### 3.1 地图的使用

在大多数非卡牌游戏中，游戏的角色都会在一定的场景中活动，比如说RGP游戏中的角色要在地图中行走，跑酷类游戏中角色的前进要沿着某些特定的范围行走，这些都需要用到地图系统。地图在实际使用中又分为瓦片地图和图片地图两类。

【本节知识点】

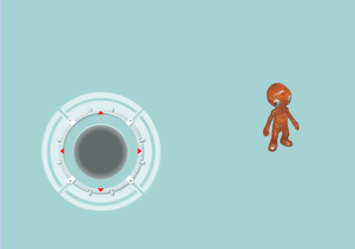
* 什么是图片地图和瓦片地图，各自的特点是什么
* 瓦片地图的制作
* 如何使用瓦片地图
* 怎样在地图中添加对象
* 地图中的路径是怎样被程序获取的
* 求路径的方法和原理

**（1）行走效果的实现**

前面实现了通过动画实现人物在场景中行走的效果，但是实现的效果只能在原地行走。在学习了定时器的实现原理后，可以通过定时器以一定的时间间隔对人物角色进行更新，当用户单击屏幕时，如果状态相同（原本就在行走切方向不变），则保持原有动作，而仅改变其前行的方向；如果状态不同，则对动画及方向都进行改变。

人物行走的效果，如何实现？本示例采用摇杆方式实现指挥人物行走的效果。

执行效果如下图所示：

****

**从类的定义开始：**

**人物**

**人物类：hero.h**

|  |  |  |
| --- | --- | --- |
| **序号** | **方法名** | **主要功能** |
| **1** | static Hero\* createHeroSprite(Point position, int direction, const char\* name); | **创建一个人物精灵，并指定位置、方向、图片名称** |
| **2** | void heroInit(Point position, int direction, const char\* name); | **初始化一个人物精灵，**createHeroSprite中调用 |
| **3** | Animate\* createAnimate(int direction, const char \*action, int num); | **创建动画。**  **num为动作帧数** |
| **4** | void setAction(int direction, const char \*action, int num); | **首先移除原有动作再加入新的动作** |
| **5** | void moveTo(float x,float y); | **人物位置移动** |

#ifndef \_\_HERO\_SCENE\_H\_\_

#define \_\_HERO\_SCENE\_H\_\_

#include "cocos2d.h"

USING\_NS\_CC;

class Hero : public Layer

{

public:

bool isRun;

int direction;

Point position;

Sprite\* sprite;

static Hero\* createHeroSprite(Point position, int direction, const char\* name);

void heroInit(Point position, int direction, const char\* name);

virtual bool init();

Animate\* createAnimate(int direction, const char \*action, int num);

void setAction(int direction, const char \*action, int num);

void moveTo(float x,float y);

CREATE\_FUNC(Hero);

};

#endif

**各方法的实现详见hero.cpp。**

**采用什么方法让人物开始行走?**

**摇杆类（HRocker.h）**

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

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

#include "cocos2d.h"

using namespace cocos2d;

enum hero\_direction

{

RIGHT\_DOWN = 1,

LEFT\_DOWN = 2,

LEFT\_UP = 3,

RIGHT\_UP = 4,

DOWN = 5,

LEFT = 6,

UP = 7,

RIGHT = 8

};

#define PI 3.1415926

/ /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:

float dx;

float dy;

int direction;

//是否可操作摇杆

bool isCanMove;

//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);

//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

**（2）为行走效果添加背景图片**

添加背景的代码：

需在场景类中添加background属性，类型为Sprite。并添加如下代码：

background = Sprite::create("background.jpg");

background->setPosition(320, 180);

addChild(background);

行走的方式可以通过两种方式实现：

方式1： 人移动

hero->moveTo(rocker->dx, rocker->dy);

方式2： 背景图片移动

float r = sqrt((rocker->dx\*rocker->dx)+(rocker->dy\*rocker->dy));

float x = rocker->dx / r;

float y = rocker->dy / r;

Point position = background->getPosition();

background->setPosition(position.x - x, position.y - y);

