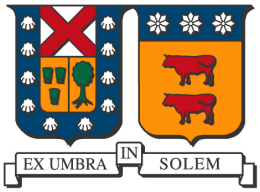


Documentación

Tarea 2: Aplicación Gráfica para Robots en Laberinto

Programación Orientada a Objetos - ELO 329
Departamento de Electrónica

Valparaíso, 30 de Abril del 2018

Integrantes	**Rol**
Matías Contreras	201321034-1
Damian Quiroz	201321056-2
Francisco Frez	201321007-4
Profesor	Agustín González
Ayudantes	Jesús Márquez
	Pilar Arancibia

Índice

1. Objetivos

En esta tarea se busca poner en práctica los conocimientos de programación orientada a objetos en lenguaje Java sobre creación, modificación y uso de interfaces gráficas, abarcando el uso de clases predefinidas en conjunto con la generación de nuevas clases, métodos e instancias para responder al problema propuesto.

Se propone abordar la situación planteada mediante un desarrollo iterativo e incremental por etapas, cumpliendo con los siguientes 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-Controlado".
- Generar documentación usando "Javadoc".

2. Descripción General

Se pide crear una interfaz gráfica para el robot definido en la tarea 1, con tal de agregar algunas funcionalidades para interactuar con la aplicación.

La interfaz gráfica del usuario incluye una barra de menú, una zona de despliegue, el mundo que incluye el laberinto y uno o más robots en él, además de un botón en la parte baja para pausar o dar continuidad al tiempo.

La barra de menú contiene el menú "**File**", dentro del cual, el usuario tiene el ítem "**Open**" (la aplicación pide al usuario seleccionar el laberinto a utilizar haciendo uso de la clase "**JFileChooser**"). Además, contiene el menú "**World**", tal que bajo él se despliegan dos ítems: "**Create Robot**" (define los parámetros propios del robot y le da elección al usuario para eligir su ubicación de inicio de recorrido en el laberinto) y "**Set delta t**" (define los intervalos del tiempo de simulación).

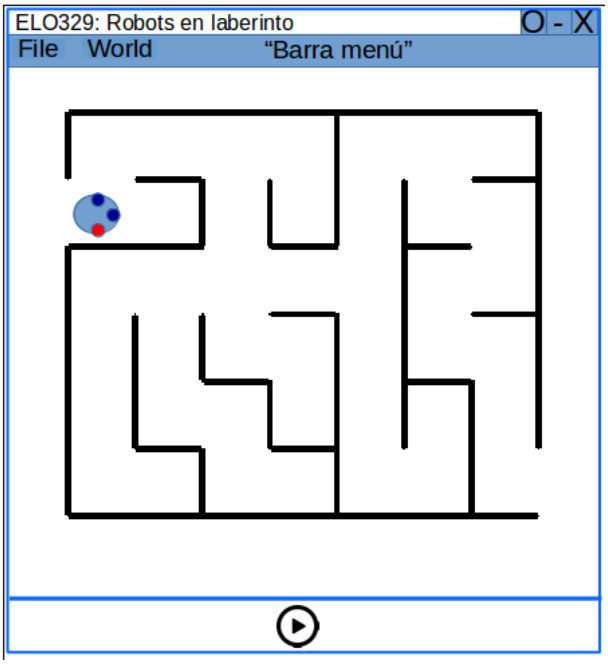


Figura 1: Ejemplo de Interfaz Gráfica: un Laberinto y un Robot.

2.1. Consideraciones Generales

1. Los robots son transparentes entre sí, por lo que pueden pasar uno sobre otro.
2. Cuando el usuario decide especificar un archivo para registrar la ruta del robot, este registra una tabla de valores incluyendo **tiempo**, **posición x** y **posición y** del robot por cada línea del archivo de texto.
3. Sólo se puede insertar nuevos robots cuando el tiempo está en pausa.
4. Cuando un robot sale del laberinto, no es importante por donde se sigue moviendo.
5. El programa termina cunado el usuario cierra la ventana principal de la aplicación.

3. Desarrollo por Etapas

Se aplica una metodología "iterativa" e "incremental" para el desarrollo de esta solución, obteniendo en cada etapa un subconjunto del requerimiento final.

3.1. Etapa 1: Ventana Principal con Menú File

En esta etapa se muestra la ventana principal de la aplicación con la programación para leer el laberinto y su despliegue en la zona central de la ventana, incluyendo un botón **Play** que sirve para pausar o iniciar (no generará más efecto que cambiar su vista), donde la aplicación termina al cerrar la ventana principal.

3.1.1. MainPanel.java

Clase que incluye los elementos gráficos para dibujar el laberinto. Define en su constructor la instancia *"SCALE_TRANSFORM"* que permite redimensionar el laberinto pasado como argumento, utilizando los métodos **"getScale()"** (obtiene la escala de dibujo) y **"setMaze()"** (setea el laberinto) como ayuda.

El método principal **"paintComponent"** se encarga de realizar el pintado del laberinto, así como implementar su cambio de tamaño en la ventana.

