Porting Cocos2d-x Games for Windows Phone 8

This article shows how to port existing Cocos2d-x games for Windows Phone 8 with an example game. Note that it uses the instructions in Creating a New Cocos2d-x Project for Windows Phone 8, which explain how to add Cocos2d-x to your Windows Phone 8 project and prepare a "bare bones template" suitable for creating a new game of copying in the resources of the game you want to port.

Note: This article was a winner in the Windows Phone 8 Wiki Competition 2012Q4.

Introduction

This article will demonstrate how to port existing Cocos2d-x games to Windows Phone 8 by porting the game Tweejump. Tweejump is a platform jumper game, where you have to tilt your phone to control the character. Currently the Cocos2d-x port for WP8 doesn't have support for accelerometer sensor, but we'll be adding that support to the game by ourselves.

TweeJump in Github

License: The code is released under the MIT License.

All images are copyrighted by Sergey Tikhonov. Please use them only for learning purposes, and don't release with your own project.

You can download the TweeJump project windows phone 8 project from https://github.com/Summeli/TweeJump4wp8

You can also take a look at the original Tweejump project page at https://code.google.com/p/tweejump-cocos2dx/

Preparation

First take a look at the Cocos2d-x test case status for Windows Phone and verify that the Cocos2d-x Windows Phone 8 port supports enough features for your application. It's not impossible to do the port, if some of the features of your application are missing. It basically means that you have to solve how to deal with the missing features by yourself. At the time of writing this article the test case status looks like this:

Test case name	Test case status
Actions Test	Pass
Transitions Test	Pass, but has a few DirectX warnings
ProgressActions Test	Pass
Effects Test	Pass
ClickAndMove Test	Pass
RotateWorld Test	Pass
Particle Test	Pass
EaseActions Test	Pass
MotionStreak Test	Pass
DrawPrimitives Test	Pass
CocosNode Test	Pass
Touches Test	Pass
Menu Test	Pass
ActionManager Test	Pass
Layer Test	Pass
Scene Test	Pass
Parallax Test	Pass
TileMap Test	Pass

(C) Copyright Nokia 2014. All rights reserved.

Interval Test	Pass
Chipmunk Test	NA
Label Test	Pass
TextInput Test	NA
Sprite Test	Pass
Scheduler Test	Pass
RenderTexture Test	The 1st test case is a bit wacky. rest of them are OK.
Texture2D Test	Some texture pixel formats are not supported: RGB5A1 (16bit), A8 (8bit), RGBA 4444 (16bit)
Box2d Test	Pass
Box2dTestBed	Pass
EffectAdvancedTest	Pass
HiRes Test	NA
Accelerometer Test	NA
Keypad Test	NA
CocosDenshion Test	Only support .wav format
Performance Test	Some texture formats are not supported
Zwoptex Test	Pass
Curl Test	Failed
UserDefault Test	Pass
Director Test	Pass
Font Test	Text alignment has not been implemented
CurrentLanguage Test	Pass
TextureCache Test	CCTextureCache::addImageAsync has not been implemented
Extensions Test	only CCNotificationCenter is available
Lua binding	NA
Javascript binding	NA
CocosBuilder Support	NA

OK, then we can start by creating a new Cocos2d-x project for Windows Phone. This is explained fully in the article: Creating a New Cocos2d-x Project for Windows Phone 8.

Differences between Windows Phone 8 and other cocos2d-x applications

The WindowsPhone port is using DirectX instead of OpenGL, but the port hides all this stuff from the user. So if the game you're porting does not use OpenGL shaders, you don't have to worry about differences between DirectX and OpenGL at all.

As usual, the Cocos2d-X application is started from **AppDelegate.cpp**. The `AppDelegate` is very similar to the other basic cocos2d-x applications. The Windows Phone 8 port hides all of the complex DirectX initializations etc. behind the `initInstance()` function and `applicationDidFinishLaunching` functions. In the `InitInstance` the `appDelegate` actually creates the DirectX surfaces etc. You have to take a closer look of the `CCEGLView` class if you want to know more about the DirectX surface initialization for the Cocos2d-X port. This may come handy if you're going to create shaders for your application.

```
bool AppDelegate::initInstance()
{
   bool bRet = false;
   do
   {
      #if (CC_TARGET_PLATFORM == CC_PLATFORM_WIN8_METRO)
      // fix bug: 16bit aligned
      void* buff=_aligned_malloc(sizeof(CCEGLView),16);
      CCEGLView* mainView = new (buff) CCEGLView();
      mainView->Create();
      mainView->setDesignResolution(320, 533);
      #endif // CC_PLATFORM_WIN8_METRO

      bRet = true;
     } while (0);
    return bRet;
}
```

The Gaming scene is created as usual in the applicationFinishLaunching function. Do not mind about the setOpenGLView etc. That's all DirectX stuff in the implementation (see CCEGLView implementation). The Cocos2d-X port just keeps the same function names to keep better compatibility to the old Cocos2d-X applications.

