Universidad Técnica Federico Santa María

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

ELO329 - DISEÑO Y PROGRAMACIÓN ORIENTADA A OBJETOS

Tarea 3

Simulador Gráfico en C++ de la Evolución de una Pandemia

Estudiante	ROL
Gustavo Matamala	201730020-5
Felipe Contreras	201621002-4
Ricardo Mardones	201621036-9
Nicolás Veneros	201621024-5

Paralelo: 2

Profesor

Patricio Olivares R.

Ayudante

Luis Torres G.

Fecha : 22/07/2021

Índice

Índice de figuras

1. Documentación

1.1. Diagrama UML

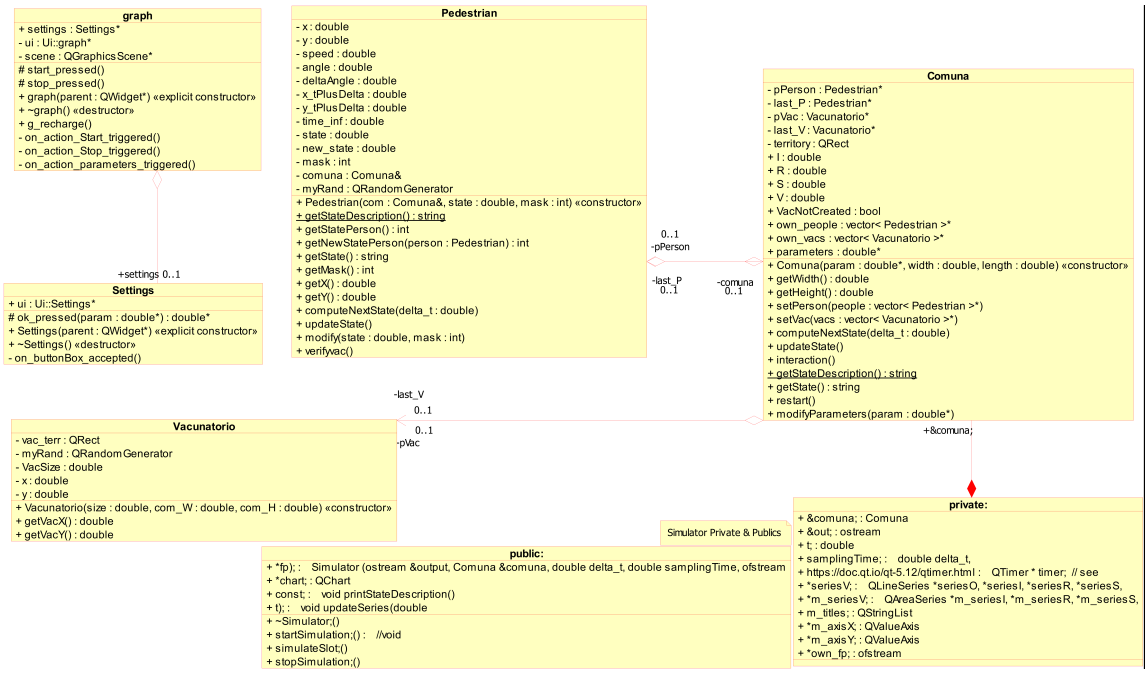

Figura 1: Diagrama UML de Stage4

La figura 1 representa el diagrama UML de la etapa 4, el cual incluye todas las clases necesarias para generar la simulación. De este se pueden observar todas las relaciones entre clases.

En primer lugar tenemos stage 4 que es la clase principal donde se utilizan el resto de clases para correr la simulación en el método start.

Luego podemos ver las lineas de dependencia (líneas discontinuas) que simbolizan el uso de un método u objeto de la clase señalada. Así mismo están las lineas de asociación (líneas continuas) que muestran el uso de una referencia a la clase señalada como un atributo.

De esto podemos apreciar que existe un centralismo en torno a la clase comuna dado que la mayoría de clases deben ocupar metodos de esta, como por ejemplo Simulator, la cual los llama cada cierto intervalo de tiempo para actualizar la simulación.

