.UBAfiuba
FACULTAD DE INGENIERÍA

FUNDAMENTOS DE PROGRAMACIÓN - CURSO MENDEZ

# Trabajo Práctico Nº 1

## Ratatouille Rush (Parte 1)

| | |
|---|---|
| Fecha presentación | 12/09/2024 |
| Fecha entrega | 03/10/2024 |

Segundo cuatrimestre 2024

# 1. Introducción

Seguimos explorando el universo de Ratatuille, esta vez desarrollando un papel más importante, porque te pone en los zapatos del mozo principal del famoso restaurante de Remy. Tu trabajo es simple, pero intenso: atender a los clientes, tomar pedidos, servir platos y, por supuesto, limpiar todo el desastre.

Cada noche se tiene un objetivo de dinero que alcanzar. Cuanto más rápido y mejor atiendas, más cerca vas a estar de llegar a la meta. Pero ojo, porque si los clientes se empiezan a impacientar o el lugar se ensucia demasiado, te podrías perder la oportunidad de lograrlo.

El restaurante se va llenando cada vez más, así que preparate para correr de acá para allá. ¿Vas a aguantar el ritmo y cumplir con todo antes de que cierre la cocina?

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

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

# 3. Enunciado

Para esta primera parte del desarrollo del juego, se requerirá la **inicialización** de todos los elementos en el terreno, la **visualización** del mismo, y la **implementación del movimiento del personaje principal**.

Cómo se juega el juego y qué hace cada elemento lo vamos a ver más adelante con el segundo TP de la materia en otro momento del cuatrimestre.

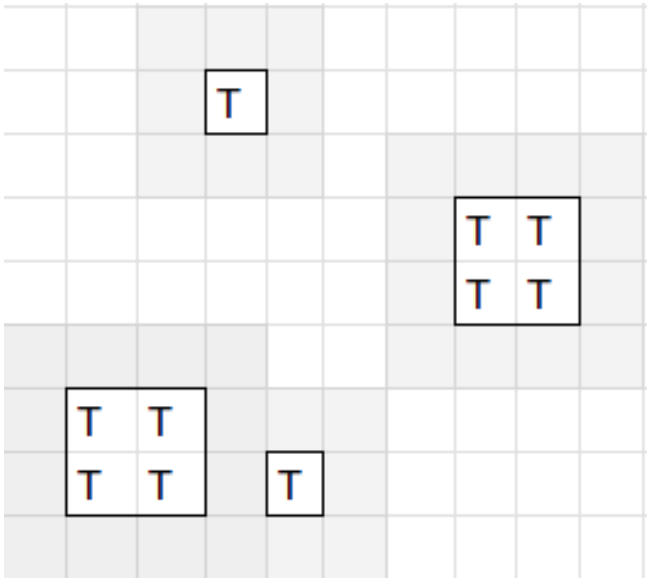
Lo importante que hay que saber del juego en esta primera parte es que el mismo consta en un terreno (el restaurante) donde estarán dispersas la cocina, Linguini (nuestro personaje principal), las mesas, los charcos, la mopa, las monedas y los patines. Se hablará más de estos elementos más adelante.

Para inicializar, primero deberán ubicarse las mesas, la cocina, luego Linguini, la mopa, las monedas, los patines y los charcos.

***IMPORTANTE!*** **Ningún elemento puede inicializarse por fuera de los límites del terreno ni pisar otros elementos ya inicializados**.