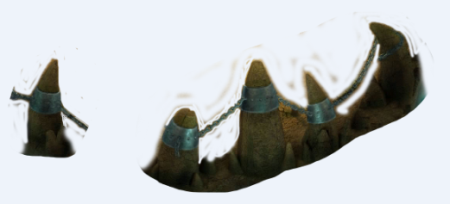
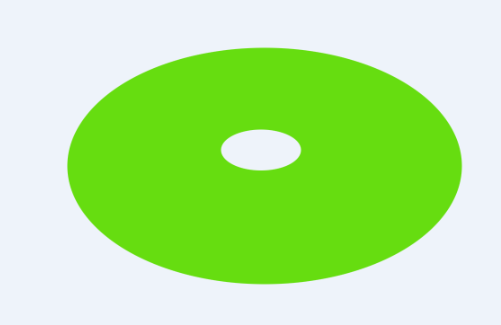
此时的问题是：人很容易就可以走出地图的边界，如何解决这个问题。

**（3）使用分层的方式来表示路径**

由于游戏中地图形状的不确定性，想要通过算法来解决此问题无疑是非常困难的，即使可以也会为开发人员带来巨大的负担。

下面我们学习一种比较简单的方法：

首先将地图分层处理。如下图所示，分为3层。其中中间图片表示的是游戏中人物可以行走的范围。右侧的图用来存放一些地图中遮罩人物的部分。这样人物就不会走在它上面了。后面两个图都使用了透明的效果，为图层增加了一定的透明度。



需要将两个图片文件加入到场景中区。

image = new Image();

image->initWithImageFile("background1.png");

background[0] = Sprite::create("background2.jpg");

//background[0]->setPosition(0, 0);

background[0]->setPosition(-600, -550);

background[0]->setAnchorPoint(Vec2(0, 0));

addChild(background[0]);

background[1] = Sprite::create("background3.png"); //遮罩层

background[1]->setOpacity(150);

//background[1]->setPosition(0, 0);

background[1]->setPosition(-600, -550);

background[1]->setAnchorPoint(Vec2(0, 0));

addChild(background[1],10);

hero = Hero::createHeroSprite(Vec2(320, 180), 1, "stand");

addChild(hero);

rocker = HRocker::createHRocker("rocker.png", "rockerBG.png", ccp(100, 100));

addChild(rocker,11);

rocker->startRocker(true);

定时器中的执行代码：

void HelloWorld::update(float dt)

{

if (rocker->isCanMove == true)

{

if (hero->isRun == false)

{

hero->isRun = true;

hero->direction = rocker->direction;

hero->setAction(hero->direction, "run", 8);

}

else if (hero->isRun == true)

{

if (hero->direction != rocker->direction)

{

hero->direction = rocker->direction;

hero->setAction(hero->direction, "run", 8);

}

}

//getColor(pix,piy);

float r = sqrt((rocker->dx\*rocker->dx) + (rocker->dy\*rocker->dy));

float x = (int)(rocker->dx / r\*100)/100.0f;

float y = (int)(rocker->dy / r\*100)/100.0f;

Point position = background[1]->getPosition();

**//下面2行代码。由于最初将背景地图的锚点设置在其左下角，当其位置为（0,0）时，实际人物应该在坐标（320,180）处，因为屏幕的尺寸为640\*360，而人物默认在场景的中央。当人物移动，比如向右移动了5个像素，那么实际上是地图向左移动了5个像素，因此时相当于人物所在的地图坐标改变了5个像素。其他方向也是类似的。**

auto pix = 320 - (int)background[1]->getPositionX();

auto piy = 180 - (int)background[1]->getPositionY();

CCLOG("%d,%d", pix, piy);

**//下面代码选择了3，而不是1的理由是，如果选择1，实际上会存在走到不可存在的区域无法返回的可能，而实际使用时，只要这个数字大于1，对游戏的影响就不大，一般在2~5之间。**

if (x > 0)

{

pix = pix - 3;

}

else

{

pix = pix + 3;

}

if (y > 0)

{

piy += 3;

}

else

{

piy -= 3;

}

if (getColor(pix, piy).a == 255)

{

background[0]->setPosition(position.x - x, position.y - y);

background[1]->setPosition(position.x - x, position.y - y);

}

}

else if (hero->isRun == true && rocker->isCanMove == false)

{

hero->isRun = false;

hero->setAction(hero->direction, "stand", 7);

}

}

上面的代码执行时，如果再让人物移动到“遮罩层”图片的位置，可以看到石头遮住了人物。这时，还有一张用于显示人物行走范围的图片没有看到，那是因为它被地图片遮挡住了。

行走范围图片的作用是，在人物行走时，通过读取对应坐标处该图片的颜色，判断行走到的位置是否“合法”，如果是“合法”的才会发生位置的移动，如果是“非法”的，则维持原地不动。

