

Integración Wiimote en una aplicación OpenGL

Curso 2k17/2k18

Apellidos, Nombres	Richart Ribes, Ignacio Revert Enguix, Jorge
Titulación	Grado de Ingeniería informática
Fecha	25 de abril de 2018



Índice

1 Resumen de las ideas clave.....	1
2 Introducción.....	1
3 Objetivos.....	1
4 Conocimientos previos.....	1
5 Desarrollo.....	2
5.1 Instalación de componentes.....	2
5.2 Escenario 3D.....	3
5.3 Animación de disparo a diana y apuntado.....	4
5.4 Integración del mando en el PC.....	12
6 Problemas encontrados y trabajos futuros.....	15
7. Conclusión.....	16
8. Bibliografía y referencias.....	16

1 Resumen de las ideas clave

En este documento vamos a presentar el desarrollo de un programa en el que se consiga integrar el uso de la librería cwiid con una aplicación desarrollada sobre OpenGL para que nos permita interactuar mediante el uso de un mando Wiimote con dicha aplicación.

2 Introducción

El trabajo a realizar consiste en diseñar un escenario 3D en el que OpenGL será el encargado de hacer aparecer dianas en diferentes puntos. El usuario lanzará proyectiles hacia las dianas para ir acumulando puntos, consiguiendo así simular un pequeño "shooter".

Para su realización, se enfocó el desarrollo en dos ramas, una destinada a obtener una versión completa del juego sin el uso del Wiimote y otra encargada de la integración del mando al juego resultante.

En este documento se van a mostrar los pasos realizados para alcanzar los objetivos, desde las librerías utilizadas, cómo instalarlas, el código implementado, su compilación y los problemas encontrados durante el desarrollo, así como los elementos que han sido necesarios.

3 Objetivos

En este apartado expondremos los objetivos del proyecto:

- Pequeño repaso de la instalación de los componentes.
- Implementación de un entorno 3D en el que han de aparecer dianas en diferentes posiciones.
- Realización de una animación de disparo para el lanzamiento de las balas o proyectiles utilizando el ratón para apuntar.
- Integración y utilización del Wiimote en el ordenador sobre GNU/Linux

4 Conocimientos previos

En este apartado se va a comentar que no es necesario ningún conocimiento previo especial, simplemente tener nociones básicas del lenguaje de programación C y compilar por terminal.

5 Desarrollo

En este apartado mostraremos el trabajo realizado cronológicamente sin perder el enfoque que habíamos comentado anteriormente.

Observando globalmente el proyecto necesitaremos: OpenGL para el renderizado del escenario 3D, la aparición de dianas; GLUT para la interacción del usuario con la ventana de la aplicación (interacción con el ratón); libbluetooth3 para conectar por Bluetooth el Wiimote y la librería cwiid para controlar el Wiimote y desarrollar código para el mismo.

5.1 Instalación de componentes

Para obtener la librería de funciones OpenGL utilizaremos el gestor de paquetes Synaptic donde buscaremos la implementación de freeglut3 tal y como se observa en la Ilustración 1.

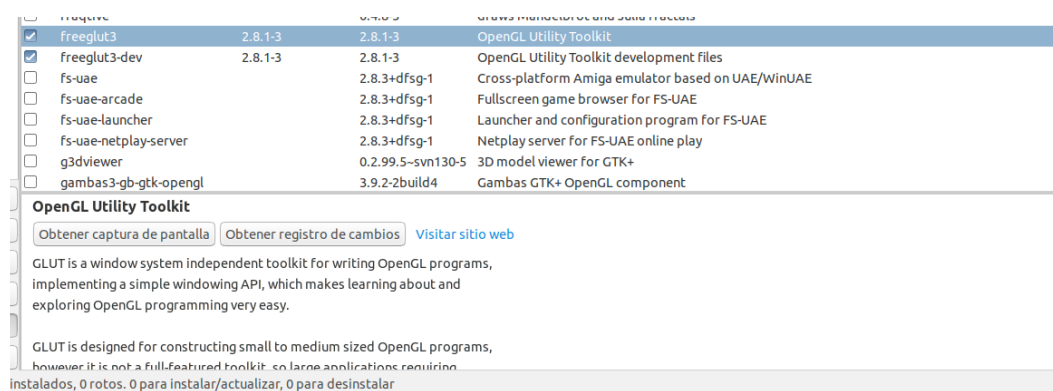


Ilustración 1: Implementación OpenGL a instalar

Con la instalación de freeglut3-dev ya disponemos de las funciones de alto y bajo nivel de OpenGL y GLUT.

Ahora pasamos a instalar la librería necesaria para gestionar las conexiones Bluetooth, por lo que la buscaremos libbluetooth-dev del mismo modo empleado anteriormente. Véase Ilustración 2.

