Universidad del Valle de Guatemala
Algoritmos y Estructura de Datos
Luis Rodrigo Urrutia Castellanos #16139
Kevin Sebastián Macario #17369
Antonio Reyes #17273

Fecha: 15 de febrero del 2018
Sección: 20
Instructor: Diego Enriquez

Proyecto 1 - Fase 1

Para el Primer proyecto de la case de Algoritmos y Estructura de Datos, se diseñará un algoritmo que permita al robot Parallax salir de cualquier laberinto. Lo que se necesita para poder realizar este proyecto serán el robot Parallax y un sensor de movimiento que serán implementados en el algoritmo. Con el fin de que el robot Parallax salga de forma más rápida posible, se investigaron distintos algoritmos que podrían utilizarse con el fin de encontrar el más rápido y eficiente.

Link de GitHub: https://github.com/RodrigoUrrutiaC/Proyecto1-Fase1

1. Algoritmo de Trémaux

Este algoritmo, creado por el ingeniero Charles Pierre Trémaux garantiza completar cualquier laberinto. El método consiste en dejar “marcas” en lugares donde el robot ya ha pasado y el seguir unas reglas fundamentales que se explicarán a más detalle a continuación.

Regla 1

Cuando el robot pase por un pasillo, debe agregar una marca desde el inicio del pasillo hasta el final del pasillo o aparezca una opción para tomar distintas direcciones, esto se realiza para que, en el caso de que el robot vaya pasar por el pasillo de nuevo, retroceda porque muestra que el pasillo ya fue recorrido. Figura 1 muestra un ejemplo hecho a computadora de un marcador que delimita los caminos ya tomados.

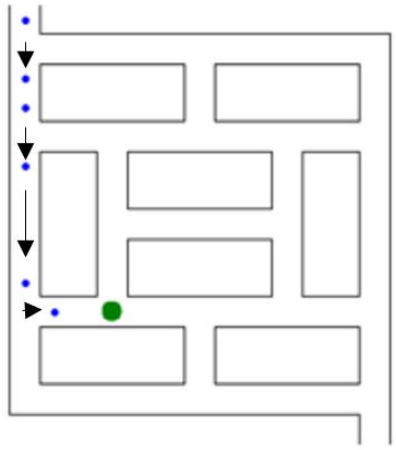


Figura 1

Regla 2

No entrar en un camino previamente visitado. Para evitar este problema, se utilizan los puntos o marcas que se mencionaron en la regla anterior.

Regla 3

Al llegar a un cruce con múltiples caminos, no importa que camino el robot desea elegir.

Regla 4

Si el robot se topa con una marca que puso previamente, agregar una segunda marca y regresar, en el caso de encontrar varios caminos donde algunos tienen marcas y otros no, elegir el camino con menos marcas.
La Figura 2 muestra un ejemplo de un ejemplo de un cruce de un laberinto utilizando el Algoritmo de Trémaux. Las X en esta imagen simboliza una segunda marca

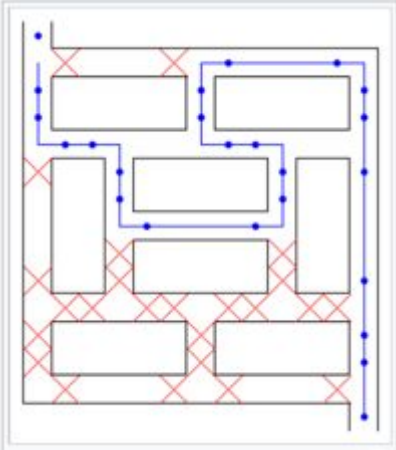


Figura 2

Ventajas del Algoritmo de Trémaux

- El algoritmo de Trémaux es muy eficiente ya que con los métodos de agregar uno o dos marcadores, el robot es capaz de eliminar pasillos completos. Esta eliminación de pasillos aumenta la eficiencia del robot al buscar la salida del laberinto.

Desventajas del Algoritmo de Trémaux

- Aunque las reglas del Algoritmo de Trémaux son bien explicadas, existen pasos que podrían desviar al robot y poner el riesgo el objetivo del proyecto. Esta sería la regla 3 (no importa qué camino desea elegir), ya que Parallax podría estar justo en frente de la salida tomar otras dirección si dicha salida se encuentra en un pasillo con múltiples direcciones.

- El método de agregar marcadores es una complicación ya que se tiene que encontrar la forma de que Parallax pueda diferenciar entre un marcador y una pared y qué método utilizar para agregar un segundo marcador.

