Aprende a programar videojuegos en C++ con Gosu

José Tomás Tocino García
josetomas.tocino@uca.es

24 de Marzo de 2011

Índice de contenidos

Índice

Introducción

¿Qué es Gosu?

Gosu es una biblioteca de desarrollo de videojuegos 2D para C++ y Ruby. Sus principales ventajas son:

- Completamente orientada a objetos, facilitando el desarrollo.
- Multiplataforma: Windows, GNU/Linux, Mac OS, iPhone...
- Aceleración gráfica, alcanzando un rendimiento alto.
- Muy amplia, incluye incluso soporte para juego en red.

Algunas desventajas...

- Desarrollo algo irregular en las diferentes plataformas.
- Al no ser muy famosa, no está en los repositorios pero es fácil de instalar.

¿Cómo se desarrolla un videojuego?

Taller introductorio

Lectura recomendada:
`http://code.google.com/p/pong-sdl-advuca/`

En esencia, un videojuego es una aplicación en la que se repiten una serie de pasos constantemente:

- Primera etapa: ver qué hace el usuario, ¿está pulsando teclas?
- Segunda etapa: hacer cálculos, procesar la lógica.
- Tercera y última etapa: mostrar los gráficos en pantalla.
- Again and again, and again and again...

Materiales del taller

- **Carpeta windows**: Ficheros para hacer el taller en Windows con Visual Studio.
- **Carpeta linux**: Ficheros para hacer el taller en Linux por línea de comandos.
- **Carpeta soluciones**: Soluciones a los ejercicios que vayamos haciendo.

Materiales del taller: Windows

Dentro de la carpeta **windows** encontraréis:

- `TallerGosu.sln`
  Fichero de la solución para Visual Studio 2008 Express. Al abrirlo, dentro están integrados los proyectos para las dos partes del taller.
- **Carpeta** `Parte 1`
  Ficheros para la primera parte del taller. Trabajaremos con el fichero `main.cpp`
- **Carpeta** `Parte 2`
  Ficheros para la segunda parte del taller. Trabajaremos con el fichero `src/ventana.cpp`

Compilar en Windows

Para compilar en Visual Studio 2008 Express, seguimos estos dos pasos:

1. Tenemos que seleccionar el proyecto que queremos compilar de la lista de la izquierda con el botón derecho y seleccionar **Establecer como proyecto de inicio**.
2. A partir de aquí, podemos compilar el proyecto pulsando el botón F5 (nos preguntará si queremos generar el proyecto, respondemos que **sí**).

Materiales del taller: Linux

Dentro de la carpeta **linux** encontraréis:

- **Carpeta "gosu"**
  Carpeta con la distribución de ficheros de Gosu necesarios para compilar los proyectos.
- **Carpeta "Parte 1"**
  Ficheros para la primera parte del taller. Trabajaremos con el fichero `main.cpp`
- **Carpeta "Parte 2"**
  Ficheros para la segunda parte del taller. Trabajaremos con el fichero `src/ventana.cpp`

Compilar en Linux

Para compilar en linux (por línea de comandos), el proceso es muy sencillo.

- Estando en cualquiera de las dos carpetas(Parte 1 ó Parte 2), ejecutamos `make libgosu` para compilar Gosu.
  OJO: Esto solo hay que hacerlo una vez en todo el taller.
- Luego, cada vez que hagamos cambios en los ficheros .cpp, simplemente ejecutamos `make` para compilar el código. Podremos lanzarlo utilizando el comando `./programa`

