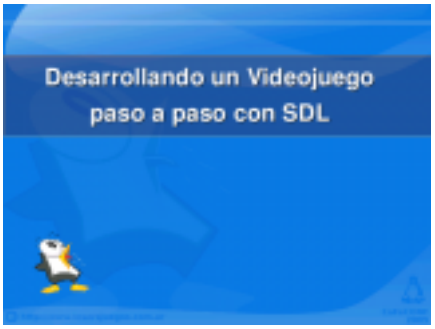
Como desarollar un videojuego paso a paso con SDL

Créditos

» **Autor:** Hugo Ruscitti / Gabriel Valentin
» **Fecha:** 23 de Octubre del 2005

Introducción

Las siguientes diapositivas fueron presentadas en el evento anual de Software Libre CaFeCONF 2005 en la ciudad de Buenos Aires (Argentina).



Mediante estas notas intentamos resumir y compartir con usted aquellos temas que pudimos tratar durante la exposición, si tiene alguna sugerencia por favor envienos un mensaje utilizando el formulario al pie de la página.

Juego base tomado como ejemplo:

Mediante esta charla buscamos seguir el desarrollo de un videojuego para a paso a paso con SDL y C.

Nuestro objetivo primordial es presentarles a los asistentes el programa en detalle para que todos tengan la posibilidad de analizar, comprender y luego, si quieren, lo modifiquen a gusto y/o realicen sus propios videojuegos.

Elegimos desarrollar un videojuego de Volley muy sencillo, similar al clásico Arcade Volleyball. Si bien el juego es muy simple, nos resultará de mucha utilidad como ejemplo ilustrativo, ya que todos los videojuegos (independientemente de su temática), requieren de una serie de caracteristicas:

» simular comportamientos.
» verificar colisiones.
» realizar animaciones.
» imprimir gráficos en pantalla.

Fases de implementación:

Para poder analizar de forma ordenada cada implementación, el juego se desarrolló siguiendo una serie de etapas.

En cada etapa añadimos nuevos requerimientos, por ejemplo, en la etapa 7 buscamos que los personajes puedan saltar y lo implementamos mediante un autómata

Cada una de las etapas tiene su propio programa de ejemplo, puede descargar todos los ejemplos juntos, o bien, descargarlos de manera individual (ver al final de cada sección)

Herramientas:

Para desarrollar el videojuego utilizamos el lenguaje de programación C. Creemos que C es uno de los lenguajes de programación que todos aprendemos en algún momento y por lo tanto será el mas adecuado para este tipo de propuestas.

Mediante el lenguaje de programación C podremos utilizar diferentes metodologías o técnicas de programación: 'lineal', 'estructural' o como en este caso 'modular'.

El objetivo de la técnica de programación modular consiste en representar a las entidades del programa (en este caso: una pelota, dos personajes etc.) mediante estructuras de datos y un conjunto limitado de funciones que nos permiten manipular dichas estructuras.

En los programas de ejemplo encontrará la codificación de cada entidad dividida en dos archivos, uno de extensión .c donde se codifica cada funcionalidad y otro archivo .h donde se especifican (declaran) las estructuras, constantes y funciones propias de la entidad.

Para nombrar y organizar el código intentamos relacionar cada función con una estructura en particular utilizando prefijos, por ejemplo, la función "pelota_actualizar" tiene el prefijo "pelota" por lo tanto está definida en "pelota.h" e implementada en "pelota.c".

Abordando el desarrollo del programa mediante técnicas y estilos de programación podemos dividir y facilitar cualquier tarea.

"Un gran problema puede dividirse en varios mas pequeños"

Para realizar las tareas mas básicas de impresión y manipulación gráfica utilizamos la biblioteca multimedia SDL, puede obtener más información en www.libsdl.org y en otros artículos (como ¿Por qué SDL?) del sitio www.losersjuegos.com.ar

Paso 1: Iniciar SDL y crear una ventana

Crear una ventana para nuestro juego es muy sencillo, el trabajo duro lo realiza SDL. Nosotros simplemente le ordenamos iniciar y crear una ventana a la biblioteca, especificando el tamaño de la pantalla y la profundidad de colores. No debemos preocuparnos por los detalles del sistema operativo, SDL se encargará de ello.

