## 主要内容：

* 游戏中的布局算法
* 分屏模式实现
* 游戏中的动画效果
* 使用flash动画和粒子编辑器
* 如何做各种进度条
* 如何解决小手机上白图的问题
* 如何判断设备是否是平板电脑，android特殊设备尺寸的换算
* 理解android版本号作用以及误区
* 理解android启动模式及支付问题

## 1.布局算法

目标：在一块可扩充的矩形区域中填充图片，要求尽量的把矩形区域填满，同时要求图片尽可能的随机和杂乱。图片有不同的尺寸，填充过程中可以旋转。

参考的方法：libgdx的合图算法，box2d的物体碰撞后稳定的思想，动态规划思想，网格划分等

思想：从左到又，从上到下往矩形区域填充图片，直到所有图片都填充完为止，被填充的图片是随机选择并随机旋转。为控制和管理能够填充的区域，需要定义一个叫corner的定义，实际是一个矩形框的左上角（带有可用边长区域，如果图片的宽高小于可用边长区域，就可以插入该corner角所代表的区域中），整个排图过程实际上就是往corner角里填图，最后把整个矩形区域填满。示例图如图1.1所示。

1

2

3

4

5

6

7

Corner

图1.1 填充示意图

插入顺序以图1.1中的编号为顺序，在每次插入一张图片后就会生成新的corner，同时需要更新之前的corner，因为插入的图片会贯穿或者对原来的corner产生新的边界，另外新产生的corner需要根据以前的插入信息来修正可用区域。插图的过程可描述为：

1. 根据当前的corner，遍历所有corner找到满足合适条件的corner插入当前图片，原corner分裂为2个新corner
2. 根据插入的新图，对原来所有的corner角进行更新:更新被纵切的corner-》跟新被横切的corner-》更新分裂的corner1-》更新分裂的corner2-》corner链表排序
3. 重复1-2步直到插入的图片完成为止
4. 最后对插入的矩形区域的右边界进行修正

排序问题：JDK的集合的排序和android API的排序在实现逻辑判断上有区别，在桌面上排序正常，而在android上不正常。android API关于comparator接口的说明an integer < 0 if lhs is less than rhs, 0 if they are equal, and > 0 if lhs is greater than rhs. JDK里关于comparator的说明：根据第一个参数小于、等于或大于第二个参数分别返回负整数、零或正整数。可以看出两个接口对排序的说明都是一样的。但如下定义一个comparator时：

**public** **static** **class** SortComparator **implements** Comparator {

**public** **int** compare(Object element1,Object element2) {

**int** x1 = 0,x2 = 0;

x1= element1.x;

x2= element2.x;

**return** x1>x2?1:0;

}

}

因为没有处理x1<x2的情况，一种可能的处理情况是：JDK先处理大于情况，if(>0){}else{}，而android　API进行处理时，先处理小于情况，if(<0){}else{}，导致总是认为x1>=x2。因此最好的写法是，对三种情况都写返回值，或者return x1-x2。

## 2.分屏模式实现

目标：游戏由横屏变为竖屏，游戏分成两个部分。

思想：考虑到游戏横屏已经实现，对于分屏只需要将横屏的逻辑单元显示改为竖屏，并改变显示的区域就可以。一种方式是重写横屏逻辑单元的坐标，原有的一个逻辑单元分成两个逻辑单元，只是坐标的位置关系上不同。另一种使用照相机，对之前的逻辑单元进行旋转到对称位置，需要改变照相机的视口大小或者改变opengl的视口大小。

实现：为避免对整个横屏的逻辑单元的坐标的修改（每一个元素都得考虑，工作量大），采用修改照相机的方式实现分屏。以横屏800\*480为例，竖屏下每一个逻辑单元还应该保持为800\*480，通过摄制照相机的旋转角度为180度可以将一个逻辑块的显示倒过去，为控制竖屏下在800\*480区域内显示，那整个摄像机的视口由以前的800\*480应调整为800\*960。转换图如下图2.1所示。

