基于 FLASH BOX2D 物理引擎的游戏框架设计与实现

摘要：随着 FLASH 游戏的不断纵深发展，具有物理引擎的 FLASH 游戏开发正在逐渐成为一个流行的发展趋势。BOX2D 作为一款性能优越的开源物理引擎，也移植到了 FLASH 平台。然而直接应用 FLASH BOX2D 引擎模拟物理特性，无论在开发的时间成本，还是在开发的难度上，都遇到了不小的挑战。本文将通过对 FLASH BOX2D 物理引擎的深入分析，针对多种需求的物理碰撞盒绘制，物理与 UI 绑定等问题，提出了整合优化的方法。运用组件封装技术，设计与实现了一套快速，灵活，高效的基于 FLASH BOX2D 的游戏开发框架。最后通过实例，进一步验证了游戏框架的实际应用价值

关键词：FLASH，BOX2D，物理碰撞，开发框架

Design and implementation of mobile game development framework based on FLASH

***

（School of Software, Shanghai Jiao Tong University, Shanghai,China）

Abstract：With the continuous development of the FLASH game, with physics engine Flash game development is becoming a more popular trends. However application BOX2D engine directly, in terms of development costs is difficult, there are a lot of constraints, this article by Flash Box2D physics engine game technology analysis, for physical collision problem difficult to achieve, the integration and optimization of the method, and finally the use of GPU acceleration technology, design and implementation of a fast, flexible, high-performance Flash Box2D game development framework.

Key words: FLASH, BOX2D, Physical collision, Development Framework

第一章 绪论

随着网络游戏的发展，FLASH 页游的市场份额越来越大，大有取代客户端游戏的趋势。而随着移动设备性能的提升，移动游戏充份融合网页游戏与客户端游戏的优势，实现双端融合，更是有着难以想像的发展空间。在移动游戏中，应用 BOX2D 物理引擎开发的 FLASH 游戏，源于其全新的操控体验，受到了越来越多玩家的肯定，《愤怒的小鸟》就是成功的经典。

在开发带物理引擎的 FLASH 游戏中，FLASH 常用的引擎主要是 BOX2D。应用 FLASH BOX2D 构建物理世界的过程是相当枯燥的，需要程序员与设计师的长时间沟通与协作，在不断磨合中，才有可能达到精确、合理的实现游戏物理世界的目标。如何简化物理游戏开发，优化碰撞盒数据自动采集，实现物理与可见 UI 之间的精确绑定，是物理游戏成功构建的关键。

本文将通过对物理基本形状的构建分析，逐步延伸到对不规则物体的模拟，并最终实现复杂物体物理碰撞的搭建。在此基础上，分析设计、实现物理碰撞盒的图形化绘制。最后应用面向对象的设计方法与相关设计模式，对 FLASH BOX2D 物理引擎进行二次封装，实现物理碰撞盒与 UI 的快速绑定，最终整合成一套快速开发 BOX2D 物理游戏的框架。

第二章 FLASH BOX2D 的技术分析

Box2D 最初是一个用于模拟 2D 刚体的 C++引擎，开发人员可以应用 Box2D 引擎，使物体在运动时拥有物理效果，像惯性、自由落体、碰撞等等。因为其出色的性能和丰富的功能，后来被开发者移植到了 FLASH 平台，便有了 FLASH Box2D 物理引擎。

Box2D 是模拟刚体的物理引擎，刚体的具有不同的形状，比如圆形，矩形，多边形。这些刚体在物理世界中创建后受到重力，摩擦力的影响，运行过程中可以施加相应的作用力。

2.1 核心概念

世界(world)：物理世界就是模拟物理的一个集合，物体由世界创建，在这个集合中的物体受到共同的特理特性，比如重力。在游戏中可以创建多个世界，但通常是不需要的。

刚体(rigid body)：刚体可以理解为一个十分坚硬的物体，其中的任何两点之间的相对距离都是完全不变的。Box2D 世界中可以移动的或交互的对象都是刚体。

形状(shape)：形状是各种二维几何体，例如圆形和多边形。Box2D 只支持凸多边形。

夹具(fixture)：其概念源于机械工程，在机械制造过程中用来固定加工对象，使之占有正确的位置，以接受施工或检测的装置。Box2D 中的夹具用来将形状绑定到物体上，并添加密度(density)、摩擦(friction)和弹力(restitution)等物理属性。

