Allegro

Una librería para
programar videojuegos

Primer paso: Incluir la librería

```
#include<allegro.h>

int main()
{
    allegro_init();
    ...
    ...
    ...
    ...
    ...
    ...
    allegro_exit();
}
END_OF_MAIN();
```

Como para cualquier librería que fuéramos a usar

En este punto estamos llamando a la librería, todavía hasta acá no está haciendo nada pero, desde acá podemos empezar a utilizar las funciones de *allegro.*

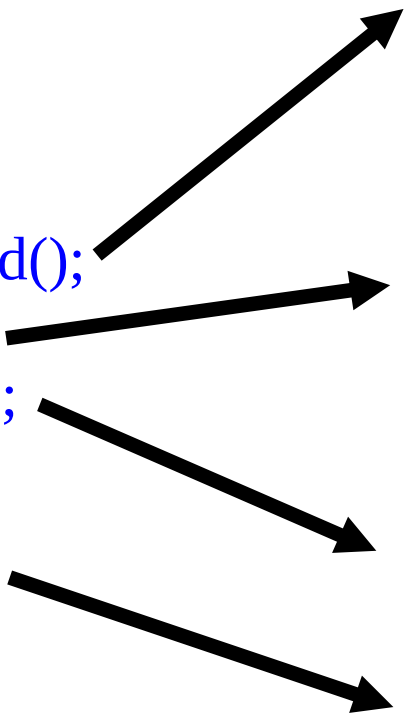
El cuerpo de nuestro programa, cualquier función de *allegro* que utilicemos tiene que ir en este espacio, o generará un error.

Aquí cerramos la librería, llamamos esta función porque ya la vamos a utilizar más

Receta de cocina, necesario para que funcione en Unix/Linux, pero no afecta el código en Windows.

Instalar los periféricos

```
#include<allegro.h>

int main()
{
    allegro_init();
    install_keyboard();
    install_timer();
    install_mouse();
    ...
    ...
    ...
    allegro_exit();
}
END_OF_MAIN();
```



Le entrega al *allegro* el control del teclado, lo que nos permitirá utilizar sus funciones para capturar datos.

Instala los temporizadores del *allegro*, funciones para manejar el control del tiempo, necesario para el mouse.

Le da al *allegro* el control del mouse esto nos permitirá utilizar las rutinas que posee para controlarlo.

Cualquier función de *allegro* que utilice el mouse, el teclado o los temporizadores, debe estar despues de los install_* y antes del allegro_exit();

Nota: Si tratamos de utilizar cualquier función de acceso a los periféricos del c/c++ después de entregarle el control al allegro generaremos un error grave

Abrir el modo gráfico

```
#include<allegro.h>

int main()
{
      allegro_init();
      install_keyboard();
      install_timer();
      install_mouse();
      set_color_depth(8);
      set_gfx_mode(GFX_AUTODETECT,800,600,0,0);
      ...
      ...
      ...
      allegro_exit();
}
END_OF_MAIN();
```

Hasta este punto, hemos empezado utilizado a algunas funciones de *allegro*, pero aún no se muestra nada nuevo en pantalla.

Especifica la profundidad de color a la que deseamos trabajar (8,16,32 bits), la profundidad por defecto es 8 bits, mientras más alta es más lento y más bonito.

Inicia el modo gráfico

Receta de Cocina

Resolución de la pantalla que deseemos abrir (800x600 pixels) *allegro* permite abrir cualquier resolución que soporte la tarjeta de video.

El driver de la tarjeta de video, "GFX_AUTODETECT" para que *allegro* la detecte.

