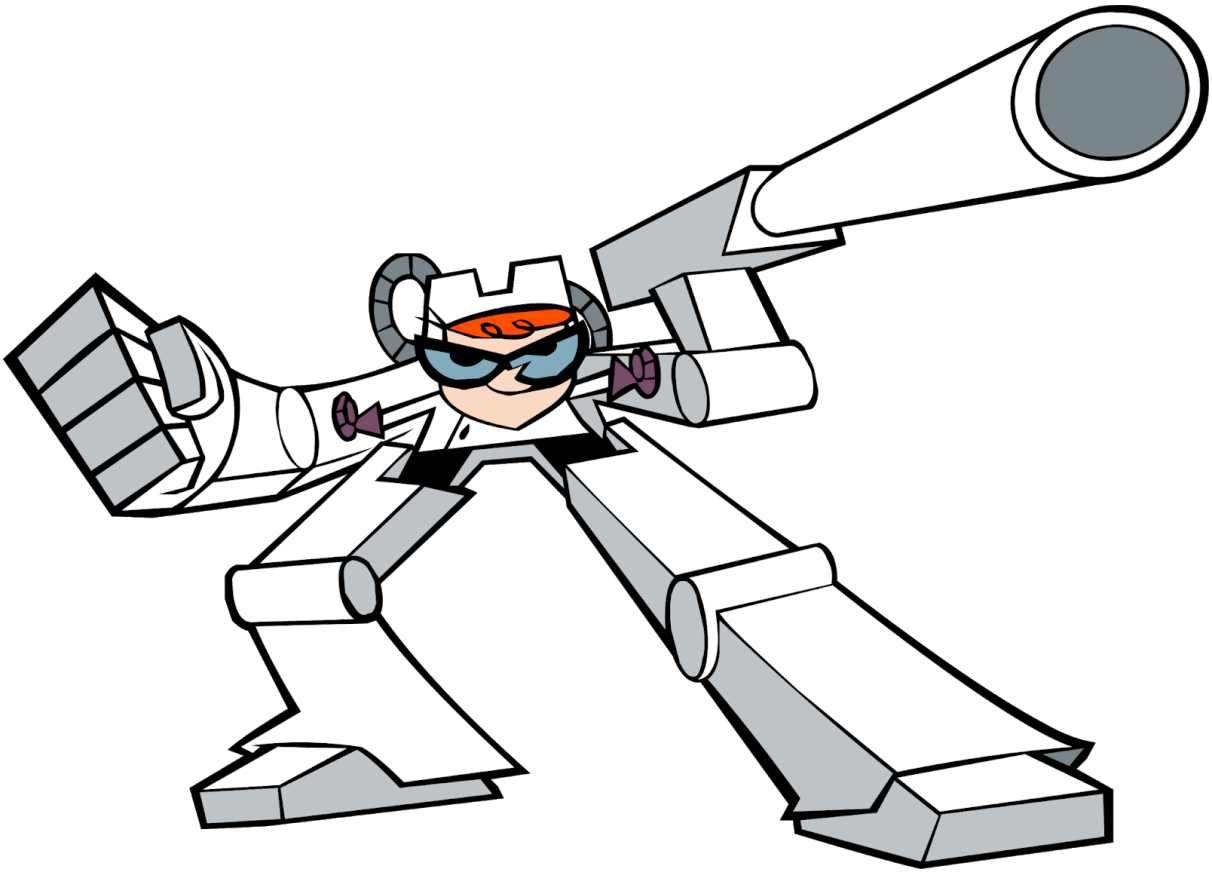
Algoritmos y Programación I
Curso Mendez

.UBAfiuba
FACULTAD DE INGENIERÍA

Escape del laboratorio

TP2



Fecha de Presentación: 12/11/2020

Fecha de Vencimiento: 17/12/2020

1. Introducción

Dexter es un niño genio de 10 años, de pelo rojo, que tiene laboratorio lleno de inventos suyos al que accede pronunciando variadas contraseñas o activando los interruptores ocultos en un estante para libros, que es realmente una puerta. En muchas ocasiones accede a su laboratorio presionando un botón dentro de un libro en especial, y a veces identificándose por medio de un lector de retina o identificación de voz.