Cuando desarrollamos un programa seguimos una serie de fases antes de ponerlo en funcionamiento. Aunque muchas veces sea inmediato, siempre debemos planificar el orden de cada tarea, pensamos (o dibujamos sobre papel) un algoritmo; en la diapositiva se representa el orden de tareas mediante un diagrama de flujos (a la derecha). Luego transcribimos ese diagrama a un lenguaje de programación (ese es el proceso de implementación) y por último debemos compilar el programa.

» Descargar ejemplo 1 (código fuente)

Paso 1: Compilando el primer ejemplo

Un programa escrito en un lenguaje de programación, comprensible por un ser humano, no es inmediatamente ejecutable por una computadora, las computadoras solo ejecutan código máquina (compuesto de 0 (ceros) y 1 (unos)). Por esa razón es habitual utilizar un programa compilador para convertir ese código comprensible por nosotros al código que ejecuta una computadora.

Para compilar programas en GNU/Linux contamos con la colección de compiladores de GNU (gcc). Mediante el paso 1 (indicado en la diapositiva) transformamos nuestro programa "volleyball.c" en código objeto, dejándolo listo para el paso 2 que lo convierte en un programa ejecutable.

Se debe tener en cuenta que nuestro programa utiliza una biblioteca, SDL, por ese motivo debemos vincularla al programa en el último paso de compilación, de otra manera no tendremos el código necesario para realizar las tareas que SDL nos provee. Para vincular la biblioteca a nuestro programa añadimos la salida del comando sdl-config (note las comillas simples invertidas (``))

Para simplificar el proceso de compilación utilizamos el programa make, por lo tanto para compilar y enlazar los programas de ejemplo sólo tiene que ejecutar "make" en el directorio donde residen los archivos fuente (.c y .h) junto con el archivo "Makefile".

Si al momento de crear el programa ejecutable deseamos conocer sus dependencias podemos ejecutar:

```
ldd ./volleyball
```

y obtendremos el listado completo de bibliotecas dinámicas asociadas al programa "volleyball".

Paso 2: Imprimir el fondo de pantalla

En este ejemplo comenzamos a utilizar superficies. Para SDL toda imagen se representa mediante una superficie. Por ese motivo es tan simple combinarlas.

Comenzamos cargando la imagen desde "ima/fondo.bmp" sobre una superficie en memoria para luego volcarla sobre la pantalla principal.

Mediante SDL_LoadBMP cargamos ese archivo en memoria; ahora tendremos una nueva superficie en memoria apuntada por la variable "fondo" (puntero a SDL_Surface). Si SDL_LoadBMP no puede cargar la

imagen nos retorna un valor nulo (NULL), llamando a SDL_GetError podremos conocer el motivo del fallo:

```
fondo = SDL_LoadBMP ("ima/fondo.bmp");

if (fondo == NULL)
{
 printf ("Error al cargar fondo %s\n", SDL_GetError() );
 return 1;
}
```

Una vez almacenada la superficie, nos proponemos 'volcarla' sobre la pantalla principal (screen) utilizando SDL_BlitSurface, los parámetros de la función especifican las superficies fuente y destino; los valores NULL indican que buscamos imprimir las superficies completas. En los siguientes ejemplos veremos como imprimir sólo algunas regiones de cada superficie.

