Tutorial de Animaciones

Resumen

Veasé los anteriores tutoriales básicos para más información acerca de creación básica de objetos, manejo del reloj y fotogramas.

Este tutorial solo cubre un uso muy básico de las animaciones en orx.

Todas las animaciones son guardadas en grafo dirigido.
Este gráfico define todas las posibles transisiones entre animaciones. Una animación es referenciada usando un único caracter de cadena. Todas las transiciones y animaciones son creadas via ficheros de configuración.

Cuando se solicita una animación, el motor evaluará la cadena que lo traerá a esta animación desde la actual.
Si esa cadena existe, a continuación se procesará automáticamente. El usuario será notificado cuando se inician las animaciones, se detienen, se corten o estén en un bucle de eventos.
Si no especificamos ninguna animación como objetivo, el motor va a seguir los enlaces naturalmente, de acuerdo a sus propiedades [1)].
También hay una manera de saltarse este procedimiento de encadenamiento y forzar una animación inmediatamente.

El Código-sabio de este sistema es muy fácil de usar con dos funciones principales para manejar todo. La mayoría del trabajo se realiza no en el código, sino en los archivos de configuración cuando se definen las animaciones y enlaces. [2)]

Detalles

Como de costumbre, comenzamos por la carga de nuestro archivo de configuración, creando una ventana, obteniendo el reloj principal y registrando nuestra función `Update` a la misma y, por último, mediante la creación de nuestro objeto principal.
Por favor, consulte los tutoriales anteriores para más detalles.

Ahora comencemos con el codigo, veamos como organizar los datos hasta el final de esta página.
En nuestra función `Update` solo tendremos que activar la animación `WalkLeft` cuando la entrada `GoLeft` está activa y la animación `WalkRight` cuando la entrada `GoRight` está activa.
Cuando ninguna entrada está activa, simplemente removemos el objetivo de animación y dejamos que el gráfico sea evaluado normalmente[3)].

```
if(orxInput_IsActive("GoRight"))
{
  orxObject_SetTargetAnim(pstSoldier, "WalkRight");
}
else if(orxInput_IsActive("GoLeft"))
{
  orxObject_SetTargetAnim(pstSoldier, "WalkLeft");
```

```
}
else
{
  orxObject_SetTargetAnim(pstSoldier, orxNULL);
}
```

¡Así es! Cómo se llega desde cualquier animación actual al objetivo que será evaluado usando el gráfico. Si las transiciones son necesarias se van a reproducir automáticamente [4)].

PD: Si hubiéramos querido ir inmediatamente a otra animación sin respetar las transiciones definidas por datos (en el caso de las animaciones de golpe o muerte, por ejemplo), podríamos haber hecho esto.

```
orxObject_SetCurrentAnim(pstSoldier, "DieNow");
```

PD: Hay mas funciones para el control avanzado de animaciones (como pausando, cambiando frecuencia, ...), pero el 99% del tiempo, esas dos funciones (`orxObject_SetCurrentAnim()` y `orxObject_SetTargetAnim()`) son las únicas que necesitarías.

Veamos ahora como podemos estar informados de que sucede con nuestras animaciones (como sincronizar sonidos, por ejemplo).
Primero, necesitamos registrar nuestra devolución de llamada(callback) `EventHandler` para los eventos de animación.

```
orxEvent_AddHandler(orxEVENT_TYPE_ANIM, EventHandler);
```

Hecho! Veamos que podemos hacer con esto ahora.

Digamos que desea imprimir que animaciones son reproducidas, se detuvieron, cortaron o continuaron en nuestro objeto. A continuación, tendríamos que escribir la siguiente devolución de llamada(callback).

```
orxSTATUS orxFASTCALL EventHandler(const orxEVENT *_pstEvent)
{
orxANIM_EVENT_PAYLOAD *pstPayload;

pstPayload = (orxANIM_EVENT_PAYLOAD *)_pstEvent->pstPayload;

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

  case orxANIM_EVENT_STOP:
    orxLOG("Animation <%s>@<%s> has stoped!", pstPayload->zAnimName,
orxObject_GetName(orxOBJECT(_pstEvent->hRecipient)));
    break;
```

```
  case orxANIM_EVENT_CUT:
    orxLOG("Animation <%s>@<%s> has been cut!", pstPayload->zAnimName,
orxObject_GetName(orxOBJECT(_pstEvent->hRecipient)));
    break;

  case orxANIM_EVENT_LOOP:
    orxLOG("Animation <%s>@<%s> has looped!", pstPayload->zAnimName,
orxObject_GetName(orxOBJECT(_pstEvent->hRecipient)));
    break;
  }

  return orxSTATUS_SUCCESS;
}
```

En primer lugar, obtener la capacidad de carga de nuestro evento. Como sabemos sólo manejamos los eventos de animación aquí, podemos con seguridad emitir la capacidad de carga de `orxANIM_EVENT_PAYLOAD` definido en orxAnim.h.

