Algoritmos y Estructura de Datos
Sección: 10
Catedrático: Douglas Barrios
Fecha: 16/02/18

Proyecto Parallax - Fase 1

Universidad del Valle de Guatemala
Oscar Juárez - 17315
Andrea Arguello - 17801
Mario Sarmientos - 17055

Proyecto Parallax. Primera Fase.

Algoritmos Existentes:

1. Wall follower: Una versión modificada de la regla de la mano derecha, ya que este es capaz de resolver cualquier tipo de laberinto sin quedarse estancado; cuenta los giros que se realizan para saber si se encuentra en un circuito cerrado o una isla y así poder salir de está. Puede encontrar la solución de un laberinto desde su perímetro hasta el centro, pero no del centro al perímetro. Una ventaja de este algoritmo es que no necesita memorizar los lugares por donde ha pasado, pero aún así debe llevar la cuenta de sus giros. (Sio-Iong Ao, 2008)

2. Mano derecha: Este algoritmo busca dividir el laberinto completo en pequeños laberintos que están encadenados y este algoritmo los resuelve todos de poco en poco. Se basa en seguir la estructura del algoritmo seguidor de paredes, pero de una manera más eficiente ya que memoriza los lugares por los que ha pasado antes. Este algoritmo no encuentra el camino más corto, pero busca caminos alternativos basado en los lugares que ya a pasado. (Carlos Martin-Vide, 2010)

3. Blind Alley Sealer: Encuentra todas las soluciones posibles al eliminar los callejones sin salida del laberinto. Sin embargo, esto solo llena y bloquea el paso de cada callejón sin salida y evita pasar por ese mismo lugar. Como resultado, esto creará secciones de paso inaccesibles para callejones sin salida. Este algoritmo se centra en el Laberinto, funciona mucho más rápido que el relleno de callejón sin salida, aunque requiere memoria extra. Asigna cada sección de paredes conectada a un conjunto único. Para hacer esto, cada sección de muro que aún no se encuentre en un conjunto, se registra en el sistema y asigna todas las paredes alcanzables a un nuevo conjunto. Después de que todas las paredes estén en conjuntos, si estas se encuentran en una sección de paso, entonces se sella ese pasaje. Tal pasaje debe ser un callejón sin salida, ya que las paredes a cada lado se une entre sí y forma un corral.(Weisstein, 2002)

4. Blind Alley Filler: Este método encuentra todas las soluciones posibles, independientemente de cuán largas o cortas sean. Lo hace llenando todos los callejones sin salida, es decir, todos aquellos lugares en los que tengas que retroceder para salir. Este algoritmo se enfoca en el Laberinto, no usa memoria extra, pero es bastante lento. Para cada unión, vea si el robot enviado por un camino regresa de ese mismo camino (en lugar de seguir a una dirección diferente o de salir del Laberinto). Si lo hace, entonces ese pasaje y todo lo que está por delante no puede estar en ninguna ruta de solución, así que cierra ese pasaje y completa todo lo que está detrás de él. (De-Shuang Huang, 2013)

5. Shortest path finder: Este algoritmo utiliza los conceptos de teoría de grafos para poder encontrar la ruta de salida más rápida, este también tiene una variantes en las que el algoritmo analiza los caminos de acorde a la distancia que hay de ellos al punto de partida, al llegar al final del laberinto basado en las distancias previamente mencionadas encuentra la ruta más cortas. (Sartaj Sahni, 1995)

6. Algoritmo de Trémaux: Este algoritmo hace que el objeto encuentre la salida marcando los puntos por los que ha pasado, de este modo descarta los caminos sin salida, a modo de enfocarse en la ruta de salida. Este algoritmo es super efectivo para todo tipo de laberintos siempre que sus pasadizos estén bien definidos. (L.Christine Kinsey, 2006)

Explicación del algoritmo a utilizar:

El wall follower requiere de sensores de proximidad en el Parallax, uno al frente y uno al lado, o bien un sensor infrarrojo de ángulo, para detectar dónde se hallan las paredes. El objetivo de este es seguir las paredes del laberinto, lo cual eventualmente llevará a que este salga del mismo en el primer intento. No requiere ninguna memoria del laberinto.

