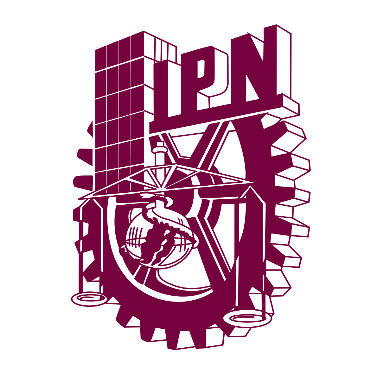
****

**Instituto Politécnico Nacional**

**Escuela Superior de Computo**

*Alumno:*

* *Monroy Ramírez Oscar G.*

***Grupo:*** *2CM23*

***Unidad de Aprendizaje:*** *Algoritmos y Estructuras de Datos*

***Evidencia:*** *Algoritmo de Hanoi, Programa y muestra Grafica.*

***Docente:*** *De Luna Caballero Roberto*

***Fecha****: jueves 22 de octubre de 2020*

Contenido

[Introducción 3](#_Toc55055339)

[Marco Teórico 3](#_Toc55055340)

[Definición de programa recursivo. 4](#_Toc55055341)

[Definición de programa iterativo. 4](#_Toc55055342)

[Diferencias, ventajas y desventajas 4](#_Toc55055343)

[Torres de Hanoi 5](#_Toc55055344)

[Explicacion del Algoritmo de Hanoi en programación. 8](#_Toc55055345)

[Programa 10](#_Toc55055346)

[Ejemplo de Programa de Hanoi. 10](#_Toc55055347)

[Explicación del programa 14](#_Toc55055348)

[Capturas del programa 15](#_Toc55055349)

[Conclusiones 32](#_Toc55055350)

[Referencias Bibliográficas: 33](#_Toc55055351)

# Introducción

En la vida de un programador podemos encontrarnos con problemas que pudiesen parecer en el fondo muy extensos o difíciles, sin embargo, en el fondo muchos son sencillos, si son planeados adecuadamente y se utiliza el tipo correcto de herramientas o método, algo así como tratara de usar para una carreta llantas cuadradas, no facilita el asunto mucho, ¿o no? Otro ejemplo es que al poner soportes, use ruedas para apoyarlos, eso tampoco sería lógico, y con esto no quiero decir que la rueda y los cuadrados sean inútiles, al contrario son tan útiles de acuerdo a sus funciones y capacidades, en todo caso el único que estaría mal, seria al que se le ocurrió semejantes aberraciones, y esto tal vez no suceda cotidianamente, o al menos no con este ejemplo tan burdo, pero siempre se dan casos de usar medios o herramientas inadecuadas para la tarea a realizar, y es ahí donde entra el asunto en cuestión.

Antes de iniciar a programar debemos analizar la situación y lo que debemos hacer, y parte de ese análisis es plantearnos qué tipo de método a programar usaremos, es super importante tener conocimiento de este tipo de herramientas, como la recursividad e iteración.

Tanto la recursión como la iteración se usan para ejecutar algunas instrucciones repetidamente hasta que alguna condición sea verdadera. Pero existe una diferencia importante entre ambas, primero conoceremos el concepto de recursividad.

Recursividad se refiere a una situación en la que una función se llama a sí misma una y otra vez.

Iteración permiten repetir una sentencias o conjunto de ellas.

La recursividad se usa para realizar algoritmos cortos y elegantes con menos código, mientras que la iteración presenta la habitual forma de uno o varios bucles.

**Pero ¿por qué usar la recursividad en algoritmos cortos??**

La recursividad usa la pila por lo tanto facilita las llamadas recursivas. Esto significa que una función es libre de llamarse de nuevo, porque se creará un nuevo stack frame para todas sus variables locales. Entonces al ejecutarse por primera vez la función se guardarán la variable, si la función se llama así misma guardará las variables nuevamente en la pila, junto a las variables anteriores. Si nosotros hacemos que la función se llame miles de veces las variables ocupan demasiada memoria, si queremos ejecutar varias instrucciones repetitivamente debemos de usar una iteración ya que no usa la pila.

# Marco Teórico

## Definición de programa recursivo.

Según Kernighan (1978) es la forma en la cual se especifica un proceso basado en su propia definición. Es evidente que el proceso especificado debe terminar con un resultado definido.

Un problema que pueda ser definido en función de su tamaño, sea este N, puede ser dividido en instancias más pequeñas del mismo problema, aplicándose inducción sobre las más pequeñas para resolver el problema en general.

## Definición de programa iterativo.

Según Kernighan (1978) en ciclo iterativo es la repetición de operaciones hasta que se cumple una condición. Las operaciones o instrucciones son las mismas pero los datos que se procesan pueden cambiar en la ejecución del ciclo.

En general, los ciclos deben terminar después de un cierto número (finito) de repeticiones, y el conjunto de operaciones a repetir se conocen como bucle.

En ocasiones, sabemos el número de veces que el ciclo se repetirá, en otras, no sabemos a ciencia cierta cuántas repeticiones se harán.

Al igual que en las estructuras de control selectivas, se requiere definir la condición con la que se evaluará y decidirá qué acciones se llevarán a cabo. Recordemos que para construir las condiciones es necesario utilizar los operadores relacionales (de comparación) y los operadores booleanos (lógicos).

## Diferencias, ventajas y desventajas

En **programación**, las operaciones pueden ser manejadas por **recursividad** o **iteraciones**. **Recursividad** es cuando utilizamos un método que se llama a sí mismo, directa o indirectamente. Una de las estructuras fundamentales de programación son los bucles, las cuales forman parte de las estructuras de la mayoría de los lenguajes. (ThemeGrill, 2013)

Por ejemplo, existen dos estrategias para calcular 1+2+3+…+n. La primera estrategia es utilizando recursividad:

|  |
| --- |
| **public** static int **sumaN\_recursivo**(int n)**{** return (n<=0)?0:n + sumaN\_recursivo(n-1); **}** |

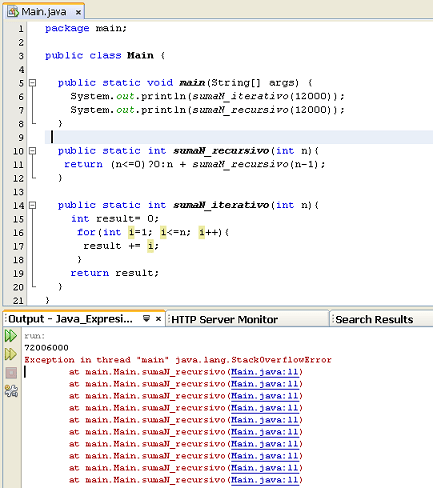
Luego, para iteración el método serio:

|  |
| --- |
| **public** static int **sumaN\_iterativo**(int n)**{** int result= 0; for(int i=1; i<=n; i++){ result +=i;} return result; **}** |

Usualmente, el código con **recursividad** es más consistente que el código con **iteración**. Solo hay una sentencia en el método de **recursividad**, y existe más de una sentencia en el método con **iteraciones**. Adicionalmente, los bucles, que incluyen varios bucles son complejos de leer y entender. La recursividad es más clara y simple para dividir problemas complejos en piezas manejables.

Pero, el uso de la **recursividad** presenta ciertas desventajas. Debido a que en la **recursividad** una función se llama a sí misma, involucra tiempo y espacio: debe utilizar mucha memoria de la pila de datos para trabajar los resultados. Por lo tanto, la solución recursiva para calcular 1+2+..+n no es tan eficiente como la solución **iterativa**.

El mayor problema que presenta la recursividad es que causa un error denominado: stack overflow, como se puede ver en la figura inferior. Al utilizar ambos métodos, en mi máquina, al realizar el cálculo para n=12 000, el método que utiliza iteraciones realizo el cálculo, mientras que el método que utiliza recursividad presento el error:

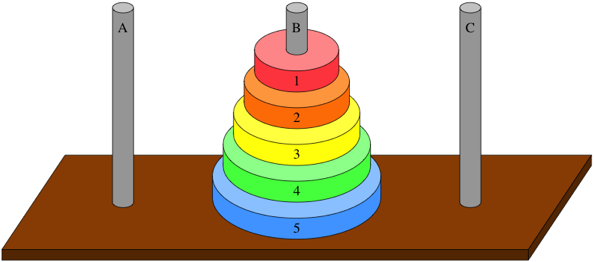
[](https://i1.wp.com/www.clubdetecnologia.net/wp-content/uploads/2013/03/java_recursividad_vs_iteracion-1.png)ThemeGrill, (2013) considera que, finalmente tenemos dos alternativas para resolver un problema. Lo adecuado es saber las ventajas y desventajas de cada uno y aplicar la mejor alternativa.

# Torres de Hanoi

Te dan un conjunto de tres varillas y n*n*n discos, con cada disco de un tamaño diferente. Llamemos a las varillas A, B y C, y numeremos los discos desde 1, el disco más pequeño, hasta n*n*n, el disco más grande. Al principio, todos los n*n*n discos están en la varilla A, en orden de tamaño decreciente de la parte inferior a la parte superior, de modo que el disco n*n*n está en la parte inferior y el disco 1 está en la parte superior. Aquí está cómo se ven las Torres de Hanoi para n = 5*n*=5n, equals, 5 discos:



El objetivo es pasar todos los n discos de la varilla A a la varilla B:



¿Suena fácil, verdad? No es tan sencillo, porque tienes que obedecer dos reglas:

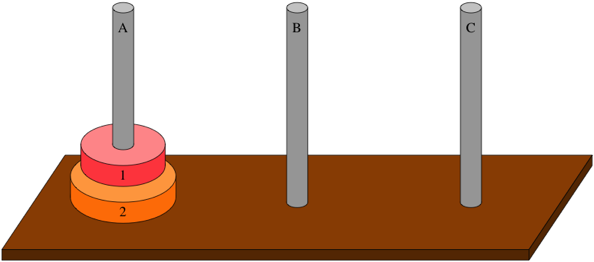
Puedes mover solamente un disco a la vez.

Ningún disco puede estar encima de un disco más pequeño. Por ejemplo, si el disco 3 está en una varilla, entonces todos los discos debajo del disco 3 deben tener números mayores que 3.

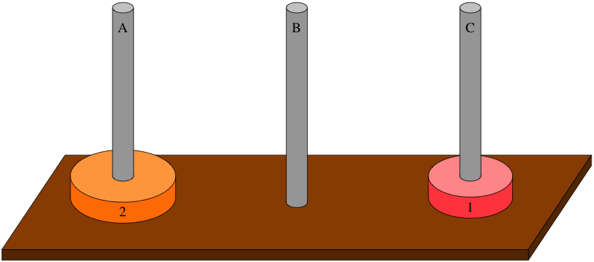
Puedes pensar que este problema no es terriblemente importante. ¡Al contrario! Cuenta la leyenda que en algún lugar de Asia (Tíbet, Vietnam, India; escoge en Internet qué leyenda te gusta), los monjes están resolviendo este problema con un conjunto de 64 discos y, según la historia, los monjes creen que una vez que terminen de mover todos los 64 discos de la varilla A a la varilla B de acuerdo con las dos reglas, el mundo se acabará. ¿Si los monjes están en lo correcto, deberíamos entrar en pánico?

Primero, vamos a ver cómo resolver el problema de manera recursiva. Vamos a empezar con un caso realmente sencillo: un disco, es decir, n = 1*n*=1n, equals, 1. El caso de n = 1*n*=1n, equals, 1 será nuestro caso base. Siempre puedes mover el disco 1 de la varilla A a la varilla B, porque sabes que cualquier disco debajo debe ser mayor. Y no hay nada especial acerca de las varillas A y B. Puedes mover el disco 1 de la varilla B a varilla C si lo deseas, o de la varilla C a la varilla A, o de cualquier varilla a cualquier varilla. Resolver el problema de las Torres de Hanoi con un disco es trivial, y requiere mover el único un disco solamente una vez.