Índice

Primeros pasos con Gosu

Cabeceras

Únicamente tendremos que usar

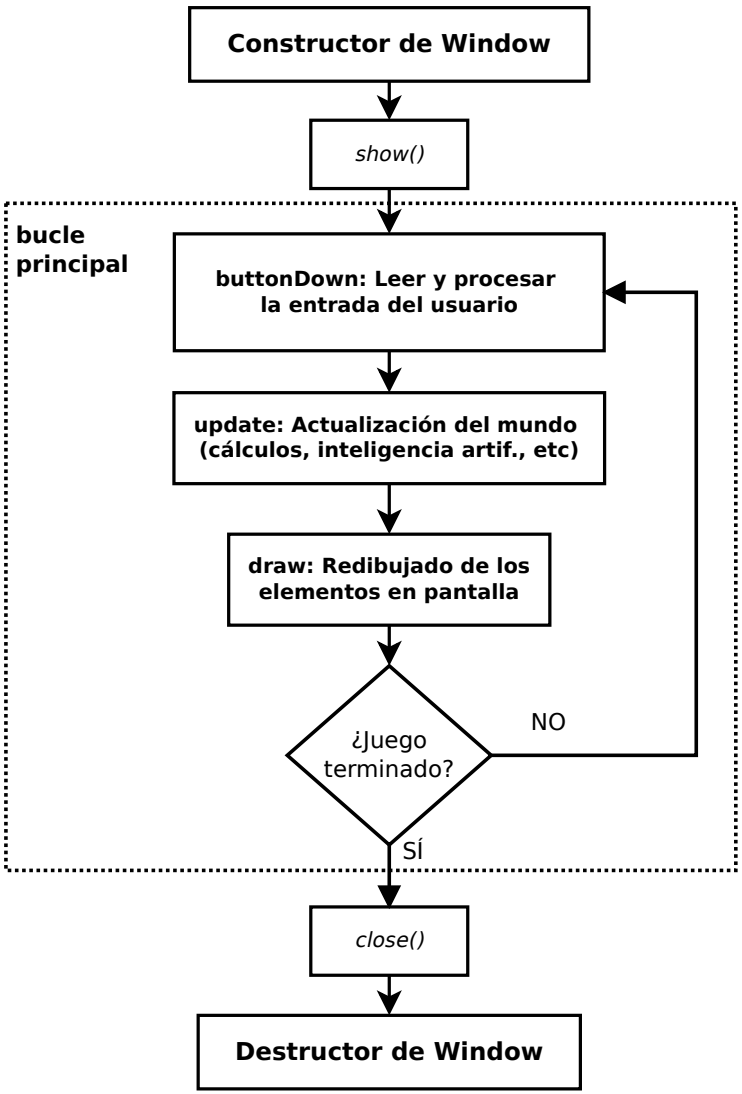
```
#include <Gosu/Gosu.hpp>
#include <Gosu/AutoLink.hpp> // Solo windows
```

Con esto ya estaremos incluyendo todos los ficheros necesarios.

La clase principal

Gosu está dividido en **clases**. La clase `Gosu::Window` representa la ventana de juego, la clase principal de nuestro juego heredará de ella.

La clase principal




```
class Ventana
   : public Gosu::Window{
public:
  Ventana()
    : Gosu::Window(800, 600, false){
  }

  ~Ventana(){ /* destructor */ }

  void update(){
    // Lógica del juego
  }
  void draw(){
    // Dibujado de gráficos
  }
  void buttonDown(Gosu::Button B){
    // Pulsaciones de teclas
  }
};
```

Constructor de Gosu::Window

```
class Ventana : public Gosu::Window{

public:
  Ventana() : Gosu::Window(800, 600, false){
```

- `Ventana` hereda de `Gosu::Window`.
- En el constructor del padre, indicamos ancho, alto y si queremos pantalla completa.