2.2 在 Box2DFlash 中定义世界

基于 Box2D 的游戏开发，第一步就是要创建物理世界。与现实世界一样，Box2D 世界（World）有重力（gravity），所以第一步先定义世界重力（world gravity）。

```
var gravity:b2Vec2=new b2Vec2(0,9.8);
```

b2Vec2 是一个 2D 的向量数据类型，它将储存 x 和 y 矢量分量。b2Vec2 的构造函数有两个参数，都是数值，代表了 x 和 y 分量。通过这种方法定义 gravity 变量作为一个矢量，它有 x=0（这意味着水平的重力）和 y=9.8（这意味着近似的地球重力）。

当在世界中的刚体静止时，可以允许他们进入睡眠状态，这样它们将不受作用力的影响。一个睡眠的刚体无需模拟，它只是表示自己的存在，并静止在它的位置上，不会对世界中的任何事物产生影响，允许 Box2D 忽略它，而且因此会提升处理速度以及让我们获得更好的性能。所以我们可以定义变量在可能时让刚体睡眠。

```
var sleep:Boolean=true;
```

最后，我们要定义的世界通过下面这句话来创建：

var world:b2World = new b2World(gravity,sleep);

2.3 在世界中加入刚体

我们通常使用以下步骤将刚体添加到世界中：

- 创建一个刚体定义，它将持有刚体的信息，例如刚体的位置信息。
- 创建一个形状，它将决定刚体的显示形状。
- 创建一个夹具，将形状附加到刚体定义上。
- 创建刚体在世界中的实体，使用夹具。

无论我们是创建圆形还是多边形，第一步都是创建一个刚体：

var bodyDef:b2BodyDef=new b2BodyDef();

b2BodyDef 类是一个刚体的定义类，它将持有创建我们刚体所需要的所有数据。

现在可以将刚体添加到世界中：

bodyDef.position.Set(x,y);

通过 position 属性显示的设置了刚体在世界中的位置，注意，这里的单位。虽然 Flash 是以像素（pixels）为度量单位，但是在 Box2D 中尝试模拟真实的世界并采用米（meters）作为度量单位。对于米（meters）和像素（pixels）之间的转换没有通用的标准，但是我们采用下面的转换标准可以有很好的运行效果：

1meter= 30Pixels

所以，如果我们定义一个变量来帮助我们将米（meters）转换成像素（pixels），我们便可以在 Box2D 世界（world）中进行操作时使用像素（pixels）而不用使用米（meters）来作为度量单位。这样将使我们在制作 Flash 游戏时，使用像素来思考，从而变得更加直观。

2.4 形状

形状（shape）是一个 2D 几何对象，例如一个圆形或者多边形，在这里必须是凸多边形（每一个内角小于 180 度）。Box2D 只能处理凸多边形。

如果要创建一个圆形，我们可以通过以下代码实现：

var circleShape:b2CircleShape;

circleShape=new b2CircleShape(25/30);

b2CircleShape 是用来创建圆形形状，并且它的构造函数需要一个半径（radius）作为参数。在之前的代码中，我们创建了一个圆形，它的半径为 25 像素（pixels）。

因为物理世界是以“米”来作为度量单位的，所以这里需要把“像素”转换成“米”，通常我们可以取 30 作为转换比例。

我们也可以设置一个常量 worldScale 来保存这个比例，以便在其他地方能够方便的访问。从现在起，每次想要使用像素进行操作时，只要将它们除以 worldScale 即可。当然也可以定义一个方法名为 pixelsToMeters 的方法，在每次你需要将像素（pixels）转换成米（meters）时调用。

当我们有了刚体定义和形状时，我们将使用夹具（fixture）来将它们粘合起来。

2.5 夹具

夹具（fixture）用于将形状绑定到刚体上，然后定义它的材质，设置密度（density），摩擦系数（friction）以及恢复系数（restitution）。

首先，我们创建夹具（fixture）：

```
var fixtureDef:b2FixtureDef = new b2FixtureDef();
fixtureDef.shape=circleShape;
```