Dexter se encuentra constantemente en conflicto con su extremadamente escandalosa hermana Dee Dee, de 12 años, quien siempre logra entrar a su laboratorio a pesar de sus esfuerzos por mantenerla afuera. Pero, Dee Dee se las ingenia para merodear por su laboratorio, y terminar saboteando accidentalmente todos sus trabajos.

Por eso, Dexter instaló un sistema de seguridad que, al detectar otro personaje, que no fuese él, se activa encerrando al personaje en habitaciones con obstáculos, guardias robots y artefactos raros, con una única forma de salir, obteniendo la llave maestra de dicha habitación.

Pero Dee Dee conoce a su hermano, y sabía que algo raro estaba pasando con su laboratorio. Para asegurarse de que ella no corra peligro, te pide que vayas a recoger una de sus muñecas que dejó "olvidada". A pesar de que has implementado su sistema de detección de personajes para poder burlarlo, eres reconocido como un intruso por el sistema, y tienes que escapar del laboratorio lo antes posible...

2. Objetivo

El presente trabajo práctico tiene como objetivo que el alumno:

- Diseñe y desarrolle las funcionalidades de una biblioteca con un contrato preestablecido.
- Se familiarice con y utilice correctamente los tipos de datos estructurados.
- Desarrolle una interfaz gráfica amigable y entendible para el usuario.

Por supuesto, se requiere que el trabajo cumpla con las buenas prácticas de programación profesadas por la cátedra.

Se considerarán críticos la modularización, reutilización de código y la claridad del código.

3. Enunciado

Salir del laboratorio es una tarea difícil pero no imposible, Dexter tiene una llave maestra por cada habitación protegida por ciertos artefactos y cosas para que no cualquiera la encuentre.

La llave está escondida en los últimos dos niveles de su laboratorio, y cada uno de estos niveles presenta algunas dificultades particulares.

Una nota en la entrada del laboratorio reza el siguiente mandamiento: CUIDADO: En este laboratorio, las baldosas son resbaladizas!!"

Los personajes solo podrán moverse de una pared a la otra, es decir que sus movimientos no serán baldosa a baldosa, sino que al moverse, irán hasta la próxima pared.

3.1. Niveles

Cada uno de los niveles puede verse como una matriz, donde cada una de las celdas de dicha matriz, representa una baldosa del nivel en el que se encuentra el personaje.

En los niveles habrá obstáculos, que impedirán salir de él y herramientas que ayudarán al personaje en la odisea de escape.

Los obstáculos pueden ser baldosas pinche, guardias robot o bombas, el rol de cada uno se explicará en la sección Obstáculos.

Las herramientas pueden ser monedas, baldosas teletransportadoras, interruptores o llaves, el rol de cada una se explicará en la sección Herramientas.

- Por cada dos baldosas pinche, habrá una moneda en el nivel.
- Por cada guardia robot, habrá una baldosa teletransportadora.
- Si en el nivel hay una bomba, entonces habrá un interruptor.

A continuación se detallan configuraciones iniciales para cada nivel que la cátedra entiende que son razonables para lograr que la dificultad del juego sea incremental a lo largo de los niveles.

3.1.1. Nivel 1

El primer nivel tiene las siguientes características:

- 4 baldosas pinche.
- 2 guardias robot.
- Sin bomba.
- Dimensión 12x12 (incluyendo paredes exteriores).

3.1.2. Nivel 2

El segundo nivel tiene las siguientes características:

- 6 baldosas pinche.
- 3 guardias robot.
- Sin bomba.
- Dimensión 17x17 (incluyendo paredes exteriores).

3.1.3. Nivel 3

El tercer nivel tiene las siguientes características:

- 6 baldosas pinche.
- 3 guardias robot.
- Con bomba.
- Dimensión 12x12 (incluyendo paredes exteriores).

3.1.4. Nivel 4

El cuarto nivel tiene las siguientes características:

- 10 baldosas pinche.
- 4 guardias robot.
- Con bomba.
- Dimensión 17x17 (incluyendo paredes exteriores).

3.2. Personajes

Cada personaje, detectado por el trabajo práctico 1, tendrá características particulares que le darán una ventaja en este reto de salir vivo del laberinto.

- **Blue y Bellota**: Blue por ser imaginario y Bellota por ser Superpoderosa obtendrán una segunda oportunidad con 10 movimientos si es que mueren en alguno de los niveles.
- **Puro Hueso y Pollito**: Puro hueso por no tener carne y Pollito por ser finito, no serán afectados por las baldosas pinche.
- **Johnny Bravo y Coraje**: Johnny por enamorarlos con sus encantos y Coraje por darles tanta ternura, no serán atrapados por los guardias robot.