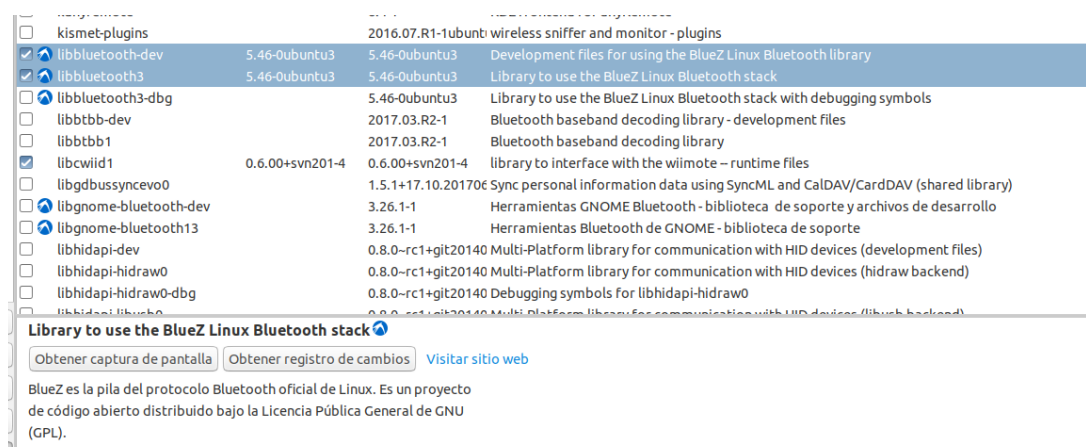


Ilustración 2: Librería Bluetooth

Por último instalaremos la librería necesaria para el control del Wiimote, libcwiid-dev. Véase Ilustración 3

<input type="checkbox"/>	cwiid-dbg	0.6.00+svn201-4	library to interface with the wiimote -- debug files
<input checked="" type="checkbox"/>	libcwiid-dev	0.6.00+svn201-4	library to interface with the wiimote -- development files
<input checked="" type="checkbox"/>	libcwiid1	0.6.00+svn201-4	library to interface with the wiimote -- runtime files
<input type="checkbox"/>	libghc-hcwiid-dev	0.0.5-6build1	Library to interface with the wiimote
<input type="checkbox"/>	libghc-hcwiid-doc	0.0.5-6build1	Library to interface with the wiimote; documentation
<input type="checkbox"/>	libghc-hcwiid-prof	0.0.5-6build1	Library to interface with the wiimote; profiling libraries
<input checked="" type="checkbox"/>	lswm	0.6.00+svn201-4	wiimote discover utility
<input checked="" type="checkbox"/>	python-cwiid	0.6.00+svn201-4	library to interface with the wiimote
<input type="checkbox"/>	transfermii	1:0.6.1-3	transfer your mii from and to your wiimotes
<input type="checkbox"/>	transfermii-gui	1:0.6.1-3	transfer your mii from and to your wiimotes -- GUI
<input checked="" type="checkbox"/>	wmgui	0.6.00+svn201-4	GUI interface to the wiimote
<input checked="" type="checkbox"/>	wminput	0.6.00+svn201-4	Userspace driver for the wiimote

Ilustración 3: Librería para desarrollo con Wiimote

5.2 Escenario 3D

En este punto se empieza a elaborar un escenario en el que aparecieran diferentes dianas en distintos puntos y jugando con la perspectiva lo dotaremos de un efecto de profundidad.

Para conseguir este escenario, partimos de la actividad realizada en la primera práctica de la asignatura, “bouncing ball” ([1]), en la cual aparece una bola que rebota en una ventana (véase Ilustración 4).

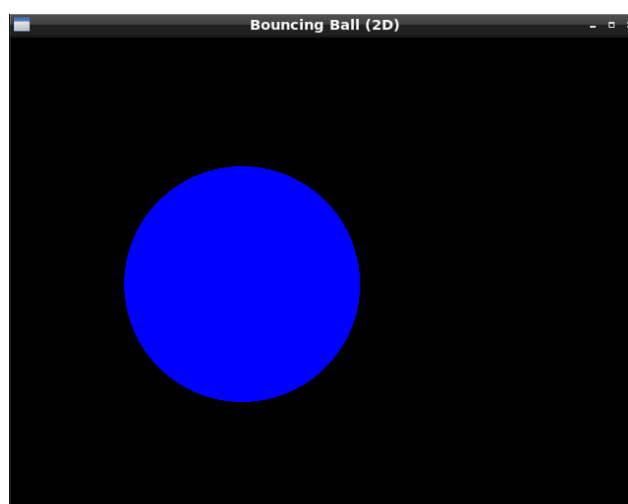


Ilustración 4: Diana inicial

Esta bola que aparece será la que se utilice como diana, la cuál modificaremos posteriormente.

Una vez tenemos la diana, se busca dotar de profundidad a la escena, ya que este ejemplo del que partimos se encuentra en 2D y con la pantalla acotada a unos valores muy bajos, por lo que se utilizó el ejemplo openal.c ([2]) con el que trabajamos en la práctica 3 de la asignatura, en el que encontrábamos objetos a una profundidad determinada.