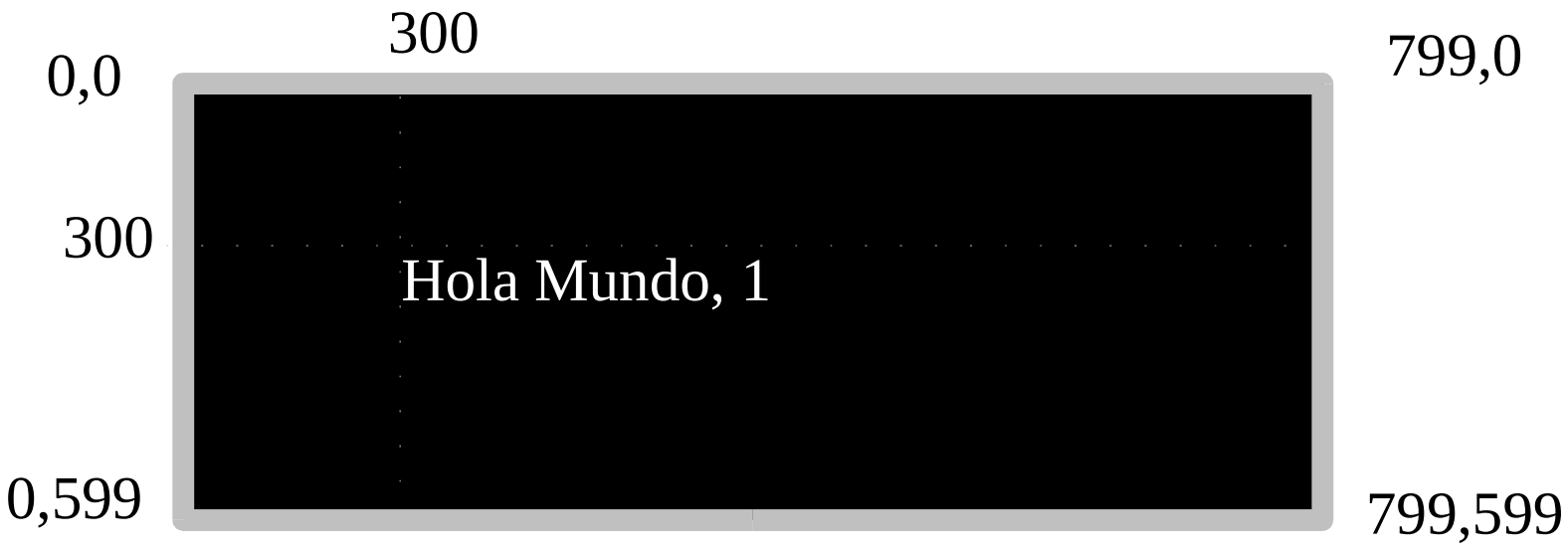
Imprimir texto

Estas funciones me permiten imprimir variables y texto en un *Bitmap* (¿Qué es eso? más adelante veremos, por el momento mirémoslo como la pantalla)

```
void textprintf(Bitmap* bmp, font f, int x, int y, int c, char* formato, var1, var2, ... , varn);
```

Imprime en *bmp*, con la fuente *f*, en la posicion (*x*,*y*) y del color *c* las variables *var1*, *var2*, ... *varn*.

Ej.
```
char* s1=”hola mundo”;
int c=1;
textprintf(screen,font,300,300,15,”%s , %d”,s1,c);
```



Imprimir texto

void textout(Bitmap* bmp, font f, char* cad,int x, int y, int c);

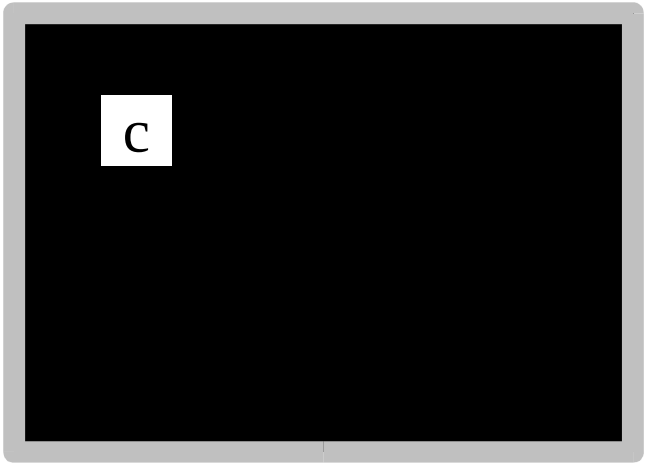
Imprime en *bmp*, con la fuente *f*, en la posicion (*x*,*y*) y del color *c* la cadena cad

Ej. textout(screen,font,”hola mundo”,100,100,15);

void text_mode(int c);

Define *c* como el color del fondo del texto a imprimir en pantalla, si se coloca el valor −1, se imprime transparente.

Ej. textmode(15);
textout(screen,font,” c ”,10,10,0);



Primitivas de dibujo

Allegro posee muchas funciones para dibujar primitivas de dibujo, que van desde puntos hasta poligonos.

```
void putpixel(Bitmap *bmp,int  x,int y,int  c);
```

Coloca un punto de color *c* en la posición *(x,y)* de bmp

```
int getpixel(Bitmap *bmp,int  x,int y);
```

Devuelve el número del color que hay en la posición *(x,y)* de bmp

```
int line(Bitmap *bmp,int  x1,int y1, int  x2,int y2, int  c);
```

Dibuja un segmento de línea, desde la posición *(x1,y1)* hasta la posición *(x2,y2)* de *bmp* con el color *c*.

```
int rect(Bitmap *bmp,int  x1,int y1, int  x2,int y2, int  c);
```

Dibuja un rectángulo, desde la posición *(x1,y1)* hasta la posición *(x2,y2)* de *bmp* con el color *c* en su borde y sin rellenar.

Primitivas de dibujo

int rectfill(Bitmap *bmp,int x1,int y1, int x2,int y2, int c);

Dibuja un rectángulo, desde la posición *(x1,y1)* hasta la posición *(x2,y2)* de *bmp* relleno con el color c, y sin borde.

void circle(Bitmap *bmp,int x,int y,int r,int c);

Dibuja un circulo de radio *r* con centro *(x,y)* de bmp y de color de borde c, sin relleno.

void circlefill(Bitmap *bmp,int x,int y,int r,int c);

Dibuja un circulo de radio *r* con centro *(x,y)* de bmp y de color de relleno c y sin borde.

int floodfill(Bitmap *bmp,int x,int y, int c);

Rellena con el color c, un área cerrada, empezando por x,y.

Teclado

Ahora veremos las funciones que presenta *allegro* para acceder al teclado, recordemos que para utilizarlas debemos llamar primero a install_keyboard()

`int readkey(void);` Detiene la ejecución del programa en espera de que presionen UNA sola tecla, devuelve el código ascii, de la tecla presionada, NO imprime en pantalla la tecla presionada.

Ej.
```
int c;
c=readkey();
textprintf(screen, font, 10,10, 15,”Tecla presionada=%c”,c);
```

`int keypressed(void);` Esta función devuelve 1 cuando se ha presionado una tecla y sigue, en este estado hasta que se limpie el buffer de teclado.

`int clear_keybuf(void);` Limpia el buffer del teclado

Teclado

Las funciones anteriores detenían la ejecución del programa, si lo que necesitamos es capturar alguna tecla sin parar la acción en el programa podremos utilizar el vector key[].

El vector key devuelve verdadero (0) si esta presionada la tecla que se le especifique en el subíndice, por ejemplo:

```
while(!key[KEY_ESC])
{
    if (key[KEY_ENTER])
        printf("Tecla enter");
    if (key[KEY_ESC])
        printf("Tecla escape");
    if (key[KEY_a])
        printf("Tecla a");
    if (key[KEY_F1])
        printf("Tecla F1");
}
```

Teclado

Si necesita capturar más de un carácter, es decir una cadena, y además desea aplicar restricciones será necesario componer una función, de escritura.

```
char* recibir_cadena(int x,int y,int color, int max)
{
char lt
do
{
    lt=readkey();
    if( lt>=’a’ && lt<=’z’)
    {
        textprintf(screen,font,x+12*i,y,color,”%c”,lt);
        cadena[i]=lt;
        cadena[i+1]=’\0’;
        i++;
    }
}while( lt!=13 && i<max );
return cadena;
}
```

Ratón

Allegro posee unas rutinas muy sencillas para el control del ratón, recuerda que para utilizarlas deberás haber llamado antes a install_mouse();

void show_mouse(Bitmap *bmp);

Muestra el cursor del ratón en el Bitmap bmp, por lo general screen, para quitar el mouse de bmp usar show_mouse(NULL).

void scare_mouse(void);

Esconde el cursor del ratón, aunque este siga estando allí, necesario cuando se va a dibujar algo en el mismo bitmap en que esta el mouse.

void unscare_mouse(void);

Muestra de nuevo el cursor del mouse

void position_mouse(void);

Lleva el mouse a la posición x,y

Ratón

void set_mouse_sprite(Bitmap *bmp);

Cambia el cursor del ratón (la flecha por defecto), por el bitmap que le mandemos.

void set_mouse_sprite_focus(int x, int y);

Especifica que posición del cursor, representará la posición actual del mouse.

void set_mouse_range(int x1, int y1, int x2, int y2);

Limita el movimiento del ratón al recuadro definido por x1,y1 -> x2,y2

void set_mouse_speed(int x, int y);

Especifica la velocidad del ratón, el valor por defecto es 2 para ambos, valores mayores implican un movimiento más lento.

Ratón

```
int mouse_x;
int mouse_y;
```

Variables globales que contienen en todo momento, la posición del ratón

```
int mouse_b;
```

Variable global que guarda el estado de los botones del ratón, para acceder a ellos se necesitará la comparación bit a bit, asi:

mouse_b&1 −> Verdadero si el botón izquierdo está presionado
mouse_b&2 −> Verdadero si el botón derecho está presionado
mouse_b&4 −> Verdadero si el botón central está presionado

Ej.

```
if (mouse_b &1)
    textprintf(screen,font,10,10,15,”Botón izquierdo”);
```

Bitmaps

Los BITMAP *, son variables que representan un rectángulo, sobre el que puedo dibujar cosas, para allegro la pantalla (screen) es un BITMAP*, y para toda función que dibuje algo hay que específicarle en que BITMAP, deseamos que lo haga.

Sin embargo, el hecho que creemos un BITMAP* y dibujemos sobre el no implica que esto último se muestre en la pantalla, sino que quedará almacenado en la memoria para poder usarlo después.

```
BITMAP * bmp;
```

Declara a bmp como una variable de tipo Bitmap.

```
BITMAP * create_bitmap(int x, int y);
```

Reserva memoria para un bitmap de tamaño x,y, para poder dibujar sobre un bitmap hay que utilizar esta función primero.

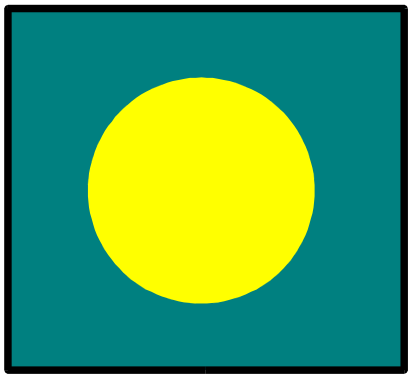
```
void destroy_bitmap(Btimap * bmp);
```

Libera la memoria que tenía reservada un bitmap.

Ej.

```
BITMAP *primero;
primero=create_bitmap(50,50);
circlefill(primero,25,25,15,12);
destroy_bitmap(primero);
```

Crea un bitmap “primero”, le reserva un tamaño de 50x50 pixels y le dibuja un círculo relleno, luego libera el bitmap, sin embargo, en la pantalla no aparece nada en ningún momento.



primero

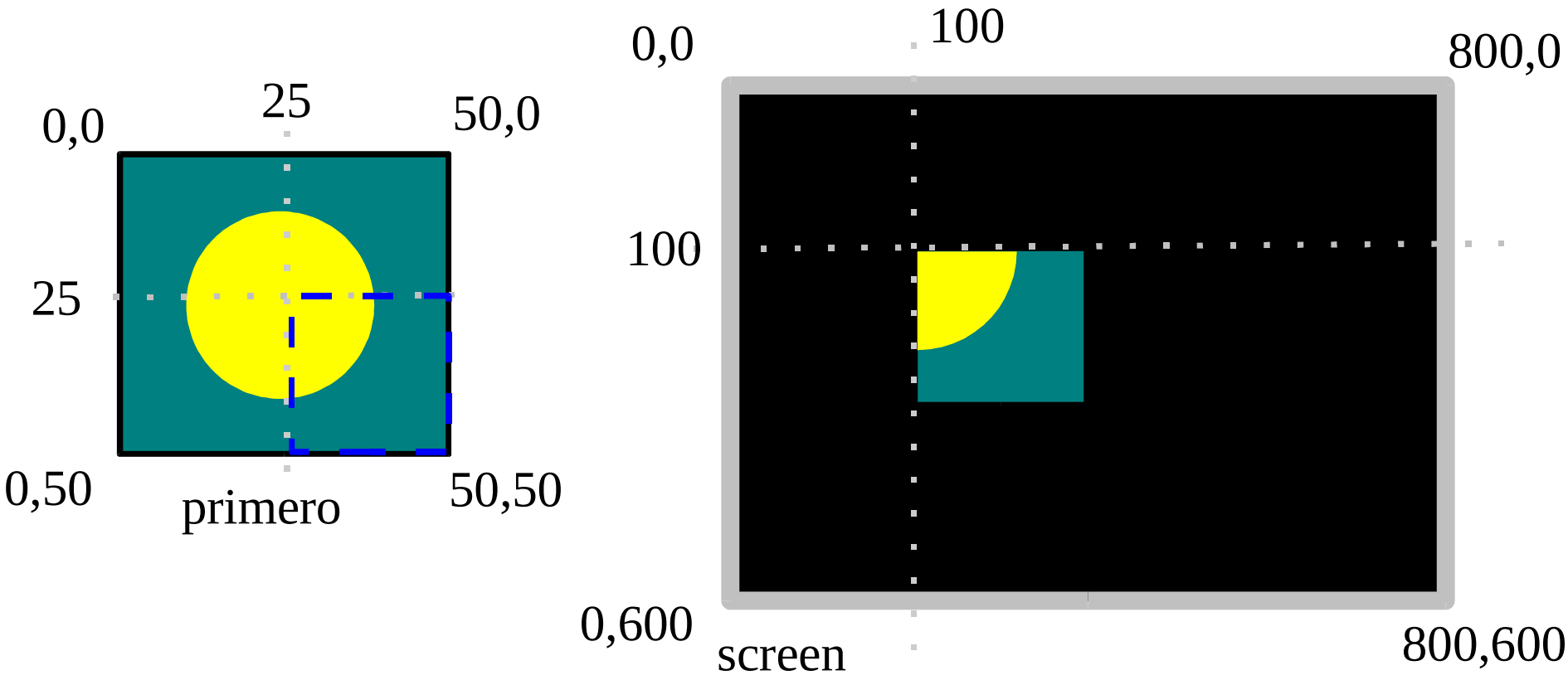
screen

Void blit(BITMAP* origen, BITPMAP* destino, int xorigen, int yorigen, int xdestino, int ydestino, int tamax, int tamy);

Dibuja del bitmap “origen” en la posición xorigen,yorigen, al bitmap destino en la posición xdestino, ydestino, un recuadro de tamaño tamax*tamay

Retomando el ejemplo anterior

blit(primero,screen,25,25,100,100,25,25);



void clear(BITMAP* bmp);

Limpia el bitmap “bmp” al color 0, negro a menos que se cambie.

void clear_to_color(BITMAP* bmp,int c);

Limpia el bitmap bmp al color c.

void draw_sprite(BITMAP* destino,BITMAP* sprite, int x, int y);

Dibuja el bitmap sprite en la posición x,y, pero dibujando el color, 0 como transparente (esto es para 8bits, en 16bits o más es el rojo fuccia, rojo y azul al máximo, verde en 0).

void stretch_blit(BITMAP* origen, BITPMAP* destino, int xorigen, int yorigen,int tamxorigen, int tamyorigen, int xdestino, int ydestino, int tamaxdestino, int tamydestino);

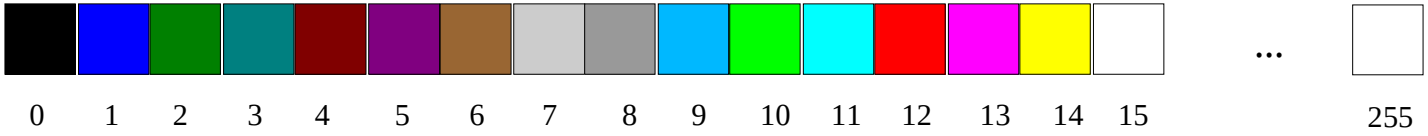
Lo mismo que el blit, pero permite cambiarle el tamaño al bitmap destino.

void rotate_sprite(BITMAP* destino, BITPMAP* sprite, int x, int y, fixed angulo);

Dibuja un sprite pero rotado "angulo", para especificar el ángulo se puede utilizar la función itofix(int a), a la que si le enviamos 255 como parametro estaremos hablando de un giro completo.

Paleta

Cuando el Allegro esta trabajando con una profundidad de color de 8 bits utiliza una paleta para dibujar los colores que se muestran en pantalla, la paleta no es más que un vector de colores, donde cada color tiene un índice asignado.



Cuando creamos un bitmap este en realidad no guarda colores, sino índices de colores y cuando le pedimos al allegro que los dibuje, este cambia esos índices por el color que contenga la paleta en esa posición.

Nosotros podemos crear nuestras propias paletas, si queremos crear nuevos colores o para cargar imágenes, sin embargo el Allegro solo empezará a utilizarla cuando se lo digamos.

Para crear una paleta declaramos una variable de tipo PALETTE, asi:

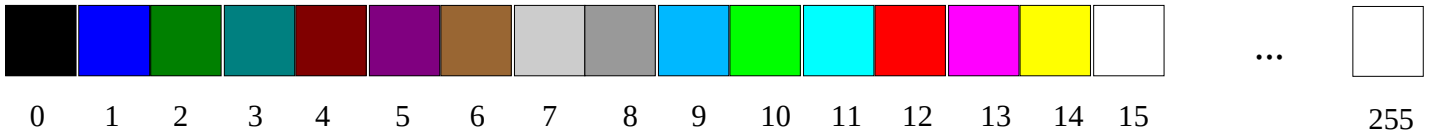
```
PALETTE paleta;
```

Para decirle al allegro que la utilice necesitamos usar la siguiente función:

```
void set_palette(PALETTE pal)
```

Si utilizamos una nueva paleta, todas las cosas que hallamos dibujado en pantalla, cambiarán de color, al que referencia la nueva paleta.

Ej: Si tenemos la paleta



Y la cambiamos por

Y habíamos dibujado en pantalla la imagen

Ahora se verá

Porque aunque la imagen no cambió, si lo hicieron los colores 0 y 1 que referenciaba

Cargar Imágenes

Allegro permite cargar imágenes en formato BMP, TGA y PCX, para hacer esto podemos utilizar la función:

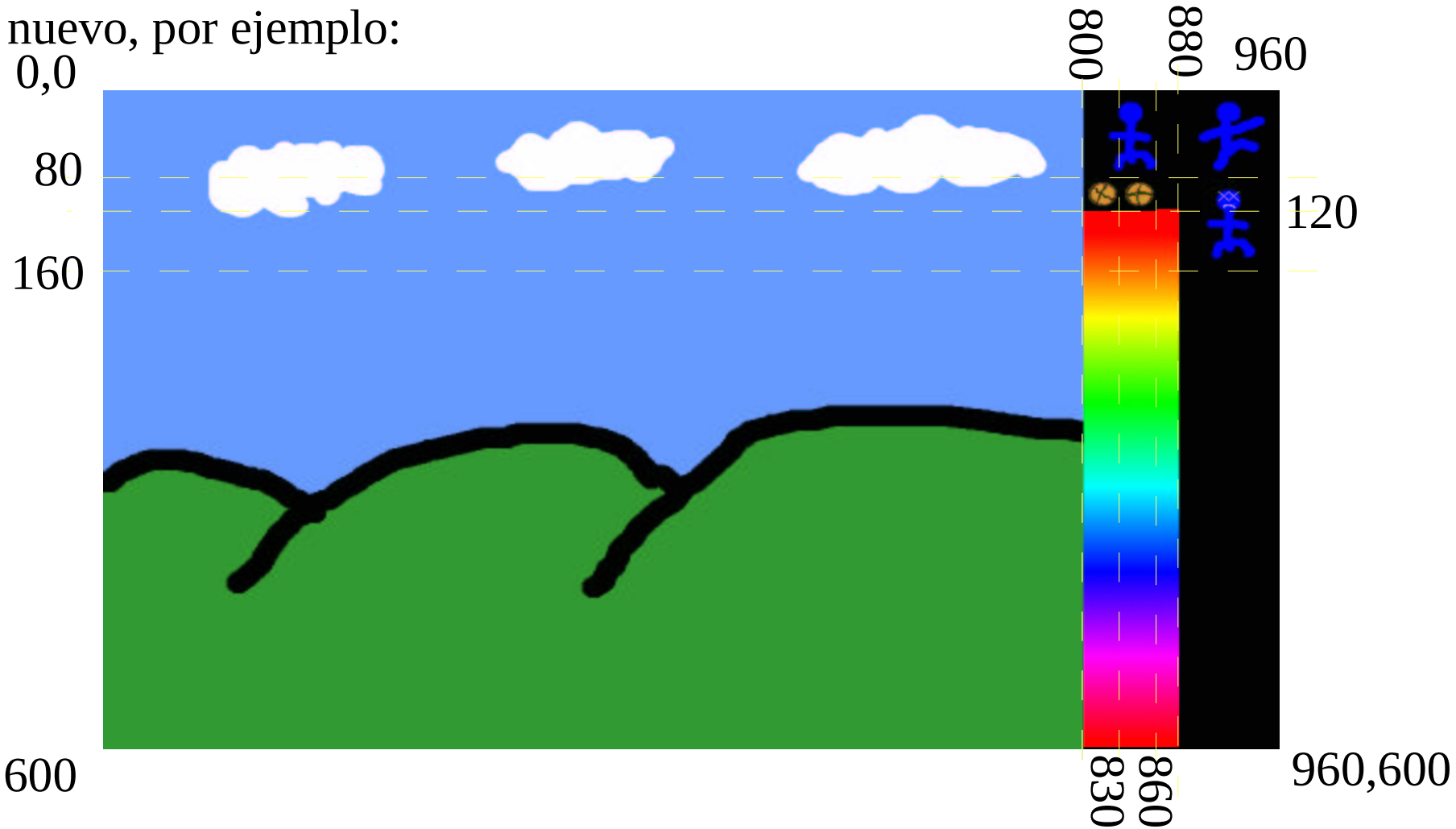
```
BITMAP* load_bitmap(char * arch, PALETTE pal);
```

Que crea y retorna un bitmap, cargado desde el archivo arch, y con la paleta pal.
Es importante resaltar que no es necesario utilizar create_bitmap si se usa esta función y que es necesario decirle al allegro que utilice la paleta de la imagen, ya que cada imagen utiliza su propia paleta, que está optimizada para mostrar lo que la imagen contenga, por ejemplo una imagen de un bosque seguramente tendrá muchos tonos verdes y café, y pocos rojos o azules.

