.UBAfiuba
FACULTAD DE INGENIERÍA

FUNDAMENTOS DE PROGRAMACIÓN - CURSO MENDEZ

Trabajo Práctico Nº 2

Ratatouille Rush (Parte 2)

Fecha presentación	24/10/2024
Fecha entrega	14/11/2024

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 restaurant 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 restaurant 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 familiarice con y utilice correctamente memoria dinámica y punteros.
- 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 y claridad del código.

3. Enunciado

*En **GRIS** encontrarán lo que ya fue realizado en la primera parte del trabajo. En **NEGRO** es lo que van a tener que realizar y tener en cuenta para esta segunda parte.*

En la primera parte del desarrollo del juego, se realizó la inicialización de los elementos en el terreno, la visualización del mismo, la implementación del movimiento del personaje principal, Linguini, y también la habilidad del personaje de agarrar y soltar la mopa.

En esta segunda parte del desarrollo del juego, el objetivo es que el mismo sea funcional, dándole el correspondiente comportamiento a cada elemento y se pueda jugar. Además se requerirá la correcta inicialización de los nuevos objetos y visualización del juego.

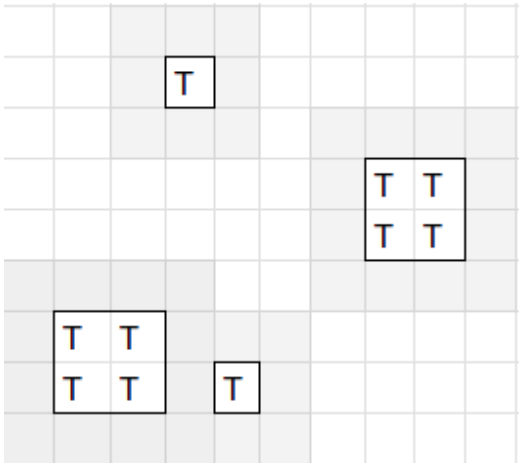
El juego consiste en un día en el restaurant. Los comensales van llegando y se sentarán en las mesas disponibles, el mozo debe acercarse a cada mesa para tomar los pedidos de las mismas, y llevar esos pedidos hasta la cocina para que se preparen. Luego de que haya pasado tiempo de preparación de cada pedido, el mozo deberá ir a buscarlos y llevarlos a la mesa correspondiente. En el medio, el mozo deberá tener cuidado con los charcos, limpiándolos de ser necesario, y deberá estar atento de matar a las cucarachas que puedan aparecer para que no moleste a sus comensales.

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**.

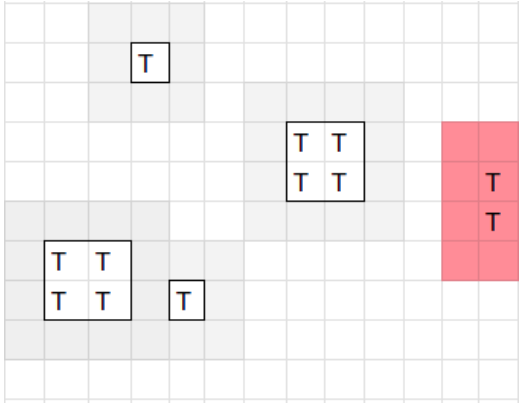
3.1. Mesas

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.

Al restaurant llegará una cantidad aleatoria de comensales (entre 1 y 4 inclusive) cada 15 movimientos, y se ubicarán en la primera mesa que encuentren vacía y con la cantidad de lugares suficientes para todos esos clientes. Si no se encuentra ninguna mesa disponible para esa cantidad de comensales, se irán y no se los ubica en ninguna mesa.

Cada mesa recién ocupada iniciará con una cantidad aleatoria de paciencia entre 100 y 200. Por cada movimiento válido que haga el mozo (que efectivamente haya cambiado de posición), se disminuirá en 1 la paciencia de las mesas ocupadas. Cuando la paciencia de una mesa llegue a 0, la misma se desocupa perdiendo los clientes.

Cuando el mozo le entrega a la mesa su pedido listo, la misma se desocupa y deberá pagar:

- 20000 si es una mesa de 4 lugares.
- 5000 si es una mesa de 1 lugar.

3.2. Mozo

Linguini deberá posicionarse de forma aleatoria en el terreno.

El mozo tendrá los pedidos de las mesas que deberá llevar a la cocina para que los preparen, también deberá ir a la cocina a buscar los pedidos ya listos y llevarlos a la mesa correspondiente.