3.1.2. MyTime.java

Clase que define la interacción de los botones al interior del panel. Tiene como constructor **"MyTime"**, el que se encarga de definir las imágenes a utilizar y los eventos necesarios para generar la interacción al interior de la ventana, incluyendo además el método **"getView"** que permite la aparación del botón en pantalla.

3.1.3. Maze.java

Clase construida y utilizada en la tarea anterior que implementa los mismos métodos ya explicitados (obvia algunos que resultan innecesarios), siendo su principal novedad el método **"draw"** que dibuja el laberinto en pantalla.

3.1.4. Vector2D.java

Clase que posee como atributos las variables **x** e **y** (plano cartesiano), implementando métodos correspondientes a un vector en dos dimensiones tales como: **"plus"** (suma vectorial), **"times"** (producto escalar), **"minus"** (resta vectorial), **"setTo"** (setea nueva posición), **"getX"** (obtiene variable x), entre otros.

3.1.5. Stage1.java

Clase que contiene el método **main**, donde se definen los parámetros y el principal frame del programa. Aquí se definen los principales elementos asociados a la aplicación gráfica como la inclusión de botones o paneles a conveniencia.

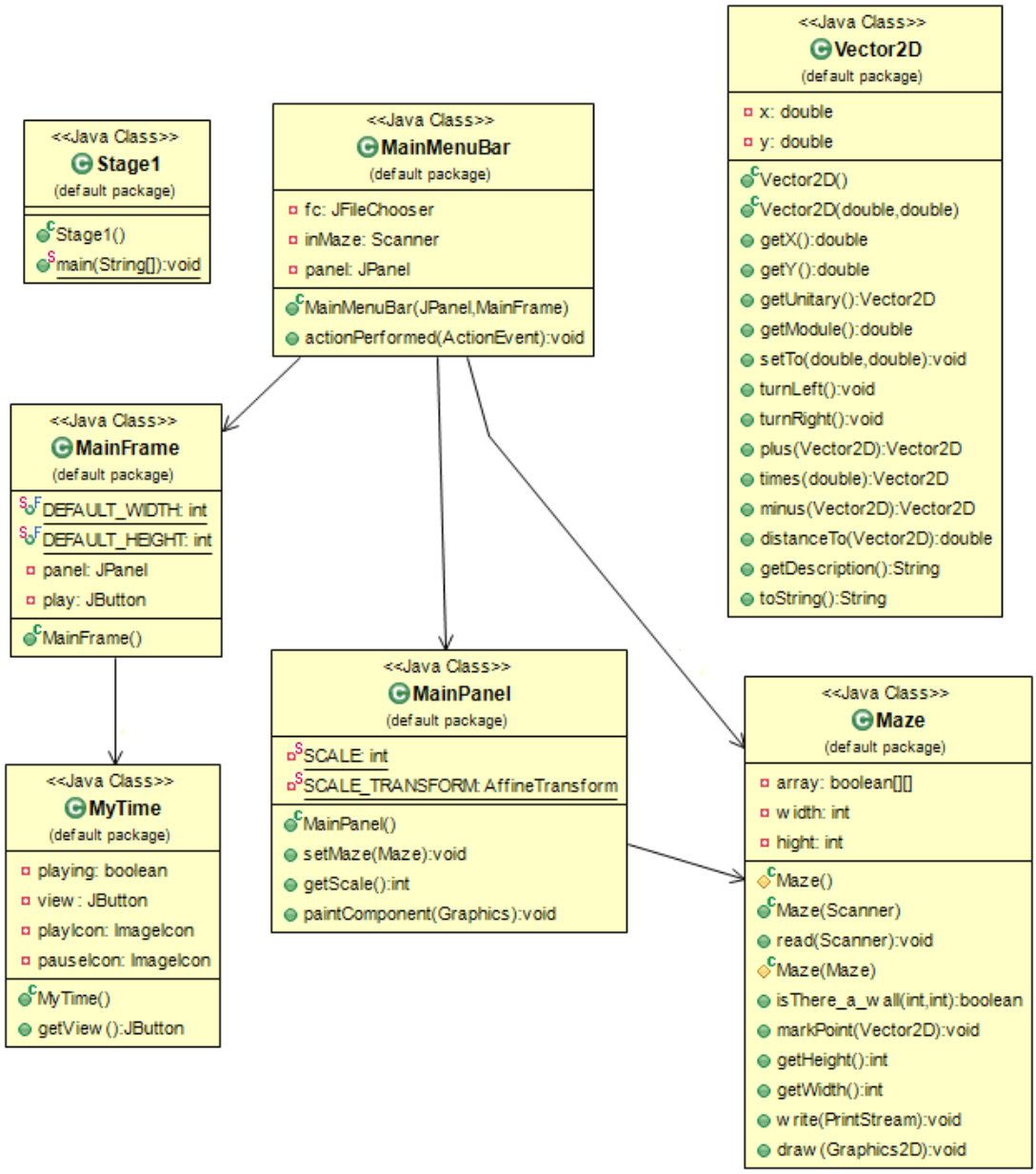
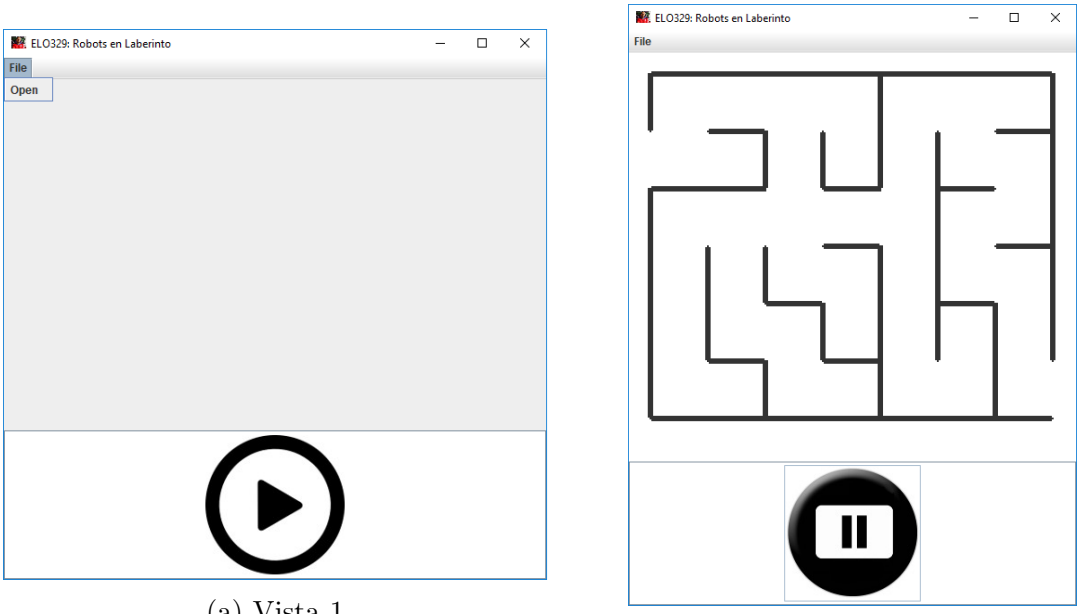


