Advanced Programming 1 - 89-210 - Ex7

January 26, 2017

Due date: 18.3.17

Intro

In the last exercise, you will integrate GUI - Graphical User Interface - into the system. The idea is to learn the GUI concept, and how to integrate two languages (which are not similar at all, in terms of what we learned in class) into one.

Implementation

The idea is to make minimal changes in the existing code in order to write the integration between the two components. The GUI, which you have an example of implementation in the image below, should consist the following components:

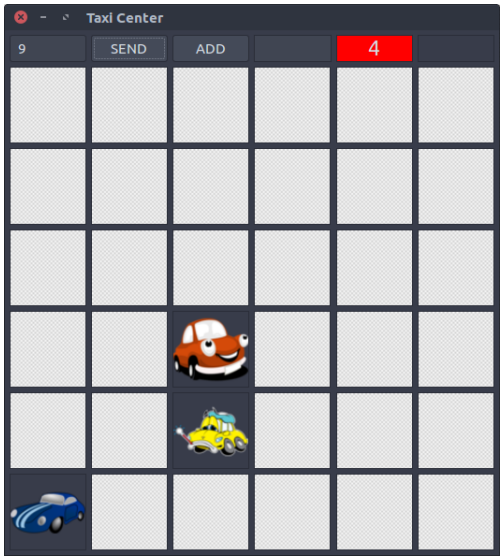
- The Grid - on which the vehicle will advance.
- A text field to insert the commands to the server.
- Another text field to send new drivers from.
- A display label with the current time - starting from 0.
- A display label to print error messages from the server.

The GUI is just another client. It communicates with the server through a socket (TCP in our case) and provide a graphic representation of the grid with the cars inside it. In the current exercise, instead of getting commands from the terminal, it will get them all from the GUI client (almost all of them). The server will be created with the grid information, and the rest will be given through the GUI.

Another thing to be done, is to open the different drivers clients using the GUI client. When inserting a new driver, the java program will create a new C++ program which are the same driver clients from last exercise. One way to do that, is:

```
Runtime.getRuntime().exec(params);
```

Where the params parameter is an array of Strings, which the first index is the name of the client program (which assumed to be in the same folder, with the name - "client.out"), and the rest of the indices are the rest of the parameter of the program.



The Program Process

1. Open the server through a terminal with the standard arguments.
2. Still in the server's terminal, insert the lines which correspond to the map creation.
3. Run the Java program, using another terminal.
4. In the GUI, within the field which is assigned to receive inputs (other than drivers) enter the normal flow
5. If the signal of the drivers was entered (option 1), receive also the expected number of drivers and then:
   (a) Through the dedicated driver window, receive the drivers (you should be able to enter them all together, or one by one)
   (b) Every such entrance should create a new client process.
6. Continue to receive inputs through the main text-field pane until the kill signal.
   (a) If the kill signal was received, take care of sending the server a closing signal, and close every client (as in last exercises) and the GUI itself at the end.

- If an error occur, it should be displayed through the error text label.
- The time should be displayed through the time text label, and get update when relevant.

Running Example

```
$ ./server.out 40000
6 6
1
1,1
```

```
$ java -jar gui.jar 127.0.0.1 40000
```

through the gui text field:

```
3
0,1,H,R
3
1,1,S,B
3
2,1,S,R
2
1,2,2,0,1,2,30,4
2
0,0,0,2,2,1,20,1
2
2,0,0,2,0,1,40,3
1
3
```

through the gui drivers test field:

```
0,30,M,1,0
1,25,D,2,1
2,22,D,3,2
```

through the gui text field:

```
9
9
9
9
9
9
9
9
9
9
9
7
```

- Notice that more than one driver was created, it means that there could be some randomness regarding who takes the ride. That's fine.

Submission:

A zip containing several items:

- All your code in hierarchical way. There should be 2 directories in the top hierarchy: ‘cpp‘ and ‘java‘, along the details file.
- In each directory, put a makefile, which will compile the relevant code (one for cpp and one for java).
- Notice that the makefiles final executable output needs to be in the top hierarchy directory (and not inside the inner cpp/java dir).
- Details file.

Notes

- Don't be late.
- Submit your work via the submit system, with your personal user account. ONLY ONE SUBMISSION PER GROUP. A duplicated submission will be penalized. Same goes for a submission without 'details.txt' file (or deprecated) [in every submission].