Por último, SDL protege que las impresiones en ventana no aparezcan inmediatamente en la pantalla, para que todas nuestras impresiones se puedan observar en la ventana tendremos que utilizar SDL_Flip.

```
SDL_BlitSurface (fondo , NULL, screen, NULL);
SDL_Flip (screen);
```

» Descargar ejemplo 2 (código fuente)

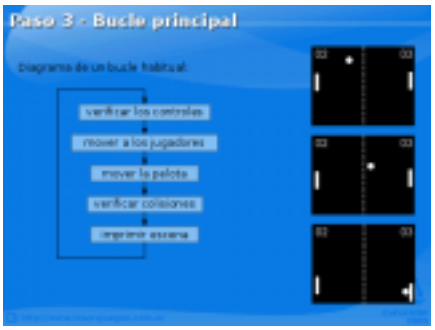
Paso 3: bucle principal

Existe una diferencia importante entre un videojuego y otro tipo de programas, por ejemplo un editor de textos.

Un editor de textos espera ordenes del usuario. Si el usuario pulsa una tecla, el editor imprime el carácter en pantalla y regresa a la espera; si otro evento ocurre, el editor ejecuta alguna tarea y continúa esperando.

En cambio un videojuego no podría esperar al usuario, al contrario, un videojuego generalmente no espera nunca, constantemente realiza verificaciones y ejecuta pequeñas acciones.

El diagrama nos presenta un esquema de funcionamiento para un juego como Pong.



El flujo de ejecución no se detiene y en cada repetición verifica el estado de los controles (rápido y sin detenerse), cambia la posición de los jugadores, mueve apenas la pelota, y luego imprime la escena completa.

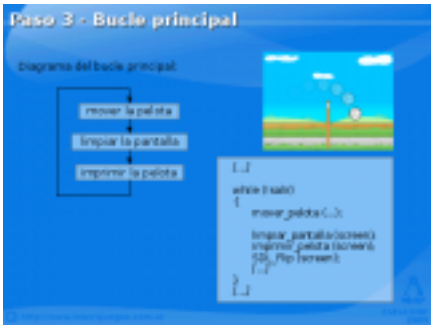
Si bien el ejemplo de esta diapositiva no tiene una relación directa con el juego de volley, en esencia es idéntico al bucle de muchos videojuegos.

Paso 3: Bucle principal

Retomando el ejemplo "volleyball", nuestro bucle es más sencillo, sólo tenemos un objeto en movimiento.

Para simular el movimiento de la pelota en pantalla, le indicamos a la biblioteca SDL que imprima el gráfico de la pelota en diferentes posiciones.

En definitiva, no estamos "moviendo objetos", simplemente realizamos sucesivas impresiones en pantalla. SDL desconoce la diferencia entre fondos de pantalla y objetos animados, para SDL todas son superficies.



En cada repetición del bucle principal debemos realizar un pequeño cambio de posición de la pelota. Ese cambio constante de posición se almacena en las variables 'x' e 'y' de la estructura "pelota" (pelota.c). A continuación el bucle principal imprime la pelota sobre la pantalla utilizando los valores 'x' e 'y' calculados con anterioridad.

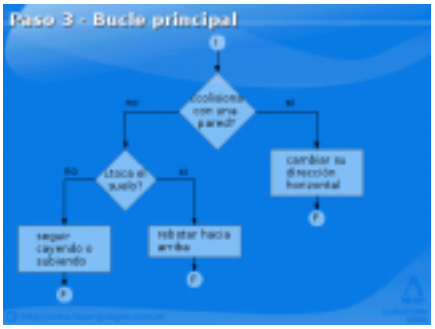
Paso 3: Bucle principal

El comportamiento de la pelota se ha implementado en el archivo "pelota.c":

```
void pelota_actualizar (struct Pelota * pelota)
{
 /* movimiento vertical (variable) */
 pelota->velocidad ++;
 pelota->y += (pelota->velocidad >> 2);

 /* movimiento horizontal (constante) */
 pelota->x += pelota->direccion * 3;

 /* limite derecho */
 if (pelota->x > 590)
  pelota->direccion = -1;

 /* limite izquierdo */
```

```
 if (pelota->x < 0)
  pelota->direccion = 1;

 /* toca el suelo */
 if (pelota->y > 375)
 {
  pelota->y = 375;

  /* velocidad inicial */
                pelota->velocidad = -50;
 }
}
```

como en otros casos, la diapositiva nos permite esquematizar el funcionamiento del programa.