Figura 2: Diagrama UML: Stage1.

3.1.6. Resultados

Se observa en la Figura 3 que el botón interactúa de forma correcta por medio de la ventana. Por otro lado, el menú *"File"* creado despliega la opción *"Open"* sin problemas, dando la opción de búsqueda de algún archivo cuya extensión sea "pbm" para ser incluido en la ventana como el laberinto a utilizar.



(a) Vista 1

(b) Vista 2

Figura 3: Resultados Stage1.

3.2. Etapa 2: Inclusión Menú World con Robot Predefinido

En esta segunda etapa se incluyen dos ítems para el menú **"World"**: *"Set delta t"* define el incremento discreto para el tiempo; sin embargo, el tiempo aún no será activo con el botón **"Play"**. El ítem *"Create Robot"* hará aparecer un robot en una posición fija con velocidad (orientación) también predefinida. El tipo de piloto (estrategia de navegación) no es relevante en esta etapa.

3.2.1. MyWorld.java

Clase que correspondiente a **"MainPanel"** que presenta una mezcla de métodos de la tarea pasada con los actuales métodos gráficos, siendo los más importantes **"paintComponent"** (pintado y escalado del laberinto) y **"simulate"** (permite correr el tiempo de simulación para el movimiento de los objetos).

3.2.2. Pilot.java

Clase que simula ser el piloto (creada e implementada en la tarea anterior). Su principal método es **"setCourse"** que implementa el algoritmo de movimiendo "pared a la derecha" (robot avanza apegado a su pared derecha).

3.2.3. MyPilot.java

Clase que hereda de **"Pilot"** (características propias del constructor base) y define el algoritmo de movimiento "pared a la izquierda" (robot avanza apegado a su pared izquierda) mediante el método **"setCourse"**.

3.2.4. Robot.java

Clase que define los atributos del robot y los sensores que van empotrados a él. No recibe modificaciones respecto de la tarea pasada, siendo sus principales métodos: giro a la derecha (**turnRight**), a la izquierda (**turnLeft**), avanzar (**moveDelta_t**), **moveRobot** y **desicion** que definen el método de movimiento del objeto.

3.2.5. RobotView.java

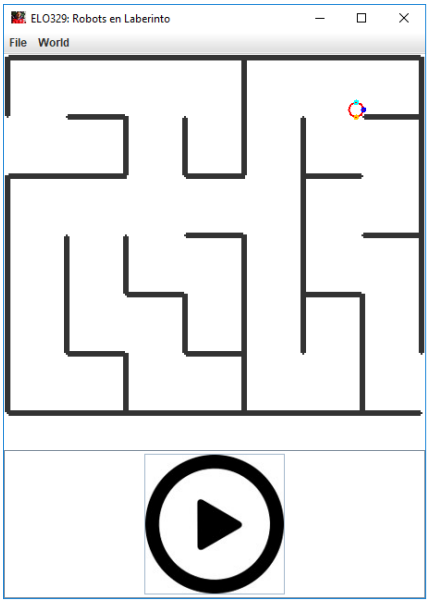
Clase que implementa los métodos necesarios para generar el robot al interior de la ventana de simulación. Su principal método es "**paintComponent**", el que se encarga de pintar el robot (óvalo rojo) y cada uno de los sensores (óvalos con diferentes colores) en un sector predefinido del entorno de simulación (JPanel).