2. Algoritmo de Garantía

Algoritmo simple que recorre todo el laberinto paralelo a una pared. Si llega a un intersección gira siempre en la misma dirección durante todo el laberinto. Es capaz de saltar islas en los laberintos. Como su nombre lo indica, da la garantía de la resolución desde cualquier comienzo desde el perímetro hasta el objetivo en el centro. No necesita marcar o mapear el recorrido de los lugares en el que pasa, como otros algoritmos, únicamente cuenta los giros que realiza a la izquierda o derecha, esto lo hace con el fin de identificar que se encuentra en una isla y así se le haga posible salir de la misma. (Salvatierra, 2017)

Ventajas del Algoritmo de Garantía:

- Garantiza encontrar la solución del laberinto.
- No recorre las mismas rutas en el laberinto ya que marca en donde ha pasado anteriormente.
- Si la solución del laberinto se encuentra en una isla no le es posible hallarla.
- Si el robot se encuentra al principio del laberinto en una isla no es capaz de salir de ella.

Desventajas del Algoritmo de Garantía

- No es capaz de resolver un laberinto al revés.
- No encuentra el camino más corto.

3. Algoritmo de Cadena

Algoritmo con base del algoritmo de garantía y de seguidor de paredes. Su forma de resolver los laberintos se basa en subdividir el laberintos en pequeños algoritmos encadenados, seguido de esto los resuelve en secuencia. Analiza si ya paso por una vía del laberinto tratando de acortar lo más posible comparándolo con caminos alternativos. (Salvatierra, 2017)

Ventajas del Algoritmo de Cadena

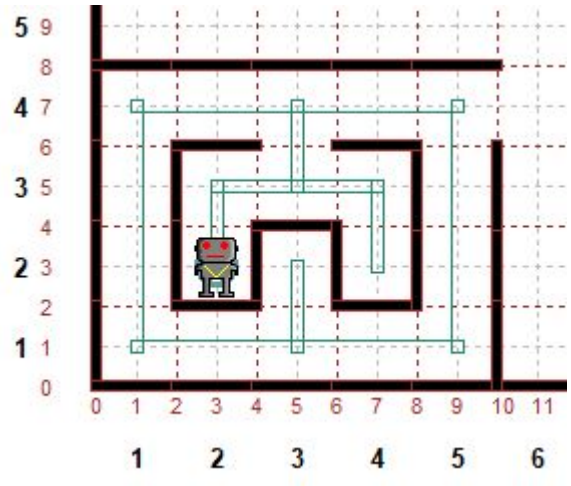
- La probabilidad de encontrar la solución del laberinto es más alta.
- No recorre las mismas rutas del laberinto ya que mapea las ubicaciones en donde ya ha pasado.

Desventajas del Algoritmo de Cadena

- No es capaz de encontrar la solución del laberinto si este se encuentra en una isla.
- No es capaz de salir de una isla en el laberinto

4. Algoritmo *Wall follower*

El *Wall follower*, también conocido como la regla de la mano izquierda o la regla de la mano derecha, es un algoritmo recursivo diseñado para salir de un laberinto. Si las paredes de este laberinto están simplemente conectadas, es decir, todas sus paredes están conectadas entre sí o al límite exterior del laberinto, siempre se logrará encontrar una salida si esta existe. Sin embargo, si el algoritmo comienza dentro de un *loop*, una "isla", como la mostrada en la siguiente imagen, no será posible encontrar una solución (ArcBotics, 2016).



Diseñado en RUR, el robot recorre todo el laberinto sin encontrar la salida.

Para este proyecto, se utilizará la versión del algoritmo que sigue la pared derecha del laberinto.

Ventajas del Algoritmo *Wall follower*

- No presenta dificultad en la implementación.
- Es posible manipular el código de una manera sencilla para hacer que la eficiencia del algoritmo incremente.

Desventajas del Algoritmo de *Wall follower*

- No resuelve todos los posibles laberintos en su versión más simple.

Algoritmo a implementar: *Wall follower*

Este algoritmo se implementará debido a la baja complejidad que tiene y a que, dado un laberinto, es posible encontrar una salida si este no cuenta con *loops*, lo cual se discutirá más adelante. Además, se busca que la solución sea rápida, y directa, sin tener que perder tiempo explorando partes del laberinto por dejar al azar las decisiones de a dónde moverse si el robot encuentra varias posibilidades.

Mediciones de ejecución para cada algoritmo:

Tiempo estimado	Trémaux	Garantía	Cadena	Wall follower
Laberinto 1	24s	20s	35s	10s
Laberinto 2	20s	16s	34s	11s
Laberinto 3	15s	12s	30s	6s

También se busca almacenar la menor cantidad de información en la memoria, balanceando la rapidez del robot (la cual, puede desviarlo ligeramente o hacer que rote y colisione con las paredes), con la certeza de que encontrará una salida.

