### 3.3 在Cocos2d-x中使用物理引擎

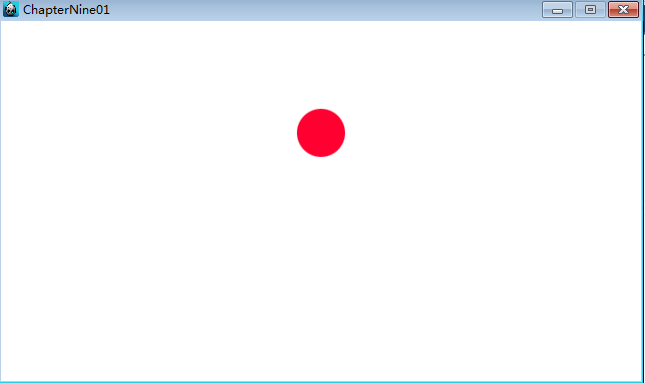
在游戏开发中，经常会模拟现实中发生的一些物理现象，如小球碰撞、物块的下落等。本节将学习利用Box2D实现一个在Cocos2d-x中模拟台球碰撞的例子，来展示使用物理引擎的优势。

【本节知识点】

* 物理引擎的起源、作用及其基本概念
* 在Cocos2d-x中物理引擎的选择
* 在Cocos2d-x中使用物理引擎的方法
* 如何在Cocos2d-x中使用物理引擎模拟现实中的世界

**（1）什么是物理引擎**

如何实现物体下落的效果，如下图：



**匀速运动：**

auto\* background = LayerColor::create(Color4B(255, 255, 255, 255));

addChild(background);

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

ball->setPosition(320, 300);

addChild(ball);

this->scheduleUpdate();

void HelloWorld::update(float dt)

{

float y = ball->getPositionY();

if (y > ball->getContentSize().height/2)

{

ball->setPositionY(y - dy);

}

}

**不是匀速运动是，update方法修改如下：**

void HelloWorld::update(float dt)

{

float y = ball->getPositionY();

if (y > ball->getContentSize().height/2)

{

**dy = dy + increment;**

ball->setPositionY(y - dy);

}

}

在物体下落过程中，其过程完全是可预料的，但是在实现游戏的过程中，一切都是不可预料的。如游戏中，被“小鸟”击中的石块的掉落，完全是不可预料的，如何在这样复杂的情形下进行响应。

物理引擎是一系列能够实现在游戏开发中对现实世界中的物理现象进行模拟的第三方库的总称。它可以在系统中创建一个独立的“世界”，在这个“世界”中通过对对象的某些属性（如弹性、速度等）来对刚体在现实世界中的行为进行模拟。

物理引擎的常用概念：

**世界：**人们将所能看到的，听到的，想到的一切都归结在这个世界中。在物理引擎中也是如此，世界是物理引擎中最高层次的一个概念。物理引擎的一切行为（比如运动、碰撞、爆炸等）等要在世界中才能够进行。

在开始使用物理引擎时，通常会对世界做出一个定义，比如当前的世界是屏幕所显示的内容，那么假设一个石块沿抛物线下落，当它废除屏幕的范围后，物理引擎将不再会对其轨迹做任何的运算。

**刚体：**刚体这个名词实际上时借用了物理学中对刚体的某些假设。在物理学中存在着这样的一种物体，在运动中或受力作用后，形状和大小不变，而且内部各点的相对位置不变，这就是刚体。在物理引擎中，刚体通常分为3类：

静态刚体：固定不动的物体。如：无法被破坏的砖、管道等。

动态刚体：世界中可以受到力的作用发生改变的刚体。如：游戏中的弹球，会受到砖块弹力而改变方向。

平台刚体：它并不像字面理解的平台那样是静止不动的，而是以一种相同的方式反复地运动。它既不像静态刚体那样静止不动，又不像动态刚体那样会受到力的作用。如：超级玛丽中的“云梯”。平台刚体也可称为“移动的平台刚体”。