El algoritmo se basa en tener un ciclo while, en donde este se mantendrá verdadero siempre y cuando el parallax se encuentre rodeado de al menos un par de paredes. Este primero verifica si su lado derecho está libre, de ser así, cruza hacia este lado. De no ser así, verifica si el frente está libre; si está libre entonces avanza hacia delante. Finalmente, si ambos derecho e izquierdo están bloqueados, gira a la izquierda y el ciclo se repite.

Nuestro algoritmo presenta una ligera variación, ya que de haber islas, el algoritmo usual no funciona y el robot se estancará y dará vueltas a la misma de manera continua. La variación implementada consta en que, al contar que ya se han hecho más de 6 giros a la derecha, este reconozca que está dando vueltas alrededor del mismo lugar, y en efecto siga de largo hasta topar con una pared para salir de la respectiva isla

A pesar de no ser el algoritmo más eficiente en cuanto a tiempo, este tiene por asegurado que el parallax salga del laberinto. La complejidad de este algoritmo está dada por:

$$N * Log(N)$$

Debido a que posee un ciclo while y puede llegar a realizar otro ciclo while dentro de este. Esto nos indica que el tiempo en el que el parallax resuelve el laberinto, crecerá conforme el tamaño del mismo, pero que será más eficiente que, digamos, uno de N^2.

Las razones principales por las que se escogió este algoritmo fue el bajo nivel de dificultad que requiere la formulación del algoritmo. Además, otros métodos toman más tiempo o, para poder realizarlo de la manera más corta posible, requieren de conocer ya el laberinto o poder verlo desde un punto de vista sobre el cual se puedan observar todos los caminos, o de recorrerlo más de una vez para poder hallar la salida.

Pseudocódigo o diagrama de flujo:

Diagrama para el programa principal:

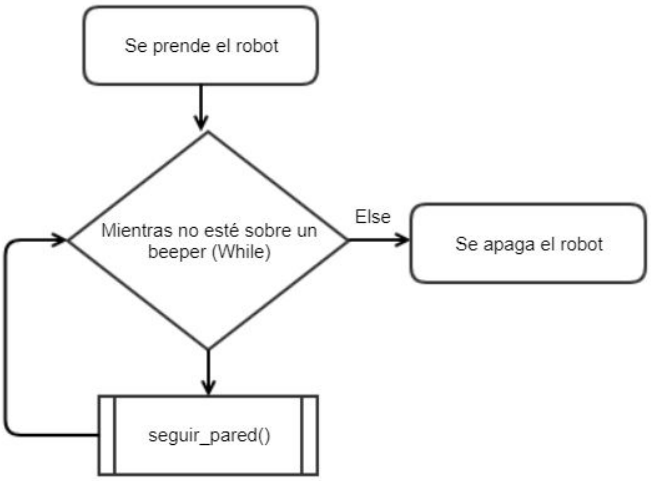


Diagrama para el método seguir_pared()