El personaje arrancará el primer nivel con 10 movimientos y en cada nuevo nivel se le sumarán 10 movimientos a los que le sobraron de los niveles anteriores.

3.3. Obstáculos

Los obstáculos son elementos que estarán en cada nivel e intentarán hacer que el personaje pierda todos sus movimientos y quede atrapado dentro del propio nivel.

- **Baldosas con pinches:** Si el jugador pisa los pinches perderá 2 movimientos.
- **Guardias Robots:** Se teletransportan a otra posición aleatoria dentro del nivel por cada movimiento del personaje, la nueva posición no debe ser la de una pared, ni otro obstáculo, herramienta, o la del personaje. Si el personaje choca al guardia, pierde todos sus movimientos.
- **Bomba:** En caso de que se pise la bomba, se pierde automáticamente.

Los obstáculos deben posicionarse aleatoriamente en el nivel al inicializar el mismo, cabe destacar que no pueden posicionarse distintos obstáculos en la misma baldosa o en una baldosa donde ya existe una pared, entrada o salida.

El orden de posicionamiento de los obstáculos al inicializar el nivel es indiferente, siempre y cuando se respete lo enunciado en el párrafo anterior.

A su vez, las baldosas pinche deben estar en baldosas contiguas.

3.4. Herramientas

Las herramientas son elementos que estarán en cada nivel e intentarán ayudar al personaje a salir vivo del nivel.

- **Llave**: La llave permite que, al obtenerla, se haga visible y accesible la salida del nivel.
- **Interruptor**: En caso de haber interruptor, encenderlo hace que se haga visible la llave. Si no hay interruptor, la llave será visible desde el comienzo.
- **Teletransportadores**: Al entrar en un teletransportador se saldrá por el siguiente teletransportador que se encuentre en el vector de herramientas.
- **AlgoCoins**: Son monedas que se encuentran en el piso del nivel, agarrarlas le suman al personaje 1 movimientos.

El hecho de encender el interruptor, agarrar una moneda o llave o entrar en un teletransportador están dados por pasar o pararse en la baldosa en la que se encuentra la herramienta.

Las herramientas deben posicionarse aleatoriamente en el nivel al inicializar el mismo, cabe destacar que no pueden posicionarse distintas herramientas en la misma baldosa o en una baldosa donde ya existe un obstáculo, pared, entrada o salida.

El orden de posicionamiento de las herramientas al inicializar el nivel es indiferente, siempre y cuando se respete lo enunciado en el párrafo anterior.

Es importante remarcar que al posicionarse sobre una moneda esta debe ser eliminada de las herramientas y brindar movimientos extra a el personaje.

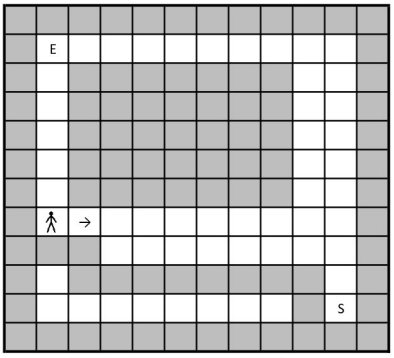
3.5. Modo de juego

Como lo informa el mensaje en la entrada del laboratorio, las baldosas de cada nivel son resbaladizas, lo que implica que para moverse dentro de cada nivel debe hacerse de una pared a la otra.

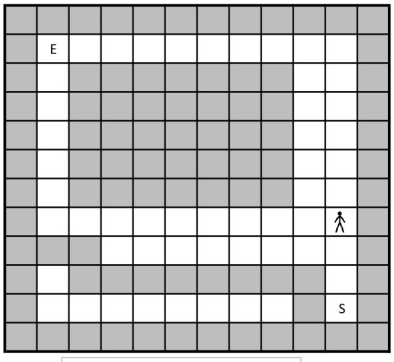
Esto implica que los personajes no pueden frenarse en una baldosa en particular, sino que su movimiento terminará cuando se choquen con una pared, con lo cual no pueden **elegir** esquivar a un obstáculo si este se encuentra en su camino.

