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 (or more) groups of pedestrians passing by each other without collision nor 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.

Unfortunately, some of these specifications aren't implemented at the moment (Obvious collisions in the middle of the screen, no respawning of pedestrians, no more than two groups are included into the simulation). More details are to be presented throughout this report.

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 several agents (Updating positions, sending their position data over the network, checking for possible collisions, etc).

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 the 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. The client class in this assignment is called Agent Handler. The Agent Handler takes care of checking for collisions between agents of same colour, 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 an Animation Timer.

The assignment in all was implemented with the MVC (Model-View-Control) principle in mind, even if it may not seem to be fully applied. This approach was taken to ease the writing and changing of code, especially once the code size gets bigger. The Model-View-Controller is represented by Agent, GUI server, Agent Handler and respectively.

To illustrate the concepts implemented above, please refer to the UML diagram attached to the end of this report.

Implementation of the pedestrian model:

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 must be entered into input fields shown on the window, and start the Agent Handlers. 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.

Notice that once two Agent Handlers are connected, the Server thread will terminate as there will be no need to keep it running. The shortcomings and pitfalls are to be discussed in the next section.

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. From this, the simulation was conducted as the user on the server side needed to start Agent Handlers (which are the clients) from the same computer.

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 the majority of the implementation of 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 major challenge faced with was the collision detection of agents at the middle of the window. A considerate amount of time was taken to think of ways of implementing such algorithm. However, having access to the data of both Agent Handlers was only possible on the GUI server side; which is not acceptable since it will violate the MVC principle as data will be manipulated (changed) on the GUI server side.

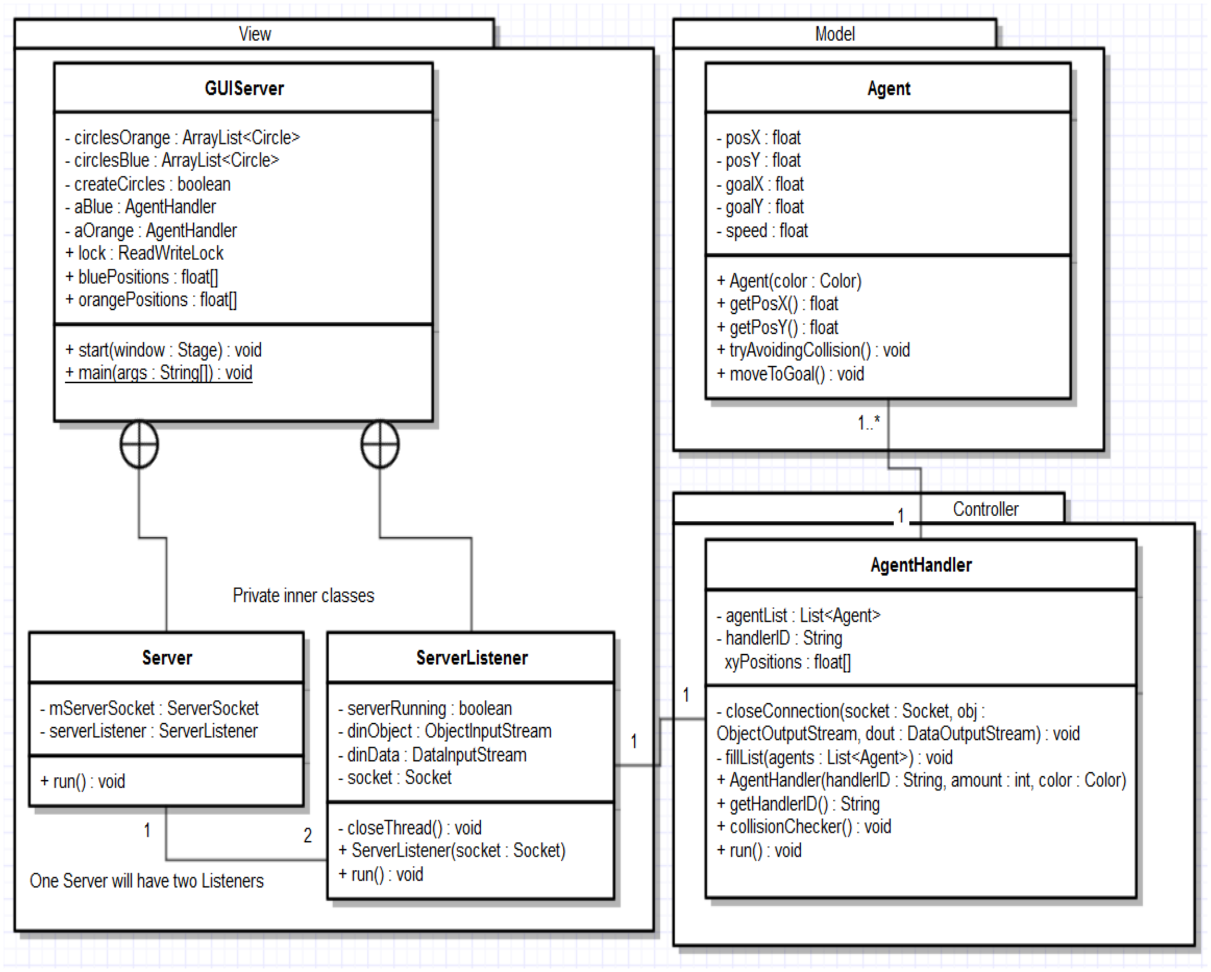
Some functionalities that are missing in the current implementation are:

- Respawning of agents: This functionality wasn't implemented due to putting more time on other parts of the project. Thus, Once the agent groups reach their goal destinations, they will remain there until the user terminates the program.
  Notice that terminating the program isn't sufficient by just closing the application window.
- Collisions of agents with different colours: As for now, only the agents of the same colour happen to implement a collision avoidance algorithm.

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 such shortcomings can be reduced.

Attachments:

Attachment 1: A UML diagram of the pedestrian simulation assignment.


View
GUIServer
- circlesOrange : ArrayList<Circle>
- circlesBlue : ArrayList<Circle>
- createCircles : boolean
- aBlue : AgentHandler
- aOrange : AgentHandler
+ lock : ReadWriteLock
+ bluePositions : float[]
+ orangePositions : float[]
+ start(window : Stage) : void
+ main(args : String[]) : void
Private inner classes
Server
- mServerSocket : ServerSocket
- serverListener : ServerListener
+ run() : void
ServerListener
- serverRunning : boolean
- dinObject : ObjectInputStream
- dinData : DataInputStream
- socket : Socket
- closeThread() : void
+ ServerListener(socket : Socket)
+ run() : void
1
2
One Server will have two Listeners
1
Model
Agent
- posX : float
- posY : float
- goalX : float
- goalY : float
- speed : float
+ Agent(color : Color)
+ getPosX() : float
+ getPosY() : float
+ tryAvoidingCollision() : void
+ moveToGoal() : void
1..*
1
Controller
AgentHandler
- agentList : List<Agent>
- handlerID : String
xyPositions : float[]
- closeConnection(socket : Socket, obj : ObjectOutputStream, dout : DataOutputStream) : void
- fillList(agents : List<Agent>) : void
+ AgentHandler(handlerID : String, amount : int, color : Color)
+ getHandlerID() : String
+ collisionChecker() : void
+ run() : void
1