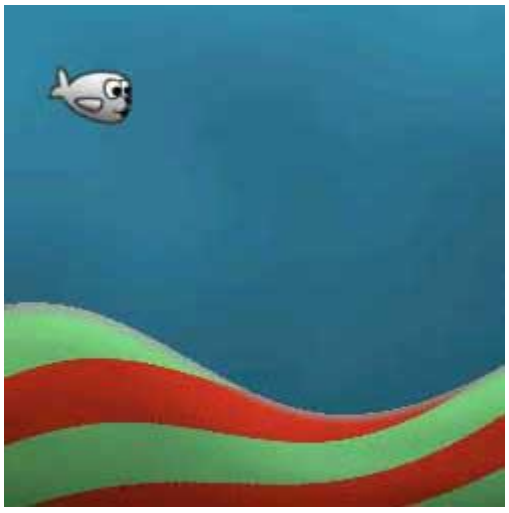
如何制作一个类似 tiny wings 的游戏：第二部分（完）

原文链接地址：
http://www.raywenderlich.com/3913/how-to-create-a-game-like-tiny-wings-part-2

教程截图：



这是本系列教程的最后一部分，主要是教大家如何制作一个类似 Tiny Wings 的游戏。

在预备教程中，我们学会了如何创建动态山丘纹理和背景纹理。

在第一部分教程中，我们学会了如何动态创建游戏里所需要的山丘。

在这篇教程中，也是本系列教程的最后一篇，我们将会学习到更加有意思的部分---如何往游戏里面添加主角，同时使用 BOX2D 来仿真主角的移动！

再说明一下，这个教程系列是基于 Sergey Tikhonov 所写的一个非常好的 demo project 制作的---所以我要特别感谢 Sergey！

这个教程假设你对 cocos2d 和 box2d 已经很熟悉了。如果你对这两者还很陌生的话，建议你先阅读本博客上翻译的 cocos2d 教程和 box2d 教程。

Getting Started

如果你还没有准备好，可以先下载上一篇教程中完成的样例工程。

接下来，我们将添加一些基本的 box2d 代码。我们将创建一个 box2d world 和一些代码来激活 debug drawing，同时还会添加一些测试用 shape，以此确保 BOX2D 环境被正确搭建起来！

首先，打开 HelloWorldLayer.h，然后作如下修改：

```
// Add to top of file
#import "Box2D.h"
#define PTM_RATIO 32.0

// Add inside @interface
b2World * _world;
```

这里包含了 box2d 的头文件和 debug draw 的头文件，同时定义一个_world 变量来追踪 box2d 的 world 与 debug draw 类。

同时，我们也声明了一个像素/米的转换率（PTM_RATIO）为 32.回顾一下，这个变量主要作用是在 box2d 的单位（米）和 cocos2d 的单位（点）之间做转换。

然后，我们在 HelloWorldLayer.mm 中添加一下新的方法，添加位置在 init 方法上面：

```
- (void)setupWorld {
b2Vec2 gravity = b2Vec2(0.0f, -7.0f);
bool doSleep = true;
_world = new b2World(gravity, doSleep);
}
```

```
- (void)createTestBodyAtPostition:(CGPoint)position {

b2BodyDef testBodyDef;
testBodyDef.type = b2_dynamicBody;
testBodyDef.position.Set(position.x/PTM_RATIO, position.y/PTM_RATIO);
b2Body * testBody = _world->CreateBody(&testBodyDef);

b2CircleShape testBodyShape;
b2FixtureDef testFixtureDef;
testBodyShape.m_radius = 25.0/PTM_RATIO;
testFixtureDef.shape = &testBodyShape;
testFixtureDef.density = 1.0;
testFixtureDef.friction = 0.2;
testFixtureDef.restitution = 0.5;
testBody->CreateFixture(&testFixtureDef);

}
```

如果你对 box2d 很熟悉的话，上面这个方法只是一个回顾。

setupWorld 方法创建一个有重力的 world--但是比标准的重力-9.8m/s^2 要小一点点。

createTestBodyAtPostition 创建一个测试对象---一个 25 个点大小的圆。我们将使用这个方法来创建一个测试对象，每一次你点击屏幕就会在那个地方产生一个圆，不过这只是测试用，之后会被删除掉。

你现在还没有完成 HelloWorldLayer.mm--现在再作一些修改，如下所示：

```
// Add to the TOP of init
[self setupWorld];

// Replace line to create Terrain in init with the following
_terrain = [[[Terrain alloc] initWithWorld:_world] autorelease];
```

```
// Add to the TOP of update
static double UPDATE_INTERVAL = 1.0f/60.0f;
static double MAX_CYCLES_PER_FRAME = 5;
static double timeAccumulator = 0;

timeAccumulator += dt;
if (timeAccumulator > (MAX_CYCLES_PER_FRAME * UPDATE_INTERVAL)) {
timeAccumulator = UPDATE_INTERVAL;
}

int32 velocityIterations = 3;
int32 positionIterations = 2;
while (timeAccumulator >= UPDATE_INTERVAL) {
timeAccumulator -= UPDATE_INTERVAL;
_world->Step(UPDATE_INTERVAL,
velocityIterations, positionIterations);
_world->ClearForces();

}

// Add to bottom of ccTouchesBegan
UITouch *anyTouch = [touches anyObject];
CGPoint touchLocation = [_terrain convertTouchToNodeSpace:anyTouch];
[self createTestBodyAtPostition:touchLocation];
```

