

## **Proyecto Fase 1 - Robot sale de un laberinto**

### *Investigación previa*

#### **Algoritmos existentes para resolver la situación que se presenta**

- *Backtrackig* - Es un algoritmo que consiste en buscar las salidas al laberinto por las laterales y retornar si se encuentra con un punto sin salida, cada vez que se encuentre con un punto sin salida va a volver a la última posición en donde se hayan encontrado dos o más caminos posibles.
- *Algoritmo de Tremaux* - Es un algoritmo eficiente en el sentido que permite encontrar la salida de cualquier laberinto independiente de su forma. Al llegar a un cruce no importa hacia dónde seguir, siempre y cuando no se haya pasado antes por ese punto anteriormente, si se llega a un camino sin salida, se vuelve al cruce anterior, si en dicho cruce ya se hicieron todos los posibles caminos, se retrocede al anterior y así sucesivamente.
- *Mano derecha* - Llegar al punto de encuentro siempre por el lado derecho tomando como referencia la pared, la velocidad para salir del laberinto con este algoritmo depende en mayor parte de la posición de partida. Este algoritmo falla en algunos modelos, como laberintos que consiste de círculos concéntricos.

#### **Razones por las que se implementará el algoritmo seleccionado**

Existen dos versiones del algoritmo, la recursiva y no recursiva. La versión no recursiva permite la implementación de un *stack* haciendo *push* de los nodos, o posiciones que va guardando el robot en donde encontró más de dos salidas posibles. Cuando el robot prueba una de estas posibilidades se agregan como nodos hijos y si resultan ser un camino sin salida se hacen *pop* volviendo nuevamente al último nodo padre. La versión recursiva permite también modelar la resolución del algoritmo con una estructura de árbol permitiendo programar un algoritmo más eficiente pero a un mayor costo de memoria.

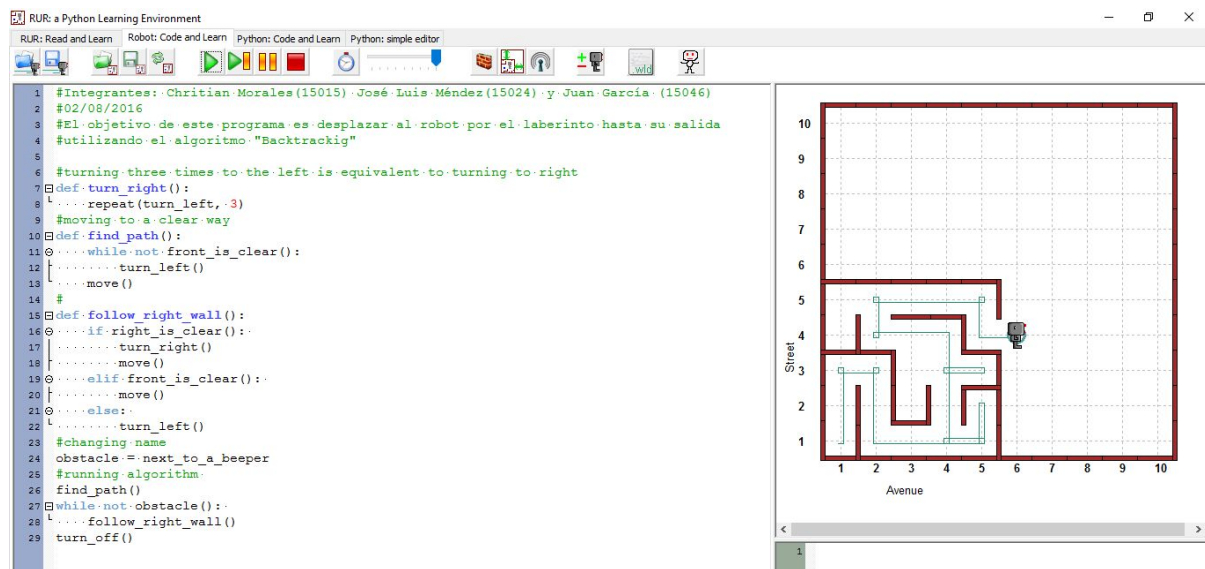
En general el *backtracking* es una forma beneficiosa de resolver el problema porque puede ofrecer soluciones al laberinto independiente de su forma y permite más interactividad con los sensores del robot a programar, a diferencia de algoritmos de fuerza bruta o el de mano derecha. Además, el hecho de contar con dos versiones permite una mejor interpretación al momento de transferirlo a cualquier lenguaje de programación, porque un algoritmo recursivo en alto nivel es en esencia un algoritmo no recursivo implementando un *stack* en lenguaje de máquina o lenguaje ensamblador.

## Algoritmo en pseudocódigo, versión no recursiva

Resolver (nodo n)

```
    Si n es un nodo padre
        Si es el nodo de meta
            print n
            return true
        else return false
    else
        para cada hijo c de n
            si resolver(c) funciona
                print n
                return true
        return false
```

## Algoritmo implementado con RUR-PLE



The screenshot displays the RUR-PLE Python Learning Environment interface. On the left, a code editor shows a Python script for a maze-solving algorithm. The script includes comments in Spanish and functions for turning, finding a path, and following a right wall. On the right, a 10x10 grid represents the maze. The grid has a red border and a light blue background. A small robot icon is positioned at the entrance of the maze (row 4, column 6). The maze is defined by black lines forming a complex path. The axes are labeled 'Street' (vertical) and 'Avenue' (horizontal).

```
1 #Integrantes: Chritian Morales (15015) , José Luis Méndez (15024) y Juan Garcia (15046)
2 #02/08/2016
3 #El objetivo de este programa es desplazar al robot por el laberinto hasta su salida
4 #utilizando el algoritmo "Backtrackig"
5
6 #turning three times to the left is equivalent to turning to right
7 def turn_right():
8     repeat(turn_left, 3)
9     #moving to a clear way
10 def find_path():
11     while not front_is_clear():
12         turn_left()
13         move()
14     #
15 def follow_right_wall():
16     if right_is_clear():
17         turn_right()
18         move()
19     elif front_is_clear():
20         move()
21     else:
22         turn_left()
23     #changing name
24     obstacle = next_to_a_beeper
25     #running algorithm
26     find_path()
27     while not obstacle():
28         follow_right_wall()
29     turn_off()
```