3.2.6. Stage2.java

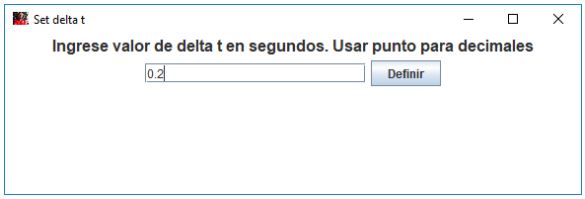
Clase que contiene el método **main**, donde se definen los parámetros y el principal frame del programa. Aquí se definen los principales elementos asociados a la aplicación gráfica como la inclusión de botones, paneles a conveniencia y pintado de diferentes objetos como el laberinto, robot (uno para esta etapa) y sensores.

3.2.7. Resultados

Se observa en la Figura 4 el resultado obtenido en la presente etapa, tal que el robot se encuentra en el sector superior derecho del laberinto y posee los 3 sensores empotrados, cada uno de diferente color.



(a) Robot en Laberinto.



(b) Configuración *delta_t*.

Figura 4: Resultados Stage2.

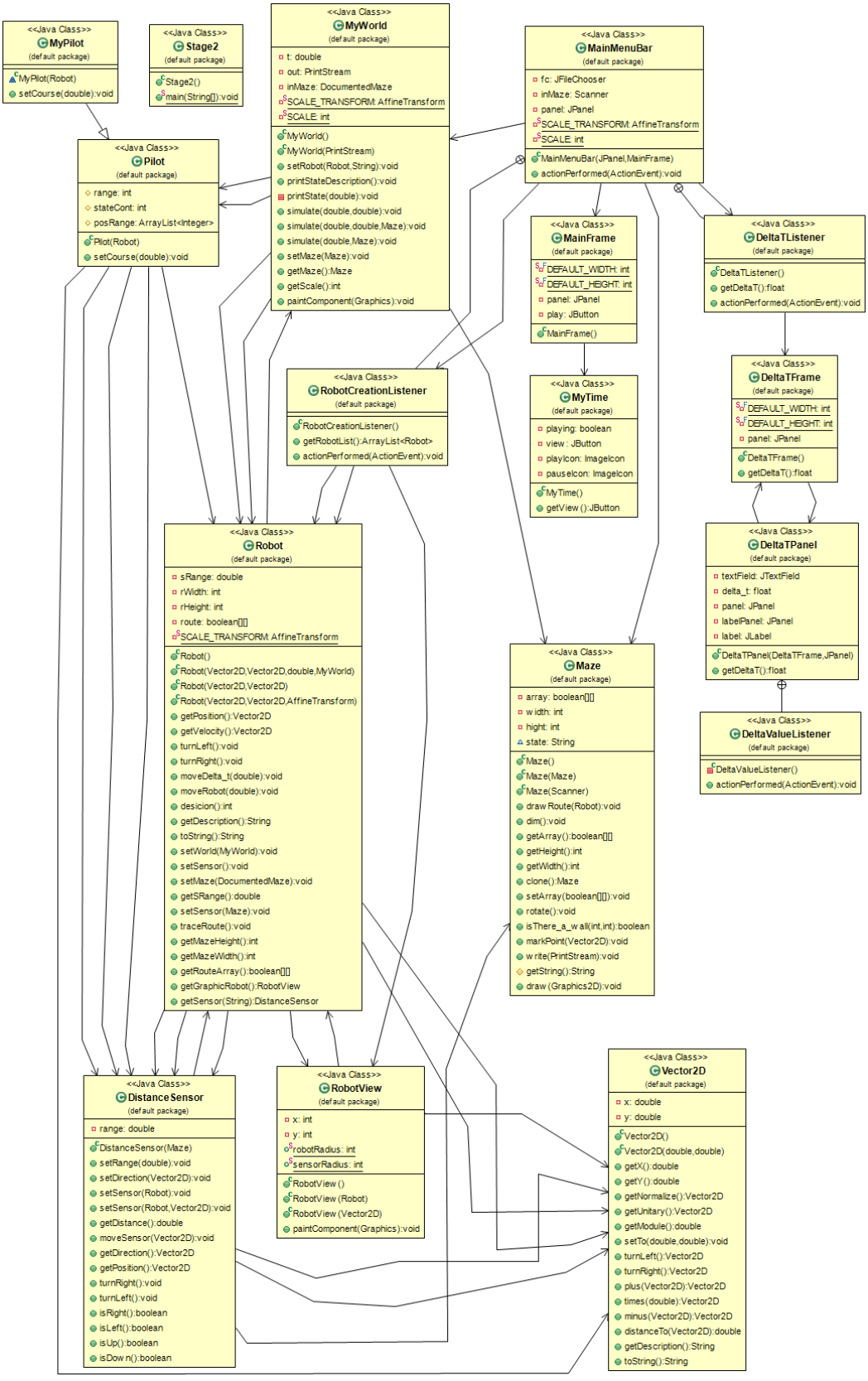


Figura 5: Diagrama UML: Stage2.