第一段代码，我们调用 setupWorld 方法来创建一个 box2d 世界。然后使用 box2d 的 world 来初始化 Terrain 类。这样，我们就可以使用这个 world 来创建山丘的 body 了。为此，我们将会写一些桩代码（placeholder）。

第二段代码，我们调用_world->Step 方法来运行物理仿真。注意，这里使用的是固定时间步长的实现方式，它比变长时间步长的方式物理仿真效果要更好。对于具体这个是怎么工作的，可以去看看我们的 cocos2d 书籍中关于 box2d 的那一章节内容。

最后一段代码是添加到 ccTouchesBegan 里面，不管什么时候你点击屏幕，就会创建一个 box2d 的 body。再说一下，这样做只是为了测试 box2d 环境可以 run 起来了。

注意，我们这里得到的 touch 坐标是在地形的坐标之内。这是因为，地形将会滚动，而我们想知道地形的位置，而不是屏幕的位置。

接下来，让我们修改一下 Terrain.h/m。首先，修改 Terrain.h，如下所示：

```
// Add to top of file
#import "Box2D.h"
#import "GLES-Render.h"

// Add inside @interface
b2World *_world;
b2Body *_body;
GLESDebugDraw * _debugDraw;

// Add after @interface
- (id)initWithWorld:(b2World *)world;
```

这里只是包含 box2d 头文件,然后创建一些实例变量来追踪 box2d 的 world,以及山丘的 body,还有支持 debug drawing 的对象。同时，我们还定义了初始化方法，它接收 box2d 的 world 作为参数。

然后在 Terrain.m 中添加一个新的方法，位置在 generateHills 上面：

```
- (void) resetBox2DBody {

if(_body) return;

CGPoint p0 = _hillKeyPoints[0];
CGPoint p1 = _hillKeyPoints[kMaxHillKeyPoints-1];

b2BodyDef bd;
bd.position.Set(0, 0);
_body = _world->CreateBody(&bd);
```

```
b2PolygonShape shape;
b2Vec2 ep1 = b2Vec2(p0.x/PTM_RATIO, 0);
b2Vec2 ep2 = b2Vec2(p1.x/PTM_RATIO, 0);
shape.SetAsEdge(ep1, ep2);
_body->CreateFixture(&shape, 0);
}
```

这里仅仅是一个辅助方法，用来创建山丘的的底部 body，代表“地面”。这里只是暂时用这个方法，用来防止随机生成的圆会掉到屏幕之外去。之后，在我们建模好山丘后，我们会再次修改。

目前，我们只是把第一个关键点和最后一个关键点用一条边连接起来。

接下来，在 Terrain.m 中添加一些代码来调用上面的代码，同时建立起 debug drawing：

```
// Add inside resetHillVertices, right after "prevToKeyPointI = _toKe
yPointI" line:
[self resetBox2DBody];

// Add new method above init
- (void)setupDebugDraw {
_debugDraw = new GLESDebugDraw(PTM_RATIO*[[CCDirector sharedDirector]
 contentScaleFactor]);
_world->SetDebugDraw(_debugDraw);
_debugDraw->SetFlags(b2DebugDraw::e_shapeBit | b2DebugDraw::e_jointBi
t);
}


// Replace init with the following
- (id)initWithWorld:(b2World *)world {
if ((self = [super init])) {
_world = world;
[self setupDebugDraw];
```

```
[self generateHills];
[self resetHillVertices];
}
return self;
}

// Add at bottom of draw
glDisable(GL_TEXTURE_2D);
glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_TEXTURE_COORD_ARRAY);

_world->DrawDebugData();

glEnable(GL_TEXTURE_2D);
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
```

每一次山丘顶点被重置的时候，我们调用 resetBox2DBody 来创建可见部分山丘的 body。目前，这个 body 是不变的（它只是添加了一条线，当作地面）。但是，接下来，我们将修改这个来建模可见部分的山丘。

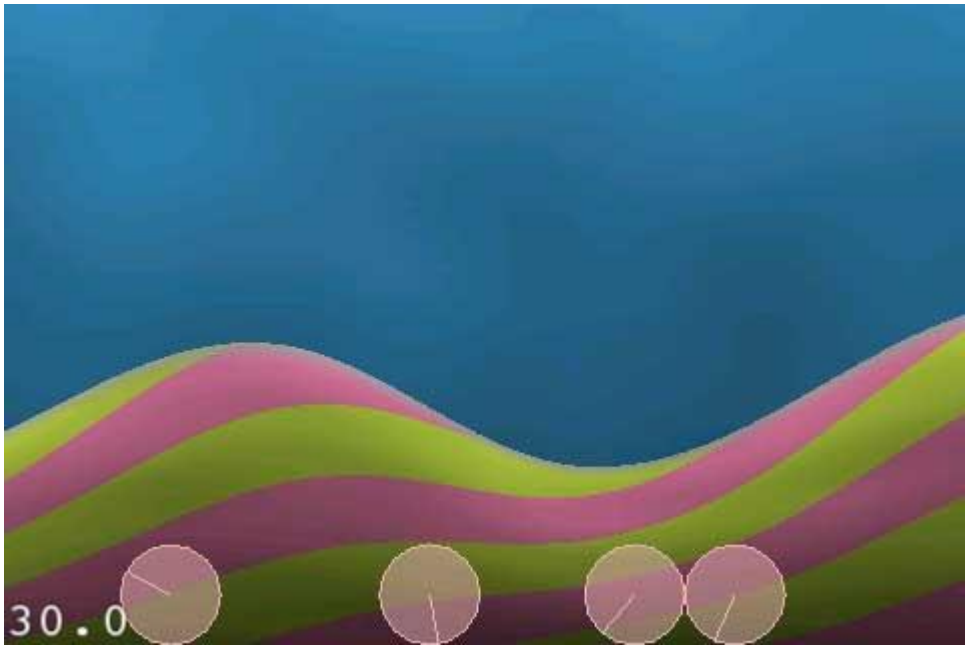
setupDebugDraw 方法设置了激活 box2d 对象 debug drawing 所需要的一些配置。如果你熟悉 box2d 的话，那么这个就是回顾啦。

然后，你可能会奇怪，为什么 debug draw 的代码要放在 Terrain.m 文件中呢？而不是放在 HelloWorldLayer.mm 中呢？这是因为，这个游戏中的滚动效果是在 Terrain.m 中实现的。因此，为了使 box2d 的坐标系统和屏幕范围内可见部分的坐标系统匹配起来，我们就把 debug drawing 代码放在 Terrain.m 中了。

最后一步，如果你现在想要编译的话，可能会出现几百个错误。这是因为 Terrain.m 导入了 Terrain.h 文件，而 Terrain.h 文件又包含了 HelloWorldLayer.h 文件，而它又导入了 Box2D.h 头文件。而不管什么时候，只要你在.m 文件中使用 c++的话，那么就会产生一大堆的错误。

不过还好，解决办法非常简单---只要把 Terrain.m 改成 Terrain.mm 就可以了。

编译并运行，现在，你点击一下屏幕，你会看到许多圆形对象掉在屏幕里面拉！



在 box2d 里面为山丘定义 body 边界

现在，我们只拥有一个 box2d 的 shape 代表屏幕的底部边界，但是，我们真正想要的是代表山丘边界的 shape。

幸运的是，因为我们拥有所有的线段了，所以添加边界会非常简单！

- 我们有一个山丘顶部所有顶点的数组(borderVertices). 在上一个教程的 resetHillVertices 方法中，我们生成了这样一个数组。
- 我们有一个方法，不管什么时候顶点为被改变了，它都会被调用，那就是 resetBox2DBody.

因此，我们需要修改 resetBox2DBody 方法，我们要为 borderVertices 组织中的每一个实体创建一条边，具体方法如下：

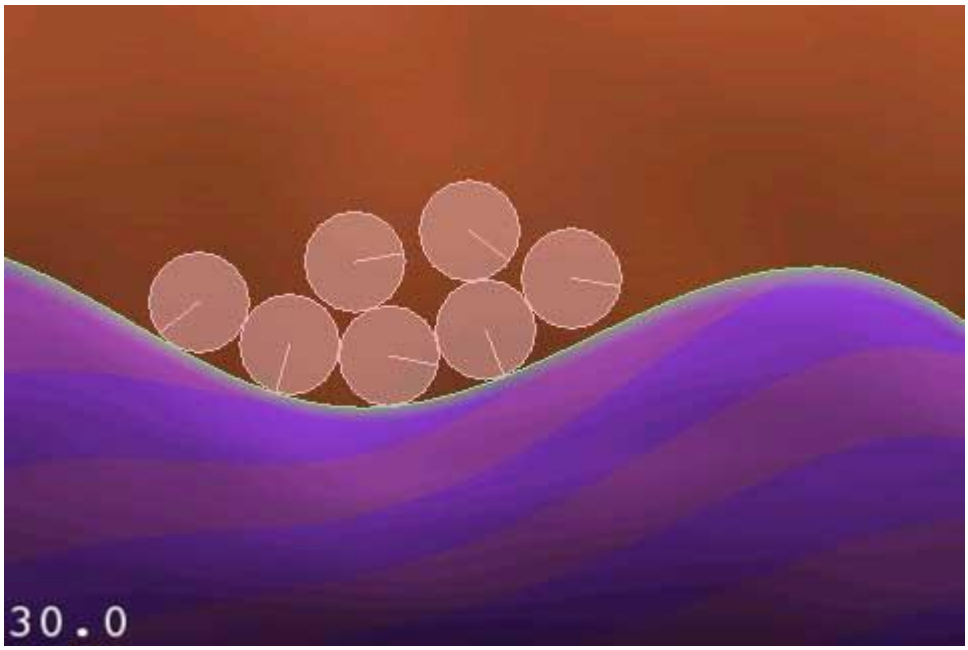
```
- (void) resetBox2DBody {

if(_body) {
_world->DestroyBody(_body);
}
```

```
b2BodyDef bd;
bd.position.Set(0, 0);

_body = _world->CreateBody(&bd);

b2PolygonShape shape;

b2Vec2 p1, p2;
for (int i=0; i<_nBorderVertices-1; i++) {
p1 = b2Vec2(_borderVertices[i].x/PTM_RATIO,_borderVertices[i].y/PTM_R
ATIO);
p2 = b2Vec2(_borderVertices[i+1].x/PTM_RATIO,_borderVertices[i+1].y/P
TM_RATIO);
shape.SetAsEdge(p1, p2);
_body->CreateFixture(&shape, 0);
}
}
```

这个新的实现首先看看是不是存在一个已有的 box2d body，如果是的话，就销毁原来的 body。

然后，它创建一个新的 body，循环遍历 border vertices 数组里面的所有顶点，这些顶点代表山丘顶部。对于每 2 个顶点，都将创建一条边来连接它们。

很简单，对不对？编译并运行，现在，你可以看到一个带有斜坡的 box2d body 了，而且它沿着山丘的纹理边界。



添加海豹

我们之前把工程命名为 Tiny Seal，可是并没有 seal 啊！

接下来，让我们把海豹添加进去！

首先，下载并解压这个工程的资源文件，然后把"Sprite sheets“和"Sounds“直接拖到工程里去，对于每一个文件夹，都要确保“Copy items into destination group’s folder”被复选中，然后点击"Finish”。

然后，点击 File\New\New File，选择 iOS\Cocoa Touch\Objective-C class，再点 Next。选择 CCSprite 作为基类，再点 Next，然后把文件命名为 Hero.mm（注意，.mm 是因为我们将使用到 box2d 的东西），最后点击 Finish.

接着，把 Hero.h 替换成下面的内容：

```
#import "cocos2d.h"
#import "Box2D.h"

#define PTM_RATIO 32.0

@interface Hero : CCSprite {
b2World *_world;
```

```
b2Body *_body;
BOOL _awake;
}

- (id)initWithWorld:(b2World *)world;
- (void)update;

@end
```

这个也非常简单---只是导入 box2d.h 头文件，然后定义一些变量来追踪 world 和海豹的 body.

然后，打开 Hero.mm，然后作如下修改：

```
#import "Hero.h"

@implementation Hero

- (void)createBody {

float radius = 16.0f;
CGSize size = [[CCDirector sharedDirector] winSize];
int screenH = size.height;

CGPoint startPosition = ccp(0, screenH/2+radius);

b2BodyDef bd;
bd.type = b2_dynamicBody;
bd.linearDamping = 0.1f;
bd.fixedRotation = true;
bd.position.Set(startPosition.x/PTM_RATIO, startPosition.y/PTM_RATIO);
_body = _world->CreateBody(&bd);

b2CircleShape shape;
shape.m_radius = radius/PTM_RATIO;
```

```
b2FixtureDef fd;
fd.shape = &shape;
fd.density = 1.0f;
fd.restitution = 0.0f;
fd.friction = 0.2;

_body->CreateFixture(&fd);

}

- (id)initWithWorld:(b2World *)world {

if ((self = [super initWithSpriteFrameName:@"seal1.png"])) {
_world = world;
[self createBody];
}
return self;

}

- (void)update {

self.position = ccp(_body->GetPosition().x*PTM_RATIO, _body->GetPosit
ion().y*PTM_RATIO);
b2Vec2 vel = _body->GetLinearVelocity();
b2Vec2 weightedVel = vel;
float angle = ccpToAngle(ccp(vel.x, vel.y));
if (_awake) {
self.rotation = -1 * CC_RADIANS_TO_DEGREES(angle);
}
}

@end
```

createBody 方法为海豹创建了一个圆形的 shape。这个方法和之前写过的 createTestBodyAtPosition 方法几乎没有什么区别，除了圆的大小和海豹图片的大小要匹配（不过实际上要比图片大小小一些，这样子碰撞检测效果会更好）

同时，这里的摩擦系数（friction）设置为 0.2（因为海豹是很滑的），同时反弹系数(restitution）设置为 0（这样子，当海豹碰撞到山丘的时候就不会反弹起来了）。

同时，我们也设置 body 的线性阻尼（ linear damping），这样子海豹就会随着时间慢慢减速。同时，设置 body 的固定旋转为真，这样子，海豹在游戏里面就不会旋转 body 了。

在 initWithWorld 方法里面，我们把精灵初始化为一个特定的精灵帧（seal1.png），同时保存一份 world 的指针，然后调用上面的 createBody 方法。

这里的 update 方法基于 box2d body 的位置来更新海豹精灵的位置，同时基于海豹的 body 的速度来更新海豹精灵的旋转。

接下来，你需要修改一下 Terrain.h 和 Terrain.mm，因为，我们将要在 Terrain.mm 中添加一个 sprite batch node。

首先，打开 Terrain.h，并作以下修改：

```
// Inside @interface
CCSpriteBatchNode * _batchNode;

// After @implementation
@property (retain) CCSpriteBatchNode * batchNode;
```

然后，打开 Terrain.mm，并作如下修改：

```
// Add to top of file
@synthesize batchNode = _batchNode;

// Add at bottom of init
_batchNode = [CCSpriteBatchNode batchNodeWithFile:@"TinySeal.png"];
[self addChild:_batchNode];
```

```
[[CCSpriteFrameCache sharedSpriteFrameCache] addSpriteFramesWithFile:
@"TinySeal.plist"];
```

这里只是为 TinySeal.png 精灵表单创建了一个 batch node，然后从 TinySeal.plist 文件中加载了精灵帧的定义信息到 sprite frame cache 中。

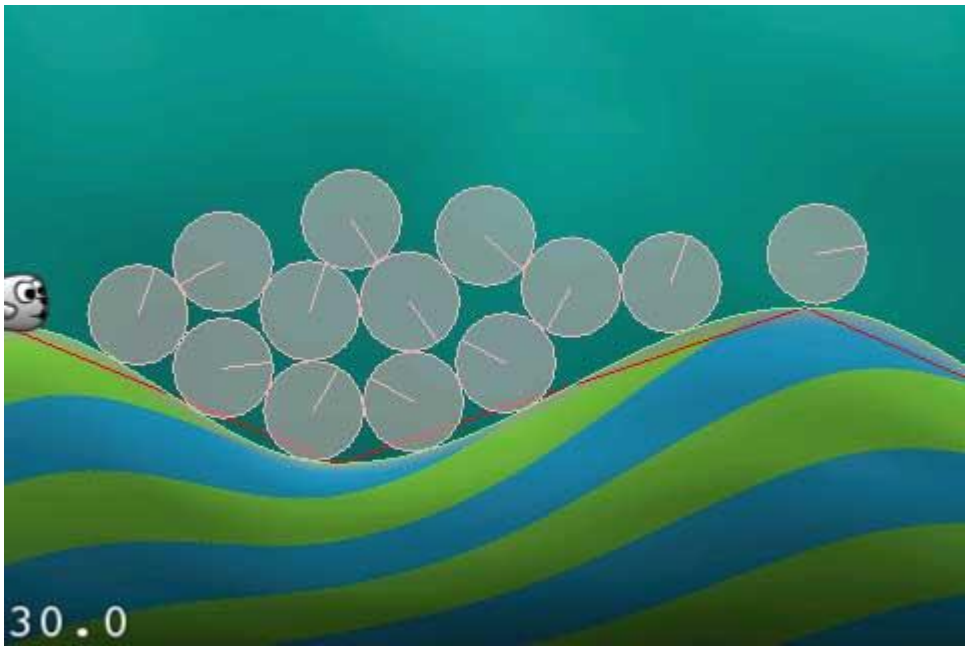
差不多完成了！接下来，让我们修改 HelloWorldLayer.h：

```
// Add to top of file
#import "Hero.h"

// Add inside @interface
Hero * _hero;
```

同时修改 HelloWorldLayer.mm：

```
// Add to bottom of init
_hero = [[[Hero alloc] initWithWorld:_world] autorelease];
[_terrain.batchNode addChild:_hero];

// In update, comment out the three lines starting with PIXELS_PER_SE
COND and add the following
[_hero update];
float offset = _hero.position.x;
```