Métodos de Gosu::Window

```
void update(){
  // Aquí debe ir la lógica del juego
}

void draw(){
  // Aquí se deben pintar los gráficos
}

void buttonDown(Gosu::Button boton){
  // Este método se lanza cuando se pulsa un botón
}
```

Abriendo y cerrando la ventana

Para abrir una ventana de juego usaremos el método `show()`:

```
int main(){
  Ventana V;
  V.show();
}
```

Para cerrar nuestra ventana, podemos usar el método `close()`, por ejemplo al pulsar cualquier botón:

```
void buttonDown(Gosu::Button B){
  close();
}
```

Primera tarea

- Abrir el fichero `main.cpp`
  - Linux: en la carpeta `Parte 1`.
    Para compilar, ejecutar "`make libgosu`" y luego "`make`".
    Para ejecutar, `./programa`
  - Windows: en el proyecto `Parte 1`
- Crear una clase `Ventana` derivada de `Gosu::Window` tal y como hemos explicado.
- Hacer que se muestre la ventana y que se cierre al pulsar una tecla.
- La solución la tenéis en el fichero `soluciones/Parte 1/main_paso1.cpp`

Gráficos en Gosu

Los gráficos son una de las piezas más importantes de un juego.

Cada objeto `Gosu::Window` tiene un método `graphics()` que devuelve el destino donde todos los elementos gráficos (imágenes y fuentes) se van a pintar.

Imágenes en Gosu

En Gosu las imágenes se representan con la clase `Gosu::Image`.

```
Gosu::Image * miImagen =
    new Gosu::Image(graphics(), L"rutaDeImagen.png");
```

Y se dibujan así (dentro del método `Ventana::draw`):

```
miImagen -> draw(posX, posY, posZ);
```

Es posible escalar las imágenes y tintarlas utilizando un color:

```
miImage -> draw(posX, posY, posZ, escalaX, escalaY
           Gosu::Color(255, 255, 0, 0)); // Tinte rojo
```

Apuntes sobre las rutas

Si os habéis fijado, las rutas de los ficheros se indican utilizando `wstring`, que se indican poniendo una L delante:

```
wstring ruta = L"rutaFichero.png";
```

Para que el juego pueda encontrar las imágenes, hay que anteponer la salida de la función `Gosu::sharedResourcePrefix()`, que nos devuelve el directorio en el que está el ejecutable:

```
wstring directorio = Gosu::sharedResourcePrefix()
wstring rutaCompleta = directorio + L"rutaFichero.png";
```

Segunda tarea

- Añadir a la clase `Ventana` un atributo `Gosu::Image *imagen`.
- En el constructor de la `Ventana`, cargar la imagen ``` ``imagen.png'' ``` en el atributo que hemos definido.
- En el método `draw` de `Ventana`, pintar la imagen en las coordenadas que queráis.
- La solución la tenéis en el fichero `soluciones/Parte 1/main_paso2.cpp`

Textos en Gosu

Los textos

también son importantes

para transmitir mensajes.

Utilizaremos la clase `Gosu::Font` para cargar una fuente y luego escribir un texto con ella. Funciona igual que `Gosu::Image`.

Cargando y mostrando fuentes

Las fuentes se cargan igual que las imágenes, con la diferencia de que hay que añadir el tamaño y si queremos negritas o no:

```
Gosu::Font * miFuente = new Gosu::Font(
                graphics(),
                directorio + L"nombreFuente.ttf",
                30, // El tamaño de la fuente en píxeles
                0); // 0 indica que no queremos negritas
```

Para escribir, el método también se llama `draw`:

```
miFuente -> draw(posX, posY, posZ,
                      L"Este es el texto que aparecerá");
```

Tercera tarea

- Añadir a la clase `Ventana` un atributo `Gosu::Font * fuente`.
- En el constructor de la `Ventana`, cargar la fuente `arial.ttf` en el atributo que hemos definido, con un tamaño de 60 píxeles.
- En el método `draw` de `Ventana`, pintar un mensaje con la fuente que acabamos de cargar usando su método `draw`.
- La solución la tenéis en el fichero `soluciones/Parte 1/main_paso3.cpp`

Interactuando con el usuario

El método `buttonDown` se ejecutará cuando el usuario pulse un botón, que recibimos como parámetro. Podemos reaccionar ante eso para, por ejemplo, cambiar el valor de una variable.

```
void buttonDown(Gosu::Button B){
  if (B == Gosu::kbEscape){
    close();
  }else{
    otraTecla = true;
  }
}
//...
private:
  bool otraTecla;
```