3.3. Etapa 3: Robots Parametrizados por Usuario

En esta etapa el usuario puede definir la velocidad y el tipo de piloto (estrategia apegado a pared derecha o apegado a pared izquierda) de cada robot creado (no se pide ofrecer opción de registrar la ruta seguida). Posteriormente, aparece un robot movible junto al mouse, tal que, al presionar el botón izquierdo de este, el robot es liberado y puesto en el lugar del evento.

Como último paso, al presionar el botón **Play**, el usuario observa el movimiento del robot por el laberinto. Al pausar la simulación, el menú **"World"** permite la creación de nuevos robots mediante *"Create Robot"*, destacando que la vista de cada sensor debe cambiar de color cuando detecta una pared cercana.

3.3.1. DeltaTFrame.java

Clase que define la ventana para la configuración de *delta_t*. Crea un frame que incluye todas las figuras y entrada de texto para definir el valor de *delta_t*.

Posee 2 constructores esenciales: **DeltaTPanel** (crea la ventana donde irá todo) y **DeltaTFrame** (se incluyen los elementos gráficos más los de texto), además del método **DeltaValueListener** (implementa la programación por eventos).

3.3.2. RobotConfigurationFrame.java

Clase que define la ventana para la configuración del robot. Crea un frame que incluye los textos necesarios, así como la programación por eventos para recibir, leer información y reaccionar ante la interacción del mouse.

sus principales métodos son **"getTextField_vx()"**, **"getTextField_vy()"** (obtiene la velocidad en x e y, respectivamente) y **"getTextField_pilot()"** (define el tipo de piloto, tal que **1** es apegado a la derecha y **0** a la izquierda).

3.3.3. Stage3.java

Clase que contiene el método **main**, donde se definen los parámetros y el principal frame del programa. Aquí se definen los principales elementos asociados a la aplicación gráfica como la inclusión de botones, paneles a conveniencia y pintado de diferentes objetos como el laberinto, robots (permite varios a la vez) y sensores.

Respecto de la Stage2, se generó un archivo aparte para limitar su cantidad de clases y métodos (*"DeltaTFrame.java"*), además

3.3.4. Resultados

Se observa en la Figura 6 la ventana creada para la configuación de los parámetros del robot (ingresar velocidad en x, en y, además del tipo de piloto) y en la Figura 7 los robots agregados al laberinto, incluyendo su momento inicial y final previo a cerrar la ventana de simulación (botón **Play** totalmente funcional).

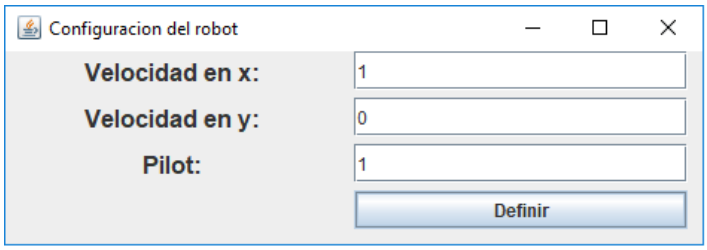
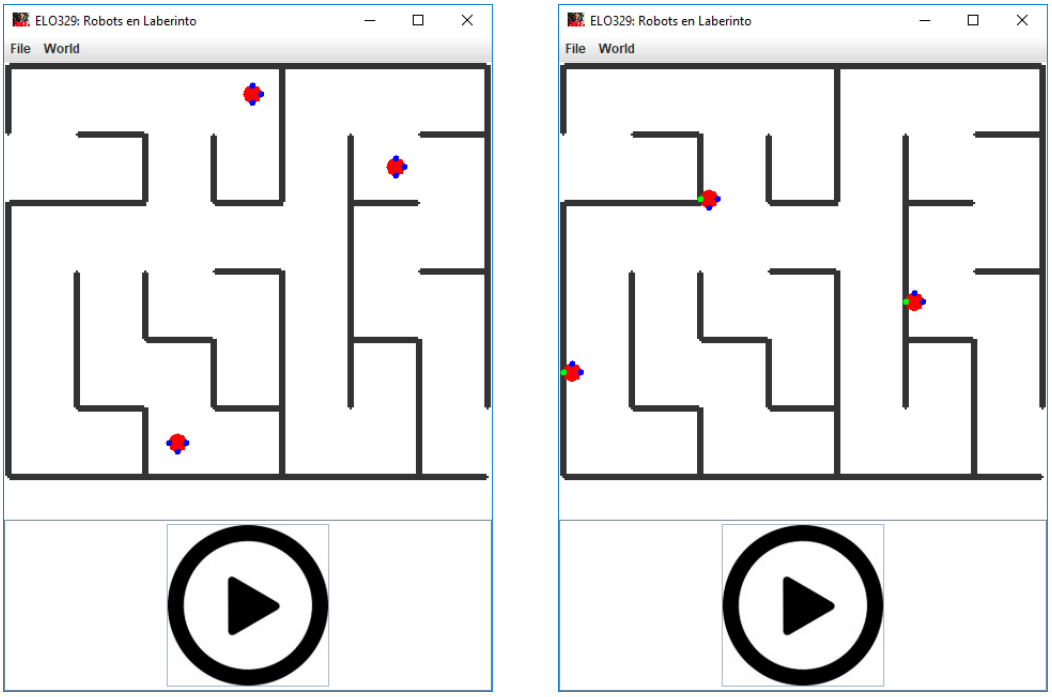


Figura 6: Ventana de Configuración del Robot.



(a) Robots en Laberinto (inicio). (b) Robots en Laberinto (fin).

Figura 7: Resultados Stage3.

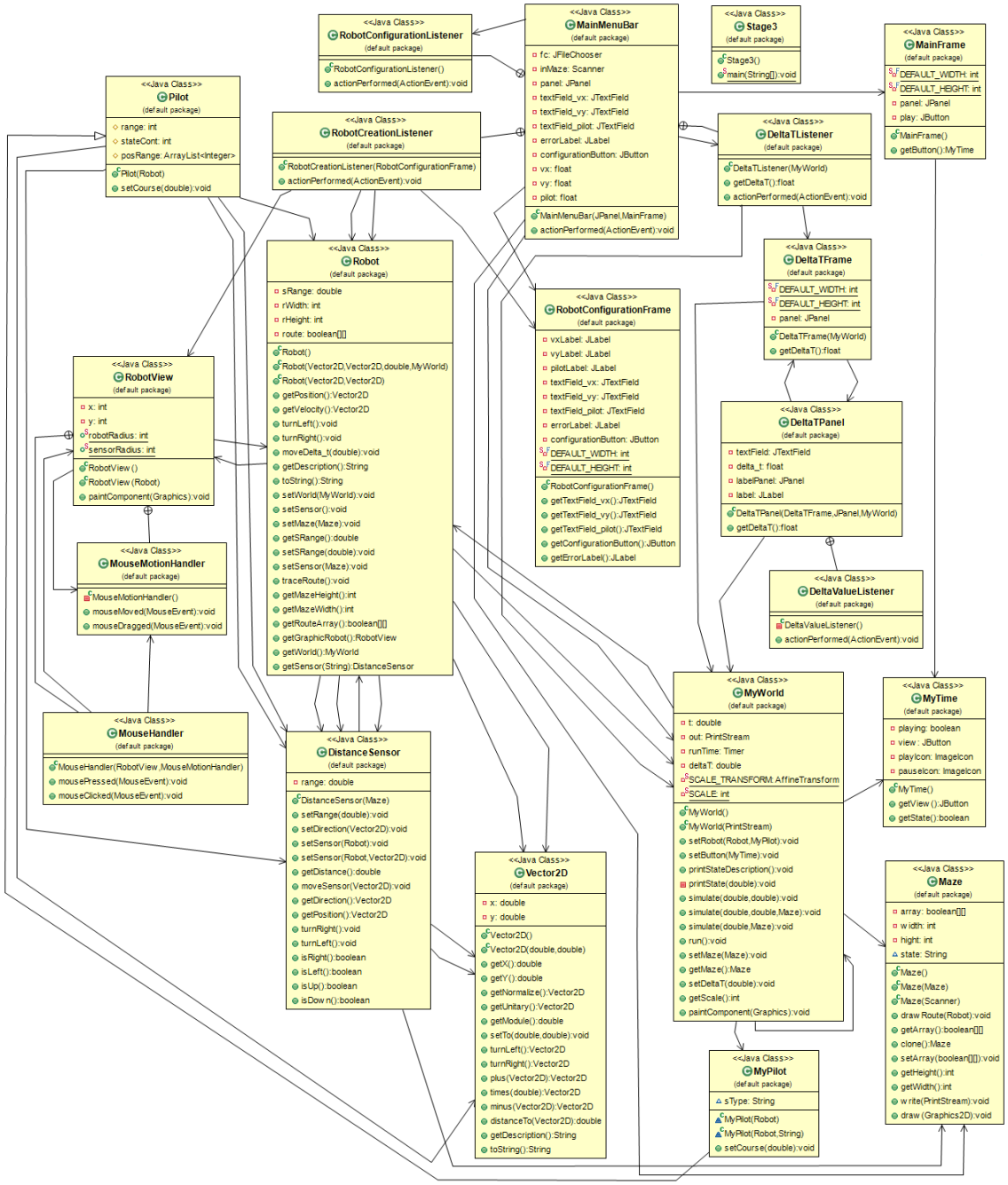


Figura 8: Diagrama UML: Stage3.