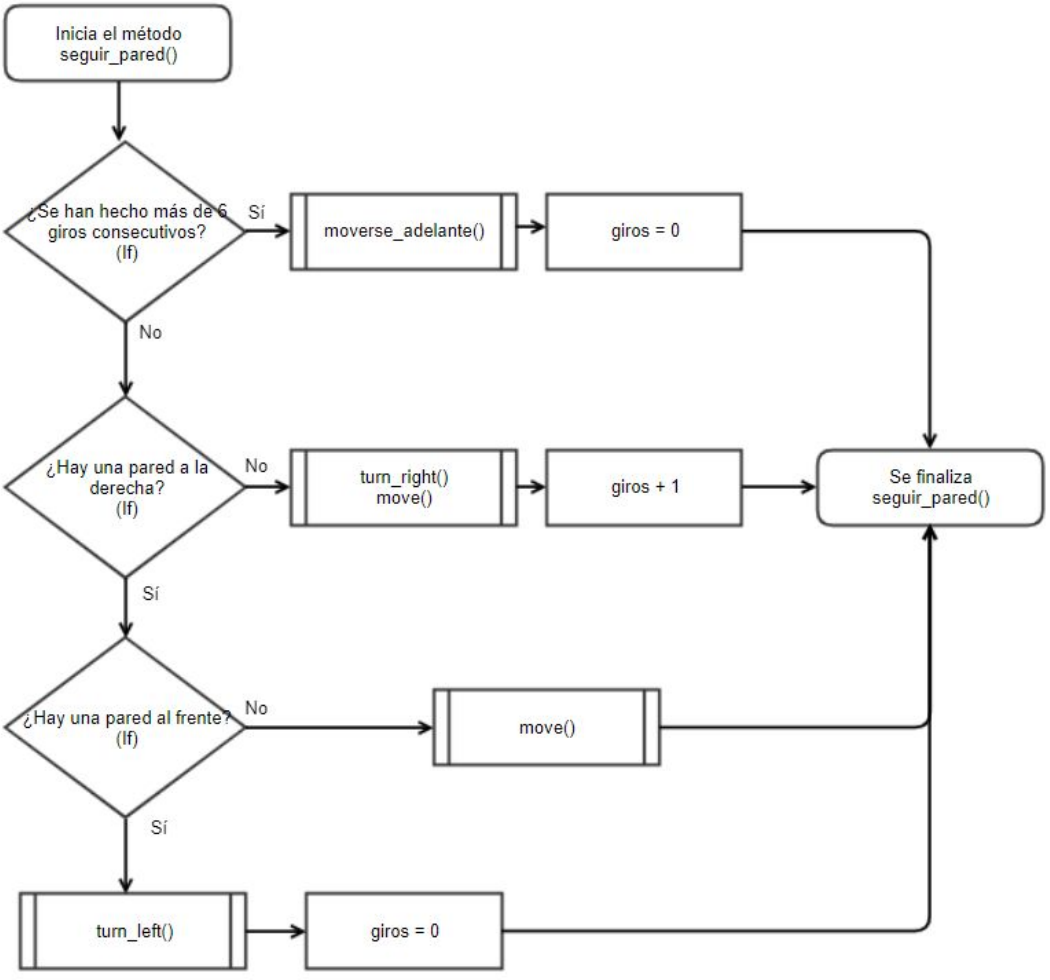
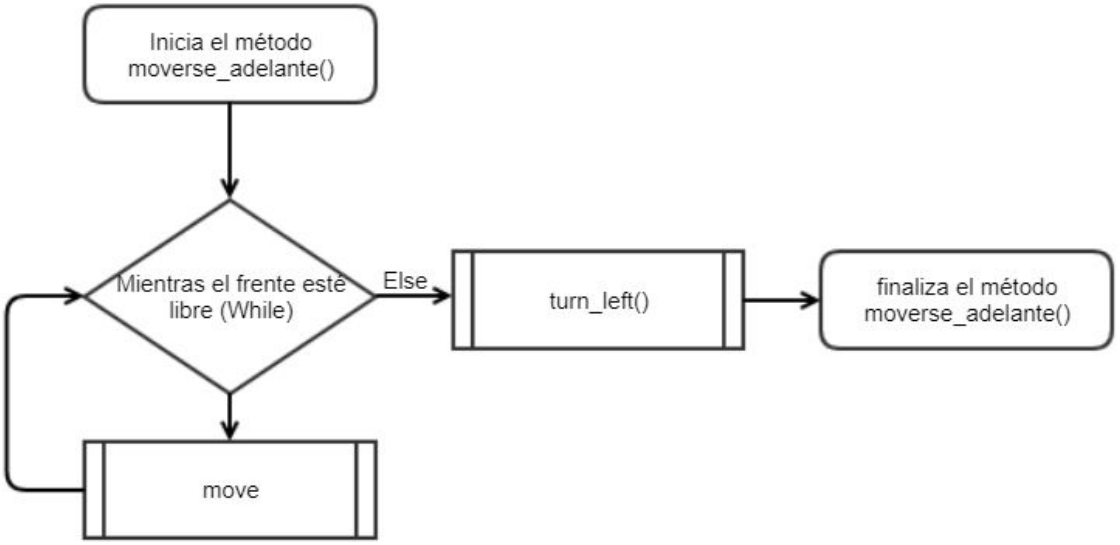


