Object Oriented Analysis and Design

CSE443

FINAL PROJECT

CAN BEYAZNAR

161044038

Problem Solution

In this project, I used MVC architecture to reflect individuals in gui, and to enable the user to stop and continue the program whenever they want. And by making the program multithreaded, I made it possible to stop and pause at any time. I will explain each class separately in order to explain the methods I have applied in the project in detail.

In this project, I used MVC architecture to reflect individuals in gui, and to enable the user to stop and continue the program whenever they want. And by making the program multithreaded, I made it possible to stop and pause at any time. I will explain each class separately in order to explain the methods I have applied in the project in detail.

First of all, I gathered the fixed properties (Map length and height, maximum speed etc.) in the program into the MapInformations_Class class. Thus, classes will be able to access constant values using this class.

Then I created the PersonClass class to keep individuals' information. This class contains all the information about the individual. In addition, this class contains a parameter from the mediator interface for the mediator design pattern to be used for interpersonal relationships. Thus, if an individual collides with another individual and becomes ill, he will be able to communicate between individuals using the mediator class. And it uses the MapInformations_Class class to have general information in the program. NOTE: While creating each individual here, mediator, mapInformations and id parameters are taken in the constructor. All created individuals reference the same mapInformations parameter. In this way, the space taken up by the program is slightly reduced. Individuals have a total of 4 states. healthy, collide, patient and dead. healthy green color, collide white, patient red colors. And these colors are used when showing on the map. In the dead state, the individual is not shown on the map. Depending on these 4 states, the algorithm of the methods in personClass changes.

After creating the PersonClass class, we need to create the required classes for the mediator design pattern. Mediator interface was created for this. Then a mediator was created that implemented this interface. Here, all individuals are kept in an ArrayList. There are also Z and R values that are constant. Thus, the transactions to be done between individuals will take place in this class. In the createPopulation method in this class, as many individuals as the population given by the user are created. And an individual is given illness. If you want, you can open the "for fun" section in the comment section. More individuals are given the disease here (May be difficult to follow). If we come to the condition of collision of individuals, the algorithm in the link "https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection" is used. If one of the two individuals has a collide state, the collision will not occur. In addition, this class is used to draw individuals on the map and to reflect the information of individuals. All transactions between individuals are in this class. Individuals walking, colliding, disease transmission and so on.

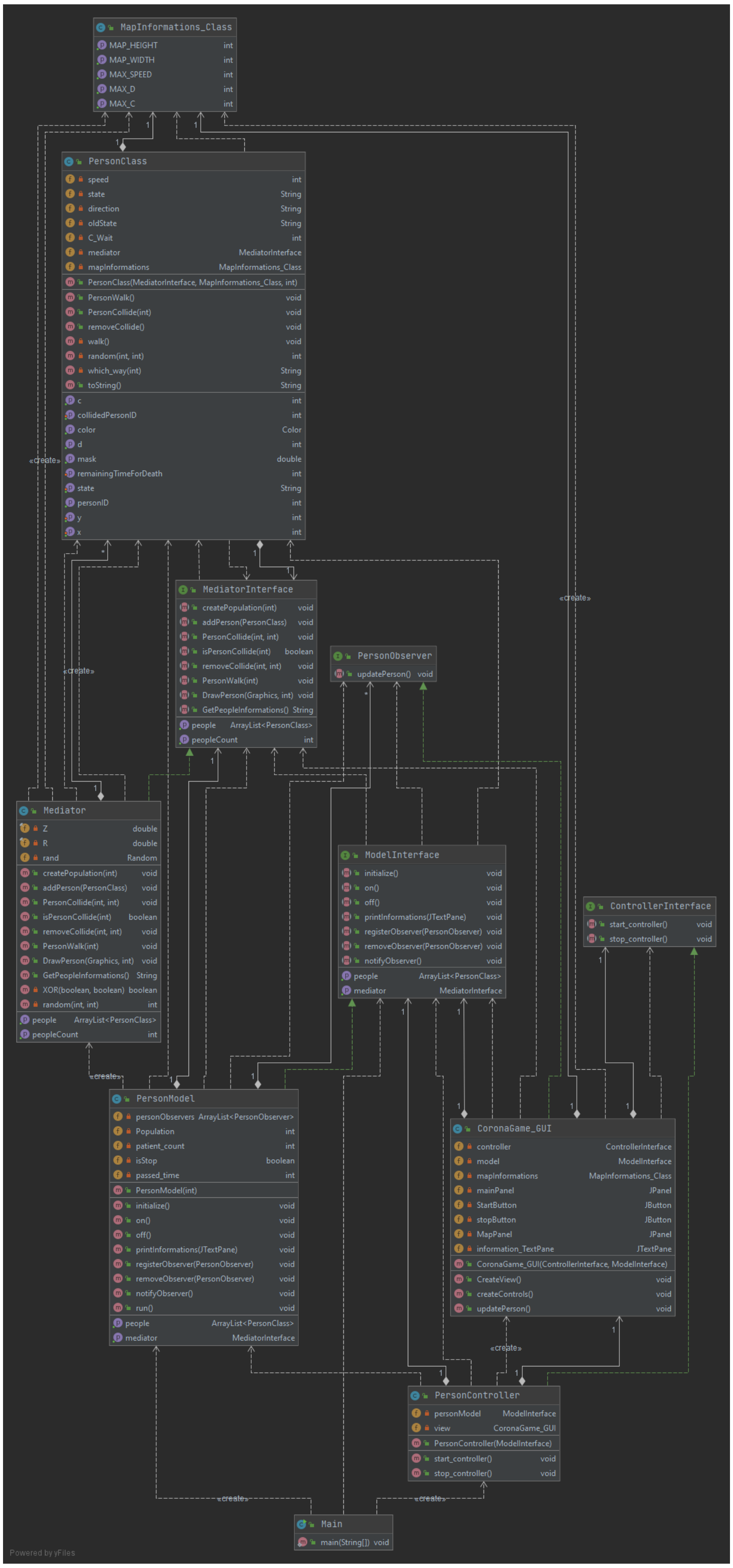
After running classes at the back of the program, I implemented the MVC architectural design. First, I created the model interface. And I implemented the PersonModel class, which implemented this interface. In PersonModel class, there is an ArrayList of PersonObserver type for View to be aware of updates. So when individuals move, a notification will go to View and draw the individuals to the panel. Apart from this parameter, it keeps the mediator parameter in order to access the information of the individuals. And in the initialize method, it will create individuals by calling the createPopulation method in the mediator. While the PersonModel class allows individuals to be projected onto the screen, it must work in thread form for the user to stop and resume the screen. Therefore, it additionally implements the Runnable interface. And in the overridden run method, observers in the mediator are notified. Thus, the model indirectly sends a message to the view and draws the individuals in the view. Here, the population is taken from the user as the constructor.

