Colisión de plataformas

Créditos

» **Autor:** Hugo Ruscitti
» **Fecha:** Domingo 17 de Abril del 2005

Introducción

Los videojuegos de plataformas fueron muy populares en los primeros años de las consolas portátiles; Mario Bros, Donkey Kong y Sonic son muy buenos ejemplos de esta clase de juegos.

En este artículo veremos como implementar los escenarios de un juego de plataformas mediante pequeños bloques (tiles) y como programar las colisiones con estos bloques.

Tiles y matriz

figura 1

Si necesitamos crear un mapa como el de la figura 1, podemos utilizar una técnica que consiste en representar todo el mapa mediante una matriz con números y un conjunto de gráficos muy pequeños llamados tiles.

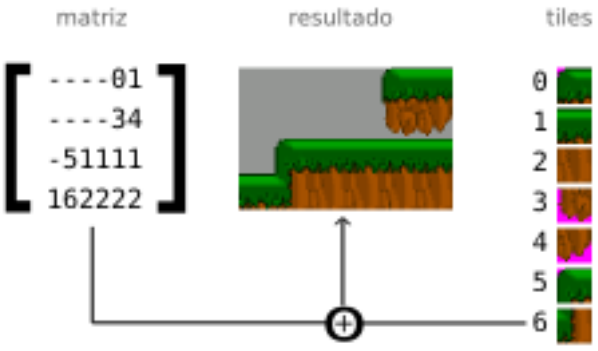

figura 2

La figura 2 muestra cómo combinar la matriz con los tiles para obtener el mapa completo. Cada gráfico tiene asignado un número y está cargado en memoria una sola vez. Los elementos (o celdas) de la matriz indican que gráficos se deben imprimir; la posición de estos elementos nos permiten determinar la posición final de cada tile en pantalla.

Almacenar la Matriz en memoria

El tamaño de la matriz depende del área total del mapa y del tamaño de los tiles. En nuestro caso el área del mapa es igual al de la pantalla, 320x240 pixels sin desplazamiento (scroll). Mientras que el tamaño de cada tile es de 16x16 pixels.

La relación es muy sencilla y debemos recordarla mas adelante:

```
int columnas_de_la_matriz;
int filas_de_la_matriz;

columnas_de_la_matriz = ancho_del_mapa / ancho_del_tile
filas_de_la_matriz = alto_del_mapa / alto_del_tile
```

Para nuestro ejemplo:

```
columnas = 640 / 16
filas = 480 / 32
```

La matriz debe tener 20 columnas y 15 filas.

Cargar el mapa

Como un juego de plataformas suele tener muchos niveles es adecuado almacenar estos niveles en un archivo (o varios) separados del código principal, ya que así podremos modificar los niveles con mayor facilidad.

Incluso podemos desarrollar un editor de niveles para manejar estos archivos.

Así, cargar el mapa consiste en leer el archivo de niveles y almacenar su contenido en las celdas de la matriz.

En este ejemplo el mapa de nivel está en un simple archivo de textos para facilitar su edición.

Imprimir el mapa

En la figura 3 podemos ver el proceso de impresión. Simplemente se debe leer el contenido de la matriz imprimiendo el gráfico indicado en la pantalla.

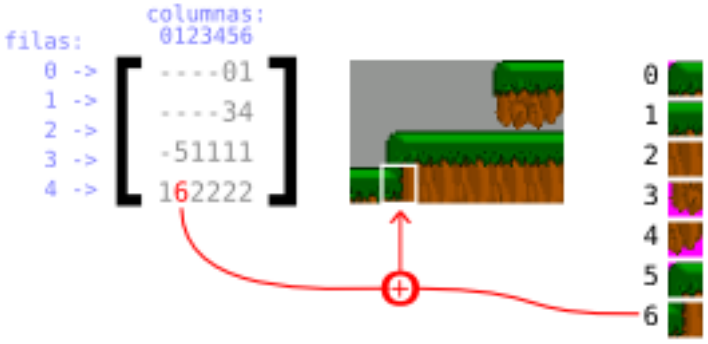

figura 3

La posición final de cada bloque sobre la pantalla se obtiene desde la matriz a medida que leemos la cada una de las celdas:

```
x_destino = columna_actual * ancho_del_tile
y_destino = fila_actual * alto_del_tile
```

Colisiones

En el ejemplo, nuestro personaje puede 'pararse' sobre cada una de las plataformas del juego y además 'colisionar' con las paredes del nivel.

Una forma muy simple de aplicar estas colisiones con las plataformas es determinar, siempre que sea necesario, cual es la distancia entre el personaje y la próxima plataforma o pared.

figura 4

La figura 4 muestra estas distancias. Si nuestro personaje está caminando hacia una pared es necesario detenerlo antes de que pueda cruzarla. Lo mismo ocurre cuando un personaje cae en su salto, debemos detenerlo en el suelo e indicarle que su caída terminó.

Todo avance del personaje puede superar el pixel, es decir, en un instante dado nuestro personaje podría necesitar avanzar unos 20 pixels a la vez. Si en ese trayecto encontramos una plataforma debemos detener el movimiento y avanzar menos de lo que queríamos recorrer.

Para implementar ese control podemos realizar una función que determine, dada una coordenada y rango, la distancia hasta el próxima plataforma:

```
int nivel :: get_dist_suelo (int x, int y, int rango)
{
 int cont;

 for (cont = 0; cont < rango; cont ++)
 {
  if (es_suelo(x, y + cont))
   return cont;
 }

 return rango; // no encuentra un suelo
}
```

El objetivo de la función get_dist_suelo es buscar, punto a punto, la existencia de un suelo sin exceder el rango (tercer parámetro). Si encuentra un suelo nos retorna la distancia al mismo, en caso contrario nos retorna el valor de rango.

La función es_suelo simplemente verifica si un bloque es sólido o no. Sus parámetros corresponden a una coordenada de la pantalla (320x240):

```
bool nivel :: es_suelo (int x, int y)
{
 // el suelo sólo es colisionable en su parte
 // superior, mediante el resto de la división (%)
 // verificamos este caso


 if (y % 16 == 0 && es_solido(x, y))
  return true;
 else
  return false;
}

bool nivel :: es_solido (int x, int y)
{
 return tiles[y / 16][x / 16].solido;
}
```

Para utilizar estas funciones simplemente debemos tener una referencia al nivel:

```
[...]

dist += nivel->get_dist_suelo(x, y, 20)

if (dist == 20)
 printf("No existe un suelo a menos de 20 pixels\n");
else
 printf("Hay una plataforma o suelo a %d pixels\n", dist);
```

```
y += dist;

[...]
```

Ejemplo

A diferencia de otros artículos, aquí he utilizado el lenguaje de programación C++ y otras técnicas que repasaremos más tarde, como los autómatas y el manejo de la gravedad.

Captura del programa funcionando

El ejemplo consiste en un personaje que puede ser manejado con los direccionales del teclado. Se puede saltar sobre las plataformas y chocar contra las paredes.

Todo el código fuente del programa puede ser descargado o consultado "en linea" directamente desde este sitio web, seleccionado a continuación "ver documentación del código fuente" y luego en "lista de archivos".

- **»** Descargar código fuente completo.
- **»** Versión compilada para Windows.
- **»** Ver documentación del código fuente.

nota: Descargando el código fuente completo puede generar la documentación usted mismo: debe tener instalando el programa Doxygen y luego ejecutar make documentacion. La documentación generada se almacena en doc/html.

Créditos

El programa de ejemplo se distribuye bajo los términos de la Licencia Pública General de GNU.