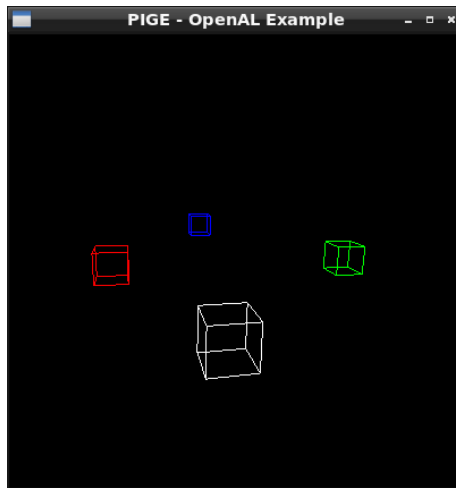


Ilustración 5: Ejemplo utilizado para dotar de profundidad a la escena

Basándonos en el ejemplo de la Ilustración 5, en cómo crea la escena y coloca sus elementos en ella, conseguimos adaptar nuestro escenario para que muestre la diana con una cierta profundidad.

5.3 Animación de disparo a diana y apuntado

Con la diana ya posicionada en el espacio tridimensional con unas coordenadas definidas para cada eje (estas son *ballX*, *ballY* y *ballZ* en nuestro código), necesitamos un elemento más para poder realizar la animación de un disparo a diana. Para ello, pintaremos una nueva bola en la escena, que será el proyectil a lanzar. Esto lo realizaremos del mismo modo que la diana, pero siendo de menor tamaño y mostrándolo a una menor profundidad, provocando la sensación de que se encuentra más cerca (las coordenadas del proyectil serán *shootX*, *shootY* y *shootZ*).

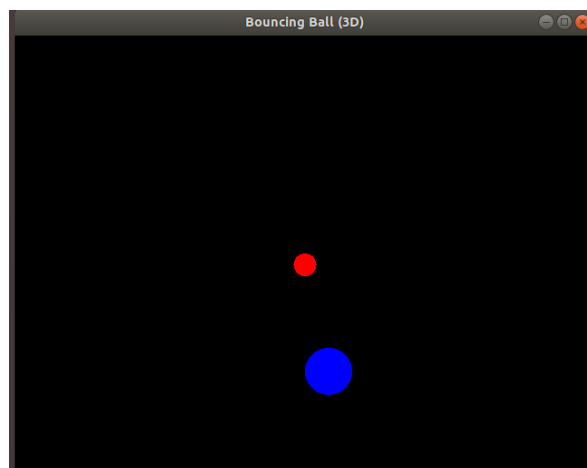


Ilustración 6: Diana y proyectil en la escena

Ahora ya tenemos todos los elementos necesarios en la escena (Ilustración 6), solo falta realizar el “disparo”. Para ello, utilizamos los eventos del ratón que gestiona OpenGL para que se active la función “disparo”, que explicaremos a continuación, cuando se pulse el botón izquierdo del mismo como observamos en la Ilustración 7.

```

////////acción del ratón////////////////////////////////////
void mouse(int button, int state, int x, int y)
{
    switch (button) {
        case GLUT_LEFT_BUTTON:
            glutIdleFunc(shoot);
            break;
    }
}

```

Ilustración 7: Ejecutar función disparo con el click derecho

Esta primera versión de la función de disparo, a la que llamaremos “shoot”, simplemente permite a la bola avanzar en profundidad (con las mismas coordenadas X e Y de las que partía) hasta llegar a la misma a la que se encuentra la diana, simulando un tiro lineal y siendo capaz de detectar si ha colisionado o no con la diana. En esta versión, en caso de colisionar con la diana, desplazaremos el proyectil en el eje X para averiguar que efectivamente, se detecta la colisión. También devolveremos el proyectil a su profundidad inicial antes del disparo, esto es “shootZ= 5.0 “. Un ejemplo de la función descrita anteriormente se puede encontrar en la Ilustración 8 (la profundidad aumenta negativamente en el eje de coordenadas Z cuanto más lejos se encuentra, de ahí que se aumente la profundidad del proyectil mediante la resta).

```

////////función de la animación del disparo////////////////////////////////
void shoot(void){
    if(shootZ > ballZ){
        shootZ -= 0.1;
    }
    else if (distancia() < ballRadius+shootRadius){
        shootX += 0.1;
        shootZ = 5.0;
    }
    else (distancia() >= ballRadius+shootRadius){
        shootX = 0.0;
        shootZ = 5.0;
    }
    glutPostRedisplay();
}
}

```

Ilustración 8: Primera versión de la función disparo

Para averiguar si se ha producido la colisión, se calcula la distancia entre los centros de la diana y el proyectil (véase Ilustración 9) y se comprueba si han coincidido (condición vista en la Ilustración 8, $distancia() < ballRadius + shootRadius$, donde $ballRadius$ y $shootRadius$ son los radios tanto de la diana como la del proyectil respectivamente).

```
//////////calcular distancia entre centros//////////
float distancia() {
    return sqrt( pow(ballZ - shootZ,2) + pow(ballY - shootY,2) +pow(ballX - shootX,2) );
}
```

