CSE 443 FINAL PROJECT REPORT

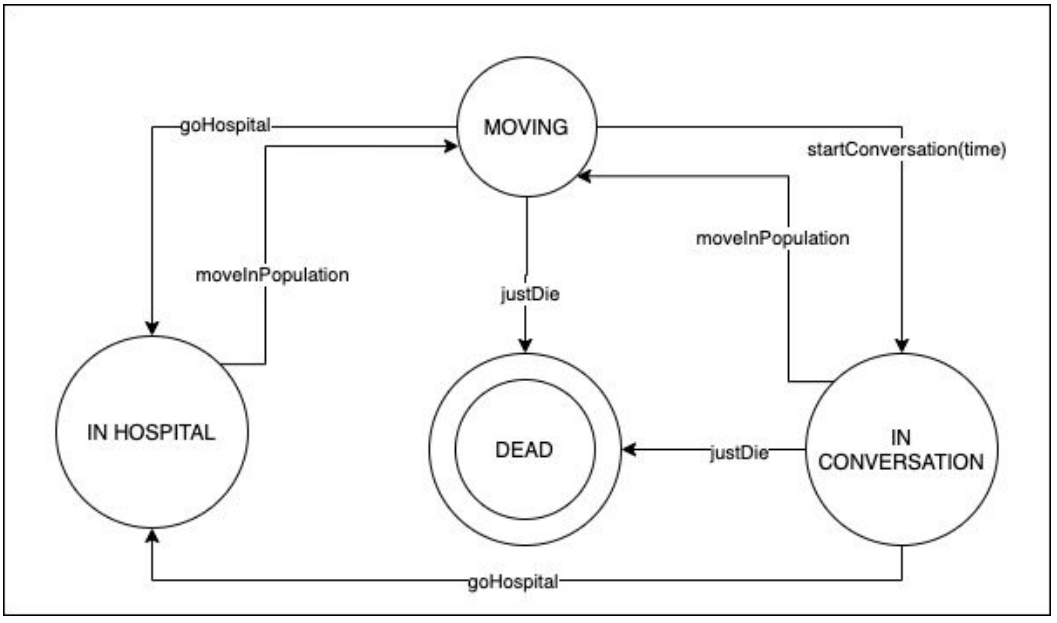
In this project we're asked to implement a GUI based epidemic simulator. We have a map 1000x600 dimension and individuals that moving on that map. At beginning we have an individual that is infected and other individuals that is healthy. From there individuals moves on map freely and collide with each other. When they collide, they locked each other for communicating and in this communication, infection happen, and disease spread. After spreading, individuals can try to go hospital, but hospital has limited size, so some of them are heal, some of them are die. This is general looking to simulation.

For simulation, we have general parameters shared by all simulation world. These are spreading factor, mortality rate, hospital itself, statistics like total death, healthy, hospitalized count. Also view side of simulation shared for whole project. For that reason, I created a class called ***Simulation*** to put general things inside it. This class holds *mortality rate*, *spreading factor*, total population size, population members, which is called individual, references, hospital reference, access to view of map(SimulationMap) and other statistics. This class is main class for simulation, everything works through this class. When this class created, it takes initial population size and dimensions of map. After that it starts to initialize simulation. Some values are random with a range. It gets mortality rate or spreading factor by randomizing in given range. Then initialize statistics of simulation, with initial numbers. After that starts to generate Individuals with amount of initial population size. Then creates simulation map, which is draws map and individuals to screen, and adds Individual references to that map class. Initialize hospital according to population size and infect first Individual to start spreading disease. To summary, this class generates simulation world with specific parameters and components has access to this class, access world's general parameters.

Also, this main ***Simulation*** class embeds Mediator structure inside itself. It implements mediator interface, and it has access to all Individual references. Individuals can't talk directly with each other to prevent complexity. So, individual talks with each other through this simulation class. As we said, Simulation class holds world specific parameters and statistics and individuals. Every individual need to access these parameters to set its own state. Every individual takes Simulation class as parameter to access world specific parameters. Also, since Simulation is also mediator class, Individuals must access to that mediator to communicate other Individuals. So Individual has access to simulation class. Simulation class has access to map side of simulation, and hospital side. It has getter for that references. So, an individual can access hospital via this reference, some other view class can access that simulation map using map reference to add simulation to frame. Simulation gives ability to access open world. Everyone takes what is needed. For perspective of Individual this is ***flyweight pattern***. Every Individual needs that parameter. Instead of some amount of reference they only use simulation reference to access world specific parameters. Other useful side of holding these references, Individuals can change state and this state changes can have some effect on simulation and view. For example, Individual can die or infected. In GUI side, we have only access to Simulation class. So, when an individual

