|  |
| --- |
|  |
| Retro-rastreador Recursivo |
| Proyecto para Taller de Lenguajes I |
|  |
| **Vignoli, Lucas Quintana Nicolás** |
| **23/06/2016** |

|  |
| --- |
| Informe del proceso de desarrollo y manual de uso del programa “Retro-rastreador Recursivo”, diseñado y escrito por Vignoli Lucas y Quintana Nicolás |

Índice Temático

[Introducción 2](#_Toc454440758)

[Manual de Uso 3](#_Toc454440759)

[Manual del Generador 3](#_Toc454440760)

[Instrucciones de dibujo de mapas 3](#_Toc454440761)

[Instrucciones de generación automática de mapas 3](#_Toc454440762)

[Manual del Juego 4](#_Toc454440763)

[Estructura de la cabecera 4](#_Toc454440764)

[Funciones creadas 4](#_Toc454440765)

[Conclusiones 6](#_Toc454440766)

[Bibliografía 6](#_Toc454440767)

# ­­­Introducción

Comenzamos la creación del programa intentando diseñar un algoritmo para generar un laberinto. Con el tiempo empezamos a investigar cómo otras personas lo hicieron antes hasta que nos encontramos con el algoritmo “Recursive Backtracker”, descrito a continuación:

1. **Hacer que la celda inicial sea la actual y marcarla como visitada**
2. **Mientras haya celdas visitadas**
   1. **Si la celda actual tiene vecinos que no hayan sido visitados**
      1. **Elegir aleatoriamente uno de los vecinos no visitados**
      2. **Colocar la celda actual a la pila**
      3. **Remover las paredes entre la celda actual y la elegida**
      4. **Hacer que la celda elegida sea la actual y marcarla como visitada**
   2. **Sino**
      1. **Sacar la celda tope de la pila**
      2. **Hacerla la actual**

Este método asume tener lo que es básicamente un grafo con celdas que cuentan con cuatro paredes y usa el método “búsqueda por profundidad” (*dfs* por sus siglas en inglés). Además usa una pila, la cual implementamos con una lista enlazada.

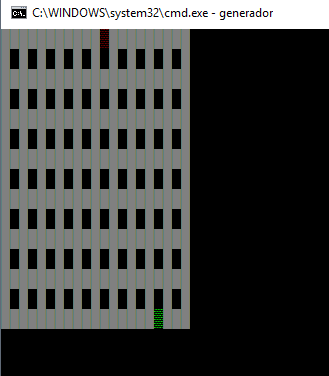


Imagen de un "laberinto" inicializado previo al algoritmo de generación, los espacios negros son vacíos, los grises son paredes y los bloques rojos y verdes representan la entrada y salida respectivamente

En mi adaptación del algoritmo cada celda tiene una variable que designa si es en sí una pared y me muevo en pasos de 2 celdas, para saltar las paredes intermedias.

# Manual de Uso

El programa consta de dos aplicaciones, el generador y el juego propiamente dicho.

## Manual del Generador

El generador de laberintos cuenta con un pequeño menú al principio que permite elegir entre generar un mapa, leer instrucciones o salir.

Las instrucciones describen brevemente los pasos a seguir para generar un mapa.

Al comenzar a generar un mapa hay que ingresar el nombre, no puede ser de más de 20 caracteres, después de eso uno debe colocar las dimensiones que tendrá, luego el programa pide elegir entre dibujar el mapa manualmente y generarlo automáticamente.

### Instrucciones de dibujo de mapas

Si uno elige dibujar el mapa, se dibujará en pantalla un mapa formado completamente por paredes con el carácter del jugador en el centro, al moverlo con las flechas dejará espacios vacíos y al usar las teclas W A S D colocará un premio en la dirección correspondiente. La barra espaciadora recorre entre modo premio (en el que se coloca premios con W A S D) y el modo pared (en el que se colocan paredes con las mismas teclas).

Cuando termine de dibujar el mapa presiona la tecla ENTER y el mapa vuelve a cargarse para seleccionar la entrada y la salida, en este modo el jugador no se puede mover a través de las paredes. Una vez que selecciona la entrada el proceso se repite para la salida.

### Instrucciones de generación automática de mapas

Si uno elige que el programa genere el mapa automáticamente, le pedirá escribir una semilla, que la computadora usará para el generador de números aleatorios, o puede poner una semilla aleatoria, escribiendo -1, o cualquier número negativo.

Al final del proceso, sin importar si eligió dibujar o al generador, el mapa completo será dibujado y aparecerá la pregunta de si desea guardarlo, si elige “sí”, el mapa será guardado en un archivo con formato legible por el juego. Una vez guardado el juego, podrá decidir si volver al menú y hacer otro mapa o salir.

