CSC 481/591 Fall 18 Homework 1: Networking Foundations

Due: 10/2/18 by the start of class

Overview

Your task for this assignment is to explore the basics of constructing a multi-threaded network server. This will be the first part of your game engine that you will develop throughout the course of the semester. As part of the process, you will familiarize yourself with the Processing environment, which you will also use for the rest of the semester. 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 the first 100 points (Sections 1–3), but may also choose to complete Section 4. 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). Students enrolled in 591 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: Processing (25 points)

You should begin by familiarizing yourself with Processing (`http://processing.org`). "Processing is an open source programming language and environment for people who want to create images, animations, and interactions." Processing contains an IDE and runtime execution environment. If you're more comfortable, you can create Java classes that extend the base Processing classes to work in your preferred IDE. Bear in mind that we will need to compile and run your code to grade it, so make sure there are no special requirements that might prevent us from doing so. You are to use Processing version 3.4, which is the current version at the time of assignment. Specific instructions on how to build a processing `PApplet` with *Eclipse* can be found here: `http://processing.org/learning/eclipse/`.

Take some time to play around with Processing and familiarize yourself with the basics. You should implement code for the following:

- Draw squares and rectangles of different colors to the screen.
- Collision detection between objects. This doesn't have to be anything fancy or efficient. Just get something working.[1]
- Move one of the squares around the screen in response to keyboard inputs: left, right, and space to "jump". The physics doesn't have to be realistic, but you should have a "jump" that maintains left or right movement, and doesn't continue off the top of the screen.

Section 2: Networking Basics (25 points)

For this section of the homework, your task is to familiarize yourself with Java's basic networking capabilities. You'll need to focus on two classes in the `java.net` package: `ServerSocket` and `Socket`.

[1] Hint: The java.awt.Shape interface, for which there are many implementing classes including Polygon, Rectangle, *etc*., provides a .intersects() method. If you convert your Processing shapes into java.awt shapes, collision detection should be straightforward.

Additionally, to demonstrate the capabilities of your system, you will want to send data back and forth between client and server using classes from the `java.io` package. You will want to pay particular attention to the `java.io.Serializable` interface.

Once you've familiarized yourself with the appropriate classes, create a simple server that accepts incoming connections and reads in data of your choosing from a fixed number of clients. Test your code for different numbers of clients. *Note: You may choose to accept an arbitrary number of clients, even if not required for this section of the assignment.*

Section 3: Putting it all Together (50 points)

For this part of the homework, your task will be to multithread your server, using asynchronous design, so an arbitrary number of clients can connect. From your main server program, create a new `Thread` to listen for incoming connections on the `ServerSocket`. This thread should then asynchronously add clients (or the socket streams associated with clients) to lists in the main program as they are established. Make sure to use synchronization appropriately. And, more importantly, use the thread-safe collections (see `java.util.concurrent`).

Lastly, start up some number of clients (to be determined at runtime) to send data to and receive data from your server. This communication should be synchronous (meaning all clients send a certain amount of data and read a certain amount at each iteration). For this assignment, you may send whatever data you choose; however, in future assignments you will be expected to send game objects (like rectangles) from clients to servers, so a little extra effort here may pay dividends in a few weeks. Hint: if you plan to integrate your client/server with Processing for this assignment, make sure your main server program is a processing sketch so you can draw things sent from your client, and carefully review the `java.lang.Serializable` interface and it's relationship to `java.io.ObjectInputStream` and `ObjectOutputStream`.

Section 4: Asynchronicity! (25 points, 591 required/481 optional)

Your task for this part of the homework is to take your asynchronous design one step further. Whereas in Section 4 of this assignment incoming connections were handled asynchronously and communications with established connections were handled synchronously, here both will be handled asynchronously on both the client and the server. You will want to have a main thread for your main game loop, a thread for accepting incoming network connections, and two additional threads for each client—one to read incoming data and one to write outgoing data. The incoming data should be read and made available asynchronously for use in the main game loop, and outgoing data should be delivered to the outgoing thread and transmitted asynchronously. Your client should have a similar architecture, but won't have the thread for accepting incoming connections. Like before, your system should handle an arbitrary number of incoming connections.

Writeup

Now that you have designed and implemented a number of engine components, write a 1–2 page paper summarizing your design. That is a minimum of 1 FULL single-spaced page. 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? What didn't work out the way you had hoped, and why? 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 1–2 page requirement. Your points for the writeup will represent 1/2 of the points allocation for each of the above sections (*i.e.*, 25 of the 50 points for the "Putting it all Together" section will come from your writeup).

What to Submit

By the start of class on 10/2/18, please upload to Moodle a .zip file containing your code (in multiple subdirectories if you have different versions for different Sections of the assignment), a README file with compilation and execution instructions, and a pdf of your writeup.