Universidad del Valle de Guatemala
Algoritmos y Estructura de Datos
David Valenzuela 171001
Sergio Romero 17909
Fernando Hengstenberg 17699

15 de febrero 2018

PROYECTO No. 1
Fase I

ALGORITMOS PARA SALIR DE UN LABERINTO:

- **Algoritmo Recursivo**: Si se le da una visión omnisciente del laberinto, un algoritmo recursivo simple puede decirle a uno cómo llegar al final. El algoritmo recibirá un valor X e Y inicial. Si los valores X e Y no están en una pared, el método se llamará a sí mismo con todos los valores adyacentes X e Y, asegurándose de que no haya usado previamente esos valores X e Y. Si los valores X e Y son los de la ubicación final, guardará todas las instancias previas del método como la ruta correcta. con una complejidad de O(n) (Juan,2012).

- **Maze-routing algorithm;** es un método de bajo costo para encontrar el camino entre dos ubicaciones del laberinto. El algoritmo se propone inicialmente para el dominio de multiprocesadores de chips (CMP) y garantiza que funcione para cualquier laberinto basado en grid. Además de encontrar caminos entre dos ubicaciones de la cuadrícula (laberinto), el algoritmo puede detectar cuando no hay una ruta entre la fuente y el destino. Además, el algoritmo debe ser utilizado por un viajero interno sin conocimiento previo del laberinto con complejidad de memoria fija, independientemente del tamaño del laberinto; requiriendo 4 variables en total para encontrar la ruta y detectar las ubicaciones inalcanzables. Sin embargo, el algoritmo no es para encontrar el camino más corto.con una complejidad O(MN) (Gabriel,2010).

- ***Algoritmo de Tremaux**:* Es un método para completar un laberinto de una forma verbal y simple. El algoritmo de Tremaux consiste en lo siguiente: No seguir el mismo camino 2 veces. Si llega a un cruce nuevo, siga el camino. Si llega a un cruce viejo o a un callejón sin salida, retroceda hasta la entrada del camino, y tome uno nuevo. Si el camino lo lleva a otro camino viejo, tome cualquier otro camino. Con una complejidad de Big O, de $O(V E^2)$. (Pineda, 2015)

- **Regla de la mano derecha (o izquierda):** Este algoritmo consiste en detectar si hay un objeto en alguno de los lados hay una pared o un obstáculo, y al no encontrar girar en esa dirección, es muy importante que siempre se detecte si hay una pared o no, para asegurar que se pasen por todas las paredes hasta encontrar la salida. El problema de este algoritmo es que la velocidad con la que encuentra la salida depende el punto en el que empieza el laberinto.

Algoritmo Para Utilizar en el Proyecto.

De todos los Algoritmos investigados anteriormente se hizo una minuciosa investigación sobre cada uno de ellos para lograr escoger el mejor entre los 4, y que se adaptará de una mejor manera a todas nuestras necesidades tratando así de resolver el laberinto en el menor tiempo posible y algunos de estos algoritmos no aseguraban poder resolver el laberinto o que pudieran tardar demasiado, estos y otros factores importantes fueron tomados en cuenta para poder tomar nuestra decisión a continuación mostraremos un pequeño resumen de la investigación:

Nuestro primer algoritmo fue el Algoritmo Recursivo, el cual no nos aseguraba que el robot pudiera terminar o encontrar la salida del laberinto con éxito, pero la principal ventaja fue que el código es corto. las desventajas es que necesita de muchas variables para poder resolver el problema del laberinto, y ademas de esto puede llegar a necesitar demasiada memoria por lo cual fue descartado para nuestro problema de los laberintos.

El segundo algoritmo era Maze-routing algorithm, el cual nos aseguraba completar o encontrar la salida de cualquier laberinto que este basado en grind, puede detectar si tiene paredes enfrente o a los lados, y puede ser utilizado por el robot sin tener previo conocimiento del laberinto antes, solo son necesarias cuatro variables con lo cual esto lo hacia uno de los mejores codigos para ser elegido, pero su única desventaja era que no encontraba el camino más rápido, esto quiere decir de que era demasiado lento, ya que para nuestro proyecto se requiere de terminar el laberinto en menos de 5 minutos o en el menor tiempo posible fue descartado de nuestras opciones.

El tercer algoritmo era *Algoritmo de Tremaux,* dice poder encontrar la salida de cualquier laberinto en cuatro pasos Ya dichos anteriormente, la desventaja de este algoritmo es que necesita de bastante tiempo para poder terminar el laberinto y de una gran variedad de variable teniendo así una complejidad de Big O, de $O(V E^2)$. este fue descartado de nuestra decisión por el tiempo que tarda en resolver los laberintos.

