Comp 734 - Assignment 1: Distributed Non-Blocking Halloween Simulation Project

Date Assigned: Tue Aug 18 2015

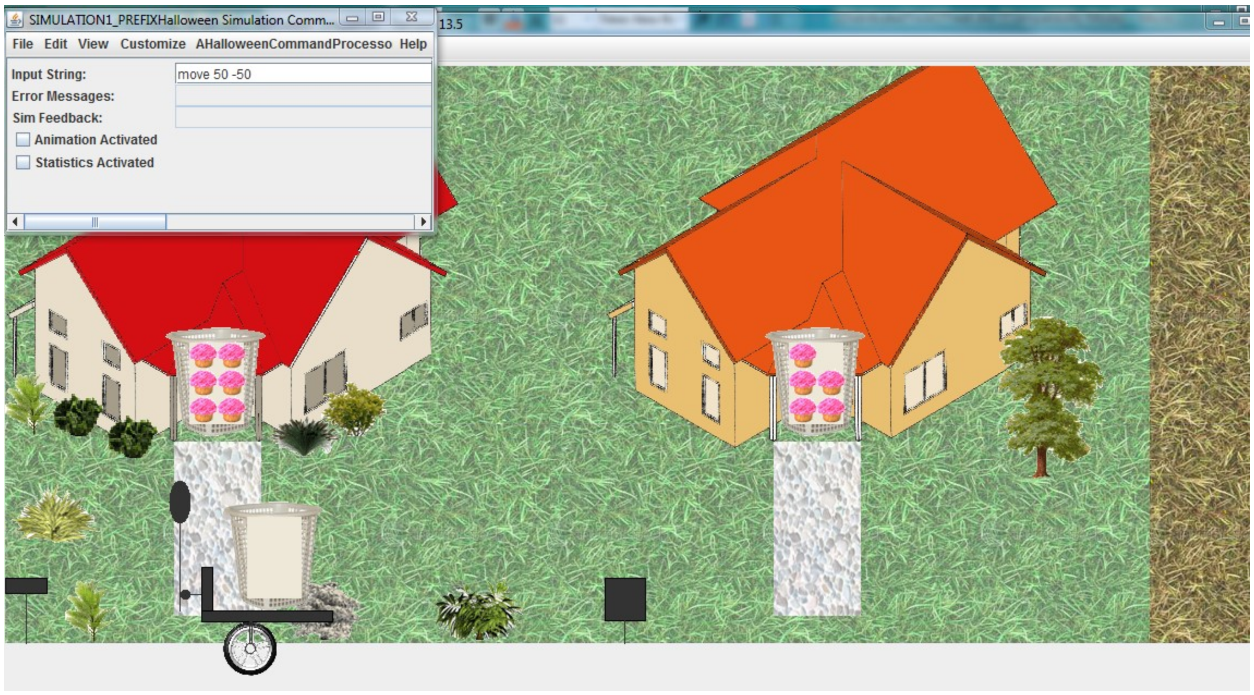
Completion Date: Tue Sep 15, 2015, 11:55pm

The goal of the assignment is allow you to use Java's non-blocking mechanisms to write a "realistic" distributed program. You will take an existing non distributed program - a Halloween simulation - and create a distributed version of it. It will be trivial to interface with this simulation.

