Developing a Windows 10.0 UWP Game with Cocos2d-x

Quick Start Challenge OSCON Portland 2015

# Overview

[Cocos2d-x](http://www.cocos2d-x.org/) is a popular open source C++ mobile gaming engine adopted by 400,000+ developers to build games such as Castle Clash or other top mobile games. In this hands-on lab, you will learn how to add resources to your game, use those resources in the game, and manage object interactions.

# What You’ll Learn

The objectives of this lab are to learn how to:

* Create a new cocos2d-x game.
* Add resources to a cocos2d-x game
* Define objects in the game.
* Manage object interactions.

# Tools You’ll Use

Visual Studio 2015 running on Windows 10 preview.

# The Challenge

* [Create a stub project](#_Creating_your_first)
* [Add assets to project](#_Task_2:_Change)
* [Add a background image](#_Task_2:_Add)
* [Add a sprite of the game](#_Task_3:_Add)
* [Enable touch control in the game](#_Task_4:_Enable)
* [Create Projectiles and enable actions for the hero](#_Task_5:_Create)
* [Create targets](#_Task_6:_Create)
* [Manage object interactions](#_Task_8:_Add_1)

## Task 1: Create a stub project

In this task, you will create a new cocos2d-x project that you will modify to create a game.

1. On the Start screen (you can get to the Start screen by pressing the Windows Button on the keyboard or the button in the bottom left corner of the desktop), type Command Prompt and then click **Command Prompt** in the search results list.
2. At the command prompt window, enter the following command:

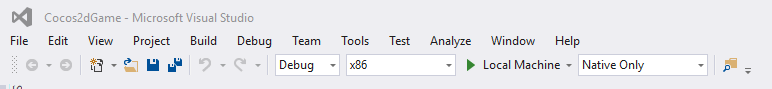
c:\labs\Cocos2dGame-v3.7\new\_cocos2d\_game.bat

Note: A new cocos2d-x game is usually created by using the cocos tool with the command:

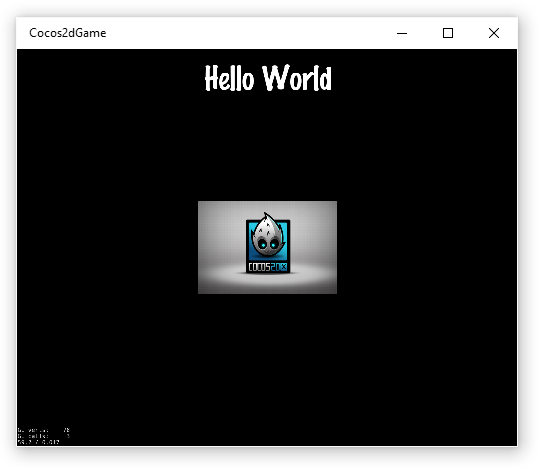
cocos new Cocos2dGame -p com.mycompany.cocos2dgame -l cpp -d projects

However, for this quick start challenge we have simplified the process to speed up building cocos2d-x. This will allow you to spend more time coding and less time building.

1. The new\_cocos2d\_game.bat script will create a new Windows 10.0 Universal Windows Platform (UWP) solution for you and will open a window containing the project files.
2. In the file explorer, double click on **Cocos2dGame.sln** to open the solution in Visual Studio.
3. Select Debug and x86 from the Configuration and Platform menus. (Visual Studio 2015 may default to Debug/ARM when it first loads your project).



1. Press **CTRL+SHIFT+B** to start a build. The build may take a few minutes so feel free to review the next steps in this doc while the project builds.
2. Press **F5** to start the app. (Select **Stop Debugging** from the Visual Studio Debug menu to stop the game.) Your app should look like the following image.



## Task 2: Add assets to project

In this task you will add image assets to your game. Game asset files are added to the projects\Cocos2dGame\Resources folder. This will allow the same assets to be used across all projects (Windows 10.0 UWP, Windows 8.1 Universal App, iOS, Android, etc.).

