### 2.1 实现交互功能

本节学习通过手势滑动、重力感应、鼠标和键盘来实现交互功能的方法

**【本节知识点】**

* **Cocos2d-x中的事件处理机制**
* **Cocos2d-x的单击与多点触摸的使用方法**
* **Cocos2d-x中如何兼容键盘和鼠标**
* **Cocos2d-x中重力感应功能的使用**
* **Cocos2d-x中自定义实现摇杆功能（选学）**

**（1）屏幕单击实现交互**

任何一款游戏都少不了对用户操作的处理，比如获取用户点击屏幕的坐标位置，或者判断用户是否点击了某个角色等。

本节通过一个单点触摸的纸牌游戏，学习通过屏幕单击进行交互功能的实现。实现后的效果如下图所示。下图为拖动第3,4张纸牌后的效果。



下面我们学习上面程序的实现方法：

bool HelloWorld::init()

{

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

// 1. super init first

if ( !Layer::init() )

{

return false;

}

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

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

// 加入背景

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

background->setPosition(visibleSize.width/2,visibleSize.height/2);

this->addChild(background);

// 加入卡牌精灵

for (int i = 0; i < 5; i++)

{

char imageName[15] = { 0 };

sprintf(imageName, "mycard0%d.png", i);

myCard[i] = Sprite::create(imageName);

myCard[i]->setScale(0.26f);

myCard[i]->setPosition(visibleSize.width\*(i+1)/6, visibleSize.height/2);

this->addChild(myCard[i]);

}

**下面代码中创建了一个EventListenerTouchOneByOne类型的对象myListener，通过它来接收用户对屏幕做出的操作。**

**EventListenerTouchOneByOne用来对单点触摸进行处理的事件监听类。在该类中封装了方法：**

**OnTouchBegan**

**返回值：当用户单击在对应物体的范围内，返回true，否则返回false。**

**OnTouchMoved**

**onTouchEnded**

**onTounchCancelled**

**下面代码是对OnTouchBegan方法的使用**

**【说明】**

**在Cocos2d 3.x版本中，提供的触发器类型包括下面5种，其中EventListenerTouchOneByOne是触摸事件的一个子类。**

**EventListenerTouch（触摸事件）**

**EventListenerkeyboard（键盘响应事件）**

**EventListenerMouse（鼠标响应事件）**

**EventLisernerAcceleration（加速记录事件）**

**EventListenerCustom（自定义事件）**

**代码解释**

* **OnTouchBegan**

**auto target = static\_cast<Sprite\*>(event->getCurrentTarget());**

**获取事件所绑定的target，即用户的滑动、拖曳行为是针对哪个“物体”进行的。**

**Point locationInNode = target->convertToNodeSpace(touch->getLocation());**

**获取用户所单击的点的坐标。用户单击的坐标通过OnTouchBegan方法的touch参数传递。**

**if (rect.containsPoint(locationInNode))**

**{**

**}**

**判断touch所传递的坐标是否包含于被单击的“物体”中，以决定是否进行if条件中的操作。**

**OnTouchBegan返回值：当用户单击在对应物体的范围内，返回true，否则返回false。如果返回值为false，则不会执行下面的onTouchMoved操作。只有的OnTouchBegan返回true时，onTouchMoved方法才能发挥作用。**

* **OnTouchMoved：作用当用户在“物体”上拖动时，根据用户手势位置的变化移动物体**
* **OnTouchEnded：对触摸事件结束的响应。**
* **下面代码第2行中setSwallowTouches方法的功能：设置事件侦听器是否允许触摸事件向下传递。如果当两张纸牌有重叠在一起的部分时，用户单击它们重叠的部分进行拖动，会发生什么。如果参数为true，选择相对上层的纸牌进行操作而忽略底层的纸牌。如果参数为false，底层的卡牌也跟着上层的卡牌一起被拖动。**
* **addEventListenerWithSceneGraphPriority将事件侦听器与“物体”绑定，以判断对哪个“物体”的单击进行响应。**