changed whole statistic of simulation, it only needs deliver these changes to Simulation class. “Simulation” holds whole state of simulation. So, GUI element can access that statistic to show it to user. In other side of things, GUI can change state of whole simulation, and these changes can affect Individuals. For example, if population size changes, these new Individuals must be added to map, also hospital’s ventilator count must increase and newly added individuals can change some parameters, but since we do that center based and everything managed by Simulation class, these changes can be taken dynamically from Simulation class and impacts of changes show itself immediately.

Individual class encapsulates Individual specific parameters. It extends Rectangle class, so it holds 5x5 square place on GUI. It has additional parameters determined randomly. Its constructor takes simulation class as a parameter. Simulation class stands for whole world parameters and communications that is needed for every individual. Individuals share same world. Individual has constant speed determined randomly between [1,500] that will be update amount of x, y coordinates every update of GUI. It has a social distance [0,9], mask indicator parameters masked (1.0) or not masked (0.2) and these are used to determine in communication, if individual will be infected or not. To utilize color of infected individual it has a health state enumeration healthy or infected. Also, every individual has conversation time for itself. This time is used to determine conversation time between two individuals. Individual uses ***state pattern*** to determine its state. Its state changes by itself. So, it maintains state by self. By that it doesn't affect any other object other than itself.



An individual can be in 4 different state. If individual is "moving", it is open for communication. If there is a collision with other individual, individual change state to "in conversation". To determine collision, Individual uses Simulation class reference. Since Simulation is also mediator, Individual uses its location update method with argument itself. Since Simulation has access to all other Individual references it can compare location of itself with other Individuals in case of collision. If individuals are match, they locked each

other and closed for communication. From there we can determine infection. Using given formula, we calculate infection probability and determine Individual is get infected or not. Also, we start conversation with other individual max of their conversation time try to change individuals state to “in conversation”. In state of "in conversation", individual is closed for communication with other individual. "In conversation" to simulate conversation there must be spend some time. After that time individual can be again in "moving" state. In state changes, individual uses its PlayPauseTimer class. This class holds remaining time in ms format. After that when GUI's main timer traverse all individuals and update them, this timer values updated and decreased by amount of refresh time of GUI. The reason behind that it, maintain one timer that updates all timers, so when user pause simulation, all individuals keep same state and position. Also, GUI has access to all individuals and draw them using main swing timer. Individual maintains different timers for different states. When it is in conversation, it spends some time in conversation, that is maximum of two communicating individual. When its timer is end, it tries to change its state is moving, if it is possible. If individual is infected, its death timer is start. In that time, if individual can't access to hospital, it will die and change state to dead. From there user cannot interact with simulation. If individual can access to hospital, now it stops all timers and go hospital. Hospital cure for every patient. It also spends some time in hospital. After that time, individual returns to population with healthy state. Timer events happens asynchronously, for that reason state changing functions is put in synchronized blocks. When a timer is finishes it removes itself from timer list to prevent taking updates again. Also, timers have actions when timer is finished. These actions determine what timer does when it is finished. Individual also holds a thread. That is used when individual needs hospitalizing. When Individual get sick, after some time it tries to go hospital but if hospital is full, it returns to population. So individual must move in population when hospital is full but also must try to go hospital to get heal. For that reason, this process needs another thread, and this thread does process of trying to go hospital. Also, these state changes can change statistics of Simulation. So, when user changes state that effect statistics of whole Simulation, it informs Simulation and Simulation update its counts according to that change.