960

800\*480

9

8088

800\*480

480

图2.1 分屏转换图

事件处理问题：在使用libgdx的ScrollPane组件时，分屏处理该组件上的两个事件时会出现第二个滑动事件不能正常响应。原因为ScrollPane组件内部添加了一个手势检测事件监听器，当组件检测到两个touchdown事件时，认为需要做zoom操作，会把第二个touchdown事件的后续滑动处理认为是第一个touchdown事件后的zoom操作，因此会出现问题。

事件处理问题解决方式：一种方式是修改手势检测部分的源码，把zoom事件的响应条件改变（这样会导致源码的混乱，比如另外有的地方需要正常的手势检测）；另一种方式是重写ScrollPane组件，把手势检测事件监听器换成需要的事件监听器，同时为处理fling等手势，需要监听滑动的速度，可以在touch事件中处理VelocityTracker，这个可以参考手势检测Gdx/Input/GestureDetector中的实现。

## 3.游戏中的动画效果

1. 多使用libgdx的action动画，了解所提供的各种动画，这里的行为一般都是对actor的尺寸，位置，透明度的改变。多了解action的插值的函数，组合起来就可能会实现一些比较的好的动画效果。一般的动画效果都能够通过action或者组合action实现，如果无法组合，就自己根据对action的理解来实现自己所需的动画效果，本质是一样的。
2. findobjects游戏中的主界面的图片波动效果，实际上是对各个actor进行的一个位置的改变，位置的偏移是根据动画的激励源计算每一个actor的振幅并基于该振幅做一个振幅衰减的正弦偏移，同时要考虑actor与激励源的距离来设置actor开始发生偏移的时刻。
3. Libgdx的action的差值函数，有线性、正弦、指数、弹性等，具体在libgdx中使用的插值函数与线性函数的差别可以从下列图中看出。函数以2-4过程进行插值，线性的为一条直线，后续不进行说明。

Linear与sineIn，sineOut：红色的曲线为sineIn，灰色的曲线为sineOut，如图3.1所示。

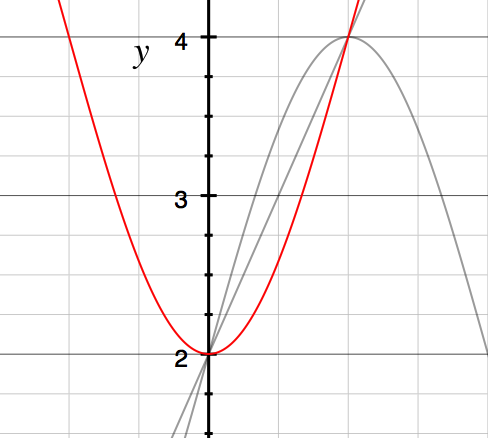


图3.1 Linear与sineIn，sineOut

Linear与circle，circleIn，circleOut：红色的曲线为circleIn，灰色的曲线为circleOut，蓝色的为circle，如图3.2所示。