(C) Copyright Nokia 2014. All rights reserved.

```
bool AppDelegate::applicationDidFinishLaunching()
{
   // initialize director
   CCDirector *pDirector = CCDirector::sharedDirector();

   pDirector->setOpenGLView(&CCEGLView::sharedOpenGLView());

   // turn on display FPS
   //pDirector->setDisplayFPS(false);

   pDirector->setDeviceOrientation(CCDeviceOrientationPortrait);
   // set FPS. the default value is 1.0/60 if you don't call this
   //pDirector->setAnimationInterval(1.0 / 60);

   // create a scene. it's an autorelease object
   CCScene *pScene = CCScene::node();
   pScene->addChild(GameScene::node());

   // run
   pDirector->runWithScene(pScene);

    return true;
}
```

The `MainScene` class illustrates the differences between the Windows Phone 8 and iOS platforms. Take a look at the **MainScene.h**, and you see comments on top of the functions.

```
   // Here's a difference. Method 'init' in cocos2d-x returns bool, instead of returning
'id' in cocos2d-iphone
   virtual bool init();

   // there's no 'id' in cpp, so we recommand to return the exactly class pointer
   static cocos2d::CCScene* scene();

   // implement the "static node()" method manually
   LAYER_NODE_FUNC(MainScene);
```

The GameScrene inherits `MainScrene` which implements the basic initialization functionality for the game.

```
CCScene* MainScene::scene()
{
   CCScene * scene = NULL;
   do
   {
      // Create our Scene
      scene = CCScene::node();
      CC_BREAK_IF(! scene);

       // Create and Add the MainScene Layer
       MainScene *layer = MainScene::node();
       CC_BREAK_IF(! layer);
       scene->addChild(layer);

       // Create and Add the GameScene Layer
       GameScene *layer1 = GameScene::node();
       CC_BREAK_IF(! layer1);
       scene->addChild(layer1);
   } while (0);
   return scene;
}
```

The main screne create game screne

```
// on "init" you need to initialize your instance
bool MainScene::init()
{
   bool bRet = false;
   do
   {
```


(C) Copyright Nokia 2014. All rights reserved.

```
        ////////////////////
        // super init first
        ////////////////////
        CC_BREAK_IF(! CCLayer::init());

        //Init all the sprites etc.

        // Enable the touch events
       setIsTouchEnabled(true);

       bRet = true;
    } while (0);
    return bRet;
```

Adding assets to the Windows Phone 8 project

Basically you should add the assets in to your project with following procedure:

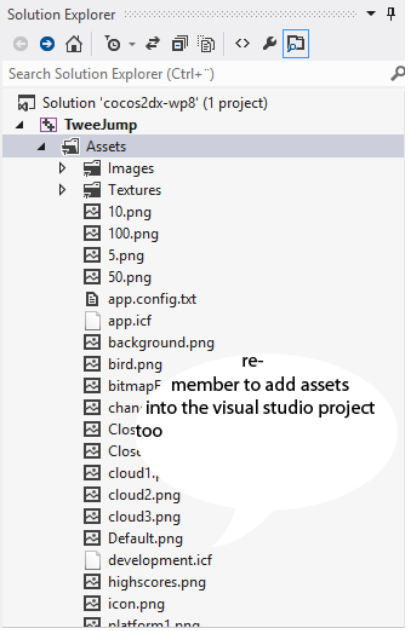
1. Copy the assets into your WP8 project under the Assets directory.
2. Right click the assets filter in Visual Studio, and choose "Add Existing Item", and add the new asset into you project.

However there are few special cases related to the assets. If you're using sub-directories in your code like this:

```
CCSprite *bird =
CCSprite::spriteWithSpriteFrame(CCSpriteFrameCache::sharedSpriteFrameCache()->spriteFrameByName("Images/bird.png"));
```

Then you should add them into your project by:

1. Select Assets filter
2. Right click -> Add New Filter, named it Image
3. Select Image filter
4. Right click -> Add Existing Item...
5. Choose the $YOUR_PROJECT/Assets/Image/bird.png

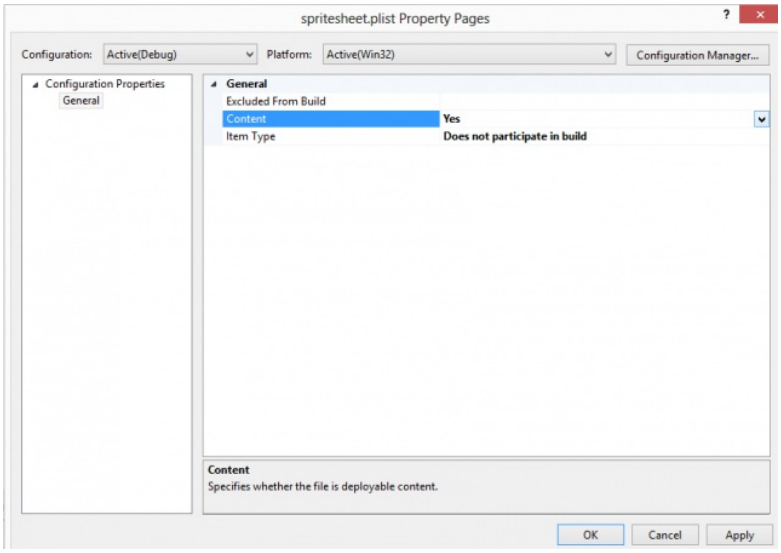


The Visual Studio will add all of your image files as assets, but you're using anything else as a resource, then you must remember to add it as "project content". For example, if you're using resource like this:

```
CCSpriteFrameCache::sharedSpriteFrameCache()->addSpriteFramesWithFile("Images/spritesheet.plist");
```

Add the resource to the Assets as usual, but after that you have to right click the asset, select properties and set the "Content" as "Yes". Select also "Do not participate in the build"

(C) Copyright Nokia 2014. All rights reserved.