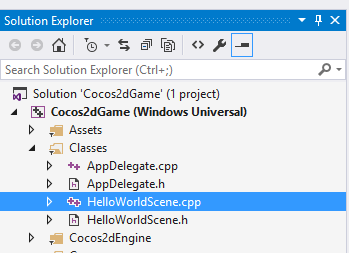
Note: Whenever you add new assets to a project, you should rebuild the Cocos2dGame.WindowsPhone and Cocos2dGame.Windows projects. Right click on each project and select the Project Only/Rebuild Only option. This will add the new assets to the build. Alternatively can also close and re-open the Cocos2dGame.sln project to make sure the new assets are added to the Cocos2dGame projects. Since the assets needed for our game were already placed in the Resources folder for you, there is no need to reload the solution or rebuild the projects.

**Note:** Most cocos2d-x games are designed to crash if they cannot find an asset. If your game crashes, make sure the asset is being loaded.

## Task 3: Add a background image

In this task, you are will the name of the game and add the background.png image that represents the field of play. We will modifying the HelloWorldScene.cpp file.

1. Expand the **Cocos2dGame** solution folder and then expand the **Classes** folder.
2. Open the **HelloWorldScene.cpp** file which can be found under the Classes directory in the Cocos2dGame.Shared solution. The shared solution will contain all of the source files for your game.



In the HelloWorldScene.cpp file, in the *HelloWorld::Init(*) method modify and add the lines highlighted below:

/////////////////////////////

// 3. add your codes below...

// add a label shows "My Game"

// create and initialize a label

auto label = Label::createWithTTF("My Game", "fonts/Marker Felt.ttf", 24);

// position the label on the center of the screen

label->setPosition(Vec2(origin.x + visibleSize.width/2,

origin.y + visibleSize.height - label->getContentSize().height));

// add the label as a child to this layer

this->addChild(label, 1);

// add background image

auto sprite = Sprite::create("background.png");

// position the sprite on the center of the screen

sprite->setPosition(Vec2(visibleSize.width/2 + origin.x, visibleSize.height/2 + origin.y));

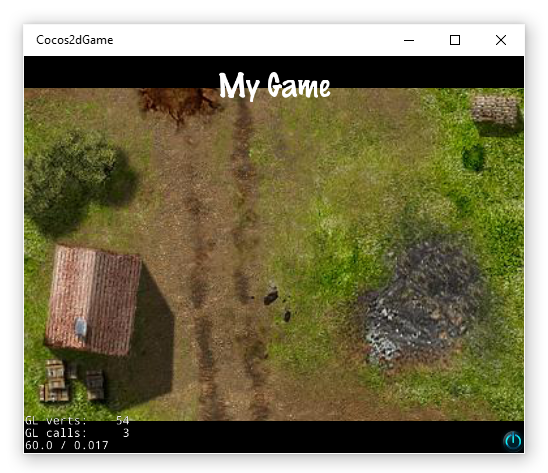
sprite->setScale(visibleSize.width / sprite->getContentSize().width);

// add the sprite as a child to this layer

this->addChild(sprite, 0);

return true;

Press **F5** to run your game. It should now look like this:



Select Stop Debugging from the Debug menu to stop the game.

## Task 3: Add a sprite to the game

In this task, you will add code to add a sprite to the game.

In the HelloWorldScene.cpp file, in the *HelloWorld::Init()* method, add the highlighted lines as show below:

// add the sprite as a child to this layer

this->addChild(sprite, 0);

// add the hero of the game

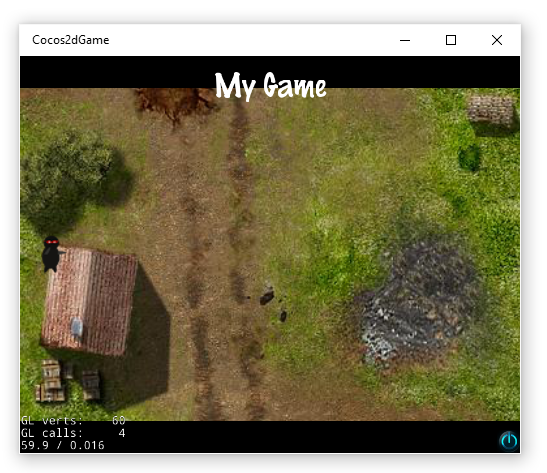
auto player = Sprite::create("Player.png");

player->setPosition(Vec2(origin.x + player->getContentSize().width,

origin.y + visibleSize.height / 2));

this->addChild(player);

You now have all of the static graphics in place. To test the game at this point, press **F5** to run the app. It should look like this:



Select Stop Debugging from the Debug menu to stop the game.