**【注意】程序代码的问题，如果行走方向变化时，人物也不可走动了。应提前判断。如何实现？可修改pix = pix + 5;中的数值。通常在2~5之间。**

**下面方法的功能是获取图片中某一像素点的颜色**

Color4B HelloWorld::getColor(int x,int y)

{

ccColor4B c = { 0, 0, 0, 0 };

//cocos2dx使用的是openGL坐标系，y坐标是从下到上递增的。

//libpng读取png图片时是从上到下逐行读取的，因此这里的纵坐标需要处理下。

y = image->getHeight() - y;

int width = image->getWidth();

unsigned char \*data\_ = image->getData();

unsigned int \*pixel = (unsigned int \*)data\_;//以unsigned int来处理

pixel = pixel + (y \* width) + x;//跳到指定的像素点。

c.a = (\*pixel >> 24) & 0xff;

float scale = (c.a + 1.0f) / 256;

c.r = \*pixel & 0xff;

c.r = c.r / scale;

c.g = (\*pixel >> 8) & 0xff;

c.g = c.g / scale;

c.b = (\*pixel >> 16) & 0xff;

c.b = c.b / scale;

CCLOG("%d,%d==%d,%d,%d,%d", x, y, c.r, c.g, c.b, c.a);

return c;

}

**【补充】由于本例仅选用了一个非常简单的地图，因此还有某些现象没有考虑到。比如在游戏中会出现一面墙或者是一棵树之类的障碍物，当人物的y坐标小于它时，人物可以对它产生遮挡效果。而当人物的y坐标大于它时则会被它遮住，如人物站立在墙外，将遮住墙。**

**可以用不同的颜色来表示当人物行走在该区域时遮罩图层所在的层次，这就能比较方便地实现人物与遮罩层次关系的更新了。**

**由于手游和PC游戏在操作方式上存在一定的差异，并且PC硬件与移动设备本身在架构和运算方式及性能上的差异导致这种方式并不适合手机游戏中使用。**

**因为使用这种方法所需要的资源实在太多了，3张背景地图所使用的图片文件一共才900多K，但这与所使用的地图实在太小了是有很大关系的。还有，游戏中不可能只有一个场景地图，个人感觉起码的10个地图场景吧，于是所需要的资源又大了不少。除此之外，一直将图片存放在内存中，对手机硬件也是一个不小的负担。基于这些理由，一般在手机游戏中都会采用另外一种方案，那就是瓦片地图。**

**（4）使用瓦片地图**

瓦片地图就是用很小的单位方格通过组合可以形成大的地图，这是在游戏设计中常用的一种技术。使用它的好处，最明显的就是能够大大的节约内存并且可提高系统渲染的效率。

* 获得瓦片地图

使用Tiled Map Editor获得（制作）瓦片地图

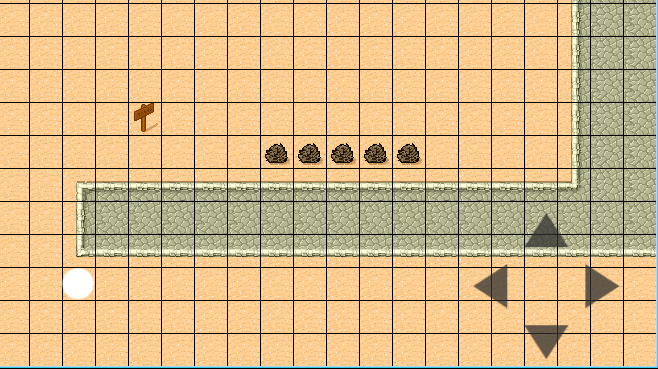
* 在Cocos2d-x中使用瓦片地图

加载瓦片地图

map = TMXTiledMap::create("map.tmx");

addChild(map);

使用瓦片地图



size = Director::getInstance()->getVisibleSize();

map = TMXTiledMap::create("map.tmx");

addChild(map);

