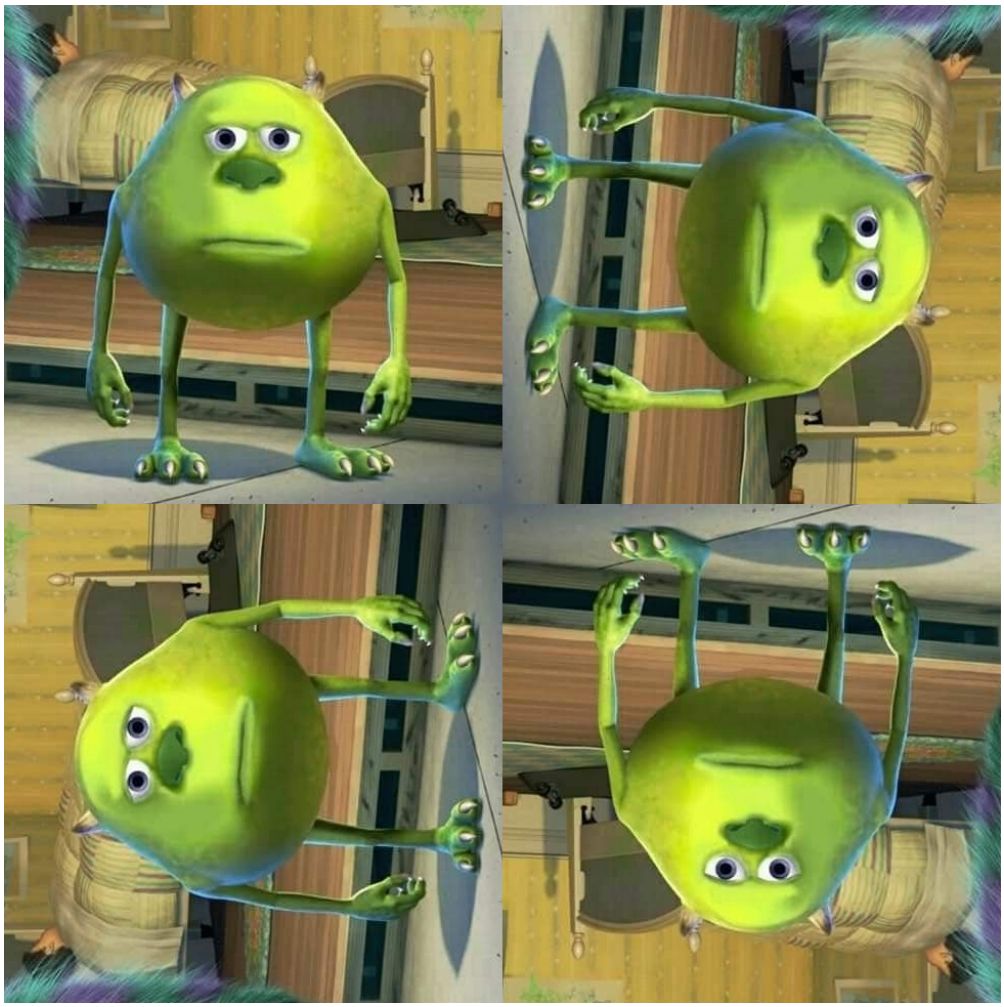
Algoritmos y Programación I

Curso Camejo

.UBAfiuba
FACULTAD DE INGENIERÍA

Trabajo Práctico N° 3

Mike en el multiverso de los papeleos



Fecha Presentación	31/05/2022
Fecha Entrega	23/06/2022

1. Introducción

Luego de todos los problemas que tuviste para llegar a FIUBA, finalmente pudiste rendir el temible examen. Ahora llegó el momento de firmar la libreta, pero te diste cuenta que el día que está el profesor vos no podes ir. Te acordás que un amigo tuyo también tiene que ir, y le pedís si te hace el favor de llevar tu libreta a firmar.

Tu amigo, **Mike Wazowski**, acepta el pedido, pero como buen interesado que es, lo hace a cambio de un obsequio.