**【提示】在多次对不同对象进行绑定时，要对触发器对象使用clone方法，以保证程序的正常运行。此代码中，\_ eventDispatcher是Node对象中自带的属性，用于统一管理各个节点的分发情况。**

// 建立事件监听器

auto myListener = EventListenerTouchOneByOne::create();

myListener->setSwallowTouches(true);

myListener->onTouchBegan = [](Touch\* touch, Event\* event)

{

// 获取事件所绑定的 target

auto target = static\_cast<Sprite\*>(event->getCurrentTarget());

// 获取当前点击点所在相对按钮的位置坐标

Point locationInNode = target->convertToNodeSpace(touch->getLocation());

Size s = target->getContentSize();

Rect rect = Rect(0, 0, s.width, s.height);

// 点击范围判断检测

if (rect.containsPoint(locationInNode))

{

// 显示当前卡牌精灵的坐标位置

log("sprite began... x = %f, y = %f", locationInNode.x, locationInNode.y);

target->setOpacity(180);

return true;

}

return false;

};

myListener->onTouchMoved = [](Touch\* touch, Event\* event)

{

auto target = static\_cast<Sprite\*>(event->getCurrentTarget());

// 显示当前卡牌精灵的坐标位置

log("sprite move... x = %f, y = %f", touch->getLocation().x, touch->getLocation().y);

// 移动当前卡牌精灵的坐标位置

target->setPosition(target->getPosition() + touch->getDelta());

};

myListener->onTouchEnded = [](Touch\* touch, Event\* event)

{

auto target = static\_cast<Sprite\*>(event->getCurrentTarget());

target->setOpacity(255);

};

// 为事件监听器绑定事件

\_eventDispatcher->addEventListenerWithSceneGraphPriority(myListener, myCard[0]);

\_eventDispatcher->addEventListenerWithSceneGraphPriority(myListener->clone(), myCard[1]);

\_eventDispatcher->addEventListenerWithSceneGraphPriority(myListener->clone(), myCard[2]);

\_eventDispatcher->addEventListenerWithSceneGraphPriority(myListener->clone(), myCard[3]);

\_eventDispatcher->addEventListenerWithSceneGraphPriority(myListener->clone(), myCard[4]);

return true;

}

在卡牌游戏中，可以使用这样的方法来实现对卡牌的拖曳、出牌等操作。以及人物行走的控制。

**思考题：上面的程序如何修改，能够实现仅有一张卡牌能够被拖动。**

答案：

修改下面代码

**if (rect.containsPoint(locationInNode))**

**{**

**}**

为：

**if (rect.containsPoint(locationInNode) && target == myCard[0])**

**{**

**}**

【注意】上面代码修改后，为了能够在onTouchBegan方法中引用外部的对象，需要将

myListener->onTouchBegan = [](Touch\* touch, Event\* event)

修改后为：

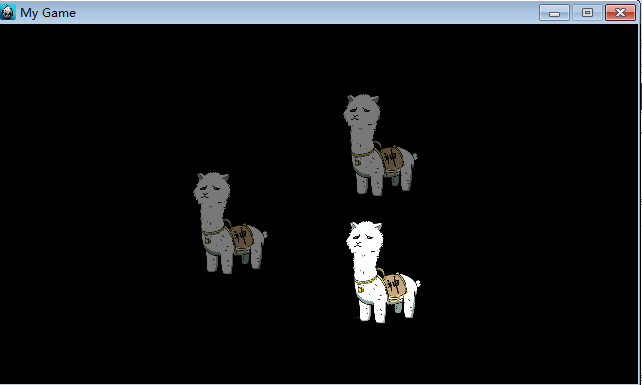
myListener->onTouchBegan = [=](Touch\* touch, Event\* event)