3.4. Etapa 4: Registro de Trayectoria

En esta última etapa, el programa permite definir un archivo de salida para cada robot, tal que el robot registra su posición conforme el tiempo avanza al interior de la simulación (los momentos de pausa no son reflejados).

3.4.1. Interpretación: Generación de Archivo de Salida

Dadas las indicaciones de esta etapa y la interpretación que estas daban, se tomó la determinación de generar en forma **automática** el archivo de salida en formato *csv* que indica el camino seguido por el robot, indicando además que se incluyó una salida tipo *pbm* que indica lo mencionado de manera gráfica.

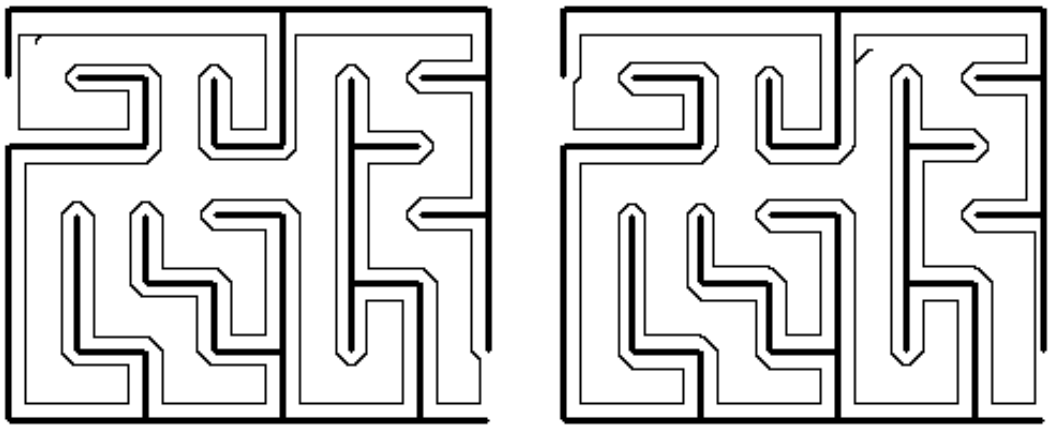
3.4.2. Stage4.java

Clase que contiene el método **main**, donde se definen los parámetros y el principal frame del programa. Aquí se definen los principales elementos asociados a la aplicación gráfica como la inclusión de botones, paneles a conveniencia y pintado de diferentes objetos como el laberinto, robot (uno para esta etapa) y sensores.

Se incluye la etapa extra, tal que se genera un archivo pbm con el laberinto para cada robot que puede ser transformado según formato de conveniencia, observando la trayectoria seguida en él (se indica en archivo README).

3.4.3. Resultados

Se observa en la Figura 9 los caminos seguidos por dos robots seteados al interior del laberinto (se denota que tienen pilotos diferentes, situación destacable en ciertos sectores). Además, es importante mencionar que, dado que no se agregaron clases ni métodos adicionales, posee el mismo diagrama UML.



(a) Camino Robot 1. (b) Camino Robot 2.

Figura 9: Resultados Stage4.

