EN ESTE NÚMERO ANALIZAMOS: XML, EL ESTÁNDAR PARA LA COMUNICACIÓN DE COMPONENTES

PROGRAMACIÓN ACTUAL

www.prensatecnica.com AÑO 3 • NÚMERO 32 995 pts.
PORTUGAL 995 ESC (CONT) 5,98 €

CONCURSO

Sorteamos 2 BORLAND Delphi 5 en pág. 41

¿LA ALTERNATIVA A ENTORNOS COMERCIALES?

OPEN SOURCE

CÓDIGO LIBRE

En Silicio
MICROCONTROLADORES: EL MC68HC11

Reportaje
PLANIFICACIÓN DE TAREAS

Espacio GNU
X-WINDOW: EL JUEGO DEL XNAKE
PROGRAMACIÓN EN INTERNET CON PERL
CREACIÓN DE INTERFACES GRÁFICAS CON GTK

Nivel Básico
LOS OBJETOS EN VRML
JAVA: LOS SECRETOS DE SU LENGUAJE

En Vanguardia
REDES NEURONALES, ALGORITMOS GENÉTICOS Y APRENDIZAJE

Análisis
VISUAL BASIC 6 PROFESIONAL EDITION DE MICROSOFT

Contenido CD-Rom

DEMO:
BORLAND DELPHI 5

DESARROLLO WEB:
Editor de CGIs en Perl

UTILIDADES:
Copy Control • Panda Antivirus 6.0 Platinium • Norton Antivirus • Internet Explorer 5 • Netscape 4.7

COMPRESORES:
WinRAR 2.6 • NetZip Clasic Deluxe • DataZip 4.005

ACCESO A DATOS DESDE VISUAL BASIC 6.0

Autor: Santiago Romero
santiago.romero iname.com

XSnake: ejemplo completo (II)

En esta entrega comentaremos todos los aspectos básicos de programación (gestión de eventos, gráficos, etc.) mostrados en el código fuente de XSnake, el juego de ejemplo que se introdujo en nuestro anterior artículo.

Para el comentario del juego será necesario tener a nuestro lado la revista del mes pasado con los 2 listados *xsnake.h* y *xsnake.c*, o bien abrir mediante un editor de textos el programa incluido en el CD-Rom de la revista, ya que vamos a comentar las funciones o secciones de código más importantes, obviando aquello que resulte demasiado básico.

FICHERO VARS.H

Antes de empezar a comentar cada variable o constante, veamos una visión general de cómo se ha estructurado el programa SNAKE en una fase previa de diseño. Para ello vamos a describir el problema.

- ***Variables:***

serpiente_x, serpiente_y: coordenadas de la cabeza de la serpiente.
cola_x, cola_y: coordenadas de la cola de nuestra serpiente.
Direcciones: variables booleanas (true/false) que indican la dirección de movimiento de nuestra serpiente.
Pantalla[ancho][alto]: en lugar de trabajar sobre la pantalla, se trabaja sobre una matriz que contendrá ceros para los huecos, unos donde haya porciones de nuestra serpiente, doses donde haya muros, treses donde haya comida, etc. *Cola[ancho][alto]:* esta matriz contiene las direcciones que va dejando nuestra serpiente en su movimiento, y se utiliza para poder borrar la cola en todo momento. Más adelante veremos porqué.

- ***Programa:***

```
InicializaVariables;
  Mientras ( NO se pulse ESCAPE )
  {
LeeMovimientoDelTeclado;
    MueveSerpiente;
    pantalla[serpiente_x][serpiente_y] = 1.
    cola[serpiente_x][serpiente_y] = direccion;
    BorraCola;
  }

Funcion LeeMovimientoDelTeclado:
  si tecla == ARRIBA entonces direccion = arriba;
  si tecla == ABAJO entonces direccion = abajo;
  (etc...)

Función MueveSerpiente:
  si direccion == arriba entonces serpiente_y =
serpiente_y - 1;
  si direccion == abajo entonces serpiente_y =
serpiente_y + 1;
  (etc...)

Funcion BorrarCola;
  pantalla[cola_x][cola_y] = 0;
  /* Calculamos la nueva cola */
  si cola[cola_x][cola_y] == arriba entonces
cola_y—;
  si cola[cola_x][cola_y] == arriba entonces
cola_y++;
  (etc...)
```

Como se puede ver en la función *BorrarCola()*; la matriz cola[][] es necesaria para borrar la antigua cola (el último recuadro de la serpiente) y poder decidir cuál es la nueva cola.