Ej:

```
BITAMP *completo;
PALETTE paleta;
completo=load_image(“completo.tga”,paleta);
set_palette(paleta);
```

Cuando en un juego necesitamos cargar muchas imágenes entramos en un problema, porque cada imagen tiene su propia paleta, y que muy seguramente serán diferentes.
Para solucionar éste problema, podemos hacer dos cosas, trabajar en 16 bits o si queremos conservar la velocidad de los 8 bits, podemos combinar todas las imágenes que necesitemos en una sola, y dentro del programa dividirlas de nuevo, por ejemplo:

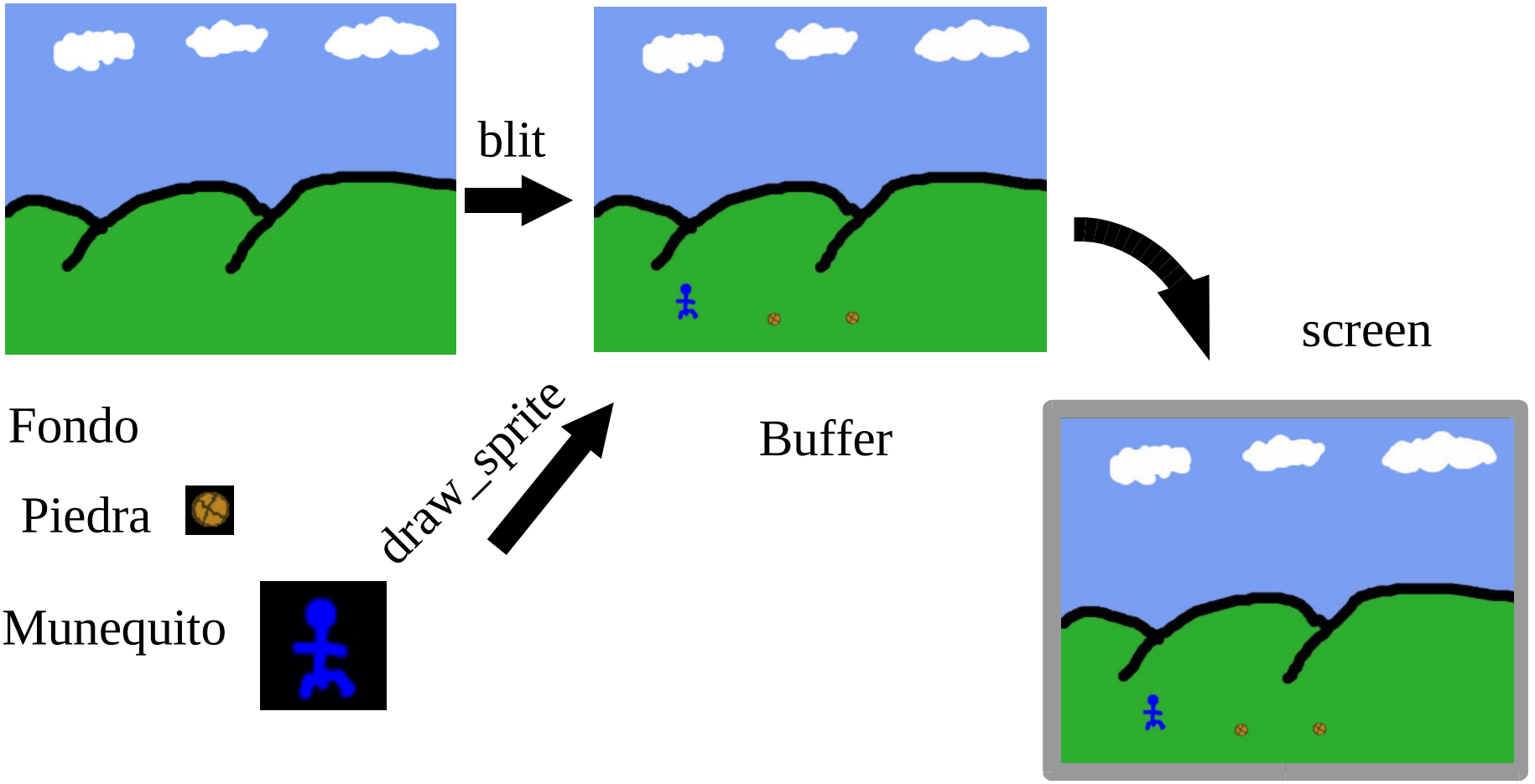


En código sería

```
#include<allegro.h>

BITMAP* completo;
BITMAP* mune;
BITMAP* piedra;
BITMAP* fondo;
PALETTE paleta;

void main()
{
	allegro_init();
	install_keyboard();

	set_gfx_mode(GFX_AUTODETECT,800,600,0,0);

	completo=load_bitmap("completo.tga",paleta);
	set_palette(paleta);
	//la imagen completa
	fondo=create_bitmap(800,600);
	mune=create_bitmap(80,80);
	piedra=create_bitmap(30,30);
	//bitmaps para dividir la imagen
	blit(completo,fondo,0,0,0,0,800,600);
	//el fondo
	blit(completo,mune,800,0,0,0,80,80);
	//el mune
	blit(completo,piedra,800,80,0,0,30,30);
	//el primer cuadro de la piedra
	blit(fondo,screen,0,0,0,0,800,600);
	//dibujo el fondo
	draw_sprite(screen,mune,100,500);
	//dibujo el mune
	for(i=0;i<=4;i++)
	{
		draw_sprite(screen,piedra,300+i*50,550);
	}
	//dibujo las piedras

	readkey();
	destroy_bitmap(mune);
	destroy_bitmap(piedra);
	destroy_bitmap(fondo);
	destroy_bitmap(completo);
	allegro_exit();
}
END_OF_MAIN();
```

Doble Buffer

Una técnica que sirve para mejorar la eficiencia de nuestro programa se llama el doble buffer, consiste en dibujar todo lo que se vaya a mostrar primero en un bitmap que sirve de buffer, y luego volcar éste último a la pantalla, ya que dibujar en pantalla es demasiado lento, por lo que debemos evitarlo.