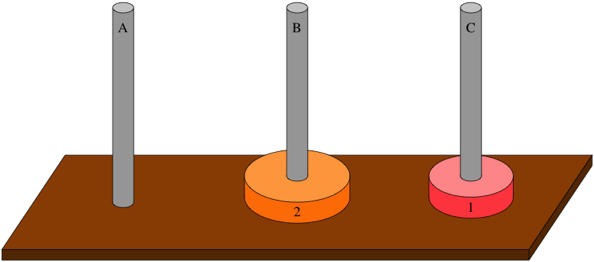
¿Qué pasa con dos discos? ¿Cómo resuelves el problema cuando n = 2*n*=2n, equals, 2? Puedes hacerlo en tres pasos. Aquí está cómo se ve al principio:



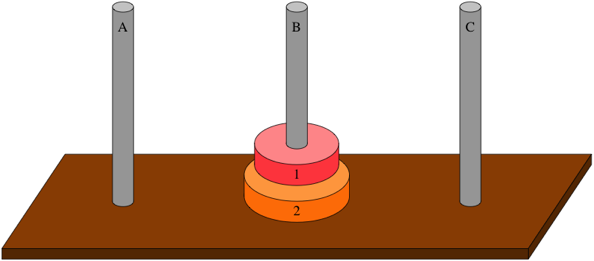
Primero, mueve el disco 1 de la varilla A a la varilla C:



Observa que usamos la varilla C como una varilla libre, un lugar en donde poner el disco 1 para que podamos llegar al disco 2. Ahora que el disco 2 (el disco inferior) está expuesto, muévelo a la varilla B:



Por último, mueve el disco 1 de la varilla C a la varilla B:



Esta solución toma tres pasos, y una vez más no hay nada especial acerca de cómo mover los dos discos de la varilla A a la varilla B. Puedes moverlos de la varilla B a la varilla C al usar la varilla A como la varilla libre: mueve el disco 1 de la varilla B a la varilla A, luego mueve el disco 2 de la varilla B a la varilla C y termina por mover el disco 1 de la varilla A a la varilla C. ¿Estás de acuerdo que puedes mover los discos 1 y 2 de cualquier varilla a cualquier varilla en tres pasos? (Di que "sí").

## Explicacion del Algoritmo de Hanoi en programación.

INICIO

T1, T2 Y T3

ALTURA INICIAL (PARA EJEMPLO DIGAMOS QUE TENEMOS 3);

**Altura 3**

Hanoi(3,T1, T2,T3)

Disco==1

FALSO

Else{

Hanoi(3-1 T1, T3, T2)

**Altura 2**

Disco==1

FALSO

Else{

Hanoi(2-1 T1,T2, T3)

**Altura 1**

VERDADERO

MOVER DISCO 1 DE T1 A LA T3

MOVER DISCO 2 DE T1 A T2

Hanoi(2-1 T3,T1,T2)

**Altura 1**

VERDADERO

Disco==1

Mover el DISCO 1 T3 A T2

MOVER DISCO 3 DE T1 A T3

Hanoi(3-1 T2, T1, T3)

**Altura 2**

Disco==1

FALSO

Else{

Hanoi(2-1 T2,T3,T1)

**Altura 1**

Disco==1

MOVER EL DISCO 1 DESDE T2 A T1

MOVER DISCO 2 DESDE TORRE B HASTA C

**Altura 1**

Hanoi(2-1 T1,T2,T3)

Disco==1

VERDADERO

MOVER DISCO 1 DESDE TORRE A HASTA C

Final

|  |  |  |
| --- | --- | --- |
| **Disco en referencia de N** | **De Donde Viene** | **Y a donde IRA** |
| 1 | T1 | T3 |
| 2 | T1 | T2 |
| 1 | T3 | T2 |
| 3 | T1 | T3 |
| 2 | T2 | T3 |
| 1 | T1 | T3 |

# Programa

El programa se ha adjuntado junto con el ejecutable correspondiente, sin embargo, se anexa aquí de igual forma junto con explicación del programa y capturas de su uso

## Ejemplo de Programa de Hanoi.

#include <stdio.h>

#include <graphics.h>

#include <conio.h>

void AlgoritmoHanoi(int, int, int, int);

int dimensionamiento();

void Discos(int,int,int);

void OcultaDisco(int, int,int);

void IntTorre();

void CambiosHanoi(int,int);

int NoDisos;

int CentralCambio[4][20];

int main(){

int Torre1=1, Torre2=2, Torre3=3, disco=0, discograph=0;

int res;

do{

dimensionamiento();

system("cls");

printf("\nA seleccionado %d disco\n",disco);

initwindow(1080,700);

setbkcolor(RGB(0, 0, 0));

cleardevice();

settextstyle(3,HORIZ\_DIR,4);

setcolor(GREEN);

outtextxy(435,50,"Torres de Hanoi");

settextstyle(3,HORIZ\_DIR,4);

setcolor(BLUE);

outtextxy(200,500,"Torre 1");

outtextxy(500,500,"Torre 2");

outtextxy(800,500,"Torre 3");

setcolor(BROWN);

setfillstyle(SOLID\_FILL,BROWN);

rectangle(135,420,965,470);

floodfill(300,450,BROWN);

rectangle(235,150,265,420);

floodfill(250,300,BROWN);

rectangle(535,150,565 ,420);

floodfill(550,300,BROWN);

rectangle(835,150,865 ,420);

floodfill(850,300,BROWN);

IntTorre();

delay(**2**000);

AlgoritmoHanoi(NoDisos,Torre1,Torre2,Torre3);

delay(3000);

cleardevice();

closegraph();

printf("\n\n¿Desea volver a jugar?\n");

printf("\n1.Si (La visualizacion puede tener errores de ubicacion en T3)");

printf("\n2.No");

printf("\n\nElija su Opcion:");

scanf("%d",&res);

system("cls");

}while (res==1);

return 0;

}

int dimensionamiento(){

int disco;

printf("\t\t\t\t\t\tMenu Hanoi\n\n");

printf("Este programa le mostrara la resolucion del Hanoi\n\n");

printf("Indique,¿con cuantos discos desea jugar?(No mas de 15 para evitar errores de visualizacion)");

scanf("%d",&NoDisos);

}

void AlgoritmoHanoi(int disco, int T1, int T2, int T3){

if(disco>0){

AlgoritmoHanoi(disco-1, T1, T3, T2);

CambiosHanoi(T1,T3);

delay(1200);

printf("Mover disco %d de Torre %d a Torre %d \n",disco,T1 ,T3);

AlgoritmoHanoi(disco-1,T2, T1, T3);

}

}

void IntTorre(){

int valor=NoDisos;

CentralCambio[1][0]=NoDisos;

for(int j=1;j<=NoDisos;j++){

Discos(1,j,valor);

CentralCambio[1][j]=valor;

valor--;

}

}

void Discos(int pos,int alt, int tam){

int altx,tamp,palox,paloy=400;

if(pos==1)

palox=250;

if(pos==2)

palox=550;

if(pos==3)

palox=850;

altx=paloy-(22\*(alt-1));

tamp=(tam\*20)+50;

if (tam==6)

tam=1;

setcolor(tam);

setfillstyle(SOLID\_FILL,tam);

rectangle(palox-(tamp/2),altx,palox+(tamp/2),altx+20);

floodfill(palox-(tamp/2)+5,altx+10,tam);

}

void OcultaDisco(int torre,int alt, int tam){

int altx,tamp,palox,paloy=400;

if(torre==1)

palox=250;

if(torre==2)

palox=550;

if(torre==3)

palox=850;

altx=paloy-(22\*(alt-1));

tamp=(tam\*20)+50;

setcolor(0);

setfillstyle(SOLID\_FILL,0);

rectangle(palox-(tamp/2),altx,palox+(tamp/2),altx+20);

floodfill(palox-(tamp/2)+5,altx+10,0);

setcolor(BROWN);

setfillstyle(SOLID\_FILL,BROWN);

rectangle(palox-15,altx,palox+15,altx+20);

floodfill(palox,altx+10,BROWN);

}

void CambiosHanoi(int salida, int destino){

int aux1,aux2;

CentralCambio[destino][0]+=1;

aux1=CentralCambio[destino][0];

aux2=CentralCambio[salida][0];

CentralCambio[destino][aux1]=CentralCambio[salida][aux2];

OcultaDisco(salida,CentralCambio[salida][0],CentralCambio[salida][aux2]);

Discos(destino,CentralCambio[destino][0],CentralCambio[destino][aux1]);

CentralCambio[salida][aux2]=0;

CentralCambio[salida][0]-=1;

}

## Explicación del programa

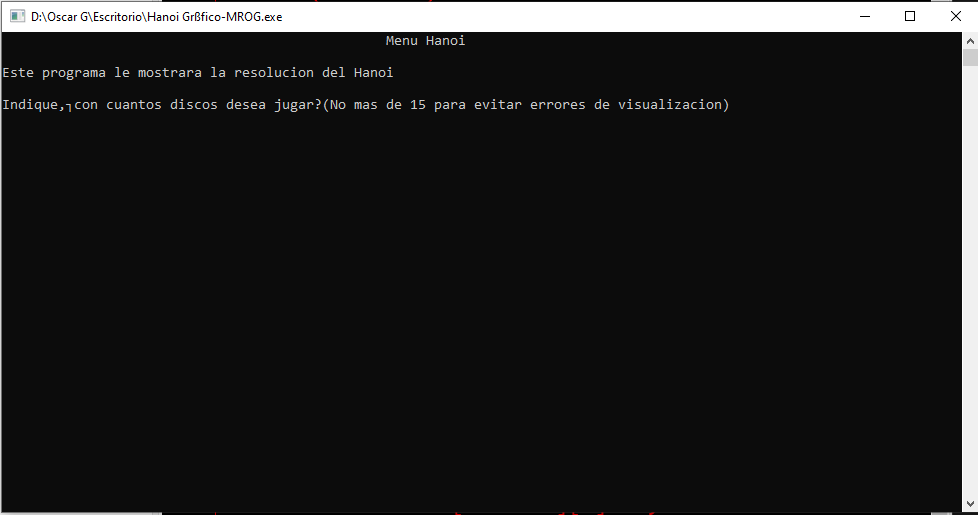
El presente programa está estructurado de varias funciones, una que le permite dimensionar el tamaño de la torre de Hanoi a utilizar, otro que le permite generar la torre, un menú concentrado de opciones, Funciones para establecer cambios en la torre, y otra para borrar los discos cambiantes., y por último como no debía de faltar el modo gráfico que enseña y muestra de una manera amigable para el usuario los cambios en Hanoi.

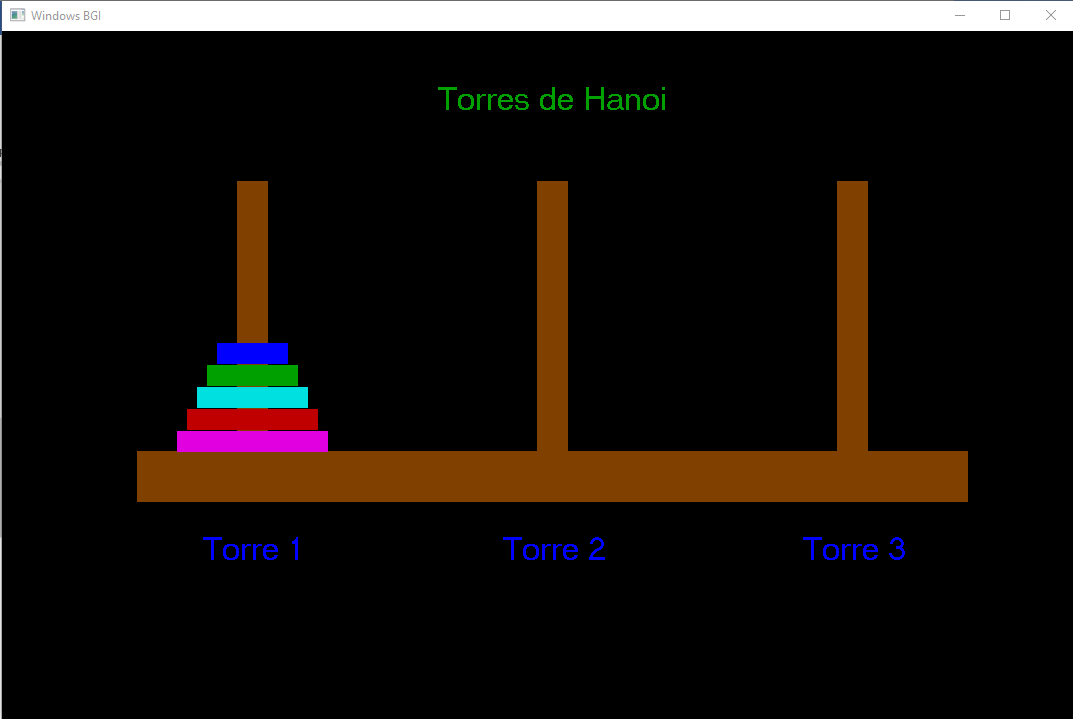
De nuevo podemos ver como este programa está considerado para abarcar todos los deseos básicos que el usuario podría tener durante el uso de este programa, este programa trabaja muy a detalle el modo grafico tal que podemos visualizar todos los cambios que tiene durante el trabajo del algoritmo.

