Tutorial de Sonido

Sumario

Ver los anteriores tutoriales básicos para más información sobre la creación básica de objetos, manejo del reloj, jerarquía de fotogramas, animaciones y cámaras & vistas.

Este tutorial muestra como reproducir sonidos(muestras) y músicas(streams). Como con otras características de comportamientos anteriores, solo se requiere, la mayoría de las veces, una sola linea de código, para empezar a manejar todos los datos. Este tutorial también demuestra como alterar las configuraciones del sonido en tiempo real, usando el gráfico del soldado como retroalimentación visual.

Si tu presionas las flechas arriba & abajo, el volumen de la música cambiará acordemente. El soldado se escalará como consecuencia. Por presionar las flechas izquierda & derecha, el tono de la música(frecuencia) cambiará. El soldado rotara como una perilla. La tecla Control Izquierda reproducirá la música(y activará el soldado) si la música fue pausada, de otra manera pausará la música(y desactivará el soldado). El efecto de sonido de la tecla Espacio presionada es la misma que Enter, excepto, de que su volumen y tono son aleatoriamente definidos en fichero de configuración por defecto.

Esta aleatoriedad de la frecuencia controlada en la configuración permite fácilmente generar de paso o golpeo de sonido con ligeras variaciones con ninguna linea adicional de código. Cambiamos aleatoriamente el color del soldado para ilustrar esto. El efecto de sonido solo se añadirá y reproducirá con el soldado activo. Si tu quieres reproducir un efecto de sonido sin ningun objeto como soporte, tu lo puedes hacer de la misma manera que creamos la música en este tutorial. Sin embargo, reproduciendo un sonido en un objecto permitiría posicionar el sonido espacialmente(no cubierto en este tutorial).

Muchos efectos de sonido pueden ser reproducidos al mismo tiempo en un solo objeto.

El atributo KeepDataInCache de configuración del sonido permite mantener la muestra de sonido en la memoria en vez de releerlo del fichero constantemente. Esto solo funciona para datos no-stream(ej. No para música). Si esto fuera falso, la muestra será recargada desde fichero, a menos que no haya otro efecto de sonido del mismo tipo que esté siendo reproducido.

También registramos los eventos de sonidos para mostrar cuando los efectos de sonidos son reproducidos y detenidos. Estos eventos son solo enviados por efectos de sonido reproducidos en objetos.

Detalles

Como es usual, comenzaremos por cargar nuestro fichero de configuración, obteniendo el reloj principal y registrando nuestra función `Update` a este y, por último, por crear nuestro objeto soldado. Por favor, diríjase a los tutoriales previos para más detalles.

Creamos entonces una música y la reproducimos.

```
orxSOUND *pstMusic;
pstMusic = orxSound_CreateFromConfig("Music");

orxSound_Play(pstMusic);
```

Como podemos ver, música y sonido son de tipo `orxSOUND`. La diferencia principal es que la música escuchada mientras el sonido es completamente cargada en memoria. Como veremos más adelante, definiremos esta diferencia en el fichero de configuración.

El último paso de nuestra función `Init`: nos suscribimos a los eventos de sonido.

```
orxEvent_AddHandler(orxEVENT_TYPE_SOUND, EventHandler);
```

Solo registramos cuando los sonidos son iniciados/parados, echemos un vistazo al código correspondiente.

```
orxSOUND_EVENT_PAYLOAD *pstPayload;

pstPayload = (orxSOUND_EVENT_PAYLOAD *)_pstEvent->pstPayload;

switch(_pstEvent->eID)
{
  case orxSOUND_EVENT_START:
    orxLOG("Sound <%s>@<%s> has started!", pstPayload->zSoundName,
orxObject_GetName(orxOBJECT(_pstEvent->hRecipient)));
    break;

  case orxSOUND_EVENT_STOP:
    orxLOG("Sound <%s>@<%s> has stoped!", pstPayload->zSoundName,
orxObject_GetName(orxOBJECT(_pstEvent->hRecipient)));
    break;
}

return orxSTATUS_SUCCESS;
```

Nada realmente nuevo como puedes ver.

Veamos como añadimos sonidos a nuestro soldado.

```
if(orxInput_IsActive("RandomSFX") && orxInput_HasNewStatus("RandomSFX"))
{
  orxObject_AddSound(pstSoldier, "RandomBip");
  orxObject_SetColor(pstSoldier, orxColor_Set(&stColor,
orxConfig_GetVector("RandomColor", &v), orxFLOAT_1));
}

if(orxInput_IsActive("DefaultSFX") && orxInput_HasNewStatus("DefaultSFX"))
{
  orxObject_AddSound(pstSoldier, "DefaultBip");
  orxObject_SetColor(pstSoldier, orxColor_Set(&stColor, &orxVECTOR_WHITE,
```

```
orxFLOAT_1));
}
```

Como podemos ver, añadir un sonido a un objeto solo requiere una única linea de código y, más importante, se hace de la misma manera para nuestro sonido aleatorio y nuestra única frecuencia fija. Como veremos después, la diferencia esta solamente expresada en el fichero de configuración. Cuando añadimos sonido `RandomBip`, solo cambiamos el color de nuestro soldado aleatoriamente, ya que está definido en nuestro fichero de configuración con la llave `RandomColor`. Cuando reproducimos un `DefaultBip` en el, simplemente regresamos su color a blanco.