» Descargar ejemplo 3 (código fuente)

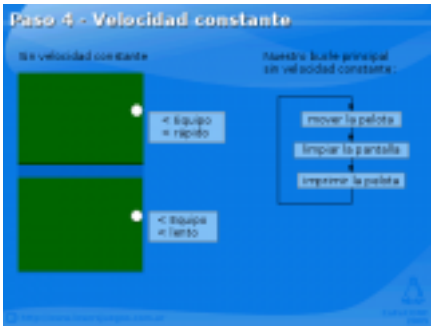
Paso 4 - Velocidad constante (1)

Nota: las diapositivas originales muestran una animación a modo de ejemplo ilustrativo, puede descargar las diapositivas originales al pie de página.

Cuando desarrollamos el videojuego en un equipo particular muchas veces no tomamos en cuenta la velocidad que podría tener nuestro programa en otros equipos.

Nuestro primer bucle principal no contempla esa necesidad, en un equipo lento, el juego funcionará a baja velocidad (sería muy aburrido), en cambio si se ejecuta en un equipo muy rápido sería imposible de jugar dada su velocidad.

A diferencia de otro tipo de programas, quisiéramos que nuestro juego funcione a la misma velocidad en diferentes equipos, independientemente de sus recursos.



Para ello podemos aplicar un algoritmo que gestione la velocidad del juego de manera independiente a la arquitectura del equipo o dispositivo. Una forma de lograr esa independencia consiste en controlar nuestro bucle mediante rutinas de "tiempo".

SDL cuenta con varias funciones para acceder al temporizador de sistema, y así permitirnos desarrollar rutinas independientes de la velocidad del equipo que las ejecuta.

Una de esas funciones es SDL_GetTicks, mediante esta función podremos averiguar cuanto tiempo a transcurrido desde la inicialización de la biblioteca. Utilizando esa función podremos calcular el tiempo transcurrido entre 2 instantes de nuestro programa:

```
antes = SDL_GetTicks ();
[...] // mucho código
ahora = SDL_GetTicks ();

printf ("Transcurrieron %d milisegundos\n", ahora - antes);
```

El efecto deseado es el de "saltar cuadros" cuando el equipo es muy lento, ya que buscamos compensar su baja velocidad.

Mientras que en un equipo rápido quisiéramos pausar el programa evitando consumir recursos.

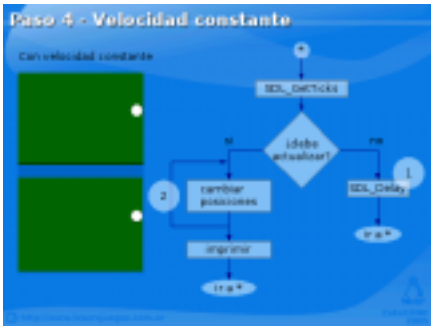
Paso 4 - Velocidad constante (2)

Pensando en un diagrama de flujos, en vistas a implementarlo en un lenguaje de programación, debemos distinguir entre las partes lógica y gráfica de un videojuego.

La lógica del juego consiste en todos los procedimientos que contabilizan el puntaje, actualizan la posición de los personajes, verifican colisiones y todos los cálculos relacionados. Esa es la parte invisible de un videojuego, y aquella que casi todos los equipos pueden procesar en muy poco tiempo.

La gráfica de un videojuego simplemente se encargara de volcar todas las imágenes en pantalla. Aunque parezca simple, ese paso sí es costoso para cualquier computadora (costoso en términos de velocidad).

Si nuestro equipo es muy lento, solo podremos ganar velocidad "evitando imprimir" algunos cuadros de animación.



En el diagrama de la derecha se indica como solucionar el problema de la velocidad constante. Si nuestro equipo es muy rápido (indicado con un 1 en la imagen) debemos detenernos por unos instantes con SDL_Delay, en cambio, si nuestro equipo es lento podremos "ganar" velocidad evitando imprimir la escena y repitiendo las rutinas lógicas del videojuego.