一旦我们通过构造函数创建了夹具（fixture），我们将分配之前创建的形状给它的shape属性。

最后，我们可以将球添加到世界中：

```
var theBall:b2Body=world.CreateBody(bodyDef);
theBall.CreateFixture(fixtureDef);
```

b2Body是刚体的实体：是物质，是通过使用bodyDef属性创建的具体刚体。

2.6 小结

通过对 Flash Box2D 的核心概念的了解，以及物理世界的初始化，物理刚体的创建，可以说我们大体上能够创建基本的基于 Box2D 引擎的游戏了。

但是我们也不难发现，基本上每次开发一个物理游戏，都要重复物理世界的初始化。另外对于游戏中的每一个物理刚体，如果我们都按照上面的步骤去创建的话，也是相当繁琐的过程。

因此，下面我们就需要去具体分析一下，看看我们的游戏框架怎样设计，才能够使我们开发物理游戏既方便实用，又性能卓越。

第三章 游戏框架需求分析

我们设计的这个框架是一个轻量级的，简单易用的基于 Box2D 的框架，因些取名 Slight，以下描述也会中 Slight 就是我们框架的名字。

3.1 物理世界的初始化分析

通过上面对 Flash Box2D 的技术分析，我们可以看到，在每次开发一款游戏的时候，都会需要对物理世界进行初始化，而这一过程在很大程序上是相似的，这是我们可以抽离出来，设计在我们游戏框架中，这样就便于每次开发时应用了。

3.2 物理刚体的创建

通常我们创建物理游戏都是要创建一个可见对象，然后为这个可见对象创建一个互之相匹配的物理对象。而创建物理对象又需要经历一套复杂的流程：

- 创建 Body definition,
- 创建形状
- 创建 Fixture definition
- 通过世界创建 Body
- 通过 Body 创建 Fixture

通过普通的流程来创建物理游戏会遇到以下几类问题

- 每次开发物理游戏都需要对物理世界做一次初始化，这对于面象对象的开发来讲，重复相同的代码是不合理的。
- 物件比较多，创建这些物体是费时费力的。
- 不光是要创建出形状相匹配的物理对象，而且质心需要与可见对象相吻合。
- 不光是要创建矩形，圆形，凸多边形等 Box2d 原生支持的形状，还会有很多不规则物体，比如凹多边形。
- 通过世界创建出来的 Body 通常由世界保存，我们之后如果想通过显示对象来访问 Body 的话就需要遍历世界，这显然是个不合理的开销。

这些都给开发人员提出了不小的挑战，也是我们的框架需要解决的问题。

第四章 Slight 框架实现

通过对 Box2D 的深入分析，我们已经找到了创建 Box2D 物理游戏的一些问题，下面我们就来具体实现解决这些问题的方法。

4.1 世界封装类

世界封装类是进入物理世界的入口， 主要是为了简化游戏开展中每次都要手动去初始化物理世界的过程，我们设计了 GameWorld 类来具体实现。

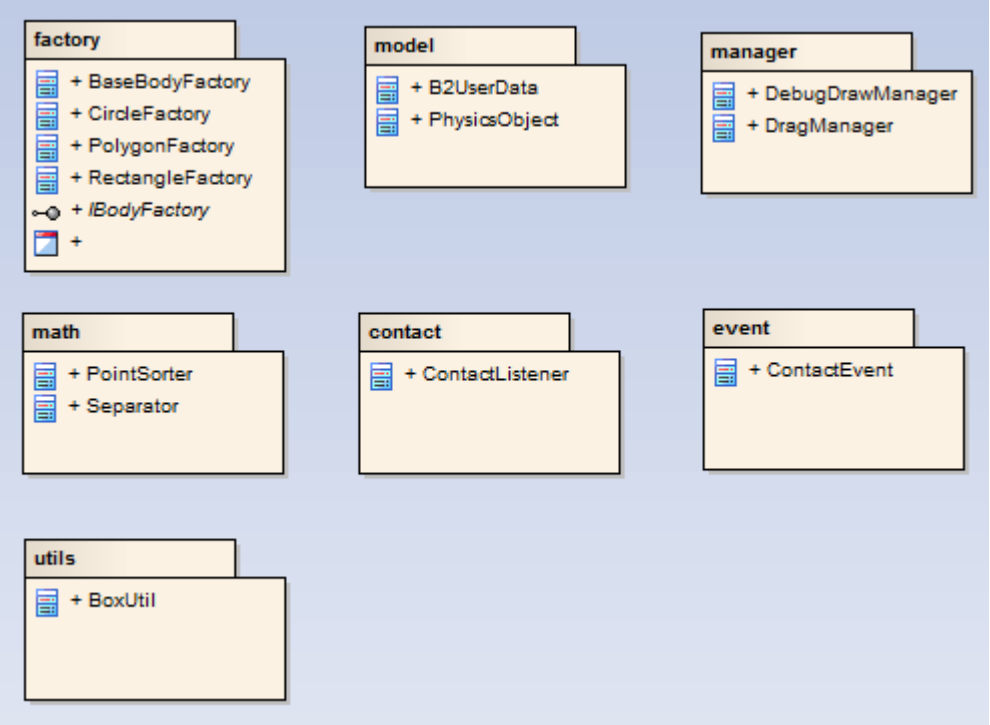
GameWorld 类是继承 b2World 的，并且是单例类，这样能保证了游戏中始终只有一个世界。因此，我们不需要去手动初始化世界了，只要在需要使用 b2world 的地方，通过 GameWorld.getInstance() 方法。