Adding the code

Copy your classes into the Classes directory, and the add the source code to the build:

1. Select Classes filter
2. Right click -> Add Existing item
3. Choose all of you source code files from $YOUR_PROJECT/Classes/

After this you should add this include

```
#include "pch.h"
```

At the beginning of all of your .cpp files.

Introduction for integrating WinRT components into your native c++ app

The Windows Phone 8 APIs are used C++ side though the Windows runtime. The Windows runtime object must be declared with public sealed C++/CX class. We're going to implement our own winRT-class to listen the accelerator sensor, so here's a short primer how to use create and use managed winRT object from standard c++ side, and how to make callbacks from winRT objects.

- The new C++ class is declared with ref-keyword

```
ref class myclass
{
    //...
};
```

- New winRT classes can be created from standard c++ side with a handle-to-object operator ^ is known as a "hat" and is fundamentally a C++ smart pointer.

```
myclass ^ myClass = ref new myclass ();
```

- The sealed C++/CX class can not have any public methods with standard c++ pointers, so you can't set callbacks from sealed winRT side to c++ side like this

```
//this will generate compiler error "signature of public member contains native type"
void setListener(myPtr* listener);
```

- One way of calling pure c++ code from winRT class is to create static c++ methods as wrappers.

```
class myCppClass {
public:
    static void myCallback(double myval);
}

//let's call the native c++ from sealed winRT object myclass
void myclass::doTheCallback()
{
myCppClass::myCallback(newVal);
}
```


(C) Copyright Nokia 2014. All rights reserved.

Adding Support for Accelerator Sensor

As we saw on the test table, the Cocos2d-x wp8 port doesn't contain support for acceleration sensor, therefore we have to do it by ourselves. Windows Phone 8 provides accelerometer support in winRT level, so we'll have to create a winRT class to receive the events from the accelerator sensor, and then pass those events to our native c++ class.

Here's a simple `acceleratorDelegate` class, which will call static function `GameScene::accelerated(double x, double y, double z );`

```
AcceleratorDelegate::AcceleratorDelegate(void)
{
    m_accelerometer = Accelerometer::GetDefault();
    if (m_accelerometer != nullptr)
    {
 // Select a report interval that is both suitable for the purposes of the app and
supported by the sensor.
        // This value will be used later to activate the sensor.
 uint32 minReportInterval = m_accelerometer->MinimumReportInterval;
 m_desiredReportInterval = minReportInterval > 16 ? minReportInterval : 16;
 m_readingToken = m_accelerometer->ReadingChanged::add(ref new
TypedEventHandler<Accelerometer^, AccelerometerReadingChangedEventArgs^>(this,
&AcceleratorDelegate::ReadingChanged));
     }
}

void AcceleratorDelegate::ReadingChanged(Windows::Devices::Sensors::Accelerometer^
sender, Windows::Devices::Sensors::AccelerometerReadingChangedEventArgs^ e)
{
 AccelerometerReading^ reading = e->Reading;
 GameScene::accelerated(reading->AccelerationX, reading->AccelerationY,
reading->AccelerationZ );
}
```

Let's create the `AcceleratorDelegate` object in the `GameScene`'s init function. I also made a global pointer to the `GameScene`, so we could easily call it's `didAccelerate()` (cocos2d-x callback) function from a static callback function.

```
GameScene* g_scene;

// Initialize the GameScene
bool GameScene::init()
{
  //the normal init, and
   g_scene = this;
   m_accel = ref new AcceleratorDelegate();
}
```

Implementation for the static function accelerated, which will be receiving the acceleration events from the managed winRT side.

```
void GameScene::accelerated(double x, double y, double z)
{
   CCAcceleration acceleration;
   acceleration.x = x;
   acceleration.y = y;
   acceleration.z = z;
   //call the cocos2d-x didAccelerate function, and handle the event in normal way
   g_scene->didAccelerate(&acceleration);
}
```

That's it, now we have tweejump running with accelerator support.

Reference links

1. Cocos2d-x for Windows Phone 8 is out: http://www.cocos2d-x.org/news/76
2. A trap of adding resources into cocos2dx-wp8 project: http://www.cocos2d-x.org/news/78
3. Ref classes and structs (C++/CX) http://msdn.microsoft.com/en-us/library/windows/apps/hh699870(v=vs.110).aspx
4. Visual C++ and WinRT/Metro - Some fundamentals http://www.codeproject.com/Articles/262151/Visual-Cplusplus-and-WinRT-Metro-Some-fundamentals

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.