## Manual del Juego

Cuando uno abre el juego se le presentan las opciones de elegir un mapa, ver las instrucciones o salir.

Si selecciones elegir mapa le será presentado con una lista de los mapas (si es que hay alguno en memoria) y podrá elegir uno o volver al menu.

Cuando elija un mapa, se abrirá y podrá mover al jugador con las flechas, el objetivo del juego es llegar a la salida, marcada en verde.

El puntaje es calculado como la cantidad de movimientos que hace el jugador. El objetivo es conseguir el menor puntaje posible, en otras palabras, llegar a la salida en la menor cantidad de movimientos posibles. Los “premios” dispersos por el mapa reducen el puntaje en una cantidad variable según el tamaño del mapa.

# Estructura de la cabecera

La cabecera contiene 5 secciones, el nombre (guardado en una cadena de 20 caracteres), las ubicaciones del punto de inicio y final y las dimensiones de la matriz que forma al mapa.

Luego de la cabecera está el “código fuente”, un código numérico que representa el orden de paredes, espacios vacíos y premios en el mapa. Al final, luego del código fuente, está el mejor puntaje en ese mapa.

# Funciones creadas

* *new\_point(x, y)*: Constructora de un punto en 2D, a través de dos enteros x e y.
* *equal\_points(a, b)*: Función que compara las coordenadas de dos puntos y devuelve verdadero si son iguales.
* *Oculta\_cursor()*: Oculta el cursor, para razones estéticas durante el juego.
* *PilaVacia()*: Constructora de pila.
* *EsPilaVacia(P)*: Toma una pila y da verdadero si esta vacía.
* *PUSH(P, x)*: Toma un elemento y lo pone en la pila.
* *TOP(P)*: Toma el elemento en el tope de la pila
* *POP(P)*: Borra el elemento del tope de la pila.
* *unvisited\_cells(mapa)*: Revisa si en el mapa quedan celdas que no hayan sido visitadas
* *rand\_choose\_cell(mapa, p)*: Elige una celda adyacente al punto p del mapa.
* *exist\_unvisited\_neighbours(mapa, p)*: Devuelve verdadero si existe al menos una celda no visitada adyacente a la celda ubicada en el punto p.
* *remove\_walls(c, p, mapa)*: Deshace la pared entre el punto c y el punto p del mapa.
* *new\_map(x, y)*: Construye e inicializa un mapa con x e y dimensiones.
* *print\_unfinished\_map(mapa)*: Dibuja el mapa incompleto (sin entradas, salidas o premios, solo paredes).
* *print\_map(mapa, inicio, fin)*: Dibuja el mapa completo.
* *generate\_map(mapa, start, end)*: Algoritmo generador del laberinto.
* *check\_prize(celda)*: Revisa si una celda tiene premio o no.
* *save\_header(mapa, nombre, inicio, fin)*: Devuelve la estructura cabecera con el nombre del mapa, las dimensiones y los puntos.
* *save\_source(mapa, fuente)*: Genera el código fuente y lo guarda en una cadena de caracteres.
* *nuevo\_jugador(inicio)*: Constructora de un jugador, pide la ubicación de inicio.
* *dibujarjugador(jugador)*: Dibuja el carácter del jugador en su ubicación.
* *moverjugador(jugador, mapa, tecla)*: Revisa la tecla presionada, revisa que en esa dirección no haya una pared y mueve acorde.
* *GameLoop(jugador, inicio, fin, mapa, lista)*: Dibuja el mapa, comienza el juego y guarda los pasos del jugador en la lista.
* *moverpincel(pincel, mapa, tecla, premio)*: Parecida a moverjugador, permite al pincel moverse en el mapa.
* *draw\_map(dimx, dimy)*: Dibuja el mapa y comienza el modo dibujo.
* *check\_for\_maps()*: Abre el archivo con la cantidad de mapas y devuelve dicha cantidad.
* *list\_maps(n, salida)*: Pide el número de mapas que hay guardados en memoria, escribe los nombres de todos y devuelve el nombre del archivo del mapa elegido.

# Conclusiones

Este proyecto comenzó con la idea de crear un laberinto, pero rápidamente creció en un contexto en el cual probar nuestras habilidades, aprender nuevas técnicas y reafirmar conocimientos de la materia.

Cuando sentíamos que lo que nosotros interpretábamos como la tarea estaba terminado, se nos ocurría una forma de mejorarlo, algo nuevo que agregarle o un profesor proponía una idea que no se nos había ocurrido y eso llevaba al proyecto por otro camino.

# Bibliografía

“El lenguaje de la programación C” de Brian Kernighan y Dennis Ritchie.

Algoritmo “Recursive Backtracker” de Wikipedia