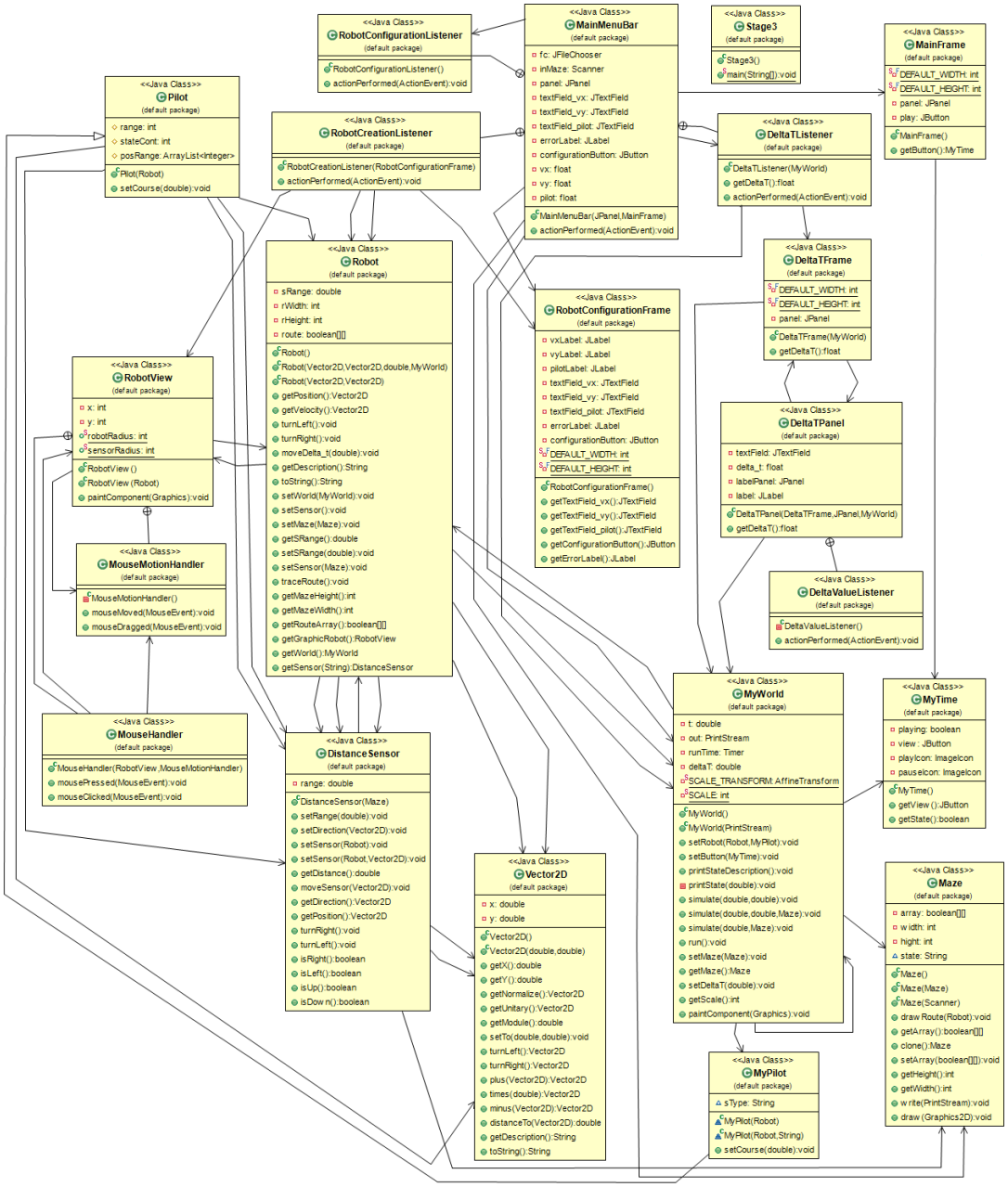


Figura 10: Diagrama UML: Stage4.

4. Problemas Presentados

4.1. Stage2: Problemas con Eclipse

Se denota que todos los integrantes del grupo trabajan con Eclipse para el desarrollo de las tareas, tal que uno de ellos presentó problemas (nula carga de imágenes en la ventana creada) con este IDE por motivos no identificados siquiera al depurar el código en más de una oportunidad, siendo aún más raro el hecho de que funcionase en óptimas condiciones en los computadores de los otros integrantes.

Este hecho no pudo ser resuelto por ningún integrante y se cree tiene relación con la versión de Eclipse utilizada en dicho computador o los paquetes de Java incluidos al tratar de correr la Stage2 (se actualizó y tampoco funcionó).

4.2. Implementación Tarea 1

Dado que la presente tarea requiere del óptimo funcionamiento de la anterior, se presentaron inconvenientes en ciertos detalles de algunas clases correspondientes a la tarea pasada como lo fueron "*Pilot.java*" y "*MyPilot.java*" (definen los algoritmos de movimiento en el laberinto: pegado a la derecha e izquierda).

Estas y todas las clases fueron revisadas entre los integrantes para evitar errores de implementación en la tarea actual, arreglando warnings y errores.

4.3. Escala de Dibujos en JPanel

Se presentó el problema de escalar la imagen del laberinto en la Stage1 dado los requerimientos de la misma, debiendo leer, informarse y pensar demasiado para dar una solución óptima. Se solucionó con facilidad ante la ayuda dada por el profesor para el desarrollo de la Stage2, implementando dicha sección de código en la Stage1 y utilizarlo cuando fuera necesario en otras etapas.

4.4. Dificultad Tiempo Simulación

Dada la Stage3 en donde se solicita iniciar la simulación con el botón **Play**, resultó un verdadero desafío unir lo desarrollado en la tarea 1 con las interfaces gráficas de la presente tarea para que todo funcionase de manera óptima.

Se solucionó esta situación mediante los códigos implementados en dicha etapa, estando el meollo del asunto en la clase **"MyWorld.java"** (define los métodos para la obtención y uso del *delta_t* definido por el usuario).

5. Conclusiones y Comentarios

Se concluye en la presente tarea que el desarrollo de interfaces gráficas en Java es bastante amigable dada la gran cantidad de clases existentes que pueden utilizarse, en las cuales, vienen una serie de constructores y métodos implementados tal que basta sólo extenderlos en las nuevas clases creadas.

Se destaca que ciertos conceptos tales como herencia, casteo y visibilidad fueron fundamentales para determinar las relaciones entre las diferentes clases, tal que permitieron la creación de archivos desde cero para ser incluidos.

Se menciona la importancia del desarrollo de documentación mediante Javadoc, aprendiendo a utilizar esta útil herramienta que da la posibilidad de desarrollar la documentación de los códigos creados en forma inmediata.

Se denota la importancia del desarrollo iterativo e incremental planteado para el desarrollo de esta tarea, táctica de gran utilidad para comprender que hace cada clase, método e instancia necesaria en el logro de la solución final.