关于物理引擎的一些要点

cocos2d-x 3.0后将物理进行了新的封装，使得其应用更为简洁

> cocos2d-x 3.0+中的物理属性：
>
> 1、物理世界被集成到场景中，当你创建一个场景，你可以直接创建基于物理世界或者不使用物理世界的场景。
>
> 2、Node拥有它自己的body属性。（sprite也是node）‘
>
> 3、cocos2d-x 3.0 已经封装了物理属性 Body(PhysicsBody),Shape(PhysicsShape),Contact(PhysicsContact),Joint(PhysicsJoint)和 World(PhysicsWorld),更加方便使用。
>
> 4、方便的使用listener-EventListenerPhysicsContact进行碰撞检测。

物理世界的创立

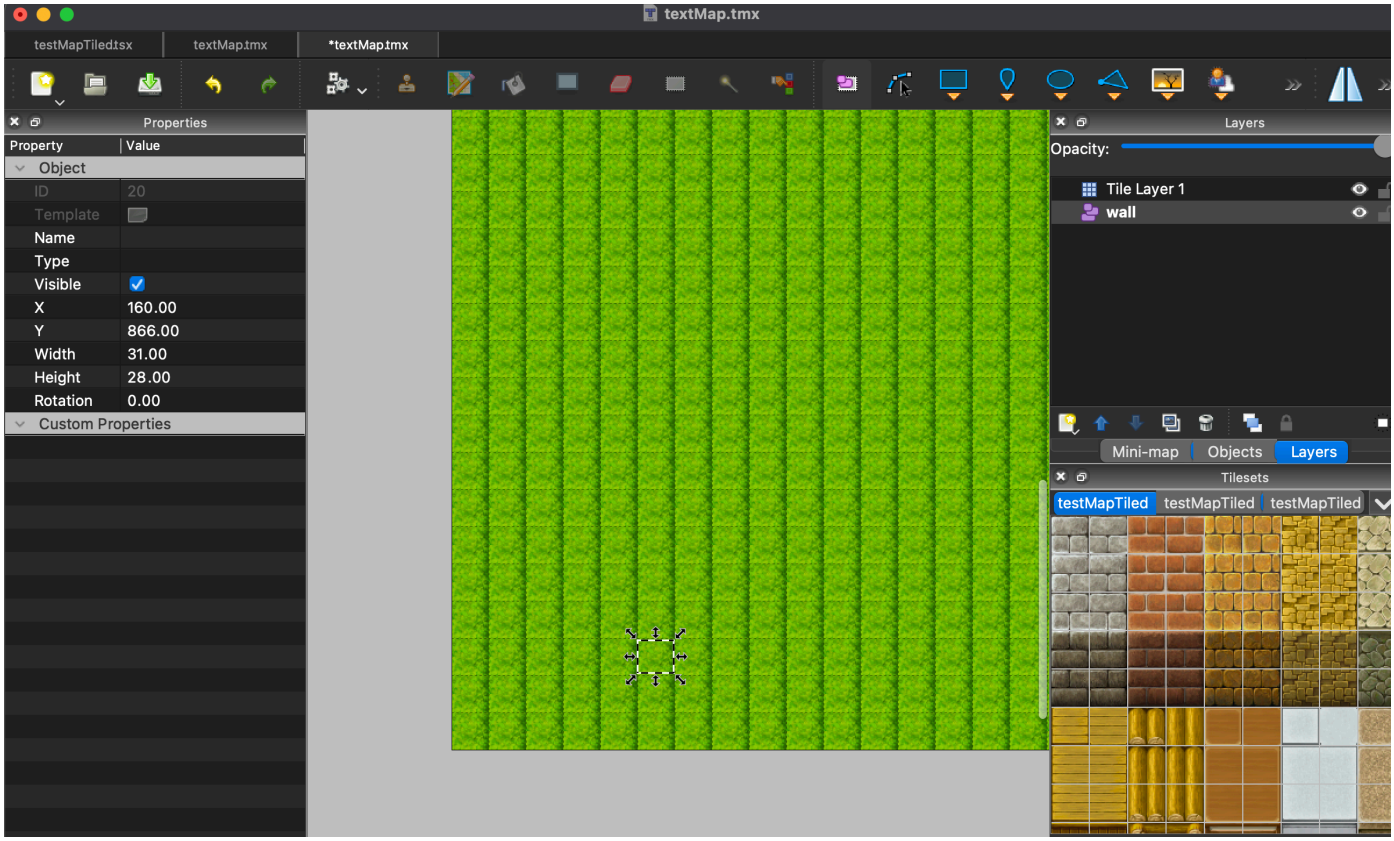
```
Scene* HelloWorld::createScene()
{
    auto scene = Scene::createWithPhysics();
    auto layer = HelloWorld::create();
    scene->addChild(layer);
  //设置重力
    scene->getPhysicsWorld()->setGravity(Vec2(0,-10.0f));
    return scene;
}
```