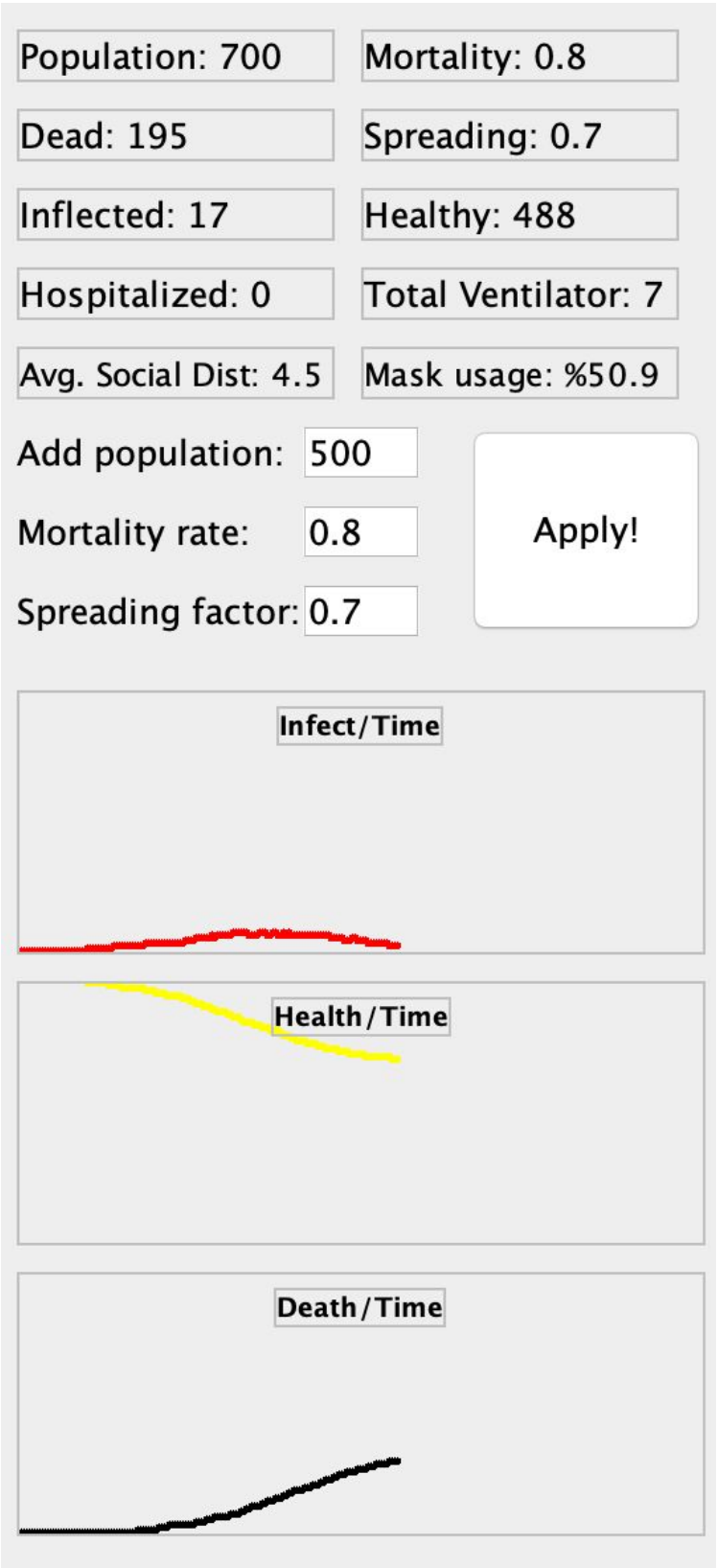
Hospital is another class created in Simulation class. This hospital simulates real life hospital and has capacity proportional to population size. As population size changes capacity of hospital change. After Individual get infected and 25 second passes, it tries to go hospital. Uses thread for that try phase, since hospital capacity can full. Instead of freezing Individual, until get into hospital we don’t change its behavior and it continues to infect other Individuals. When access granted for hospital, Individual goes to hospital, changes its state and timers and start healing. To simulate that producer/consumer state, I used a counting semaphore. This semaphore has count as ventilator count. When an Individual try to get into hospital, in real it tries to acquire semaphore. If it accesses, it starts healing timer and changes its own state. After that time, healing finishes, Individual exit hospital and release semaphore count. By using that, other individuals can get into hospital. Because of async events, sometimes dead individuals can get hospital access. In that time, by the help of current state, this access gives back to semaphore immediately.

GUI has another class called View and holds a reference to Simulation. All components of GUI placed on that frame. Simulation map used to draw map and individuals added to that frame as component and gets from Simulation reference. Also, statistics used by labels that show current state of simulation, pull updates periodically from Simulation itself and show them to user. User can change mortality rate, spreading factor and population size from GUI. Filled places applied after user click Apply button. These changes delegated to Simulation class and simulation class updates its parameters. To stop simulation user click play/pause button. This button delegates this request to Simulation class and Simulation class stop Simulation Map class' timer. When this timer is stopped, no update comes to Individual. Since updates stopped, internal timers of Individual also stops and Individual stop by preserving current state and place. After user click play, Individual start getting update by swing timer of SimulationMap and update its state if it is needed.