## Task 4: Enable touch control in the game

In this task, you will add touch control to the app. You will also add code to register the application with the touch dispatcher of the gaming engine, and a callback to handle touch/mouse events.

1. In the HelloWorldScene.h file, at the end of the of the public declaration, add the highlighted lines below:

class HelloWorld : public cocos2d::Layer

{

public:

// there's no 'id' in cpp, so we recommend returning the class instance pointer

static cocos2d::Scene\* createScene();

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

virtual bool init();

// a selector callback

void menuCloseCallback(cocos2d::Ref\* pSender);

// implement the "static create()" method manually

CREATE\_FUNC(HelloWorld);

private:

// touch input

bool onTouchBegan(cocos2d::Touch\* touches, cocos2d::Event \*event);

void onTouchMoved(cocos2d::Touch\* touches, cocos2d::Event \*event);

void onTouchEnded(cocos2d::Touch\* touches, cocos2d::Event\* event);

};

1. In the HelloWorldScene.cpp file, at the end of the file, add the corresponding method definitions:

bool HelloWorld::onTouchBegan(Touch\* touch, Event \*event)

{

auto location = touch->getLocation();

CCLog("onTouchBegan x:%f, y:%f", location.x, location.y);

return true;

}

void HelloWorld::onTouchMoved(Touch\* touch, Event \*event)

{

auto location = touch->getLocation();

CCLog("onTouchMoved x:%f, y:%f", location.x, location.y);

}

void HelloWorld::onTouchEnded(Touch\* touch, Event \*event)

{

auto location = touch->getLocation();

CCLog("onTouchEnded x:%f, y:%f", location.x, location.y);

}

1. To register for touch events in the *HelloWorld::Init()* method, add the highlighted lines just before the return statement:

// Register Touch Events

auto listener = EventListenerTouchOneByOne::create();

listener->setSwallowTouches(true);

listener->onTouchBegan = CC\_CALLBACK\_2(HelloWorld::onTouchBegan, this);

listener->onTouchMoved = CC\_CALLBACK\_2(HelloWorld::onTouchMoved, this);

listener->onTouchEnded = CC\_CALLBACK\_2(HelloWorld::onTouchEnded, this);

\_eventDispatcher->addEventListenerWithSceneGraphPriority(listener, this);

this->setTouchEnabled(true);

Now the app is touch enabled. To test it, press **F5** to run the app, then touch or click on the field. You should see the touch coordinates in the Debug “Output” window. Select Stop Debugging from the Debug menu to stop the game.

*Note: If you do not see the Debug Output menu, go to the “View” menu and select “Output”.*

## Task 5: Create Projectiles and enable actions for the hero

In this task, you will add code that enables the hero to shoot projectiles.

1. You will need to add some member variables to the *HelloWorld* class to hold a collection of active projectiles. Add the following highlighted lines to the the HelloWorldScene.h file.

private:

// touch input

bool onTouchBegan(cocos2d::Touch\* touches, cocos2d::Event \*event);

void onTouchMoved(cocos2d::Touch\* touches, cocos2d::Event \*event);

void onTouchEnded(cocos2d::Touch\* touches, cocos2d::Event\* event);

// a vector to hold the active projectiles

cocos2d::Vector<cocos2d::Sprite\*> \_projectiles;

};

1. Now add the code to fire a projectile when the user touches the screen. The projectile is created when there is a touch began event. The code will initiate the movement of the projectile based on the position of touch event relative to the hero location. You will add the following code to HelloWorldScene.cpp file in the HelloWorld::onTouchBegan() method.

bool HelloWorld::onTouchBegan(Touch\* touch, Event \*event)

{

auto location = touch->getLocation();

CCLog("onTouchBegan x:%f, y:%f", location.x, location.y);

// Set up initial location of projectile

Size winSize = Director::getInstance()->getVisibleSize();

Vec2 origin = Director::getInstance()->getVisibleOrigin();

Sprite \*projectile = Sprite::create("Projectile.png", Rect(0, 0, 40, 40));

projectile->setPosition(Vec2(origin.x + 40, origin.y + winSize.height / 2));

// Determinie offset of location to projectile

float offX = location.x - projectile->getPosition().x;

float offY = location.y - projectile->getPosition().y;

// Bail out if we are shooting backwards

if (offX <= 0)

{

return true;