**形状：**前面学习的刚体通常都有一定的形状，如云梯的形状，就看作是矩形，弹球则是圆形。在物理引擎中，通常将刚体的形状分为两类：圆形和多边形，这里的多边形可以是三角形、矩形、六边形，但通常指的是凸多边形。

一些比较复杂的形状，通常是由多个多边形组合而成的。

**夹具：**“火柴人”是由多个刚体组成的，那么组成它整体的每一个刚体都可以看做是一个夹具，比如说头是一个夹具，而每根胳膊都是由两个矩形的刚体作为夹具组合而成的。因此可以说：**夹具是组成复杂形状的简单形状的单位**。

复杂的形状是通过夹具组合在一起的，但是却不可能凭空组合，就好比现实中的两块钢板，把它们放在一起回很容易被分开。但是却可以使用胶水、螺丝等材料使它们彻底地组合在一起，这就是关节的作用。

**关节：在物理引擎中存在着许多种不同的关节，通过关节的组合，物体除了能够组成比较复杂的形状，还能够实现不同的动作**，如汽车的车轮滚动、机械臂的伸长缩短等。

关节的种类主要有：线关节、棱镜关节、旋转关节、滑轮关节、齿轮关节、鼠标关节等。

线关节：只允许夹具间进行直线运动，单并不限制夹具的旋转。（例如：圆形在轴上的平移和旋转运动）

棱镜关节：仅允许夹具间进行规定的直线方向的相对滑动，但是却不允许夹具自身转动。

旋转关节：就像一个车轴，仅允许夹具围绕规定好的锚点进行旋转，不如车轮和车体通过车轴连接时车轴就是一个旋转夹具。

滑轮关节：允许开发者对一些复杂的机械运动进行定义。

齿轮关节：滑轮关节的驱动是通过类似于绳子的作用来实现的，而齿轮关节则更多的是一种直接驱动的关系。

鼠标关节：这是在物理引擎尤其是当开发移动游戏时使用物理引擎非常重要的一种关节，用来实现玩家的操作与世界中的物体进行连接。

**碰撞：**可以说在物理引擎中的许多操作都是为了进行一个终极目的----实现碰撞检测来服务的。在使用了物理引擎的游戏中，最令人激动的部分就是通过物理引擎对碰撞做出的响应带来的不确定的结果所产生的多样性。

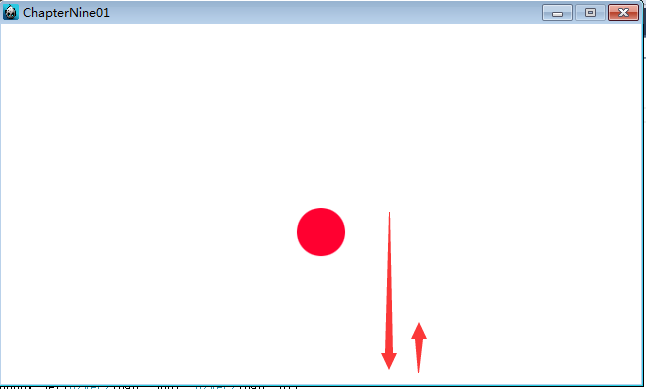
**（2）在Cocos2d-x中使用物理引擎**

Cocos2d-x默认集成了Box2D和Chipmunk，有一件非常麻烦的事情就是这两款引擎无法同时使用，因此在使用时必须选定某一个固定的引擎。

Box2D使用C++编写，而Chipmunk使用C语言编写，提供了Objective-C风格的接口。基于这一点，相信我们对于使用哪款引擎已经有了自己的决定。如果开发IOS游戏使用Chipmunk是一个不错的方法，而在Cocos2d-x中自然是使用Box2D会更加方便。

示例：在Cocos2d-x中使用Box2D

程序运行后界面如下图所示。可以看到程序运行后一个红色的小球出现在屏幕中央偏上的位置，然后开始做自由落体运动。通过观察能够发现，小球的运动速度会不断加块，知道落到地面。在屏幕上还能看出小球落地后，因为弹力弹起知道最终完全静止的效果。



