Metadata

A7, Distributed and Concurrent Programming

Ben Grass (bdg83), Andrey Yao (awy32), and Cameron Russell (cfr59)

Operating system developed & tested on: macOS

Summary (50-300 words)

For assignment seven, the final assignment, our design proved to be surprisingly ordinary compared to previous assignment. We added one additional server class and modified our existing GUI build in A6 to conform to the updated specifications. The most significant issue we ran into was accidently deleting the completed ring buffer code a mere four hours before the assignment was due. This proved to be a major setback. Additionally, determining how to retrieve the specific subsection of the world to redraw on the canvas was particularly challenging, perhaps because it was a more mathematical task. As for testing, making sure both the server and clients behaved according to the API was time consuming, as restarting both was a tedious process. Currently, there is no missing functionality, however there are a few known bugs, which are detailed at the bottom of this overview.

Specification (10-500 words)

At first, we were using SeverSockets and Sockets to implement our server and clients, however, we quickly found that Spark provided all this functionality and much more, so we appropriately modified our implementation strategy. All the post, get, and delete request functions have been abstracted into their own methods, with a single method repeatedly calling them. This makes the server code far easier to understand.

As for the client, we also added a client class that stores important variables that allow the user to quickly connect to the server. This class also provides methods that allow the client to send requests to the server and to retrieve important information. This choice made the most

intuitive sense, as adding all the functionalities on top of our GUI controller appeared too cumbersome.

One unique feature of our simulation is its ability to become full screen, with the canvas and controls scaling properly (it also looks cooler this way). Finally, because the client is now required to connect to a server (well, not technically, but we did not provide offline functionality), we added a cool login page for the clients to easily connect to the server before loading a world simulation.

Design and Implementation (150-1000 words)

This assignment resulted in the creation of two significant classes, the server and client classes, as well as a main class that allows for the creation of a jar file. Moreover, we used the model, view, controller design paradigm, as a distributed application with many game-like properties naturally lends itself to this design. Our GUI represents the view, and in order to update the state of the world, a request is made to the controller via the server. The controller then makes requests to our world model, which are propagated back to the client without exposing the details of internal operations.

In terms of data structures and algorithms, we implemented Dijksta's algorithm to provide the critters with a smell capability and the server uses a hash table to store key information regarding the clients that are connected to the server. Moreover, we made some performance trade-offs for the sake of code simplicity. Our primary objective was to meet the project specification and this priority came at a performance cost, as many functions are not optimized.

For our implementation, we primarily coded with a bottom-up strategy, testing the server and client as we built. This allowed us to ensure the code we wrote was functional, minimizing the frequency of code rewriting. Also, because this was an extension to the previous three assignments, taking a top-down approach is almost infeasible. We also fixed several display bugs from the previous assignment, as the highlighted tile no longer disappears when the simulation is running and food no longer suddenly generates on the canvas. In terms of which group members

wrote which parts of the code, Andrey wrote the code that updates the display of the world on the central canvas in addition to writing part of the server. Cameron wrote the other part of the server, handled the code dealing with client-server interactions, and wrote the ring buffer. Ben implemented the smell sense for the critters.

Testing

To preface this, the following is our testing strategy, however, not all went according to plan. To test our completed simulation, we heavily relied on Postman and the simultaneous running of our server and client. Because we implemented the server first (before touching any GUI code), we initially relied heavily on Postman to test every post, get, and delete request implemented on our server, ensuring the resulting response matched the API specifications. After the server was passing our test cases (ideally, we would have automated the testing using JUnit, comparing the response from our server to that of the server set up by the course), we moved on to testing our client. This proved to be more challenging and time consuming, as running and re-running the client was tedious. However, we primarily made sure all the functionality was working, by loading worlds, critters, playing the simulation, pausing it, etc,. After this, we made sure our server could handle incoming requests from multiple clients. To do this, we (us three) ran multiple clients connected to the same server and all made requests at the same time to ensure there were no concurrency issues and to verify that the game state was updating as needed on each of our clients.

Work Plan (20-100 words)

For this assignment, we divided the tasks based on the following description. Ben completed the written problems and was responsible for implementing the smell feature for the critters. Meanwhile, Andrey and Cameron primarily worked on the same task: creating the server class to handle post, get, and delete requests from the client, and updated the GUI to become a client that properly sent/retrieved information from a server to update the simulation and corresponding data fields. Cameron also implemented the thread safe the ring buffer.

Known Problems (1-500 words)

As a forewarning, connecting our GUI with the class server and using postman is going to be necessary to test the functionality of our implementation.

With A7:

- Tile info does not appear
- Our server is broken in a lot of places, as worlds nor critters are properly loaded
- Sometimes there are pauses when highlighting tiles

We unfortunately ran out of time to fix most bugs with previous assignments, but the major ones that are still outstanding:

- Mutations work most of the time, however no mutation is entirely bug free
- Mating duplicates the parents' rules and add new one
- Does not display extra memory locations