Si estamos usando la misma rutina para diferentes tipos de eventos, lo primero que deberíamos ver es si estábamos recibiendo un evento animación. Esto puede hacerse con el código siguiente.

```
if(_pstEvent->eType == orxEVENT_TYPE_ANIM)
```

Finalmente, nuestro evento receptor (`_pstEvent→hRecipient`) es en realidad el objeto sobre el que se reproduce la animación. Lo emitimos como un `orxOBJECT` usando la macro de ayuda `orxOBJECT()`. [5)]

Ahora vamos a echar un vistazo al lado de los datos.

En primer lugar, tenemos que definir un set de animación que va a contener todo el gráfico para animaciones de nuestro objeto específico.
El set de animación nunca se duplicará en la memoria y contendrá todas las animaciones y enlaces para el correspondiente graficado.
En nuestro caso tenemos 4 animaciones y 10 posibles enlaces para todas las transiciones.

```
[AnimSet]
AnimationList = IdleRight#WalkRight#IdleLeft#WalkLeft

LinkList =
IdleRightLoop#IdleRight2Left#IdleRight2WalkRight#WalkRightLoop#WalkRight2Idl
eRight#IdleLeftLoop#IdleLeft2Right#IdleLeft2WalkLeft#WalkLeftLoop#WalkLeft2I
dleLeft
```

Ahora definamos nuestras animaciones! 

Anterior a eso, a fin de reducir la cantidad de texto que tenemos que escribir, vamos a utilizar la herencia del sistema de configuración de orx.
Vamos a comenzar a definir una sección para la posición de nuestro pivote [6)].
Como usted pudo haber visto en el tutorial de objeto en su archivo de configuración, el pivote del objeto es la posición que coincide con las coordenadas del espacio. Si no se especifica, la esquina

superior izquierda se utiliza por defecto.
El pivote se puede definir literalmente usando palabras clave, como `top`(superior), `down`(abajo), `center`(centro),`left`(izquierda) y `right`(derecha), o dando una posición real, en píxeles.

```
[Pivot]
Pivot = (15.0, 31.0, 0.0)
```

Ahora vamos a definir nuestro objeto gráfico que heredará de este pivote. En nuestro caso se trata de un mapa de bits que contiene todos los fotogramas de nuestro objeto.
Las propiedades comunes son por lo tanto el nombre del archivo de mapa de bits y el tamaño de un fotograma [7)]

```
[FullGraphic@Pivot]
Texture     = ../../data/anim/soldier_full.png
TextureSize = (32, 32, 0)
```

Ok, está todo preparado para la creación de todos los fotogramas.
Vamos a definir todos nuestros fotogramas, para ambas animaciones orientadas a la derecha: tenemos 6 de ellas.

```
[AnimRight1@FullGraphic]
TextureCorner = (0, 0, 0)

[AnimRight2@FullGraphic]
TextureCorner = (0, 32, 0)

[AnimRight3@FullGraphic]
TextureCorner = (0, 64, 0)

[AnimRight4@FullGraphic]
TextureCorner = (32, 0, 0)

[AnimRight5@FullGraphic]
TextureCorner = (32, 32, 0)

[AnimRight6@FullGraphic]
TextureCorner = (32, 64, 0)
```

Como puedes ver, ellas todas heredan de `FullGraphic` y la única propiedad que ellas tienen aparte es `TextureCorner`. OK, ahora vamos a definir todos esos objetos gráficos (todas ellas se convierten en estructuras `orxGRAPHIC` cuando se cargan), dejemos que las animaciones se definan solas.
Comencemos con el `IdleRight` quien contiene un solo fotograma con una duración de 0.1 segundo.

```
[IdleRight]
KeyData1      = AnimRight6
KeyDuration1  = 0.1
```