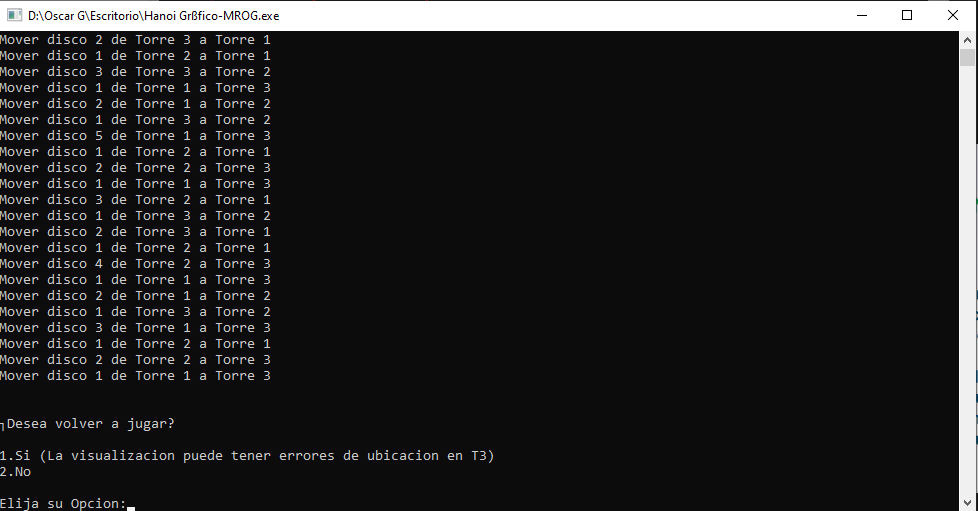
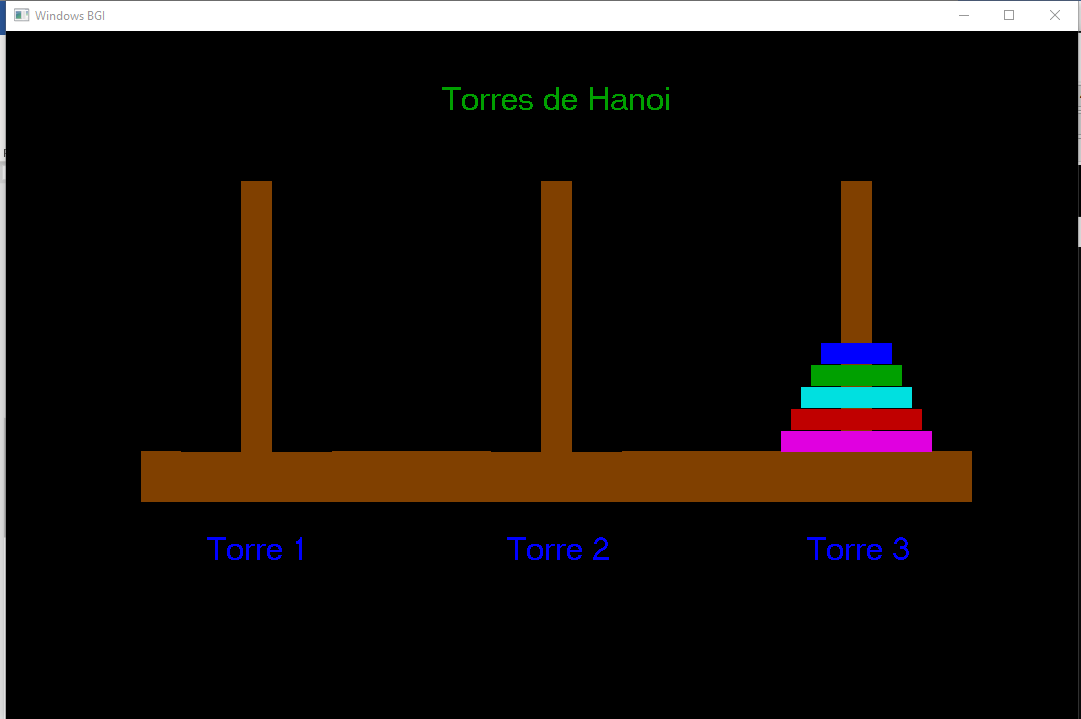
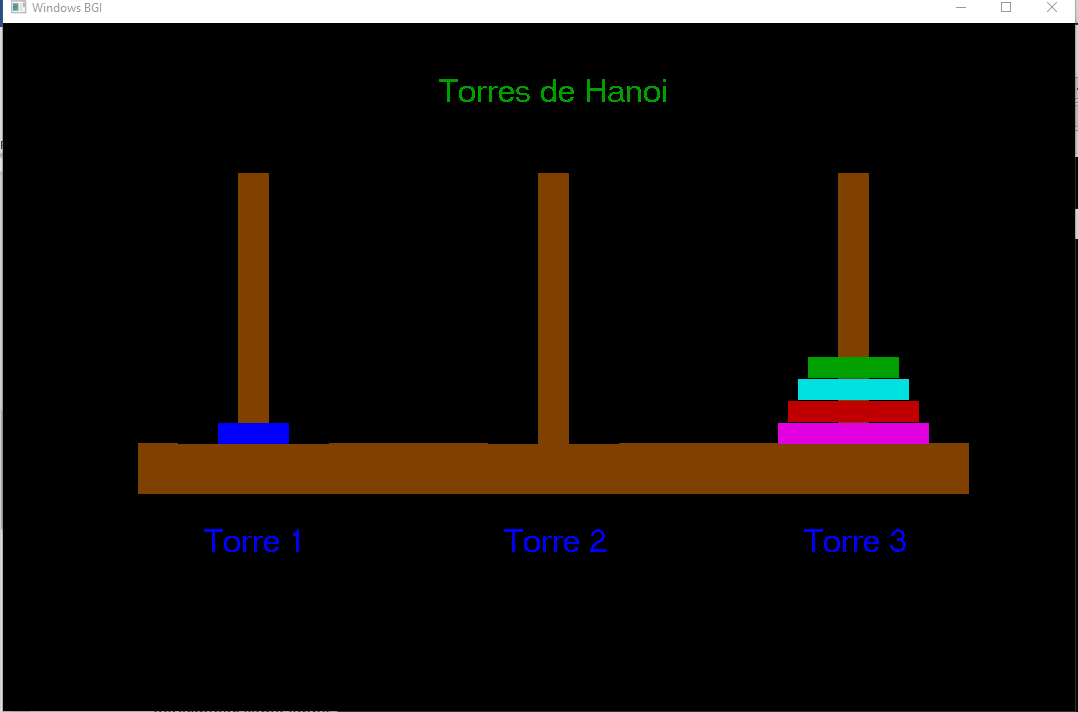
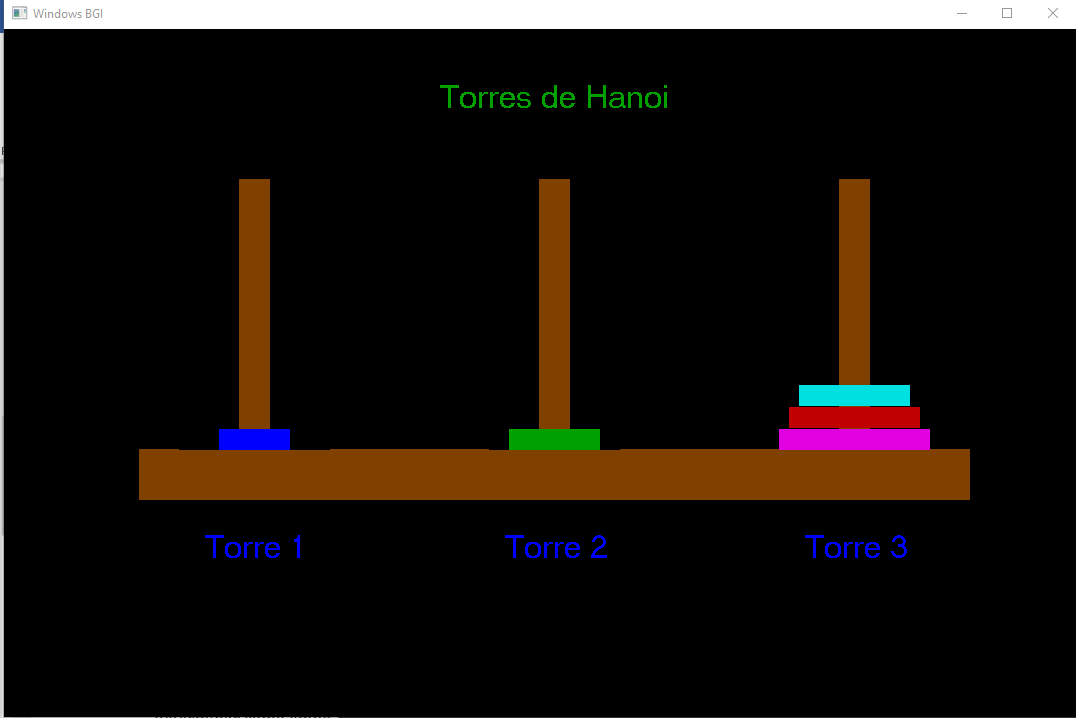
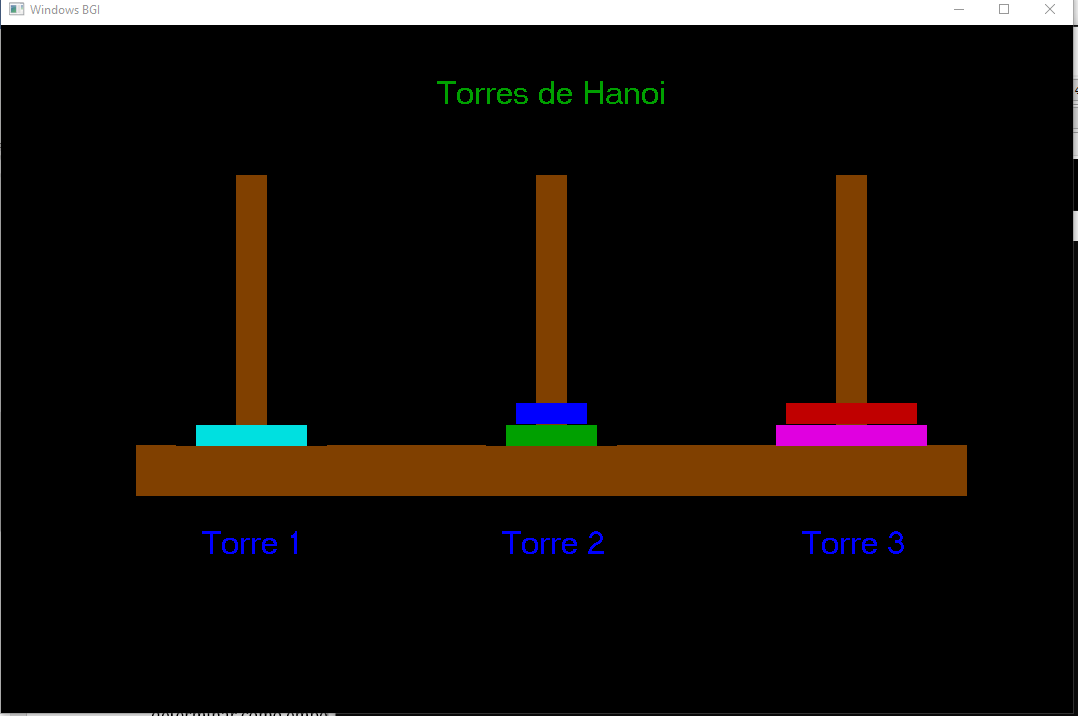
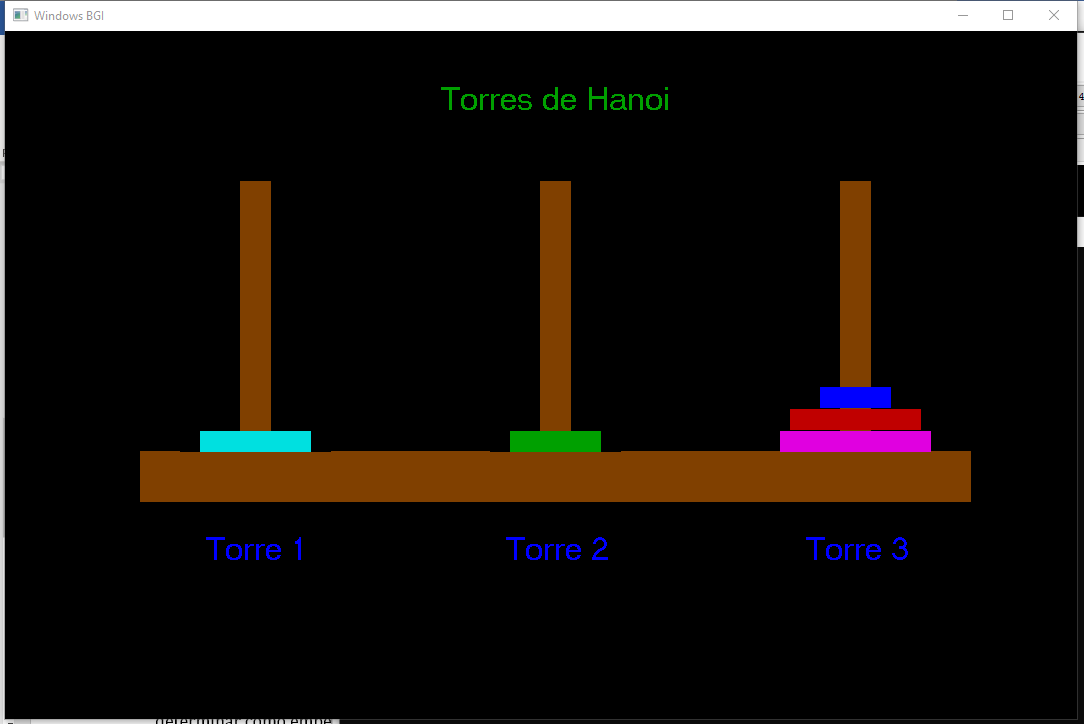
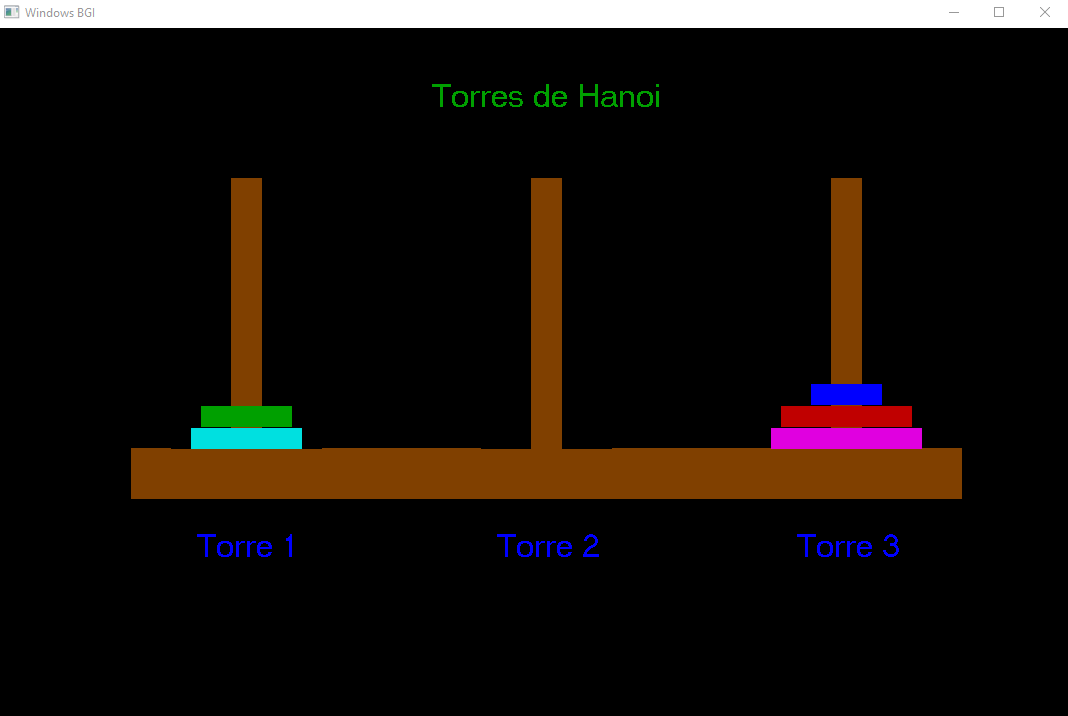
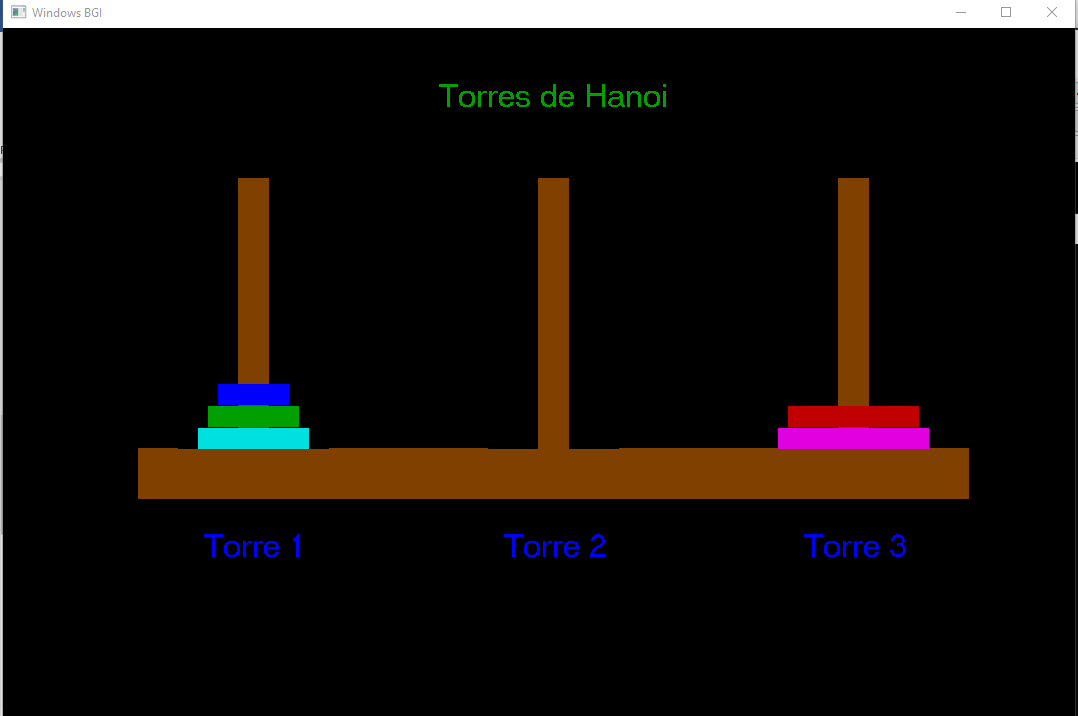
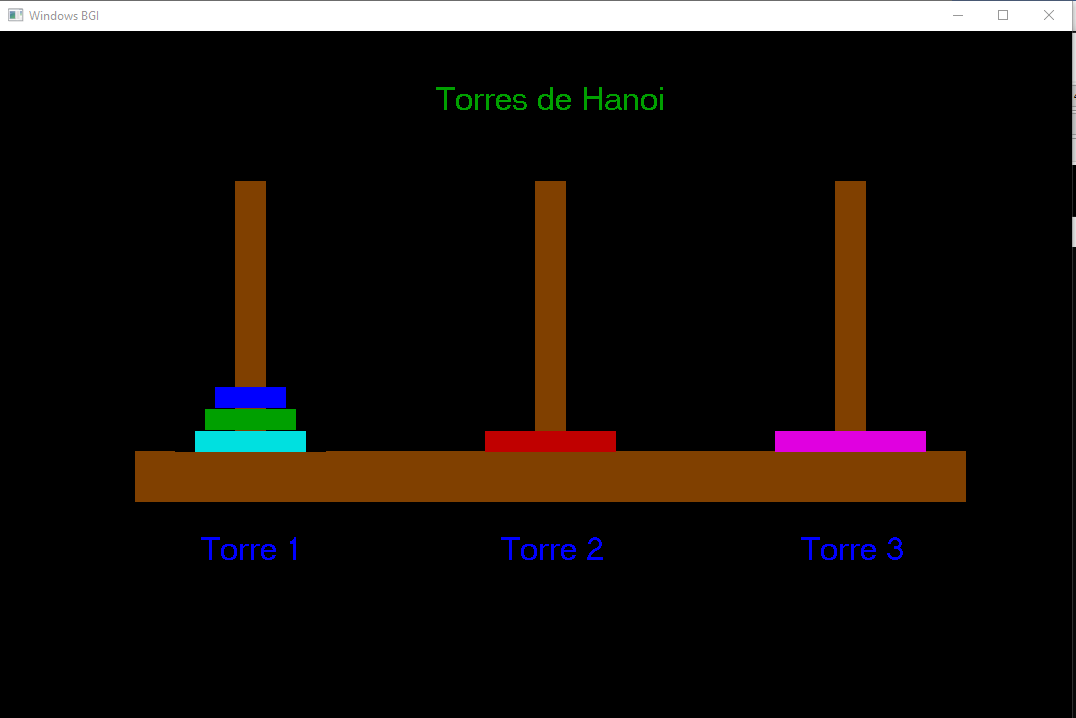
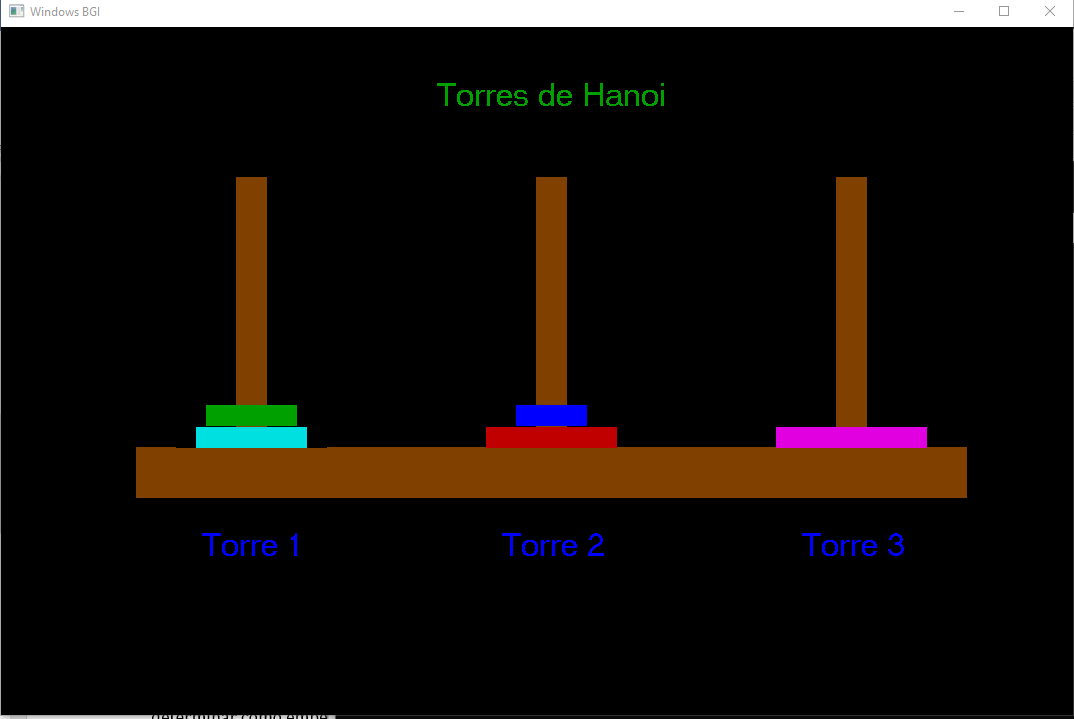
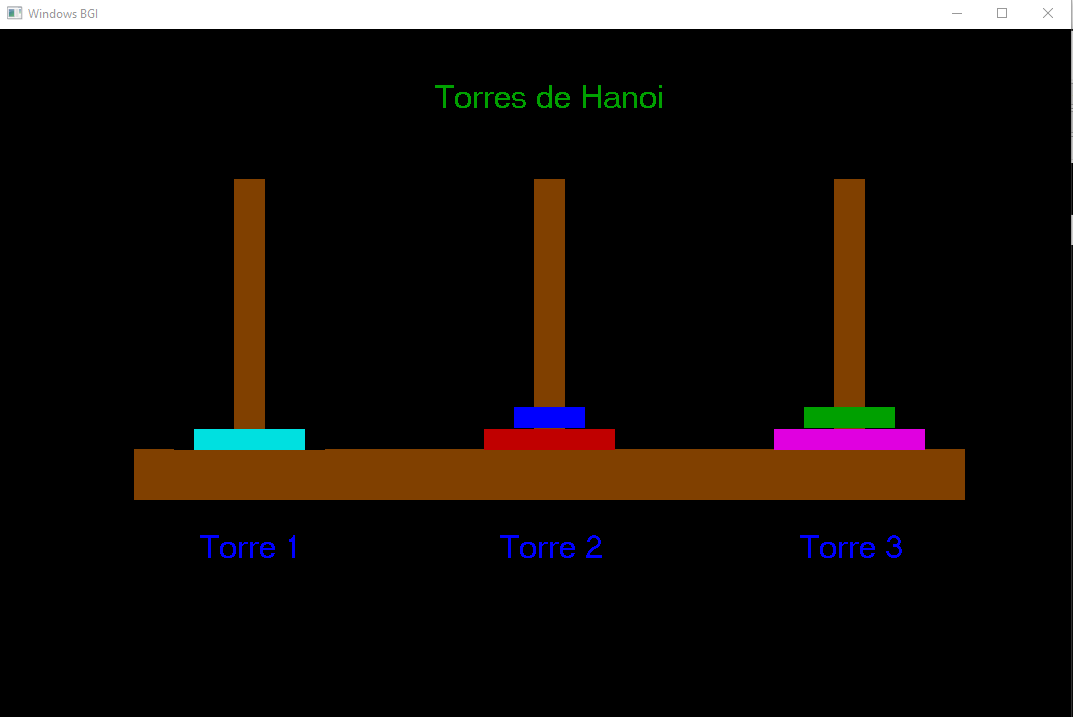
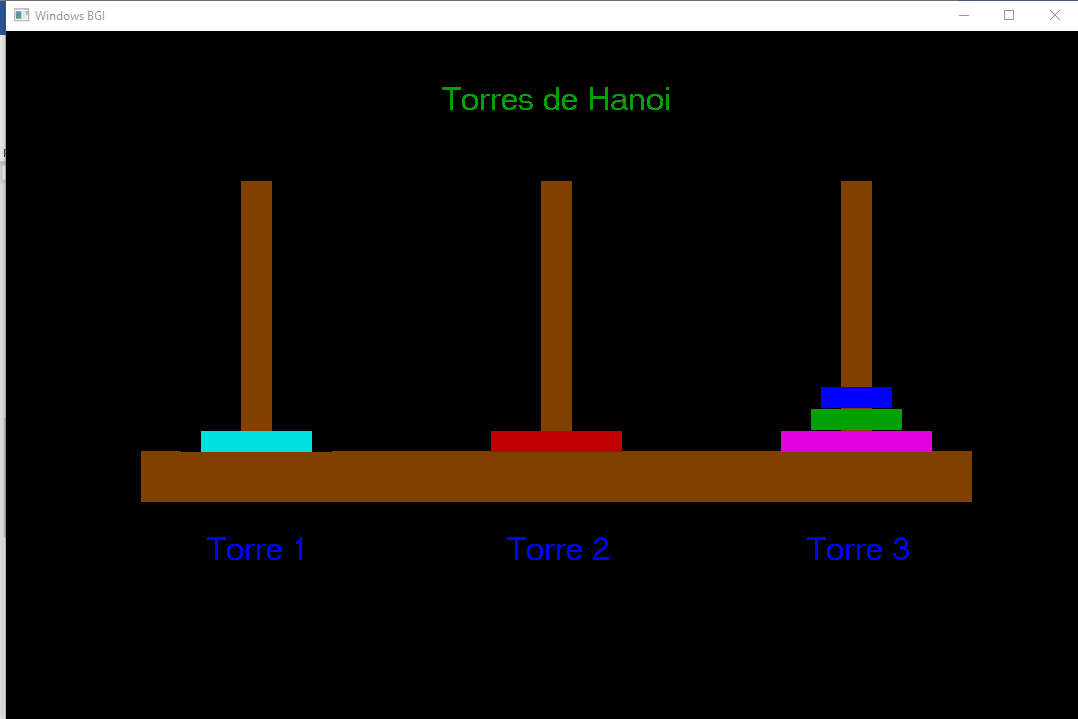
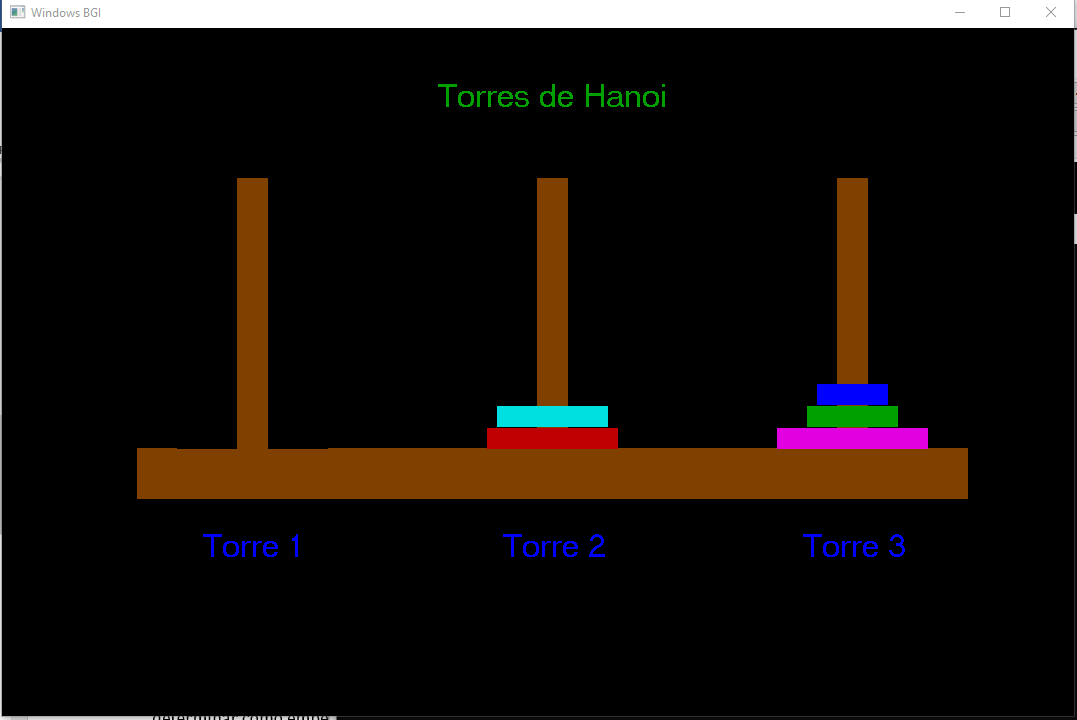
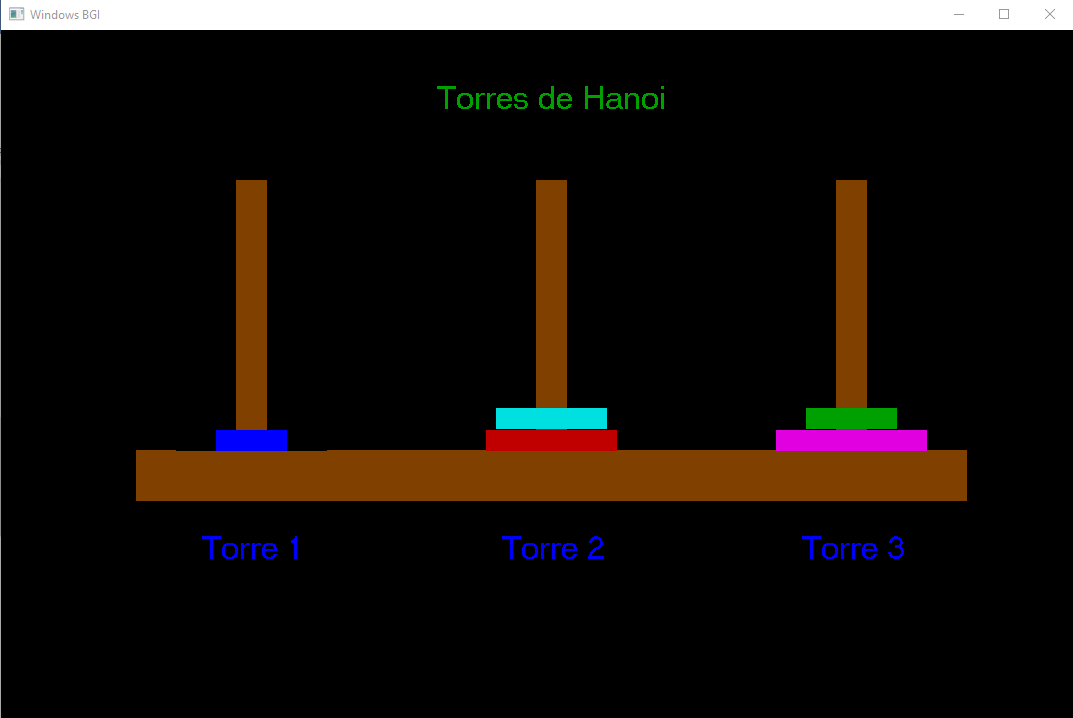