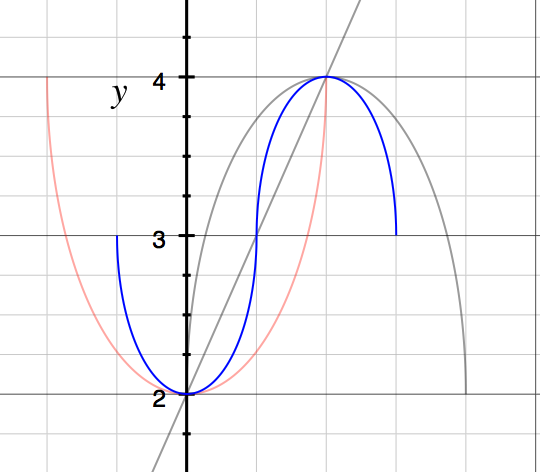


图3.2 Linear与circle，circleIn，circleOut

Linear与pow3，pow5：红色的曲线为pow3，蓝色的曲线为pow5，如图3.3所示。

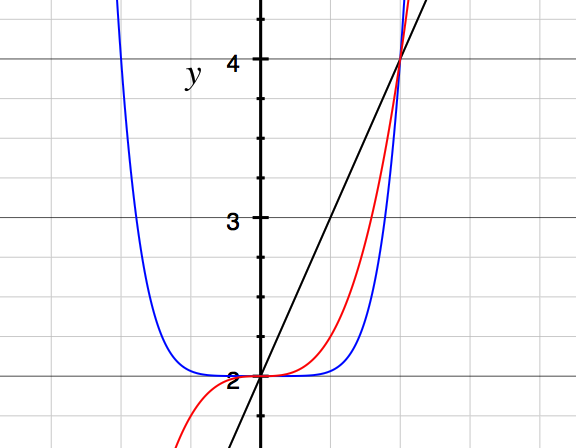


图3.3 Linear与pow3,pow5

Linear与exp10In，exp10Out，exp5Out：红色的曲线为exp10In，蓝色的曲线为exp10Out，黄色曲线为exp5Out，如图3.4所示。