Al llegar al Departamento de Computación, Mike se encuentra con **Roz**, quien, una vez más, le recrimina que **"no ordenó su papeleo anoche..."**. Y en venganza, repartió todos los papeles por distintas habitaciones para que se le haga más difícil a Mike ordenarlos. Y, lamentablemente, Roz no le va a firmar la libreta hasta que él junte todo su papeleo.

Sin posibilidad de volver a evadir a Roz, llegó el momento de que Mike junte su papeleo, y nosotros vamos a acompañarlo en esta aventura.

2. Objetivo

El presente trabajo práctico tiene como objetivo evaluar a los alumnos en aspectos fundamentales de la programación. Estos aspectos son:

- Validación de datos ingresados por el usuario.
- Diseño y desarrollo de funcionalidades de una biblioteca con un contrato preestablecido.
- El correcto uso de estructuras de control.
- Tipos de dato simples y estructurados.
- Buenas prácticas de programación.
- Modularización.

3. Enunciado

Como desarrolladores de este juego, debemos ayudar a Mike a juntar todo su papeleo y llevárselo a Roz para que finalmente tu libreta sea firmada.

El juego consiste en pasar por una serie de raras habitaciones, para que Mike junte el papeleo en orden, que estará disperso en ellas.

Habrá 3 habitaciones donde buscar. Dichas habitaciones, estarán delimitadas por los bordes del terreno, y además habrá paredes, obstáculos y herramientas para ayudarte, o perjudicarte, en tu búsqueda.

Mike tendrá distinta cantidad de movimientos iniciales disponibles en cada nivel, y los movimientos que le sobren del nivel anterior, se sumarán a los iniciales del próximo nivel.

Pero cuidado! La habitación va a rotar con tus movimientos.

En esta segunda etapa de desarrollo, se deberá implementar la lógica del funcionamiento total del juego.

3.1. Gravedad

El jugador podrá caminar horizontalmente sobre paredes, en caso de moverse hacia una coordenada en la cual no haya una pared debajo, este deberá caer hasta chocar con alguna.

3.2. Acciones del jugador

El jugador podrá moverse, rotar el mapa, utilizar martillos o extintores.

A continuación se detallan las acciones y los caracteres que se deben ingresar por cada una de ellas:

3.2.1. Movimientos laterales

El jugador podrá moverse lateralmente de a una coordenada a la vez.

- **Movimiento a izquierda**: 'A'.
- **Movimiento a derecha**: 'D'.

Por cada movimiento lateral logrado se deberá restar un movimiento al jugador.

3.2.2. Movimientos rotacionales

El jugador podrá rotar el mapa en sentido horario o antihorario.

- **Rotación horaria**: 'E'.
- **Rotación antihoraria**: 'Q'.

Por cada movimiento rotacional se deberá restar un movimiento al jugador.

3.2.3. Martillos

El jugador podrá utilizar un martillo de su mochila en alguna pared a la cual se encuentre adyacente no diagonal y esta no forme parte del borde del mapa.
Para indicar esto, el usuario tiene que ingresar el siguiente caracter:

- **Utilizar martillo**: 'Z'.

Si el jugador tiene martillos suficientes, se deberá preguntar al usuario la posición respecto al jugador en la que se quiere martillar, es decir arriba del jugador (ingresa caracter 'W'), abajo ('S'), a su derecha ('D') o a su izquierda ('A').
Si hay una pared (no perteneciente al borde) en la posición indicada, se debe borrar la pared del vector de paredes y restar un martillo.

3.2.4. Extintores

El jugador podrá utilizar un extintor de su mochila en algún fuego al cual se encuentre adyacente a su izquierda, adyacente a su derecha o adyacente arriba.
Para indicar esto, el usuario tiene que ingresar el siguiente caracter:

- **Utilizar extintor**: 'C'.

Si el jugador tiene extintores suficientes, se deberá preguntar al usuario la posición respecto al j en jugador a que se quiere extinguir el fuego, es decir arriba del jugador (ingresa caracter 'W'), a su derecha ('D') o a su izquierda ('A').
Si hay un fuego en la posición indicada, se debe borrar el fuego del vector de obstáculos y restar un extintor.

3.3. Interacción Jugador-Objeto

A continuación se listarán las distintas interacciones que tendrá el jugador con los demás objetos del juego. Es importante aclarar que el jugador 'choca' con otro elemento cuando se mueve a su posición o pasa sobre la misma.