El personaje entrará por la baldosa seleccionada como entrada y deberá salir por la baldosa demarcada como salida, para esto, deberá moverse de pared a pared según las posibilidades brindadas por cada nivel, para intentar no toparse con los obstáculos, aprovechar las herramientas y así llegar a la salida.

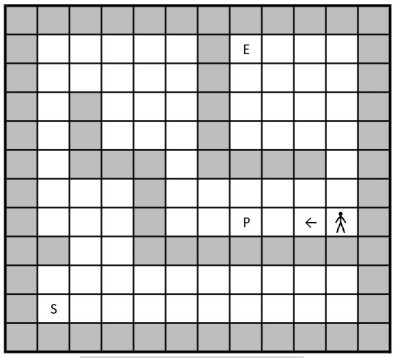
A continuación se muestran algunos ejemplos:



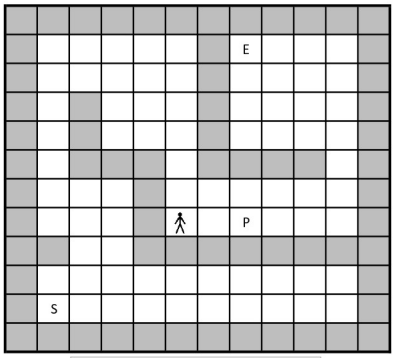
Posición inicial del personaje.



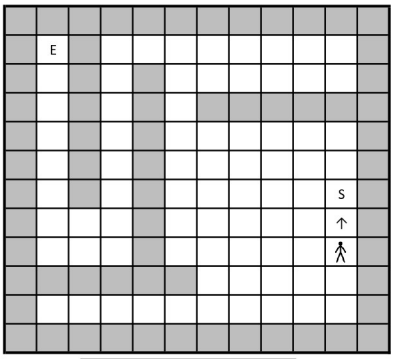
Posición final del personaje al moverse hacia la derecha.



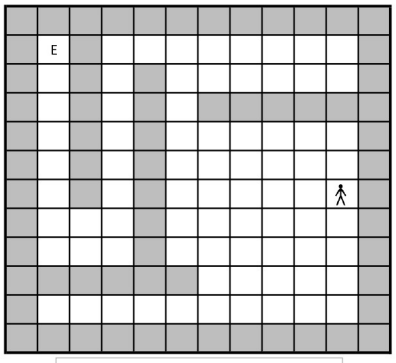
Posición inicial del personaje.



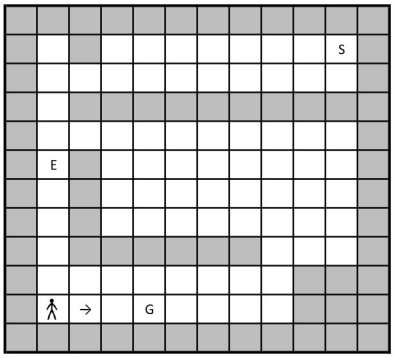
Posición final del personaje al moverse hacia la izquierda. El pinche solo le restó movimientos.



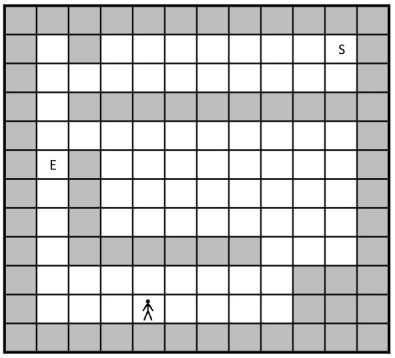
Posición inicial del personaje.



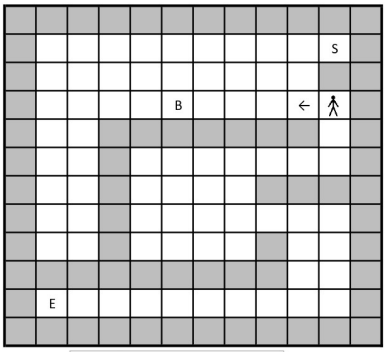
Posición final del personaje al moverse hacia arriba y chocar con la salida.



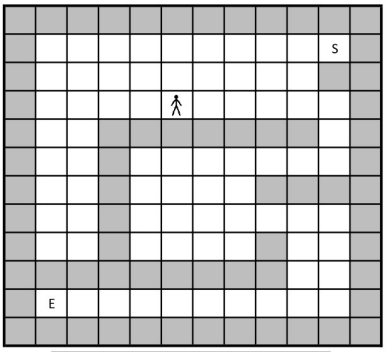
Posición inicial del personaje.