**【代码解释】**

**头文件：**

**#include "Box2D\Box2D.h"**

**声明世界中的重力。如果是失重状态或是要做类似桌球那样不受重力影响的游戏，则可以将其值全部设为0。**

**b2Vec2 gravity; // 创建重力**

**gravity.Set(0.0f, -2.0f); // 定义重力参数**

**创建世界**

**world = new b2World(gravity); // 生成世界对象，初始化时引入gravity重力对象**

**world->SetAllowSleeping(true); //允许休眠**

**world->SetContinuousPhysics(true); //设置检测连续碰撞**

**地面定义。需要观察小球落地后被弹起来的效果，因此定义地面是比不可少的。一般在弹球类游戏中，会直接将屏幕的四边封死。使用的方法是定义4个刚体作为夹具组成一个“框”，来阻止小球落到屏幕外。**

**groundBox是一个形状，分别定义了上、下、左、右4个方向的边界的形状。通过CreateFixture方法将每一个形状规定到刚体上去。**

**//根据钢体的描述创建钢体对象**

**b2Body\* groundBody = world->CreateBody(&groundBodyDef);**

**groundBody->SetType(b2\_staticBody); //刚体类型，静态刚体**

**b2EdgeShape groundBox; //定义地面盒形状**

**//2个点确定一条线，以下分别设置了屏幕的4个边**

**groundBox.Set(b2Vec2(0, 0), b2Vec2(640, 0));**

**groundBody->CreateFixture(&groundBox, 0);**

**groundBox.Set(b2Vec2(0, 360), b2Vec2(640, 360));**

**groundBody->CreateFixture(&groundBox, 0);**

**groundBox.Set(b2Vec2(0,360), b2Vec2(0, 0));**

**groundBody->CreateFixture(&groundBox, 0);**

**groundBox.Set(b2Vec2(640, 360), b2Vec2(640, 0));**

**groundBody->CreateFixture(&groundBox, 0);**

**创建小球，动态刚体**

**b2BodyDef bodyDef; // 定义动态刚体**

**bodyDef.type = b2\_dynamicBody; // 刚体类型为动态刚体**

**bodyDef.position.Set(320, 300);**

**b2Body \*ballbody = world->CreateBody(&bodyDef);**

**ballbody->SetUserData(ball);**

**ballbody->SetActive(true);**

**在update方法中对小球的状态进行更新。**

**Update方法的作用是：获取世界中的全部刚体，然后进行遍历。**

**Sprite \*sprite = (CCSprite \*)b->GetUserData();**

上面一行代码的作用是：获取在场景中创建的精灵对象。

【代码】

auto\* background = LayerColor::create(Color4B(255, 255, 255, 255));

addChild(background);

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

ball->setPosition(320, 300);

addChild(ball);

b2Vec2 gravity; // 创建重力

gravity.Set(0.0f, -2.0f); // 定义重力参数

world = new b2World(gravity); // 生成世界对象

world->SetAllowSleeping(true); //允许休眠

world->SetContinuousPhysics(true); //设置检测连续碰撞

b2BodyDef groundBodyDef; // 定义刚体

groundBodyDef.position.Set(0, 0); // 设置左下角为坐标原点

//根据钢体的描述创建钢体对象

b2Body\* groundBody = world->CreateBody(&groundBodyDef);

groundBody->SetType(b2\_staticBody);

b2EdgeShape groundBox; //定义地面盒形状

//2个点确定一条线，以下分别设置了屏幕的4个边

groundBox.Set(b2Vec2(0, 0), b2Vec2(640, 0));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(0, 360), b2Vec2(640, 360));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(0,360), b2Vec2(0, 0));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(640, 360), b2Vec2(640, 0));

groundBody->CreateFixture(&groundBox, 0);

b2BodyDef bodyDef; // 定义动态刚体

bodyDef.type = b2\_dynamicBody;

bodyDef.position.Set(320, 300);

b2Body \*ballbody = world->CreateBody(&bodyDef);

ballbody->SetUserData(ball);

ballbody->SetActive(true);