3.3.1. Fuegos

Si el jugador choca con un fuego pierde todos sus movimientos, es decir pierde el juego.

3.3.2. Medias

Si el jugador choca con una media, pierde 10 movimientos.

3.3.3. Interruptor 'ahuyenta Randall'

Si el jugador choca con el interruptor lo activa si este se encuentra desactivado o lo desactiva en caso contrario.

3.3.4. Botellas de gritos

Si el jugador choca con una botella de gritos, gana 7 movimientos.

3.3.5. Papeleos

Si el jugador choca con un papeleo, podrá recolectarlo si y solo si los papeleos con 'id_papeleo' menor a este fueron recolectados.

Si esto se cumple, se debe borrar lógicamente el papeleo del vector de papeleos utilizando el campo 'recolectado'.

Observación: Las medias, interruptores y botellas de gritos no se eliminan del vector una vez que el jugador se choca con algunos de estos. En el caso de los fuegos no se tendrá que eliminar (a menos que se los apague con un extintor), ya que terminará el juego si el jugador se choca con un fuego,

3.4. Nuevos Obstáculos

Se introducirán acciones a medida que el jugador va utilizando movimientos (laterales o rotaciones) que representarán nuevos obstáculos:

3.4.1. Randall

Si el interruptor 'ahuyenta Randall' no se encuentra activado, un papeleo (no recolectado) aleatorio se transportará a otra posición aleatoria, válida y libre si la cantidad de movimientos realizados por el jugador es divisible por R, siendo R un valor distinto por nivel. Decimos que esto lo hace nuestro enemigo Randall quien, cómo todo camaleón, tiene la habilidad de camuflarse para no ser percibido.

Valor divisible de R por nivel

- Nivel 1: 7
- Nivel 2: 5
- Nivel 3: 3

3.4.2. Nuevas paredes random

Durante los primeros X movimientos del jugador en el respectivo nivel, aparecerá una nueva pared en una posición aleatoria, válida y libre.

Cantidad de paredes random por nivel

- Nivel 1: 40
- Nivel 2: 30
- Nivel 3: 20

3.5. Objetivo del juego

El juego consiste en atravesar tres niveles, donde cada uno de estos se gana si se recolectan los papeleos en el orden establecido.

El jugador acumulará solamente los movimientos ahorrados de nivel en nivel.

4. Especificaciones

Para poder lograr ayudar a Mike con su papeleo, se te pedirá implementar nuevas funciones y procedimientos.

4.1. Funciones y procedimientos

```c
#ifndef __PAPELEO_H__
#define __PAPELEO_H__

#include "utiles.h"
#include <stdbool.h>

#define MAX_OBSTACULOS 100
#define MAX_HERRAMIENTAS 100
```

```c
#define MAX_PAPELEOS 10
#define MAX_NIVELES 3

typedef struct papeleo{
    coordenada_t posicion;
    int id_papeleo;
    bool recolectado;
} papeleo_t;

typedef struct objeto{
    coordenada_t posicion;
    char tipo;
} objeto_t;

typedef struct nivel{
    coordenada_t paredes[MAX_PAREDES];
    int tope_paredes;
    objeto_t obstaculos[MAX_OBSTACULOS];
    int tope_obstaculos;
    objeto_t herramientas[MAX_HERRAMIENTAS];
    int tope_herramientas;
    papeleo_t papeleos[MAX_PAPELEOS];
    int tope_papeleos;
    coordenada_t pos_inicial_jugador;
}nivel_t;

typedef struct jugador{
    coordenada_t posicion;
    int movimientos;
    int martillos;
    int extintores;
    bool ahuyenta_randall;
    int movimientos_realizados;
}jugador_t;

typedef struct juego{
    int nivel_actual;
    nivel_t niveles[MAX_NIVELES];
    jugador_t jugador;
    char personaje_tp1;
}juego_t;

/*
 * Procedimiento que recibe el juego e imprime toda su información por pantalla.
 */
void imprimir_terreno(juego_t juego);


/*
 * Inicializará el juego, cargando toda la información inicial, los datos del jugador,
 * el personaje resultado del tp anterior, y los 3 niveles. El campo "nivel_actual"
 * comienza en 1.
 */
void inicializar_juego(juego_t* juego, char personaje_tp1);


/*
 * El nivel se dará por terminado, si se reolectan todos los papeleos en el mismo.
 * Devolverá:
 * -> 0 si el estado es jugando.
 * -> 1 si el estado es ganado.
 */
int estado_nivel(papeleo_t papeleos[MAX_PAPELEOS], int tope_papeleos);

/*
 * Recibe un juego con todas sus estructuras válidas.
 *
 * El juego se dará por ganado, si terminó todos los niveles. O perdido, si el personaje queda
 * sin movimientos.
 * Devolverá:
 * -> 0 si el estado es jugando.
 * -> -1 si el estado es perdido.
 * -> 1 si el estado es ganado.
 */
int estado_juego(juego_t juego);