Diagrama para el método moverse_adelante()



Algoritmo empleado en Rur-ple para la resolución de laberintos:

```
giros = 0;

def turn_right():
    repeat(turn_left, 3)

def seguir_pared():
    global giros

    if giros == 7:
        while front_is_clear():
            move()
        else:
            turn_left()
        giros = 0;
    elif right_is_clear():
        turn_right()
        move()
        giros = giros+1
    elif front_is_clear():
        move()
    else:
        turn_left()
        giros = 0

while not on_beeper():
    seguir_pared()

turn_off()
```

Tabla 1. Resultado en cuanto al tiempo conforme a distintos laberintos.

Laberinto	Tiempo en salir (segundos)	Dificultad
1	1.66	baja
2	8.21	Alta
3	7.29	Media
4	16.18	Alta
5	13.23	Alta

Figura 1. Laberinto 1, complejidad baja.

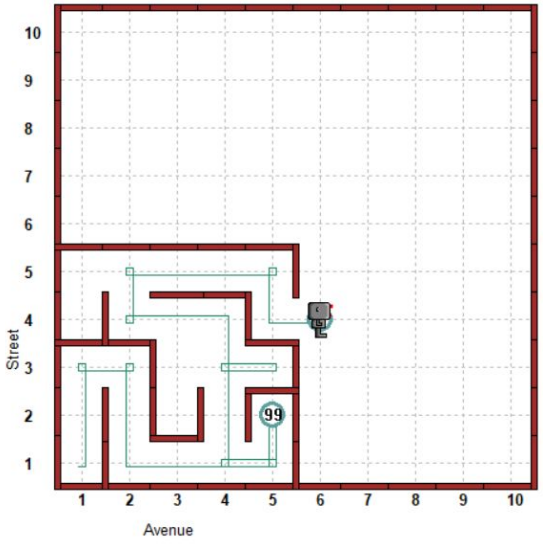


Figura 2. Laberinto 2, complejidad media.

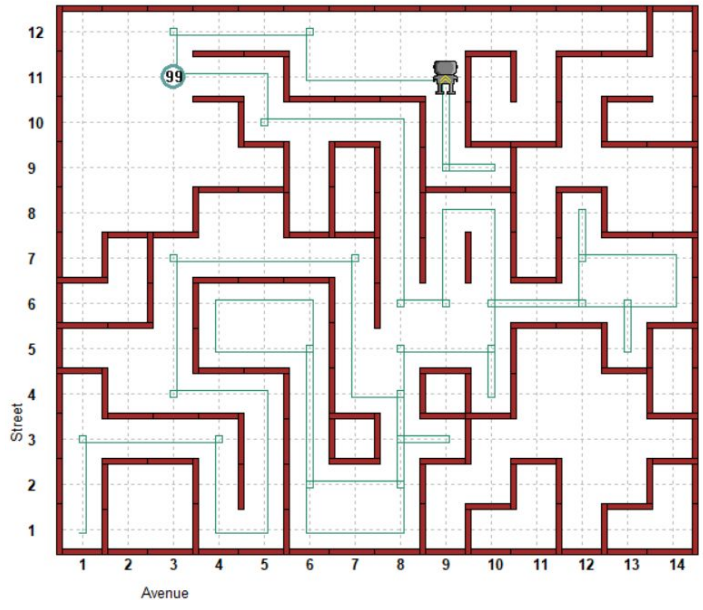


Figura 3. Laberinto 3, Complejidad media.