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

//map->addChild(sprite,10); **//小球在移动过程会在障碍物的上方**

map->addChild(sprite); **//小球在移动过程会被障碍物遮挡**

sprite->setPosition(16.5f, 16.5f);

auto\* pLeft = MenuItemImage::create("left.png", "left1.png", this, menu\_selector(HelloWorld::left));

auto\* left = Menu::create(pLeft, NULL);

left->setPosition(500, 80);

addChild(left);

auto\* pUp = MenuItemImage::create("up.png", "up1.png", this, menu\_selector(HelloWorld::up));

auto\* up = Menu::create(pUp, NULL);

up->setPosition(550, 130);

addChild(up);

auto\* pRight = MenuItemImage::create("right.png", "right1.png", this, menu\_selector(HelloWorld::right));

auto\* right = Menu::create(pRight, NULL);

right->setPosition(600, 80);

addChild(right);

auto\* pDown = MenuItemImage::create("down.png", "down1.png", this, menu\_selector(HelloWorld::down));

auto\* down = Menu::create(pDown, NULL);

down->setPosition(550, 30);

addChild(down);

向上行走的事件代码：

void HelloWorld::up(cocos2d::Object\* pSender)

{

float y = sprite->getPositionY();

if (y + 16.5<MAP\_SIZE)

{

sprite->setPositionY(y + 33);

if ((y + map->getPositionY() > size.height / 2) && ((MAP\_SIZE - y)>size.height / 2))

{

map->setPositionY(map->getPositionY() - 33);

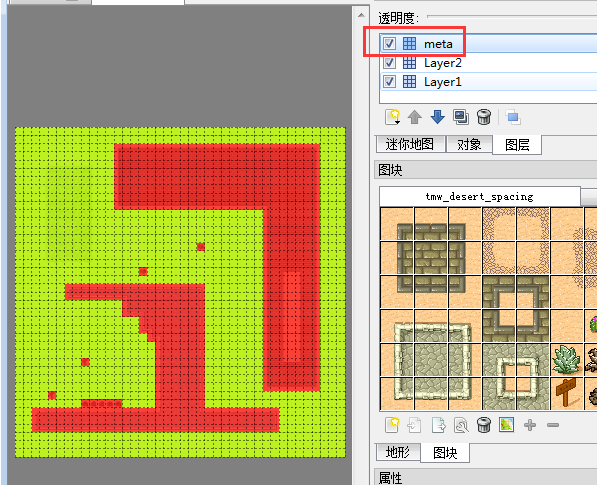
}

}

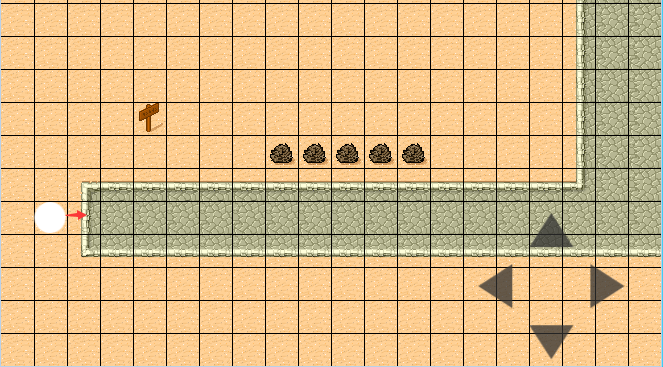
}

如何获取障碍物

使用Tiled Map Editor工具增加图层



在程序中加载地图后，运行效果如下图所示：



程序代码较上个示例改动的部分：

size = Director::getInstance()->getVisibleSize();

map = TMXTiledMap::create("map.tmx");

addChild(map);

**meta = map->layerNamed("meta");**

**meta->setVisible(false);**

增加判断是否达到障碍物的方法

其中tileGIDAt方法的作用是，获取瓦片的Grid值。

下面两行代码的作用是，将精灵在地图中的坐标转换为瓦片地图的坐标。

int mapX = (int)((x - 16.5) / 33);

int mapY = (int)(39 - (y - 16.5) / 33);

bool HelloWorld::isCanReach(float x, float y)

{

bool result;

int mapX = (int)((x - 16.5) / 33);

int mapY = (int)(39 - (y - 16.5) / 33);

int tileGid = meta->**tileGIDAt**(Vec2(mapX, mapY));

if (tileGid == 49)

{

result = true;

}

else

{

result = false;

}

return result;

}

**（5）实验3-1 用摇杆控制人物行走**

**（6）实验3-2 实现带背景的人物行走效果**

**（7）实验3-3 使用分层的方式来控制人物行走路径**

**（8）实验5-2 瓦片地图的使用方法**