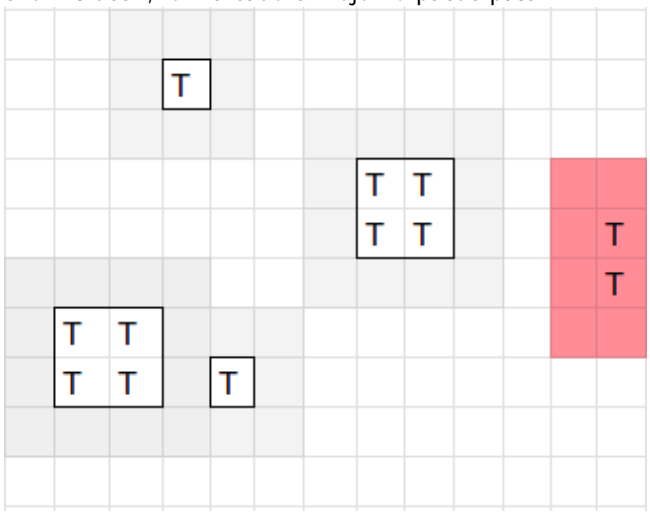
## 3.1. Mesas

Habrá 10 mesas de distintos tamaños distribuidas por el terreno de forma aleatoria. Las mesas pueden ser de 1 o 4 comensales, se tendrán 4 mesas de 4 y 6 mesas de 1. **Aclaración**: las mesas no pueden estar pegadas, es decir debe haber un espacio alrededor de cada una como se muestra en el siguiente gráfico (lo gris es para mostrar el espacio entre las mesas):




*Esto no significa que no pueda haber elementos alrededor de las mesas, simplemente significa que al poner una mesa en el terreno hay que asegurarse que la misma no quede pegada a ninguna otra que haya en el terreno anteriormente.*

Además, las mesas tienen que inicializarse enteras, no puede haber una mesa de 4 comensales que no entre en el terreno. Es decir, lo marcado en rojo no puede pasar:




En caso que ocurra lo marcado anteriormente (que la mesa no entra entera en el terreno), se deberá poner la mesa en otra posición aleatoria en la que sí entre completa.

## 3.2. Personaje

Linguini deberá posicionarse de forma aleatoria en el terreno.

## 3.3. Cocina

La cocina deberá posicionarse de forma aleatoria en el terreno.

## 3.4. Herramientas

### 3.4.1. Monedas

Se tendrán 8 monedas dispersas por el terreno de forma aleatoria.

### 3.4.2. Mopa

Habrá una mopa que se debe inicializar en el terreno de forma aleatoria.

#### 3.4.3. Patines

Se tendrán 5 patines dispersos por el terreno de forma aleatoria.

### 3.5. Obstáculos

#### 3.5.1. Charcos

Se tendrán 5 charcos dispersas en el terreno de forma aleatoria.

#### 3.5.2. Cucarachas

No habrá ninguna cucaracha al principio del juego.

### 3.6. Terreno

El terreno será representado con una matriz de 20x20, donde estarán dispersos todos los elementos mencionados anteriormente.

### 3.7. Modo de juego

Al moverse, Linguini no puede pasarse de los límites del terreno ni puede caminar por las mesas. Por ejemplo, si Linguini está en la fila 0 y el usuario lo mueve para arriba, Linguini debería quedarse ahí, no se debería mover porque estaría saliéndose del terreno. Lo mismo si se choca con una mesa, no debería poder moverse.

El personaje se podrá mover en 4 direcciones:

- **Arriba**: W
- **Abajo**: S
- **Derecha**: D
- **Izquierda**: A

Además, deberá poder levantar y dejar la mopa en el terreno con la siguiente instrucción:

- **Mopa**: O

Como aclaración, no se puede dejar la mopa en un lugar que coincida con algún otro elemento del terreno.

## 4. Especificaciones

### 4.1. Convenciones

- **Linguini**: L.
- **Mesas**: T.
- **Cocina**: C.

Se deberá utilizar la siguiente convención para los obstáculos:

