How to create a Progress Bar in orxScroll

Having a progress bar to represent health meters, damage or fuel reserves is a common element in games. The object oriented orxScroll wrapper for Orx is the perfect tool for making components like these for your game.

Create a new orx/Scroll Project

Create a new orx/Scroll project by following the steps here.

Class for the Progress Bar

Create a new ProgressBar.cpp class file and ProgressBar.h header:

```
#ifndef __PROGRESSBAR_H__
#define __PROGRESSBAR_H__
#include "mygame.h"

class ProgressBar : public ScrollObject
{
public:
protected:

    void            OnCreate();
    void            OnDelete();
    void            Update(const orxCLOCK_INFO &_rstInfo);

private:
    orxFLOAT        progressPercent;
    orxBOOL         isPaused;
    void            ColourTheme(orxVECTOR backgroundColour, orxVECTOR
levelColour);
};

#endif // __PROGRESSBAR_H__
```

```
#include "ProgressBar.h"

void ProgressBar::OnCreate()
{
    orxConfig_SetBool("IsObject", orxTRUE);
}

void ProgressBar::OnDelete()
{
```

```
}

void ProgressBar::Update(const orxCLOCK_INFO &_rstInfo)
{
}
```

In your main code file, which I'll call mygame.cpp, add the binding for your ProgressBar class to your `BindObjects` function:

```
void myGame::BindObjects()
{
    ScrollBindObject<ProgressBar>("ProgressBar");
}
```

And include the file at the top of `mygame.cpp` so that your code knows about your new class:

```
#include "ProgressBar.h"
```

The Data Config

You should be able to compile now, but we still need to define our ProgressBar object in our config. Let's do that now:

```
[ProgressBar]
ChildList   = ProgressBarBackground # ProgressBarLevel
Pivot       = center

[ProgressBarBackground]
Graphic     = @
Texture     = pixel
Size        = (600, 25)
Color       = cornflowerblue
Pivot       = center

[ProgressBarLevel]
Graphic     = @
Texture     = pixel
Size        = (592, 18)
Scale       = (0.5, 1)
Position    = (-296, 0, -0.1)
Color       = white
Pivot       = left
```

The `ProgressBarBackground` object is a simple solid rectangle of dark blue pixels which provides a border for the inner bar that grows or shrinks.

`ProgressBarLevel` is the inner rectangular pixel bar that changes it's size over time:

To create the progress bar in the game, remove the scene creator in the `Init` function:

```
CreateObject("Scene");
```

And replace with:

```
progressBar = CreateObject("ProgressBar");
```

And declare the progressBar ScrollObject pointer at the top of the the mygame.cpp file:

```
#include "ProgressBar.h"
ScrollObject *progressBar;
```

Now compile up and you should see the progress bar in the centre of the screen.

Because we have a ScrollObject that represents our ProgressBar, we can use the `Update` inside our ProgressBar class and change the bar over time. Add a private variable to the ProgressBar.h header to track what percent the bar should grow to:

```
private:
    orxFLOAT        progressPercent = 75;
```

Add the following code to the `Update` function in ProgressBar.cpp to make the size change over time:

```
void ProgressBar::Update(const orxCLOCK_INFO &_rstInfo)
{
    //cheat a little to get the ProgressBarLevel
    ScrollObject *object = this->GetOwnedChild();
    if (orxString_Compare(object->GetModelName(), "ProgressBarBackground")
== 0) {
        //want ProgressBarLevel instead
        object = object->GetOwnedSibling();
    }
    progressPercent -= 0.1;
    if (progressPercent < 0)
        progressPercent = 100;
    orxFLOAT fractionOfProgress = progressPercent/100;
    orxVECTOR sv = {fractionOfProgress, 1};
    object->SetScale(sv, orxFALSE);
}
```

(I used a cheap routine to get to the right child object - however, this is a better function to use).

That's pretty good! The bar shrinks over time like a health bar. However, because the ProgressBar is a

ScrollObject and therefore has it's own class, we can add our own functions to add more functionality.

This is what makes working with the orxScroll wrapper much easier than working in plain orx: everything is object orientated, and is therefore easy to scale and extend, neat and tidy. Less chance of getting messed up and confused.

Let's add a function to set the function at any time with a key press.

Extending our class for more features

First, let's add a public function to the ProgressBar.h header file:

```
public:
    void SetProgressPercent(orxFLOAT percent);
```

And add the function to the ProgressBar.cpp class:

```
void ProgressBar::SetProgressPercent(orxFLOAT percent){
    progressPercent = percent;
}
```

We need a key to press to change the percent to something like 90% every time it is pressed. In the config, set to spacebar for this:

```
[Input]
KEY_ESCAPE      = Quit
KEY_SPACE  = SetPercentPressed
```

Then, in the `Update` function of `mygame.cpp` where the quit key is being tested, let's add our key there:

```
void myGame::Update(const orxCLOCK_INFO &_rstInfo)
{
    if (orxInput_HasBeenActivated("SetPercentPressed")){
        progressBar->SetProgressPercent(90);
    }
...
...
```

Oops a problem right? Either your IDE's intellisense has picked up that there is no such function there, or the compiler has complained.

This is because our progressBar is of a type of `ScrollObject`, and we need to cast it to a type of `ProgressBar` to successfully use it.

This is easy to do. First, change the type of object at the top of the `mygame.cpp` file:

```
#include "ProgressBar.h"
```

```
ProgressBar *progressBar;
```

And change the code that creates the ProgressBar object in the `Init` function to be:

```
progressBar = CreateObject<ProgressBar>("ProgressBar");
```

Compile again and press the space key. Your new function on the ProgressBar class is being called and you are affecting changes in your progress bar from your game code.

And that's it. You have all the tools you need now to make rich and intelligent game components or characters, with clean code that is easy to follow and modify.