对于每一个物理世界，都必须先设置其重力，图示代码中setGravity(Vec2(0,-10.0f)即设置一个向下的大小为10g的重力加速度，由于吃鸡为俯视地图，所以在真正设置时需设为(0,0)，本周代码中为体现其效果方设为(0,-10)。

```
auto body = PhysicsBody::createEdgeBox(visibleSize, PHYSICSBODY_MATERIAL_DEFAULT, 3);//设置要添加到节点中的物理body
    auto edgeNode = Node::create();
    edgeNode->setPosition(Point(visibleSize.width/2,visibleSize.height/2));
    edgeNode->setPhysicsBody(body);//将物理body加入到创建的节点中
    this->addChild(edgeNode);//场景中添加创建的物理节点
```

不用太在意的代码，就是给可视窗口四周加上了物理边界

创建障碍物（位置由tiled地图的对象层确定）



右栏中wall即是为一类障碍物所建立的对象层，对象层的对象没有对应的图像，左栏为对象层中某个对象（草丛中箭头所指的一格）的属性。

下面的代码将根据对象层中的对象创建带有**静态刚体属性**的障碍物

```
    //根据对象层名称“wall”获得该对象层
    auto groupwall = testMap->getObjectGroup("wall");
    //获得对象层中的所有对象
    ValueVector objectsWall = groupwall->getObjects();


    for (auto wall : objectsWall)
    {
        ValueMap& dict = wall.asValueMap();

        float x = dict["x"].asFloat();
        float y = dict["y"].asFloat();
        float width = dict["width"].asFloat();
        float height = dict["height"].asFloat();
        float rotate = dict["rotation"].asFloat();
//创建一个PhysicsBody并以对象的形状设置其形状
        PhysicsBody* tmpPhysicsBody = PhysicsBody::createBox(Size(width, height));
        tmpPhysicsBody->setDynamic(false);
      //设置接触掩饰值（之后会具体谈）
        tmpPhysicsBody->setCategoryBitmask(1);
        tmpPhysicsBody->setCollisionBitmask(1);
```

```
        tmpPhysicsBody->setContactTestBitmask(1);

        Sprite* tmpSprite = Sprite::create("HelloWorld.png");
        tmpSprite->setPosition(Vec2(x, y));
        tmpSprite->setAnchorPoint(Vec2::ZERO);
        tmpSprite->setContentSize(Size(width, height));
      //把上面的tmpPhysicsBody设置为Sprite的PhysicsBody属性
        tmpSprite->setPhysicsBody(tmpPhysicsBody);

        addChild(tmpSprite,1,5);
    }
```

静态刚体没有速度，不受力的作用，但会与动态刚体产生碰撞

动态刚体的创建

创建方法与静态刚体大致相同，密度使用默认值

```
PhysicsBody* tmpPhysicsBody = PhysicsBody::createBox(Size(40, 40));
    tmpPhysicsBody->setDynamic(true);
    tmpPhysicsBody->setCategoryBitmask(1);
    tmpPhysicsBody->setCollisionBitmask(1);
    tmpPhysicsBody->setContactTestBitmask(1);

    Sprite* tmpSprite = Sprite::create("sprite1.png");
    tmpSprite->setPosition(135,250);
    tmpSprite->setContentSize(Size(40, 40));
    tmpSprite->setAnchorPoint(Vec2::ZERO);
    tmpSprite->setPhysicsBody(tmpPhysicsBody);

    addChild(tmpSprite,1,3);
```

物理碰撞监听

下面代码注册碰撞响应事件和回调函数

```
auto contactListener = EventListenerPhysicsContact::create();
contactListener->onContactBegin = CC_CALLBACK_1(HelloWorld::onContactBegin, this);
_eventDispatcher->addEventListenerWithSceneGraphPriority(contactListener, this);
```

> 每一次碰撞检测事件是有EventListenerPhysicsContact来进行监听的。监听到碰撞事件时，会回调响应事件onContactBegin()来进行碰撞事件的处理。_eventDispatcher是事件派发器，由它管理所有的注册事件。

我们可以在onContactBegin通过条件判断来处理不同物体碰撞的差异

以下是本周代码中的onContactBegin函数，效果是碰撞后二者都得到1000倍的相反方向速度

```
 bool onContactBegin(PhysicsContact& contact)
```

```
    {
        auto nodeA = contact.getShapeA()->getBody()->getNode();
        auto nodeB = contact.getShapeB()->getBody()->getNode();

        log("=========contact happen=========");

        auto va = nodeA->getPhysicsBody()->getVelocity();
        va.normalize();
        nodeA->getPhysicsBody()->setVelocity(-1000 * va);
        auto vb = nodeB->getPhysicsBody()->getVelocity();
        vb.normalize();
        nodeB->getPhysicsBody()->setVelocity(-1000 * vb);
        return true;
    }
```