Non Distributed Simulation

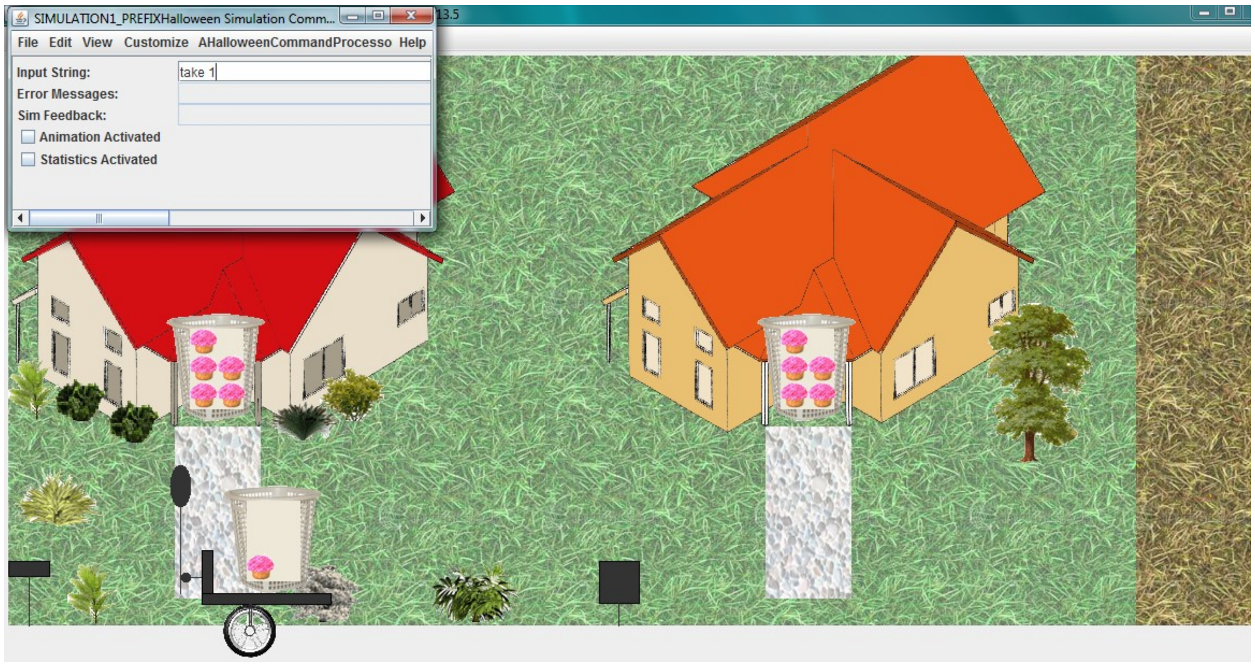
The non distributed simulation is a project I gave to my Fall Comp 401 (http://www.cs.unc.edu/~dewan/comp114/f10/) class. You will work with the code of one of the students in the class – Beau Anderson – who created a particularly nice simulation. The simulation created two windows, a command window, and a graphics window. The command window is used to manipulate the objects shown in the graphics window.

As shown in Figure 1, each graphics window consists of (a) one or more houses, each with a path and a candy container; and (b) a movable avatar with a candy container. The move command can be used to move the avatar in both the x and y directions, as shown in Figure 1(a).


SIMULATION1_PREFIXHalloween Simulation Comm...
File Edit View Customize AHalloweenCommandProcesso Help
Animation Activated
Statistics Activated

Input String: move 50 -50
Error Messages:
Sim Feedback:

(a) Avatar moves into path of leftmost house


SIMULATION1_PREFIXHalloween Simulation Comm...
File Edit View Customize AHalloweenCommandProcesso Help
Animation Activated
Statistics Activated

Input String: take 1
Error Messages:
Sim Feedback:

(b) Avatar takes one candy

Figure 1Beau's Non-Distributed Simulation

If the feet of the avatar are in the path of a house, then the take and give commands can be used to transfer candies between the house and avatar containers. In addition, commands are provided to add and remove a house in the simulation, and undo or redo previous commands. The following is the syntax of the commands:

<Command> → <Move Command> | <Add Command> | <Remove Command> | <Take Command> | <Give Command> | <Undo Command> | <Redo Command>
<Move Command> → move <number> <number>
<Take Command> → take <number>
<Give Command> → give <number>
<Add Command> → addHouse
<Remove Command> → removeHouse
<Undo Command> → undo
<Redo Command> → redo

You can ignore most implementation details of the non distributed simulation. All you need to know is how to trap (using an observer) and execute a command entered by the user. I have created an Eclipse project, Coupled Halloween Simulations, which shows how this is done. The program TwoCoupledHalloweenSimulations in the project couples two simulations in the same process – your task will be to couple simulations in (possibly an arbitrary number of) processes – this means you must run a different program for each process. The project references both Beau's code and a user-interface library I wrote, ObjectEditor, which is used by Beau's code. All three pieces of code are available from the course home page. The Eclipse project must be uncompressed, while ObjectEditor and Beau' code are also compressed, but can be used in this form as they are referenced as external libraries in the Eclipse project. You need to change the paths in the project in order to use them correctly.

The coupled simulations use public methods in the following interface:

```
public interface HalloweenCommandProcessor {
	…
}
```

Two important methods it does not show are:

```
	boolean isConnectedToSimulation();
	void setConnectedToSimulation(boolean connectedToSimulation);
```

Normally, the command processor changes the simulation and then notifies observers. To support automatic broadcast, you will need to disable local processing, and send the command only to the observers. By calling the setter with the false value, you can disable the local processing. This feature is needed in later assignments, but you are welcome to use it immediately as I do in my demos.