PD: Un sonido será reproducido en el soldado siempre y cuando la correspondiente entrada sea activada.
Hasta ahora solo importaba si una entrada estaba activa o no, aquí queremos hacer algún tipo de acción en el preciso momento que la entrada es activada.
Para hacer eso, usaremos la función `orxInput_HasNewStatus()` que retornará `orxTRUE` cuando la entrada vaya de pasiva a activa y, inversamente, cuando vaya de activa a pasiva.
Utilizando el resultado de esta función a lo largo de la que obtenemos de `orxInput_IsActive()`, se asegura de que sólo se reproducirá el sonido cuando la entrada va de pasivo a activo!

Ahora juguemos con la activación de la música.

```
if(orxInput_IsActive("ToggleMusic") && orxInput_HasNewStatus("ToggleMusic"))
{
  if(orxSound_GetStatus(pstMusic) != orxSOUND_STATUS_PLAY)
  {
    orxSound_Play(pstMusic);
    orxObject_Enable(pstSoldier, orxTRUE);
  }
  else
  {
    orxSound_Pause(pstMusic);
    orxObject_Enable(pstSoldier, orxFALSE);
  }
}
```

Este código simple, cuando activando la entrada `ToggleMusic`, o comenzamos la música y activamos nuestro soldado si la música no se estaba reproduciendo o, por el contrario, la pararemos y desactivamos nuestro soldado si la música se estaba reproduciendo.

Ahora, vamos a cambiar el terreno de juego.

```
if(orxInput_IsActive("PitchUp"))
{
  orxSound_SetPitch(pstMusic, orxSound_GetPitch(pstMusic) + orx2F(0.01f));
  orxObject_SetRotation(pstSoldier, orxObject_GetRotation(pstSoldier) +
orx2F(4.0f) * _pstClockInfo->fDT);
}
```

Nada realmente sorprendente aquí. Haremos lo mismo con valores opuestos para la entrada `PitchDown`.

Por último cambiemos el volumen.

```
if(orxInput_IsActive("VolumeDown"))
{
  orxSound_SetVolume(pstMusic, orxSound_GetVolume(pstMusic) - orx2F(0.05f));
  orxObject_SetScale(pstSoldier, orxVector_Mulf(&v,
orxObject_GetScale(pstSoldier, &v), orx2F(0.98f)));
}
```

Lo mismo, nada inusual.

PD: Podemos notar que la rotación de nuestro único objeto será consistente en el tiempo (tutorial del reloj).
El tono de la música y el volumen, así como la escala del objeto serán dependientes de las imagenes por segundo, lo cual es una mala idea.
Para arreglar esto solo necesitamos usar el reloj DT para ponderar los valores dados.

Hemos terminado con el código. Miremos entonces hacia los datos!

Primero, definimos nuestra música.

```
[Music]
Music = ../../data/sound/gbloop.ogg
Loop  = true
```

Bastante fácil! Si no tuviéramos que preguntarle explícitamente, la música no pudiera ser cíclica cuando fuera reproducida.

Miremos ahora nuestro `DefaultBip`.

```
[DefaultBip]
Sound       = ../../data/sound/bip.wav
KeepInCache = true;
Pitch       = 1.0
Volume      = 1.0
```

Como dijimos anteriormente, el atributo `KeepInCache` se asegurará de que este sonido no sea descargado automáticamente de la memoria.

El tono y el volumen están explícitamente definidos a su valor por defecto que no se necesita realmente.

Por último, veamos nuestro `RandomBip`.

```
[RandomBip@DefaultBip]
Pitch   = 0.1 ~ 3.0
Volume  = 0.5 ~ 3.0
```

Como podemos ver, nuestro `RandomBip` hereda desde `DefaultBip`. Esto significa que si cambiamos la muestra por `DefaultBip`, también será cambiado por `RandomBip`.
Ademas de eso, solo preguntamos por valores aleatorios para el Tono(`Pitch`, ej. frecuencia) y el

volumen.
Esto significa que el `RandomBip` será reproducido todo el tiempo, que tendrá una frecuencia y volumen diferentes, pero no se requiere nada de código sabio para lograr este comportamiento.

Recursos

Código fuente: 06_Sound.c

Fichero de configuración: 06_Sound.ini