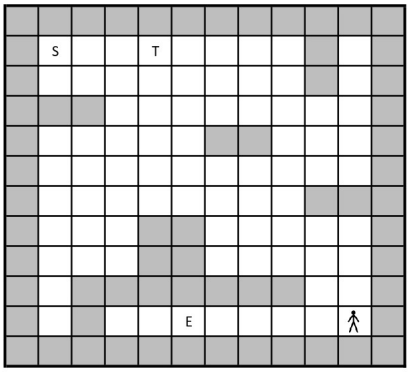
Posición final del personaje al moverse hacia la derecha, choca con el guardia y muere.



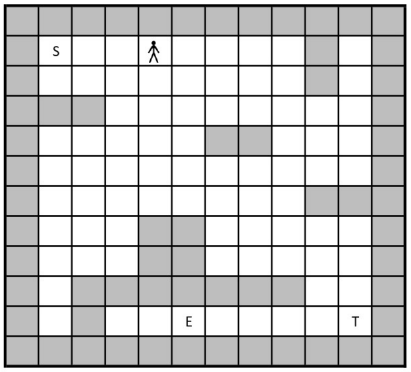
Posición inicial del personaje.



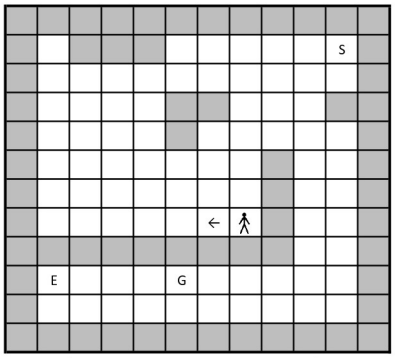
Posición final del personaje al moverse hacia la izquierda, choca con la bomba y muere.



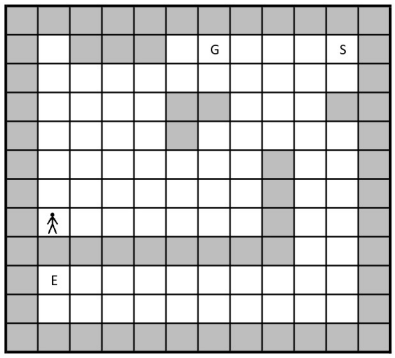
Posición inicial del personaje (sobre una baldosa teletransportadora).



Posición final del personaje (sobre otra baldosa teletransportadora).



Posición inicial del personaje y del guardia.



Luego de que el personaje se mueve hacia la izquierda, el guardia se mueve hacia otra posición aleatoria.

