Trabajo Práctico Final

1 Actividad 1

1.1 Introducción

En robótica, el problema de planeamiento de camino (*path planning*, en inglés) consiste en elegir un camino viable para un robot desde un punto de partida hasta un punto de destino. En general, partimos de un modelo del entorno a recorrer y buscamos generar un camino que optimice alguna característica (distancia recorrida, energía utilizada, cantidad de giros, etc.).

En este trabajo, el modelo del entorno es una cuadrícula de N filas numeradas $0, 1, ..., N-1$ de arriba hacia abajo y M columnas numeradas $0, 1, ..., M-1$ de izquierda a derecha, donde algunas casillas están libres (el robot puede pasar por ellas) y otras casillas están ocupadas por un obstáculo (el robot no puede pasar por ellas). El robot comienza en una posición (i_1, j_1) y debe encontrar un camino que lo lleve a la posición (i_2, j_2). Al estar en una casilla libre (i, j), el robot se puede mover hacia $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ y $(i, j+1)$ (arriba, abajo, izquierda y derecha, respectivamente), si esas casillas están libres.

Normalmente un robot móvil no conoce todo su entorno de antemano, así que tiene que, primero, descubrir el entorno (como un *Roomba* que se va chocando con las paredes hasta que se las aprende). Pero si queremos que el robot vaya de un punto a otro una sola vez, no tiene sentido que se tome su tiempo para aprender todo el entorno.

Entonces la consigna de esta actividad es diseñar e implementar un algoritmo de control para un robot que pueda ir del punto (i_1, j_1) al punto (i_2, j_2) sin conocer el entorno de antemano, sino que lo vaya descubriendo a medida que avanza. Esto implica que el algoritmo tiene que ser capaz de cambiar el plan a medida que descubre mas informacíon sobre el entorno (como mandar una unidad al otro lado del mapa en un juego de estrategia con *fog of war* y que termine recorriendo todo el mapa porque se va encontrando obstáculos inesperados).

1.2 Consigna

Escriba un programa que tome un nombre de archivo como argumento, lea el archivo, verifique que el archivo represente un entorno válido (i.e. que cumpla con el formato descrito en la sección 3) y luego simule las acciones que tomaría el robot en el entorno para ir de la posición (i_1, j_1) a la posición (i_2, j_2). Al alcanzar el final del camino, debe informarlo por consola e imprimir la secuencia de movimientos que se deben realizar, representando los movimientos hacia arriba, abajo, izquierda y derecha con las letras `'U'`, `'D'`, `'L'` y `'R'`, respectivamente.

Como su programa debe simular el movimiento del robot y también el hecho de que el robot no tiene información sobre todo el entorno, solo puede usar la información del entorno para verificar si el robot está a punto de chocarse con una pared. (En particular, implemente una función `posicionValida()` que tome la posición a la que está a punto de moverse el robot y responda si esta es una casilla libre en el entorno).

Su programa debe ser lo más eficiente posible. Esto incluye tanto el uso de estructuras de datos adecuadas como, por ejemplo, que evite recorrer todo el entorno si no hace falta para encontrar un camino. Por otro lado, secundariamente, se considerará mejor que el programa encuentre un camino lo más corto posible, pero no es necesario que encuentre el camino más corto. Por ejemplo, con el siguiente archivo de entrada:

Listing 1: laberinto.txt

```
5 6 1
0 0
4 5
.....#
.##.##
..#..#
...#.#
.#.#..
```

su programa podría imprimir una línea con la cadena `"DDRDLUUURRRDDRLRDDR"`, pero consideramos que es mejor que imprima la cadena `"RRRDDRDDR"` (sin comillas).

Se tomará en cuenta en la corrección que su programa no "haga trampa". O sea, que solo lea la información del entorno de casillas a las que está a punto de moverse el robot, que el robot no se mueva a posiciones que – hasta donde sabe el robot – podrían contener un obstáculo y que no mienta al imprimir la secuencia de movimientos realizada por el robot.

Sólo se evaluará el funcionamiento del programa en entornos donde existe al menos un camino viable desde (i_1, j_1) hasta (i_2, j_2).

2 Actividad 2

2.1 Introducción

En esta actividad introducimos una nueva forma de recolección de información para el robot. Esta vez el robot estará equipado con un *scanner* tetradireccional (es decir que tiene cuatro direcciones) que tira rayos (uno en cada dirección cardinal) y calcula la distancia a la pared más cercana en la dirección de cada rayo. Además, los rayos del *scanner* tienen una distancia límite. Si no hay ninguna pared a esa distancia o menos, entonces no detecta nada en esa dirección. Por ejemplo, si la distancia límite es 5 y hay una pared a distancia 7, no la detecta. Más aún, usar el *scanner* se considera indeseable porque usa mucha batería y esto limita la autonomía del robot, así que también hay que minimizar la cantidad de veces que se usa el *scanner*.

Para hacerlo más realista, el scanner no será parte de su programa, sino que correrá dentro de otro programa con el cual deberá comunicarse mediante un protocolo de comunicación detallado más abajo.

2.2 Consigna

Escriba un programa (de forma separada al programa realizado en la actividad 1) que simule las acciones que tomaría el robot en el entorno para ir de la posición (i_1, j_1) a la posición (i_2, j_2). Cada vez que su programa quiera usar el sensor, debe informar su posición actual por consola y tomar la respuesta del sensor por teclado. Al alcanzar el final del camino, debe informarlo por consola e imprimir el camino tomado en el mismo formato que en la actividad 1.

