Informe PPROG: Cubo de Rubik.

Pablo Cuesta Sierra.

Nombre del grupo: **is_prime()**.

Componentes del grupo: Álvaro Zamanillo, Ignacio Bernardez y Pablo Cuesta.

1. Descripción del proyecto

Este proyecto consiste de dos interfaces distintas (una en la propia terminal y otra que hace uso de una ventana renderizada con una biblioteca de SDL) con las mismas funcionalidades: jugar con un cubo de Rubik (3x3x3 ó 2x2x2).

En el juego se le proporciona al usuario una serie de controles del teclado con los que se puede mover el cubo, desordenarlo aleatoriamente[1], obtener la solución del cubo obtenida mediante un algoritmo de resolución (así como ver la resolución proporcionada por dicho algoritmo) y utilizar un cronómetro con el que se puede medir el tiempo que se tarda en resolver el cubo.

Además, se puede acceder al iniciar el juego al último estado que se tuvo del cubo en la anterior partida, ya que éste se guarda siempre que se cierra el juego.

1.1. Dinámica del juego

Al ejecutar el juego se abre un menú, donde el usuario elige si quiere iniciar una partida nueva, o la partida guardada; así como si prefiere utilizar la interfaz de la terminal, o la implementada con la biblioteca *OpenGL*; además de si utiliza el cubo 3x3x3 o al 2x2x2.

Una vez se hace esta selección, el juego consiste en ver el cubo en la pantalla mientras se hacen los movimientos, se mezcla o se resuelve. En cualquier momento se puede volver al menú anterior para hacer otra selección.

2. Aspectos técnicos relevantes

2.1. Bibliotecas especiales

Para desarrollar este proyecto hemos utilizado una serie de bibliotecas SDL, la más significativa: *OpenGL*, para poder obtener la imagen del cubo de Rubik moviéndose en la pantalla al hacer cada movimiento.

[1] La mezcla se selecciona (pseudo)aleatoriamente de entre una lista de mezclas que se encuentra en un fichero.

Además, hemos usado *pthread.h* (hilos) para la implementación del cronómetro.

Por otro lado, hemos utilizado una función que no es nuestra: `_term_init()`, que modifica los parámetros de la terminal para que las letras que se leen desde `stdin` no se impriman. Esta función es de Simone Santini. La hemos puesto en un fichero aparte (*terminal_funct.c* y *.h*), nombrando al autor. Aunque no estábamos seguros de cómo hacer esta referencia.

2.2. Implementación del cubo de Rubik (3x3x3)

La implementación de la estructura **Cubo**, que se utiliza internamente en el juego para poder manejar la información tiene la siguiente forma:

La estructura **Cubo**, que consta de 26 **Piezas** (un array). Así como la estructura **Pieza** que está constituida por dos vectores de tres coordenadas: *vector posición* y *vector colores*.

El *vector posición* es un vector de tres coordenadas, donde cada una de ellas puede tomar los valores {-1, 0, 1}, indica la posición de la pieza en el espacio. El *vector colores* tiene también tres coordenadas (X, Y, Z) y cada coordenada contiene el color de la pieza que apunta a dicha dirección (X es el color que apunta en la dirección x, etc.). Además, estos colores se han definido en macros de manera que si se suman las tres coordenadas, podemos diferenciar la pieza de todas las demás, facilitando así la búsqueda de piezas.

De este modo es muy sencillo realizar un movimiento: consiste en aplicar una función de rotación a las piezas que nos interesan (por ejemplo, si giramos la capa derecha, R, solo tenemos que aplicar una rotación a las piezas situadas en con *vector posición* en el plano y=1; etc.). De este modo al *vector posición* se le aplica una función de rotación de 90^o y al vector colores se le cambia el orden de forma que las piezas se comporten como en un cubo real.

Luego se implementaron funciones con las que buscar piezas y colores, para posteriormente programar el *solver.h*.

Para el cubo 2x2x2 se ha utilizado esta misma estructura, teniendo en cuenta solamente las esquinas.