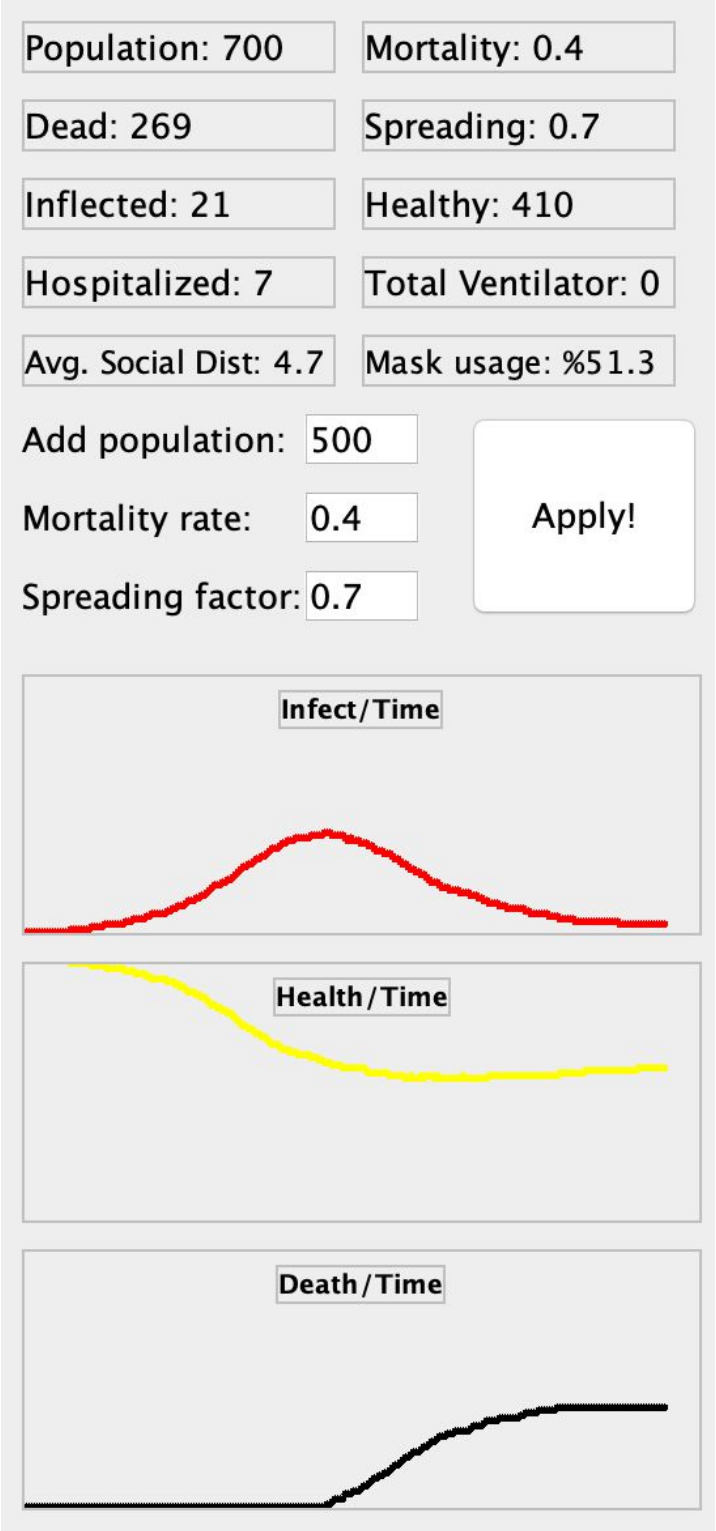
GUI also generates graphs for current counts and show its shape. GraphPanel class takes data point updates every second and for every point draw circle on its panel rescaled for that panel. By that we can see general convention of epidemic in graph visualization.

As we can see, deaths and infection count increase exponential. Since the mortality rate is small, Individuals infect other individuals without knowing they are infected. Their hospital go timer is too much and they have enough time to spread disease. After some time, disease becomes more common between individuals and they understand they are infected and try to go to hospital. They try at the same time, so hospital can't handle this much patients and Individuals cannot heal. By that infect count starts decreasing and stops exponential rise, since infected ones start dying because of lack of hospital service. Since they are dead, they cannot infect others and disease starts stabilizing.

