PEDESTRIAN SIMULATION

Java for User Interface and Networking

Abstract

An individual project assignment on implementing a simulation program of pedestrians walking in groups and passing by other groups. This assignment tackles important aspects of programming such as networking, GUI illustration of data and others.

Amin Yassin

Amin.yassin@live.com

Introduction:

The purpose of this assignment is to implement a program that simulates two groups of pedestrians passing by each other without collision or blockage. Such a simulation should ideally be carried out on several devices or computers. Each group of pedestrians is managed on a computer: Mainly updating their positions and sending them over the network to a computer that observes and controls the simulation. Such pedestrians are required to be respawned once they've reached their final destinations and to avoid colliding with each other if they come in range of each other.

Tools:

Throughout this assignment, IntelliJ IDEA Community Edition was used as an IDE. Also, online resources and documentations were used to better understand the implementation of some code snippets; such as Tutorial Points, Oracle documentation for Java Platform, Stack Overflow and others.

Terminology:

For better understanding of this report, the following terms are defined.

Agent: A single agent represents a pedestrian.

Agent Handler: A class that takes care of a set of agents (Updating positions, sending their position data over the network, checking for collisions, etc). The number of agents is predetermined by the user.

GUI server: The class responsible for showing and controlling the simulation.

Server: For this assignment, the server is the class that waits and accepts the connection of the clients (Agent Handlers).

Server Listener: This class receives position data from an Agent Handler and saves them into a global variable for the GUI server to use for animation.

Design of the simulation platform:

The Server and Server Listener (hereafter known as "Server" and "Listener") were implemented as inner classes of the GUI server class; This means that those two classes are created inside another class. The client class in this assignment is called Agent Handler. The Agent Handler takes care of checking for collisions between agents, updating their positions, and sending the coordinates to the Listener. The Agent Handler sends positions of a group of agents in an array to the Listener and the latter takes it and assigns it to a global variable for the GUI server. Once the global variable is updated, they are animated on the screen using the Animation Timer class; A class that allows the creation of a timer that is invoked per frame, if the timer is active.

The assignment in all was implemented with the MVC (Model-View-Control) principle in mind. This is an architectural pattern that is commonly used when developing programs with interfaces which separates the implementation into three distinguish but connected parts. This approach was chosen here to ease the writing and changing of code, especially once the code size gets bigger. In this implementation, one can think that the Agent class represents the Model, the GUI server class represents the View, and the Agent Handler represents the Controller.

Implementation of the pedestrian model:

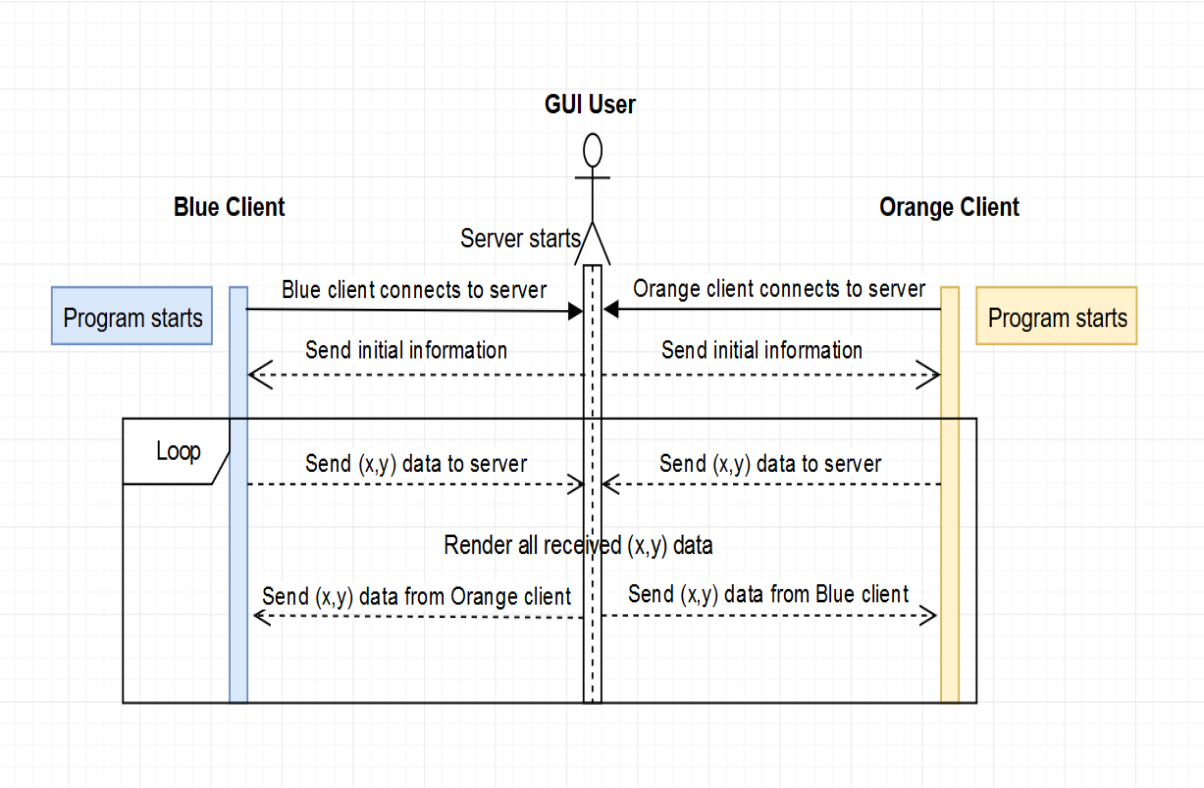

Figure 1: A sequence diagram of the interaction between the user on the server side and the Agent Handlers on the client side.

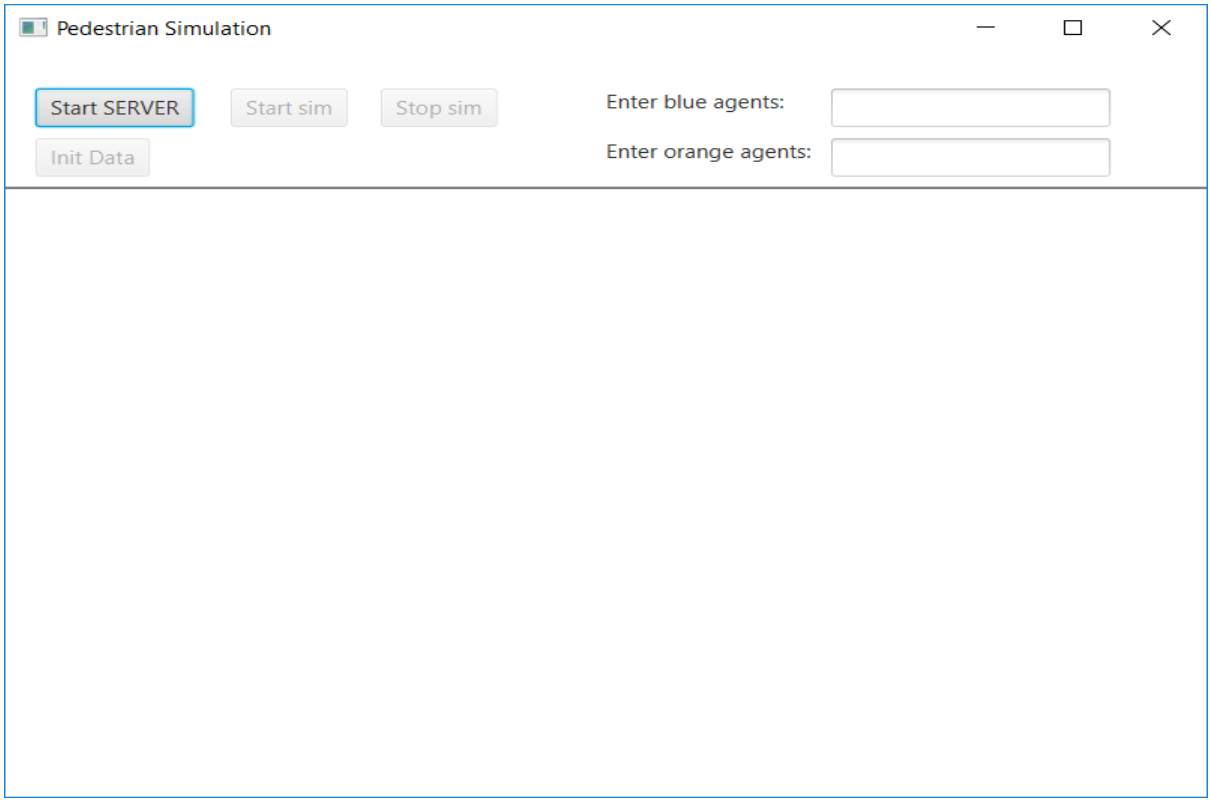

Figure 2: The application window once the GUI server is started.

In this version of the pedestrian model that has been implemented, the user needs to - first and foremost – start the Server that will listen to a certain port and wait for Agent Handlers to connect. After that, the desired number of agents to be simulated later may be entered in the input fields shown on the window. After that, the Agent Handlers are to be started from the client programs, which are separated from the GUI program. Each Agent Handler will receive a data package that contains the following information:

- A string ID indicating the colour of the set of agents. This string will be used in the Agent constructor to create the agents with their respective colours.
- Two integers: One that defines the number of agents that will be created for respective Agent Handlers, and the other integer will indicate how many agents the opposite Agent Handler has.

Once that's done, the user will be able to start the simulation with a button which will draw circles (blue and orange) representing two groups of pedestrians migrating from each side of the display window towards the opposite end. The simulation program on the server side may look as in figure 2.

