UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Documentación Tarea 2: Aplicación Gráfica para Semáforos en Intersección

ELO329 Diseño y Programación Orientada a Objetos

Profesor: Cristobal Nettle

Nombres alumnos: Paula Amigo (201504013-3)
Luis Bahamondes (201421077-9)
Jairo González (201304502-2)

Fecha de entrega: Viernes 3 de Abril 2019

Índice

1. Descripción del Desafío

El desafío planteado consta en crear una interfaz gráfica que represente el comportamiento de los semáforos en la intersección de las Avenida Los Placeres y Manuel Antonio Matta. Sin embargo, se consideran solo x semáforos ya que el resto son duplicados de estos. En el sistema hay además dos botones cuya finalidad es solicitar la luz verde para cruces peatonales, y un sensor inductivo que detecta autos que necesiten doblar a la izquierda desde Los Placeres a Matta.

La interfaz constará de un esquema con los semáforos a simular y sus respectivos estados en cada momento. Además, tendrá una parte donde se encontrarán 3 botones, uno para cada botón de cruce peatonal y uno para el sensor inductivo, los que podrán ser presionados interactivamente en esta interfaz para solicitar ya sea cruce o giro. Así, se simulará la llegada de peatones o autos, y como esto afecta el estado de los semáforos.

2. Objetivos

- Manejar proyectos vía GIT.
- Crear Interfaces gráficas en Java.
- Manejar eventos de software.
- Ejercitar la creación y extensión de clases para satisfacer nuevos requerimientos.
- Ejercitar el patrón de diseño "Modelo-Vista-Controlador". Generar documentación usando Javadoc.

3. Uso de GIT

Esta tarea se desarrolló en GitLab, servicio web de control de versiones y desarrollo de software colaborativo. Sin embargo, a veces resulta poco intuitivo el resultado de las acciones realizadas en los repositoriosm con este software. Debido a esto, se utilizó el programa *GitKraken*, el cual ofrece una interfaz gráfica y distintas herramientas visuales para convertir las operaciones con los repositorios en tareas más intuitivas y sencillas de realizar.

Con GitKraken, se puede ver el panel de commits correspondientes a un proyecto. En este caso, este panel se muestra en la figura 1.