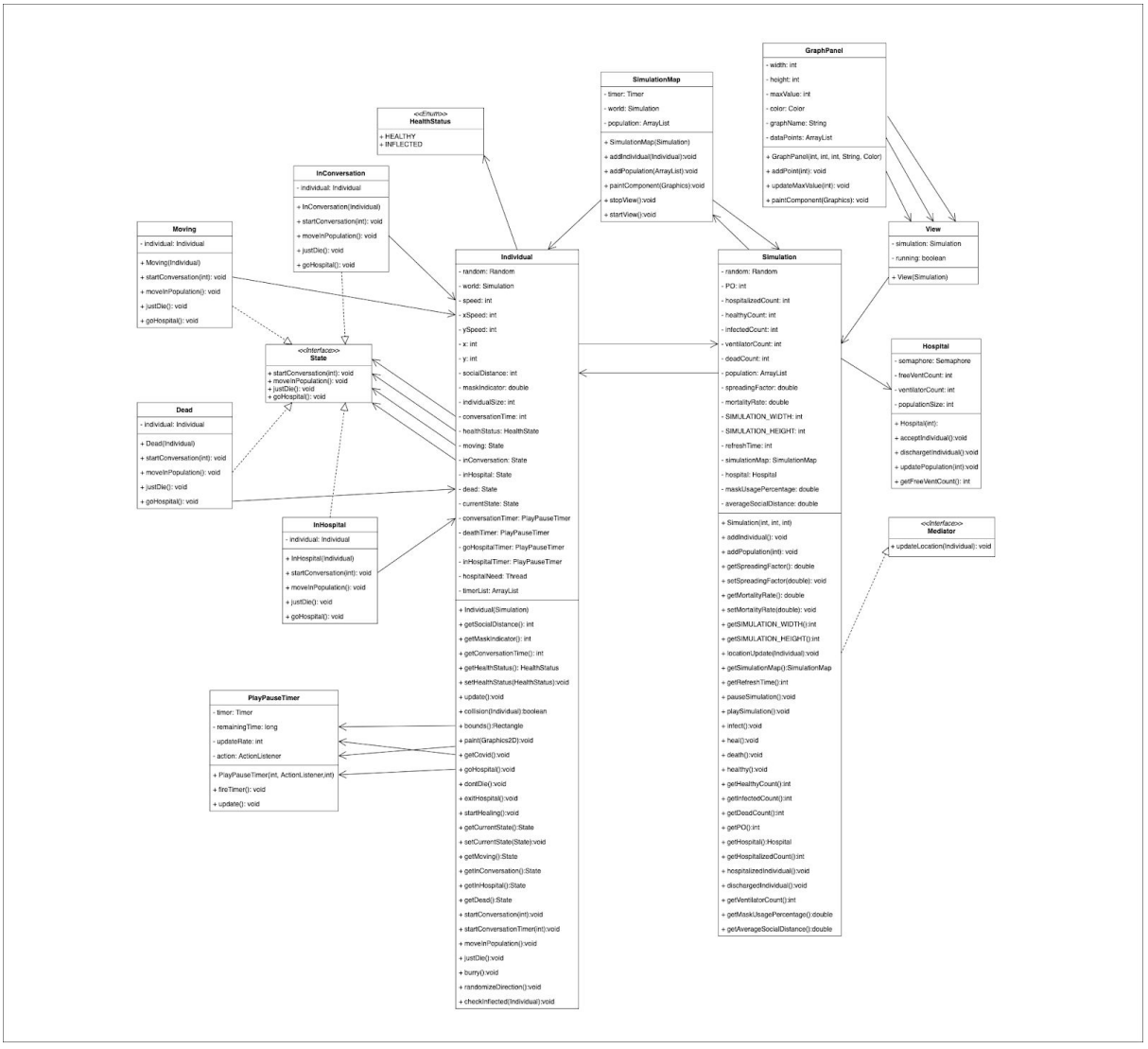
If the mortality rate is too much, individuals start dying without even going to hospital. Also, since they die too early, they cannot infect enough people to spread disease and they decrease. As you can see from graphs deaths starts too early, but they stabilize at some point



If population size increase, since they have more ventilator, they can access to more hospital service. Here population size can be a disadvantage, but mortality rate is too much. So, people die too early to spread disease.



If population size increase, with low mortality, shapes is same as low population but peek is much lower because of more hospital service.