接触掩饰码

关于接触掩饰码，我直接应用CSDN上的一段解释

> 在上面说了这么多的东西，最重要的东西就是下面的，没有下面的东西，碰撞事件根本不起作用，这就是我第一次运用碰撞时遇到的问题。也就是设置物理接触相关的位掩码值，默认的接触事件不会被接受，需要设置一定的掩码值来使接触事件响应。
> 接触掩码值有三个值，分别是：
> 1、CategoryBitmask，默认值为0xFFFFFFFF
> 2、ContactTestBitmask,默认值为 0x00000000
> 3、CollisionBitmask，默认值为0xFFFFFFFF
> 这三个掩码值都有对应的set/get方法来设置和获取。
> 这三个掩码值由逻辑与来进行操作测试。
>
> 一个body的CategoryBitmask和另一个body的ContactTestBitmask的逻辑与的结果不等于0时，接触事件将被发出，否则不发送。
>
> 一个body的CategoryBitmask和另一个body的CollisionBitmask的逻辑与结果不等于0时，他们将碰撞，否则不碰撞
>
> 默认情况下的body属性会进行物理碰撞，但不会发送碰撞检测的信号，也就不会响应碰撞回调函数，这个可以看下默认情况下的掩码值的逻辑与

```
CategoryBitmask = 0xFFFFFFFF;
ContactTestBitmask = 0x00000000;
CategoryBitmask & ContactTestBitmask = 0//所以不会发送碰撞信号

CollisionBitmask = 0xFFFFFFFF;

CategoryBitmask & CollisionBitmask = 0xFFFFFFFF
//所以物体会碰撞，但是不会响应碰撞回调函数。
```

> 上面介绍的掩码值是碰撞检测回调中最重要的，没有上面的掩码值，所有的碰撞回调函数都不会发生。
> EventListenerPhysicsContact有四个接触回调函数：
> 1、onContactBegin,在接触开始时被调用，仅调用一次，通过放回true或者false来决定两个物体是否有碰撞。同时可以使用PhysicsContact::setData()来设置接触操作的用户数据。当返回false时，onContactPreSolve和onContactPostSolve将不会被调用，但是onContactSeperate将被调用一次。
> 2、onContactPreSlove ，会在每一次被调用，通过放回true或者false来决定两个物体是否有碰撞，同样可以用ignore()来跳过后续的onContactPreSolve和onContactPostSolve回调函数。(默认返回true)
> 3、onContactPostSolve，在两个物体碰撞反应中的每个步骤中被处理调用。可以在里面做一些后续的接触操作。如销毁body
> 4、onContactSeperate，在两个物体分开时被调用，在每次接触时只调用一次，和onContactBegin配对使用。
> 上述中最重要的就是碰撞检测事件的讲解，这是游戏中用到碰撞经常要用到的。

本周代码的一些说明

下图为代码的运行效果，主要包括：5个静态刚体障碍物，1个不能移动的静态刚体与1个可以移动的hero（也是动态刚体）

主要要说明的点在于hero类，我在hero的构造函数中加入了动态刚体的创立与绑定

```
hero::hero()
{
    currentMovingState = ms_standing;
    auto* dispatcher = Director::getInstance()->getEventDispatcher();
    auto* keyListener = EventListenerKeyboard::create();
    scheduleUpdate();
```

```
    bindSprite(Sprite::create("sprite1.png"));
    if (delegateSprite == nullptr)
    {
        log("==============failed to create sprite for hero====================== ");
    }
    //设置hero的物理模型
    delegateSprite->setPosition(Point(100, 200));
    PhysicsBody* heroPhysicsBody = PhysicsBody::createBox(Size(this-
>getContentSize()));
    heroPhysicsBody->setDynamic(true);
    heroPhysicsBody->setCategoryBitmask(1);
    heroPhysicsBody->setCollisionBitmask(1);
    heroPhysicsBody->setContactTestBitmask(1);

    this->delegateSprite->setPhysicsBody(heroPhysicsBody);
}
```

然后原本的移动函数无法在碰撞检测中生效，所以我将键盘响应改成了设置物理对象的速度

```
void hero::onKeyPressed(cocos2d::EventKeyboard::KeyCode keycode, cocos2d::Event* event)
{
    switch (keycode)
    {
        case EventKeyboard::KeyCode::KEY_W:
            delegateSprite->getPhysicsBody()->setVelocity(Vec2(0,100));
            break;
        case EventKeyboard::KeyCode::KEY_S:
            delegateSprite->getPhysicsBody()->setVelocity(Vec2(0,-100));
            break;
        case EventKeyboard::KeyCode::KEY_A:
            delegateSprite->getPhysicsBody()->setVelocity(Vec2(-100,0));
            break;
        case EventKeyboard::KeyCode::KEY_D:
            delegateSprite->getPhysicsBody()->setVelocity(Vec2(100,0));
            break;
        default:
            break;
    }
}
```

```
void hero::onKeyReleased(cocos2d::EventKeyboard::KeyCode keycode, cocos2d::Event*
event)
{
    log("key %d up", keycode);
    delegateSprite->getPhysicsBody()->setVelocity(Vec2(0,0));
}
```

但由于我对这一方面的不熟悉，所以还要麻烦相关同学帮忙完善，麻烦了🙏🙏

tip：1.虽然是1000倍速度，不过由于默认相关系数的存在，实际上并没有那么离谱，但有时由于子弹效应（速度过大），物体会穿透边界。

2.代码可能有点乱，会于这两天整理一下