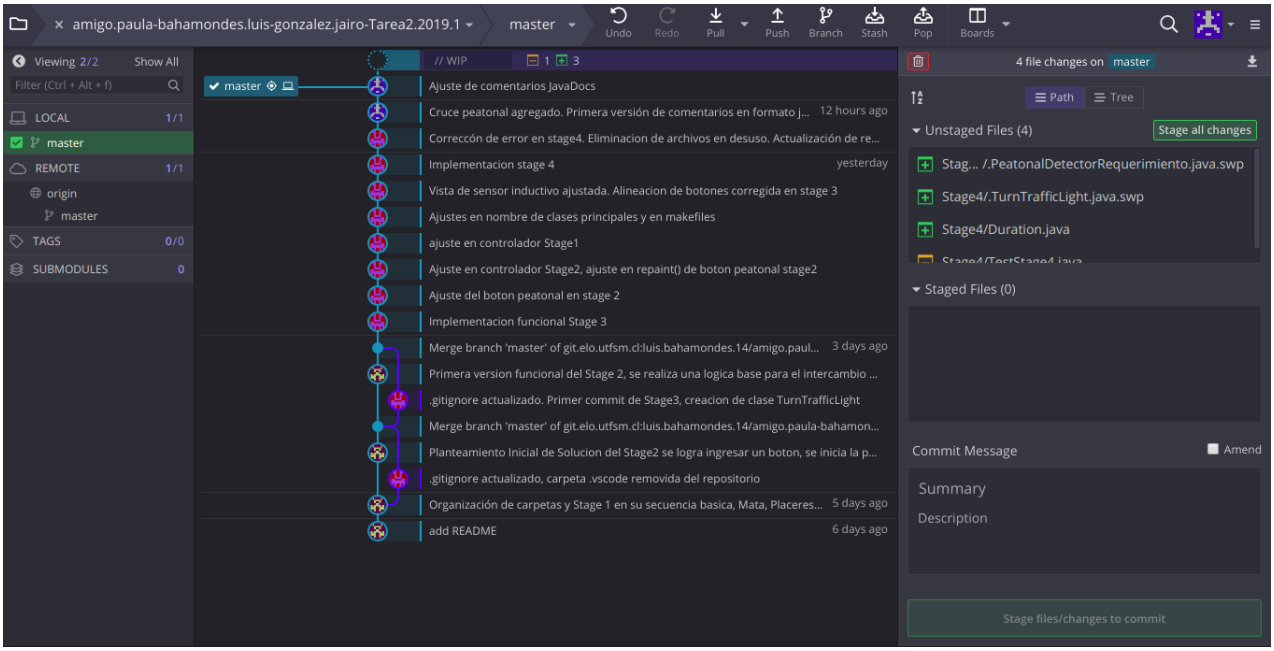

Figura 1: Panel de Commits del Proyecto

El programa también permite hacer la vista diferencial de un archivo en particular, en la cual se muestran los cambios que se le aplicaron a un archivo luego de un *commit*. Las líneas de código agregadas se destacan con color verde, mientras que las eliminadas con color rojo. Además se puede ver el número de linea al que corresponde una linea de código tanto en la versión actual como en la anterior.

Esto se ejemplifica en la figura 2, donde se tomó el archivo *Controller.java* de la Etapa 2 luego del *commit* 'Ajuste en controlador Stage2, ajuste en repaint() de boton peatonal stage2'.

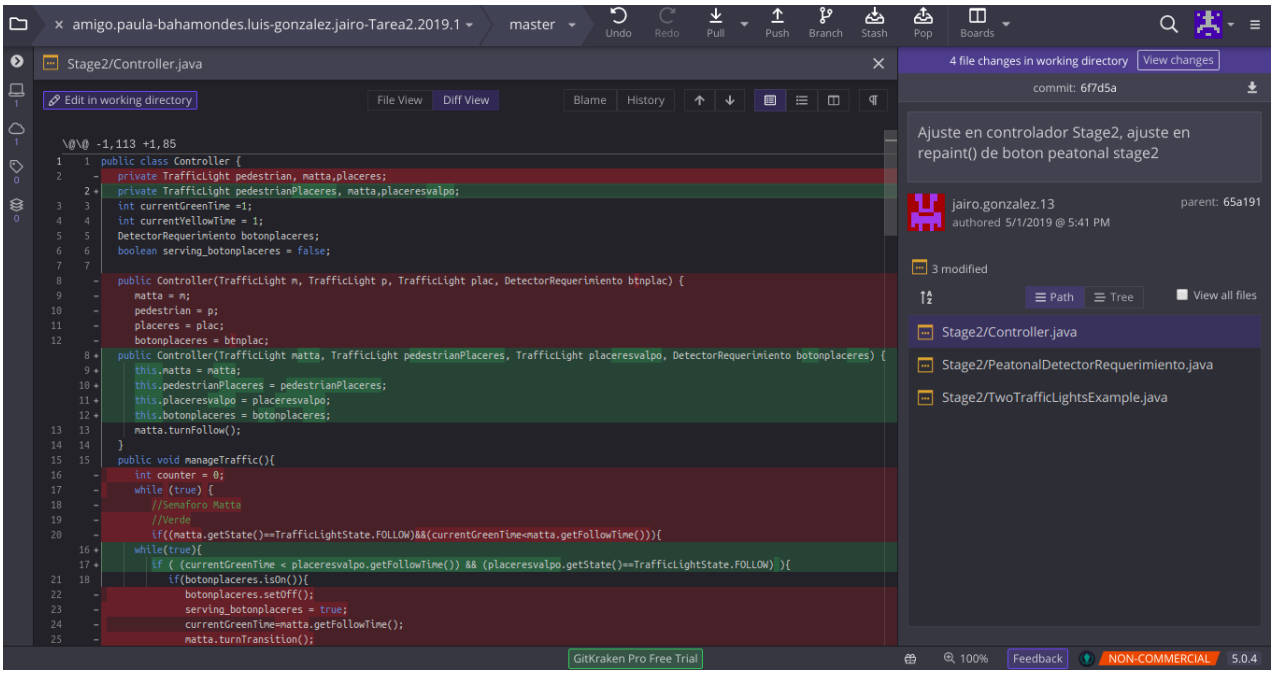

Figura 2: Vista Diferencial en GitKraken

4. Diagrama de Alto Nivel - Stage 4

A continuación se presenta un diagrama general de alto nivel que representa las interacciones entre las diferentes clases pertenecientes a la etapa 4 (Stage4).

