Semi-Dynamic Objects and Level Mapping

Orx has a very powerful config system that can allow you to define all objects, onscreen positions, lists for tiles, screens or anything else you can think up. In this way you can create level maps for your game very easily.

But it can be labourous, especially with lots of mapping data and not so simple to make tweaks here and there.

Sometimes it is nice to have your map defined as an array in your own code and to generate the needed objects onto the screen.

This tutorial aims to show you how to:

1) Define a config file for your tile graphics and object template.

2) Dynamically place objects on screen as dictated by an array of index values.

3) Clear and Repaint.

Getting a Tile Graphic

I have provided a set of tiles for this tutorial here.



They're rough but they'll do for illustration purposes. Each tile is 80 x 80 pixels on a 240 x 240 pixels sized graphic. Save to the data/scenery folder.

Define your Config File

Besides the standard Display, Viewport & Camera sections, you'll need to specify a default object as a template, a graphic, and then the individual tiles:

```
[TilesObject]
```

```
Graphic        = tile1; This will do as a default.

[TilesGraphic]
Texture     = ../../data/scenery/tutorial-tiles.png
TextureSize = (80, 80, 0)

[tile1@TilesGraphic]
TextureOrigin = (0, 0, 0)

[tile2@TilesGraphic]
TextureOrigin = (80, 0, 0)

[tile3@TilesGraphic]
TextureOrigin = (160, 0, 0)

[tile4@TilesGraphic]
TextureOrigin = (0, 80, 0)

[tile5@TilesGraphic]
TextureOrigin = (80, 80, 0)

[tile6@TilesGraphic]
TextureOrigin = (160, 80, 0)

[tile7@TilesGraphic]
TextureOrigin = (0, 160, 0)

[tile8@TilesGraphic]
TextureOrigin = (80, 160, 0)

[tile9@TilesGraphic]
TextureOrigin = (160, 160, 0)
```

Painting the Tiles

First job is to create a const to define how many tiles to paint across the screen:

```
const int TILES_ACROSS = 8;
```

The following variable is there to say what position in the array we start painting our tiles from.

Now to create a map of tiles:

```
int map[] = {1, 2, 3, 1, 1, 1, 4, 5, 6, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1, 9};
```

The integers in there represent the defined tiles in the config file, ie 7 is tile7, 2 is tile2. I'll show how this resolves to the tile name next...

We need a PaintTiles method:

```
bool PaintTiles(int tileIndexPosition){

    int mapLength = sizeof(map) / sizeof(map[0]);
        orxLOG("Length %d map[0] %d map %d", mapLength, sizeof(map[0]),
sizeof(map)    );
    if (tileIndexPosition >= mapLength)
        return false;

    for (int x=0; x<TILES_ACROSS; x++){
        int i = x + tileIndexPosition;

        if (i > mapLength-1){
            return false; //out of tiles, no more drawing
        }

        //Create String in format: Tile{Number}, for example "tile8".
        orxCHAR buffer[16] = {};
        orxString_NPrint(buffer, sizeof(buffer) - 1, "tile%d", map[i]);

        //Create Tile{Number} object from config file, utilizing string
created above.
        //Note our tiles are stored as "[tile8@TilesGraphic]" in our config
file.
        orxGRAPHIC *graphic;
        graphic = orxGraphic_CreateFromConfig(buffer);

        //Create Default Tile object, which holds our fullsize image.
        orxOBJECT *tile;
        tile = orxObject_CreateFromConfig("TilesObject");

        //Link our Tile{Number} config to our fullsize image.
        //The Tile{Number} config contains the portion of our fullsize
tilemap image that we want to display.
        orxObject_LinkStructure(tile, orxSTRUCTURE(graphic));

        //Position our tile on the screen.
        orxVECTOR tilePos;
        tilePos.fX = (80 * x);
        tilePos.fY = 160;
        tilePos.fZ = 0;
        orxObject_SetPosition(tile, &tilePos);

    }

    return true;
}
```

Fairly self explanatory, but the method takes an index which is start position in the array to paint from. Next, the size of the array is checked to make sure the passed index is not out of range.

An object is created each loop pass using [TilesObject] in the config as a template. This is preferred to orxObject_Create. Saves a lot of manual effort.

Next, a graphic object is created and the name passed is a combination of tile + 'map value'. This will give us the correct name pulled from the config, eg. tile4.

Finally, the graphic structure is linked to the object structure and then positioned.

And that's it for painting.

Clearing and Repainting

Now we need way of clearing the objects if we want to repaint the map in a different starting index position.

```
void ClearTiles(){
    orxSTRUCTURE *structure = orxStructure_GetFirst(orxSTRUCTURE_ID_OBJECT);
    while (structure != orxNULL){

        orxOBJECT *object = orxOBJECT(structure);
        const orxSTRING orxStr = orxObject_GetName(object);

        orxSTRUCTURE *nextStructure = orxStructure_GetNext(structure);
//temp holder
        if (orxString_Compare(orxStr, "TilesObject") == 0 ){ //only clear
"TilesObject" objects
            orxGRAPHIC *linkedGraphic = orxGRAPHIC(orxOBJECT_GET_STRUCTURE(
orxOBJECT(structure), GRAPHIC) ) ;

                        orxObject_UnlinkStructure(object,
orxSTRUCTURE_ID_GRAPHIC);
            orxGraphic_Delete(linkedGraphic);
            orxObject_Delete(object);
        }
        structure = nextStructure;
    }

}
```

This routine loops through any objects that are named “TilesObject”, get's the graphic that is linked to it, deletes the graphic and then the object itself.

Back in our main code we can demonstrate the painting of some tiles with:

```
PaintTiles(0);
```

To clear the tiles and repaint starting somewhere else in the array:

```
ClearTiles();
PaintTiles(2);
```

And you're done.

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

Permanent link:
https://www.orx-project.org/wiki/en/tutorials/mapping/semi-dynamic_objects_and_level_mapping

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