Ilustración 9: Función para calcular la distancia entre los centros

Una vez la animación del disparo lineal se encuentra implementada, queremos que el proyectil no se dirija a una posición fija, sino que pueda ser lanzado al lugar que nosotros escojamos, para que podamos apuntar a la diana. Para lograr ese objetivo, utilizamos también el evento del ratón para detectar en que lugar de la ventana se ha pulsado y dirigir hacia allí el proyectil. Sin embargo, las coordenadas de la ventana que detecta el ratón no coinciden con las de nuestra escena 3D, por lo que hay que transformarlas para que coincidan. Para conseguirlo, se ha encontrado un programa ([3]) donde se transforman las coordenadas para adaptarlas a un escenario que utiliza OpenGL, por lo que adaptándolo al nuestro (código mostrado en la Ilustración 10), conseguimos que la posición donde se encuentra el ratón se traduzca a unos valores introducidos manualmente ($0.018 * anchura$ de la ventana para el eje X y $0.019 * altura$ de la ventana para el eje Y) que coinciden con la escena 3D.

```
//////acción del ratón//////
void mouse(int button, int state, int x, int y)
{
    xmin0 = -(0.018*windowWidth)/2; //valor negativo mínimo de la coordenada X en la escena 3D
    xmax0 = (0.018*windowWidth)/2; //valor positivo máximo de la coordenada X en la escena 3D
    ymax0 = (0.019*windowHeight)/2; //valor positivo máximo de la coordenada Y en la escena 3D
    ymin0 = -(0.019*windowHeight)/2; //valor negativo mínimo de la coordenada Y en la escena 3D

    ancho_total_seccion_espacio = xmax0 - xmin0;
    alto_total_seccion_espacio = ymax0 - ymin0;

    relacionX = ancho_total_seccion_espacio/windowWidth;
    relacionY = alto_total_seccion_espacio/windowHeight;

    //variable X traducida del valor obtenido por el ratón al valor correspondiente en la escena
    espX = x * relacionX + xmin0;
    espY = (windowHeight-y) * relacionY + ymin0;
    //variable Y traducida del valor obtenido por el ratón al valor correspondiente en la escena

    //raton[] es la variable donde se guardan las posiciones obtenidas por el ratón ya transformadas
    raton[0] = espX;
    raton[1] = espY;
}
```

Ilustración 10: Transformación de valores obtenidos por el ratón a nuestro escenario

Una vez se adapten, ya tenemos guardadas las coordenadas a las que apunta el ratón en nuestra escena 3D (variable *raton[]*), por lo que pasamos a ajustar la función disparo, que calculará la distancia a la que se encuentra el proyectil de las posición seleccionada para calcular su avance en cada coordenada. Este valor de avance se ha fijado en la distancia total de una coordenada dada (posición inicial del proyectil hasta posición destino) dividida entre 20 (valor seleccionado de forma intuitiva para lograr una animación fluida), consiguiendo así que en 20 iteraciones el proyectil alcance su destino en todas las coordenadas (en el eje Z la profundidad a alcanzar será la profundidad a la que se encuentre la diana). Este calculo de la distancia a recorrer se puede observar en la Ilustración 11. Una vez hemos calculado esa distancia a recorrer en cada iteración observando las coordenadas destino (Ilustración 12), ajustaremos la función disparo para aumentar o decrementar la posición de cada coordenada en función del destino seleccionado, esto es, actualizar la posición del proyectil para que se dirija hacia cualquier punto del eje de coordenadas, completando así la función disparo para un tiro lineal (véase Ilustración 13).

```

//////////distancia entre puntos//////////
float distanciaX() {
    return sqrt(pow(raton[0] - shootX,2) )/20;
}

float distanciaY() {
    return sqrt( pow(raton[1] - shootY,2) )/20;
}

float distanciaZ() {
    return sqrt( pow(ballZ - shootZ,2) )/20;
}

```

Ilustración 11: Cálculo de la distancia del proyectil a las coordenadas seleccionadas con el ratón

```

switch (button) {
    case GLUT_LEFT_BUTTON:

        if(state == GLUT_DOWN && shootZ == 5.0){
            glutPassiveMotionFunc(NULL);
            dZ= distanciaZ();
            dY= distanciaY();
            dX= distanciaX();
            glutIdleFunc(shoot);
        }
        break;
}

```

Ilustración 12: Calcular las distancias de las coordenadas antes de ejecutar la función disparo

```

//////función de la animación del disparo//////////.
void shoot(void){ //usaremos raton[]
    if(shootZ > ballZ){
        if (raton[0]>0 && raton[1]>0){
            shootX += dX;
            shootY += dY;
            shootZ -= dZ;
        }

        else if (raton[0]<0 && raton[1]>0){
            shootX -= dX;
            shootY += dY;
            shootZ -= dZ;
        }

        else if (raton[0]<0 && raton[1]<0){
            shootX -= dX;
            shootY -= dY;
            shootZ -= dZ;
        }

        else if (raton[0]>0 && raton[1]<0){
            shootX += dX;
            shootY -= dY;
            shootZ -= dZ;
        }
        //caso en que las coordenadas X o Y sean 0
        else (raton[0]==0 && raton[1]>=0){
            shootX -= dX;
            shootY += dY;
            shootZ -= dZ;
        }
    }
}

```

Ilustración 13: Función disparo lineal a cualquier punto del eje de coordenadas

En la Ilustración 12 podemos observar que se ha fijado la ejecución del disparo solo cuando se haga el primer click del ratón y el proyectil se encuentre en su posición inicial.