}

// Ok to add now - we've double checked position

this->addChild(projectile);

// Determine where we wish to shoot the projectile to

float realX = origin.x + winSize.width + (projectile->getContentSize().width / 2);

float ratio = offY / offX;

float realY = (realX \* ratio) + projectile->getPosition().y;

Point realDest(realX, realY);

// Determine the length of how far we're shooting

float offRealX = realX - projectile->getPosition().x;

float offRealY = realY - projectile->getPosition().y;

float length = sqrtf((offRealX \* offRealX) + (offRealY\*offRealY));

float velocity = 480 / 1; // 480pixels/1sec

float realMoveDuration = length / velocity;

// Move projectile to actual endpoint

projectile->runAction(Sequence::create(

MoveTo::create(realMoveDuration, realDest),

CallFuncN::create(this,

callfuncN\_selector(HelloWorld::spriteMoveFinished)),

NULL));

// Add to projectiles array

projectile->setTag(2);

\_projectiles.pushBack(projectile);

return true;

}

1. You will also need to add some code to handle when we are finished with a projectile. Add the following code to the HelloWorldScene.cpp file

void HelloWorld::spriteMoveFinished(Node\* sender)

{

Sprite \*sprite = (Sprite \*) sender;

this->removeChild(sprite, true);

switch (sprite->getTag())

{

case 2: // projectile

\_projectiles.eraseObject(sprite);

break;

default:

break;

}

}

1. Add the following highlighted lines to the HelloWorldScene.h file:

private:

void onTouchEnded(cocos2d::Touch\* touches, cocos2d::Event\* event);

// removes a sprite from the game

void spriteMoveFinished(cocos2d::Node\* sender);

You now have a hero who shoots projectiles. Press **F5** to run the game and click on the screen to shoot a projectile. Select Stop Debugging from the Debug menu to stop the game.

## Task 6: Create targets

In this task, you will add code to add targets to shoot at. For the game to be fun, you need the targets to be added dynamically by the game engine. To do so, you will use the “schedule” feature of cocos2D-X. You will add a task that the game engine will schedule regularly.

1. In the HelloWorldScene.h file, add the following highlighted lines to add a collection of targets and a method to create them.

private:

// removes a sprite from the game

void spriteMoveFinished(cocos2d::Node\* sender);

// adds an enemy target to the game

void addTarget();

// a vector to hold the active projectiles

cocos2d::Vector<cocos2d::Sprite\*> \_projectiles;

// a vector to hold the active enemy targets

cocos2d::Vector<cocos2d::Sprite\*> \_targets;

1. In the HelloWorldScene.cpp file, add the *addTarget()* method to the class:

void HelloWorld::addTarget()

{

Size winSize = Director::getInstance()->getVisibleSize();

Vec2 origin = Director::getInstance()->getVisibleOrigin();

Sprite \*target = Sprite::create("Target.png", Rect(0, 0, 54, 80));

// Determine where to spawn the target along the Y axis

float minY = target->getContentSize().height/2;

float maxY = winSize.height - target->getContentSize().height/2;

int rangeY = (int) (maxY - minY);

// srand( TimGetTicks() );

int actualY = (rand() % rangeY) + (int) minY;

// Create the target slightly off-screen along the right edge,

// and along a random position along the Y axis as calculated

target->setPosition(Vec2(winSize.width + (target->getContentSize().width/2),

origin.y + actualY));

this->addChild(target);

// Determine speed of the target

int minDuration = (int)2.0;

int maxDuration = (int)4.0;

int rangeDuration = maxDuration - minDuration;

int actualDuration = (rand() % rangeDuration) + minDuration;

// Create the actions

FiniteTimeAction\* actionMove = MoveTo::create((float) actualDuration,

Point(0 - target->getContentSize().width/2, actualY));

FiniteTimeAction\* actionMoveDone = CallFuncN::create(this, callfuncN\_selector(HelloWorld::spriteMoveFinished));

target->runAction(Sequence::create(actionMove, actionMoveDone, NULL));

// Add to targets array

target->setTag(1);

\_targets.pushBack(target);

}

1. If you were to run your game now, no targets would be created. We need to schedule the adding of targets using the cocos2d-x scheduler. Add the following method to the end of HelloWorldScene.cpp:

void HelloWorld::gameLogic(float dt)

{

this->addTarget();

}

1. Add the following highlighted lines to HelloWorldScene.h

// adds an enemy target to the game

void addTarget();

// called by the scheduler to update the game logic

void gameLogic(float dt);

1. Now schedule the gameLogic task to add targets by adding the highlighted line to the end of the *HelloWorld::Init()* method.

this->schedule(schedule\_selector(HelloWorld::gameLogic), 1.0);

return true;

}

1. Press **F5** to run your game. Now targets are flying all around you… but you still can’t shoot them. You need to implement collision detection between your projectiles and the targets. Select Stop Debugging from the Debug menu to stop the game.