Ahora, para solucionar el problema de los loops, se propone la solución siguiente:

Como los pasos a llevar se guardarán en un stack, se tendrá un método para controlar cuándo el robot ha dado un set de pasos que lo llevan a la misma posición inicial. Por ejemplo, los siguientes sets de pasos llevan a un desplazamiento cero.

Arriba - derecha - abajo - izquierda.
Arriba - abajo
etc.

Entonces, cuando el robot regrese a su posición inicial, se cuantificará cuántos movimientos hizo antes de regresar a su origen. Esta cantidad se dividirá dentro de dos y, en el caso de ser impar la cantidad de movimientos, se obtendrá el entero mayor; en ambos casos se obtendrá el valor de x. Luego, se le instruirá al robot que, luego de esa x cantidad de pasos, evalúe si a su izquierda se encuentra una pared. Si esta se encuentra, el robot gira 180 grados y sigue con el algoritmo. Si esta pared no se encuentra, el robot gira hacia la izquierda y luego hacia delante hasta encontrar una pared; cuando la encuentra, el robot gira de nuevo a la izquierda y sigue con el algoritmo predeterminado. Además, si el robot se encamina a otra isla, se repite este proceso hasta que el robot salga de la misma.

Estructura de datos a utilizar

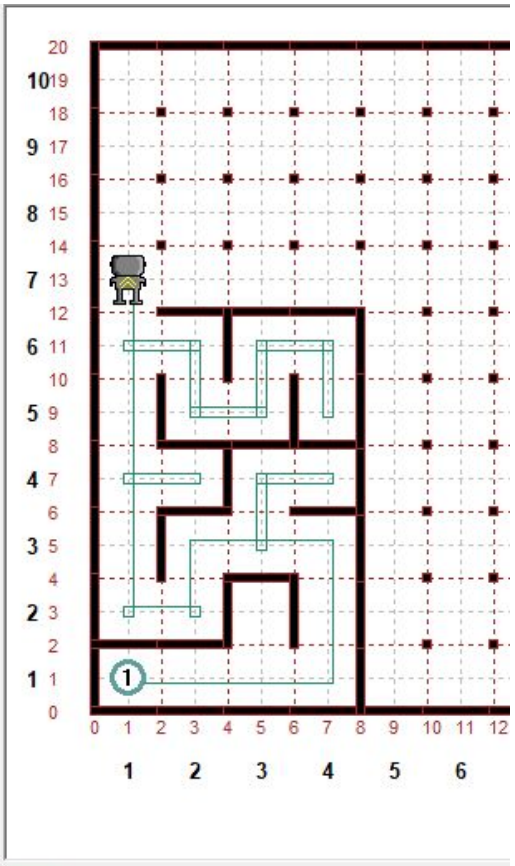
Stack, la cual archivará cada uno de los movimientos.

Pseudocódigo:

1. Se inicializa el robot
2. Iterar hasta encontrar punto inicial
3. if Pared derecha = false
4. Girar derecha
5. Moverse delante
6. else if Pared frontal = false
7. Moverse delante
8. else
9. Girar izquierda
10. Apagar al salir del laberi

Evidencia de funcionamiento e implementación en RUR

```
# Proyecto 1 - AED
# Antonio Reyes
# Kevin Macario
# Rodrigo Urrutia

def turn_right():
    repeat(turn_left, 3)

def mark_starting_point_and_move():
    put_beeper()
    while not front_is_clear():
        turn_left()
    move()

def follow_right_wall():
    if right_is_clear():
        turn_right()
        move()
    elif front_is_clear():
        move()
    else:
        turn_left()

found_starting_point = next_to_a_beeper#

#=== End of definitions; begin solution

mark_starting_point_and_move()
--> --> --> -->
while not found_starting_point():
    follow_right_wall()
--> -->
turn_off()
```



Literatura consultada

Algoritmo de Tremaux. (15 de Diciembre de 2016). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Algoritmo_de_Tremaux

Gámez, J. C. (2015). *¿Cómo escapo de este laberinto?* Obtenido de Matemáticas Digitales: http://www.matematicasdigitales.com/como-escapo-de-este-laberinto/

Snapp, R. R. (19 de Septiembre de 2014). *Trémaux's Algorithm for Threading a Maze*. Obtenido de cems.uvm.edu: http://www.cems.uvm.edu/~rsnapp/teaching/cs32/notes/tremaux_rules.pdf

Salvatierra D. (2017). Robot resuelve laberintos. Extraido de: https://juegosrobotica.es/robot-resuelve-laberintos/

ArcBotics. (2016). Maze Solving. Retrieved from ArcBotics: http://arcbotics.com/lessons/maze-solving-home-lessons/

Sandoval, D. (2017). Robot resuelve laberintos. Retrieved from Juegos Robotica: https://juegosrobotica.es/robot-resuelve-laberintos/