Con la funcionalidad básica implementada, se va a completar el resto del juego añadiendo un marcador ([4]) que irá recogiendo los puntos obtenidos (se puede ver en la Ilustración 15) al acertar a la diana y que la diana se reposicione aleatoriamente en caso de que se acierte el disparo (Ilustración 14).

```
//posicionar las dianas en la escena
void diana(void){

    numeroX = drand48() * (9.0-0.0) - 4.5;
    numeroY = drand48() * (3.5-0.0);

    ballX = numeroX;
    ballY = numeroY;
    ballZ = -1.0;
}
```

Ilustración 14: Posicionar diana aleatoriamente en la escena

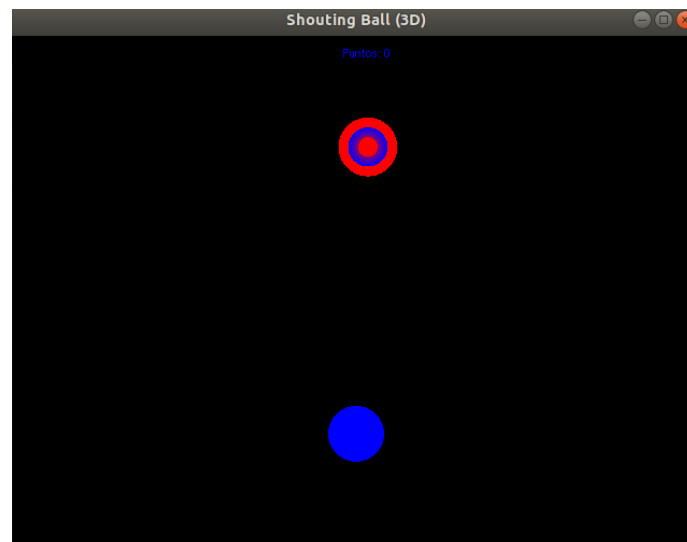


Ilustración 15: Versión final del tiro lineal

Como se observa en la Ilustración 15, la diana ha sido modificada para marcar tres zonas, las cuales proporcionan distinta puntuación cuando se realiza impacto en ellas. Esto se ha conseguido dibujando otras dos circunferencias más pero con un menor radio y de diferente color, consiguiendo así que se visualice una diana con varias zonas. El juego se muestra como en la Ilustración 15, siendo ya completamente funcional. Este finaliza al pulsar la tecla ESC.

Llegados a este punto, se ha querido añadir un poco más de dificultad al juego, por lo que se ha desarrollado otro modo de disparo, cambiando el supuesto tiro lineal que habíamos implementado por un tiro parabólico que proporciona al lanzamiento una determinada fuerza o velocidad al inicio del disparo (véase Ilustración 16).

```

/////función de la animación del disparo////////////////////////////////////
void shoot(void){ //usaremos raton[]
    if(shootZ > ballZ){
        if(shootZ > 1.0){

            if (raton[0]>=0.0 && raton[1]>=-0.7){
                shootX += dX;
                shootY += dY+fuerza;
                shootZ -= dZ;
            }

            else if (raton[0]<=0.0 && raton[1]>=-0.7){
                shootX -= dX;
                shootY += dY+fuerza;
                shootZ -= dZ;
            }

            else if (raton[0]<=0.0 && raton[1]<=-0.7){
                shootX -= dX;
                shootY -= dY-fuerza;
                shootZ -= dZ;
            }

            else if (raton[0]>=0.0 && raton[1]<=-0.7){
                shootX += dX;
                shootY -= dY-fuerza;
                shootZ -= dZ;
            }
        }
    }
}

```

Ilustración 16: Fragmento de la función disparo con una fuerza inicial

Cuando recorre una distancia, el disparo pierde esa fuerza y experimenta una caída, lo que completa el efecto del tiro parabólico (Ilustración 17, en el que solo se añade un fragmento de como se ha implementado la caída).

```

    if (raton[0]>=0.0 && raton[1]>=-0.7){
        shootX += dX;
        shootY -= dY-caída;
        shootZ -= dZ;
    }

```

Ilustración 17: Fragmento de la función disparo con caída

Por último, indicar que este disparo se realizará siempre desde la misma posición, disparando el proyectil hacia arriba partiendo de la posición -0.7 en el eje Y (de ahí que se tome en cuenta este valor como referencia en la coordenada Y en la Ilustración 16 y 17 para aumentar o reducir su posición). Por tanto, para mejorar el apuntado y que sea más sencillo apuntar y ver los efectos de la caída en el tiro, se ha dibujado una mira que te muestra la posición a la que se apunta (ejemplo en la ilustración 18, la pequeña bola azul indica donde se está apuntando), completando así la animación del juego y sus funcionalidades.