Next, a Controller must be created to control stop, resume buttons. That's why I first created the Controller interface. Then I created the PersonController class that implements this interface. Very little processing is done here. In the Constructor, the model interface is taken. And a view is created for gui to work. Then, components that should be in GUI are created (such as Buttons, JFrame). And the initialize () method is called in the model. Finally, by calling the model class in the thread, the program works in a multithreaded way and can get a response from the user at any time. And so, using the controller class, both the GUI is called and the model class is created, which contains the individuals to be reflected in the gui and processes these individuals. Finally, when the user presses the buttons, the on () off () methods in the model are called in the start_controller () and stop_controller () methods in order for the program to stop or continue.

And finally, the part, which is the view part and where individuals will be shown on the screen, was implemented. CoronaGame_GUI class implements the PersonObserver interface. Thus, by registering to the model, it will be able to receive the changes from the model and reflect the new positions and information of individuals on the screen. Here it takes controller and model class as constructor. Thus, view, controller and model classes will be in communication with each other by being linked. In the CreateView () and createControls () methods, the properties of the components in the GUI are set. The updatePerson () method is implemented from the PersonObserver interface and the current state of the individuals is obtained through this method.

Design patterns and methods applied in the project are as explained above. Detailed explanation of the code is available in the comments. The parts that are not made in the project are quite high. Hospital section, graphics etc. The 4 different states available in PersonClass can increase maintenance cost and complexity. To reduce this cost, strategy design pattern or state design pattern could be used.

The UML diagram of the project is as follows. It may not look clear because it is too big. The net version is available in the diagram.png file.