Leyendo la posición del ratón

Además, la clase `Gosu::Window` nos ofrece un método `input()` que nos devuelve un objeto que nos da información sobre la entrada del usuario, como por ejemplo la posición del ratón:

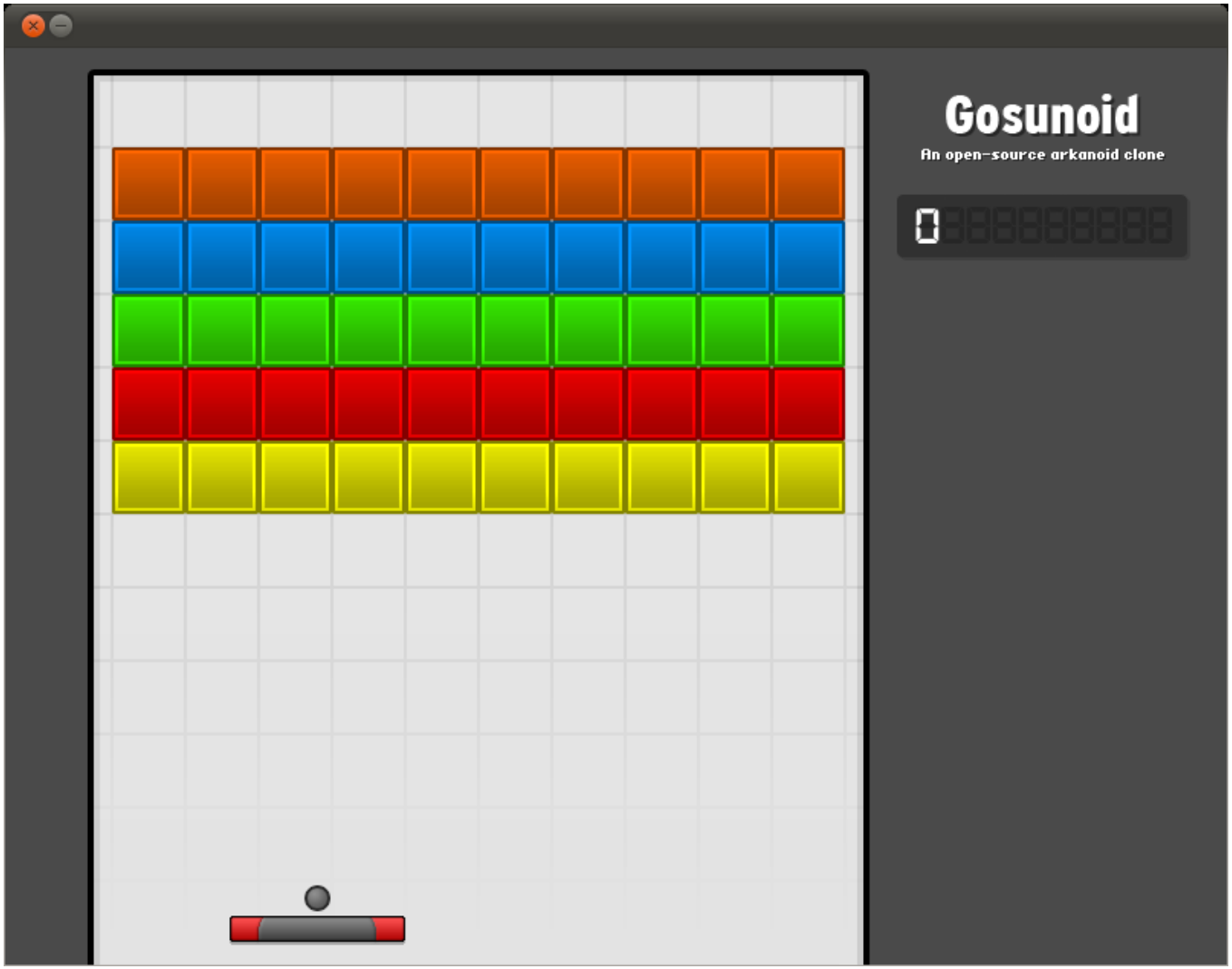
```
input().mouseX();
input().mouseY();
```