3. Dificultades imprevistas

La mayor dificultad que nos hemos encontrado ha sido nuestra inexperiencia en la organización. Deberíamos haber tenido en cuenta antes de ponernos a programar cómo podíamos afrontar las posibles dificultades que nos íbamos a encontrar. Sin embargo, los primeros dos meses fueron un poco caóticos, y estuvimos improvisando sobre la marcha y afrontando con dificultad los problemas que nos iba causando la falta de planificación.

Por otro lado, desde el principio tuvimos dudas sobre si podríamos finalmente obtener la representación del cubo con la biblioteca o no. Cuando conseguimos esto, fue mediante unas funciones muy repetitivas que en su mayoría se podían haber hecho mejor con un poco de planificación.

Tras la primera representación que conseguimos con la biblioteca (un poco fea, ya que no había ninguna línea que delimitara los bordes de las pegatinas, y sin posibilidad de ver la animación de cada movimiento); diseñé una estructura: sCube, sPiece, sSticker. Con esta estructura pudimos simplificar enormemente la implementación de las funciones que imprimen el cubo, así como mejorar su apariencia con mucha facilidad, implementar los movimientos de las piezas y aumentar la eficiencia y la legibilidad del código. Si hubiéramos planificado esta estructura antes, habríamos ahorrado mucho tiempo.

3.1. Solver

El algoritmo que resuelve el cubo es el algoritmo conocido como Método de Principiantes (ver anexo A). Este algoritmo es bastante sencillo y permite dividir el código en distintas partes muy sencillas y que se pueden iterar fácilmente, reduciendo así el trabajo. Se hace distinción entre el cubo 2x2x2 y el 3x3x3, cambiando el orden u omitiendo pasos para el 2x2x2, pero se utilizan las mismas funciones.

En general, estas funciones tienen la misma estructura: se busca con las funciones proporcionadas por *cube.h* una (o varias) piezas concretas y (teniendo en cuenta todos los casos posibles) se ejecuta una cadena de movimientos en el cubo u otra dependiendo del estado. Por la naturaleza de estas funciones, al tener que comprobar todos los casos, resultan un poco difíciles de seguir, ya que hay muchas condiciones (`if`s anidados): creo que esto es inevitable, aunque no muy agradable de leer.

Para simplificar la solución, había que quitar movimientos innecesarios. Por ejemplo: '`Rr`', que hace un movimiento y lo deshace inmediatamente; o '`RRR`', que se puede sustituir por '`r`', etc. El *solver* ejecuta al final una función que se encarga de limpiar la solución de este tipo de cadenas. Tuve un problemilla con `strncat` al programar esta función. Necesitaba eliminar parte de la cadena de la solución, para simplificarla. Para ello, supongamos que quiero eliminar el i-ésimo caracter, intenté hacer lo siguiente:

```
solution[i]='\0';
strncat(solution, solution+i+1, MAX_SOL);
```

Sin embargo, la función `strncat` no hacía lo que se esperaba – supongo que porque estaba copiando una cadena dentro de sí misma – y finalmente, hice la función `concatenate`, que hacía lo que yo pensaba que tenía que hacer `strcat`:

```
void concatenate(char *a, char *b){
    for(; *a != 0; a++);
    for( ; *b != 0 ; *(a++) = *(b++) );
    (*a) = 0;
}
```

3.2. Lentitud al imprimir

Inicialmente, en la interfaz de la terminal, cada vez que se refrescaba el cubo se leía de un fichero la "imagen"del cubo, mientras se iba modificando el color para imprimir el cubo. Esto era aceptable para que el usuario viera sus movimientos al hacerlos. Sin embargo, al ejecutar la solución había problemas de velocidad, ya que no se refrescaba del todo en la pantalla antes de cada movimiento, debido a estar leyendo constantemente del fichero.