代码行中的方括号中加入一个等号，或者直接引用要引用的对象名。否则会提示如下错误信息：封闭函数局部变量不能再lambda体中引用，除非其位于捕获列表中。

**（2）多点触摸操作的实现**

当用户同时将两个或两个以上的手指伸到屏幕上时，游戏对多个触摸点同时响应，就叫做多点触摸。另外，在一些游戏中，还会使用触控来进行一下（放大、缩小等）的操作。

应用程序实现效果如下图。该程序无法在PC端用鼠标模拟手指单击效果。需要在手机上进行测试。在手机上安装运行后，屏幕上黑漆漆一片，用手指单击屏幕将会有“羊驼”图案出现在屏幕上，滑动手指羊驼将跟随手指移动，同时或依次加入多个手指将会呈现出多点触摸的效果。



**代码解释：**

**EventListenerTouchAllAtOne类型的监听器。使用方法与单点触摸类似。EventListenerTouchAllAtOne类中封装的方法有：**

**onTouchesBegin**

**onTouchesMoved**

**onTouchesEnded**

**for (auto &item : touches)实现对括号中内容数组的遍历。由于多点触摸功能中系统需要同时判断多个触摸事件，因此需要对触摸事件进行编号，以一个数组的形式传递和使用。for循环中的代码用来在界面中添加一个精灵对象。**

**在onTouchesMoved方法中，使用getID方法来获取所对应的编号，从而知道当前使用的是哪一次的触摸，然后根据这个编号来获取当前场景中的子节点，从而得到对应羊驼图案进行坐标位置的改变。**

**在onTouchesEnded方法中，找到对应羊驼图案，将其透明度恢复为不透明。**

代码如下：

// on "init" you need to initialize your instance

bool HelloWorld::init()

{

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

// 1. super init first

if ( !Layer::init() )

{

return false;

}

auto myListener = EventListenerTouchAllAtOnce::create();

myListener->onTouchesBegan = [=](const std::vector<Touch\*>& touches, Event \*unused\_event)

{

for (auto &item : touches)

{

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

sprite->setPosition(item->getLocation());

sprite->setOpacity(120);

sprite->setScale(0.2f);

this->addChild(sprite);

}

};

myListener->onTouchesMoved = [=](const std::vector<Touch\*>& touches, Event \*unused\_event)

{

for (auto &item : touches)

{

int id = item->getID();

auto array = this->getChildren();

auto sprite = array.at(id);

auto point = item->getLocation();

sprite->setPosition(point);

CCLOG("id=%d\tx=%f\ty=%f",id,point.x,point.y);

}

};

myListener->onTouchesEnded = [=](const std::vector<Touch\*>& touches, Event \*unused\_event)

{

for (auto &item : touches)

{

int id = item->getID();

auto array = this->getChildren();

auto sprite = array.at(id);

sprite->setOpacity(255);

auto point = item->getLocation();

sprite->setPosition(point);

CCLOG("id=%d\tx=%f\ty=%f", id, point.x, point.y);

}

};

\_eventDispatcher->addEventListenerWithSceneGraphPriority(myListener, this);

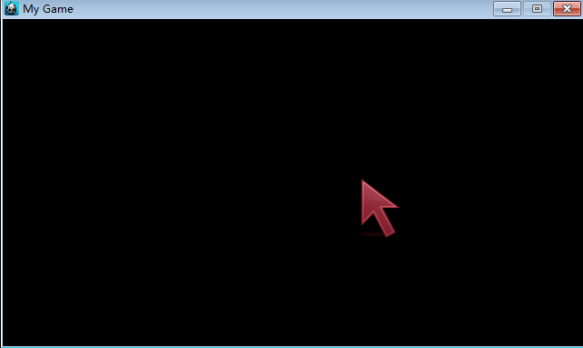
return true;

}

**（3）鼠标和键盘**

由于现在移动设备性能越来越强悍，已经基本开始支持OTG功能，可以通过转接线直接给平板或手机插上键盘鼠标来进行辅助操作，依次在移动游戏开发中考虑鼠标和键盘的使用也是必要的。