Por último tenemos al Algoritmo de la Mano derecha, el cual será el que utilizaremos para nuestro proyecto y para resolver todo tipo de laberintos con nuestro robot por las siguientes razones, el algoritmo se trata de que el robot detecte las paredes de los lados y de enfrente, al momento de que este no detecte ninguna pared al lado derecho este girara hacia ese lado y seguirá de frente, al momento de detectar una pared de frente girara hasta que no tenga una pared de frente o que tenga un espacio para girar a la derecha hasta que pueda resolver el laberinto y llegar a la salida, la única desventaja de este algoritmo es que si el robot no comienza dentro del laberinto o no tenga ninguna pared que detectar no podrá ni comenzar el laberinto, pero nos asegura que podra terminar todo tipo de laberinto cuando comienze con una pared para detectar, ademas de esto no necesita demasiadas variables ni mucha memoria, es uno de los algoritmos más rápidos en terminar los laberintos, esto también depende de donde empiece el laberinto o de donde se encuentre la salida. por estas razones escogimos este algoritmo para poder completar nuestro proyecto y para poder resolver cualquier tipo de laberintos en el menor tiempo posible.

Implementación de Estructuras de datos:

Una de los problemas principales del algoritmo que estamos implementando es el hecho que podría entrar en un ciclo infinito si el laberinto tienen paredes con seguidas del lado derecho, para solucionarlo se implementara un Stack, el stack almacenará los datos del camino que ya se ha recorrido, estos datos son las coordenadas (x,y) de los puntos que ha recorrido. Cada punto que avanza se verifica si es la última posición del stack, si es asi dio una vuelta, en este caso se tomara otro camino diferente, de este modo se utilizara un ADT para solucionar el problema de nuestro algoritmo.

Diagrama de clases:

Implementación de fase 1 es solamente del algoritmo en el entorno de python rurple, por lo que el stack aun no fue implementado y no forma parte de este diagrama.

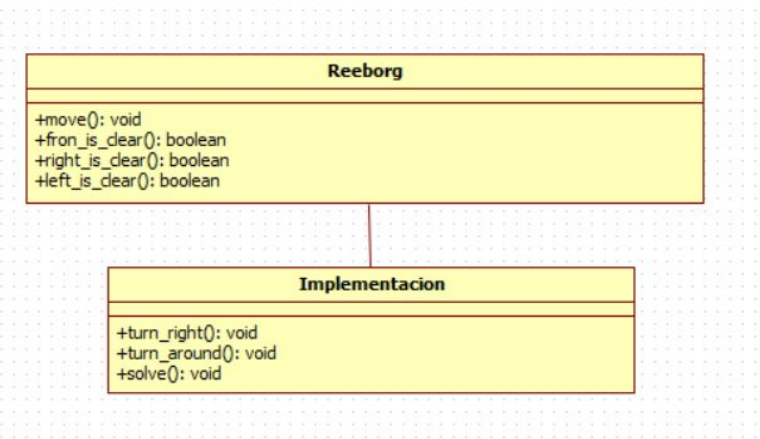
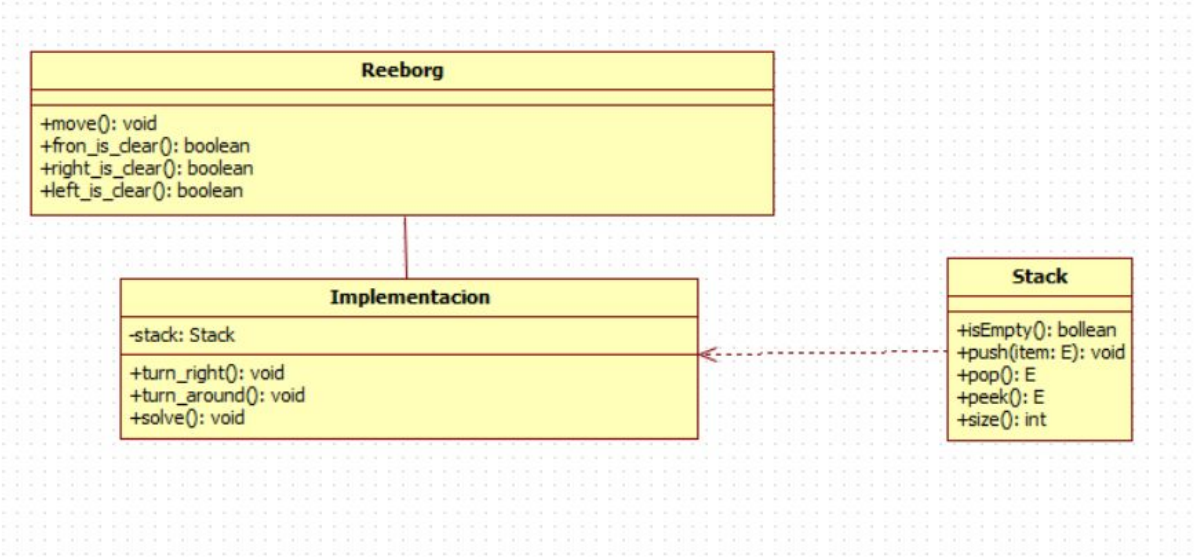