Para solucionar esto, al iniciar la partida se carga en la memoria un *buffer* que contiene la imagen (leída desde el fichero), y al refrescar se accede a este *buffer* que está en memoria; no al fichero. De esta forma no hay ningún problema y se carga sin ninguna dificultad la pantalla. Lo mismo hemos tenido que hacer con las letras que imprimen la solución/mezcla, como cada letra se encuentra en un fichero distinto, la cadena se imprimía con mucha lentitud (había que abrir decenas de ficheros). Pero cargar los *buffers* soluciona este problema.

4. Organización del trabajo

En el proyecto hemos tenido cada uno un papel bien definido. Álvaro se ha encargado principalmente de la representación del cubo en la terminal (la impresión de elementos en la teminal: letras, clores, etc.), así como del menú inicial y del cronómetro (los hilos). Ignacio se encargó desde el principio de investigar cómo representar el cubo con la biblioteca. Por último, yo me encargué de diseñar las interfaces que mueven el cubo y permiten manejarlo con las librerías SDL, así como de las funciones que lo resuelven (*solver.h*).

Inicialmente probamos a usar repl.it, pero pronto nos dimos cuenta de sus muchas limitaciones, por lo que casi desde el principio, hemos usado GitHub. Ha sido realmente útil para organizar el trabajo y que cada uno suba con facilidad su parte del trabajo. Además, permite acceder a todas las versiones anteriores y en varias ocasiones esto nos ha resultado de mucha ayuda para recuperar código que habíamos descartado. Sin duda, volvería a utilizar GitHub para el proyecto.

En cuanto a la comunicación con los compañeros, hemos trabajado durante todo el cuatrimestre on-line, haciendo llamadas (Discord) para acordar los aspectos más importantes y WhatsApp para pequeñas dudas.

Personalmente, yo creo que las limitaciones de este año no nos han creado grandes dificultades significativas en cuanto a la organización, ya que hemos tenido muy buena comunicación dentro del grupo. Sin embargo, yo he echado un poco en falta la presencialidad, ya que, en mi opinión, es mucho más productiva una reunión cara a cara que por una llamada, especialmente a la hora de comunicar ideas, y resolver los problemas y dudas que nos surgen. Habría sido de mucha ayuda tener una clase semanal para preguntar las dudas en el laboratorio y poder así tener más información del profesor sobre cómo estamos haciendo el trabajo, ya que es la primera vez que hacemos un proyecto de esta magnitud.

5. Mi papel. Si volviera atrás...

Para esta asignatura habíamos pensado hacer un juego de "usuario contra la máquina", similar al Monopoly. Sin embargo, no nos entusiasmaba para nada la idea y a las dos semanas de empezar propuse a mis compañeros hacer un cubo de Rubik. Yo siempre he tenido bastante interés por este tema y, por tanto, al inicio me dediqué a dirigir el proyecto. La idea inicial era hacer una simple interfaz, donde el usuario podía mover un cubo con el teclado. Finalmente esta idea fue aumentando según vimos que podíamos hacer bastante más: introdujimos las mezclas, la resolución, el cronómetro...

Como ya mencioné antes, al principio hice la interfaz que mueve internamente el cubo (sin la cual no se podía empezar a hacer la impresión del cubo), así como el algoritmo que lo resuelve. Cuando terminé esto me dediqué a ayudar a mis compañeros: simplifiqué con las estructuras ya comentadas la implementación de las funciones de la biblioteca SDL para poder implementar las rotaciones del cubo, e hice *wrappers* para las funciones de la interfaz, para facilitar la legibilidad de *SDL_interface.c* – la interfaz de la biblioteca SDL.

Lo más importante que he aprendido al realizar este proyecto es, sin duda, la importancia de la planificación: no sirve de nada ponerse a programar sin pensar. Si volviera atrás, esto es lo que habría cambiado de nuestro enfoque a la hora de enfrentarnos al proyecto. En particular, como ya digo en la sección 3, habríamos ahorrado mucho tiempo si hubiéramos pensado cómo evitar hacer un código repetitivo, usando tipos abstractos de datos (ADT) desde el principio. A pesar de todo, yo creo que finalmente lo recondujimos bien.