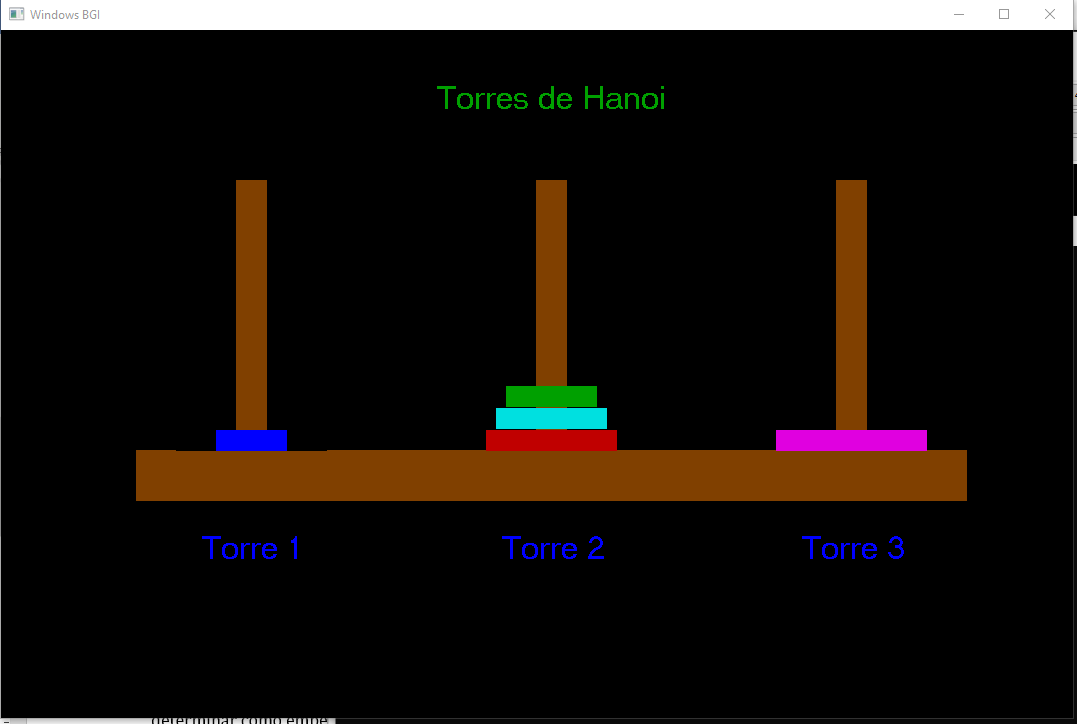
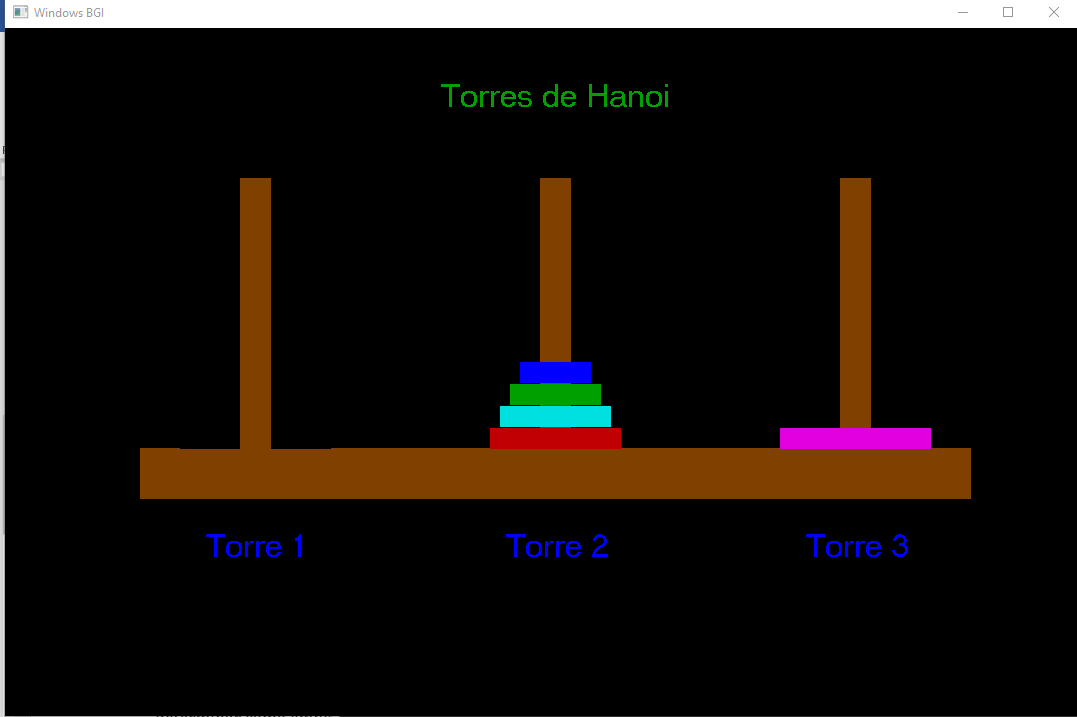
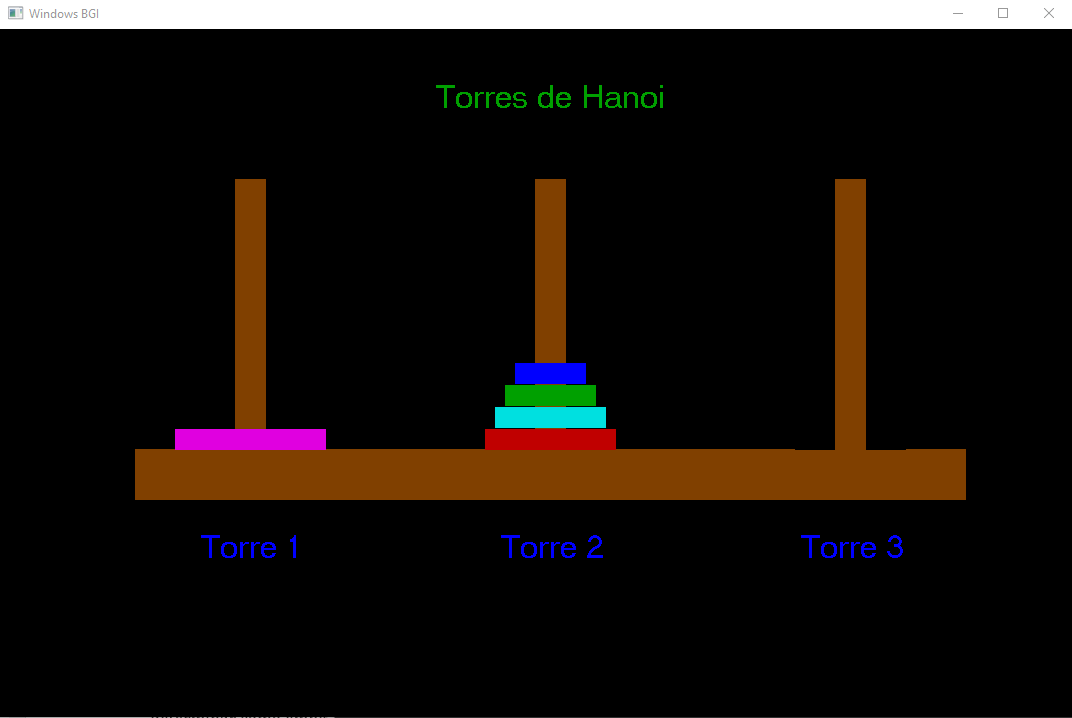
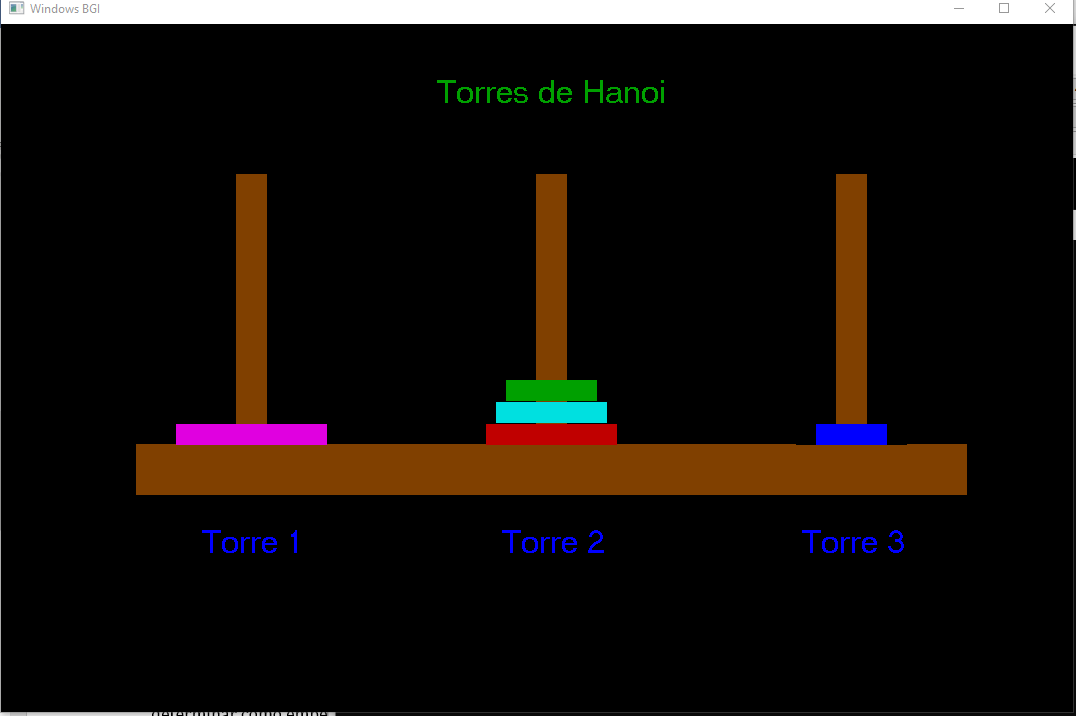
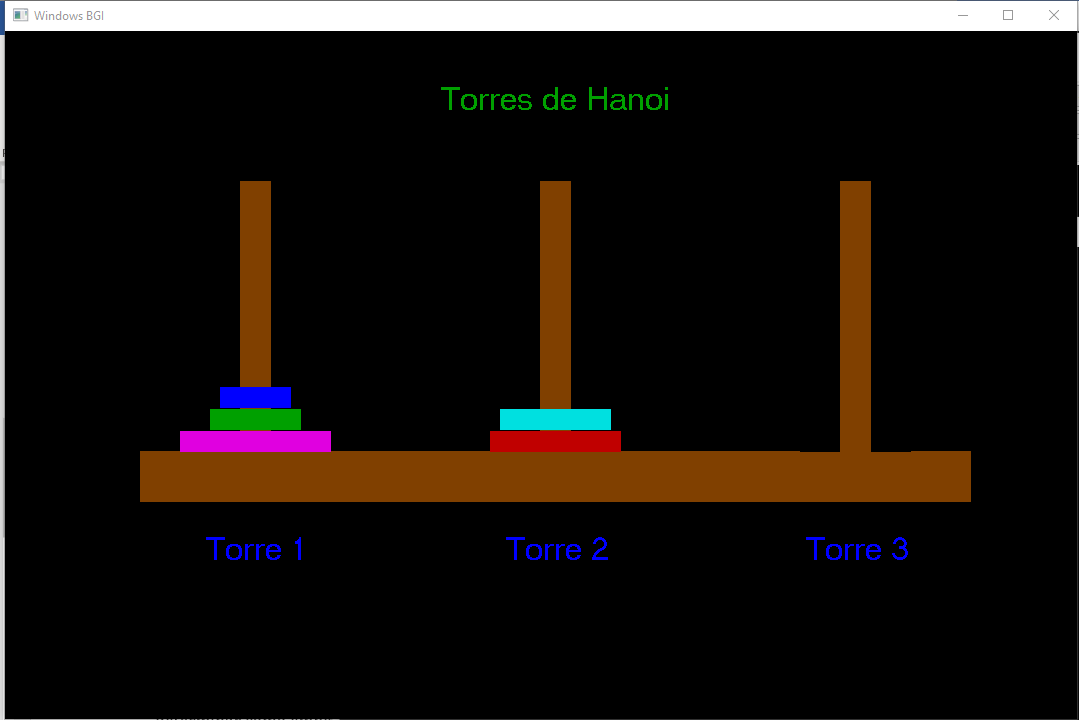
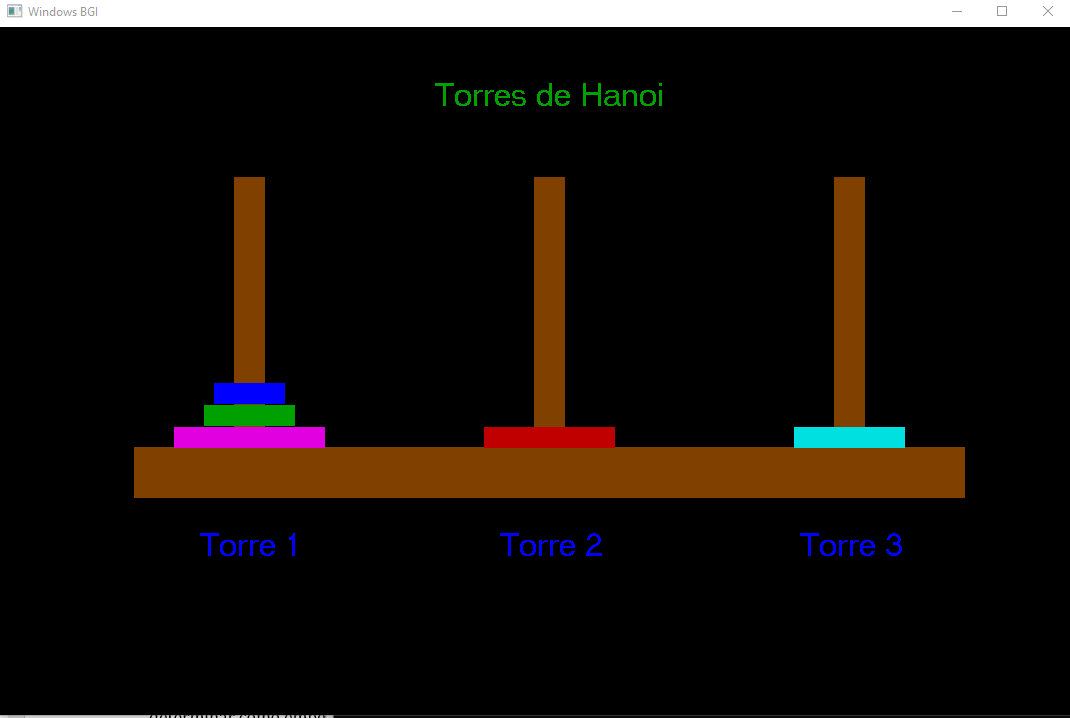
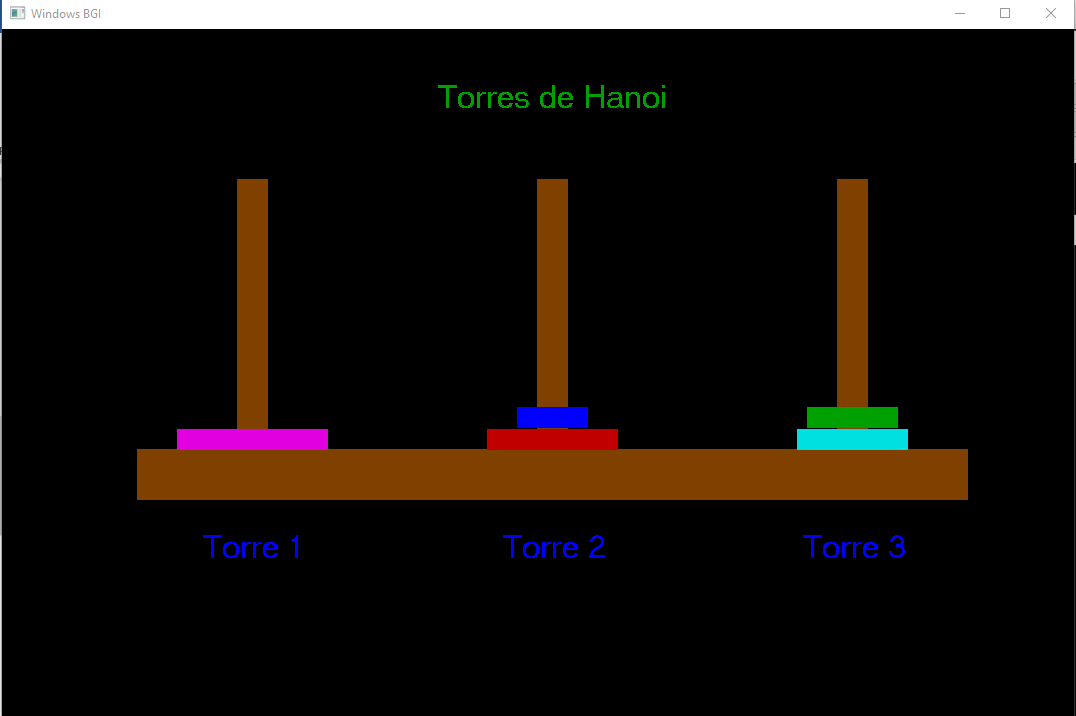