4. Especificaciones

4.1. Biblioteca escape_laboratorio.h

```c
#ifndef __ESCAPE_LABORATORIO_H__
#define __ESCAPE_LABORATORIO_H__

#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

#define MAX_OBSTACULOS 25
#define MAX_HERRAMIENTAS 25
#define MAX_NIVELES 4
#define MAX_PAREDES 250

typedef struct coordenada {
        int fil;
        int col;
} coordenada_t;

typedef struct personaje {
        char tipo;
        int movimientos;
        coordenada_t posicion;
        bool tiene_llave;
        bool presiono_interruptor;
} personaje_t;

typedef struct elemento {
        char tipo;
        coordenada_t posicion;
} elemento_t;

typedef struct nivel {
        coordenada_t entrada;
        coordenada_t salida;

        coordenada_t paredes[MAX_PAREDES];
        int tope_paredes;

        elemento_t obstaculos[MAX_OBSTACULOS];
        int tope_obstaculos;

        elemento_t herramientas[MAX_HERRAMIENTAS];
        int tope_herramientas;
} nivel_t;

typedef struct juego {
        nivel_t niveles[MAX_NIVELES];
        int nivel_actual;
```

```
48          personaje_t personaje;
49  } juego_t;
50
51
52  /*
53   * Inicializará el juego, cargando la informacion de los cuatro niveles
54   * y los datos del personaje.
55   */
56  void inicializar_juego(juego_t* juego, char tipo_personaje);
57
58  /*
59   * Recibe un juego con todas sus estructuras válidas.
60   * El juego se dará por ganado si el personaje está en el último nivel
61   * y posicionado en la salida.
62   * El juego se dará por perdido, si el personaje queda sin movimientos.
63   * Devolverá:
64   * ->  0 si el estado es jugando.
65   * -> -1 si el estado es perdido.
66   * ->  1 si el estado es ganado.
67   */
68  int estado_juego(juego_t juego);
69
70  /*
71   * Recibe el personaje con todas sus estructuras válidas, y la coordenada de la salida del nivel en
         ejecución.
72   * El nivel se dará por ganado cuando el personaje se
73   * posicione en la salida habiendo obtenido previamente la llave.
74   * Devolverá:
75   * ->  0 si el estado es jugando.
76   * ->  1 si el estado es ganado.
77   */
78  int estado_nivel(personaje_t personaje, coordenada_t salida);
79
80  /*
81   * Inicializara un nivel cargando su entrada,
82   * salida, obtáculos, herramientas y paredes.
83   */
84  void inicializar_nivel(nivel_t* nivel, int numero_nivel, int cantidad_baldosas_pinches, int
         cantidad_guardia, bool hay_bomba);
85
86
87  /*
88   * Mueve el personaje en la dirección indicada por el usuario
89   * y actualiza el juego según los elementos que haya en el camino
90   * del personaje.
91   * El juego quedará en un estado válido al terminar el movimiento.
92   * El movimiento será:
93   * -> w: Si el personaje debe moverse para la arriba.
94   * -> a: Si el personaje debe moverse para la derecha.
95   * -> s: Si el personaje debe moverse para la abajo.
96   * -> d: Si el personaje debe moverse para la izquierda.
97   * En caso de que en la dirección que se quiere mover haya una pared
98   * se contará como un movimiento, los guardias se moverán y
99   * el personaje quedará en la misma baldosa.
100  */
101 void mover_personaje(juego_t* juego, char movimiento);
102
103 /*
104  * Mostrará el juego por pantalla.
105  * Se recomienda mostrar todo aquello que le sea de
106  * utilidad al jugador, como los movimientos restantes,
107  * el nivel, los obstaculos posicionados, las paredes, etc.
108  */
109 void mostrar_juego(juego_t juego);
110
111 #endif /* __ESCAPE_LABORATORIO_H__ */
```

4.2. Convenciones

Se deberá utilizar la siguiente convención para los obstáculos y herramientas:

- **Baldosa Pinche**: P.
- **Moneda**: M.
- **Llave**: L.

- **Bomba**: B.
- **Interruptor**: I.
- **Entrada**: E.
- **Salida**: S.
- **Baldosa Teletransportadora**: T.
- **Guardia**: G.

Y para los personajes:

- **Coraje**: C.
- **Blue**: B.
- **Bellota**: S.
- **Puro Hueso**: H.
- **Pollito**: P.
- **Johnny Bravo**: J.

4.3. Biblioteca detector_personaje.h

Se debe crear una biblioteca con el trabajo práctico 1, ésta biblioteca solo tendrá un procedimiento con la siguiente firma:

```
1 void detectar_personaje(char* personaje_detectado);
```

5. Resultado esperado

Se espera que se creen las funciones y procedimientos para que el juego se desarrolle con fluidez.

Muchas de las funcionalidades quedan a criterio del alumno, solo se pide que se respeten las estructuras y especificaciones brindadas.

El trabajo creado debe:

- Interactuar con el usuario.
- Mostrarle al jugador la información del juego a cada momento.
- Informarle al jugador correctamente cualquier dato que haya sido ingresado incorrectamente.
- Informarle al jugador si ganó o perdió.
- Cumplir con las buenas prácticas de programación.
- Mantener las estructuras propuestas actualizadas a cada momento.

6. Compilación y Entrega

El trabajo práctico debe ser realizado en un archivo llamado juego.c, y la biblioteca de funciones para jugarlo escape_laboratorio.c y escape_laboratorio.h, y debe poder ser compilado sin errores con el comando:

```
1 gcc juego.c escape_laboratorio.c detector_personajes.c utiles.o -o juego -std=c99 -Wall -Wconversion
      -Werror -lm
```

detector_personajes.c y detector_personajes.h corresponden a la biblioteca creada con la parte del trabajo práctico 1 para obtención del personaje detectado.

utiles.o es un archivo compilado realizado por la cátedra, que pondrá a su disposición 3 funciones que pueden ser,justamente, útiles y su funcionamiento se explica en el anexo.

