts and individual parts. As always, you are expected to abide by the University's Academic Integrity Policy (http://policies.ncsu.edu/policy/pol-11-35-01), which includes providing appropriate attribution for all external sources of information consulted while working on this assignment.

There are 125 points available on this assignment. Students enrolled in 481 are required to complete the first 100 points (Sections 1–5) but may also choose to complete Section 6. Students enrolled in 481 earning scores above 100 will receive a 100 on the assignment (i.e., extra points will not be given as extra credit unless explicitly mentioned). Students enrolled in 581 are required to complete all 125 points and will receive a grade as the percentage of the 125 points they earn. All students are required to submit a writeup addressing the sections of the assignment they have completed.

Section 1: Measuring and Representing Time (25 points)

The first part of this assignment is to implement an explicit representation of time. You are to incorporate a Timeline class that is flexible enough to represent both real time and local time, can represent time at varying scales (real time, game time, loop iterations, etc). Your class must support an adjustable tic size and enable you to anchor your timeline to another, arbitrary timeline (or to a measure of real time). Furthermore, your timelines must support pausing and un-pausing.

To demonstrate your representation of time, you must incorporate the following functionality in your engine, demonstrable in your game:

1. [Engine Feature] Object movement based on elapsed time: you need to maintain a timeline object that covers "game time", and any moving object should update its position based on the elapsed time of the timeline object.

2. [Engine Feature] Pause and un-pause: Your timeline class in the engine should allow you to enter a paused state, and all objects which depend on the timeline for movement should remain stationary. Another function should exist to un-pause the timeline, and the objects should begin moving again from the position they were at during the paused period.

3. [Engine Feature] The timeline should be able to change its scale from 1.0 (real time) to .5 speed and up to 2.0 speed. This effect should be realized without making any changes to any other objects in the game.

4. [Game Feature] The pause and un-pause feature, as well as the timeline scaling, should be switched on/off or changed by using keyboard input through your input system for Homework 1.

Section 2: Client-Server (25 points)

For this part of the assignment your task will be to implement the 0MQ client-server setup built in Section 2 into your game engine. The server will be responsible for coordinating clients; however, rather than sending messages, the server should ensure that any movement of the character object on one client gets reflected on other clients. The server should run headless (no SDL window), but the clients must have an SDL window. Your code from Homework 1 and the previous parts of this assignment are likely a good starting point for this effort, since all clients must display the same environment, and have the same characters, with one character per client.

As in section 2, your program must handle at least 3 simultaneous clients, each of which should run in their own separate process (no shared memory allowed), and the clients should be able to be started at any time. Successful implementation of Section 3 sufficiently demonstrates Section 2.

Section 3: Multithreaded Loop Architecture (25 points)

This part of the assignment is to build upon the ThreadExample provided on Moodle to make your main game loop multithreaded. You should have at least two threads handling distinct subsets of the regular update tasks. For example, one thread may handle moving platforms, and another thread could handle the keyboard input and resulting character changes. This is a suggestion; you may divide how you see fit. Take care to be thread safe, using getters and setters with mutexes for data access as an example, whenever necessary.

Section 4: Asynchronicity (25 points)

Your task for this part of the homework is to make your server design fully asynchronous, to support clients running at different speeds (i.e. able to leverage the functionality from Section 1 to increase or decrease the speed of a client's main game loop, thereby doubling or halving the rate of 0MQ messages) and the correct engine functionality should occur. In other words, slowing down one client should not make all other clients slow down as well. Similarly, increasing the speed of a client should not make other clients run faster. 0MQ does provide semantics that will give you this for free (i.e. Router or Dealer) but you may not use them for this assignment. You must create these semantics using multiple threads with synchronous communications from server thread to client process.

To accomplish this, you'll need to pay attention to two things:

1. Moving platforms should be governed by the server and be in the same place for all clients, so you will probably want to tie your character object to a timeline rather than the moving platform.

2. You must have threads on the server dedicated to reading/writing to each client.

Section 5: Peer-to-peer (25 points, 581 required, 481 optional)

Your task for this part of the assignment is to incorporate peer-to-peer networking for your game engine. This model does not relay information from one player to another using a server, instead the peers directly update the others with their player information. There are two options for this section of the homework:

1. You may start with a centralized server which synchronizes the data and transition the clients to peers in a hybrid model which uses a server to control common details of the simulation such as moving platforms, while the peers send each other player data directly. This server could be dedicated (a separate server holds authority) or a listen-server (a server hosted by one of the players holds authority)

2. ***[+10 pts extra credit for 481/581]*** You do not use a centralized server to synchronize the data, and instead negotiate positions of objects with no centralized authority for the game. This will likely be challenging to design and implement. You will need to consider how to synchronize moving platforms with no singular authority on where the platforms should be at any specific time.

Writeup

Now that you have designed and implemented a number of engine components, write a 1-2 page paper summarizing your work, and the design your team made. What did the design enable, why did you and your team make the design decisions you did, what will those decisions enable in the future? What worked the way you wanted and what didn't? Successful writeups will contain evidence that you have thought deeply about decisions and implementations as what they can produce and have gone beyond what is written in the assignment.

As an appendix to your paper, please include relevant screenshots of your program and support your points by referencing the screenshots. These screenshots do not count towards the page requirement. Your points from the writeup represent 50% of the total grade for the assignment.

Submission

There are three deliverables for this assignment:

1. A team submission of the game engine. This is a single submission for all team members, and it should include the general features used in each team member's game.

2. Individual game submissions showcasing engine features. These submissions are per student, not per team, and are graded individually.

3. An individual Reflection writeup, which discusses the design decisions and choices made in the games and engines.