下面是 GameWorld 的类图

我们在这个类中初始化了世界，提供了相关属性的设置，比如 Gravity 等。并且提供了通过显示对象来快速获取物理对象的公开方法 getBodyByDO(displayObject:*):b2body，这极大的方便了对物理对像的调用。

4.2 Slight 框架结构设计

我们通过 GameWorld 类来实现对物理世界的封装，这只是我们框架的开始，下面我们要来实现对物理刚体创建的框架实现，我们先来看下框架的包结构：

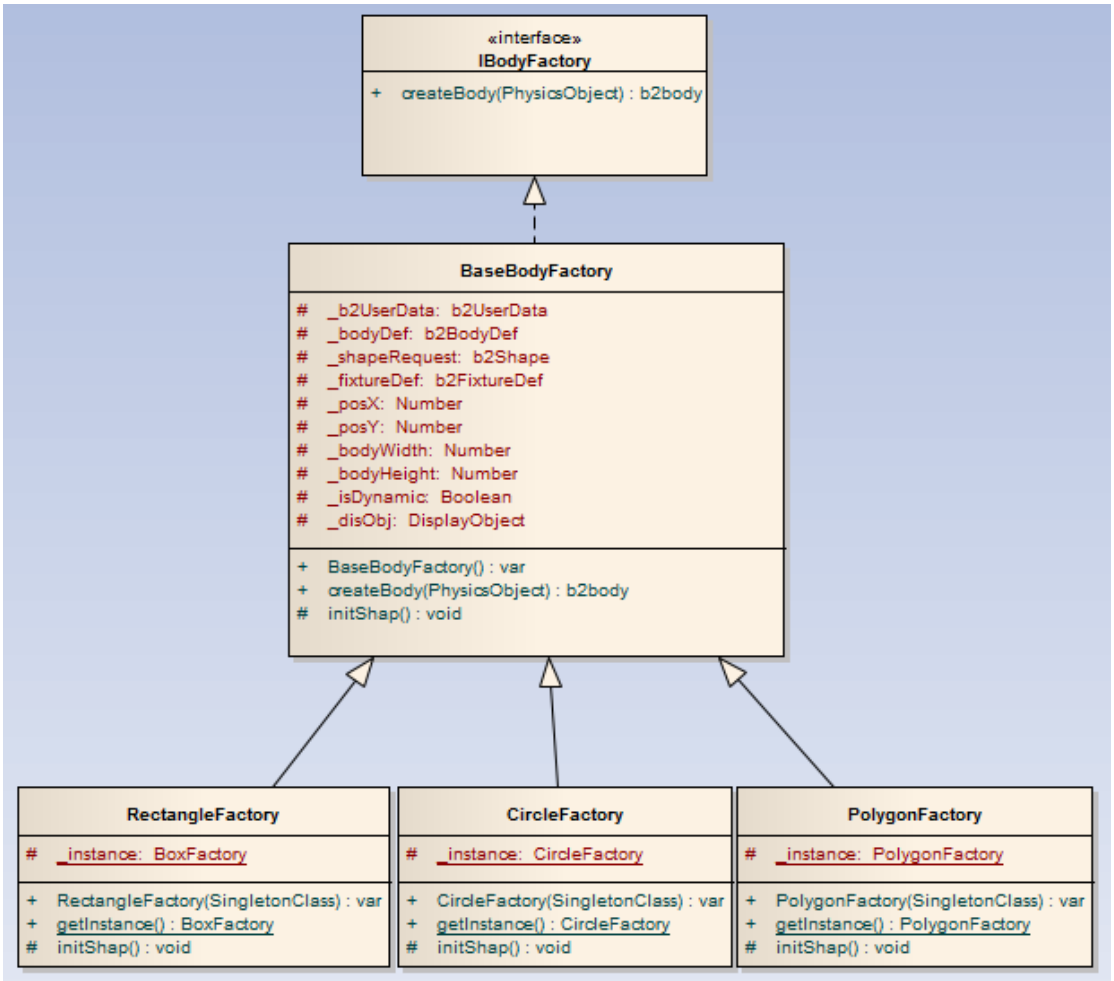


我们主要是应用工厂模式来实现对物理刚体的创建，model 包主要是物理数据的格式，manager 包提供了相对独立功能的管理类，math 包将独立处理一些几何图形相关的数据，contact 和 event 包为以后实现自定义的碰撞方案预留了接口，utils 包主要是提供一些静态的工具类方法，以方便我们开发中使用。

下面我们具体的叙述一下每个包的功能与概念

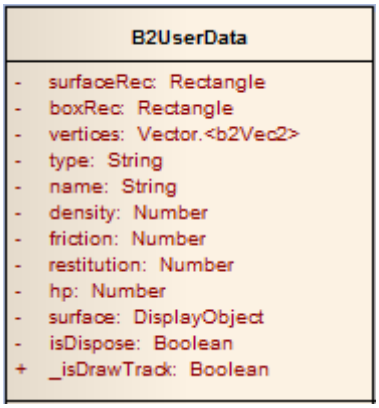
4.2.1 Factory 包

Factory 是方便创建物理对象的工厂，主要包括基本形的创建和任意多边形的创建，为了在将来的开发中能更好的扩展， 这里采用了工厂模式的设计方法，如果以后有一些特殊类型的物件，也能很方便的通过继承 BaseBodyFactory 类来扩展，只要复写 initShap()方法来提供特殊的形状。



4.2.2 model 包

model 包主要处理数据，其中 B2UserData 是对各种类形物体的物理特性的数据描述，当然为了方便相关数据的存取，也加入了表示物体生命值的变量 hp， 以及可见对象的 Reference 引用



