COMP-4300: Final Project
University of Manitoba
Prof. Sara Rouhani
Zachary Kolton
7838513
December 14, 2022
GitHub Link: https://github.com/zacKolton/comp-4300-term-project

Table of Contents

Project Summary

The basis for this project came from a memory of a scene in the 2015 movie “Project Almanac”:

This depicts a student showing his friend a program he developed to “bounce” the balls back and forth between 2 computers/tablets. While it doesn’t look exactly like the movie, this project aimed to mimic the concept of “seemingly seamless” motion between multiple displays.

There is one server for *x* number of clients, essentially in the form of a centralized network, which sends and receives information to and from each client. Specifically, the server’s job is to make sure the “ball” gets from one display to another. The client’s job, aside from drawing the ball, is to tell the server where the ball should go next.

Thus, when the ball gets near the edge, it either bounces off or needs to go to the adjacent display. So, it would send a message to the server, and the server would instruct the next display to draw a ball. In a nutshell.

Technical Components

Environment: Processing 
Language: Java

How to Run

1. Download Processing
2. Configure `settings.json`
3. Start the server
4. In the same order as the `computer-configuration` array, start the clients
5. Confirm all clients are running (and waiting)
6. Click on the server drawing pane to begin

Files

- `computer.pde`
- `sketch_server.pde`
- `settings.json`
- `README.MD`

Multiple instances of `computer.pde` can be run. Whereas, making multiple files tailored for each computer would have been significantly more work and difficult to maintain. This saved a significant amount of development time.

In contrast, `sketch_server.pde` is only made for one instance of itself.

Why Processing?

COMP-4300 is a computer networks course. Naturally, “networking” was this projects’ focus. However, the only way to actually *see* it in action was to draw it. Unfortunately, the effort it took to comprehend Java’s regular drawing library did not justify the use case. By the time developing the animation was done, the projects timeline would have been nearly over.

Processing took care of that.

Additionally, it also had networking capabilities which fit perfect for this project.

settings.json

This will go over each setting in the file.

Setting Name	Description
`window-width`	The width of the display (when running)
`window-height`	The height of the display (when running)
`ball-radius`	The radius of the ball
`port`	Port number for server and clients
`x-speed`	Speed of the ball on the horizontal plane
`y-speed`	Speed of the ball on the vertical plane
`ip`	IP address for the server and clients
`frame-rate`	Refresh rate of the displays
`computer-configuration`	An array of JSON objects that define (for a client) - Name (the hypothetical name of the computer) - left (the computer to the left of it) - right (the computer to the right of it) - in-use (Is [this] computer already initialized)

computer.pde

This will go over the functions in the computer/client file.

`void setup()` – Processing core function, gets executed before draw
`void draw()` – Processing core function, continuously loops to refresh the display
`void drawBall()` – Draws an ellipse

`void askForConfig()`

This function is called in `void setup()` and it is also a part of what allows the "multiple instances" aspect.

The computers/clients need to differentiate themselves from another. This is done with "`computer_name`". It's how one computer/client doesn't take messages from the server if it is not intended for them.

Processing's Server class can only "broadcast" to all clients. There is no way to specifically refer to a client.

Thus, `askForConfig()` sends a message to the server saying, "what are my settings". This would include its name and its "neighbors'" names to the left and right (if any).

`void notifyServer(float, float, int, int, String)`

When the ball on [this] display is moving into another, this function notifies the server. It sends the position, direction, sender (its name), the target (it's neighbors name), and the request type.

`void clientEvent(Client c)`

This listens for a message from the server. Then, it will check if the message is for itself. Otherwise, nothing happens.

For instance, when a ball is *incoming*, it would receive the coordinates for where it should print on its display.

`void applySettings()`

Strictly for readability. This is called in `void setup()`.
Notably, this is also where the computer/client is given a temporary name so that the server can refer to it later. At which point it would provide its "real" name. Defined by the user in the settings file.

sketch_server.pde

This will go over the functions in the server file.

`void setup()` – Processing core function, gets executed before draw
`void draw()` – Processing core function, continuously loops to refresh the display
`void mouseClicked()` – Processing core function, starts the animation by clicking the drawing pane

`void startAnimation()`

Sends an instruction to the first computer to begin the animation.

`void applySettings()`

Strictly for readability. This is called in `void setup()`.

Communication Overview

This section will go over (in more detail) how the communication works.

The IP address used is the one from my (home) local network. The port number comes from an example on the Processing.org website.

When a computer/client starts, it sends a message to the server requesting a given name and the other computer/client's names it needs to know. The server differs to `settings.json` and looks for the first computer/client that is not `in-use`.

The server then broadcasts out the requester's *new* information. Other clients see it but ignore it since they are not the *named* receiver.

Essentially, anytime the server is broadcasting, all clients receive the message. Anytime a client "broadcasts", only the server receives it. This is why each client needs some sort of identity.

Final Remarks

This section will address some closing thoughts on this project.

Reflection
I can't help but notice how analogous (albeit generalized) this project is to a real-world centralized system. Possibly a certificate provider or even "ordering" computer time way back before I was born. The client has to *ask* for information that which only the server is allowed to read. You cannot connect more than the computer-configuration defines. It has to be specified.

In a nutshell, the client draws and the server computes.

Challenges
- How will this computer to talk to that one?
- How will the clients know if they are supposed to draw?
- Access permissions issues with GitHub
- How will a client know if there is a computer to its left or right?

Original Idea
The original idea with this project was to have multiple balls whipping around everywhere. I wanted to see what would happen if I kept on increasing the number of balls, which would mean number of requests and responses.

How would the server handle that?
Would there be mix ups?
What would it look like if balls bounced off of one another?
Do some balls stay in one pane/client than another? Could this be due to a network issue? Coding flaw?

Practical Aspect
Obviously, in the real-world, it is hard to see what practicality this would have. Simulations would be cool. Maybe even for a networks course…