6. La clase en general

La situación en la que hemos tenido que trabajar ha afectado al resultado, aunque no tiene por qué haber sido negativamente. Al fin y al cabo, la forma de trabajar y de encontrar ideas en equipo ha sido distinta. Yo creo que eso ha cambiado la forma en la que enfocamos el proyecto; pero, como ya he comentado, es solo una forma distinta, no necesariamente peor.

Sin embargo, es cierto que en la clase he echado mucho de menos la presencialidad. Esto nos ha impedido resolver con mayor facilidad nuestras dudas. Porque tener tiempo en el laboratorio para preguntar al profesor, buscar errores, etc. es muy útil y este año no hemos podido hacerlo.

El enfoque de las clases de los lunes me ha parecido muy bueno. Debido a la disparidad de los proyectos, lo más útil para todos los grupos era, en mi opinión, tener un tiempo por separado para poder hablar sobre cada proyecto. Y estas clases definitivamente nos han ayudado mucho para consolidar las ideas y confirmar que íbamos por buen camino.

En cuanto a las clases de los jueves, a mí me hubiera gustado que fueran más parecidas a las que tuvimos al principio, que fueron de mucha ayuda: con información sobre cómo organizar el trabajo, errores comunes que solemos cometer, cómo estructurar los ficheros del proyecto y cómo dividir las funciones, cómo

incluir funciones que no son nuestras referenciando al autor, etc.; ya que nunca habíamos hecho un proyecto de esta magnitud y no sabíamos mucho sobre esta clase de temas.

7. Agradecimientos

En esta asignatura, al no tener ninguna evaluación parcial, es un poco más complicado mantener un trabajo constante durante todo el cuatrimestre. Además de compaginarlo con las demás asignaturas, este proyecto requiere un flujo continuo de nuevas ideas, a la par que trabajo para terminarlo a tiempo. Por este motivo, quiero mencionar especialmente el interés de Álvaro Zamanillo, que, constantemente, ha propuesto nuevas funcionalidades del juego (con grandes ideas en la implementación de la interfaz de la terminal) y no ha dejado de mejorarlo hasta el final.

Por último, me gustaría también agradecer la libertad que se nos ha dado en la asignatura. Poder elegir el proyecto al que dedicamos un cuatrimestre es bastante más motivador que estar programando el Juego de La Oca. De este modo hemos podido investigar y aprender mucho por nuestra cuenta, y con muchas más ganas que de la otra forma.

A. Método de principiantes

El método de principiantes, el implementado en este proyecto para resolver el cubo de Rubik 3x3x3, consiste en los siguientes pasos:

1. **Cruz**: consiste en colocar correctamente las cuatro aristas[2] de la cara inferior, formando así una "cruz".

2. **Esquinas de la primera capa**: en este paso se insertan las esquinas[3] de la cara inferior en sus respectivas posiciones, completando así la cara inferior.

3. **Completar la segunda capa**: se insertan las aristas de la segunda capa, completando las dos primeras. Ya solo queda resolver la capa superior.

4. **Orientar las aristas superiores**: se orientan todas las aristas superiores de modo que se forme en la cara superior una cruz del color de esta cara (formada por el centro y las aristas).

5. **Permutación de las aristas superiores**: se resuelven las aristas superiores.

[2] Las aristas son las piezas del cubo que tienen dos pegatinas.
[3] Las esquinas son las piezas del cubo con tres pegatinas.

6. **Permutación de las esquinas superiores**: se mueven las esquinas superiores a sus respectivas posiciones, aunque todavía no estén correctamente orientadas.

7. **Orientación de las esqinas superiores**: se orientan las esquinas de la capa superior, cada una ya en su sitio, para completar el cubo.

Cada uno de estos pasos se ejecuta de forma que no se deshace lo anterior.

En la implementación de este algoritmo para nuestro juego, se hace una copia del cubo que se va a resolver, y esta copia se va moviendo según avanza el algoritmo y se guardan en una cadena los movimientos que llevan a la solución. Cada uno de estos pasos está resuelto en una función distinta.

Para implementar este algoritmo no he utilizado información de ninguna fuente, me he basado simplemente en mis conocimientos del cubo.