- Para tomar el pedido de los comensales de una mesa el mozo debe encontrase a distancia manhattan 1 de cualquier coordenada que compone esa mesa e ingresar la acción correspondiente.
- Para pasar los pedidos a la cocina y convertirlos en platos en preparación, el mozo deberá posicionarse sobre la misma.
- Para pasar los platos listos de la cocina y convertirlos en platos en la bandeja del mozo, este deberá posicionarse sobre la cocina. En este proceso, se deberá liberar la memoria reservada para el plato listo.
- Para entregar un plato de la bandeja a su correspondiente mesa, el mozo deberá encontrase a distancia manhattan 1 de cualquier coordenada que compone esa mesa.

3.3. Pedidos

Los pedidos que hacen las mesas se definen de modo aleatorio para cada comensal seleccionando un número entre 1 y 4 (ambos inclusive), y están limitados a este menú:

- **Milanesa napolitana**: Se representa con el número 1. Tiene un tiempo de preparación de 30 movimientos.
- **Hamburguesa**: Se representa con el número 2. Tiene un tiempo de preparación de 15 movimientos.
- **Parrilla**: Se representa con el número 3. Tiene un tiempo de preparación de 20 movimientos.
- **Ratatouille**: Se representa con el número 4. Tiene un tiempo de preparación de 25 movimientos.

El tiempo de preparación del pedido de una mesa será el máximo tiempo de preparación entre los platos de los comensales.

3.4. Cocina

La cocina deberá posicionarse de forma aleatoria en el terreno.

La cocina va a ser el lugar al cual el mozo deberá llevar los pedidos de las mesas para que hagan los platos que pidieron los clientes. Para esto, la cocina guardará en un **vector dinámico**. Esto quiere decir que cada vez que se cree un plato en preparación se deberá reservar la memoria necesaria para el mismo.

Una vez terminado el plato (cada uno tendrá un tiempo de preparación), la cocina cambiará de lugar el plato del vector de platos en preparación al vector de platos listos, el cual también deberá ser representado por un vector dinámico. En este proceso, se deberá liberar la memoria reservada para el plato en preparación y reservar nueva memoria para el plato listo.

Una vez hecho un plato, el mozo deberá acercarse a la cocina, poner los platos en su bandeja y luego llevarlos a las mesas.

3.5. Herramientas

3.5.1. Monedas

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

Las monedas le darán al mozo un monto de 1000 de dinero extra. Para recolectar una moneda y sumar dinero Linguini deberá ubicarse encima de la misma. Al consumirse las monedas deben eliminarse físicamente del vector.

3.5.2. Mopa

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

Esta mopa permitirá que el mozo, cuando tenga la mopa en su posesión, pueda limpiar los charcos. Si el mozo tiene la mopa podrá limpiar charcos pero no podrá realizar ninguna otra acción (tomar pedidos a las mesas, tomar pedidos listos de la cocina, dejar los pedidos para que se preparen en la cocina, matar cucarachas).

Al tomarla se debe eliminar físicamente del vector. Al soltarla se debe insertar físicamente en el vector.

3.5.3. Patines

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

Para recolectar los patines, el mozo deberá ubicarse encima del mismo. Cuando sean activados por el usuario, el mozo tendrá patines que le permitirán que su próximo movimiento sea hasta que se choque con una pared o con una mesa. Es decir que con un sólo movimiento podrá moverse toda una fila o toda una columna (a no ser que haya una mesa antes). El mozo va a interactuar con todos los elementos que se cruce durante ese trayecto. Al consumirse los patines, deben eliminarse físicamente del vector.

3.6. Obstáculos

3.6.1. Charcos

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

Los charcos ocasionarán que si el mozo pasa por arriba de alguno de ellos, se le caiga todo el pedido que tiene en la bandeja, y por lo tanto se libere la mesa que había hecho ese pedido, perdiendo los clientes. Si la bandeja del mozo está vacía, el charco no tiene ningún efecto.

El charco sólo desaparece si el mozo pasa por arriba con la mopa (en ese caso no se le cae la bandeja). Cuando se limpia un charco, este desaparece físicamente del vector.

3.6.2. Cucarachas

No habrá ninguna cucaracha al principio del juego.

Las cucarachas aparecerán en una posición aleatoria libre del terreno cada 25 movimientos. Para matar una cucaracha, el mozo debe pasar por arriba de la misma. Cuando se mata a una cucaracha, esta desaparece físicamente del vector. Si una mesa tiene una cucaracha a distancia manhattan menor o igual a 2, le quita 2 unidades extra de paciencia a esa mesa por cada movimiento.

3.7. Terreno

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

3.8. 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 mozo 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.

Para activar los patines, se deberá ingresar la siguiente instrucción:

- **Patines**: P.

Para tomar un pedido, se deberá ingresar la siguiente instrucción:

- **Tomar pedido**: T.

Luego de realizar una acción en caso de chocar o estar a la distancia de reacción con un elemento se activará la reacción relacionada al mismo que afectará al mozo y el estado del juego.