Por último debe ser entregado en la plataforma de corrección de trabajos prácticos **Chanutron2021** (patente pendiente), en la cual deberá tener la etiqueta ¡Exito! significando que ha pasado las pruebas a las que la cátedra someterá al trabajo.

Para la entrega en **Chanutron2021** (patente pendiente), recuerde que deberá subir un archivo zip conteniendo únicamente los archivos antes mencionados, sin carpetas internas ni otros archivos. De lo contrario, la entrega no será validada por la plataforma.

IMPORTANTE! Esto no implica necesariamente haber aprobado el trabajo ya que además será corregido por un colaborador que verificará que se cumplan las buenas prácticas de programación.

7. Anexos

7.1. Obtención de Números Aleatorios

Para obtener números aleatorios debe utilizarse la función **rand()**, la cual está disponible en la biblioteca stdlib.h.

Esta función devuelve números pseudo-aleatorios, esto quiere decir que, cuando uno ejecuta nuevamente el programa, los números, aunque aleatorios, son los mismo.

Para resolver este problema debe inicializarse una semilla, cuya función es determinar desde donde empezarán a calcularse los números aleatorios.

Los números arrojados por **rand()** son enteros sin signo, generalmente queremos que estén acotados a un rango (queremos números aleatorios entre tal y tal). Para ésto, podemos obtener el resto de la división de **rand()** por el valor máximo del rango que necesitamos.

Aquí dejamos un breve ejemplo de como obtener números aleatorios entre 10 y 30.

```c
#include <stdio.h>
#include <stdlib.h> // Para usar rand
#include <time.h>   // Para obtener una semilla desde el reloj

int main(){
        srand ((unsigned)time(NULL));
        int numero = rand() % 20 + 10; // la amplitud del rango es 20 y el valor mínimo es 10.
        printf("El valor aleatorio es: %i\n", numero);

        return 0;
}
```

7.2. Limpiar la Pantalla durante la Ejecución de un Programa

Muchas veces nos gustaría que nuestro programa pueda verse siempre en la pantalla sin ver texto anterior.

Para ésto, podemos utilizar la llamada al sistema **clear**, de esta manera, limpiaremos todo lo que hay en nuestra terminal hasta el momento y podremos dibujar la información actualizada.

Y se utiliza de la siguiente manera:

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
        printf("Escribimos algo\n");
        printf("que debería\n");
        printf("desaparecer...\n");

        system("clear"); // Limpiamos la pantalla

        printf("Solo deberiamos ver esto...\n");
        return 0;
}
```

7.3. utiles.o

En esta oportunidad la cátedra pondrá a disposición un biblioteca con 2 funciones, una para obtener la distribución de paredes de un nivel y otra para detener el tiempo por un tiempo.

Es pertinente aclarar que se envían 2 archivos **utiles.o**, uno para arquitecturas de 32 bits y otra para arquitecturas de 64 bits, use la que corresponda a su sistema operativo.

IMPORTANTE! Para el uso de esta biblioteca, deberá incluirse previamente la biblioteca **escape_laboratorio**, ya que la misma necesita de la definición de la estructura **coordenada_t**.

El .h de la biblioteca se muestra a continuación con las firmas propuestas.

```
#ifndef __UTILES_H__
#define __UTILES_H__
#include <stdio.h>

#define MAX_PAREDES 250


/* Pre condiciones: El parametro nivel debe contener el valor de un nivel (1 a 4).
 * Post condiciones: Devuelve el vector de coordenadas de las paredes cargado, junto a su respectivo tope.
*/
void obtener_paredes(int nivel, coordenada_t paredes[MAX_PAREDES], int *tope_paredes);

/* Pre condiciones: La variable segundos debe contener un valor positivo.
 * Post condiciones: Detendrá el tiempo los segundos indicados como parámetro.
*/
void detener_el_tiempo(float segundos);

#endif /* __UTILES_H__ */
```

De más está decir que no es obligatorio el uso de éstas funciones, pero la generación de la distribución de paredes de un nivel resuelve un problema que puede llevarles demasiado tiempo y detener el tiempo les permite acelerar o detener la ejecución del juego, en valores mas flexibles, ya que existe una función llamada **sleep**, pero su mínimo valor es 1 segundo.

Referencias

https://cartoonnetwork.fandom.com/es/wiki/El_Laboratorio_de_Dexter