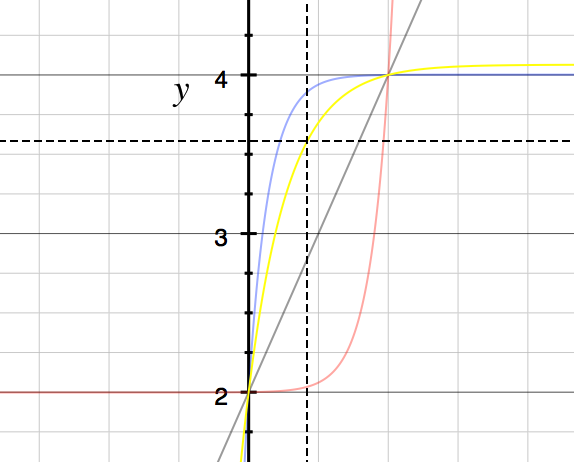


图3.4 Linear与exp10In，exp10Out，exp5Out

Linear与elasticIn，elasticOut：红色的曲线为elasticIn，灰色的曲线为elasticOut，如图3.5所示。

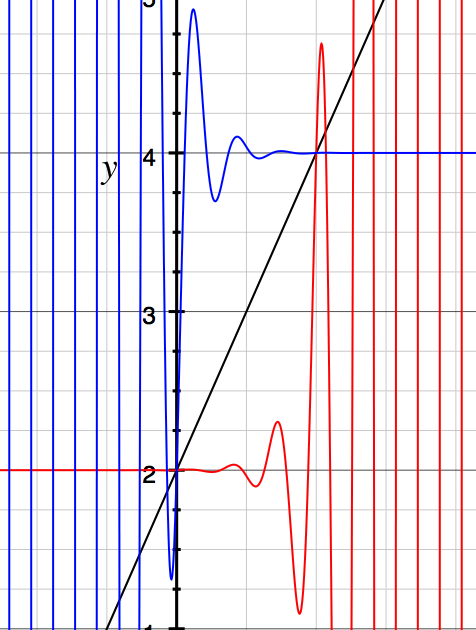


图3.5 Linear与elasticIn，elasticOut

Linear与SwingIn，SwingOut：红色的曲线为SwingIn，灰色的曲线为SwingOut，如图3.6所示。

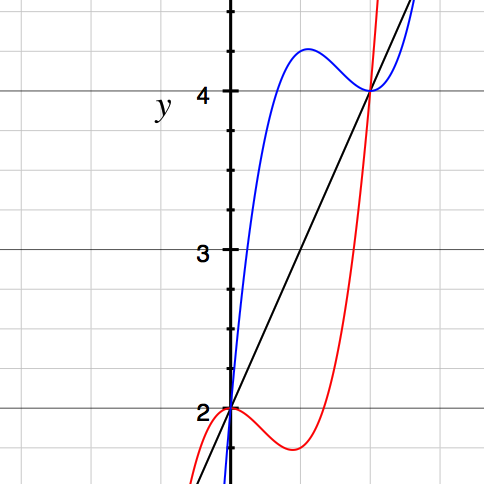


图3.6 Linear与SwingIn，SwingOut

另外还有Bounce插值，这个不是线性的，关于BounceIn可以理解为一个球从高处落下来要弹几下然后就静止了，做一个类似的从上到下的位移插值就是这种现象。

## 4.使用flash动画和粒子编辑器

### flahs动画

优点：能够还原设计师的最真实意图，省掉工程师自己做动画耗费的巨大代价。比较适合于游戏中需要制作特效的UI以及游戏中的不受游戏状态影响的特效（主要是能够保证一个flash动画顺利的播完），对于在动画播放过程中需要改变到其它的状态，这涉及到动画之间的衔接和控制问题，不太建议去使用flash动画，但是如果需要，还是可以做的，关键是要加一些动画跳转的监听以及动画之间的过渡工作。另外，使用flash动画能够大量的减少图片的资源大小，像2d类人物游戏等比较适合使用这种方式，但必须处理好上述问题。

使用：就是把用flash cs5等工具制作的flash动画文件，转换成程序中易于使用的格式文件，然后根据文件信息还原出flash动画。需要两个工具包，一个是转换工具，另一个是生成动画包。目前使用flash解析版本是由严凯写的，工具介绍如下。