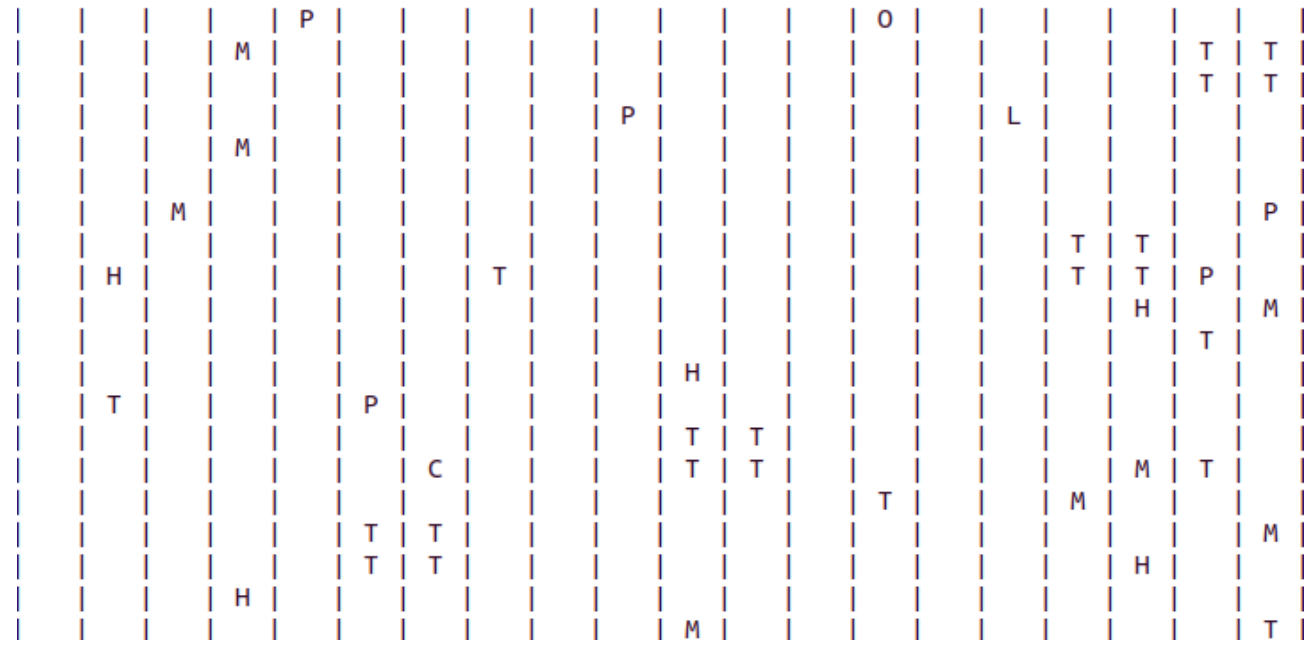
- **Charcos**: H.

Para las herramientas:

- **Mopa**: O.
- **Monedas**: M.
- **Patines**: P.

### 4.2. Ejemplo

A continuación hay un ejemplo de cómo debería verse el terreno luego de haber inicializado todos los elementos. Vale aclarar que los elementos se van a colocar aleatoriamente así que no todos los terrenos van a verse igual. Además, la forma de mostrar los diferentes elementos queda a criterio del alumno. Por ejemplo, en este caso se agregaron | para separar las diferentes columnas.



## 4.3. Funciones y procedimientos

Para poder cumplir con lo pedido, se pedirá implementar las siguientes funciones y procedimientos.

```c
#ifndef __RESTAURANT_H__
#define __RESTAURANT_H__


#include <stdbool.h>

#define MAX_COMENSALES 4
#define MAX_POSICION 9
#define MAX_DESCRIPCION 50
#define MAX_PLATOS 10
#define MAX_PEDIDOS 10
#define MAX_BANDEJA 10
#define MAX_HERRAMIENTAS 20
#define MAX_OBSTACULOS 20
#define MAX_MESAS 20

#define MAX_FILAS 20
#define MAX_COLUMNAS 20

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

typedef struct pedido {
    int id_mesa;
    char platos[MAX_PLATOS];
    int cantidad_platos;
    int tiempo_preparacion;
} pedido_t;

typedef struct cocina {
    coordenada_t posicion;
} cocina_t;

typedef struct mozo {
    coordenada_t posicion;
    int cantidad_patines;
    pedido_t pedidos[MAX_PEDIDOS];
    int cantidad_pedidos;
    pedido_t bandeja[MAX_BANDEJA];
    int cantidad_bandeja;
    bool tiene_mopa;
    bool patines_puestos;
} mozo_t;