//给这个动态钢体创建盒子形状

b2PolygonShape dynamicBox;

dynamicBox.SetAsBox(0.5f, 0.5f);

b2FixtureDef fixtureDef; //为该钢体定义夹具

fixtureDef.shape = &dynamicBox; //绑定物体的形状

fixtureDef.density = 0.2f; //密度

fixtureDef.restitution = 0.9f; //弹性系数

fixtureDef.friction = 0.1f; //摩擦力

ballbody->CreateFixture(&fixtureDef); //在刚体的夹具工厂中创建夹具

this->scheduleUpdate();

void HelloWorld::update(float dt)

{

int velocityIterations = 8;

int positionIterations = 1;

//时间定义成固定的，如果用dt的话，会随着帧率的变化而变化

world->Step(0.2f, velocityIterations, positionIterations);

//枚举世界中的所有物体，找出有userdata的物体，强制转化为精灵

for (b2Body \*b = world->GetBodyList(); b; b = b->GetNext())

{

if (b->GetUserData() != NULL)

{

Sprite \*sprite = (CCSprite \*)b->GetUserData();

if (sprite != NULL)

{

b2Vec2 bodyPos = b->GetPosition(); // 获取小球对象

Point pos = ccp(bodyPos.x, bodyPos.y); // 将小球的坐标转换为box2d的坐标

float32 rotation = -1 \* CC\_RADIANS\_TO\_DEGREES(b->GetAngle()); //转换角度

sprite->setPosition(pos); //设置精灵坐标

sprite->setRotation(rotation); //设置精灵角度

}

}

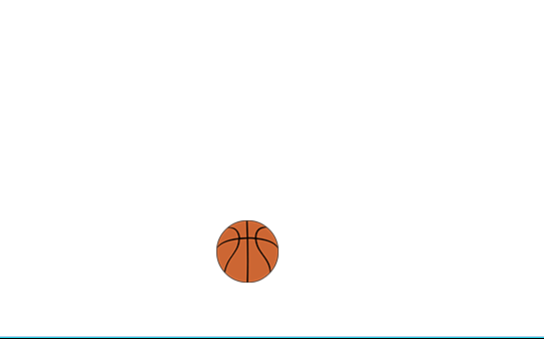
}

}

**（3）玩家实现对刚体的操纵**

前面学习了物理引擎最基本的使用，使一个小球从半空中落下。但是开发游戏，就少不了与玩家的交互，本节在上一节内容的基础上，学习为物理引擎加入以下交互性的功能，使玩家可以通过手指在屏幕上滑动来操作小球。

程序运行后，运行效果如下图所示。初始运行，屏幕中央偏上的位置会出现一个下落的小球。如果操作员用手滑动屏幕，屏幕中的小球就会随着玩家的手指改变方向。除了位移的改变，在本例中还能够看到由于摩擦力的作用，小球在从地面弹起时，发生了一下旋转。



【代码解释】

【代码】

auto\* background = LayerColor::create(Color4B(255, 255, 255, 255));

addChild(background);

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

ball->setPosition(320, 200);

addChild(ball);

b2Vec2 gravity = b2Vec2(0.0f, -0.5f);

\_world = new b2World(gravity);

\_world->SetAllowSleeping(true);

\_world->SetContinuousPhysics(true);

b2BodyDef groundBodyDef;

groundBodyDef.position.Set(0, 0);

// 刚体

b2Body \*groundBody = \_world->CreateBody(&groundBodyDef);

groundBody->SetType(b2\_staticBody);

// 形状

b2EdgeShape groundBox;

// 夹具

b2FixtureDef boxShapeDef;