GraphPanel
- width: int
- height: int
- maxValue: int
- color: Color
- graphName: String
- dataPoints: ArrayList
+ GraphPanel(int, int, int, String, Color)
+ addPoint(int): void
+ updateMaxValue(int): void
+ paintComponent(Graphics): void
SimulationMap
- timer: Timer
- world: Simulation
- population: ArrayList
+ SimulationMap(Simulation)
+ addIndividual(Individual):void
+ addPopulation(ArrayList):void
+ paintComponent(Graphics):void
+ stopView():void
+ startView():void
<<Enum>>
HealthStatus
+ HEALTHY
+ INFECTED
InConversation
- individual: Individual
+ InConversation(Individual)
+ startConversation(int): void
+ moveInPopulation(): void
+ justDie(): void
+ goHospital(): void
Moving
- individual: Individual
+ Moving(Individual)
+ startConversation(int): void
+ moveInPopulation(): void
+ justDie(): void
+ goHospital(): void
Individual
- random: Random
- world: Simulation
- speed: int
- xSpeed: int
- ySpeed: int
- x: int
- y: int
- socialDistance: int
- maskIndicator: double
- individualSize: int
- conversationTime: int
- healthStatus: HealthState
- moving: State
- inConversation: State
- inHospital: State
- dead: State
- currentState: State
- conversationTimer: PlayPauseTimer
- deathTimer: PlayPauseTimer
- goHospitalTimer: PlayPauseTimer
- inHospitalTimer: PlayPauseTimer
- hospitalNeed: Thread
- timerList: ArrayList
+ Individual(Simulation)
+ getSocialDistance(): int
+ getMaskIndicator(): int
+ getConversationTime(): int
+ getHealthStatus(): HealthStatus
+ setHealthStatus(HealthStatus):void
+ update():void
+ collision(Individual):boolean
+ bounds():Rectangle
+ paint(Graphics2D):void
+ getCovid():void
+ goHospital():void
+ dontDie():void
+ exitHospital():void
+ startHealing():void
+ getCurrentState():State
+ setCurrentState(State):void
+ getMoving():Stste
+ getInConversation():State
+ getInHospital():State
+ getDead():State
+ startConverssation(int):void
+ startConversationTimer(int):void
+ moveInPopulation():void
+ justDie():void
+ burry():void
+ randomizeDirection():void
+ checkInfected(Individual):void
Simulation
- random: Random
- PO: int
- hospitalizedCount: int
- healthyCount: int
- infectedCount: int
- ventilatorCount: int
- deadCount: int
- population: ArrayList
- spreadingFactor: double
- mortalityRate: double
- SIMULATION_WIDTH: int
- SIMULATION_HEIGHT: int
- refreshTime: int
- simulationMap: SimulationMap
- hospital: Hospital
- maskUsagePercentage: double
- averageSocialDistance: double
+ Simulation(int, int, int)
+ addIndividual(): void
+ addPopulation(int): void
+ getSpreadingFactor(): double
+ setSpreadingFactor(double): void
+ getMortalityRate(): double
+ setMortalityRate(double): void
+ getSIMULATION_WIDTH():int
+ getSIMULATION_HEIGHT():int
+ locationUpdate(Individual):void
+ getSimulationMap():SimulationMap
+ getRefreshTime():int
+ pauseSimulation():void
+ playSimulation():void
+ infect():void
+ heal():void
+ death():void
+ healthy():void
+ getHealthyCount():int
+ getInfectedCount():int
+ getDeadCount():int
+ getPO():int
+ getHospital():Hospital
+ getHospitalizedCount():int
+ hospitalizedIndividual():void
+ dischargedIndividual():void
+ getVentilatorCount():int
+ getMaskUsagePercentage():double
+ getAverageSocialDistance():double
View
- simulation: Simulation
- running: boolean
+ View(Simulation)
<<Interface>>
State
+ startConversation(int): void
+ moveInPopulation(): void
+ justDie(): void
+ goHospital(): void
Dead
- individual: Individual
+ Dead(Individual)
+ startConversation(int): void
+ moveInPopulation(): void
+ justDie(): void
+ goHospital(): void
InHospital
- individual: Individual
+ InHospital(Individual)
+ startConversation(int): void
+ moveInPopulation(): void
+ justDie(): void
+ goHospital(): void
Hospital
- semaphore: Semaphore
- freeVentCount: int
- ventilatorCount: int
- populationSize: int
+ Hospital(int):
+ acceptIndividual():void
+ dischargeIndividual():void
+ updatePopulation(int):void
+ getFreeVentCount(): int
<<Interface>>
Mediator
+ updateLocation(Individual): void
PlayPauseTimer
- timer: Timer
- remainingTime: long
- updateRate: int
- action: ActionListener
+ PlayPauseTimer(int, ActionListener,int)
+ fireTimer(): void
+ update(): void