Final Project Report

Elif Akgün
1801042251

1. Design Explanation

In my project, I used Compound Pattern because a compound pattern combines two or more patterns into a solution that solves a recurring or general problem. Also I used MVC pattern. There is an user interface in this project and MVC allows us to part the program to 3 parts. These are model, view and controller. The model holds all the data, state and application logic. The model is oblivious to the view and controller, although it provides an interface to manipulate and retrieve its state and it can send notifications of state changes to observers. The view gives us a presentation of the model. It usually gets the state and data it needs to display directly from the model. Takes user input and figures out what it means to the model.
Each part uses different design pattern. The model implements the Observer Pattern to keep interested objects updated when state changes occur. With Observer Pattern, model becomes independent from view and controller part. Thus we can use different views with same model. The display consists of a nested set of windows, panels, buttons, text labels and so on. Components may be composite, like the window; or components may be leaf like the button. This makes it easy to update the window. It updates only top component then all components are updated. The view and controller implement Strategy Pattern. They change their behaviours in run time. The view is concerned only with the visual aspects of the application, and delegates to the controller for any decisions about the interface behavior.
I explained my design in more detail below sections for each part.

1.1 View

I created a View class for this part. It implements Observer interface because it is an observer of Model class. I created a GridPane and a canvas to draw people. Also I created 4 Buttons, a TextLabel and 3 Labels and I added all of them to GridPane. Buttons are used for start, pause, continue, and add new person behaviours. Labels are used for writing some messages to window.I also used TextLabel to take number of individual in the society from user. View listens the button actions. When the user clicks the buttons, view delegates it to controller.
There are 3 update methods in this class: update_draw, update_clear, and update_counter.

update_draw and update_clear methods take ID as parameter. They draws and deletes specified individual's rectangle respectively. update_counter method updates healthy individual number and infected individual number. They are called from Model class. This class also keeps a ModelInterface object and ControllerInterface object. ControllerInterface object is used for delegating actions to Controller class. ModelInterface object is used for getting x and y positions to be drew or deleted.
In this project, you can see movements of individuals and current healthy individual number and infected individual number.

1.2 Controller

I created a Controller class for this part. It keeps a ModelInterface object to notify Model class when an action came. It can interact directly with View class so it keeps a View class object, too.
It has 4 methods: start, pause, continue_, and addPerson. start method calls Model class's start method, pause method calls Model class's pause method, continue_ method calls Model class's continue_ method, and addPerson method calls Model class's addPerson method.

1.3 Model

I created Model class and Individual class for this part. Individual class represents individuals and it keeps M, S, C, D, P, B fields that given homework file. It implements Runnable interface. In the run method, firstly it notified observers to draw its position. Then it takes new position as randomly, calculates distance between last position and new position. It calculates distance because distance = speed * time. According the this formula, threads sleep to simulate movement. Threads clear last position and draw new position. It also call Model's notifyObservers_counter method to update healthy person number and infected person number.
Model class implements ModelInterface interface. According to Observer Pattern it is a subject. So it has registerObserver, removeObserver, notifyObservers_draw, notifyObservers_clear, and notifyObservers_counter methods. registerObserver adds the observer to observer list and removeObserver removes the observer to observer list. notifyObservers_draw notifies observers to draw individual's pixels, notifyObservers_clear notifies observers to delete individual's pixels, and notifyObservers_counter notifies observers to update counter.
It has some other methods. There is a start method and when the start button is clicked, controller called this method. It takes number of individual as parameter. It starts threads as many as entered individual number. It has a individuals list and adds the individuals to this list. At the beginning of method, it produces a random number between 0 and individuals number. I assumed the individual in the random numberth index is infected. There are also pause and continue_ methods. When the pause button is clicked, controller called pause method and it suspends the threads.