Cuarta tarea

- Añadir a la clase `Ventana` un atributo entero: `int contador` e inicializarlo a 0 en el constructor.
- Comprobar en el método `buttonDown` si se ha pulsado la tecla escape (cuyo código es `Gosu::kbEscape`). En ese caso, cerrar la aplicación (usando `close()`).
- Si se ha pulsado otra tecla, aumentar el contador. Cuando el contador sea mayor que 5, cambiar el mensaje que escribimos en la tarea anterior.
- La solución la tenéis en el fichero `soluciones/Parte 1/main_paso4.cpp`

Índice

Gosunoid



Es un clon del clásico **Arkanoid**, que a su vez fue un clon del juego **Breakout** de Atari.

La idea es acabar con todos los bloques.

Recursos que utilizaremos

Figura: media/pala.png - Pala que controla el jugador

Figura: media/bola.png - Bola

Recursos que utilizaremos

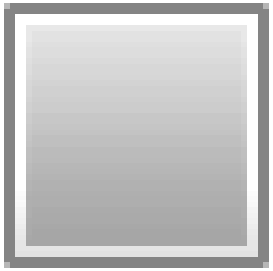
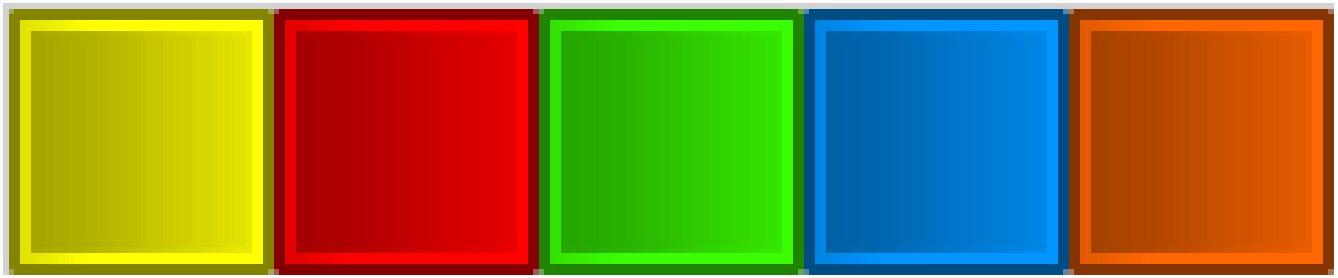


Figura: `media/bloque.png`

Utilizaremos una misma imagen para pintar todos los bloques, ya que el método `draw` de la clase `Gosu::Window` nos permite indicar un color con el que tintar el bloque.