FICHERO VARS.H

En este fichero se definen las variables y constantes utilizadas en el programa. La variable *VELOCIDAD* especifica el tiempo que usaremos de espera entre fotograma y fotograma, estando por defecto en 70.000 microsegundos, lo que daría lugar (teniendo en cuenta que 1 segundo son 1 millón de microsegundos) a 1.000.000/70.000 = 14 fotogramas por segundo.

Es recomendable trabajar sobre una matriz que represente a la pantalla

La variable *INCREMENTO* hace referencia al número de recuadros en que aumentará nuestra serpiente al coger una manzana, y *MAX_COMIDA* el número máximo de manzanas en pantalla simultáneamente. Posteriormente también se declaran las dimensiones de nuestras matrices (80x40) en *xmax, ymax, xmin, ymin*. Lo siguiente son definiciones clásicas (*boolean, true* y *false*), seguidas de los valores que vamos a introducir dentro de la matriz (0=hueco, 1=serpiente, 2=pared, etc.). Para la comida hay 4 valores, puesto que vamos a crear manzanas de 2x2 huecos en la matriz, con lo cual necesitamos 4 valores dentro de pantalla[][] para cada una de sus 4 esquinas.
A estas constantes les siguen las anchuras de los *sprites*, y unas definiciones para las direcciones (*DIR_IZQ, DIR_DER*, etc.). Por último se definen los *scancodes* de las teclas que se van a detectar, para su posterior uso en el bucle de recogida de eventos (evento KeyPress). Estos *scancodes* están definidos en */usr/X11R6/lib/X11/xkb/keycodes/xfree86*, y de ellos hemos cogido los básicos: la tecla de *ESCAPE* (9) y los cursores (98, 104, 100, 102).
A continuación vienen las estructuras y variables globales utilizadas.

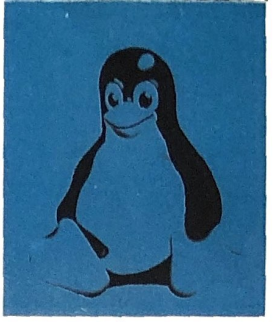
Listado 2. Pseudocódigo del bucle de eventos

```
/* hacer done = 1 para salir */
done = 0;

while( !done )
{
  Mientras ( haya eventos en la cola de Display )
  {
    CogerEvento;
    si evento.ventana != nuestra ventana
      entonces seguir;
    ActuarSegunEvento( tipo );
  }

  LeerMovimiento;
  MoverSerpiente;
  GenerarComida;
  DibujarSerpiente;
  BorrarSerpiente;

  parar( TIEMPO_PARADA )
}
```



El vector de *pixmaps* llamado *sprites*, servirá para almacenar los gráficos del juego.
A continuación tenemos algunas variables que usaremos para almacenar colores, seguidas de las variables de tiempo, la variable *done*, y *tecla*, usada para decirle a otras funciones la última tecla pulsada. La variable *comida* indica el número de manzanas que hay en pantalla. Seguido de esto vienen las 2 matrices pantalla y cola, y la definición de la estructura serpiente, que almacena sus coordenadas (de cabeza y cola), la dirección, los puntos, etc.
Por último, tenemos las estructuras de tipo *timeval* que se usarán para tomar diferencias de tiempo, seguidas de las declaraciones de funciones del programa.

FICHERO XSNAKE.C

El programa comienza con la función *main()*, en la cual se definen bastantes variables. Vamos a comentar sólo el código más importante (ver Listado 1).
Mediante *XSetNormalHints* y la estructura *XSizeHints* que se le pasa como parámetro, podemos especificar las coordenadas (x,y) en que queremos que aparezca nuestra ventana. Para ello lo que hacemos es rellenar los campos *myhints.x* y *myhints.y* con los valores de posición deseados, e indicar en *myhints.flags* que deseamos modificar la posición inicial de la ventana (*flag PPosition*). En este caso, además vamos a hacerla aparecer en el centro de la pantalla. Para ello mediante *XGetGeometry()* obtenemos la anchura y altura de la ventana raíz, esos valores los dividimos por 2 para obtener las coordenadas del centro de la pantalla, y le restamos la anchura y altura de nuestra ventana dividida por 2, para poner el píxel central de nuestra ventana en el centro de la pantalla.

Siempre hay que utilizar el menor número de variables globales posible

Otra de las funciones de esta porción de código es que nuestra ventana no se pueda redimensionar, es decir, que mediante el ratón no se pueda hacer más grande ni más pequeña. Para ello se rellenan los campos *max_width*, *min_width*, *max_height* y *min_height* de la estructura *myhints*, con los valores de anchura y altura que deseamos para la ventana de nuestra aplicación, y seleccionamos los *flags PMinSize* y *PMaxSize* de *myhints.flags*, antes de llamar a *XSetNormalHints()*.
A continuación se crea un *pixmap* (*XCreatePixmap*) con el mismo tamaño que la ventana creada, así como un GC (contexto gráfico) que usaremos para especificar posteriormente colores en las funciones de texto, y operaciones en las de copiado de mapas de bits, como unas líneas más abajo donde mediante *XFillRectangle* se dibuja un rectángulo relleno negro, que borra el *pixmap* recién creado:

```
imagen=Imlib_load_image(id, "xsnake0.jpg");
Imlib_render( id, imagen, ANCHURA_BLOQUE,
ALTURA_BLOQUE );
sprites[0] = (Pixmap) Imlib_move_image( id,
imagen );
imagen=Imlib_load_image(id, "xsnake1.jpg");
Imlib_render( id, imagen, ANCHURA_BLOQUE,
ALTURA_BLOQUE );
sprites[1] = (Pixmap) Imlib_move_image( id,
imagen );
        (etc...)
```

Mediante *Imlib_load_image()* se carga una imagen de formato JPEG en la estructura imagen declarada al principio del programa, se renderiza mediante *Imlib_render()* a un *pixmap* interno de determinado tamaño (*ANCHURA_BLOQUE*ALTURA_BLOQUE*), y se mueve mediante *Imlib_move_image()* al *pixmap* desde el cual lo usaremos nosotros. Esto se hace para cada imagen que se desea cargar desde disco, guardando luego cada una de ellas en uno de los elementos del vector de *pixmaps* llamado *sprite*. Posteriormente usaremos estos *pixmaps* para dibujar todos los elementos de juego en la pantalla, y gracias a *Imlib_render()*, estos elementos tendrán el tamaño apropiado, ya que al especificarle la anchura y altura deseadas, la librería se encarga de reducir o ampliar la imagen del disco para adaptarla a las dimensiones dadas. También convertirá el *pixmap* a la profundidad de color de la pantalla, de modo que nosotros no tendremos que realizar conversión alguna, disponiendo tras estas 3 llamadas de los *pixmaps* preparadas para su volcado en *sprites[4]*. Como se puede ver en el código, la manzana es renderizada a 2*2 cuadritos, multiplicando por 2 la anchura y altura a la que renderizarla.
Por último, antes de entrar en el bucle de eventos, se establece la semilla para números aleatorios como una función del tiempo mediante *srand(time(NULL));*, que para aquellos que hayan programado en MSDOS, es el equivalente a un *randomize();*.

EL BUCLE DE EVENTOS

El bucle de eventos está encuadrado dentro de un *while()* que abarca a todo el programa, Listado 2.
Como se puede ver en el pseudocódigo, sólo recogemos los eventos si hay alguno, de modo que nuestra aplicación no se quedará parada esperando eventos, sino que repetirá constántemente el bucle *while(!done)*. Las funciones *LeerMovimiento*, *MoverSerpiente*, etc., se ejecutan una vez por cada paso del bucle durante todo el programa, a menos que en alguna de ellas se haga *done=1*, con lo que se saldría del juego. Más interesante es la función *parar()*. En nuestro programa, *parar()* será una función que detendrá nuestra aplicación un número determinado de microsegundos, sin realizar ninguna otra acción y sin ocupar para nada la CPU.
Pero el uso de la función *parar()* tiene una función más importante: limitar la velocidad de funcionamiento del juego.

Listado 1. Función main()

```
Display *display;
Window window, rootwin;
int pant, anchura, altura;
XSetWindowAttributes attr;
XSizeHints myhints;

display = XOpenDisplay( NULL );
XGetGeometry( display, XDefaultRootWindow(display), &rootwin,
    &pant, &pant, &anchura, &altura, &pant, &pant );

id = Imlib_init(display);
window = XCreateWindow(display,
            DefaultRootWindow(display),
            (anchura/2)-((ANCHURA_BLOQUE*(xmax+1))/2),
            (altura/2)-((ALTURA_BLOQUE*(ymax+1)+40)/2),
            ANCHURA_BLOQUE*(xmax+1), ALTURA_BLOQUE*(ymax+1)+40,
            0, id->x.depth, InputOutput, id->x.visual, 0, &attr);

myhints.min_width =
myhints.max_width = ANCHURA_BLOQUE*(xmax+1) ;
myhints.min_height =
myhints.max_height = ALTURA_BLOQUE*(ymax+1)+40 ;
myhints.x = (anchura/2)-((ANCHURA_BLOQUE*(xmax+1))/2);
myhints.y = (altura/2)-((ALTURA_BLOQUE*(ymax+1)+40)/2);
myhints.flags = PMinSize | PMaxSize | PPosition;
XSetNormalHints( display, window, &myhints );
```

Necesitamos limitar la velocidad de alguna forma para que sólo se realicen una serie determinada de pasos del bucle por segundo. Si, por ejemplo, deseamos que la serpiente avance 10 recuadros por segundo, necesitamos que la función *parar* detenga la ejecución del programa durante 1.000.000/10 = 100.000 microsegundos. Esto hará que cada vez que se llegue a la función *parar(100.000)*, la ejecución se detenga durante este tiempo, transcurrido el cual se continuará, de modo que durante un segundo este bucle se ejecutará 10 veces, así las funciones *MueveSerpiente*, etc., también lo harán 10 veces y se limitará la velocidad de funcionamiento del programa:

```
while ( !done )
{
  gettimeofday(&start, NULL);

  while (QLength(display) > 0)
  {
    XNextEvent(display, &evento);
    if (evento.xany.window != window) continue;

    switch(evento.type)
    {
     case Expose:
        RedibujaPantalla(display, window, gc,
pixmap,
              evento.xexpose.x, evento.xexpose.y,
              evento.xexpose.width,
evento.xexpose.height );
        XFlush(display);
        break;

     case KeyPress:
        key = XKeycodeToKeysym(display,
evento.xkey.keycode, 0);
        if( evento.xkey.keycode == ESC_KEY )
          done = 1;
        tecla = evento.xkey.keycode;
        break;

    }
  }
  if( tecla != 0 )
      LeerMovimiento ( tecla, &serpiente1 );

  GeneraComida( display, pixmap, window, gc
);
  Movimiento( display, pixmap, window, gc,
&serpiente1 );
  DibujaSerpiente(display, pixmap, window, gc,
&serpiente1 );
  BorraSerpiente(display, pixmap, window, gc,
&serpiente1 );

  /* coger el tiempo actual */
  gettimeofday(&end, NULL);
  /* calculamos el tiempo que ha pasado desde el
anterior  fotograma, para poder cronometrar el
estado del juego */
  if( end.tv_usec < start.tv_usec )
     end.tv_usec += 1000000;
  time_dif = end.tv_usec - start.tv_usec;

  XSync(display, False);
  usleep(VELOCIDAD-time_dif);

  /* implementamos un cronómetro de useg a
segundos */
  tiempo += VELOCIDAD;
  if( tiempo > 1000000 )
  {
    tiempo -= 1000000;
    segundos++;
  }
}
```

La función *QLength(display)* es la encargada de decirnos cuántos eventos quedan en la cola, mediante el bucle while(*QLength(display) > 0*) estamos recogiendo todos los eventos de la cola o, lo que es lo mismo, estamos repitiendo la recogida de eventos hasta que la longitud de la cola de eventos sea cero.
A continuación se recogen los eventos y se actúa según el tipo, bien redibujando la pantalla (*Expose*) o recogiendo la pulsación del teclado detectada (*KeyPress*), dejándola en la variable tecla para que posteriores funciones (*LeerMovimiento()*):

```
gettimeofday(&start, NULL);
bucle_de_eventos();
funciones_del_juego();
gettimeofday(&end, NULL);
```

Las estructuras timeval se pueden usar para calcular diferencias de tiempo

Lo que hacemos con *gettimeofday()* es tomar el tiempo en microsegundos que ha tardado en ejecutarse el bucle de recogida de eventos y las funciones:

```
if( end.tv_usec < start.tv_usec )
   end.tv_usec += 1000000;
time_dif = end.tv_usec - start.tv_usec;
usleep(VELOCIDAD-time_dif);
```

Si debemos esperar 100.000 microsegundos por cada fotograma y en un ordenador lento, por ejemplo, se han tardado 10.000 microsegundos en ejecutar la porción de código dedicada a recoger eventos y dibujar en pantalla, el tiempo que debemos esperar es, realmente, de 100.000-10.000 = 90.000 microsegundos. Este es el motivo del uso de *gettimeofday()* y de los cálculos realizados antes

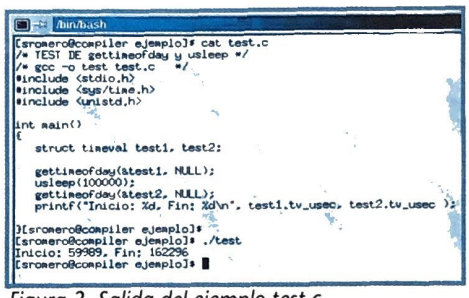


Figura 2. Salida del ejemplo test.c.

del *usleep()*, que es el equivalente a la función *parar()* que hemos

inventado en el pseudocódigo.
Finalmente tenemos un pequeño trozo de código encargado de contar el tiempo transcurrido desde el inicio del programa:

```
tiempo += VELOCIDAD;
if( tiempo > 1000000 )
{
   tiempo -= 1000000;
   segundos++;
}
```

Mediante XSetNormalHints podemos fijar las coordenadas de la ventana del programa

Para ello, a la variable *tiempo* le sumamos el *tiempo* que hemos esperado, sabiendo que cuando tiempo sobrepase el millón de usegundos, habrá pasado un segundo.
La porción de código fuera del bucle principal simplemente libera algunos recursos, muestra algunos mensajes y sale del programa.

FUNCIONES GENÉRICAS DE XSNAKE

La mayoría de las funciones restantes se dedican a modificar las variables del programa, o bien a dibujar en pantalla y en el *pixmap*.
Otras funciones interesantes son *EliminaCola()* y *BorraSerpiente()*, que son las encargadas de borrar la cola de la serpiente, en función de un recuadro de cola por cada avance de la serpiente:

```
void BorraSerpiente( Display *display, Pixmap
pixmap, Window window, GC gc,  struct serpiente
*serp )
{

 if ( serp->espera == 0 )
   EliminaCola( display, pixmap, window, gc, serp
);
 else
    serp->espera = serp->espera - 1;
}
```

Como puede verse, la función *BorraSerpiente*, sólo borra el último recuadro de la serpiente, si la

variable *espera* es igual a cero. En caso contrario, simplemente la decrementa. La utilidad de esto es muy sencilla: sirve para hacer crecer la cola cuando nosotros queramos.
Si procede borrar la cola, se hace:

```
void EliminaCola( Display *display, Pixmap
pixmap, Window window, GC gc, struct serpiente
*serp )
{
 int direccion;

   direccion = cola[serp->colaserx][serp->colasery];
   DibujaBloque( display, pixmap, window, gc,
            serp->colaserx, serp->colasery, VACIO );

   pantalla[serp->colaserx][serp->colasery] =
VACIO;
   cola[serp->colaserx][serp->colasery] = VACIO;
   if( direccion == DIR_IZQ )
     serp->colaserx = (serp->colaserx)-1;
   else if( direccion == DIR_DER )
     serp->colaserx = (serp->colaserx)+1;
   else if( direccion == DIR_ARR )
     serp->colasery = (serp->colasery)-1;
   else if( direccion == DIR_ABJO )
     serp->colasery = (serp->colasery)+1;
}
```

Lo que se hace en esta función es tomar la dirección de la serpiente en la posición *(cola_x,cola_y)*, borrar tanto en pantalla como en el *array* el último recuadro de la serpiente (*pantalla[cola_x][cola_y]*) y modificar estas 2 variables para que apunten a la nueva cola. Esta nueva cola es uno de los 8 recuadros adyacentes a la anterior cola.
La función *DibujaComida()* simplemente realiza 100 intentos de encontrar un hueco para la manzana a situar, comprobando que los 4 huecos necesarios (x,y, x+1,y, x,y+1 e x+1,y+1) estén libres, situando después la manzana en el *array* y dibujándola en la pantalla mediante *XCopyArea*.