Diagrama de clase implementando Stack:



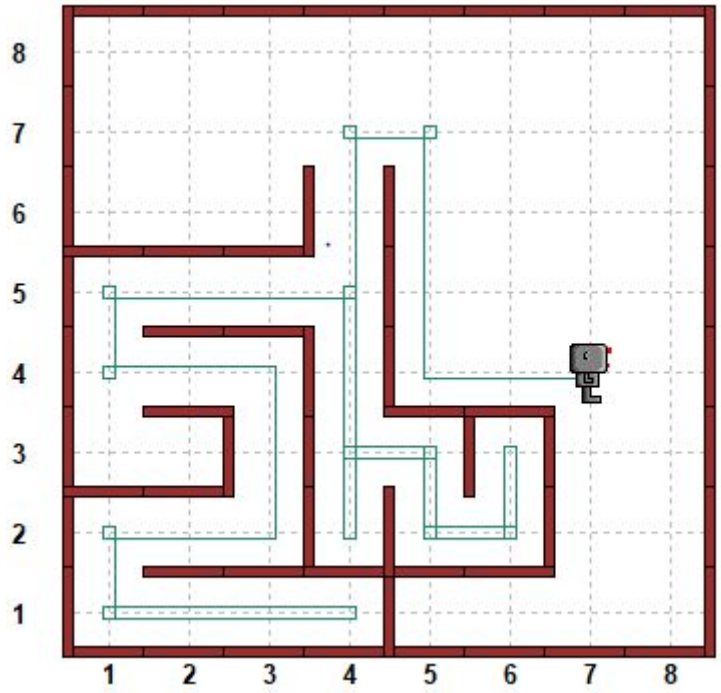
Pseudocódigo del algoritmo de la mano derecha:

1. Si el frente está libre y hay una pared a la derecha, avanzar.
2. Si el frente no esta libre, pero no hay pared a la derecha, girar a la derecha y avanzar.
3. Si solo la izquierda está libre, girar a la izquierda.
4. Si llego a un callejón sin salida, dar una vuelta de 180 grados.

5. Repetir hasta salir del laberinto.

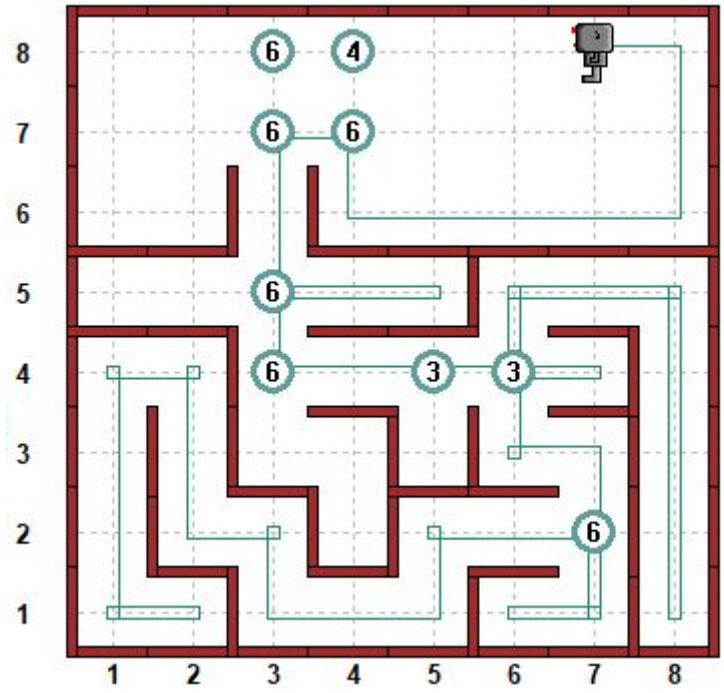
EJECUCIÓN DEL LABERINTO EN RUR PLE

Laberinto prueba 1:



Corrida 1; 1.27 segundos
Corrida 2: 1.22 segundos

Laberinto Prueba No. 2



Corrida No. 1: 1.52 segundo

Corrida No. 2: 1.57 segundo

Link repositorio de github: https://github.com/val171001/proyectoADT.git

REFERENCIA BIBLIOGRAFICAS

Pineda P. (2015). Obtenido de Algoritmos de Tremaux extraído de:
https://sinfallas.wordpress.com/2015/03/23/algoritmo-de-tremaux/

Gamez J. (2015). Obtenido de ¿Cómo escapo de este laberinto?, extraido de:
http://www.matematicasdigitales.com/como-escapo-de-este-laberinto/

Juan G. (2012). Obtenido de Algoritmos.pdf Extraido de:
http://www.cime.cl/archivos/ILI134/9261_Complejidad_Algoritmos_Recursivos.pdf

Gabriel A (2010). obtenido de Lec6.pdf Extraído de:
http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf