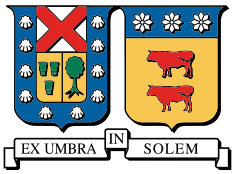

ELO329:POO

Tarea II: Documentación

Profesor: Agustín Gonzalez

Alumnos:
Leonardo Solis 201104505-k
Diego Riquelme 201303041-6
Gabriel Rudloff 201303044-0

30 de abril de 2018

Introducción

Tal como se realizó en la tarea 1, esta tarea consiste en modelar un robot que ingresa en un laberinto y busca la manera de salir de el. Lo principal de ésta tarea será mostrar los movimientos del robot y las estrategias empleadas a través de una interfaz gráfica en donde se seleccionen una serie de parámetros de configuración del robot y también se muestre el laberinto empleado el cuál se debe leer a través de un archivo *PMB*.

La primera Etapa, consiste en desarrollar un Frame en donde se represente el entorno de trabajo para la visualización del laberinto y diferentes herramientas que ayuden a la selección del archivo *PBM* y también a desarrollar la etapa por el robot.

La segunda Etapa, tiene por objetivo poder seleccionar una opción de la barra de Menú con la finalidad de dibujar en pantalla (con el laberinto ya dibujado) una figura que represente de manera inicial lo que será el robot dentro del laberinto.

La siguiente Etapa, busca poder utilizar los diferentes eventos del Mouse para así ubicar al robot dentro del laberinto y poder fijar su posición de manera arbitraria. Aquí se pedirá ver el desarrollo del trayecto del robot dentro del laberinto de forma gráfica con ayuda del botón play y la selección del delta t. Se busca poder incorporar varios robots que trabajen de manera independiente, con un valor de delta t independiente y velocidades independientes.

Posteriormente se definió una cuarta etapa que debía generar un archivo describiendo la ruta de los diferentes robots mediante un archivo que tuviese el dato de las coordenadas y del valor del tiempo, para determinar o estimar si logró llegar al final de la etapa o no.

Además se abordo el requerimiento extra, el que lleva por meta elaborar un dibujo del mapa el cuál entregue la ruta seguida por cada robot.

A continuación se describirán alguna de las de las clases y sus métodos que requieran más explicación que la del código mismo, que fueron programadas para el cumplimiento de las etapas.

ETAPA 4

En esta etapa se logró poder recorrer el laberinto no con uno sino que con una serie de robots que funcionaban de forma independiente, los cuales entregaban un registro de la trayectoria seguida mediante un archivo que contenía dentro la información de los diferentes (t, x, y) que se obtuvieron del recorrido. Este archivo de salida es seleccionado por el usuario luego de fijar que tipo de piloto quiere usar. Además se generó un archivo *.pbm* de salida el cuál almacenará la forma del laberinto y la ruta seguida por cada robot. La siguiente imagen muestra la relación entre las clases diseñadas he instanciadas dentro del programa.