Mediante XSizeHints es posible hacer que nuestra ventana no sea redimensionable

La función de borrado de la manzana debe detectar por qué lado de la manzana hemos chocado para realizar el borrado correcto tanto en pantalla como en la matriz de elementos. Para ello, según el valor de *pantalla[x][y]* borraremos los cuadros apropiados de alrededor. Por último, el núcleo del juego está prácticamente en la función de movimiento, donde se cambian las coordenadas de la serpiente en función de la dirección que ésta lleva, se testea si ha chocado con comida y se comprueba la muerte de la misma:

```
void Movimiento( Display *display, Pixmap
pixmap, Window window, GC gc, struct serpiente
*serp )
{
 int dircola, cabserantx, cabseranty;
 char dato;

 /* guardamos la cabeza antes de hacer
modificaciones */
 cabserantx = serp->cabserx;
 cabseranty = serp->cabsery;

 if ( serp->izquierda == true )
 {
  if ( serp->cabserx > xmin+1 )
  {
    serp->cabserx = (serp->cabserx)-1;
    dircola = DIR_IZQ;
  }
  else done = 1;
 }

 else if ( serp->derecha == true )
 {
  if ( serp->cabserx < xmax-1 )
  {
    serp->cabserx = (serp->cabserx)+1;
    dircola = DIR_DER;
  }
  else done = 1;
 }

  else if ( serp->arriba == true )
 {
  if ( serp->cabsery > ymin+1 )
  {
    serp->cabsery = (serp->cabsery)-1;
    dircola = DIR_ARR;
  }
  else done = 1;
 }

 else if ( serp->abajo == true )
 {
  if ( serp->cabsery < ymax-1 )
  {
    serp->cabsery = (serp->cabsery)+1;
    dircola = DIR_ABJO;
  }
  else done = 1;
 }

/* chequear choque con serpiente */
 if( pantalla[serp->cabserx][serp->cabsery] ==
SERPIENTE )
    done = 1;

 else
 {
  /* chequear si hemos cogido la comida */
  if( pantalla[serp->cabserx][serp->cabsery] >=
COMIDA1 )
   {
    BorraComida( display, pixmap, window, gc,
            serp->cabserx, serp->cabsery );
    comida—;
    serp->espera = serp->espera +
INCREMENTO;
    serp->puntos = serp->puntos + 10;
    ActualizaPuntos(display, pixmap, window,
gc, serp );
   }
   pantalla[serp->cabserx][serp->cabsery] =
SERPIENTE;
   cola[cabserantx][cabseranty] = dircola;
 }
}
```

FUNCIONES GRÁFICAS

Para el dibujado de las formas gráficas en pantalla, se utilizan funciones básicas de *BitBlt* de Xlib, las cuales simplemente toman el *pixmap* que se le pasa como parámetro y lo vuelcan en las coordenadas deseadas del pixmap destino. Las funciones son las siguientes:

```
/*— Función que redibuja la pantalla necesaria
————*/
void RedibujaPantalla( Display *display, Drawable
drawable,
 GC gc, Pixmap pixmap, int x, int y, int anchura,
int altura )
{
 XCopyArea(display, pixmap, drawable, gc,
        x, y, anchura, altura, x, y);
}

/*— Funcion que dibuja un bitmap ————
————————*/
void DibujaBitmap( Display *display, Drawable
drawable,
            GC gc, Pixmap pixmap, int x, int y )
{
  XCopyArea(display, pixmap, drawable, gc,
        0, 0, ANCHURA_BLOQUE,
ALTURA_BLOQUE, x, y);
}

/*— Dibuja un bloque ——————————
————————*/
void DibujaBloque( Display *display, Drawable
pixmap,
            Drawable window, GC gc, int x, int y,
            char tipo )
{
 DibujaBitmap( display, pixmap, gc, sprites[tipo],
            x*ANCHURA_BLOQUE,
y*ALTURA_BLOQUE);
 DibujaBitmap( display, window, gc, sprites[tipo],
            x*ANCHURA_BLOQUE,
y*ALTURA_BLOQUE);
}.
```