When the continue button is clicked, controller called continue_ method and it resumes the threads. There is addPerson button and when the add person button is clicked, controller called this method and it adds a person to the society with creating a thread.
In notifyObservers_draw method, after update observers, I check every individual whether there is a collision with another individual. To check this I used isCollision method. It takes two Individual object and checks collision. If there is a collision and flag is true, it calculates collision time. I keep a flag because when 2 individuals collide with each other, and if another individual is in collision course with either of them in the meantime, s/he cannot interact with them and ignores them as if they weren't there. Then, I checked infecting situations of them. If one of them is infected, s/he will infect the other. Healthy person number decreases one and infected person number increases. I calculated infecting probability and ventilator numbers with using given formulas in the homework file.

1.4 Mediator Pattern

I used Model class for this pattern. My Model class is also a Mediator. Individuals call notifyObservers_draw method instead of call update_draw method. As I explained above, notifyObservers_draw checks each Individual and if there is a collision, it makes the threads sleep. Individuals don't check this.

Figure 1: User Interface of Application

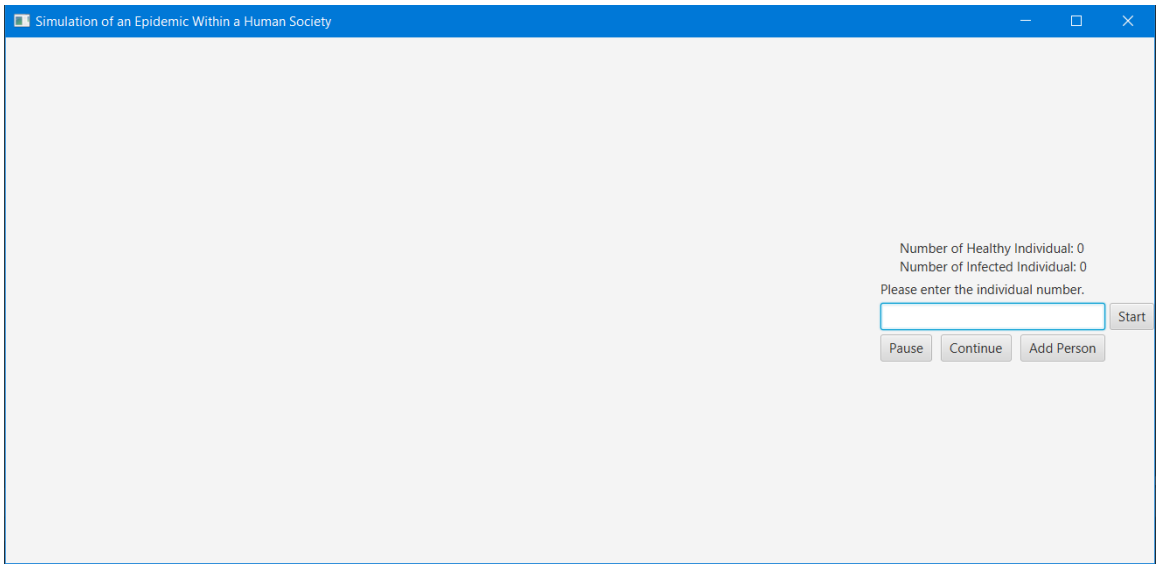


Figure 2: After Adding 100 Individual

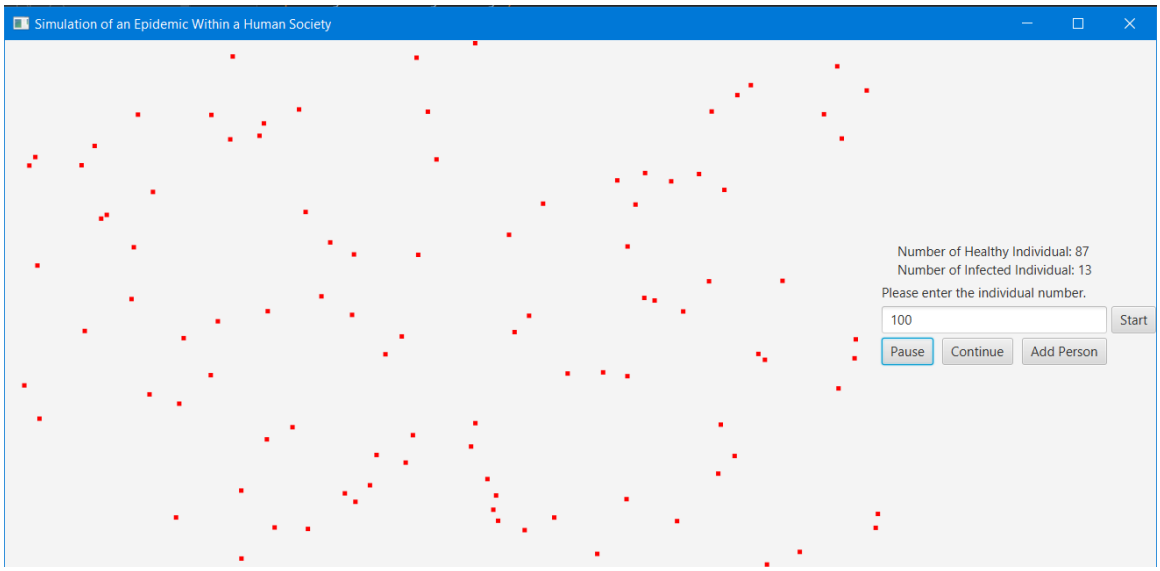
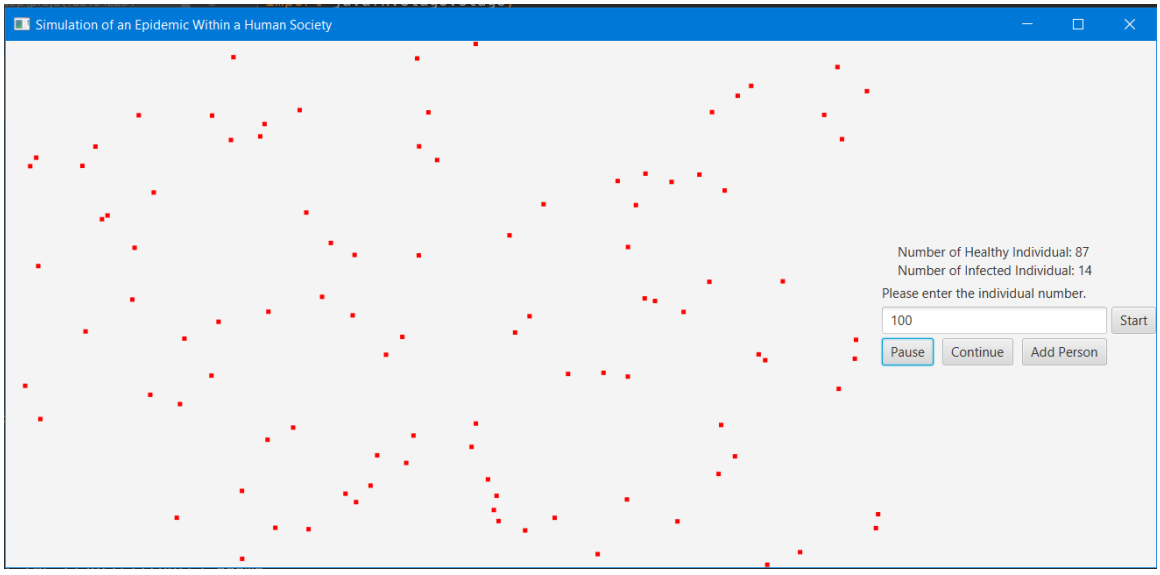


Figure 3: After Adding 1 Individual



2. Class Diagram

Figure 4: Class Diagram of Project