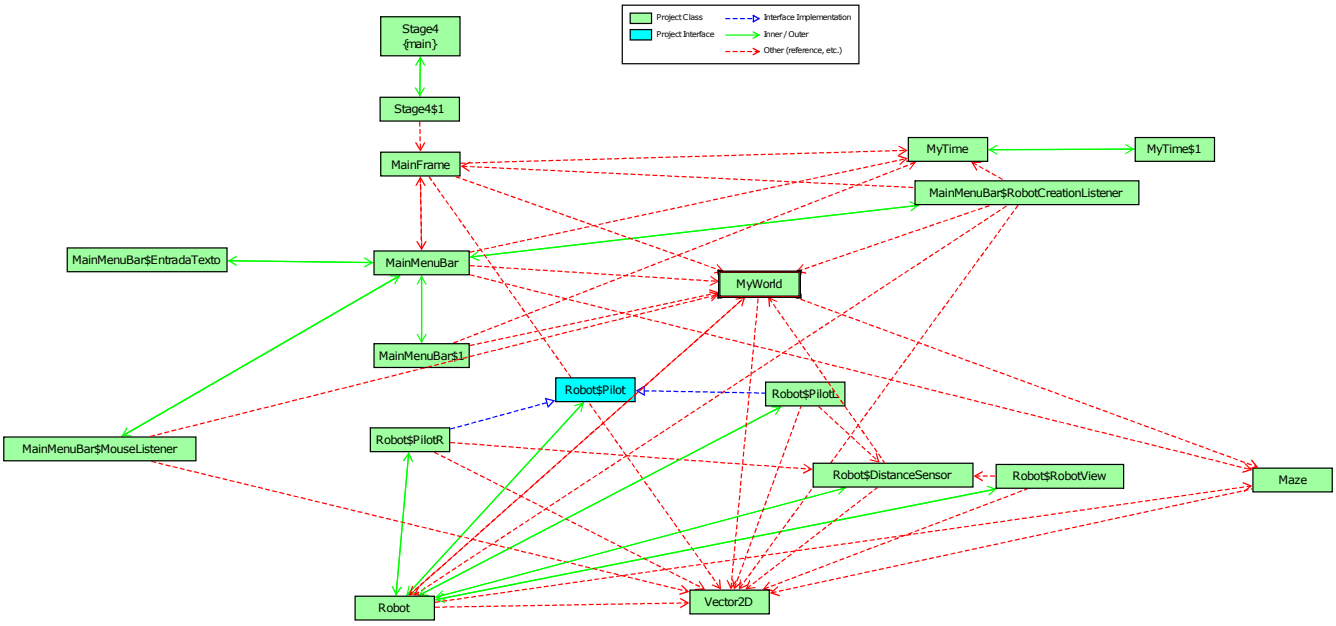

Figura 1: Diagrama de clases

Clases Principales

Por el lado del entorno gráfico se tiene un $MainFrame$ que extiende $JFrame$, este es la ventana principal, a la que se le agrega un $MyWorld$ que extiende a $JPanel$ donde se dibujará el laberinto y los robots. Se le agrega también un $MyTime$ que contiene el funcionamiento del botón de play/pause, además de un $MainMenuBar$ que extiende $JMenuBar$ donde se tendrán las opciones de file y world.
En cuanto al controlador, lo principal es que $JMenuBar$ agrega los ActionListeners para todas las opciones de su menú junto a ActionListeners del mouse además de iniciar el $Timer$. Los ActionListeners de los items del menú permiten setear archivos de entrada y de salida, junto a crear robots y el diferencial de tiempo de la simulación.
El $Timer$ es importante en el funciónamiento de la simulación de los robots. El permite simular un avance dado por un diferencial de tiempo para cada robot (esto demora el orden de $1[ms]$), y luego espera hasta que pase la ventana de tiempo y corre nuevamente. Si se cambia la ventana de tiempo el timer se detiene y ser crea uno nuevo.
Otra clase importante es MyWorld. En esta se pinta el laberinto junto a los robots. Además contiene los robots y todo lo necesario para simular su comportamiento y guardarlo.

Dificultades y observaciones

Para llegar a desarrollar la etapa 4, existieron una serie de dificultades que fueron resueltas en esta etapa o en las etapas previas a ésta. Estas se describen a continuación.

Uso de try-catch

En un principio, para la elaboración de la etapa 1, lo que se debía hacer era pedir desde el directorio de trabajo, el archivo *.pbm* que contendría al laberinto para poder dibujarlo dentro del frame. Existía el problema de que no se reconocía el archivo en el directorio siendo que éste se encontraba guardado ahí. El problema se hacía notar con la excepción **FileNotFoundException**. Para poder dar solución a esto, se recurrió a utilizar el bloque **try**{}, dentro del cuál se procedía a leer el archivo ubicado en el directorio y poder almacenarlo en una variable tipo *Scanner*. En caso de que arrojara una excepción, entonces ésta sería recepcionada en el bloque **catch** {}, que retendría la excepción para así poder seguir compilando el código y seguir trabajando de forma normal.

Mouse Listener

Una de las dificultades encontradas durante la implementación de la etapa 3, fue entender el orden en el cuál se iban creando las clases y en que momento saber instanciarlas. Este problema se dio con la implementación de la interfaz *MouseAdapter*, la cuál fue implementada a través de una clase que se instanciaba dentro de la clase *ActionPerformed*. Luego de realizar esto, fue necesario asociar los eventos del mouse con el laberinto ya dibujado en el frame, lo que a nivel de código quedó representado por *patern.addMouseListener()* para detectar los eventos correspondientes a los clicks y también con el código *patern.addMouseMotionListener()* que ayuda a detectar los eventos relacionados a los movimientos del mouse.

SCALE

Luego de poder hacer que el dibujo del robot se asociara con la ubicación del mouse dentro del dibujo del laberinto, se podía observar que a medida que existía más alejamiento de la posición del mouse con respecto al origen del laberinto, o sea, de la coordenada $(0, 0)$, el dibujo del robot más se alejaba de la ubicación del mouse, llegando a desaparecer de la pantalla cuando ya el mouse se ubicaba en mitad del laberinto (mitad horizontal y/o mitad vertical). En base a esto se determinó que el problema venía a partir de la variable **SCALE** ubicada en *MyWorld.java*, ya que ésta tenía la finalidad de escalar el laberinto en la pantalla, haciendolo más grande o más pequeño dependiendo del factor de escala. Este factor también escalaba las coordenadas correspondientes a *MouseEvent* que llegaba como parámetro a los métodos derivados de la interfaz *MouseAdapter*. Por lo tanto, dichas coordenada fueron divididas por *SCALE* para así poder observar el dibujo del robot sobre la posición del Mouse.

Utilización de flags

Se encontró necesaria la utilización de flags para poder relacionar cada robot que se creaba con el correspondiente archivo de salida que se obtenía al momento que el robot llegaba a la meta. Esto fue necesario porque lo que sucedía era que al terminar la etapa, el programa realizaba el archivo correspondiente a la ruta del robot pero solamente alcanzaba a realizarlo hasta la mitad ya que se volvía a verificar que el robot estaba en la meta y esto hacía que se generara un archivo de salida nuevo, reiniciando el trabajo de elaboración del archivo *.pbm*. El flag por lo tanto servía para indicar que el robot llegaba a la meta, pudiendo así generar solamente una vez cada archivo de salida con los datos de la ruta del robot.

Problema metodo pack()

Como parte de la etapa 1, se pedia que el tamaño de la ventana se ajustára automáticamente al tamaño del laberinto. Justamente el método que logra eso es $pack()$ de $MainFrame$, pero el comportamiento no fue el esperado ya que la ventana diminuía el tamaño hasta solo mostrar el menú y el botón. Esto se logro solucionar usando el método $setPreferedSize()$ para la clase $MyWorld()$ donde se le entrega el tamaño del arreglo que contiene al laberinto.

Falta condición de detención

Lo lógico seria que el usuario también señalara el punto de salida del laberinto, pero como no se señala que esto es necesario, el robot podría recorrer el laberinto por siempre. La solución encontrada fue determinar manualmente un punto de termino exit junto a su respectivo exitRadius que coincide con el punto de salida del laberinto. Así, al momento de llegar a ese punto se imprime el archivo de salida *.pbm* de ese robot asociado.