## 

## Task 7: Manage object interactions

In this task, you will add code to manage the collisions between targets and projectiles.

1. In the HelloWorldScene.h file, add the highlighted lines below:

// called by the scheduler to update the game logic

void gameLogic(float dt);

// update the game by applying collision detection

void updateGame(float dt);

1. In the HelloWorldScene.cpp file, in the *HelloWorld::Init()* method, just before the return statement add the following highlighted line:

this->schedule(schedule\_selector(HelloWorld::gameLogic), 1.0);

this->schedule(schedule\_selector(HelloWorld::updateGame));

return true;

}

1. In the HelloWorldScene.cpp file, at the end of the file, add the *updateGame()* method to manage target and projectile interaction:

void HelloWorld::updateGame(float dt)

{

cocos2d::Vector<cocos2d::Sprite\*> projectilesToDelete;

for (Sprite \*projectile : \_projectiles)

{

Rect projectileRect(

projectile->getPosition().x - (projectile->getContentSize().width/2),

projectile->getPosition().y - (projectile->getContentSize().height/2),

projectile->getContentSize().width,

projectile->getContentSize().height);

cocos2d::Vector<cocos2d::Sprite\*> targetsToDelete;

for (Sprite \*target : \_targets)

{

Rect targetRect(

target->getPosition().x - (target->getContentSize().width/2),

target->getPosition().y - (target->getContentSize().height/2),

target->getContentSize().width,

target->getContentSize().height);

if (projectileRect.intersectsRect(targetRect))

{

targetsToDelete.pushBack(target);

}

}

for (Sprite \*target : targetsToDelete)

{

\_targets.eraseObject(target);

this->removeChild(target, true);

}

if (targetsToDelete.size() > 0)

{

projectilesToDelete.pushBack(projectile);

}

targetsToDelete.clear();

}

for (Sprite \*projectile : projectilesToDelete)

{

\_projectiles.eraseObject(projectile);

this->removeChild(projectile, true);

}

projectilesToDelete.clear();

}

1. In the *spriteMoveFinished()* method, add lines the highlighted below to remove targets sprites.

void HelloWorld::spriteMoveFinished(Node\* sender)

{

Sprite \*sprite = (Sprite \*) sender;

this->removeChild(sprite, true);

switch (sprite->getTag())

{

case 1: // target

\_targets.eraseObject(sprite);

break;

case 2: // projectile

\_projectiles.eraseObject(sprite);

break;

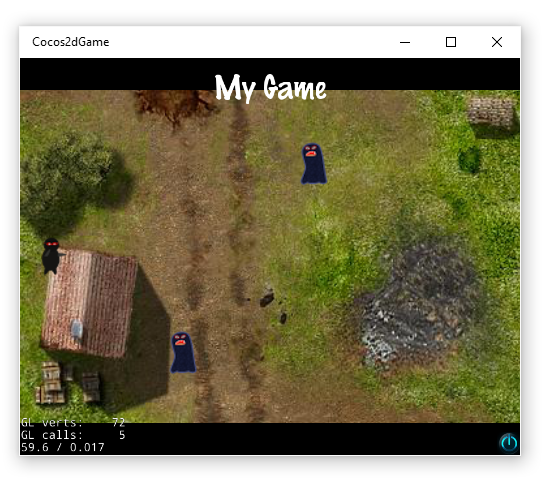
default:

break;

}

}

You now have a basic, playable cocos2d-x game. To test the finished game, press **F5**. Your finished game should look similar to this. Your completed game should look similar to the following image:



**Summary**

In this hands-on lab, you learned the basics of creating a cocos2D-X project, how to interact with the screen and objects in the game.

To get started creating your own Windows 10.0 UWP game go to: https://github.com/cocos2d/cocos2d-x