4.2.3 manage 包

manage 包主要提供对物理对象的管理

因为Box2D 的刚体是不可见的，这对于调试是个问题，为了能够使刚体可视化，我可需要设置成Debug模式，Box2D在Debug模式下会绘制线框图来表示刚体的形状和位置，具体

有6种显示选择：

- e_shapeBit　绘制形状
- e_jointBit　绘制关节连接
- e_aabbBit　绘制刚体的边界框线
- e_pairBit
- e_centerOfMassBit
- e_controllerBit

为了便于对调试模式的管理，我们设计了 DebugDrawManager 类，用来控制调试模式的开启与关闭，能及对各种调试数据的输出选择，这有助于我们方便的管理 Box2D 刚体的调试。

在物理游戏中鼠标拖拽功能也是非常基本，常用的功能，因些对这个功能进行了封装。DragManager 类提供了对物理对象的鼠标响应， 当鼠标点击的时候，自动获取在鼠标下的物体，并提供了拖拽的功能。

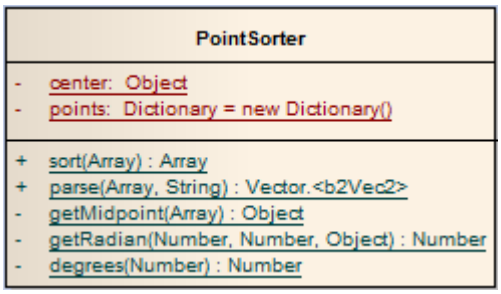
4.2.4　math 包

math 包主要提供数学几何相关的计算支持。
因为在 Box2d 只支持矩形，圆形和凸多边形的创建，而且对于凸多边形的顶点数据也有顺序的要求，必须是顺时针的顶点数组，这给物理形状的创建带来了很大的不便。

我们在项目中如果是凸多边形的创建，一般会使用 Flash cs6 工具

- 先将可见图形放在一个图层中作为参考对象
- 使用钢笔等绘画工具绘制多边形。使用钢笔工具的好处是可以方便的调整各个顶点的位置。
- 使用JSFL(Flash 提供的角本语言) 对图形顶点进行扫描并记录X与Y坐标，并输出成数据格式