```

```c
/*
 * Moverá el personaje, y realizará la acción necesaria en caso de chocar con un elemento
 */
void realizar_jugada(juego_t* juego);

#endif /* __PAPELEO_H__ */
```

Observación: Queda a criterio del alumno/a el hacer o no, más funciones y/o procedimientos para resolver los problemas presentados. No se permite agregar dichas firmas al .h.

4.2. Modificación en el registro del Jugador

Como se puede observar en el papeleo.h, el struct jugador contará con un nuevo campo 'movimientos_realizados' el cuál se inicializará en 0 al empezar cada nivel. El mismo aumentará en 1 cuando el jugador pueda realizar un movimiento lateral o rotacional.

```c
typedef struct jugador{
    coordenada_t posicion;
    int movimientos;
    int martillos;
    int extintores;
    bool ahuyenta_randall;
    int movimientos_realizados;
}jugador_t;
```

4.3. Convenciones

Se deberá utilizar la siguiente convención para los elementos que tengan una coordenada asociada como obstáculos, algunas herramientas y para el jugador, y para el ingreso de acciones por el usuario:

4.3.1. Convención para identificadores de elementos

- **Fuegos**: F.
- **Medias**: M.
- **Botellas de gritos**: G.
- **Interruptores 'ahuyenta Randall'**: I.
- **Mike**: W.

4.3.2. Convención para ingreso de acción

- **Utilizar martillo**: Z.
- **Utilizar extintor**: C.
- **Mover o martillar/extinguir a izquierda**: A.
- **Mover o martillar/extinguir a derecha**: D.
- **Martillar/extinguir hacia arriba**: W.
- **Martillar hacia abajo**: S.
- **Mov. rotacional horario**: E.
- **Mov. rotacional antihorario**: Q.

5. Resultado esperado

Se espera que el trabajo creado cumpla con las buenas prácticas de programación y todas las funciones y procedimientos pedidos funcionen acorde a lo solicitado, respetando las pre y post condiciones propuestas.

6. Compilación y Entrega

El trabajo práctico debe ser realizado en un archivo llamado papeleo.c, lo que sería la implementación de la biblioteca papeleo.h. Además, deberán crear una bilioteca propia con lo resuelto en el trabajo práctico 1, y deberá realizarse el cuestionario previo al juego. El trabajo debe ser compilado sin errores al correr desde la terminal el comando:

```
gcc *.c utiles.o -o juego -std=c99 -Wall -Wconversion -Werror -lm
```

Aclaración: El main tiene que estar desarrollado en un archivo juego.c, el cual manejará todo el flujo del programa.

utiles.o es un archivo compilado realizado por la cátedra, que pondrá a su disposición funciones que son necesarias para el armado del juego. Ver anexo.

Lo que nos permite ***.c** es agarrar todos los archivos que tengan extensión .c en el directorio actual y los compile todos juntos. Esto permite que se puedan crear bibliotecas a criterio del alumno, aparte de la exigida por la cátedra.

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** que contenga únicamente los archivos antes mencionados, sin carpetas internas ni otros archivos. De lo contrario, la entrega no será validada por la plataforma.

IMPORTANTE! Obtener la etiqueta **¡Exito!** en **Chanutron2021** (patente pendiente) no implica necesariamente haber aprobado el trabajo. El trabajo será corregido por un colaborador que verificará que se cumplan las buenas prácticas de programación.