Es bastante fácil, vamos a tratar con la segunda: `WalkRight`

```
[WalkRight]
```

```
DefaultKeyDuration  = 0.1
KeyData1            = AnimRight1
KeyData2            = AnimRight2
KeyData3            = AnimRight3
KeyData4            = AnimRight4
KeyData5            = AnimRight5
KeyData6            = AnimRight6
```

Realmente no es difícil definir el mismo tiempo para todos los fotogramas usando la propiedad `DefaultKeyDuration`.
Podemos sustituir a cualquier fotograma mediante la especificación de una duración de teclas tal como lo hicimos para la animación `IdleRight`.

Vamos a hacer exactamente lo mismo para las animaciones de orientación izquierda. En realidad lo que estamos utilizando objetos gráficos volteados, sólo podríamos haber volteado el objeto en tiempo de ejecución en el código.
Pero eso no hubiera servido a nuestros fines didácticos! Vamos a suponer que estas animaciones izquierda son completamente diferentes de las derechas! 

Ahí están solamente los enlaces perdidos ahora y está hecho!.
La estructura de enlace básico es fácil: especificamos la fuente y la animación de destino.

```
[IdleRightLoop]
Source      = IdleRight
Destination = IdleRight
```

This link goes from `IdleLoop` animation to the `IdleLoop` animation. No wonder we called this link `IdleRightLoop`!
So basically, when we are in the `IdleRight` animation and we asked `IdleRight` animation as a target, we just stay there, looping.
Also, if no target is defined when we are there, this link will keep us looping as there isn't any higher priority link starting from `IdleRight`.

Let's see now how we go from `IdleRight` to `WalkRight`.

```
[IdleRight2WalkRight]
Source      = IdleRight
Destination = WalkRight
Property    = immediate
```

Here we have the same basic info as before but we also have the `immediate` value for the key `Property`.
This means that when we are in `IdleRight` animation and we target `WalkRight`, we won't wait till `IdleRight` is over to follow the link, we'll go there directly: that gives us a way to cut animations.

As we've seen in the code, we don't explicitely ask for the idle animation when we are already walking. How do this work then??
Let's see the link that goes from `WalkRight` to `IdleRight`.

```
[WalkRight2IdleRight]
```

```
Source      = WalkRight
Destination = IdleRight
Property    = immediate; <= If you remove this property, the animation won't be cut to go immediately back to idle
Priority    = 9
```

When we are in `WalkRight` and we remove our target, the engine will have to follow the links in a natural way. That means it'll favorise the higher priority links.
By default a link `Priority` is 8. It can range from 0 to 15. Having here a link at priority 9 means that this will be the one taken when we have no target.
It will bring us back to `IdleRight`.
We also have added the `immediate Property` so that we won't wait the end of the walk cycle to go back to idle.

NB: This is a very basic graph that shows only basic transitions, but the system is very expandable. Let's say you want to begin walking from a sitting pause without transition.
But, later in the game development, you want to add a standing up transition for it to look nicer. You'll only have to add this extra step (with the associated links) in the config file! Your code will remain unchanged:

```
orxObject_SetTargetAnim(MyObject, "Walk");
```

Recursos

1)

tales como contadores de bucles que no serán cubiertos por este tutorial básico

2)

Un gráfico de animación muy básica será utilizada para este tutorial: lo hicimos a fin de mantener limitada la cantidad de datos de configuración necesarios.

3)

en nuestro caso, va a reproducir la animación de inactividad correspondiente, incluso si nuestros datos sólo contienen un único fotograma para cada uno

4)

recordar que en nuestro caso nos fuimos por el camino más recto, sin animaciones de giro, por ejemplo; pero eso no iba a cambiar nuestro código en absoluto!

5)

Recuerde que usamos tales macros con el fin de asegurarnos de que estamos lanzando el tipo correcto.

6)

también llamado `HotSpot` en algunos motores

7)

que no tiene que mantenerse constante, pero por lo general es más fácil para los artistas y eventualmente sea un obstáculo para algunos otros motores y bibliotecas

From:
https://www.orx-project.org/wiki/ - **Orx Learning**

Permanent link:
https://www.orx-project.org/wiki/es/orx/tutorials/anim?rev=1330542700

Last update: **2025/09/30 17:26 (14 months ago)**