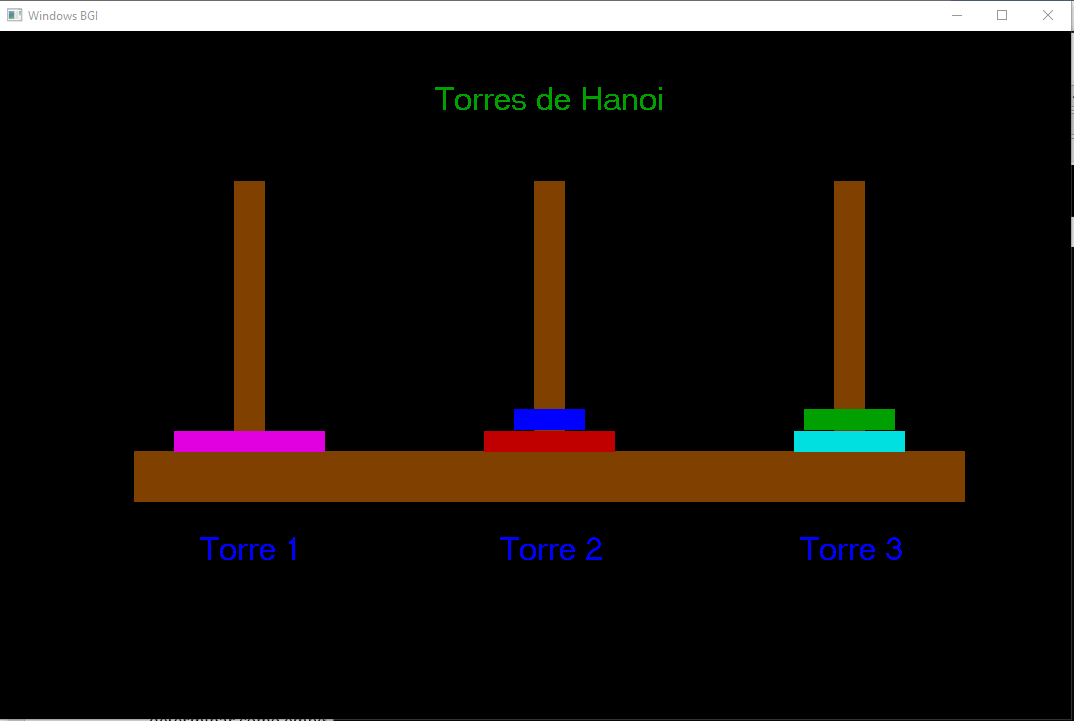
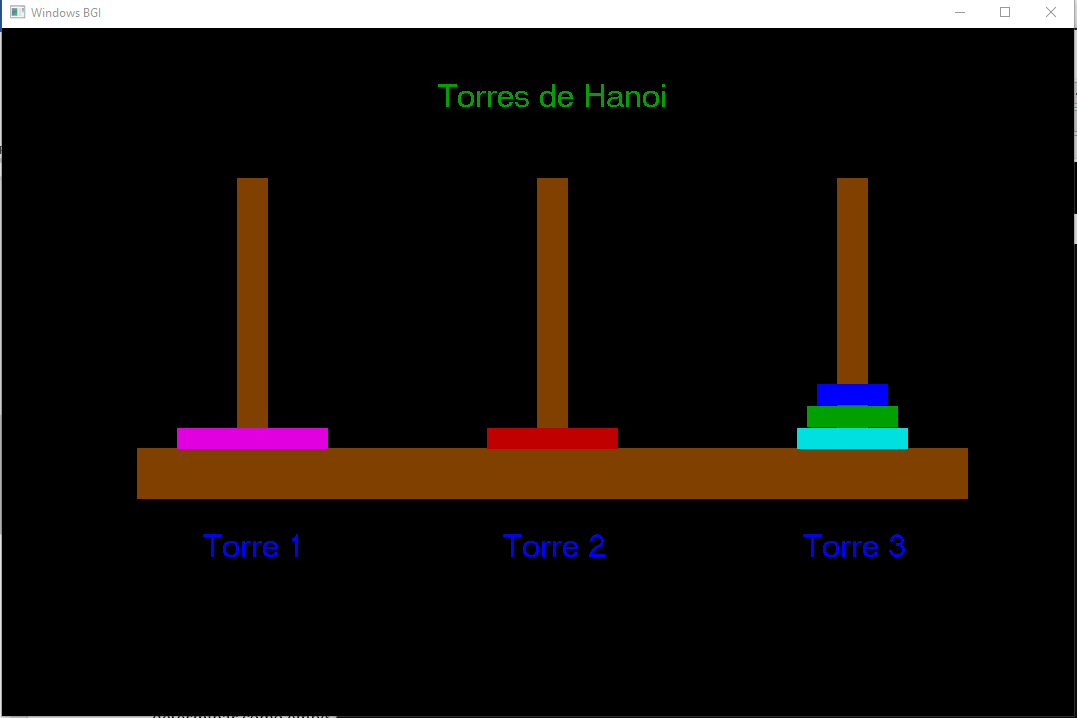
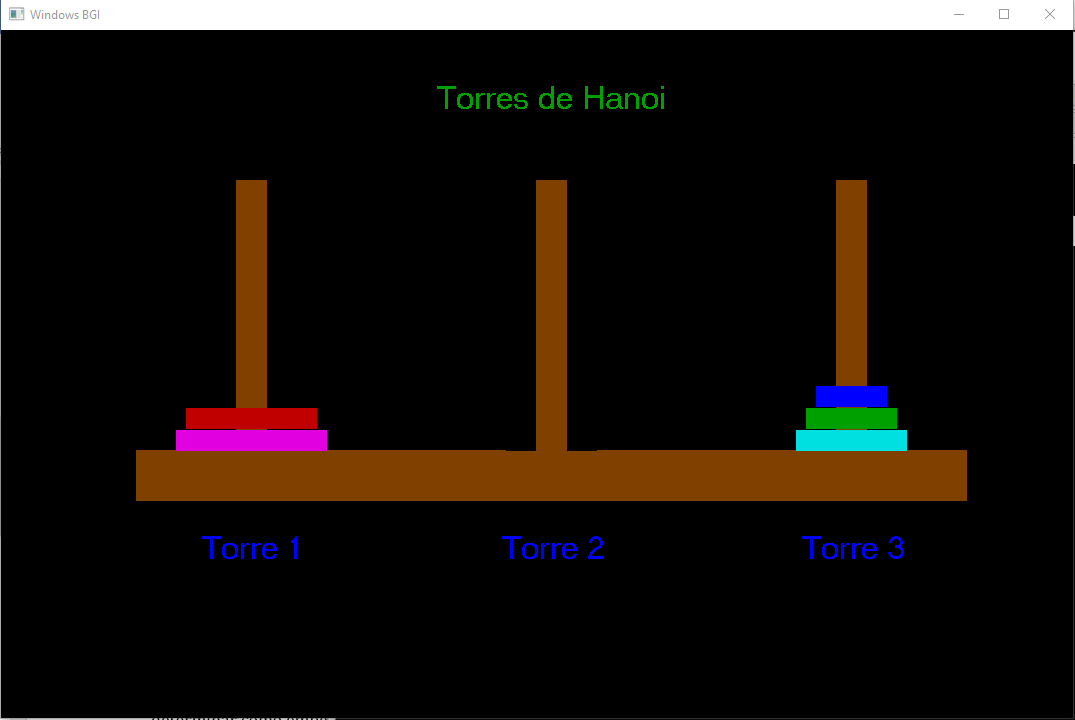
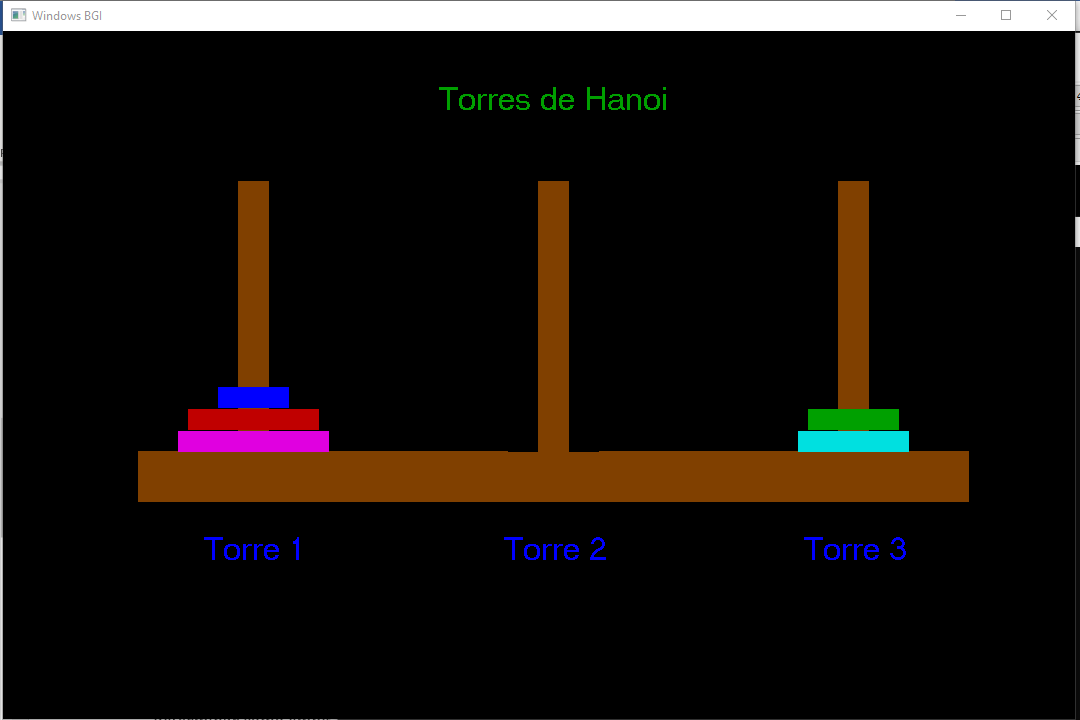
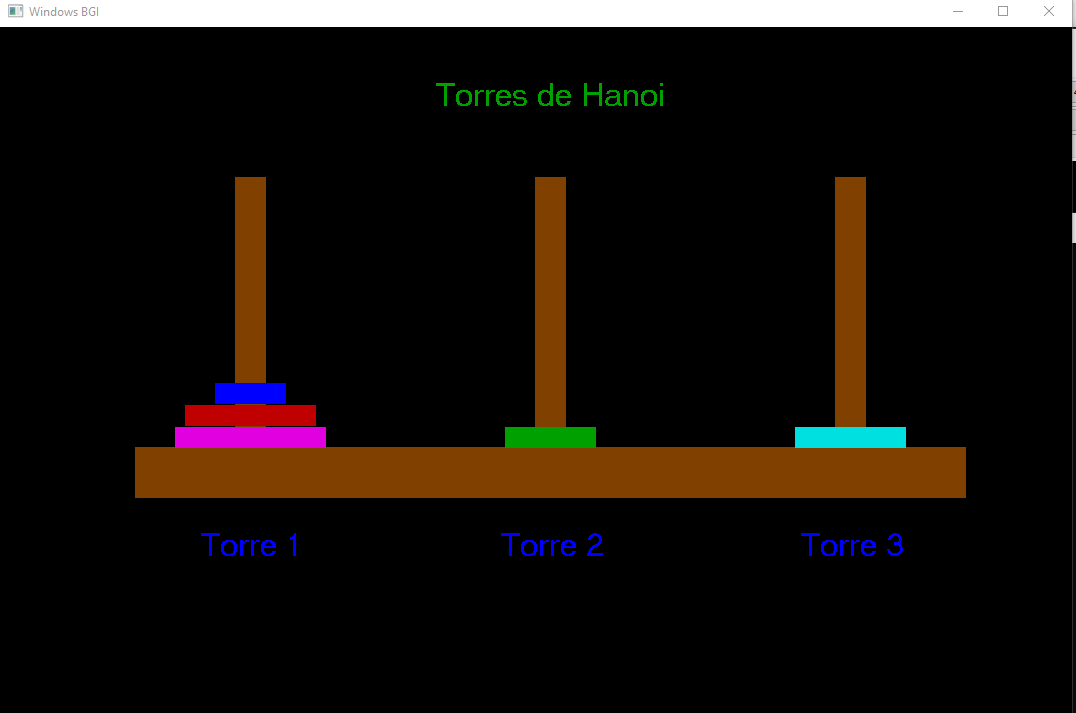
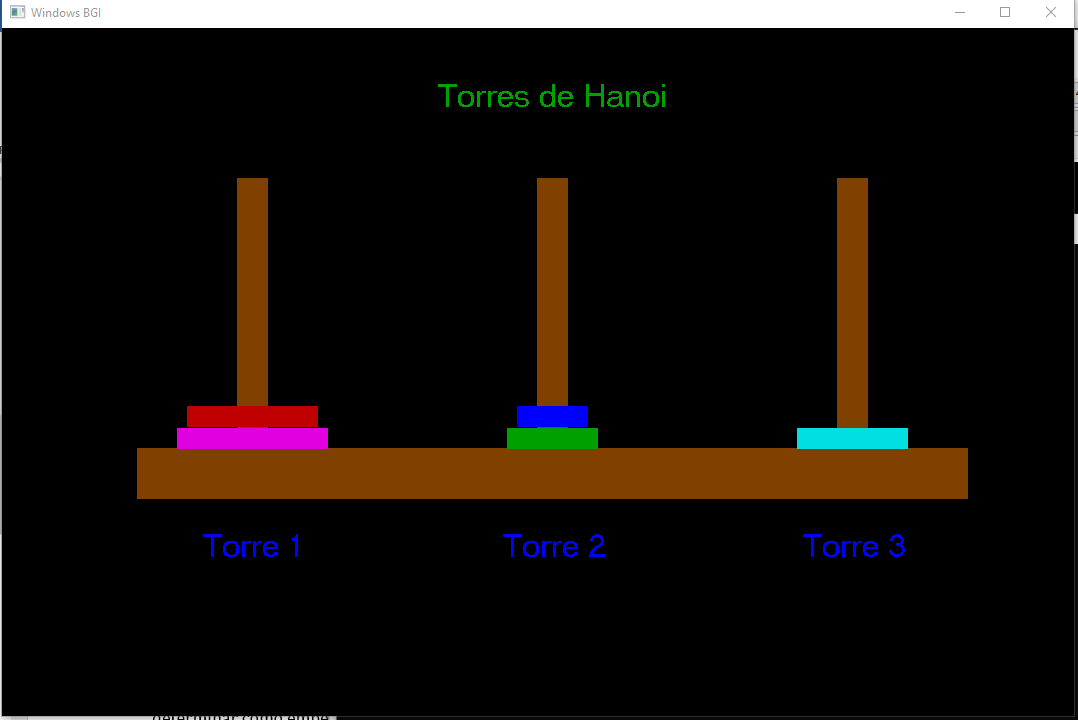
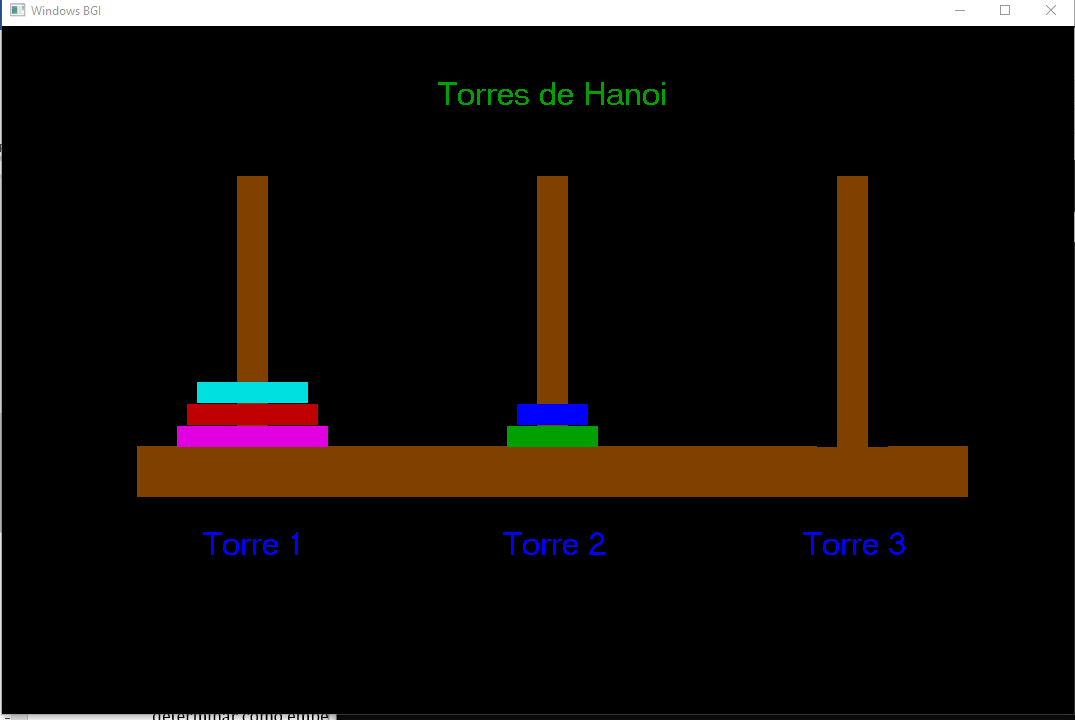