7. Anexos

7.1. Rotaciones

7.1.1. Sentido horario

Sea M una matriz de dos dimensiones con **n** filas y **m** columnas. Una coordenada (x,y) perteneciente a M puede ser rotada en sentido horario si se le aplica la función: E(x,y) = (y, |x - n|)

7.1.2. Sentido antihorario

Sea M una matriz de dos dimensiones con **n** filas y **m** columnas. Una coordenada (x,y) perteneciente a M puede ser rotada en sentido antihorario si se le aplica la función: Q(x,y) = (|y - m|, x)

Observación: Cabe aclarar que estas funciones matemáticas están pensadas de forma general, queda a cargo del alumno adaptarlas al código, recordando como se manejan los arreglos en C.

7.2. Cálculo de valor absoluto

Para obtener el valor absoluto de un valor se puede utilizar la función **abs**.

```c
#include <stdio.h>
#include <stdlib.h> //Para poder usar abs

int main(){
    int valor = -5;
    int valor_absoluto = abs(valor); //se obtiene como retorno un 5

    return 0;
}
```

7.3. 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 mismos.

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 esto, 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 29 (inclusives).

```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)); // Genera la semilla aleatoria.
        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;
}
```

7.4. Distancia Manhattan

Para obtener la distancia entre 2 puntos mediante este método, se debe conocer a priori las coordenadas de dichos puntos.

Luego, la distancia entre ellos es la suma de los valores absolutos de las diferencias de las coordenadas. Se ve claramente en los siguientes ejemplos:

- La distancia entre los puntos (0,0) y (1,1) es 2 ya que: $|0 - 1| + |0 - 1| = 1 + 1 = 2$
- La distancia entre los puntos (10,5) y (2,12) es 15 ya que: $|10 - 2| + |5 - 12| = 8 + 7 = 15$
- La distancia entre los puntos (7,8) y (9,8) es 2 ya que: $|7 - 9| + |8 - 8| = 2 + 0 = 2$

Observación: En este TP no están obligados a usar la distancia manhattan, sino que sólo se menciona como una posible herramienta.

7.5. 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;
}
```

7.6. utiles.h

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.

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

```c
#ifndef __UTILES_H__
#define __UTILES_H__

#define MAX_PAREDES 500

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


/* Pre condiciones: El parametro nivel debe contener el valor de un nivel (1 a 3 inclusives).
 * Post condiciones: Devuelve el vector de coordenadas de las paredes cargado, junto a su respectivo
     tope, las pos inicial del jugador.
*/
void obtener_paredes(int nivel, coordenada_t paredes[MAX_PAREDES], int* tope_paredes, coordenada_t*
    pos_inicial_jugador);


/* 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.

7.7. controlador.h

Esta es otra biblioteca brindada por la cátedra, cabe aclarar que su uso es **opcional**.

Nos permite poder ingresar caracteres en la terminal sin tener que tocar la tecla 'enter', lo que hace que termine siendo más dinámico el jugar.

El .h de esta biblioteca tiene la forma:

```c
#ifndef __CONTROLADOR__
#define __CONTROLADOR__

#include <stdio.h>
#include <termios.h>
#include <unistd.h>

/*
 * Pre:
 * Post: Cambia la configuración de la terminal para automatizar la
 *        entrada del teclado.
 */
void inicializar_controlador();

/*
 * Pre:
 * Post: Devuelve la terminal a la configuración original.
 */
void terminar_controlador();

#endif /* __CONTROLADOR__ */
```

IMPORTANTE: El uso de esta biblioteca tiene que ser en el main del archivo juego.c. Si no se cumple con esto puede que genere problemas en **Chanutron2021** (patente pendiente) a la hora de correr las pruebas.

A continuación se muestra como se debe usar:

```c
#include "controlador.h"

int main(){
        inicializar_controlador();

        // Todos los scanf que se llamen entre estos dos llamados se ingresarán automáticamente

        terminar_controlador();
        return 0;
}
```

Es decir, se debe tener un único llamado a **inicializar_controlador** al empezar el main, y un único llamado a **terminar_controlador** al terminar el main.