Diseño y Programación Orientados a Objetos
1er. Sem 2018
Tarea 2: Aplicación Gráfica para Robots en Laberinto

Lea detenidamente la tarea. Si algo no lo entiende, consulte en clases. Si es preciso, se incorporarán aclaraciones al final.
Objetivos de la tarea:
* 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.

Descripción General

En esta tarea se pide crear una interfaz gráfica para el robot definido en la Tarea 1. En la interfaz gráfica usted agregará algunas funcionalidades para interactuar con la aplicación, por ejemplo, selecionando desde un menú el archivo a utilizar como laberinto. La figura 1 muestra un ejemplo de aplicación esperada luego de haber seleccionado un laberinto e incluido un robot en ella. 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 y en la parte baja un botón para pausar o dar continuidad al tiempo.

La barra menú contiene el menú ***File*** y ***World***. Bajo el menú ***File*** el usuario tiene el ítem ***Open***. Con ***Open*** la aplicación pide al usuario seleccionar el laberinto a utilizar. Para ello la aplicación hace uso de la clase JFileChooser. Bajo el menú World se despliegan dos ítems: ***Create robot*** y ***Set delta t***. La opción ***Create Robot*** despliega una nueva ventana o sucesivas ventanas para pedir al usuario definir la velocidad, la distancia de detección de sus sensores, tipo de piloto para el robot y el nombre de archivo donde se registrará la ruta si el usuario así lo quiere. Luego un robot es desplegado junto al mouse y es liberado en el punto donde el usuario presiona el botón izquierdo el mouse. Los robots son transparentes entre sí, luego pueden pasar uno sobre otro. Cuando el usuario decide especificar un archivo para registrar la ruta del robot, lo hace vía la clase JFileChooser. Cuando es definido, el archivo registra una tabla de valores incluyendo tiempo, posición x e y del robot por cada línea del archivo de texto. Sólo se puede insertar nuevos robots cuando el tiempo está en pausa. Cuando un robot sale del laberinto, no es importante por donde se sigue moviendo. El programa termina cunado el usuario cierra la ventan principal de la aplicación.

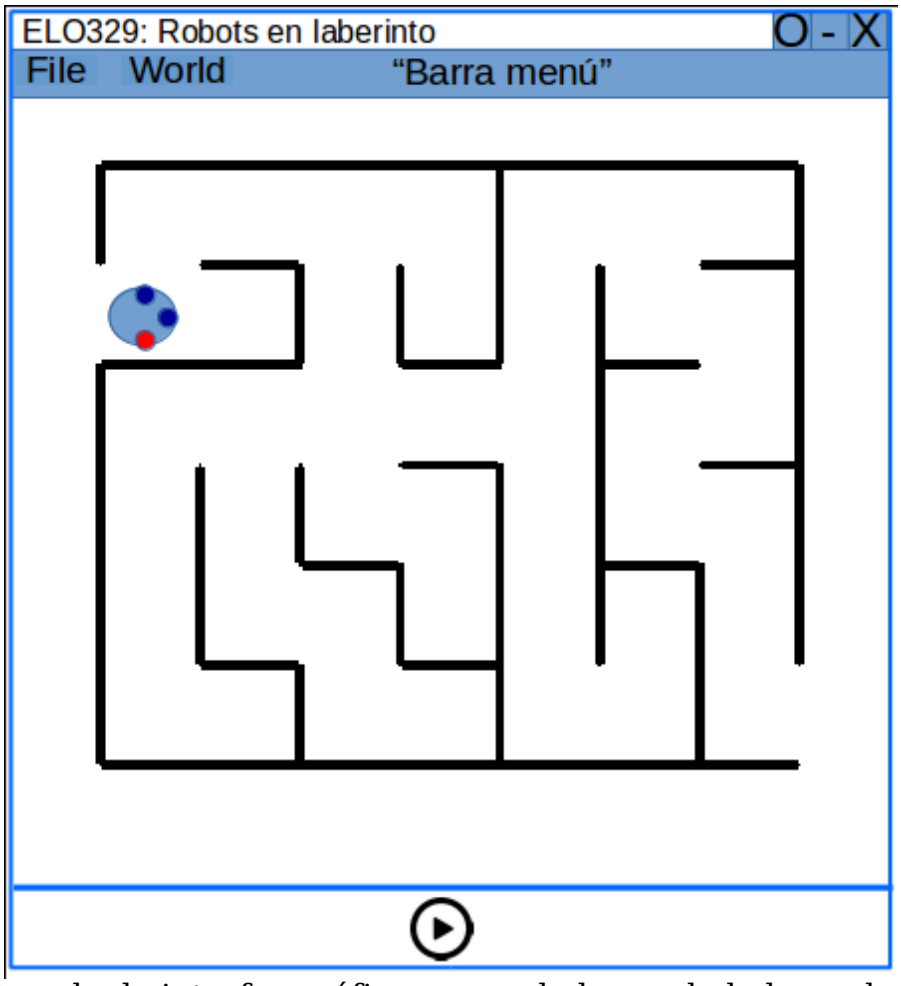

Figura 1: Ejemplo de interfaz gráfica esperada luego de haber seleccionado un laberinto e incorporado un robot

Desarrollo en Etapas

Para llegar al resultado final de esta tarea usted aplicará la metodología "Iterativa e Incremental" para desarrollo de software. Su grupo irá desarrollando etapas que irán abordando los requerimientos gradualmente. En cada etapa usted obtendrá una solución que funciona para un subconjunto de los requerimientos finales. **Su grupo deberá entregar una solución para cada una de las etapas** aún cuando la última integre las primeras. Esto tiene por finalidad, educar en la metodología iterativa e incremental.

Primera Etapa: Ventana principal con menú *File*

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. En la zona inferior debe mostrarse un botón play ~~pausa~~ que al presionarlo cambia a pause ~~play~~. En esta etapa el botón inferior no generará más efecto que cambiar

su vista. Una vez definido y desplegado el laberinto, la venta principal debe redefinir su tamaño para mostrar todo el laberinto en la zona central. La aplicación termina al cerrar la ventana principal. Como ayuda siga los pasos del taller del 18.04.18.

Segunda Etapa: Inclusión de menú World con creación de robot predefinido

En esta segunda etapa se incluyen los dos ítems para el menú ***World***. El ítem ***Set delta t*** define el incremento discreto para el tiempo; sin embargo, el tiempo aún no será activa 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.

Tercera Etapa: Creación de varios robots definidos parametrizados por el usuario

Además de la etapa previa, al crear un robot, el usuario puede ahora definir su velocidad y el tipo de piloto (estrategia apegado a pared derecha o apegado a pared izquierda). No se pide ofrecer opción de registrar la ruta seguida. Luego de definir el piloto, se crea un robot movible junto al mouse, al presionar el botón izquierdo el mouse el robot es liberado. Luego usando el botón ***Play*** y ***Pause*** el usuario observa la ruta seguida por el robot. Al pausar el avance de robot, el menú ***World*** permite la creación de nuevos robots. La vista de cada sensor debe cambiar de color cuando detecta una pared cercana.

Cuarta Etapa: Registro de trayectoria

En esta última etapa el programa permite definir un archivo de salida para cada robot. En éste el robot registra suposición conforme el tiempo avanza. Los momentos de pausa no son reflejados.

Resultados Esperados de su Grupo

Usted deberá documentar, usando notación "JavaDoc" las clases Robot y RobotView de su última etapa.
Prepara un archivo makefile para compilar y ejecutar su tarea en aragorn. Además incluya rótulos "clean" para borrar todos los .class generados, y "doc" para generar la documentación en directorio "documentation".
Entregue todo lo indicado en Normas de Entrega de Tareas. Su documentación automática con javadoc debe ser generable con:
$ make doc
Para que esto funcione usted debe incluir el rótulo doc en su archivo makefile. (OJO no incluya las páginas html generadas por javadoc, éstas serán generadas por este comando cuando el ayudante revise su trabajo)
En su archivo de documentación (pdf o html) incorpore el diagrama de clases de

la aplicación. Éste lo puede generar con jgrasp u otro programa.

Extra créditos

Su grupo puede aspirar a 5 puntos adicionales (la nota igualmente se satura en 100%) si además de registrar la trayectoria, cada robot genera un archivo con el laberinto y su trayectoria en él. En archivo Readme indique si abordó este requerimiento.

Ayudas
* Revise las instrucciones para la realización de tareas.
* 18.04.18: Para ajustar el tamaño de un JFrame, usted puede invocar el método ***pack*** de Window (clase de la cual JFRame hereda). Para componentes gráficas dentro de un JFrame, usted puede consultar por la Window que los contine invocando ***getTopLevelAncestor***().
* 18.04.18: Para cambiar el tamaño de un ImageIcon, considere su método getImage que retorna una instancia de Image, ésta puede ser escalada y luego usted puede invocar setImage de la clase ImageIcon.
* No dude en consultar al profesor o ayudantes sobre dudas de esta tarea.

Sobre la arquitectura Modelo Vista Controlador
Para organizar interfaces gráficas una "solución de software general recomendada" (éstas son conocidas como patrones de diseño) es el "modelo-vista-controlador". El **modelo** es la clase que caracteriza a un objeto y almacena los datos significativos de éste. Por ejemplo, en este caso la clase Robot maneja el modelo de un robot, también las clases Dashboard, y DistanceSensor manejan el modelo para esas categorías de objetos. Por otro lado tenemos las **vistas**, estas clases indican cómo un objeto se muestra visualmente. Estas clases representan visualmente los objetos del problema. En nuestro caso Robot, Dashboard, y DistanceSensor poseen sus vistas en las clases RobotView, DashBoard, y DistanceSensorView. No es el caso de esta tarea, pero un objeto podría tener varias vistas para sus datos; por ejemplo un objeto termómetro tiene un modelo y podría tener varias formas de mostrarse: como columna de mercurio, como número digital, como un color, etc. Finalmente tenemos el **controlador**. Las clases controladoras son aquellas que modifican los datos, por ejemplo a través de las acciones del usuario en la interfaz. Generalmente las clases controladoras son los "listeners" o manejadores de los eventos que usted estima de interés. Una clase puede cumplir dos roles, por ejemplo modelo y controlador. Así como podemos tener varias vistas, es posible tener varias clases controladoras de un modelo. Otras clases controladoras son el listener que responde a las opciones de selección del menú y el listener que responde a los eventos del mouse cuando presionamos sobre un botón.