这时我们可以使用凸多边形的顶点数据来绘制物理对象，但有个问题是使用 JSFL 输出的顶点数组顺序是乱的，这就带来了一个问题：直接使用 Box2d 原生方法不能正常绘制图形。当然我们可以手动的来记录这些凸多边形的顶点数据，从而形成相应顺序的顶点数据，但这将是一项非常耗时的工作，基于此，我们设计了了 PointSorter 类来处理顶点排序的问题，这将大大提供我们的工作效率。



那么对于凹多边形呢，因为凹多边形的顶点排序有着不唯一性，因些，我们不能使用上面的方法来处理，经过多方搜寻，我们得到了一个变相的解决方案，原理如下：将凹多边形切割成多个三角形或四边形，然后组合成凹多边形。

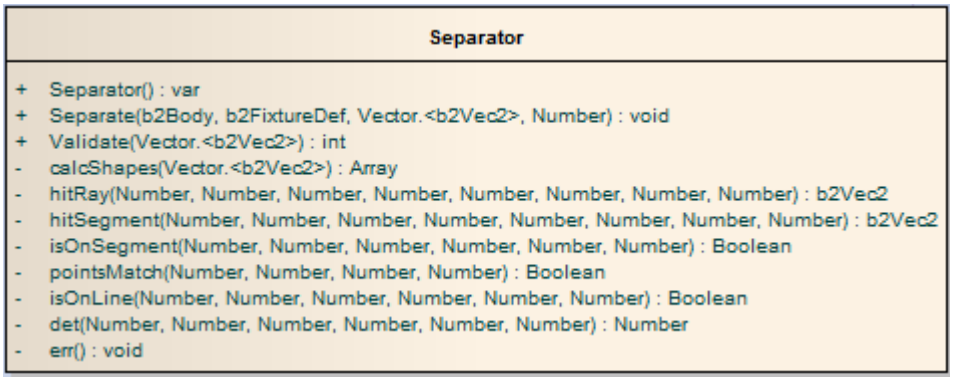
我们首先创建一个记录顶点的工具软件，为了快速创建这个工具软件，我们使用 Flash CS6,

首先导入参考图

然后实现鼠标移动记录顶点功能，进行多边形顶点采集。这个过程是根据相对时间来读到鼠标所在位子的坐标信息的。

最后结束绘制时按先后顺序保存顶点数据到一个数组，以便程序使用。

从开源社区获得一个工具类 Separator，我们这里采用这个工具类来帮助我们处理。



4.2.5 contact 包

contact 包主要是提供对碰撞的个性化处理。

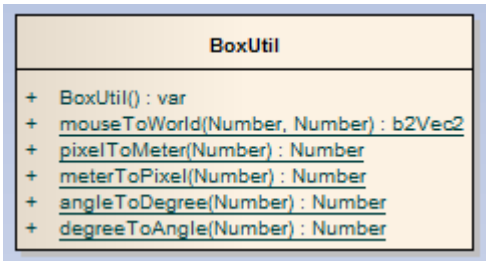
因为碰撞的处理具有很多的不确定性，对 hp 的伤害规则通常也比较复杂，所以我们的框架只是提供了碰撞的基本信息的提供，并以事件的形式向外抛出，在界面层去做个性化的处理。