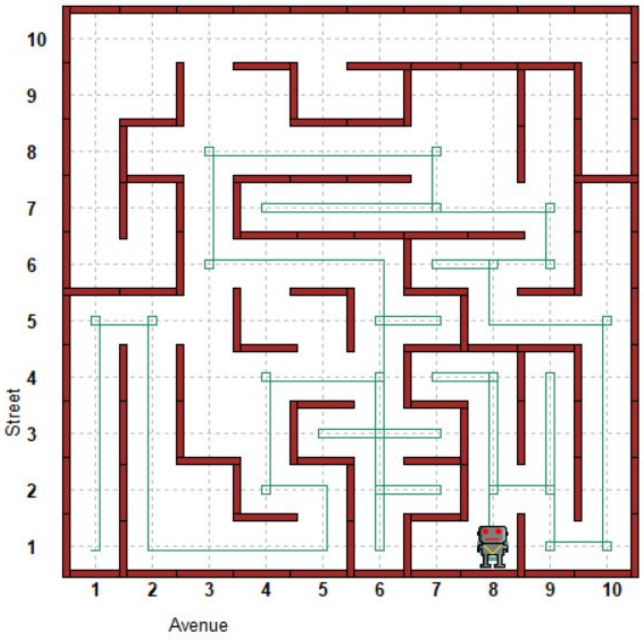


Figura 4. Laberinto 4, Complejidad alta

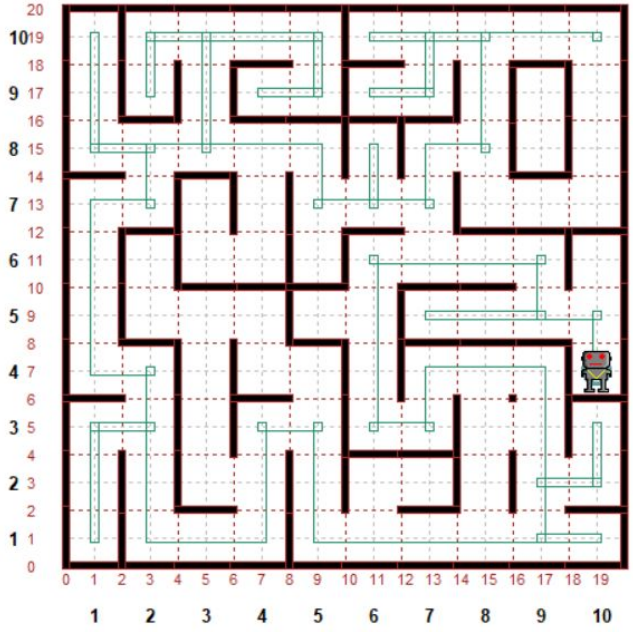


Figura 5. Laberinto 5, complejidad alta.