1. 转换工具：

作用：提取flash原始动画文件信息，并重新组装，同时对flash中使用的图片资源进行整合，生成一个png文件和一个xml动画描述文件。

转换工具地址：<http://192.168.1.100:8090/XuSheng/flashparsertool>

导入工程后，运行程序出现如图4.1的运行界面。

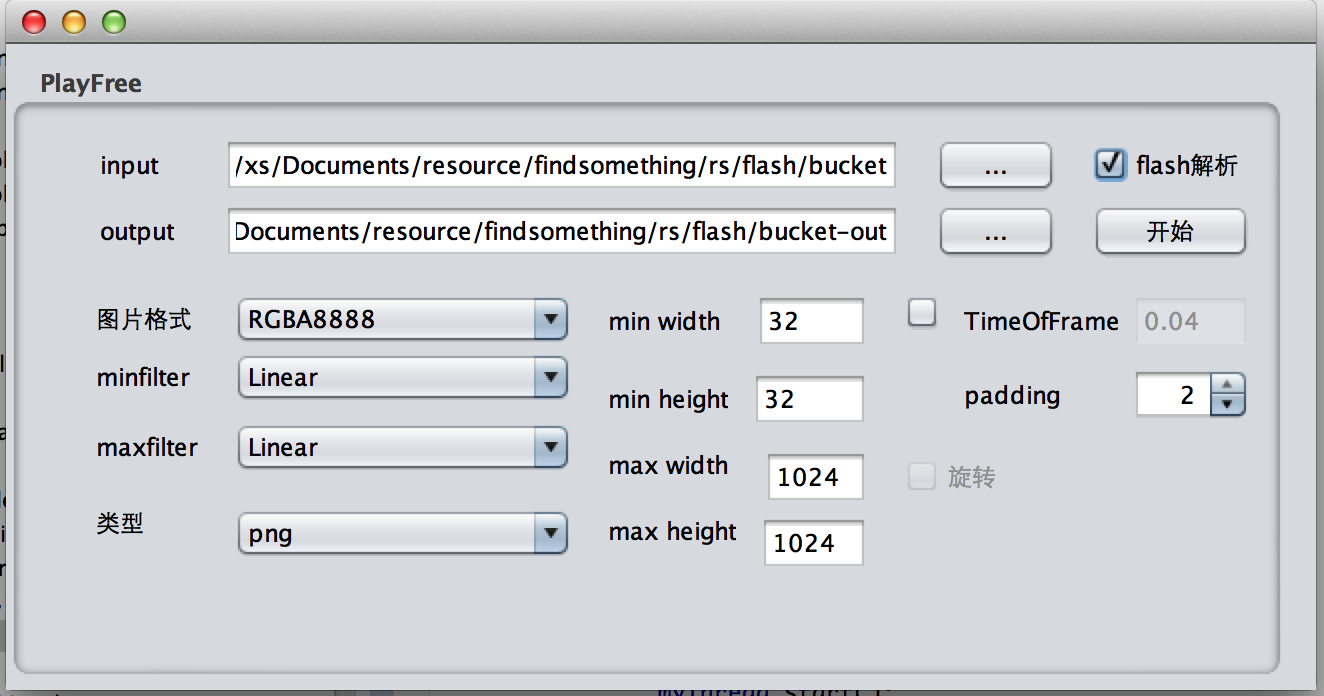


图4.1 flash转换工具

Input表示使用flash工具制作的flash的整个文件（flash工具保存时必须存储为xfl文件格式的flash），output为转换后的文件的输出路径，勾选flash动画解析选择按钮，点击开始，在output路径下就会生成需要的文件信息。

1. 动画生成包

源码地址：在demon的com.fd.kingsky.frame.flash文件下，demo地址见下。

将下载后的源码拷贝到自己当前的工程下，将生成的资源文件拷贝到工程下的asset目录下，只需要如下就可以生成一个动画对象并播放动画：

starFlash=new FlashPlayer("data/flash/xingxing-test.xml", new TextureAtlas(Gdx.files.internal("data/flash/xingxing-test")), new Vector2(200,100));

在render()中调用

starFlash.updateRunTime(Gdx.graphics.getDeltaTime());

starFlash.drawFlash(batch)；

在需要播放flash的地方调用starFlash.rePlay()。

1. 具体的使用可以参见这个demo，地址为：

<http://192.168.1.100:8090/XuSheng/demo_flash_progress_particle>

因在flash工具包中将顶点的变换信息等最后都转换成了顶点的位置信息，在SpriteBatch中没有提供直接使用顶点位置的这样的函数，因此需要在SpriteBatch中添加一个函数：public void draw (TextureRegion region, float fx1, float fy1, float fx2, float fy2,float fx3, float fy3,float fx4, float fy4)。

另外，因程序处理需要，在设计师制作flash时，需要在flash动画的每一层的最后一帧中补上一个空帧。

### 粒子编辑器

优点：只需一张图片，可以做出多种不同的粒子效果，比较适合于一些特效，如游戏结束后的星星爆炸，游戏中的一些道具特效，达成成就后的衬托效果等。

粒子编辑器的源码在com.badlogic.gdx.tools.particleeditor包中，libgdx源码的下载地址：<https://github.com/libgdx/libgdx/>

粒子编辑器主要就是对粒子的发射数量，粒子的生命周期，粒子的物理属性等控制来实现各种不同的效果，粒子编辑器的使用可参考：

<http://blog.sina.com.cn/s/blog_940dd50a0101cn6e.html>

粒子编辑器中的参数说明可参考：

<http://hylxinlang.iteye.com/blog/1884964>

关于粒子编辑器的使用见demo：

<http://192.168.1.100:8090/XuSheng/demo_flash_progress_particle>

## 5.如何做各种进度条

目标：为解决一些特殊形状，如圆角矩形，圆形等形状的进度显示。

思想：一般的矩行进度条只需要拉伸矩形就可以，对于圆角矩形部分，可以通过改变圆角图片纹理的uv值来实现圆角图片的部分显示。libgdx中可以改变TextureRegion中的uv值。对于圆形形状的绕圆心转动的进度条（如dota里面的技能冷却效果），可以使用libgdx中的ShapeRender来实现，将一个圆分成n份，每一份为一个三角行，然后根据进度算出当前需要显示的三角形数量，用灰色的阴影画出三角行就可以。另外对于方形的，圆形旋转的进度条，基本上都可以通过拉伸，分裂，改变uv映射值来实现。对于比较特殊情况，如一个人物显示的进图条，可能需要两张相同的图片来实现，但是原理都是一样的。