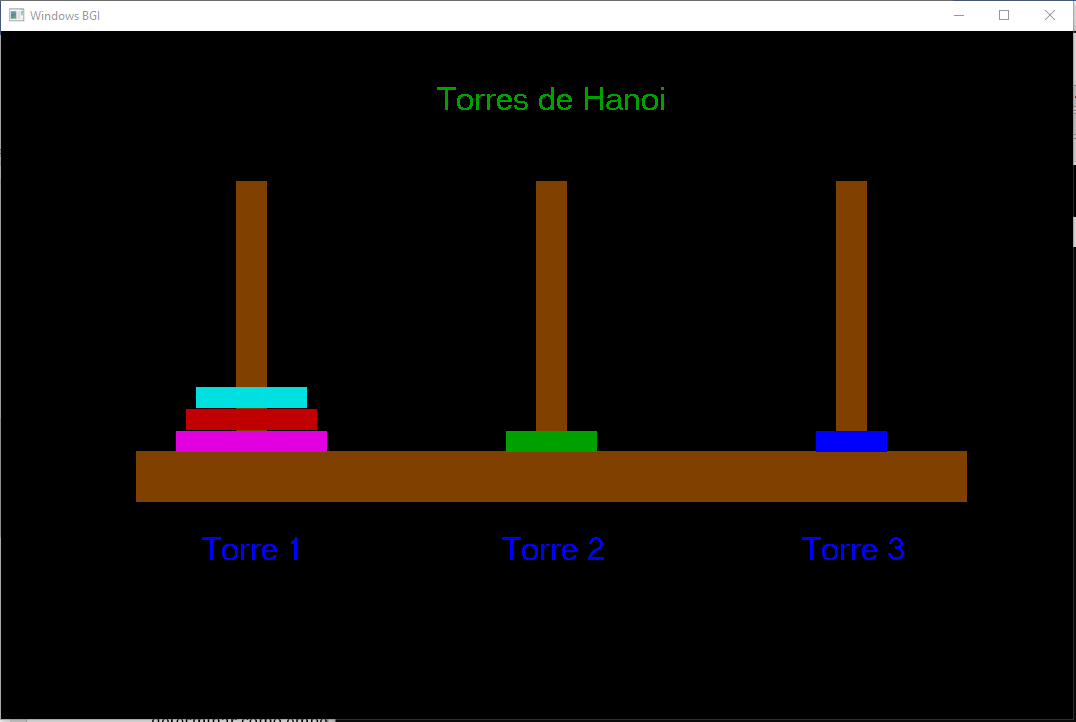
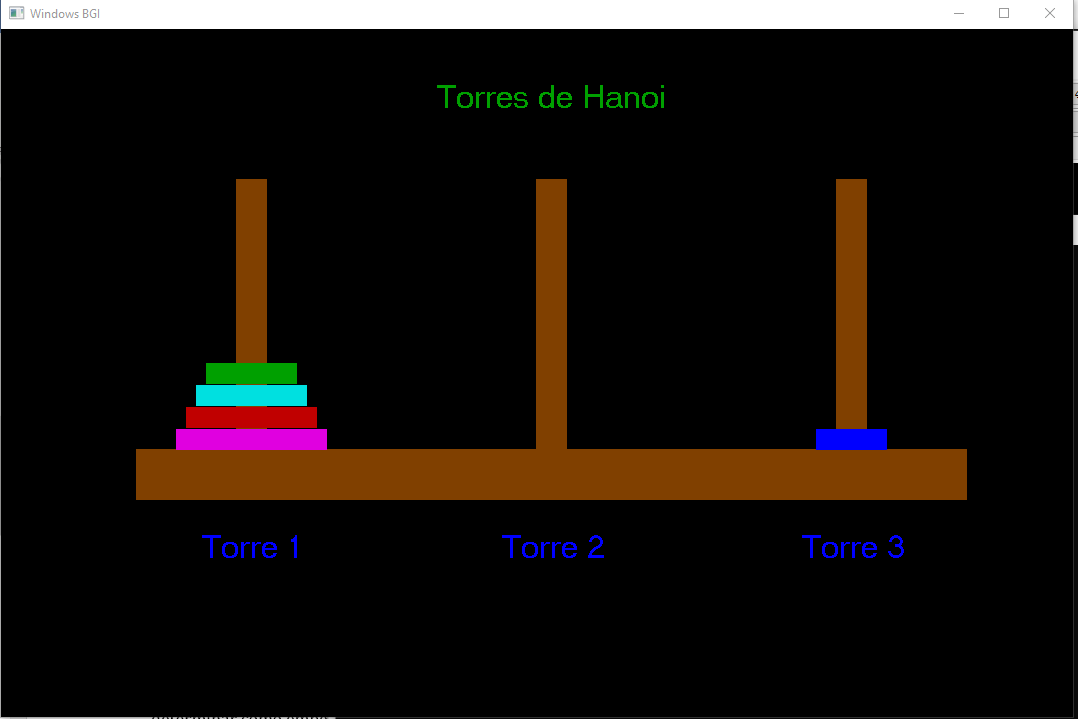
No podemos perder de vista que este programa contiene recursos recursivos que son vitales para poder hacer este algoritmo para N discos, sin tener que tener que programar infinidad de lineas para hacerlo posible.

