Universidad Técnica Federico Santa María

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

ELO329 - DISEÑO Y PROGRAMACIÓN ORIENTADA A OBJETOS

Tarea 2

Simulador Gráfico de la Evolución de una Pandemia

Estudiante	ROL
Tamara Carrera	201621010-5
Felipe Contreras	201621002-4
Ricardo Mardones	201621036-9
Nicolás Veneros	201621024-5

Paralelo: 2

Profesor

Patricio Olivares R.

Ayudante

Luis Torres G.

Fecha : 13/06/2021

Índice

Índice de figuras

1. Documentación

1.1. Diagrama UML

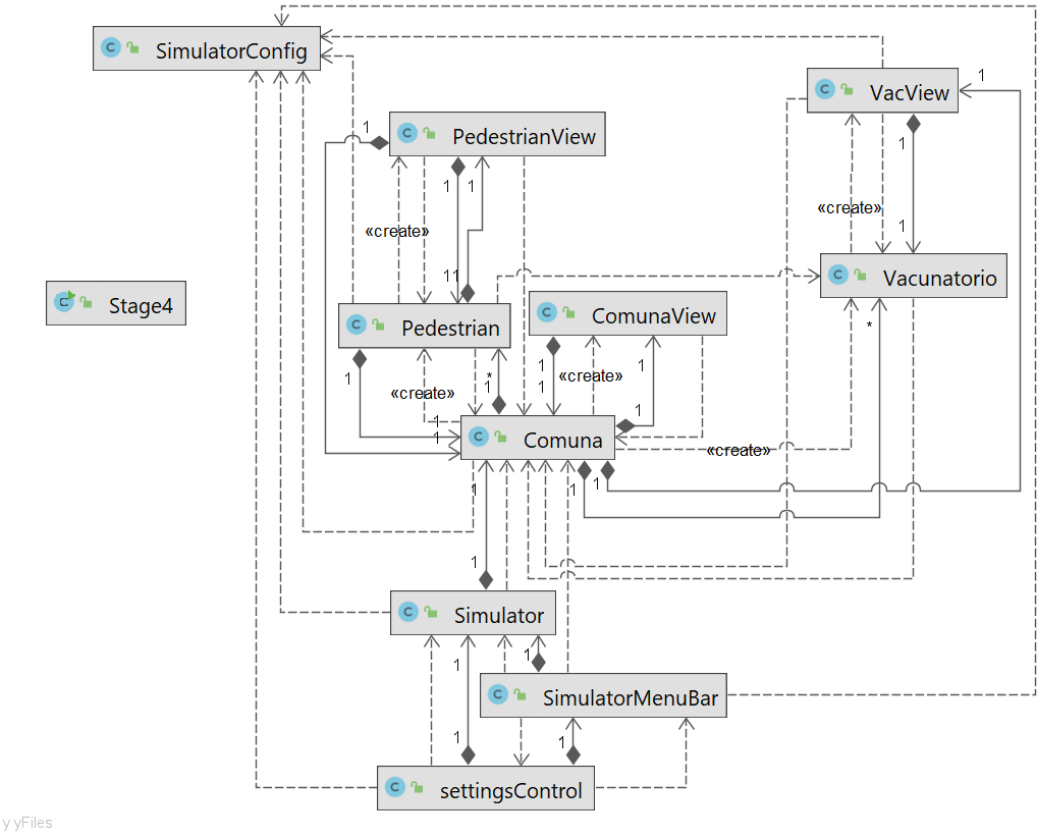

Figura 1: Diagrama UML de Stage4

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.

Por otro lado también vemos la relación entre Pedestrian, Comuna y Vacunatorio con las clases usadas para crear sus animaciones, las cuales son PedestrianView, ComunaView y VacView respectivamente.

Por ultimo otra observación importante son las conexiones entre SimulatorConfig y el resto de clases, dado que es desde esta que se obtienen atributos estáticos y parámetros que controlan 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 *getNewState(Pedestrian, double, double, double, double)* 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 **??**. En esta clase los métodos más importantes son

  populate_comuna() 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 2 métodos importantes son *getState()* y *updateView()*. El primero maneja la construcción del gráfico de áreas mientras que el segundo va actualizando la posición de los individuos en la interfaz gráfica.

- **Vacunatorio**: Crea el área correspondiente a un vacunatorio, recibiendo como atributo una comuna y definiendo como origen un punto aleatorio entre 0 y la diferencia entre el ancho/largo de la comuna con el tamaño del lado del 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: *takeAction()* el cual llama a todos los

métodos actualizadores cada cierto tiempo y permite el manejo del mismo con las flechas izquierda y derecha del teclado, *start()* que inicializa la simulación y *stop()* que para la simulación.

- **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.

- **VacView, PedestrianView y ComunaView**: Estas 3 clases se encargan de crear la animación en la interfaz gráfica de las clases Vacunatorio, Pedestrian y Comuna respectivamente.
- **settingsControl**: 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.
- **SimulatorConfig**: Se encarga de almacenar como parámetros modificables los datos ingresados a través del *configurationFile.txt*.
- **SimulatorMenuBar**: Crea el menu de la interfaz gráfica y sus items Control y Setting. Además tambien maneja el funcionamiento de ambos botones.

