Cocos2D-X

Luis Fernando de Mingo López

2019

Contents

1 Cocos2d-x

1.1 Install and run

1.2 Basic concepts

2 Scenes

3 Events

4 Game Loop

Sobre algún elemento de la clase `Node`, normalmente una escena `Scene`.

```
this->scheduleUpdate();
```

Donde el parámetro `delta` representa el tiempo transcurrido desde la última llamada (debería ser 1/60). Normalmente se multiplica la acción por `delta` para conseguir el mismo efecto en dispositivos con menor capacidad de proceso.

```
void HelloWorld::update(float delta)
{
    ...
}
```

5 Accelerometer

Dispositivo real

```
Device::setAccelerometerEnabled(true);
auto accListener = EventListenerAcceleration::create(CC_CALLBACK_2(MainScene::accelerate
getEventDispatcher()->addEventListenerWithSceneGraphPriority(accListener, this);

void accelerated(Acceleration *acceleration, Event *event);
```

O con expresiones Lambda

```
Device::setAccelerometerEnabled(true);
auto listener = EventListenerAcceleration::create([=](Acceleration* acc, Event* event){
    auto gravity = Vec2(acc->x*100.0f, acc->y*100.0f);
    world->setGravity(gravity);
});
this->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener...
```

6 Jerarquía de `listeners` a través del grafo de la escena

```
auto listener1 = EventListenerTouchOneByOne::create();
listener1->setSwallowTouches(false);
listener1->onTouchBegan = [=](Touch* touch, Event* event) {
    if (event->getCurrentTarget()->getBoundingBox().containsPoint(touch->getLocation()))
        sel = (Sprite *)event->getCurrentTarget();
        return true;
    }
    return false;
};

listener1->onTouchMoved = [=](Touch* touch, Event* event){
    return true;
};

listener1->onTouchEnded = [=](Touch* touch, Event* event){
    if (event->getCurrentTarget()->getBoundingBox().containsPoint(touch->getLocation()))
        sel->setTexture("cross.png");
        return true;
    }
    return false;
};

for (int i=0; i<3; i++)
    for(int j=0; j<3; j++)
        getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener1->clone(),
```

---

7 Infinite Parallax

7.1 Archivo `InfiniteParallaxNode.h`

El método `updatePosition()` se encarga de ajustar los offset de los *hijos* contenidos en el `parallax` y si han sobrepasado el extremo izquierdo de la pantalla los ajustará al extremo derecho.

```
#ifndef InfiniteParallax_hpp
#define InfiniteParallax_hpp
USING_NS_CC;

class InfiniteParallaxNode : public ParallaxNode
{
public:
    static InfiniteParallaxNode* create();
```

```
    void updatePosition();
};
#endif
```

7.2 Archivo `InfiniteParallaxNode.cpp` (parte 1/3)

```
#include "InfiniteParallax.h"
USING_NS_CC;

class PointObject : public Ref
{
public:
    inline void setRation(Point ratio) {_ratio = ratio;}
    inline void setOffset(Point offset) {_offset = offset;}
    inline void setChild(Node *var) {_child = var;}
    inline Point getOffset() const {return _offset;}
    inline Node* getChild() const {return _child;}
private:
    Point _ratio;
    Point _offset;
    Node* _child;
};
```

7.3 Archivo `InfiniteParallaxNode.cpp` (parte 2/3)

```
InfiniteParallaxNode* InfiniteParallaxNode::create()
{
    InfiniteParallaxNode* node = new InfiniteParallaxNode();
    if(node) node->autorelease();
    else {
        delete node;
        node = 0;
    }
    return node;
}
```

7.4 Archivo `InfiniteParallaxNode.cpp` (parte 3/3)

```
void InfiniteParallaxNode::updatePosition()
{
    int safeOffset = -10;
    Size visibleSize = Director::getInstance()->getVisibleSize();
    for(int i = 0; i < _children.size(); i++)
    {
        auto node = _children.at(i);
        if(convertToWorldSpace(node->getPosition()).x + node->getContentSize().width < s
            for(int i = 0; i < _parallaxArray->num; i++)
            {
                auto po = (PointObject*)_parallaxArray->arr[i];
                if(po->getChild() == node)
                    po->setOffset(po->getOffset() +
                                  Point(visibleSize.width + node->getContentSize().width
            }
    }
}
```

7.5 Uso del Parallax Infinito

A través del `scheduleUpdate()` (bucle principal del juego controlado por `void update(float delta)`) hay que controlar el movimiento del `parallax`y efectuar una llamada al método `updatePosition()`, para ajustar los `offset` de los nodos contenidos.

```
void MenuScene::update(float delta) {
    par->setPosition(Vec2(par->getPosition().x-1, par->getPosition().y));
    par->updatePosition();
}
```