Paso 4 - Técnicas de impresión

Nos queda por resolver otro problema, la impresión de gráficos.

En el ejemplo 3 se observa que el gráfico de la pelota es inconsistente, se produce un molesto "parpadeo". Esta clase de problemas no suelen ser habituales cuando se utiliza la biblioteca SDL con superficies en memoria de sistema (SDL_SWSURFACE o SDL_HWSURFACE en ventana). Pero creemos conveniente tratar el asunto de todas maneras, ya que este problema suele surgir cuando se trabaja con otras bibliotecas o dispositivos.

Nuestro problema se produce debido a las sucesivas impresiones sobre la pantalla principal; no estamos moviendo ningún objeto, simplemente imprimimos el gráfico de la pelota en diferentes posiciones.

Cada impresión del gráfico de la pelota sobreescribe otra superficie, así se pierden muchos pixeles del fondo de pantalla, por lo tanto, si no restauramos el fondo de pantalla, la imagen de la pelota parecerá duplicarse en cada instante.

En el ejemplo anterior, para evitar la duplicación de gráficos, utilizamos un criterio muy simple: antes de imprimir el gráfico de la pelota en su nueva posición restauramos la imagen de fondo. Ahí está el problema, el usuario percibe en pantalla cada uno de nuestros pasos, por ese motivo el gráfico parece inconsistente.

Una solución a este problema consiste en ocultarle al usuario nuestro trabajo de restauración:

El primer esquema nos muestra una técnica llamada Double Buffer, donde realizamos todo el trabajo de impresiones parciales sobre una superficie invisible al usuario y cuando terminamos de completar la imagen

volcamos todo sobre la superficie principal.

Pero en algunos juegos como este, podemos ahorrarnos el trabajo de imprimir constantemente toda la pantalla, ya que nuestro juego no realiza muchos cambios en pantalla entre cuadro y cuadro.

Podremos, utilizando otra técnica llamada Dirty Rectangles, restaurar solamente aquellas regiones de pantalla que han sido modificadas.

En este ejemplo hemos añadido un archivo llamado "dirty.c" donde se encapsulan el manejo de rectángulos y la restauración de pantalla; el proceso es simple. Utilizamos un vector de estructuras llamadas SDL_Rect para almacenar las zonas de pantalla modificadas.

Por ejemplo, luego de imprimir la pelota, almacenamos el rectángulo modificado y mostramos los cambios en pantalla utilizando la función SDL_UpdateRects (en lugar de SDL_Flip), por último restauramos sólo aquellas zonas de pantalla que alteramos leyendo el vector.

» Descargar ejemplo 4 (código fuente)

Paso 5: procesos manejados por el teclado

En este quinto paso buscamos añadir a los protagonistas del juego, queremos que los personajes se puedan manejar utilizando el teclado.

Para ello, recordando que el videojuego no se detiene en ningún momento, necesitamos realizar algunos cambios:

» El ciclo gráfico imprime ahora a los 3 actores, la pelota y los dos protagonistas.
» En cada actualización lógica debemos capturar el estado de las teclas y cambiar la posición de los personajes.
Añadimos algunos archivos al proyecto, por ejemplo personaje.c, personaje.h e "ima/personaje1.bmp".

La estructura principal de cada personaje contiene su posición en pantalla, el gráfico a imprimir, la configuración de teclas y los límites del movimiento.

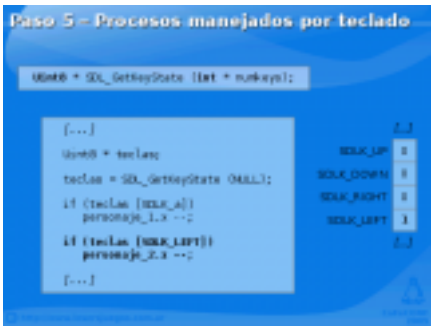
```
typedef struct Personaje
{
 int x;
 int y;
```

```
 SDL_Surface * imagen;
 int limite_izquierdo;
 int limite_derecho;
 SDLKey tecla_izquierda;
 SDLKey tecla_derecha;
} Personaje;
```

Paso 5: procesos manejados por el teclado

Para modificar la posición de los personajes utilizando el teclado necesitamos conocer el estado de cada tecla. Para ello SDL nos provee una función llamada SDL_GetKeyState.