// 为夹具加入形状

groundBox.Set(b2Vec2(0, 0), b2Vec2(640, 0));

groundBody->CreateFixture(&groundBox,0);

groundBox.Set(b2Vec2(0, 360 ), b2Vec2(640 , 360 ));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(0, 0), b2Vec2(0, 360 ));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(640 , 0), b2Vec2(640 , 360 ));

groundBody->CreateFixture(&groundBox, 0);

boxShapeDef.shape = &groundBox;

// 加入球

b2BodyDef ballBodyDef;

ballBodyDef.type = b2\_dynamicBody;

ballBodyDef.position.Set(320, 200 );

ballBodyDef.userData = ball;

b2CircleShape circle;

circle.m\_radius = 32.0 ;

\_body = \_world->CreateBody(&ballBodyDef);

b2FixtureDef ballShapeDef;

ballShapeDef.shape = &circle;

ballShapeDef.density = 1.0f;

ballShapeDef.friction = 0.1;

ballShapeDef.restitution = 0.9f;

\_body->CreateFixture(&ballShapeDef);

auto\* dispatcher = Director::getInstance()->getEventDispatcher();

auto\* listener = EventListenerTouchOneByOne::create();

// 绑定监听事件

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

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

listener->setSwallowTouches(true);

dispatcher->addEventListenerWithSceneGraphPriority(listener, this);

this->scheduleUpdate();

void HelloWorld::update(float dt)

{

int velocityIterations = 8;

int positionIterations = 1;

\_world->Step(0.2f, velocityIterations, positionIterations);

for (b2Body \*b = \_world->GetBodyList(); b; b = b->GetNext())

{

if (b->GetUserData() != NULL)

{

Sprite \*sprite = (CCSprite \*)b->GetUserData();

if (sprite != NULL)

{

b2Vec2 bodyPos = b->GetPosition(); // 获取小球对象

Point pos = ccp(bodyPos.x, bodyPos.y); // 将小球的坐标转换为box2d的坐标

float32 rotation = -1 \* CC\_RADIANS\_TO\_DEGREES(b->GetAngle()); //转换角度

sprite->setPosition(pos); //设置精灵坐标

sprite->setRotation(rotation); //设置精灵角度

}

}

}

}

**单点触摸摩擦效果的实现：**

**在玩家开始触摸屏幕时，在onTouchBegan方法中，将玩家触摸屏幕位置的坐标保存在变量prePos中。当用户触摸结束时，在onTouchEnded方法中获得了一个新的坐标position，将它们两者相减就获得了玩家滑动屏幕的向量。将这个向量缩小若干倍之后就能得到一个新的向量。**

**GetLinearVelocity方法获得小球的当前速度。将这个速度与之前获得的向量进行相加，就得到了一个新的速度。**

**SetLinearVelocity方法将新速度设置到小球上，小球就会沿着新的方向运动了。**

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

{

prePos = touch->getLocation();

return true;

}

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

{

auto position = touch->getLocation();

float dx = (position.x - prePos.x)/16;

float dy = (position.y - prePos.y)/16;

b2Vec2 v = \_body->GetLinearVelocity();

v.x += dx;

v.y += dy;

\_body->SetLinearVelocity(v);

}

**（4）Box2D中关于倍数的设定**

在前面的示例中，没有特别的设置，场景中的一个像素实际上就代表现实世界中的1米。按照这样的比例，在上一节中使用的“小”球实际上应该是一个半径为32米的庞然大物。如果，如果真让它能够到处滚动，那杀伤力无疑是巨大的。

为了让小球处于0.1~10米这样的大小以便Box2D能够对其进行优化，就需要定义一个类似比例尺一样的东西。

本节示例如下图所示。运行程序后，我们发现屏幕上的小球比之前活泼多了。运行速度比之前有了很大提高。原因是：同样是1米，在没有使用比例尺时，可能看不出任何效果，但是如果在使用了比例尺后，在屏幕上移动了会是很大的一段距离，毕竟，对于重力参数的设置是没有改变的。



