ScrollObject to ScrollObject Communication

Your ScrollObject might want to know what is going on in the world around it. Perhaps it might need to know about a particular type of enemy, perhaps how close it is.

A good example of this is a Tower Defence game. Imagine you had a tower in the form of a ScrollObject. It wants to know about the closest robot enemy marching near it. Each enemy can be calculated to determine the one closest to the Tower. Then the tower can open fire on that object.

We'll work through this kind of scenario.

The routine will work with the following steps:

1. The Tower object will ask the Scroll class via a function to affect the nearest enemy.
2. The function in the Scroll class will enumerate all the Enemy objects available in the game.
3. The one that is closest will have it's colour changed.

Create a new orx/Scroll Project

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

Rename the Scroll class to `TowerDistance` and rename the scroll header and cpp files to: `tower-distance.h` and `tower-distance.cpp`. This will help remove and confusion when we create our ScrollObject classes.

Assets

Download the `tower.png` and `enemy.png` assets and save them to the `data/texture` folder.



Creating the Tower ScrollObject

First, create the config for the Tower object:

```
[Tower]
Graphic         = @
Texture         = tower.png
Pivot           = center
Position        = (0, 0, 0)
AngularVelocity = -2
```

Next, create the ScrollObject class and header for our Tower:

Tower.h

```
#include "tower-distance.h"

class Tower : public ScrollObject
{
public:

private:
    virtual void    OnCreate();
    virtual void    OnDelete();
    virtual void    Update(const orxCLOCK_INFO &_rstInfo);

};
```

Tower.cpp

```
#include "Tower.h"

void Tower::OnCreate()
{
}

void Tower::OnDelete()
{
}

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

Then add the binding for a Tower in the tower-distance Scroll class:

```
void TowerDistance::BindObjects()
{
    ScrollBindObject<Tower>("Tower");
}
```

Don't forget to add the prototype to the `tower-distance.h` header.

Next, include the file into the `TowerDistance` Scroll class:

```
#define __SCROLL_IMPL__
#include "tower-distance.h"
```

```
#include "Tower.h"
#undef __SCROLL_IMPL__
```

Then in the init() function, create an instance of the Tower:

```
orxSTATUS TowerDistance::Init()
{
    orxSTATUS result = orxSTATUS_SUCCESS;

    CreateObject("Tower");

    return result;
}
```

Compile and run. There should be a Tower sitting in the middle of the screen.

Great. Now for enemies.

The Enemy ScrollObject

We will creating a largely empty ScrollObject for the Enemy. The reason is because we will later need to enumerate over all Enemy objects using the `GetNextObject<>` function. But more on that later.

First, the config for the Enemy:

```
[Enemy]
Graphic         = @
Texture         = enemy.png
Pivot           = center
AngularVelocity = 18
FXList          = MovementFX
```

Next, the header and class:

Enemy.h

```
#include "tower-distance.h"

class Enemy : public ScrollObject
{
public:

private:
    virtual void    OnCreate();
    virtual void    OnDelete();
    virtual void    Update(const orxCLOCK_INFO &_rstInfo);
```

```
};
```

Enemy.cpp

```
#include "Enemy.h"

void Enemy::OnCreate()
{
}

void Enemy::OnDelete()
{
}

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

Next, include the file into the `TowerDistance` Scroll class:

```
#define __SCROLL_IMPL__
#include "tower-distance.h"
#include "Tower.h"
#include "Enemy.h"
#undef __SCROLL_IMPL__
```

Then add the binding to the `TowerDistance` Scroll class:

```
void TowerDistance::BindObjects()
{
    ScrollBindObject<Tower>("Tower");
    ScrollBindObject<Enemy>("Enemy");
}
```

The Tower wanting the closest Enemy

We'll be using the `Update` function of the `Tower` ScrollObject to test for the closest Enemy every frame:

```
void Tower::Update(const orxCLOCK_INFO &_rstInfo)
{
    orxVECTOR towerPosition = orxVECTOR_0;
    this->GetPosition(towerPosition, orxTRUE);

    TowerDistance::GetInstance().ColourClosestEnemy(towerPosition);
```

```
}
```

Notice the function that we will call: `ColourClosestEnemy`, lives in the Scroll class which is accessed by the singleton `GetInstance()`.

Of course this function does not yet exist. Add it to the `tower-distance.cpp` Scroll class:

```
void TowerDistance::ColourClosestEnemy(orxVECTOR towerPosition) {
    orxVECTOR red = { 1.0, 0, 0 };
    orxVECTOR grey = { 0.5, 0.5, 0.5 };

    orxCOLOR highlight;
    highlight.vRGB = red;
    highlight.fAlpha = 1.0;

    orxCOLOR neutral;
    neutral.vRGB = grey;
    neutral.fAlpha = 1.0;

    orxFLOAT closestDistance = 10000; //set to overly large amount
    Enemy *closestEnemy = orxNULL;

    for (Enemy *enemy = GetNextObject<Enemy>();
        enemy != orxNULL;
        enemy = GetNextObject<Enemy>(enemy)) {

        orxVECTOR enemyPosition = orxVECTOR_0;
        enemy->GetPosition(enemyPosition, orxTRUE);

        orxFLOAT distance = orxVector_GetDistance(&towerPosition,
&enemyPosition);
        if (distance < closestDistance) {
            closestDistance = distance;
            closestEnemy = enemy;
        }

        enemy->SetColor(neutral, orxFALSE);
    }

    if (closestEnemy != orxNULL) {
        closestEnemy->SetColor(highlight, orxFALSE);
    }
}
```

Also, add a `ColourClosestEnemy` protoype to the header.

This is a long function, so I'll just explain it simply that the Enemy objects are enumerated using:

```
GetNextObject<Enemy>()
```

This was the reason we needed to create a minimal ScrollObject class for the Enemy.

Finally, the distances are calculated. The nearest is coloured red, and all others are coloured grey.

Create Enemies via a spawner

We'll spawn 5 enemies and no more. First an `EnemyObjectCreator` and `EnemySpawner`:

```
[EnemyObjectCreator]
Spawner  = EnemySpawner
Position = (-400, 100, 0)

[EnemySpawner]
Object           = Enemy
WaveSize         = 1
WaveDelay        = 2.0
TotalObject      = 5
```

Next, create an instance of the `EnemyObjectCreator` in the init() function:

```
orxObject_CreateFromConfig("EnemyObjectCreator");
```

Make the Enemies march back and forth

At the moment, unless the enemies move, none of this will work. Give the enemies some movement FX:

```
[MovementFX]
SlotList      = @
Loop          = true
Type          = position
Curve         = sine
StartTime     = 0.0
EndTime       = 10.0
StartValue    = (0,0,0)
EndValue      = (800, 0, 0)
Absolute      = false

[Enemy]
Graphic         = @
Texture         = logo.png
Pivot           = center
Scale           = 0.25
AngularVelocity = 18
FXList          = MovementFX
```

Compile and run. Observe the closest enemies will always be red, all other being grey. That's it. The point was to demonstrate that a ScrollObject can call into the Scroll class to perform tasks for it, in this case, to find all the Enemy objects.

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

Permanent link:
https://orx-project.org/wiki/en/tutorials/orxscroll/scrollobject_to_scrollobject_communication

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