4.2.6 event 包

event 包目前只是对碰撞事件的支持，以后将根据实际情况来调整

4.2.7 utils 包

utils 包提供了简化数据转换的一些工具类



4.3 Slight 框架的总结

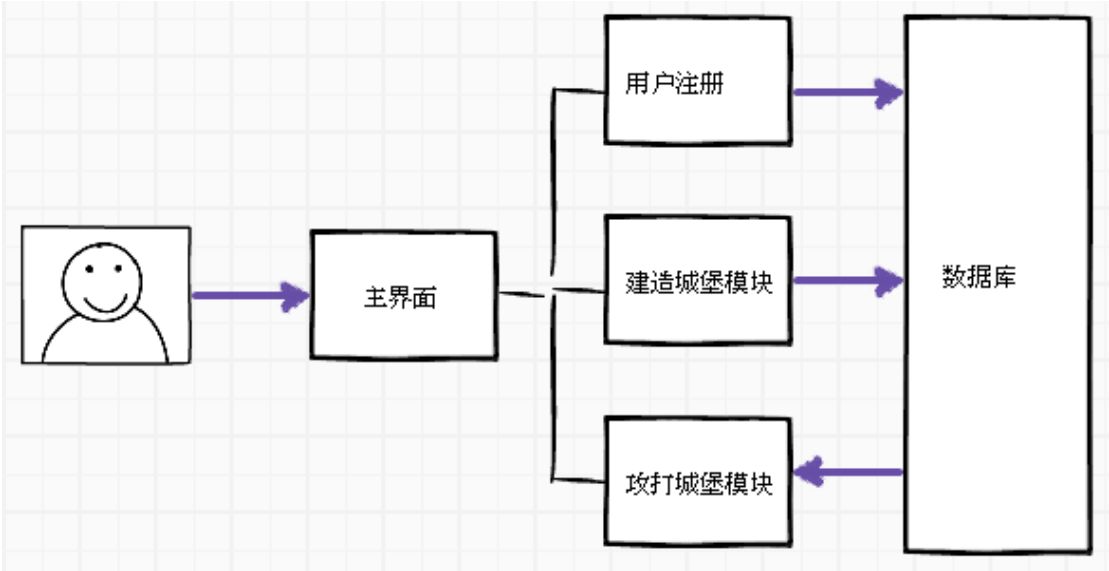
- GameWorld 实现了对物理世界的封装，减少了每次开发游戏的重复对物理世界的初始化操作。
- 用 Factory 实现所有 Body 的创建，只需要准备好相应物理属性，一步实现物理的创建。
- 在处理质心的问题时，默认显示对象的注册点在左上角，质心在整体矩形的 1/2 宽高处。
- 对凸凹多边形进行分类处理，找到了相应的解决方案。
- GameWorld 还提供了通过显示对象快速查找物理对象的公开方法，方便了对物理对象的控制。

第五章 基于 Slight 框架的物理游戏开发

5.1 游戏架构概述

我们开发的是一款基于 Slight 的 IOS 游戏， 游戏主要由三个模块组成：用户注册，建造城堡，攻打城堡。

游戏架构见下图：



用户注册和城堡建造主要是 Flash 编程的部份，只是对用户数据和城堡信息进行存储，以但在攻打城堡模块能够读取数据

攻打城堡模块是物理引擎来实现的，因些也是我们论述的重点

5.2 游戏中物理城堡的实现

5.2.1 物理城堡概述

物理城堡是用不同矩形块组成的， 我们对矩形块进行分类为石块，木块，屋顶以及城堡主。我们要搭建的城堡是为了保护城堡主的安全，城堡主的灭亡则视为城堡攻破。

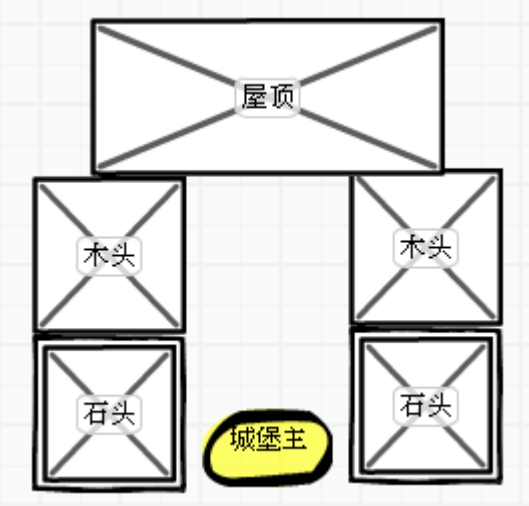
5.2.2 物理城堡的构建

针对不同的建筑块，会有不同的物理属性，为了便于测试以及扩展，我们将这些属性保存在相应的配置文件中

```
<?xml version="1.0" encoding="UTF-8"?>

<Boxdata>
    <box type="ground" density="80" friction="100" restitution="0" hp="0" damageOthers="1"/>
    <box type="hero" density="20" friction="50" restitution="0.1" hp="100" damageOthers="1"/>
    <box type="stone" density="75" friction="90" restitution="0" hp="25" damageOthers="2"/>
    <box type="roof" density="60" friction="90" restitution="0" hp="20" damageOthers="4"/>
    <box type="pillar" density="20" friction="20" restitution="0" hp="15" damageOthers="1"/>
    <box type="king" density="50" friction="50" restitution="0.1" hp="30" damageOthers="1"/>
</Boxdata>
```