**【代码解释】**

#define PTM\_RATIO 32

**上面语句的目的是将项目中使用到的数据比如坐标、半径等都统一地按照一定的比例缩小。比如半径为32的小球，经过缩小之后，半径就变成了1，直径也不过是2米，属于0.1~10米的范围，Box2D就会对它进行优化。**

**【代码】**

#define PTM\_RATIO 32

**部分代码**

// 为夹具加入形状

groundBox.Set(b2Vec2(0, 0), b2Vec2(640 / PTM\_RATIO, 0));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(0, 360 / PTM\_RATIO), b2Vec2(640 / PTM\_RATIO, 360 / PTM\_RATIO));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(0, 0), b2Vec2(0, 360 / PTM\_RATIO));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(640 / PTM\_RATIO, 0), b2Vec2(640 / PTM\_RATIO, 360 / PTM\_RATIO));

groundBody->CreateFixture(&groundBox, 0);

boxShapeDef.shape = &groundBox;

// 加入球

b2BodyDef ballBodyDef;

ballBodyDef.type = b2\_dynamicBody;

ballBodyDef.position.Set(320 / PTM\_RATIO, 200 / PTM\_RATIO);

ballBodyDef.userData = ball;

b2CircleShape circle;

circle.m\_radius = 32.0 / PTM\_RATIO;

\_body = \_world->CreateBody(&ballBodyDef);

void HelloWorld::update(float dt)

{

int velocityIterations = 8;

int positionIterations = 1;

\_world->Step(0.2f, velocityIterations, positionIterations);

for (b2Body \*b = \_world->GetBodyList(); b; b = b->GetNext())

{

if (b->GetUserData() != NULL)

{

Sprite \*sprite = (CCSprite \*)b->GetUserData();

if (sprite != NULL)

{

b2Vec2 bodyPos = b->GetPosition(); // 获取小球对象

Point pos = ccp(bodyPos.x\*PTM\_RATIO, bodyPos.y\*PTM\_RATIO); // 将小球的坐标转换为box2d的坐标

float rotation = -1 \* CC\_RADIANS\_TO\_DEGREES(b->GetAngle()); //转换角度

sprite->setPosition(pos); //设置精灵坐标

sprite->setRotation(rotation); //设置精灵角度

}

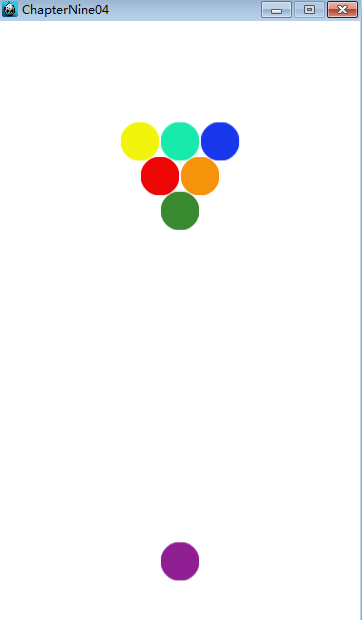
}

}

}

**（5）在屏幕中加入多个小球**

本节实现一个在平台上打台球的效果，运行效果如下图所示。



**【代码】**

Point ballPos[7] = { Vec2(180,100),

Vec2(180,450),

Vec2(180 - 20, 450 + 20\*SQRT\_3 ),

Vec2(180 + 20, 450 + 20 \* SQRT\_3 ),

Vec2(180 - 40, 450 + 40\*SQRT\_3 ),

Vec2(180, 450 + 40\*SQRT\_3 ),

Vec2(180 + 40, 450 + 40 \* SQRT\_3) };

for (int i = 0; i < 7; i++)

{

ball[i] = Sprite::create(String::createWithFormat("ball%d.png", i )->getCString());

ball[i]->setPosition(ballPos[i]);

addChild(ball[i]);

}

// 创建世界，桌球为平面，没有重力

b2Vec2 gravity = b2Vec2(0.0f, 0.0f);

\_world = new b2World(gravity);

\_world->SetAllowSleeping(true);