La función nos informa acerca del estado de cada tecla, SDL_GetKeyState retornará la dirección de memoria de un vector en donde se almacena el mapa de teclado completo.



En la figura se observa una pequeña sección del vector, cada celda nos informa acerca del estado de una tecla en particular. Para acceder al estado de cada tecla podemos utilizar un índice como SDLK_LEFT, por lo tanto la celda de índice SDLK_LEFT almacenará un 0 (cero) si la tecla no ha sido pulsada y un 1 (uno) si está siendo pulsada.

En nuestro ejemplo de código ocurre esto, en un instante particular el usuario mantiene pulsada la tecla de direccionales izquierda (SDLK_LEFT) y nuestro programa ejecuta la línea:

```
personaje_2.x --;
```

» Descargar ejemplo 5 (código fuente)

Paso 6 - Colisiones con los personajes

Para continuar, quisiéramos que los personajes del videojuego puedan interactuar con la pelota.

Para que eso ocurra, debemos verificar en todo momento cuando se produce una colisión.

Existen varias formas de determinarlo, la colisión se produce cuando los dos gráficos entran en contacto.

Podríamos determinar que existe una colisión calculando cuando 2 pixeles se solapan; también podríamos utilizar "mascaras de imagen", o bien, aproximarnos con un sencillo cálculo geométrico.

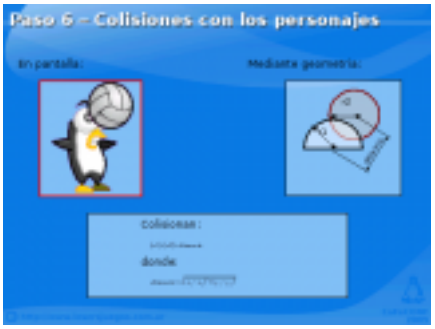
Esta última aproximación nos resultará mas sencilla, la forma de la pelota se asemeja a una circunferencia, y la forma del área de colisión del pingüino también.

Por lo tanto sabremos si se produce una colisión comparando la distancia entre circunferencias y la suma de los radios.

En la imagen no tenemos un indicio de colisión, dado que la suma de los radios es inferior a la distancia entre circunferencias.

Paso 6 - Colisiones con los personajes

En cambio en la siguiente imagen sí tenemos una colisión, la suma de los radios es mayor a la distancia de las circunferencias.



Para verificar la distancia de circunferencias hemos utilizado el teorema de Pitágoras en los archivos "pelota.c" y "util.c"

» Descargar ejemplo 6 (código fuente)

Paso 7 - Salto y animaciones

Hasta aquí, cuando necesitábamos imprimir una imagen almacenada en memoria realizábamos una impresión completa. Nuestra impresión se resumía a llamar a SDL_BlitSurface con un valor nulo:

```
SDL_BlitSurface (pelota, NULL, pantalla, & destino);
```

Es decir, con NULL le indicamos a SDL_BlitSurface que buscamos imprimir la imagen completa.

Ahora bien, nos gustaría que la pelota presente varios cuadros de animación, que los personajes parezcan caminar y moverse. Hasta ahora todos los gráficos se veían estáticos.

Existen muchas formas de gestionar animaciones, elegimos implementar una muy simple:

Todos los cuadros de animación se almacenan en una superficie, cada cuadro de animación tiene el mismo tamaño. Por lo tanto resulta muy sencillo capturar cada cuadro por separado e imprimirlo sobre la pantalla.

La función "imprimir_grafico", implementada en "util.c" se encarga de imprimir un gráfico individual de nuestra grilla sobre otra superficie. En la imagen se muestra como llamar a la función "imprimir_grafico" para realizar una animación de 18 cuadros.