El juego finaliza al llegar a los 200 movimientos:

- Se habrá ganado si el dinero es mayor o igual a 150.000.
- Se habrá perdido si el dinero es menor a 150.000.

4. Especificaciones

4.1. Convenciones

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

Se deberá utilizar la siguiente convención para los obstáculos:

- **Charcos**: H.
- **Cucarachas**: U.

Para las herramientas:

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

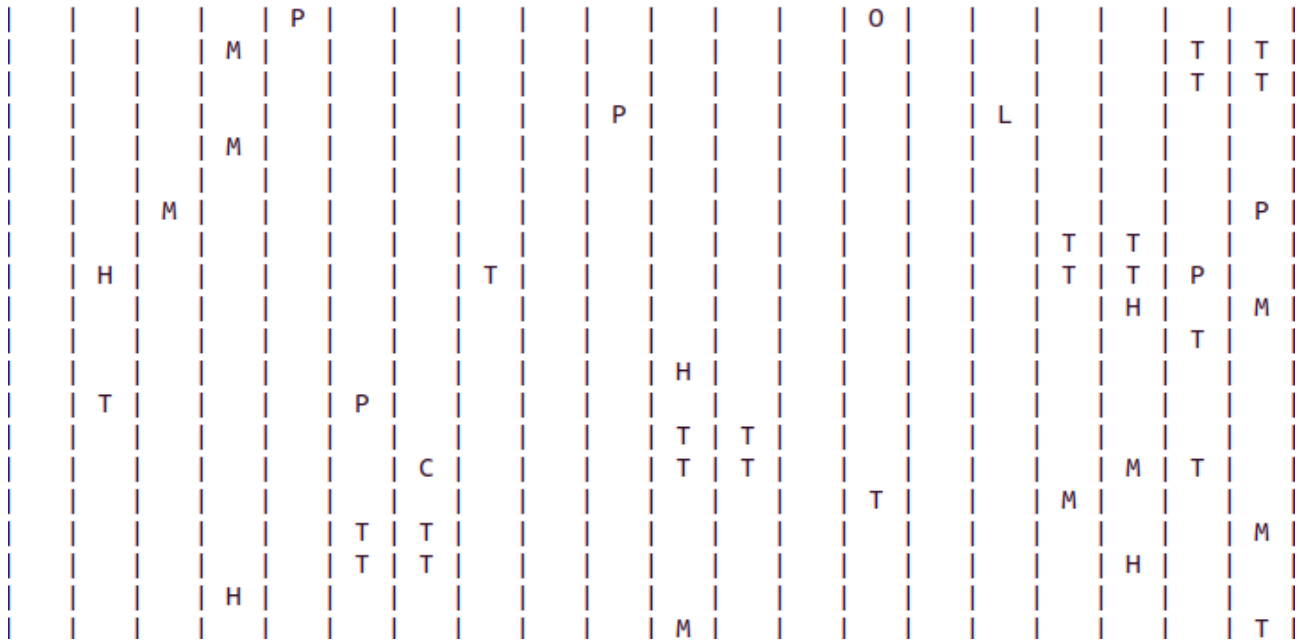
Para los platos:

- **Milanesa napolitana**: M.
- **Hamburguesa**: H.
- **Parrilla**: P.
- **Ratatouille**: R.

Para los **comensales** (cuando las mesas están ocupadas): X.

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 6
#define MAX_BANDEJA 6
#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;
    pedido_t* platos_preparacion;
    int cantidad_preparacion;
    pedido_t* platos_listos;
    int cantidad_listos;
} 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;

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;
```

```c
} 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.
 */
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);

/*  Pre condiciones: Los campos `platos_preparacion` y `platos_listos` del campo `cocina` del juego
    deben estar inicializados.
    Post condiciones: Libera la memoria dinámica reservada para el juego.
*/
void destruir_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 segunda parte del juego que inicializaron en el trabajo anterior. La idea es implementar las funcionalidades para que el juego pueda efectivamente jugarse. Esperamos que el trabajo sea un juego completo y cumpla la funcionalidad mencionada anteriormente. Se deberá:

- Implementar todas las funciones especificadas en la biblioteca.
- Inicializar **todos** los campos del registro juego_t (incluyendo los que se agregaron en esta segunda parte).
- Pedirle al usuario que ingrese una acción **válida** a realizar cada turno.
- 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.
- Usar memoria dinámica correctamente, reservando y liberando la misma sin que haya pérdidas de memoria.
- Que lo pedido en la primera parte del trabajo esté correctamente implementado.

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 que el trabajo sea corregido por un colaborador, el mismo debe pasar todas las pruebas del TP1 y unas pruebas mínimas del TP2.***.

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: $|7 - 9| + |8 - 8| = 2 + 0 = 2$