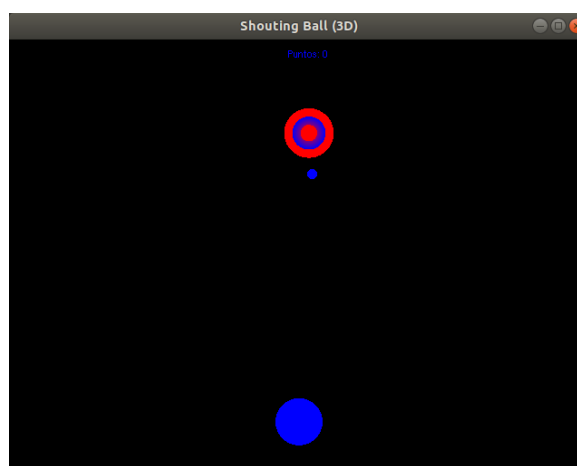


Ilustración 18: Versión final tiro parabólico

5.4 Integración del mando en el PC

Comentar que encontramos un ejemplo para el Wiimote llamado **wmgui**(buscando en Synaptic).

Después de instalarlo veremos que se trata de una pequeña aplicación gráfica que nos permite monitorizar los sensores del mando.

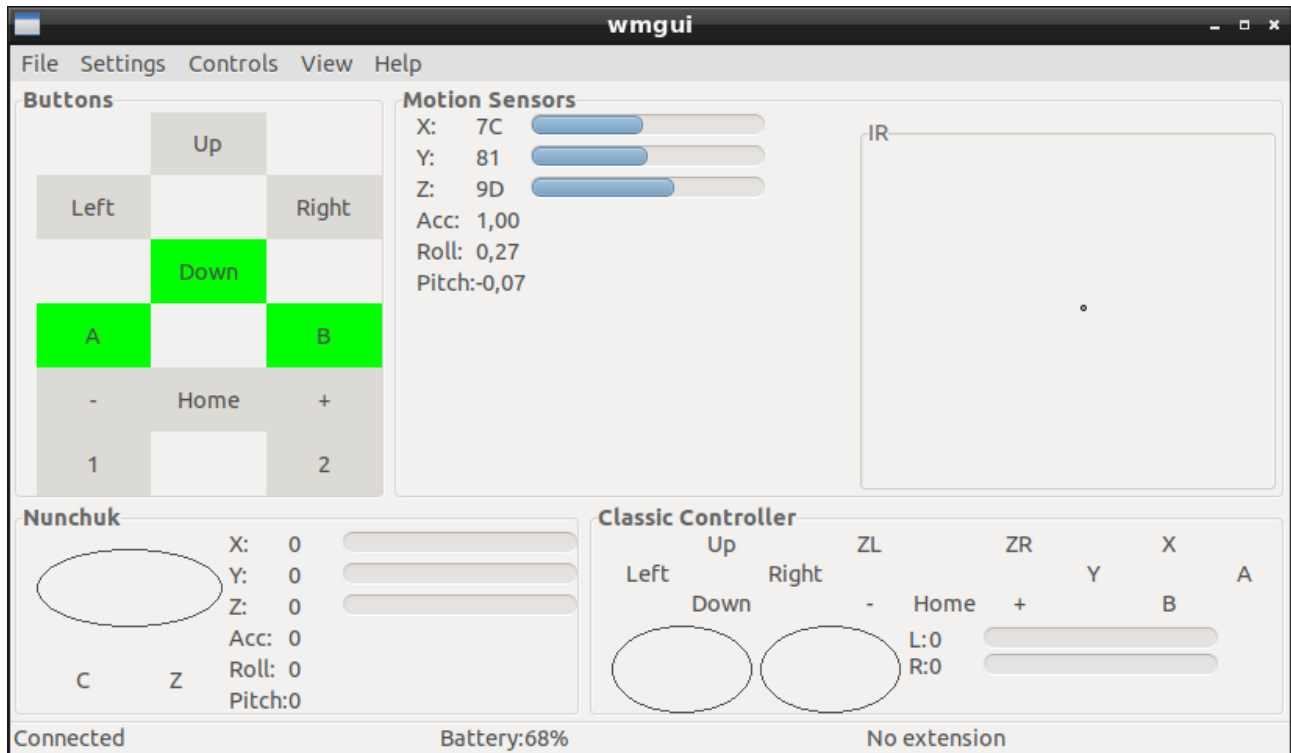


Ilustración 19: En verde los botones pulsados, al centro información del acelerómetro y a la derecha aparece un punto negro que es donde apunta el IR.

*Para que el sensor IR del Wiimote funcione es necesario que reciba luz infrarroja, en la consola esto se hacía con la barra la barra de sensores que dispone de 4 fuentes de IR; en nuestro caso lo hacemos con un mando a distancia de TV normal y corriente.

También encontramos la librería cwiid colgada en Github[5]. Esta librería es una de las varias que implementa la comunicación de los Wiimotes con el computador.

La descargamos y vimos que hay una carpeta llamada wmdemo.

La carpeta contiene una demo llamada **wmdemo.c**, la cual compilamos para obtener un ejecutable con el que empezar a trastear con el mando.