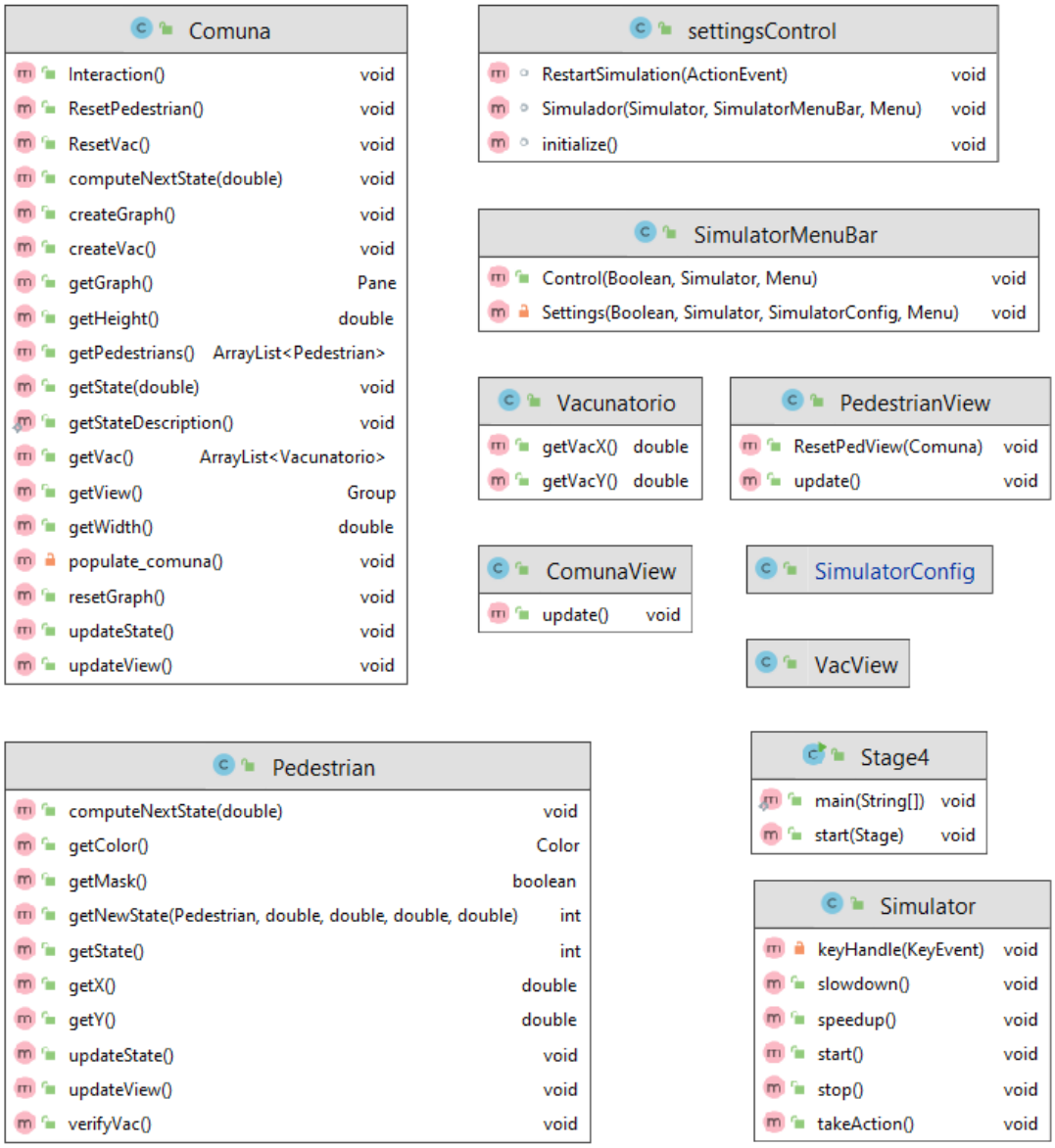

Figura 2: Diagrama de clases con sus respectivos métodos para la Etapa 4.

1.3. Dificultades

Algunas dificultades fueron:

1. Que los individuos vacunados adopten forma triangular. Dado que en un principio para genearar los circulos correspondientes al estado de ifectado solo se suavizaban los bordes del rectangulo original, el cambio a triangulo implicaba una aproximación distinta.

   Como solución se tuvo que importar la clase polygon, y al mismo tiempo que se hacía transparente el cuadrado, se crea un nuevo objeto de tipo poligono con la posición del individuo a representar.

2. Que el área de los vacunatorios visible en la interfaz gráfica coincida con el área con el que trabaja el computador. En una primera instancia los individuos se vacunaban en áreas que no correspondían a las representaciones visuales de la simulación, dicho de otra manera el momento en que se vacunaban los peatones no necesariamente era cuando pasaban por los vacunatorios. Esto se debía a que en un principio no se consideraron las diferencias entre como se construye un rectángulo como objeto y como se construye la visualización de un rectángulo.

   Como solución solo fue necesario ajustar los parámetros con los que se estaba generando el componente visual del vacunatorio.

3. Como forma de simplificar el proceso de pasar a una siguiente etapa de la tarea, considerando que se en cada una se debían agregar cosas a lo ya previamente hecho, se decidió aprender a usar Scene Builder, lo que en un principio tuvo la dificultad de relacionar lo construido con el programa.

4. Para que los colores del gráfico coincidieran con los de los individuos se tuvo que trabajar con un tipo de archivo desconocido.