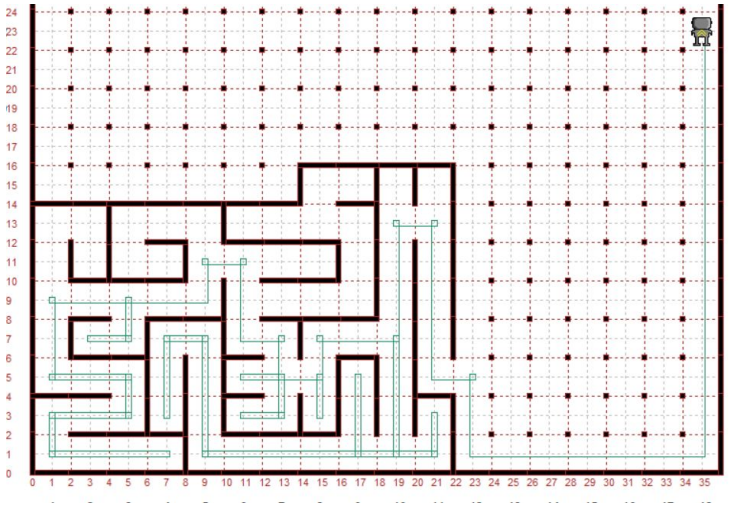
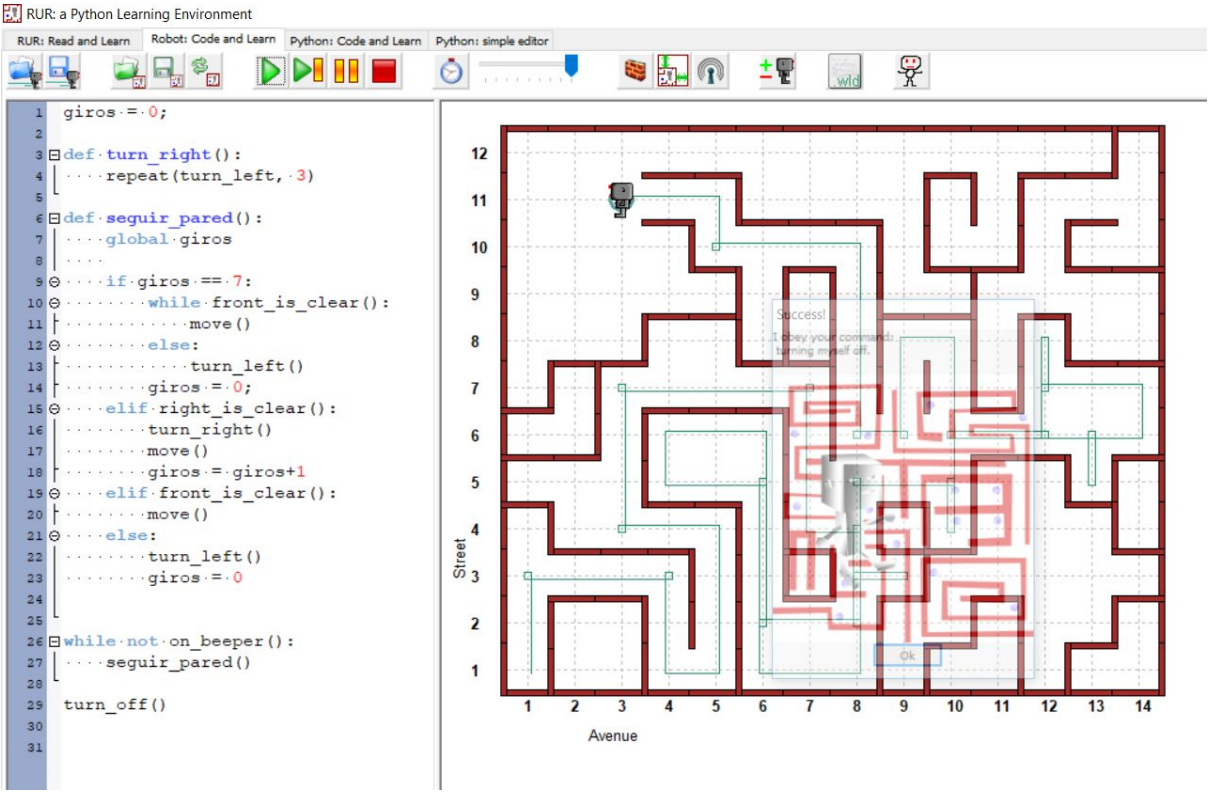


Figura 6. Algoritmo ejecutado en Rur-ple.



Repositorio en Github:

https://github.com/OJP98/ProyectoParallax

Bibliografía

Carlos Martin-Vide, H. F. (2010). *Language and Automata Theory and Applications: 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010, Proceedings.* Munich: Springer.

De-Shuang Huang, K.-H. J.-Q. (2013). *Intelligent Computing Theories and Technology: 9th International Conference, ICIC 2013, Nanning, China, July 28-31, 2013. Proceedings.* Munich: Springer.

L.Christine Kinsey, T. E. (2006). *Symmetry, Shape and Space: An Introduction to Mathematics Through Geometry.* Berlin: Springer Science & Business Media,.

S.G. Poonambalam, J. P. (2012). *Trends in Intelligent Robotics, Automation, and Manufacturing: First International Conference, IRAM 2012, Kuala Lumpur, Malaysia, .* munich: Springer.

Sio-Iong Ao, B. B.-S. (2008). *Advances in Computational Algorithms and Data Analysis.* berlín: Springer Science & Business Media.

Weisstein, E. W. (2002). *CRC Concise Encyclopedia of Mathematics, Second Edition.* Florida: CRC Press.

Sartaj Sahni, R. F. (1995). *Software Development in C.* Milan: Silicon Press.