Podemos tambien notar que el programa maneja muchas funciones que trabajan como un reloj, en esta ocacion el modo grafico es uno de los mas complejos que se han hecho, hasta el momento, y que requiere una visualizacion diferente.

## Capturas del programa







# Conclusiones

El presente programa fue sin duda uno de los programas en los que más trabajo me costó determinar cómo empezar (De nuevo) , pues había muchos caminos que se podían tomar sin embargo estoy satisfecho con la elección que se realizó en cada uno de los caminos con opciones a tomar, el más relevante de ellos fue el uso de las distintas funciones ocupadas en el programa aplicando así la recursividad y la iteraticidad

En con lo que respecta al uso de la recursividad, consideró seriamente una de las herramientas más importantes a la hora de programar, pues nos permite evitarnos el cúmulo de variables basura, y pasos repetitivos que lo único que hace es entorpecer muchas veces el manejo de la información albergada en ellas ,y es por ello que considero que es uno de los recursos más valiosos para la construcción de estructuras de datos y qué será de gran ayuda en los aprendizajes y conocimientos por adquirir a lo largo de esta unidad de aprendizaje , por lo que entiendo totalmente la lógica de que sea tan importante para continuar nuestro cu

# Referencias Bibliográficas:

* Brian W. K. & Dennis M. R. (1978) Lenguaje de programación C. Pearson Education Recuperado 29 septiembre, 2020 Disponible en:<http://www.tecnica1lomas.com.ar/tutoriales/lenguaje_C.pdf>
* ITERATIVOS. (2020). Retrieved October 31, 2020, from Uacj.mx website: <http://www3.uacj.mx/CGTI/CDTE/JPM/Documents/IIT/repetitivas/iterativos.html#:~:text=ITERATIVOS&text=Las%20estructuras%20de%20control%20iterativas,veces%20un%20conjunto%20de%20instrucciones.&text=Cuando%20en%20un%20programa%20necesita,necesario%20implementar%20un%20ciclo%20iterativo>
* ThemeGrill. (2013, March 11). Recursividad o Iteración. Retrieved October 31, 2020, from Club de Tecnología website: <https://www.clubdetecnologia.net/blog/2013/recursividad-iteracion/>
* Rubén Sánchez Sancho. (2018). Estructuras Iterativas · ciencia-de-datos-con-r. Retrieved October 31, 2020, from Gitbooks.io website: <https://rsanchezs.gitbooks.io/ciencia-de-datos-con-r/content/estructuras_control/iterativas/estructuras_iterativas.html>
* ‌Introducción a la recursividad. – Instinto Lógico. (2013, March). Retrieved October 31, 2020, from Instintologico.com website: <http://instintologico.com/introduccion-a-la-recursividad/>
* Diferencia entre iteración y recursividad. (2020). Retrieved October 31, 2020, from CódigoFacilito website: <https://codigofacilito.com/articulos/articulo_16_10_2019_16_22_35>
* Torres de Hanoi (artículo) | Algoritmos | Khan Academy. (2018). Retrieved October 31, 2020, from Khan Academy website: <https://es.khanacademy.org/computing/computer-science/algorithms/towers-of-hanoi/a/towers-of-hanoi>