```

```c
typedef struct mesa {
    coordenada_t posicion[MAX_POSICION];
    int cantidad_lugares;
    int cantidad_comensales;
    int paciencia;
    bool pedido_tomado;
} mesa_t;

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

typedef struct juego {
    cocina_t cocina;
    mozo_t mozo;
    mesa_t mesas[MAX_MESAS];
    int cantidad_mesas;
    objeto_t herramientas[MAX_HERRAMIENTAS];
    int cantidad_herramientas;
    objeto_t obstaculos[MAX_OBSTACULOS];
    int cantidad_obstaculos;
    int movimientos;
    int dinero;
} juego_t;

/*
 * Pre condiciones: -
 * Post condiciones: Inicializará el juego, cargando toda la información inicial de Linguini, las
 * mesas, las herramientas y los obstáculos.
 */
void inicializar_juego(juego_t *juego);

/*
 * Pre condiciones: El juego debe estar inicializado previamente con `inicializar_juego` y la acción
 * debe ser válida.
 * Post condiciones: Realizará la acción recibida por parámetro. Para este primer TP solo se
 * implementará el funcionamiento para mover al jugador y agarrar/soltar la mopa.
 */
void realizar_jugada(juego_t *juego, char accion);

/*
 * Pre condiciones: El juego debe estar inicializado previamente con `inicializar_juego`.
 * Post condiciones: Imprime el juego por pantalla.
 */
void mostrar_juego(juego_t juego);

/*
 * Pre condiciones: El juego deberá estar inicializado previamente con `inicializar_juego`
 * Post condiciones: Devuelve:
 *      --> 1 si es ganado
 *      --> -1 si es perdido
 *      --> 0 si se sigue jugando
 *  El juego se dará por ganado cuando se termina el día y se consiguieron las monedas necesarias.
 *  Se dará por perdido si se termina el día y no se llegó al monto.
 */
int estado_juego(juego_t juego);

#endif /* __RESTAURANT_H__ */
```

**Observación:** Queda a criterio del alumno/a hacer o no más funciones y/o procedimientos para resolver los problemas presentados. No se permite agregar funciones al .h presentado por la cátedra, como tampoco modificar las funciones ni los structs dados.

## 5. Resultado esperado

Este TP es la primera parte de un juego más complejo que van a terminar de implementar en la segunda parte. La idea es tener una base de la cual se podrá construir un juego más divertido. Aún así esperamos que el trabajo tenga una funcionalidad mínima obligatoria que cumpla con las especificaciones mencionadas anteriormente. Se deberá:

- Implementar todas las funciones especificadas en la biblioteca. Algunas todavía no podrán ser probadas dentro del juego (como estado_juego) pero su funcionamiento debería ser correcto de todas formas.
- Inicializar **todos** los campos del registro juego_t.

- Pedirle al usuario que ingrese una acción **válida** a realizar cada turno. Por el momento las acciones son moverse y agarrar/soltar la mopa. También se tiene que validar que Linguini no pueda subirse a las mesas ni salir del terreno.
- Imprimir el terreno de forma clara con información que pueda serle útil al usuario (cuanto dinero tiene, los pedidos, etc)
- Respetar las buenas prácticas de programación que profesamos en la cátedra.

# 6. Compilación y Entrega

El trabajo práctico debe ser realizado en un archivo llamado restaurant.c, lo que sería la implementación de la biblioteca restaurant.h. El trabajo debe ser compilado sin errores al correr el siguiente comando:

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

**Aclaración:** Además del desarrollo de la biblioteca, se deberá implementar un main, el cual tiene que estar desarrollado en un archivo llamado juego.c. En este se manejará todo el flujo del programa, utilizando las funciones de la biblioteca realizada.

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

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

**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***.

Para la entrega en **AlgoTrón** (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.

# 7. Anexos

## 7.1. FAQ

En este link encontrarán el documento de FAQ del TP, donde se irán cargando dudas realizadas con sus respuestas. Todo lo que esté en ese documento es válido y oficial para la realización del TP.

## 7.2. 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 dónde 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 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;
}
```

## 7.3. 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 esto, 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;
}
```

## 7.4. 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: $|7 - 9| + |8 - 8| = 2 + 0 = 2$