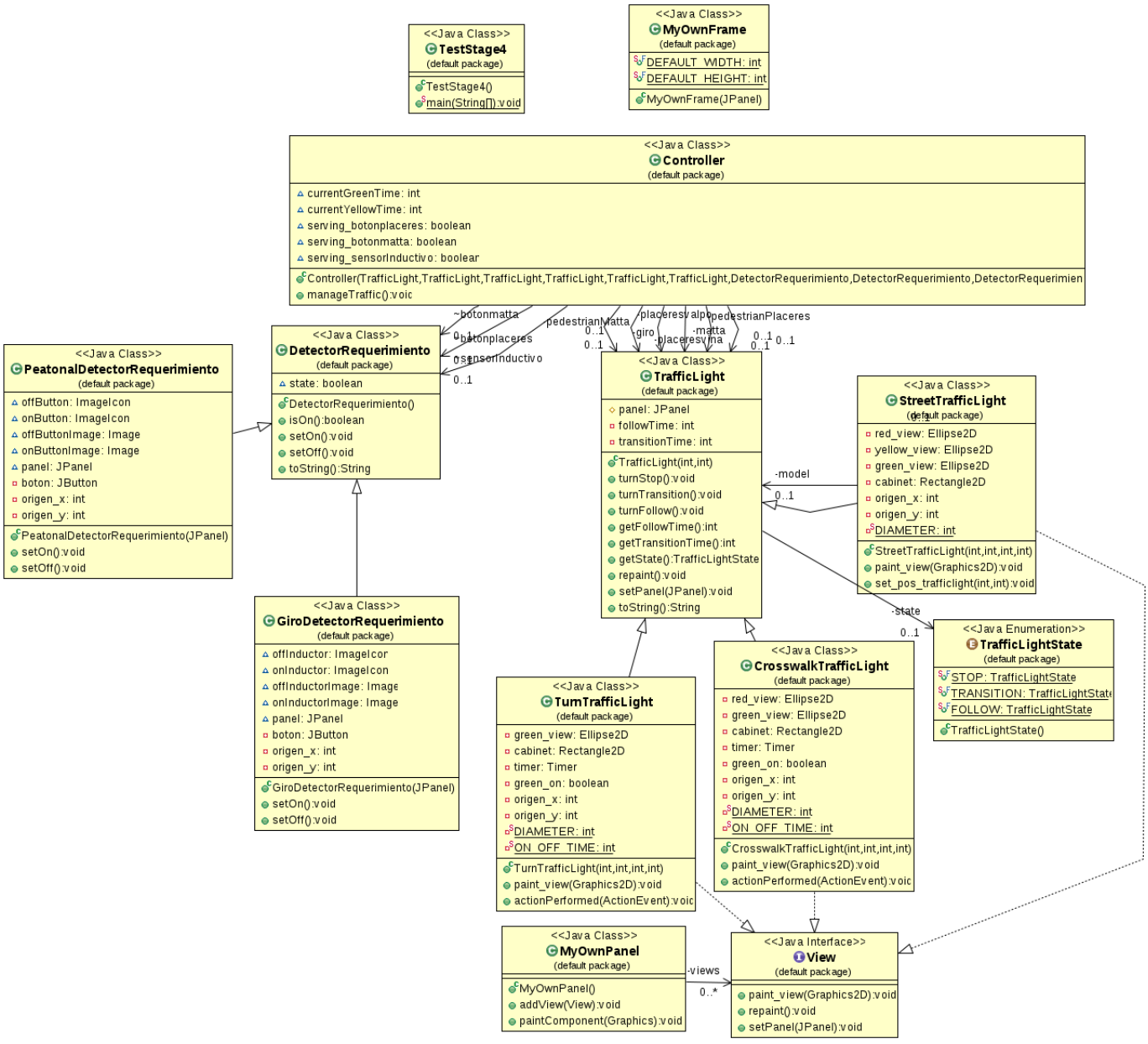

Figura 3: Diagrama de Alto Nivel Stage 4

5. Explicación de las clases - Stage 4

A continuación se explican las clases que componen el stage final del proyecto.

5.1. TrafficLight

Clase base de los semáforos, el modelo. Esta clase incluye los métodos para cambiar de estado entre FOLLOW, TRANSITION y STOP. Aquí se definen los parámetros que contienen los tiempos en luz verde y amarillo. Es posible leer los tiempos de funcionamiento y el estado actual mediante los métodos definidos. Al cambiar de estado, se ejecuta un repaint() para actualizar los gráficos del semáforo.