编译并运行，你现在可以看到一只 happy 的海豹在屏幕左边了！

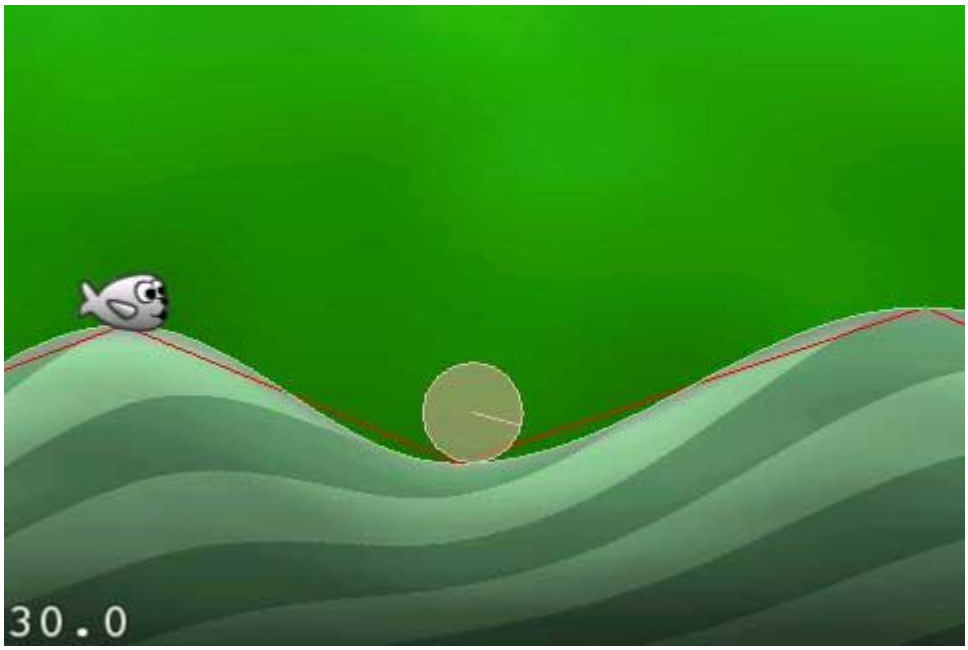


但是，看起来有起奇怪，它在屏幕之外！如果我们把它往右边挪一下，那样子看起来会更好。

当然，这个改起来很简单！打开 Terrain.mm，然后把 setOffsetX 改成下面的样子：

```
- (void) setOffsetX:(float)newOffsetX {
CGSize winSize = [CCDirector sharedDirector].winSize;

_offsetX = newOffsetX;
self.position = CGPointMake(winSize.width/8-_offsetX*self.scale, 0);
[self resetHillVertices];
}
```

这里把海豹的位置旋转在屏幕的 1/8 处，这样子海豹看起来就会往右边一点点了。编译并运行，现在可以看到海豹的全貌啦！



使海豹移动

我们离一个完整的游戏越来越近了---我们有一只海豹，我们只需要让它飞起来就可以啦！

我们采取的策略如下：

- 第一次点击屏幕的时候，我们让海豹稍微往右边跳起来一点点，代表开始了！
- 不管什么时候点击屏幕，我们应用一个冲力使海豹往下落。当海豹下山时，会使它的速度变得更快，这样到下一个山头的时候就可以飞起来了。
- 添加一些代码让海豹移动的距离稍微远一点，我们可不想让我们的海豹卡住！

让我们来实现这些策略吧！打开 Hero.h，作如下修改：

```
// Add after @implementation
@property (readonly) BOOL awake;
- (void)wake;
- (void)dive;
- (void)limitVelocity;
```

然后对 Hero.mm 作如下修改：

```
// Add to top of file
@synthesize awake = _awake;

// Add new methods
- (void) wake {
_awake = YES;
_body->SetActive(true);
_body->ApplyLinearImpulse(b2Vec2(1,2), _body->GetPosition());
}

- (void) dive {
_body->ApplyForce(b2Vec2(5,-50),_body->GetPosition());
}

- (void) limitVelocity {
if (!_awake) return;

const float minVelocityX = 5;
const float minVelocityY = -40;
b2Vec2 vel = _body->GetLinearVelocity();
if (vel.x < minVelocityX) {
vel.x = minVelocityX;
}
if (vel.y < minVelocityY) {
vel.y = minVelocityY;
}
_body->SetLinearVelocity(vel);
}
```

这个 wake 方法应用一个冲力（impulse)使得海豹刚开始往右上方飞。

dive 方法应用一个比较大的向下的冲力，和一个比较小的向右的力。这个向下的冲力会使得海豹往山丘上撞，这时，山丘的斜坡越大，那么小鸟就飞得越高。（应该是上山的时候，下山相反）

limitVelocity 方法确保海豹速度至少在 x 轴方向 5m/s²，Y 轴方向 -40m/s²。

基本上要完成了---只需要再修改一下 HelloWorldLayer 类。首先打开 HelloWorldLayer.h，然后添加一个新的实例变量：

```
BOOL _tapDown;
```

同时修改 HelloWorldLayer.mm：

```
// Add at the top of the update method
if (_tapDown) {
if (!_hero.awake) {
[_hero wake];
_tapDown = NO;
} else {
[_hero dive];
}
}
[_hero limitVelocity];

// Replace ccTouchesBegan with the following
- (void)ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
[self genBackground];
_tapDown = YES;
}

// Add new methods
-(void)ccTouchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
_tapDown = NO;
}

- (void)ccTouchesCancelled:(NSSet *)touches withEvent:(UIEvent *)even
t {
_tapDown = NO;
}
```

编译并运行，现在你有一只可以飞的海豹啦！

修正海豹身体的摇晃

你可能注意到了，当海豹往下飞的时候，身体摇摇晃晃的。

一种方式就是，使用之前的线性速度和现在得到的速度作加权平均。

让我们来实现一下。先打开 Hero.h：

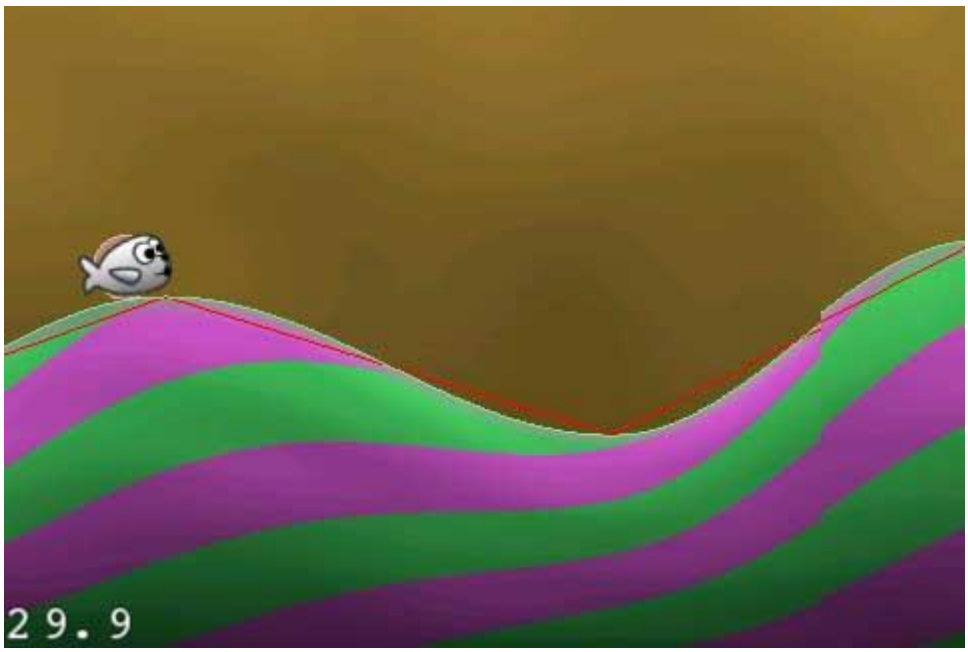
```
// Add to top of file
#define NUM_PREV_VELS 5

// Add inside @interface
b2Vec2 _prevVels[NUM_PREV_VELS];
int _nextVel;
```

然后修改 Hero.mm 的 update 方法：

```
- (void)update {

self.position = ccp(_body->GetPosition().x*PTM_RATIO, _body->GetPosit
ion().y*PTM_RATIO);
```

```
b2Vec2 vel = _body->GetLinearVelocity();
b2Vec2 weightedVel = vel;

for(int i = 0; i < NUM_PREV_VELS; ++i) {
weightedVel += _prevVels[i];
}
weightedVel = b2Vec2(weightedVel.x/NUM_PREV_VELS, weightedVel.y/NUM_P
REV_VELS);
_prevVels[_nextVel++] = vel;
if (_nextVel >= NUM_PREV_VELS) _nextVel = 0;

float angle = ccpToAngle(ccp(weightedVel.x, weightedVel.y));
if (_awake) {
self.rotation = -1 * CC_RADIANS_TO_DEGREES(angle);
}
}
```

