Using orxObjects in Classes with an EventHandler

Using an EventHandler is handy to use to update your orx Objects (like movement) as I demonstrated in my last tutorial. You could have many objects being updated by the EventHander when they animate and all would well. Even if your object is embedded into a custom class, orx can still update your object.

But what if your class contained an UpdateMovement() function? As the Eventhandler only knows about orx objects, how would it know to go that specific function in that specific class for that orx object? The solution is very cool...

This tutorial aims to show you how to:

1) Embed an orxObject into a custom class.

2) Create a function for the class that does some simple movement.

3) Associate an orx object with an instance of a custom class.

4) Enable an EventHandler to update an orxobject but also access class functions assoctiated with the orx object.

Prerequisite

You are best to first be familar with the Realistic Walking Movement Tutorial or 04 Anim Tutorial before this one.

Orx objects in custom classes

Of course, as any game grows your character will become more than just a simple orx object. There are many attributes like health, lives, etc etc. So it is handy to have a class that contains these attributes and the orx object together. This can be best illustrated with an enemies class:

```
class Enemy {
private:
    orxOBJECT *enemyObject;
    orxVECTOR enemyPos;

    int energy;
    bool dead;

public:
    Enemy() {
```

```
        enemyObject    =
orxObject_CreateFromConfig("EnemyObjectNameFromConfig");

        energy = 10000;
        dead = false;
    }

    void ReduceEnergy() {
        if (energy > 0){
            energy--;
        }
    }

    void Animate() {
        orxObject_SetTargetAnim(enemyObject, "Jump");
    }
};
```

Having many enemies

Let's start by setting up three enemies and animating them:

```
Enemy *enemy01 = new Enemy();
Enemy *enemy02 = new Enemy();
Enemy *enemy03 = new Enemy();

enemy01->Animate();
enemy02->Animate();
enemy03->Animate();
```

The EventHandler

As usual, we want the EventHander to catch the fact that our three objects are animating. If they are, we want it to execute the ReduceEnergy() function in each class object that each orx object belongs to:

```
orxSTATUS orxFASTCALL EventHandler(const orxEVENT *_pstEvent) {
    orxANIM_EVENT_PAYLOAD *pstPayload;
    pstPayload = (orxANIM_EVENT_PAYLOAD *)_pstEvent->pstPayload;
    const orxSTRING animName =     pstPayload->zAnimName;

    if (     orxString_Compare(
orxObject_GetName(orxOBJECT(_pstEvent->hRecipient)),
"EnemyObjectNameFromConfig" )   == 2 )
```

```
        return orxSTATUS_SUCCESS;

switch(_pstEvent->eID) {
    case orxANIM_EVENT_START:
    {

      orxOBJECT *thisObj = orxOBJECT(_pstEvent->hRecipient);
      //uh oh.. No way to get to the orx object's class! How to execute the
ReduceEnergy() function?

      break;
    }
```

Ok, there's the problem. The EventHandler knows about the orx object, the recipient of the current playing animation (_pstEvent→hRecipient). But not the class that the orx object belongs to, and no way to access any of its functions.

The Solution

There is an elegant way to fix this. Orx provides two very handy object functions:

- orxObject_SetUserData
- orxObject_GetUserData

To use orxObject_SetUserData, let's head back the constructor of our Enemy class:

```
    Enemy() {
        enemyObject    =
orxObject_CreateFromConfig("EnemyObjectNameFromConfig");

        energy = 10000;
        dead = false;

        orxObject_SetUserData(enemyObject, this);
    }
```

What this means is, associate the current Enemy class object with the orx object.

Now back into the EventHandler to use orxObject_GetUserData:

```
    case orxANIM_EVENT_START:
    {

        orxOBJECT *thisObj = orxOBJECT(_pstEvent->hRecipient);
        Enemy *e = (Enemy *)orxObject_GetUserData( thisObj  );
        e->ReduceEnergy();
```

```
    break;
  }
```

Now you can access the class functions that the orx object belongs to from within an EventHandler.