Paso 7 - Salto y animaciones

Para "animar" a los protagonistas del juego hemos utilizado la misma técnica, todos los cuadros de la animación se almacenan en una grilla que contiene celdas del mismo tamaño.

La diferencia mas importante, entre la animación de la pelota y los protagonistas, está en el comportamiento de los personajes.

En este caso, los protagonistas tienen varias animaciones por realizar, por el momento quisiéramos que puedan caminar, saltar y permanecer en su lugar.

Para gestionar comportamientos como este, existe un modelo lógico que se suele aplicar con mucha frecuencia en los videojuegos, el autómata.

En el sitio LosersJuegos existe un artículo completo dedicado a los autómatas. Por ese motivo, aquí solo

daremos un breve comentario acerca de su utilidad.

Mediante un autómata podremos estudiar e implementar el comportamiento de cada personaje dividiendo su actividad en diferentes estados. "saltando" será un estado, "camina" será otro. De forma que cada estado tenga asociado un comportamiento y una animación particular.

La imagen muestra como podemos relacionar los eventos del teclado con el comportamiento del protagonista, cuando el usuario pulsa hacia arriba el personaje cambia su estado a "saltando", cuando el usuario pulsa hacia la izquierda el personaje cambia su estado a "camina".

» Descargar ejemplo 7 (código fuente)

Paso 8 - Algunas mejoras

En este paso hemos agregado algunas mejoras adicionales, agregamos un contador de tantos, la posibilidad de marcar puntos, una red y algunos efectos visuales.

Para comenzar, cuando la pelota colisionaba con el protagonista teníamos que cambiar la dirección y sentido de rebote, así lográbamos 'aparentar' una colisión verdadera. Ahora bien, podremos mejorar ese efecto si además mostramos la reacción del pingüino ante la interacción.

Para que la animación resulte alusiva a la situación de choque, cuando se produce una colisión obtenemos el ángulo que forman el radio de la pelota y aquel radio que "imaginamos" como propio del protagonista.

Si el ángulo está entre 135 y 180 grados, mostramos al personaje recibiendo un golpe desde la espalda. En cambio para un ángulo entre 45 y 135 grados el golpe es superior, y por último, un ángulo entre 0 y 45 grados muestra un gráfico alusivo a un golpe frontal.

Paso 8 - Algunas mejoras

Si marcamos un tanto, hablando del Volley, la pelota debe regresar a su posición inicial, para que podamos reanudar el juego. En lugar de cambiar inmediatamente la posición de la pelota hemos utilizado un efecto de transparencias.

Para mostrar a los usuarios que el juego a terminado, es decir, cuando la pelota ha tocado el suelo, aplicamos una transparencia uniforme a la superficie original.

SDL nos permite aplicar transparencias uniformes a cualquier superficie utilizando la función SDL_SetAlpha.

En nuestro caso, cuando la pelota colisiona con el suelo, llamamos varias veces a la función SDL_SetAlpha incrementando en cada llamada el valor del último parámetro. De forma que el gráfico aparente desvanecerse de manera gradual.

Si bien el siguiente ejemplo no corresponde al juego de volley, será de mucha utilidad para ilustrar la utilización de SDL_SetAlpha:

```
for (i = 0; i < 255; i ++)
{
 SDL_SetAlpha (pelota, SDL_RLEACCEL | SDL_SRCALPHA, i);

 [..] // imprimir en pantalla
}
```

» Descargar ejemplo 8 (código fuente)

Preguntas ...

Si tiene alguna sugerencia acerca de la exposición o las notas de este sitio por favor comuníquese con nosotros.

Muchas gracias.

» Descargar las diapositivas completas
» Descargar el apunte entregado en la charla
» Descargar todos los ejemplos

Este documento ha sido generado automáticamanete a partir del archivo 'un_juego_paso_a_paso.xml' el Mon Jan 19 21:06:33 2009
La versión mas reciente de este documento se almacena en www.losersjuegos.com.ar. Visitenos para obtener mas recursos y actualizaciones.