1.2. Explicación breve

Para el desarrollo de la actividad, se utiliza como base las clases dispuestas por los profesores de la asignatura en el repositorio del ramo. Se ejecuta de manera incremental lo pedido en la tarea.

Esto se logra aplicando las siguientes clases:

- **Pedestrian**: Posee como atributos y métodos los observados en la figura 1. Los métodos importantes para el desarrollo de la actividad corresponden a: *computeNextState(double)*, el cual recibe como argumento el tiempo *delta_t* en el que se está computando el estado, aquí se mueve al individuo.

  El siguiente método de importancia en la clase es *getNewStatePerson(Pedestrian,)* que retorna un valor *int* el cual indica el estado del individuo, siendo los posibles valores 0, 1, 2 y 3 para susceptible, infectado, recuperado y vacunado respectivamente. Para el computo primero se verifica si el peatonal esta vacunado, de no ser este el caso se genera una interacción con otro individuo si la distancia entre ambos es menor a d. La interacción puede cambiar el estado de la persona basándose en el estado del segundo individuo y si se esta o no ocupando mascarilla.

  verifyVac() que verifica si el individuo está dentro de algún área de vacunación, en cuyo caso cambia su estado state y newstate a 3. En caso de terminar de recorrer la lista de vacunatorios y no haber coincidido con ningún área se mantiene el estado anterior.

  Finalmente, *updateState()* actualiza los nuevos estado y posición a los calculados.

- **Comuna**: Posee como atributos y métodos los observados en la figura 1. En esta clase los métodos más importantes son

  setPerson() que añade peatones a un arreglo de pedestrian de forma que se cumplan las especificaciones ingresadas, *Interaction()* que efectúa la interacción entre todos los individuos del arreglo de pedestrian, *computeNextState(double)* que recibe como argumento un delta_t y calcula para cada instancia de pedestrian su nueva posición según el método de mismo nombre descrito en la clase pedestrian, *updateState()* que actualiza los estados de cada individuo de la comuna.

  Otros método importante es *getState()* . El cual recibe la condicion del estado de los individuos para la construcción del gráfico de áreas.

- **Vacunatorio**: Crea el área correspondiente a un vacunatorio.

- **Simulator**: Se definen atributos propios de la simulación, los cuales se leen en Stage4 desde el documento configurationFile.txt y son guardados en ella para posterior utilización. Algunos métodos utilizados son: *updateSeries()* el cual actualiza los valores del grafico, *startSimulation()* que inicializa la simulación y *stopSimulation()* que detiene la simulación, y por ultimo *simulateSlot()* el cual permite ligar la simulacion con el timer creado dentro de la clase.

- **Stage4**: Crea un nuevo objeto de la clase Comuna, un nuevo objeto de la clase Simulador, los inicializa con los datos necesarios según descripción de la tarea, y hace correr la simulación.

- **settings**: Modifica los parámetros de la simulación dependiendo de los datos ingresados en la ventana *Modify Parameters* que se abre al apretar el boton settings del menu.

- **graph**: Crea la ventana , el menu de la interfaz gráfica y sus items Control y Setting. Además tambien maneja el funcionamiento de ambos botones, y contiene el elemento gráfico de la simulación.

1.3. Dificultades

Algunas dificultades fueron:

1. Las dificultades más grandes que se encontraron durante el desarrollo de la tarea fueron, en primer lugar, el cambio de codificacion utilizado en este caso, ya que se utilizo C++ en vez de java .

2. En segundo lugar Cambiar los parametros de las comunas desde la interfaz grafica settings, ya que no era trivial el pasar parametros a traves del metodo connect. Ademas de esto, nos costo reconocer el funcionamiento de los vacunatorios, junto con el manejo de vectores, ya que se utilizaron para el manejo de vacunatorios y de pedestrians cuyo manejo fue necesario dentro de la clase comuna, sin embargo, no era factible crear dichos vectores dentro de la comuna

3. En Tercer lugar, un problema que se tuvo fue limpiar la interfaz grafica tras reiniciar la simulacion, ya que inicialmente no se hallaba la forma de volver a ajustar la ventana a los valores iniciales. .