5.2. TrafficLightState

Es una clase *enum*, (clase especial que limitan la creación de objetos a los especificados explícitamente en la implementación de la clase) que enumera los 3 posibles estados del semáforo: FOLLOW, TRANSITION y STOP.

5.3. View

Esta clase engloba las funciones escenciales para la representación gráfica de elementos 2D, como los semáforos y la intersección misma de calles. Se compone de los métodos $pain_view()$, $repaint()$ y $setPanel()$,los cuales son redefinidos en cada clase de vista pertinente.

5.4. StreetTrafficLight

Esta clase hereda de *TrafficLight*. Corresponde a los semáforos de tres tiempos y se compone de los métodos $paint_view()$ y $set_pos_trafficlight$. Se definen los párametros necesarios para graficar este tipo de semáforo, y pintarlos según su estado.

5.5. CrosswalkTrafficLight

Esta clase hereda de *TrafficLight*. Corresponde a los semáforos peatonales y se compone de los métodos $paint_view()$ y $action_performed$. Se definen los parámetros necesarios para graficar este tipo de semáforo, y pintarlos según su estado.

5.6. TurnTrafficLight

Esta clase hereda de *TrafficLight*. Corresponde a la luz de giro de un semáforo, en este caso del que se encuentra en Avendia Los Placeres. Se compone de los métodos $paint_view()$ y $action_performed$ y se definen los parámetros necesarios para graficar este tipo de semáforo, y pintarlos según su estado.

5.7. DetectorRequerimiento

Basado en la tarea 1, la clase DetectorRequerimiento conserva su estructura. Esta clase permite configurar como encendido o apagado algún requerimiento: solicitud de cruce peatonal o solicitud de giro vehicular. Representa el modelo para los sensores y botones.

5.8. PeatonalDetectorRequerimiento

Esta clase hereda de la clase DetectorRequerimiento. Permite implementar la vista de un botón de cruce peatonal.

5.9. GiroDetectorRequerimiento

Esta clase hereda de la clase DetectorRequerimiento. Permite implementar la vista de un sensor inductivo para la solicitud de giro vehicular.

5.10. Controller

La clase controller representa el controlador de todos los modelos. En esta clase se define el estado inicial de la intersección vial y se define el método mannageTraffic(), el cual funciona igual que en la tarea 1, pero con diferencias en nombres de variables.

Respecto a la versión de la tarea anterior, en esta tarea se quitaron las consideraciones de disminuir tiempos de otros semáforos o de quitar la poción de cruzar por quedar poco tiempo de verde en el semáforo de la calle contraria (implementados anteriormente para hacer cambios de luces prudentes y seguros vialmente).

En la versión actual, una solicitud de cruce peatonal gatilla una transición en el semáforo vehicular. No se puede solicitar cruce si el cruce ya está disponible. Si está en transición se considera disponible.

La luz de giro al igual que en la tarea anterior, se pone en cola hasta que sea el turno de avanzar en calle placeres y se dará permiso de giro al finalizar el tiempo en verde el semáforo desde Valparaíso a Viña del Mar.

5.11. TestStage4

Clase principal, la cual incluye el método main(). Aquí se crean todos los objetos que participan en el funcionamiento de la intersección vial (semáforos, botones, sensores).

Es en esta clase donde se crea la ventana que contiene los elementos gráficos, organizados en dos paneles (centro y sur), donde el panel centro incluye los gráficos 2D y el panel sur

incluye los botones y sensores organizados en un BoxLayout.

Incluye el método respectivo para pintar objetos 2D y para crear el frame principal.

6. Dificultades y Soluciones

Una de las dificultades fue la implementación de imágenes como botones, para implementar así el botón de cruce peatonal y la bobina de detección del semáforo de giro. La dificultad recae en pintar el botón nuevamente una vez que el requerimiento ya ha sido satisfecho, y por otro lado existió la dificultad de posicionarlo en el lugar correcto.

Para solucionarlo, se investigó como agregar iconos a los botones y se aprendió que los botones ya incluyen su propio método de *repaint()* que permite actualizar el botón cuando corresponda.

Para posicionar correctamente los botones se pensó en un principio en no utilizar layouts y aplicar posicionamiento absoluto, pero esto resultó en errores de superposicion de elementos y gráficos que no se dibujaban. Finalmente se optó por integrar diferente layouts para lograr la disposición de botones deseadas (BorderLayout y BoxLayout).