Temporizadores

Son funciones que nos permiten llevar dentro del programa una cuenta del tiempo que ha pasado, esto nos permite colocarle al juego una velocidad fija en cualquier máquina

```
void install_int(proc funcion, int tiempo);
```

Llama a la función "función", cada que pasen "tiempo" milisegundos, sin importar en donde se encuentre la ejecución del programa

Toda variable que se utilice dentro de la función deberá ser preferiblemente global y volátil, así:

```
volatile int  t;
```

La función que le enviemos al temporizador debe ser lo más sencilla posible y solo debe realizar cálculos de enteros, por ejemplo:

```
void tiempo()
{
        t++;
}
END_OF_FUNCTION(tiempo);
```

Y en el código se deberá incluir antes de llamar a install_int:

```
LOCK_VARIABLE(t);
LOCK_FUNCTION(tiempo);
```

Como utilizarlos?, haciendo que el ciclo principal del juego no se llame sino cada medio segundo por ejemplo, garantizamos que nuestro juego corre a 0,5 segundos por ciclo.

```
volatile int  t;

void tiempo()
{
	t++;
}
END_OF_FUNCTION(tiempo);

BITMAP* buf;

void main()
{
	allegro_init();
	install_keyboard();
	install_mouse();
	install_timer();
	set_gfx_mode(GFX_AUTODETECT,800,600,0,0);
	 buf=create_bitmap(800,600);
	 clear(buf);
	//instalamos el temporizador
	LOCK_VARIABLE(t);
	LOCK_FUNCTION(tiempo);
	t=0;
	install_int(tiempo,500);  //que cambie cada 0,5 seg
	int auxt; //tiempo auxiliar
	auxt=t;
	int x=0;
	 while(!key[KEY_ESC])  //el ciclo principal
	{
		if(aux!=t)  //Si ya avanzó el tiempo 0,5 seg.
		{
			//limpiamos el buffer
			clear(buf);
			//dibujamos un rectangulo en el buff
			rectfill(buf,x,10,x+100,110,12);
			//pasamos el buff a la pantalla
			blit(buf,screen,0,0,0,0,800,600);
			//volvemos a guardar el tiempo actual
			auxt=t;
			//para que se desplace el cuadro
			x=x+10;
		}
	}
	allegro_exit();
}
```