Para obtener esa demo:

- partir de wmdemo.c y añadir estas dos líneas:
 - `#include <bluetooth/bluetooth.h>`
 - `#include <cwiid.h>`

- compilar con: **gcc wmdemo.c -o wmdemo `pkg-config cwiid --cflags -libs`**
- **./wmdemo** y pulsamos 1+2 en el Wiimote para emparejarlo y veremos lo siguiente.

```
nacho@nacho-pc: ~/Desktop/IMD/cwiid_github/cwiid-master/wmdemo
Archivo Editar Pestañas Ayuda
nacho@nacho-pc:~/Desktop/IMD/cwiid_github/cwiid-master/wmdemo$ ./wmdemo
Put Wiimote in discoverable mode now (press 1+2)...
Note: To demonstrate the new API interfaces, wmdemo no longer enables messages by default.
Output can be gathered through the new state-based interface (s), or by enabling the messages interface (m).
1: toggle LED 1
2: toggle LED 2
3: toggle LED 3
4: toggle LED 4
5: toggle rumble
a: toggle accelerometer reporting
b: toggle button reporting
c: enable motionplus, if connected
e: toggle extension reporting
i: toggle ir reporting
m: toggle messages
p: print this menu
r: request status message ((t) enables callback output)
s: print current state
t: toggle status reporting
x: exit
-
```

Ilustración 20: La demo nos ofrece varias opciones, veamos como usarlas.

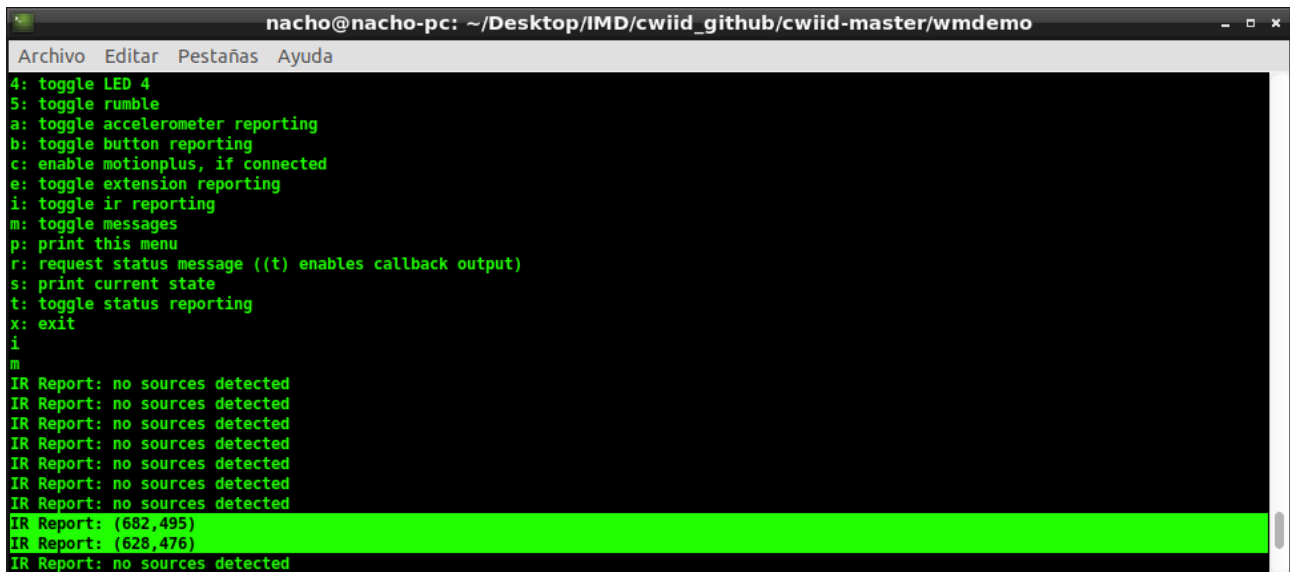
Para comprobar que funcionan los botones habilitamos “button reporting” con la letra ‘b’ y Enter.

Luego con ‘m’ habilitamos que la demo muestre mensajes cuando se produce un evento, pulsamos Enter, y cuando pulsemos algún botón aparecerá esto:

```
nacho@nacho-pc: ~/Desktop/IMD/cwiid_github/cwiid-master/wmdemo
Archivo Editar Pestañas Ayuda
b: toggle button reporting
c: enable motionplus, if connected
e: toggle extension reporting
i: toggle ir reporting
m: toggle messages
p: print this menu
r: request status message ((t) enables callback output)
s: print current state
t: toggle status reporting
x: exit
b
m
Button Report: 0004
Button Report: 0000
Button Report: 0008
Button Report: 0000
Button Report: 0001
Button Report: 0000
Button Report: 0002
Button Report: 0000
Button Report: 0001
Button Report: 0000
Button Report: 0004
Button Report: 0000
Button Report: 0100
```

Ilustración 21: Devuelve 0000 cuando se suelta un botón, por ejemplo el 0004 es el gatillo.