Recursos que utilizaremos

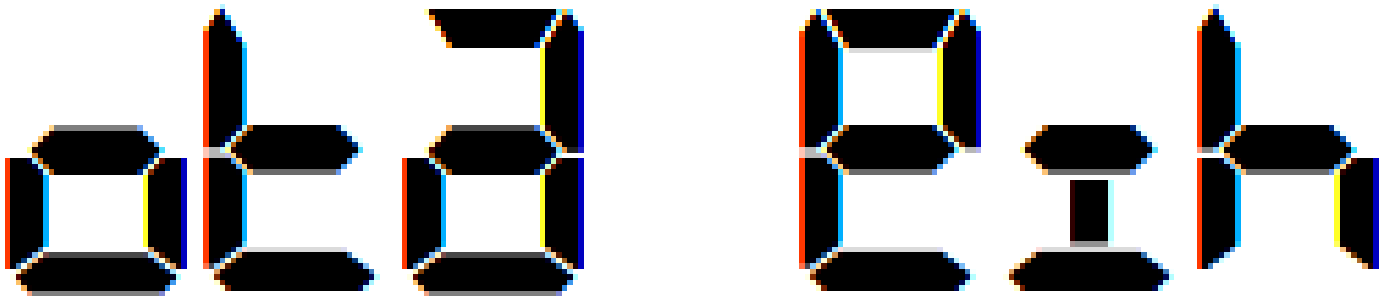


Figura: `media/fuentelcd.ttf` - Fuente tipo LCD

Organización del proyecto

En Linux

- **Carpeta include**
  Ficheros de cabecera (.h)
- **Carpeta src**
  Ficheros de código fuente: `main.cpp` y `ventana.cpp`
- **Carpeta obj**
  Ficheros temporales.
- **Carpeta media**
  Recursos (imágenes y tipos de letra .ttf).
- **Makefile**
  Script para compilar, solo hace falta escribir `make`.

Organización del proyecto

En Windows

- **Carpeta include**
  Ficheros de cabecera (.h)
- **Carpeta src**
  Ficheros de código fuente: `main.cpp` y `ventana.cpp`
- **Carpeta ArchivosTemporales**
  Ficheros de código objeto.
- **Carpeta Ejecutable**
  Aquí se genera el .exe
- **Parte 2.vcproj**
  Fichero del proyecto

Trabajaremos con el fichero `src/ventana.cpp`.

Como tenemos poco tiempo, trabajaremos con el proyecto ya comenzado en lugar de empezar de cero.

Iremos rellenando las partes que faltan hasta que funcione por completo. Las partes que faltan por rellenar están marcadas con la siguiente línea:

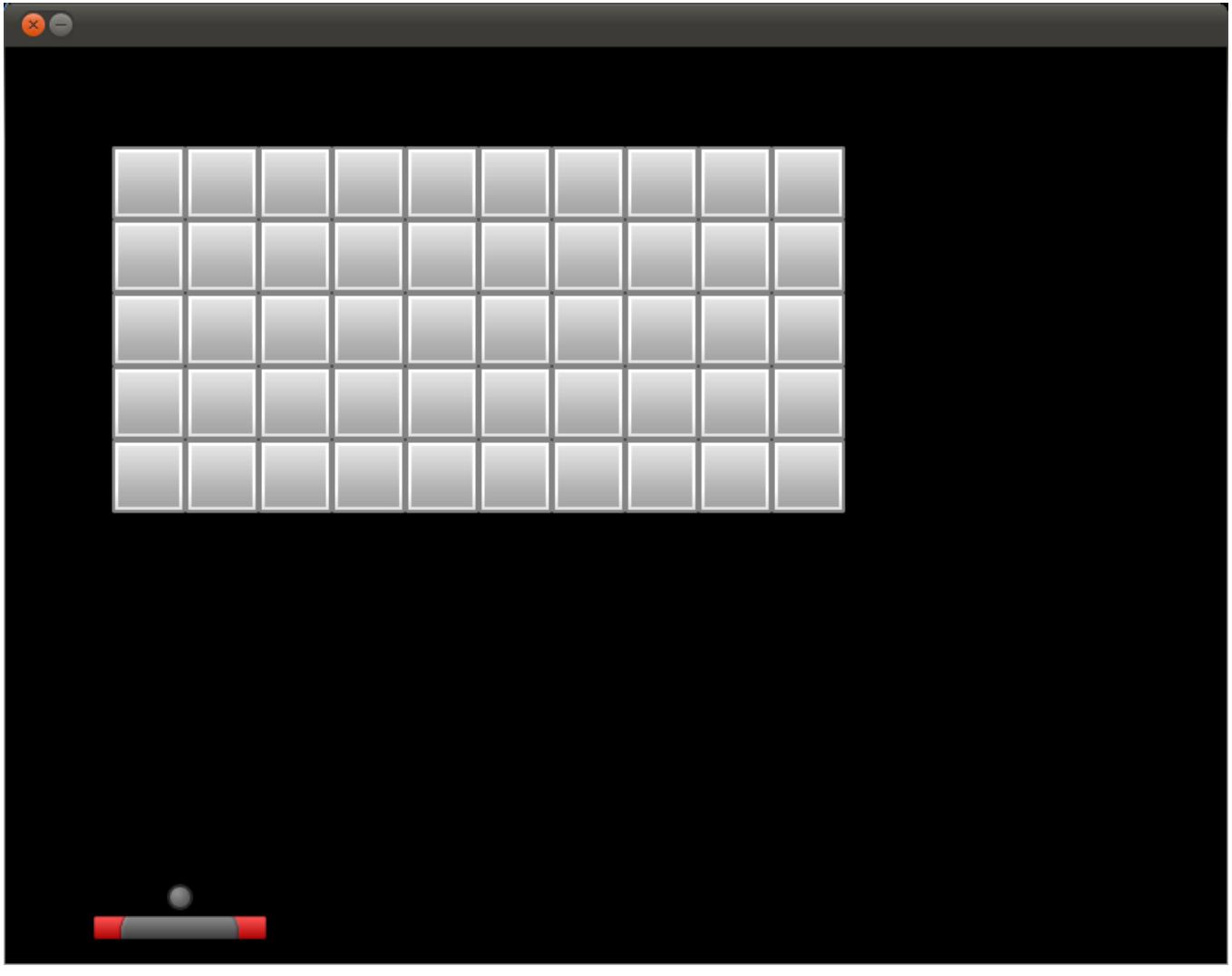
```
// @@RELLENAME@@
```

El código completo lo encontraréis en `soluciones/Parte 2/ventana.cpp`

Overview del proyecto

- La clase principal es `Ventana`, que deriva de `Gosu::Window`.
- Tiene los métodos típicos que hemos comentado antes:
  - `update()`
  - `draw()`
  - `buttonDown()`
- Tiene los siguientes elementos gráficos:
  - Imagen para la pala, la bola, y el fondo.
  - Imagen genérica para los bloques.
  - Fuente para el marcador de puntos.
- El enumerado `tipoEstado` indica los dos posibles estados del juego:
  - `ESTADO_INICIAL`, cuando la bola está sobre la pala.
  - `ESTADO_MOVIMIENTO`, cuando la bola está en movimiento.
  - El estado se guarda en la variable `estadoActual`.

Estado inicial



Parece que faltan unas cuantas cosas para que esto salga adelante...

Problema 1: ¿qué pasa con el fondo?

Parece que el fondo no se está pintando, habrá que ver qué pasa en el método `draw` de nuestra clase `Ventana`...

Escribe el código necesario para que se pinte el fondo en las coordenadas (0,0,0)

Problema 2: bloques de... ¿colores?

Los bloques parece que están de luto. Se dibujan en pantalla, sí, pero parece que no con el color adecuado.

Fíjate en el método `draw` de la clase `Ventana`, hay una variable `bloqueColor` que guarda el color asociado al bloque, pero no parece que se utilice luego en ningún lado.

Modifica el código que pinta los bloques para que acepte el atributo de color `bloqueColor`

Problema 3: ¡quiero ver mis puntos!

No hay nada más triste que un marcador de puntos sin puntos. Parece que se nos ha olvidado cargar la fuente para mostrar el marcador.

En el constructor de la clase `Ventana`, añade el código necesario para cargar en el atributo `fuente` una nueva `Gosu::Font` que cargue el archivo `"media/fuentelcd.ttf"` a un tamaño de 33 píxeles y sin negritas.

Problema 4: ¡la pala no se mueve!

Aunque aparentemente todo pinta bien, la pala no sigue al ratón al moverse, siempre se queda fija en la misma posición. Esto no pinta nada bien!

Tal vez puedas conseguir leer la posición horizontal del ratón y asignársela a la variable `mouseX` dentro del método `update` de `Ventana`.

¡Esto ha sido todo, hamijos!

Gracias por venir :)