Beau created the project using an older version of a library I have written: oeall17.jar. The latest version in oeall22.jar, but it gives warnings that oeall17.jar does not. These can be disabled using the following call:

Tracer.showWarnings(false)

Both versions are on the web site.

Distributed Simulation

You should create a distributed version of this simulation that involves an arbitrary number of processes. The simulations created by these processes are coupled similarly to the way the two simulations are coupled in the demo project – after a user submits a command to one simulation, the command should be executed by each of the simulations in some order chosen by you. You should use the Java NIO library for communication and all I/O including the connection call should be non-blocking. I used the following tutorial to understand it. http://rox-xmlrpc.sourceforge.net/niotut/. You can download and modify the example program or write it from scratch.

You can assume a simulation process does not join a session dynamically, that is, joins a session after a user of some existing process in the session has entered a command. More simply, your implementation need not support latecomers.

Some of the issues you should think about and discuss are:

(a) How do simulation processes join a session?
(b) Are there any processes other than the simulation processes involved in mediating the connection and communication between the simulation processes? You can use a server (the easy approach) or try and create a P2P system.
(c) If you have created a server, how can you share the code your write for the server and client?
(d) What are some of the other design and implementation issues you have addressed?
(e) Trace the sequence of actions that take place in each process when a user command is entered? Be sure to trace the flow between processes and between your objects and the simulation objects in the same process.
(f) How do the interest ops registered with the NIO selector change? Give the transitions.
(g) How do you handle failure of one or more processes in the simulation?
(h) Are there special commands (such as locking and access control commands) entered by the user for controlling the nature of the distributed simulation?
(i) If there are delays, is it possible for race conditions to results in different output for the same input.
(j) If so, how are messages sent by different processes synchronized, if at all?

Future assignments and class material will be addressing issues (f) – (i) in depth. Thus, you are free to address these in a simplistic manner. These have been mentioned here for the more ambitious students who wish to try their own solutions to these problems early in the course. *In yours answers, make sure you present the question before the answer.*

Downloading and Installation Instructions

First get oeall22 install it in some location.

Next get the zip beau_project.zip , uncompress it, and create a new Java project from the uncompressed version. In Eclipse this means Project>New Java Project> , uncheck default location and then browse to the location where the project is downloaded. Right click the project, Properties>Build Path. Go the Projects tab and remove any referenced projects. Then go to the Libraries tab and remove all libraries other than the JRE. In this tab, do Add External Library and reference oeall22. Run main.BeauAndersonFinalProject. Type move 100 -100 in the command window and see if the avatar moved.

Now get CoupledTrickOrTreat.zip and go through the same process as above. However, this time, in the Project tab, add Beau's project to the build path. As before you need oeall22. Run coupledsims. TwoCoupledHalloweenSimulations. Two command windows and simulations should show up. Typing in one window should change avatars in both windows. This project tells you how to take intercept a command from one simulation and inject it in another simulation in the same address space. Your task in this project is to inject the commands in a remote simulation.

Finally clone the following Git repository https://github.com/pdewan/GIPC.git. In Eclipse do File>Import>Git>Projects from Git>Clone URL and paste the link above and follow the instructions to add the project. Now go through the process of removing any projects and libraries in the build path and adding oeall22.

The niotut package contains the code form the NIO tutorial mentioned above (http://rox-xmlrpc.sourceforge.net/niotut/) Run niotut.NIOServer. Next run niotut.NIOClient. You should see "Hello World" in the client console. You can also run niotut.NIOClientGoogle to get a google page echoed back.

Your task is to combine the CoupledTrickOrTreat project with the code in the NIO Tutorial. I would first change the NIO tutorial code to create a collaborative program that echoes HelloWorld from one client to all of the other clients. Next I would change the program to interact with Beau's simulations.

Submission Instructions

For each submission date, create a YouTube video, demonstrating the working of your program, (b) answer the questions posed and submit a printout of these answers in my mail box or in class, and (c) submit your code on Sakai (the assignment will be created before Tuesday). The video can be submitted as a private or public link in piazza. If Piazza and YouTube make you uncomfortable, you can create a shared folder or use email. I used the following software to make videos:
http://sourceforge.net/projects/camstudio/files/legacy/Camstudio2-0.exe/download