Con 'i' habilitamos "IR reporting", y con la 'm' empezará a monitorizar el sensor IR(debemos estar con el mando TV apuntando al Wiimote), devolviendo una posición en formato (x,y), de dónde apunta el mando. "No sources detected" aparece porque el mando a distancia no envía un pulso continuo de luz infraroja.



```
nacho@nacho-pc: ~/Desktop/IMD/cwiid_github/cwiid-master/wmdemo
Archivo  Editar  Pestañas  Ayuda
4: toggle LED 4
5: toggle rumble
a: toggle accelerometer reporting
b: toggle button reporting
c: enable motionplus, if connected
e: toggle extension reporting
i: toggle ir reporting
m: toggle messages
p: print this menu
r: request status message ((t) enables callback output)
s: print current state
t: toggle status reporting
x: exit
i
m
IR Report: no sources detected
IR Report: no sources detected
IR Report: no sources detected
IR Report: no sources detected
IR Report: no sources detected
IR Report: no sources detected
IR Report: no sources detected
IR Report: (682,495)
IR Report: (628,476)
IR Report: no sources detected
```

Ilustración 18: Captura IR reporting de wmdemo.

El primer ejemplo wmgui es menos interesante pero nos sirvió para asegurarnos que el mando conectaba al PC correctamente y los botones, acelerómetro y sensor IR también funcionaban. Sin embargo gracias al código de wmdemo.c comenzamos a codificar.

Para integrar el mando de la Wii tanto en el PC como en nuestra aplicación comenzaremos por comentar como conectarlo, para ello debemos declarar estas variables:

```
17     cwiid_wiimote_t *wiimote; /* wiimote handle */
18     struct cwiid_state state; /* wiimote state */
19     bdaddr_t bdaddr; /* bluetooth device address */
20
```

Ilustración 19: Variables necesarias

Posteriormente, en el main:

```
613
614  /* Main function: GLUT runs as a console application starting at main() */
615  int main(int argc, char** argv) {
616
617
618      unsigned char rpt_mode = 0;
619
620      /* Connect to address given on command-line, if present */
621      if (argc > 1) {
622          str2ba(argv[1], &bdaddr);
623      }
624      else {
625          bdaddr = *BDADDR_ANY;
626      }
627
628      /* Connect to the wiimote */
629      printf("Put Wiimote in discoverable mode now (press 1+2)...\\n");
630      if (!(wiimote = cwiid_open(&bdaddr, 0))) {
631          fprintf(stderr, "Unable to connect to wiimote\\n");
632          return -1;
633      }
```

Ilustración 20: La función cwiid_open conecta por bluetooth nuestro mando

6 Problemas encontrados y trabajos futuros.

Durante el desarrollo del proyecto hemos encontrado algunos problemas que no hemos podido solucionar, dejando algún objetivo por alcanzar.

- A la hora de adaptar las coordenadas que detecta el ratón a nuestra escena 3D, se han traducido utilizando unos valores introducidos manualmente de forma intuitiva, por lo que la transformación de las coordenadas no es 100% exacta. Esto también provoca que si hay cambios en la dimensión de la ventana, la animación del disparo no irá al punto deseado, habiendo un pequeño desvío.
- En cuanto al apuntado y disparo con el mando en el juego, hemos tenido problemas debido a la falta de tiempo para entender completamente los métodos y estructura(métodos de conexión, obtener estado, printarlo, etc...) y poder haber codificado una versión completa. De hecho en wmdemo.c esta todo lo que hubiésemos requerido.
- Como trabajo futuro, se propone al lector interesado finalizar con los problemas mencionados justo arriba, para lo que seria conveniente seguir perfeccionando la función reshape y la transformación de las coordenadas del ratón al escenario(obtener la posición sin desviación del ratón independientemente del tamaño de la ventana) y seguir indagando en el código de wmdemo.c sobretodo funciones print_state, request_state, y estructuras asociadas al mando y sus sensores y botones; para integrar totalmente el mando en el juego.

7. Conclusión

Como conclusión, comentar que a pesar de no haber podido alcanzar el objetivo de jugar al shooter con el Wiimote no pensamos que el proyecto haya sido un completo fracaso, pues durante el desarrollo del mismo hemos aprendido algunos conceptos importantes que desconocíamos o que hasta cursar IMD no habíamos interiorizado.

Conceptos como desde el simple gestor de paquetes Synaptic hasta instalación de librerías, cabeceras necesarias, problemas con las versiones o el uso correcto del compilador y pkg-config.

8. Bibliografía y referencias

- [1] Bouncing Ball: Código para realizar la diana, apartado 6,8 Example 7.
C. Hock-Chuan. (2012). OpenGL Tutorial. An Introduction on OpenGL with 2D Graphics.
https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_Introduction.html#zz-6.8
- [2] Ejemplo openal.c:
<http://www.edenwaith.com/products/pige/tutorials/openal.php>
- [3] Transformación de coordenadas:
<https://www.lawebdelprogramador.com/foros/Open-GL/908564-Transformacion-de-coordenadas.html>
- [4] Cálculo y mostrar puntuación:
<https://www.foro.lospillaos.es/c-con-opengl-shotter-en-2d-con-dianas-vt6723.html>
- [5] Cwiid GitHub:
<https://github.com/abstrakraft/cwiid>