Setting up a game project on the Mac

This is the third article in a series for Orx on Mac OS X.

The first article helped the user set up their Mac for development by downloading the *Xcode Commandline Developer Tools* which gives the user the GCC compiler, the `git` command, and a MacOSX SDK.

The second article guided the user how to compile the Orx library.

This article will teach how to create your own game project using the Orx tools and Orx library.

Creating a new game project with "init"

In the root of the orx project folder, you will see a script called `init.sh`.

This tool let's you create a project for the Mac in all available IDEs and compilers.

1. Open a Terminal
2. cd into the `~/Documents/orx/` folder
3. Type: `./init.sh ~/Documents/MyGame`

You will see output like:

```
Misc junk - fix

[ 19:37:48 ] Initializing [ MyGame ] in [ ~/Documents/ ]
[ 19:37:48 ] == Creating files:
  + MyGame\.editorconfig
  + MyGame\build\premake4.lua
  + MyGame\data\config\CreationTemplate.ini
  + MyGame\data\config\SettingsTemplate.ini
  + MyGame\data\config\MyGame.ini
  + MyGame\data\config\MyGamed.ini
  + MyGame\data\config\MyGamep.ini
  + MyGame\data\sound\appear.ogg
  + MyGame\data\texture\logo.png
  + MyGame\src\MyGame.cpp
[ 19:37:48 ] Generating build files for [ windows ]:
  * gmake
Building configurations...
Running action 'gmake'...
Generating windows/gmake/Makefile...
Generating windows/gmake/MyGame.make...
Done.
  * codelite
Building configurations...
```

```
Running action 'codelite'...
Generating windows/codelite/MyGame.workspace...
Generating windows/codelite/MyGame.project...
```

You can see that Solution Projects have been created for gmake, Codelite, Code::Blocks and Xcode.

Use Finder to take a look at the structure of your game folders:

Your project can be compiled and developed in any of these IDE's (or a text editor in the case of gmake). We'll use gmake because if you have been following the series with a clean Mac, gcc/make is what we have.

Compiling our game project with gmake

1. cd into `~/Documents/MyGame/build/mac/gmake`
2. type: `make`

(the above is the same as typing the default of: `make config=debug32`)

You'll see your game compile with:

```
==== Building orxLIB (debug32) ====
Creating obj/x32/Debug/orxLIB
orxPlugin_EmbeddedList.cpp
orxAnim.c
orxAnimPointer.c
orxAnimSet.c
----- >8 snip ------
orxLinkList.c
orxString.c
orxTree.c
Linking orxLIB
ld: warning: directory not found for option '-L/usr/lib32'
Running post-build commands
cp -f ../../../lib/dynamic/liborx*.dylib ../../../bin
==== Building orx (debug32) ====
Creating obj/x32/Debug/orx
orxMain.c
Linking orx
ld: warning: directory not found for option '-L/usr/lib32'
==== Building Bounce (debug32) ====
Creating obj/x32/Debug/Bounce
orxBounce.c
Linking Bounce
```

```
ld: warning: directory not found for option '-L/usr/lib32'
```

You'll also notice the post-build step of:

```
Running post-build commands
cp -f ../../../lib/dynamic/liborx*.dylib ../../../bin
```

This step is used to copy the dylib(s) into the game's bin folder. How is the compiler able to do this if our game project is in a completely different place to the Orx library project? The answer is: the location stored in the $ORX environment variable.

The variable is taken from your `.bashrc` and `.profile` files. The variable holds the location of the Orx Includes and Orx libraries (dylib) to link to.

Running our game project

Now to run our compiled program.

- cd `~/Documents/MyGame/bin`
- Type: `./MyGamed`

You're all done! Congratulations on compiling the Orx library, and your very first Orx game project.

You might be interested to move onto the Beginner's Tutorial and create a platform game.