En esta actividad se evaluará que use el *scanner* la menor cantidad de veces posible y que el algoritmo sea lo más eficiente que se pueda.

Toda entrada y salida que realice su programa mediante `stdin` y `stdout` debe seguir los formatos explicados en la sección 2.4. Para *debuggear* su código recomendamos usar un *debugger*, pero tambien puede imprimir mensajes por la consola usando `fprintf(stderr, ...)`. Se tomará en cuenta en la corrección que su programa no "haga trampa". O sea, que no lea el archivo que contiene el entorno, que no use el sensor en una posición que no es la posición actual del robot, que no se mueva a posiciones que – hasta donde sabe el robot – podrían contener un obstáculo y que no mienta al imprimir la secuencia de movimientos realizada por el robot.

Sólo se evaluará el funcionamiento del programa en entornos donde existe al menos un camino viable desde (i_1, j_1) hasta (i_2, j_2).

2.3 Programa sensor

Desde la cátedra preparamos un programa que vamos a usar para testear todo esto usando *named pipes*. Esto nos permite leer la salida que envía su programa por `stdout` (el lugar donde va el texto que imprime `printf` o alguna función similar) y recibirla por `stdin` (el lugar de donde lee datos `scanf` y compañía), al mismo tiempo que nuestra salida se envía a la entrada de su programa. De esta forma cada programa lee lo que el otro imprime.

La dinámica termina siendo la siguiente: su programa envía un comando usando `printf` y se pone a esperar la respuesta usando `scanf`. Mientras tanto, el programa sensor recibe el comando usando `scanf` y luego imprime su respuesta (usando `printf`), lo cual permite que su programa siga ejecutándose libremente.

Si bien es super interesante, no hace falta entender cómo funciona en profundidad. Sólo se debe abrir la terminal, navegar hasta el directorio donde se encuentran los archivos `correr.sh`, `sensor.c` y el ejecutable de su programa (el cual debe llamarse `robot`) y correr el comando `./correr.sh laberinto.txt`.

El script `correr.sh` está pensado para correr en Linux. Si no funciona, probablemente no tenga permisos de ejecución. Estos se pueden otorgar con el comando `chmod +x correr.sh`.

2.4 Interacción

2.4.1 Inicio

Al iniciar, el programa sensor envía el alto N y ancho M del entorno, la distancia máxima del sensor D, la posición inicial (i_1, j_1) y la posición final (i_2, j_2) del robot en este formato:

```
N M D
i1 j1
i2 j2
```

Cada número esta separado por un espacio, no hay espacio al final del renglón y cada renglón termina con exactamente un salto de línea. Se puede leer así:

```c
int N, M, D;
scanf("%d%d%d", &N, &M, &D);
int i1, j1;
scanf("%d%d", &i1, &j1);
int i2, j2;
scanf("%d%d", &i2, &j2);
```

2.4.2 Comando

Para enviar un comando, su programa debe imprimir dos numeros con la posición actual (i, j) del robot en este formato:

```
? i j
```

Un caracter '?', un espacio, dos números separados por un espacio, sin espacios al final y debe haber exactamente un salto de línea al final. Para garantizar que el programa sensor reciba el comando, debe vaciar el *buffer* de salida después de imprimir, como se muestra a continuación:

```c
printf("? %d %d\n", i, j);
fflush(stdout);
```

Ante esto, el sensor responde con 4 numeros (d_1, d_2, d_3, d_4) que se corresponden con la distancia hacia arriba, abajo, izquierda y derecha, respectivamente, en el siguiente formato:

```
d1 d2 d3 d4
```

Cuatro números separados por exactamente un espacio, sin espacios al final, terminado por exactamente un salto de línea. Si una de las distancias d_i queda fuera del rango del sensor, entoces la respuesta tendrá $d_i = D+1$. Si se hace una consulta inválida (por ejemplo, la posición dada está fuera del tablero o dentro de una pared) la respuesta tendrá $d_1 = d_2 = d_3 = d_4 = 0$.

2.4.3 Finalización

Al terminar la búsqueda, su programa debe avisar al sensor así este también termina su ejecución. Para hacer esto se envía un comando en este formato:

```
! camino
```

Un caracter '`!`' seguido de un espacio, una cadena compuesta de caracteres '`U`', '`D`', '`L`' y '`R`' que representa los movimientos realizados por el robot y un salto de línea. Se debe hacer un *flush* después.

3 Formato de archivo

La primera línea contiene tres números naturales N, M y D separados por espacios: la cantidad de filas, de columnas y el rango del sensor. La siguiente línea contiene dos numeros i_1 y j_1, la posición inicial del robot. La siguiente línea tiene dos numeros i_2 y j_2, la posición final objetivo del robot. Cada una de las siguientes N líneas contiene una cadena de longitud M formada por los caracteres '`.`' y '`#`' que representa una fila del laberinto. El caracter '`.`' representa una casilla libre y '`#`' representa una casilla ocupada por un obstáculo. Cada liínea termina con exactamente un caracter de salto de línea '`\n`'

Por ejemplo, debajo se puede ver un entorno de cinco por seis con un radar de rango uno, donde el objetivo es ir de la esquina superior izquierda (posición $(0, 0)$) a la esquina inferior derecha (posición $(4, 5)$).

Listing 2: laberinto.txt

```
5 6 1
0 0
4 5
.....#
.##.##
..#..#
...#.#
.#.#..
```