Using CocosBuilder on Cocos2D-X games

USING COCOSBUILDER ON COCOS2D-X GAMES

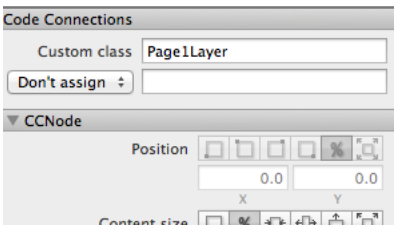
CocosBuilder is a great tool that will save you hundreds of hours of development. It is useful to create game UI's, but also to design animations and even to create gameplay scenes for some kind of games. It's a productivity tool that will speed up your development and will allow you to focus on important things. The best of all is that CocosBuilder is a free Open Source application.

This is NOT an introductory tutorial to CocosBuilder, I'm not going to explain how to create animations or screens, I'm just going to explain how to connect CocosBuilder with our Cocos2d-x games. This is something that we could learn in the TestCPP projects provided by the Cocos2d-x engine, but the code there is quite complex and too weird in my opinion, and it's confusing to many developers, so here I'll show a simple example on how to integrate a CocosBuilder scene in your Cocos2d-x game.

UPDATE: If you are a newbie with cocosbuilder, please read this article first.

Here it is our little sample, a small screen where we have several animated characters, and some interactive elements (a CCMenuItem with several CCMenuItemImage).

The first thing we need to do is to assign a class name to the main CCLayer of the cocosbuilder project. In our case it will be called "Page1Layer":



That Page1Layer custom class is expected to exist into our cocos2d-x project so we need to create it. It will handle all the menu item clicks, among other events. This is our declaration and implementation for the Page1Layer class:

```
#include "cocos2d.h"
#include "cocos-ext.h"

class Page1Layer :
        public cocos2d::CCLayer,
        public cocos2d::extension::CCBSelectorResolver
{
        virtual bool init();
        virtual cocos2d::SEL_MenuHandler onResolveCCBCCMenuItemSelector(CCObject * pTarget, cocos2d::CCStr
        virtual cocos2d::extension::SEL_CCControlHandler onResolveCCBCCControlSelector(cocos2d::CCObject *

        void appleClicked(CCObject *pSender, cocos2d::extension::CCControlEvent pCCControlEvent);

public:
        CREATE_FUNC(Page1Layer);
};

class Page1BuilderLoader : public cocos2d::extension::CCLayerLoader {
public:
        CCB_STATIC_NEW_AUTORELEASE_OBJECT_METHOD(Page1BuilderLoader, loader);

protected:
        CCB_VIRTUAL_NEW_AUTORELEASE_CREATECCNODE_METHOD(Page1Layer);
};

#include "Page1Layer.h"

using namespace cocos2d;
using namespace extension;

bool Page1Layer::init()
{
        if (!CCLayer::init())
        {
                return false;
        }

        return true;
}

SEL_MenuHandler Page1Layer::onResolveCCBCCMenuItemSelector(CCObject * pTarget, CCString * pSelectorName)
{
        CCB_SELECTORRESOLVER_CCMENUITEM_GLUE(this, "appleClicked", Page1Layer::appleClicked);

        return NULL;
}

void Page1Layer::appleClicked(CCObject *pSender, CCControlEvent pCCControlEvent) {
        CCLOG("button clicked");
}

SEL_CCControlHandler Page1Layer::onResolveCCBCCControlSelector(CCObject *pTarget, CCString*pSelectorName)

        return NULL;
}
```

Notice that we have created a CCLayerLoader class called Page1BuilderLoader. Wee need it in order to load the Page1Layer. Now we need to prepare a scene class, where we will load our CCB nodes, in our case, it will be called Page1Scene. This is it's declaration and implementation:

```
#include "cocos2d.h"
#include "CCBAnimationManager.h"

class Page1Scene :
        public cocos2d::CCScene,
        public cocos2d::extension::CCBAnimationManagerDelegate
{
        cocos2d::extension::CCBAnimationManager *animationManager;

        virtual bool init();
        virtual void completedAnimationSequenceNamed(const char *name);

        void setBlendFuncToAllChildren(CCNode* node, const cocos2d::ccBlendFunc& blend);

        ~Page1Scene();

public:
        CREATE_FUNC(Page1Scene);
};

#include "Page1Scene.h"
```

```
#include "cocos-ext.h"
#include "Page1Layer.h"

using namespace cocos2d;
using namespace extension;

bool Page1Scene::init()
{
        CCNodeLoaderLibrary * ccNodeLoaderLibrary = CCNodeLoaderLibrary::newDefaultCCNodeLoaderLibrary();
        ccNodeLoaderLibrary->registerCCNodeLoader("Page1Layer", Page1BuilderLoader::loader());

        cocos2d::extension::CCBReader * ccbReader = new cocos2d::extension::CCBReader(ccNodeLoaderLibrary)

        animationManager = NULL;

        CCNode* node = ccbReader->readNodeGraphFromFile("page1.ccbi", this, &animationManager);

        animationManager->setDelegate(this);
        animationManager->runAnimations("ArmPigForeground");

        ccbReader->release();
        ccBlendFunc normalBlend = (ccBlendFunc) {GL_ONE, GL_ONE_MINUS_SRC_ALPHA};

        if(node != NULL)
        {
                setBlendFuncToAllChildren(node, normalBlend);
                this->addChild(node);
        }

        return true;
}

void Page1Scene::setBlendFuncToAllChildren(CCNode* node, const ccBlendFunc& blend)
{
        CCArray* children = node->getChildren();

        CCObject* current = NULL;
        CCARRAY_FOREACH(children, current)
        {
                CCNode* child = dynamic_cast(current);

                if (child)
                {
                        CCSprite* sprite = dynamic_cast(current);

                        if (sprite && sprite->getTag() == 50)
                        {
                                sprite->setBlendFunc(blend);
                        }

                        setBlendFuncToAllChildren(child, blend);
                }
        }
}

void Page1Scene::completedAnimationSequenceNamed(const char *name) { }

Page1Scene::~Page1Scene() { }
```

If you spend some time investigating the code you will notice that is very simple, and will see something we haven't talked about, the CCBAnimationManager. This class is intended, as it name describes, to manage the animations of a CCB scene 😃 to run an animation we just need to set the delegate to the current class (the current class must inherit from CCBAnimationManagerDelegate), and call the runAnimations method.

That's it! Super simple code that allows us to do amazing thinks in few minutes with a very little amount of code!

Please, feel free to follow us on twitter: @jboschaiguade and @plungeint if you liked this article and want to keep updated about game development technical articles and more.

This blog is being maintained by Plunge Interactive professionals, a game development company.