What happens once everything is running?

Once the server and the client programs are running, updated position data will be sent over the network from the Agent Handlers to the Listener. From the Listener, the GUI server will save these data. The Animation Timer class will render all the data that was stored.

The Agent Handlers will also receive from the Listener the position data of the opposite agents. This is so that they can be used to check any possible collisions between all the agents in the simulation. Also, there is collision checking amongst the agents with the same colour as well. After checking all collisions, the updated positions are sent over the network once again, and with that one cycle has completed. Figure 1 shows a sequence diagram of what is described here.

Discussions:

To test the functionality of the implemented solution for this assignment, only one computer was used to simulate two clients (Agent Handlers) and a server with a GUI, even though the intention is to have three computers/ devices for the final use of this solution.

One of the most tedious challenges was how to send/ receive positions and how to use them to draw circles on the application's window. Some of the previous solutions were to: send agents and extract their positions periodically, send array lists/ queues containing positions, send float numbers per agent, etc. This part took most of the implementation time for this assignment. Another obstacle was that sending/ receiving objects seemed difficult as an object with the same position values were sent over and over again. The solution was to use the functions readUnshared/ writeUnshared for the stream objects.

After that came the next obstacle: How to give the GUI server the positions received by Listener. This happened due to having separate class files for Server and Listener. The solution was to have these classes as inner classes inside the GUI server class.

Another challenge faced with was the collision detection of agents. A considerate amount of time was taken to think of ways of implementing such an algorithm. However, having access to the data of both Agent Handlers was only possible on the GUI server side, this led to the solution that the server – once it receives all the data, will send over the data of the opposite agents for collision checking. Some screenshots of the simulation are shown in figure 3.

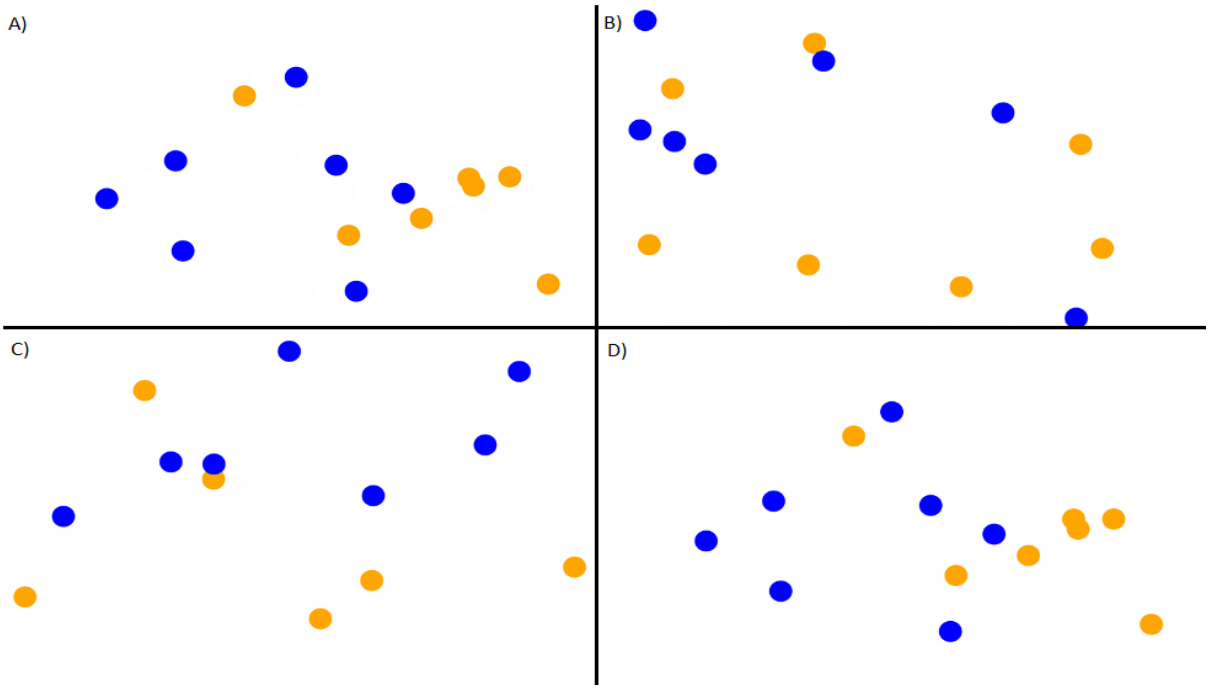

Figure 3: Four screenshots of the pedestrian simulation.

The current implementation of the pedestrian model is lacking in certain areas and attempts to refine and add more and better functionalities were taken. Due to the deadline being reached, it was important to deliver an application that creates instances of agents, manipulate them via Agent Handlers which in their turn send data to the GUI server, and for the latter to plot such data on the screen. With more time and guidance most shortcomings can be reduced.