CSC 481/591 Fall 20 Homework 4

Due: 10/28/20 by the start of class

Overview

Your task for this assignment is to add more functionality to your game engine. Specifically, you will focus on the addition of an *event system* and *replays*. As with previous assignments, you will be using your existing engine code to add additional functionality. You will submit your code and a writeup of your results, including screenshots where appropriate. This is an ***individual assignment***, you are to work alone. 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 Parts 1 & 2 and the writeup, but may also choose to complete Part 3. Students enrolled in 481 earning scores above 100 will receive a 100 on the assignment. Students enrolled in 591 are required to complete all parts (1–3) and the writeup, and will receive a grade as the percentage of the 125 points they earn. All students are required to submit the writeup addressing the sections of the assignment they have completed.

Part 1: Event Management System (35 Pts)

For this part of the assignment, you are tasked with adding an event management system to your game engine. You must queue events, and must allow different priorities for events. In order to make your priority system work, you will also need to keep track of game time and time stamp all of your events accordingly (*i.e.*, you will need a functioning timeline implementation). Note that in addition to your *event management architecture*, you will have to implement an *event representation* that will allow you to easily raise and handle events. Further, due to the networked architecture of your engine, take care to ensure that your events are `Serializable`, or can be shared across the network in some way. Note: this requirement does mean that your event objects (should you choose to use objects) must be able to be sent over the network; it is perfectly acceptable to use any protocol of your choosing provided the event information can be re-constructed by the receiving process.

You will need to write an *event manager class* to keep track of *which engine systems or game objects are registered* to receive events of arbitrary types (registration), *receive events* after they have occurred (raising), and *dispatch those events* to the appropriate objects or subsystems at the appropriate time (handling). You must make use of your `Timeline` class in event management (most likely during raising to timestamp and during handling to determine the current time and which events should be handled).

To demonstrate your event management system, create *at least* four types of events: *character collision, character death, character spawn,* and *user input*. Reimplement each of those functions of your game from Assignment 3 using your event representation, event raising, and event handling—in other words, using `sf::Event` in lieu of your own event representation and associated functions is not allowed. You must demonstrate at least one example of events being sent across the network. For example, you could use a `ClientConnect` event whenever a new client joins the server which would be raised on the client, but handled on the server. You have freedom to decide how you want to meet this requirement—it does not have to follow this example.

A few suggestions to ponder before writing any code:

1. Consider whether or not integrating an event handler into your game object model makes sense or not.

2. Put some thought into the interaction between your network architecture and your event management systems. In particular, if an event occurs on the client, where does it get queued and handled? For some ides to seed your planning, read ahead to Part 3 of this assignment.

Some questions to think about for your writeup: How and why did you implement and represent events? What changes to your design did you need to make to accomplish this? How does your design promote reuse and extensibility?

Part 2: Replays (35 Pts)

Your next task is to use your event system and timeline capabilities to implement a replay system. You should be able to press a button while controlling a client to begin recording a replay, another button to terminate recording, and then select from at least three replay speeds ($\frac{1}{2}$ time, normal time, and double time). Note, the *starting and stopping of recordings must be handled as events* (*i.e.*, there are at least two additional event types required for this part of the assignment).

You should support changing replay speeds (among, at the minimum, your three defined speeds) at any time during the replay. When the replay is done, the system state should return to where it was prior to the replay and the engine should continue. To simplify things, you do not need to display the replay on multiple machines; however, you do need to have the characters from multiple clients reflected in the replay. Further, to simplify things, it is not necessary for character or object movement to pause on all machines during a replay on one client. Further, the character controlled by the client displaying the replay should remain in a fixed location on other clients.

Hint: it may be advantageous to refactor your code so all movement (either from characters of other moving objects) generates an event, which may require you create yet another event type.

If your event management system is implemented with the appropriate flexibility, you should be able to record events in a log file or memory buffer and replay the list of events by raising events according to the correct (local) timeline. Here are a few things to consider:

- It may be advantageous to support "*wildcard*" registration in your event manager, so, for example, your replay system can register to be notified of every event.
- You will need to take care to ensure that the initial state of your replay is dealt with appropriately, perhaps by teleporting characters and objects on the screen (which could also be events!). You will also want to save their location so you can teleport them back after the replay is complete (another type of event?).
- You will also need to make sure that you timestamp events appropriately, so that you can control the timing of their replay.

Pay close attention to how your represent time. If you are successful at these tasks, your event management system should "handle" the rest and replays should require little more effort than teleporting objects, reading the log of events, and raising them at the appropriate time.

Note: You will definitely need to be careful about event chains, where handling of an event triggers another. In those cases, the replay event and the event raised due to the chaining may conflict. That's a boundary case you need to design for explicitly. You may find an explicit "age" in your event representation useful for this application.

Some questions to ponder for your writeup: Were you successful at leveraging your event system for replays simply, or did it require a lot of work? What changes to your design did you have to make to get it working?

Part 3 (591 required, 481 optional): Performance Comparison (25 Pts)

Your task for this part of the assignment is to compare two schemes for event management:

1. Server-centric: In this design, event registration, event queuing, and the scheduling of event handling should be done on the server. Events can be raised anywhere (server or any client), but should be sent to the server for handling and clients' states should be updated based on the results.

2. Distributed: In this design, both the client and the server are to implement all features of event handling (registration, queuing, and scheduling handling). In this scenario, any machine (server or clients) can raise events that are sent (possibly via the server) to all other machines to be handled.

The intent behind this part of the assignment is to do a performance comparison, much in the spirit of Part 4 of homework 3, whereby you should experiment with different numbers and locations of events to get a sense of the conditions under which a *server-centric* or a *distributed* design for event management performs best. Some things to ponder: What metrics will you collect to characterize performance?

Writeup (30 Pts)

Now that you have designed and implemented a number of engine components, write a 2-–3 page paper summarizing your design. That is a minimum of 2 *FULL* single-spaced pages. It is **strongly** suggested that you do not limit yourself to only answering the questions posed in this assignment. Think creatively about what you have done. Why did you make the design decisions you made? What did they enable for this assignment? What will they enable in the future? The most successful writeups will contain evidence that you have thought deeply about these decisions and implementations and what they can produce and have gone beyond what is written in the assignment.

As an appendix to your paper, please include all relevant screenshots to support your discussion. The appendix does not count toward your 2—3 page requirement.

What to Submit

By the start of class on 10/28/20, please upload to moodle a .zip file containing your code, a compiled executable of your system, a README file with compilation and run instructions, and a .pdf of your writeup. Please make sure your README not only addresses how to run your game, but also how to play it (*i.e.*, what buttons to press to move the character, change a replay speed, *etc*.).