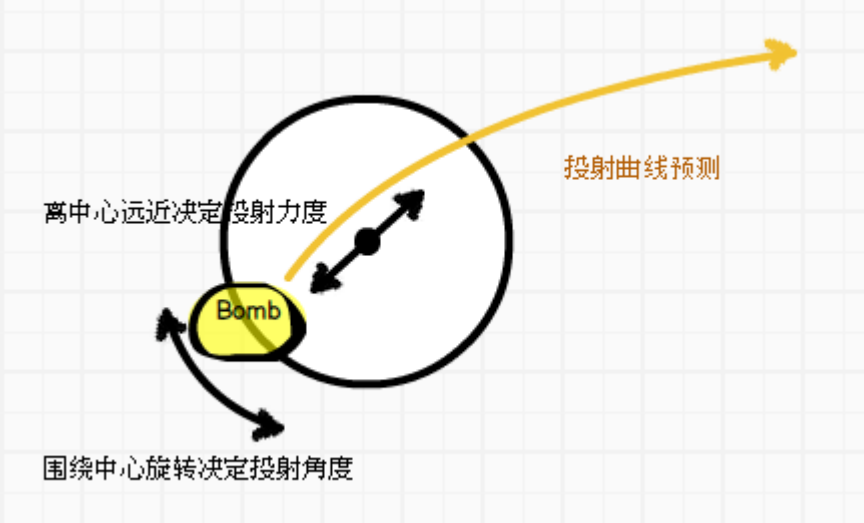
我们在进入攻城模块之后，会读取相应的存储于数据库中的城堡信息，然后根据每一个城堡建筑块的类型，去匹配相关的物理属性，最后根据建筑块的类型，调用框架中相应的工厂（Factory）去创建具有物理属性的建筑块



5.3 游戏中投弹攻打的实现

5.3.1 投弹攻打概述

投弹机制类似愤努的小鸟，投射物可以围绕圆中心内旋转，最后释放时的角度将决定投射的角度，而距离中心点的远近则决定投射的力度，在拖拽投射物的过程中，将实理绘制投射预测曲线。



5.3.2 相应公式介绍

角度公式

```
_distanceX=_heroClip.x-_originPoint.x;
_distanceY=_heroClip.y-_originPoint.y;
_heroAngle=Math.atan2(_distanceY,_distanceX);
```

速度公式

```
var velocityX:Number=-_distanceX/_physicsData._velocityFactor;
var velocityY:Number=-_distanceY/_physicsData._velocityFactor;
_arrowVelocity = new b2Vec2(velocityX,velocityY);
```

投射代码

```
_hero.SetLinearVelocity(_arrowVelocity);
```

5.4 本章小节

5.4.1 本章应用我们的框架，实现了城堡的搭建。

5.4.2 通过对投射机制的分析，计算出相应的作用力，并通过 Box2D 物理引擎将相应的作用力加到投射物上，使之产生相应的动能，实现抛物投射的效果。

5.4.3 当投射物与城堡碰撞时，进行相应的碰撞检测，并实行不同的减血策略来控制城堡的损坏。

下图为攻城模式的游戏截图

第六章 总结与展望

6.1 总结

本文通过对 Flash Box2D 的细致分析，在此基础上设计实现了一套轻量级的物理开发框架

通过 GameWorld 实现了对物理世界的初始化封装，简化了开发的流程，并重点对物理对象的创建进行了工厂模式的设计，极大的简化了物理对象的创建。

6.2 展望

对于任意多过形的顶点的采集，在将来的开发中将作进一步的整合，真正实现界面化操作。

目前只是对 Body 的创建实现了封装，对相对简单的应用已经足够。将来将会把关切的创建也整合进框架中

因为能力时间有限，还有很多不足之处，希望批评指正，共同进步。

参考文献

[1]
Box2d 类图结构
http://www.cnblogs.com/Memo/archive/2012/10/05/2712400.html

Create non-convex, complex shapes with Box2D
http://www.emanueleferonato.com/2011/09/12/create-non-convex-complex-shapes-with-box2d/

运行时绘制多边形刚体
http://bbs.9ria.com/forum.php?mod=viewthread&tid=141365&highlight=box2d

Box2D 封装
E:\BOX2D\PhysInjector-master\physinjector\docs\index.html

JSFL
http://wenku.baidu.com/view/e313d21fc5da50e2524d7f42.html