这里使用之前的 5 个线性速度作加权平均，然后使用平均值来修正海豹的旋转。编译并运行，现在你可以看到更加平滑的海豹啦！



缩小

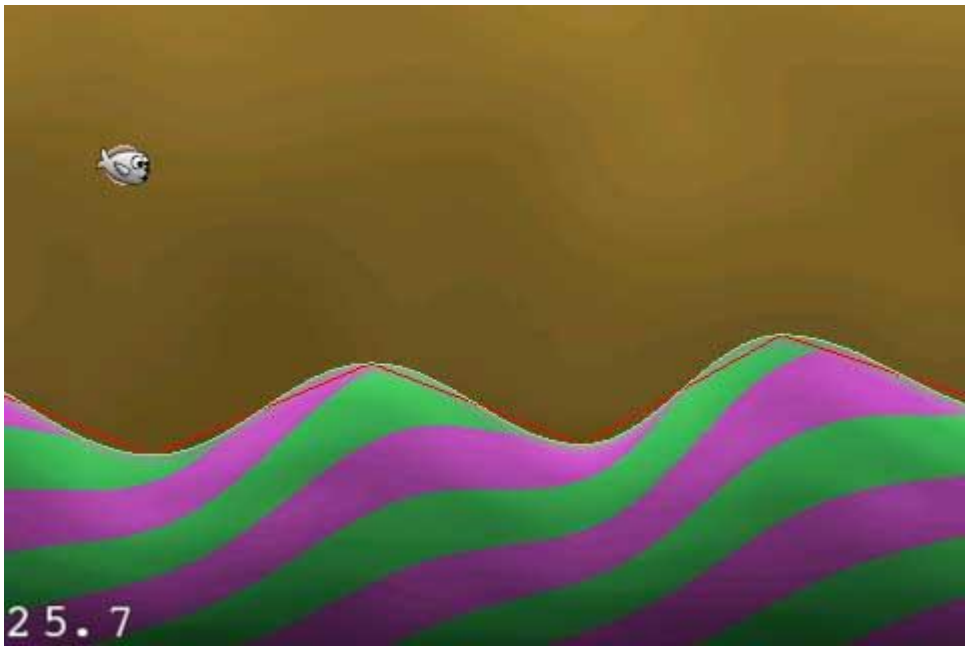
Tiny Wings 有一个很酷的特性就是，你飞得越高，那么屏幕就会越小。这使得视觉感观更加逼真！

为了实现这个，我们只需要在 HelloWorldLayer.mm 的 update 方法里面的[_hero update]调用之后，再添加下面代码就行了：

```
CGSize winSize = [CCDirector sharedDirector].winSize;
float scale = (winSize.height*3/4) / _hero.position.y;
if (scale > 1) scale = 1;
_terrain.scale = scale;
```

如果 hero 在 winSize.height*3/4 以下，那么 scale 就为 1.如果它大于 winSize.height*3/4，那么 scale 就会小于 1，就会有缩小的感觉了。

编译并运行，现在看看你能飞多高吧！



免费的动画和音乐

你懂的，我不能让你们这些粉丝没有一些免费的动画和音乐可玩。：）

只需要花上几秒钟的时间就可以使游戏变得更有趣！首先，打开 Hero.h，并作如下修改：

```
// Add inside @interface
CCAnimation *_normalAnim;
CCAnimate *_normalAnimate;

// Add after @interface
- (void)nodive;
- (void)runForceAnimation;
- (void)runNormalAnimation;
```

这里声明我们即将创建的动画，还有一个新方法将在海豹没有 diving 的时候被调用。

接下来，修改 Hero.mm：

```
// Add new methods
- (void)runNormalAnimation {
if (_normalAnimate || !_awake) return;
_normalAnimate = [CCRepeatForever actionWithAction:[CCAnimate actionW
ithAnimation:_normalAnim]];
[self runAction:_normalAnimate];
}

- (void)runForceAnimation {
[_normalAnimate stop];
_normalAnimate = nil;
[self setDisplayFrame:[[CCSpriteFrameCache sharedSpriteFrameCache] sp
riteFrameByName:@"seal_downhill.png"]];
}

- (void)nodive {
[self runNormalAnimation];
}

// Add at bottom of initWithWorld:
_normalAnim = [[CCAnimation alloc] init];
[_normalAnim addFrame:[[CCSpriteFrameCache sharedSpriteFrameCache] sp
```

```
riteFrameByName:@"seal1.png"]];
[_normalAnim addFrame:[[CCSpriteFrameCache sharedSpriteFrameCache] sp
riteFrameByName:@"seal2.png"]];
_normalAnim.delay = 0.1;
```

这里为海豹的正常飞行创建了动画效果，同时添加一个方法来播放这个动画。diving 动画实际上只是一个精灵帧，因此我们添加了一个辅助方法来完成 divng 动画的播放。

最后，我们修改一下 HelloWorldLayer.mm：

```
// At top of file
#import "SimpleAudioEngine.h"

// At end of init
[[SimpleAudioEngine sharedEngine] playBackgroundMusic:@"TinySeal.caf"
 loop:YES];

// At start of update, add an else case for the if (_tapDown):
else {
[_hero nodive];
}

// Inside ccTouchesBegan
[_hero runForceAnimation];

// Inside ccTouchesEnded AND ccTouchesCancelled
[_hero runNormalAnimation];
```

最后，打开 Terrian.mm，注释掉 draw 方法里面的_world->DrawDebugData。

编译并运行代码，大功造成了！

何去何从?

这里有本系列教程的全部源代码。

到目前为止，你有一个基本的游戏框架可以玩了。为什么不尝试着完善这个游戏呢？把海豹移动的行为修改得更加逼真、更加平滑一些？或者，你可以添加一些物品和道具，充分发挥你的想象力吧！

如果你扩展了本项目，不妨拿出来分享一下，大家一起学习一下吧！

PS：译者水平有限，翻译不准的地方望不吝指出，谢谢！