\_world->SetContinuousPhysics(true);

b2BodyDef groundBodyDef;

groundBodyDef.position.Set(0, 0);

// 刚体

b2Body \*groundBody = \_world->CreateBody(&groundBodyDef);

groundBody->SetType(b2\_staticBody);

// 形状

b2EdgeShape groundBox;

// 夹具

b2FixtureDef boxShapeDef;

// 为夹具加入形状

groundBox.Set(b2Vec2(0, 0), b2Vec2(360 / PTM\_RATIO, 0));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(0, 640 / PTM\_RATIO), b2Vec2(360 / PTM\_RATIO, 640 / PTM\_RATIO));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(0, 0), b2Vec2(0, 640 / PTM\_RATIO));

groundBody->CreateFixture(&groundBox, 0);

groundBox.Set(b2Vec2(360 / PTM\_RATIO, 0), b2Vec2(360 / PTM\_RATIO, 640 / PTM\_RATIO));

groundBody->CreateFixture(&groundBox, 0);

boxShapeDef.shape = &groundBox;

// 加进去七个球

for (int i = 0; i < 7; i++)

{

b2BodyDef ballBodyDef;

ballBodyDef.type = b2\_dynamicBody;

ballBodyDef.position.Set(ballPos[i].x / PTM\_RATIO, ballPos[i].y / PTM\_RATIO);

ballBodyDef.userData = ball[i];

b2CircleShape circle;

circle.m\_radius = 20.0 / PTM\_RATIO;

\_body[i] = \_world->CreateBody(&ballBodyDef);

b2FixtureDef ballShapeDef;

ballShapeDef.shape = &circle;

ballShapeDef.density = 1.0f;

ballShapeDef.friction = 0.5;

ballShapeDef.restitution = 0.8f;

\_body[i]->CreateFixture(&ballShapeDef);

}

【代码解释】

上面代码的作用是：定义了世界范围和创建了7个刚体小球，以及用于显示它们的精灵对象。

由于这次模拟的对象是桌面桌球而不是星星下落的现象，因此在这次创建的世界中，重力为0.

ballBodyDef.userData = ball[i];刚体在被创建时，就可以将刚体与精灵对象直接绑定，这样有利于刚体的遍历，而且也能够解决更新显示内容时寻找对应精灵对象的问题。

void HelloWorld::update(float dt)

{

int velocityIterations = 8;

int positionIterations = 1;

\_world->Step(0.2f, velocityIterations, positionIterations);

for (b2Body \*b = \_world->GetBodyList(); b; b = b->GetNext())

{

if (b->GetUserData() != NULL)

{

Sprite \*sprite = (CCSprite \*)b->GetUserData();

if (sprite != NULL)

{

b2Vec2 bodyPos = b->GetPosition(); // 获取小球对象

Point pos = ccp(bodyPos.x\*PTM\_RATIO, bodyPos.y\*PTM\_RATIO); // 将小球的坐标转换为box2d的坐标

float rotation = -1 \* CC\_RADIANS\_TO\_DEGREES(b->GetAngle()); //转换角度

sprite->setPosition(pos); //设置精灵坐标

sprite->setRotation(rotation); //设置精灵角度

}

}

}

}

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

{

prePos = touch->getLocation();

return true;

}

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

{

auto position = touch->getLocation();

float dx = (position.x - prePos.x) / 16;

float dy = (position.y - prePos.y) / 16;

b2Vec2 v = \_body[0]->GetLinearVelocity();

v.x += dx;

v.y += dy;

\_body[0]->SetLinearVelocity(v);

}

**【代码解释】**

**Update方法中的**

**\_world->Step(0.2f, velocityIterations, positionIterations);**

**每次在update方法中，通过step方法来将时间推进0.2秒，看起来就像是正常前进一样。**

**（6）实验3-6 可操作的刚体**

**（7）实验3-7 可操作的刚体—增加倍数设定**

**（8）实验5-1 多个小球的刚体**