下面的示例实现对鼠标和键盘的支持。程序运行结果如下图。在屏幕上“箭头”图案跟随着鼠标指针的移动而移动，而当单机鼠标按键时，屏幕中的“箭头”会随着用户的操作缩小和恢复原状。当按下键盘上某个按键时，在VS的输出面板上会弹出按键信息。



**代码解释：**

**本节代码使用的EventListenerKeyboard类和EventListenerMouse类，用来监听键盘和鼠标的事件。**

**【说明】触摸事件与鼠标事件的区别。触摸事件只有当onTouchMoved事件被触发时才会得到响应，而鼠标事件则可以随时对鼠标指针的坐标做出响应。**

**鼠标事件中，在事件的参数中并没有直接传递当前鼠标指针的坐标信息，因此需要通过转换来获取。**

**this->mouse->setPosition(e->getCursorX(), e->getCursorY() + visibleSize.height);**

**代码如下：**

// on "init" you need to initialize your instance

bool HelloWorld::init()

{

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

// 1. super init first

if ( !Layer::init() )

{

return false;

}

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

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

this->mouse = Sprite::create("mouse.png");

this->mouse->setScale(0.3f);

this->addChild(mouse);

auto myKeyListener = EventListenerKeyboard::create();

myKeyListener->onKeyPressed = [](EventKeyboard::KeyCode keycode, cocos2d::Event \*event)

{

CCLOG("key is pressed,keycode is %d",keycode);

};

myKeyListener->onKeyReleased = [](EventKeyboard::KeyCode keycode, cocos2d::Event \*event)

{

CCLOG("key is released,keycode is %d", keycode);

};

auto myMouseListener = EventListenerMouse::create();

myMouseListener->onMouseMove = [=](Event \*event)

{

EventMouse\* e = (EventMouse\*)event;

this->mouse->setPosition(e->getCursorX(), e->getCursorY() + visibleSize.height);

};

myMouseListener->onMouseDown = [=](Event \*event)

{

this->mouse->setScale(0.2f);

};

myMouseListener->onMouseUp = [=](Event \*event)

{

this->mouse->setScale(0.3f);

};

\_eventDispatcher->addEventListenerWithSceneGraphPriority(myKeyListener, this);

\_eventDispatcher->addEventListenerWithSceneGraphPriority(myMouseListener, this);

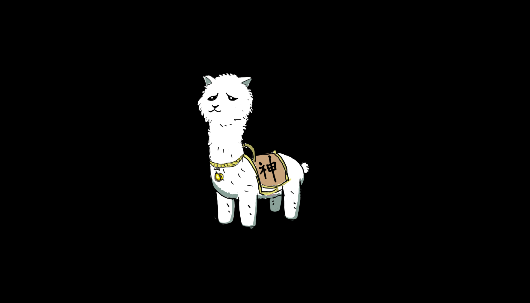
return true;

}

**（3）加速度传感器的使用—重力感应**

是一种在赛车游戏中被广泛使用的操作方式，具体原理是：通过加速度传感器分别获取来自x、y、z三个方向的加速度。因为当手机静止时，一定会受到重力，因此就可以确定合力所在的方向就是下方。而当手机被移动时，也可以根据手机静止时的状态以及各个方向加速度的变化，确认各个方向。

下面的示例展示利用加速度传感器获取操作信息的方法。运行结果如下图所示。程序运行后，会发现屏幕中的羊驼图案会随着手机方向的改变，不断改变运动方向以及尺寸大小。



代码解释：

**首先在init方法中对当前的场景进行一些简单设置。然后新建一个onAcceleration方法，处理加速度传感器获取的数据。**

* **init方法实现代码解释：**

**Device::setAccelerometerEnabled(true);**

**打开重力感应功能**

**auto listener = EventListenerAcceleration::create(CC\_CALLBACK\_2(HelloWorld::onAcceleration, this));**

**创建用于对加速度传感器进行响应的触发器Listerner，让它在被触发时调用onAcceleration方法。**

**\_eventDispatcher->addEventListenerWithSceneGraphPriority(listener, this);**

**将触发器与当前场景绑定**

* **onAcceleration方法实现代码解释**

**onAcceleration将设备x、y、z方向的3个加速度存放在参数acc中，然后分别判断它在3个方向上应该进行各种运动，将这些变化保存在参数scale和point中。最后实现这些变化。**

程序代码：

* **init方法实现**

// on "init" you need to initialize your instance

bool HelloWorld::init()

{

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

// 1. super init first

if ( !Layer::init() )

{

return false;

}

Size visibleSize = Director::getInstance()->getVisibleSize(); //获取场景尺寸

Vec2 origin = Director::getInstance()->getVisibleOrigin(); //获取坐标原点

screenWidth = visibleSize.width;

screenHeight = visibleSize.height;

scale = 1.0f;

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

sprite->setScale(0.3f);

sprite->setPosition(screenWidth / 2, screenHeight / 2);

this->addChild(sprite);

Device::setAccelerometerEnabled(true);

auto listener = EventListenerAcceleration::create(CC\_CALLBACK\_2(HelloWorld::onAcceleration, this));

\_eventDispatcher->addEventListenerWithSceneGraphPriority(listener, this);

return true;

}

* **onAcceleration方法实现**

void HelloWorld::onAcceleration(Acceleration\* acc, Event\* event)

{

auto point = this->sprite->getPosition();

if (acc->x < 0 && point.x>0)

{

point.x--;

}

else if (acc->x > 0 && point.x<this->screenWidth)

{

point.x++;

}

if (acc->y < 0 && point.x>0)

{

point.y--;

}

else if (acc->y > 0 && point.y<this->screenHeight)

{

point.y++;

}

if (acc->z<0 && this->scale>0.1)

{

this->scale = this->scale\*0.9;

}

else if (acc->z>0 && this->scale < 10)

{

this->scale = this->scale\*1.1;

}

this->sprite->setPosition(point);

this->sprite->setScale(0.3\*this->scale);

}

**（4）实现游戏摇杆（选学）**

自定义创建虚拟遥感的示例。程序执行结果如下图所示。当用户滑动摇杆是，摇杆将会随着用户的操作进行响应。



代码中需要创建HRocker类。HRocker.h文件内容如下：

#ifndef \_\_HROCKER\_H\_\_

#define \_\_HROCKER\_H\_\_

#include "cocos2d.h"

using namespace cocos2d;

//枚举型：用于标识摇杆与摇杆的背景

typedef enum{

tag\_rocker,

tag\_rockerBG,

}tagForHRocker;

class HRocker : public Layer

{

public:

//创建摇杆，摇杆图片，摇杆背景图片，起始坐标

static HRocker\* createHRocker(const char\* rockerImageName, const char\* rockerBGImageName, Point position);

//启动摇杆

void startRocker(bool \_isStopOther);

//停止摇杆

void stopRocker();

private:

EventListenerTouchOneByOne\* listener;

void rockerInit(const char\* rockerImageName, const char\* rockerBGImageName, Point position);

//得到半径为r的圆周运动上一个角度应对应的x,y坐标

Point getAnglePosition(float r, float angle);

//是否可操作摇杆

bool isCanMove;

//得到摇杆与用户触屏点的角度

float getRad(Point pos1, Point pos2);

//摇杆背景的坐标

Point rockerBGPosition;

//摇杆背景的半径

float rockerBGR;

//触屏事件

virtual bool onTouchBegan(Touch\* touch, Event\* event);

virtual void onTouchMoved(Touch\* touch, Event\* event);

virtual void onTouchEnded(Touch\* touch, Event\* event);

CREATE\_FUNC(HRocker);

};

#endif

**（5）实验2-1单点触摸**

**（6）实验2-2 重力感应**

**（7）实验5-1 实现模拟游戏摇杆**