MapInformations_Class
MAP_HEIGHT int
MAP_WIDTH int
MAX_SPEED int
MAX_D int
MAX_C int
PersonClass
speed int
state String
direction String
oldState String
C_Wait int
mediator MediatorInterface
mapInformations MapInformations_Class
PersonClass(MediatorInterface, MapInformations_Class, int)
PersonWalk() void
PersonCollide(int) void
removeCollide() void
walk() void
random(int, int) int
which_way(int) String
toString() String
c int
collidedPersonID int
color Color
d int
mask double
remainingTimeForDeath int
state String
personID int
y int
x int
MediatorInterface
createPopulation(int) void
addPerson(PersonClass) void
PersonCollide(int, int) void
isPersonCollide(int) boolean
removeCollide(int, int) void
PersonWalk(int) void
DrawPerson(Graphics, int) void
GetPeopleInformations() String
people ArrayList<PersonClass>
peopleCount int
PersonObserver
updatePerson() void
Mediator
Z double
R double
rand Random
createPopulation(int) void
addPerson(PersonClass) void
PersonCollide(int, int) void
isPersonCollide(int) boolean
removeCollide(int, int) void
PersonWalk(int) void
DrawPerson(Graphics, int) void
GetPeopleInformations() String
XOR(boolean, boolean) boolean
random(int, int) int
people ArrayList<PersonClass>
peopleCount int
ModelInterface
initialize() void
on() void
off() void
printInformations(JTextPane) void
registerObserver(PersonObserver) void
removeObserver(PersonObserver) void
notifyObserver() void
people ArrayList<PersonClass>
mediator MediatorInterface
ControllerInterface
start_controller() void
stop_controller() void
PersonModel
personObservers ArrayList<PersonObserver>
Population int
patient_count int
isStop boolean
passed_time int
PersonModel(int)
initialize() void
on() void
off() void
printInformations(JTextPane) void
registerObserver(PersonObserver) void
removeObserver(PersonObserver) void
notifyObserver() void
run() void
people ArrayList<PersonClass>
mediator MediatorInterface
CoronaGame_GUI
controller ControllerInterface
model ModelInterface
mapInformations MapInformations_Class
mainPanel JPanel
StartButton JButton
stopButton JButton
MapPanel JPanel
information_TextPane JTextPane
CoronaGame_GUI(ControllerInterface, ModelInterface)
CreateView() void
createControls() void
updatePerson() void
PersonController
personModel ModelInterface
view CoronaGame_GUI
PersonController(ModelInterface)
start_controller() void
stop_controller() void
Main
main(String[]) void
«create»
«create»
«create»
«create»
«create»
«create»
«create»
1
*
Powered by yFiles

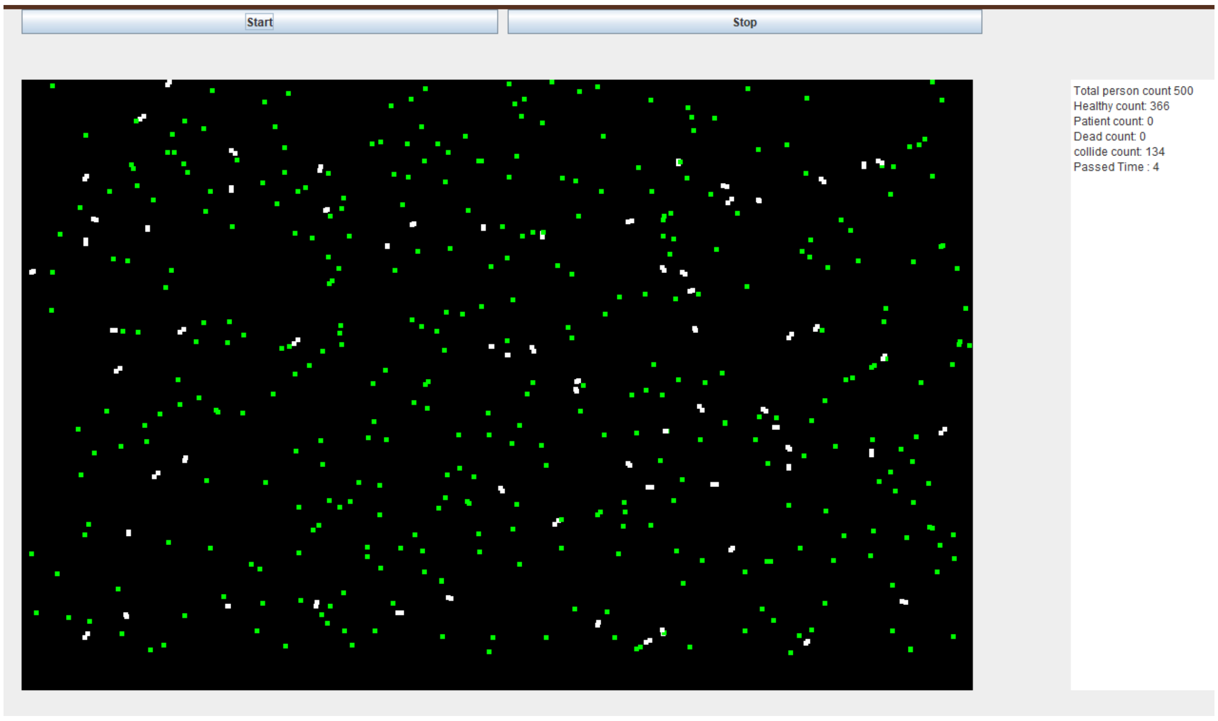
TEST

The population is set to 500 because the program cannot be implemented dynamically. Articles are written to the terminal in order to better observe the working logic of the program.

Green -> healthy

Red -> patient

White -> collide



```
Total person count 500
Healthy count: 366
Patient count: 0
Dead count: 0
collide count: 134
Passed Time : 4
```