实现：可以参考demo，效果如图5.1所示。

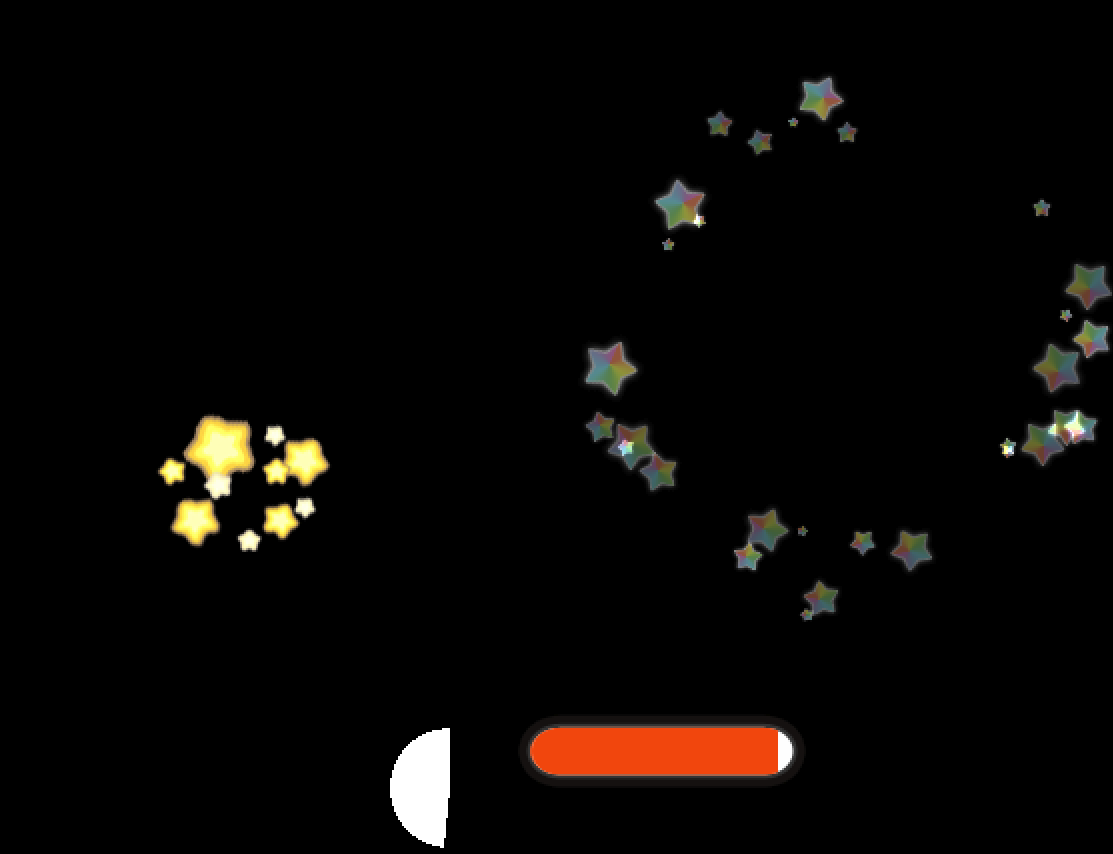


图5.1 demeo示意图

demo源码地址（关于flash，粒子编辑器，进度条在同一个demo中）：

<http://192.168.1.100:8090/XuSheng/demo_flash_progress_particle>

## 6.解决小手机上白图的问题

目标：为解决低端手机渲染图片时由于图片过多导致内存不足出现的白图现象

思想：目前公司最低端手机支持的图片数大概为4张1024\*1024的图片，对公司早期的游戏可能能够满足需求，但目前的游戏一般都会有6张1024\*1024以上的图片需求。为增加图片的载入量可以适当的减少图片的质量，如原来以RGBA8888的方式载入图片，可以改为以RGBA4444的方式载入图片，这样可以增加一倍的图片数量，这样基本能够满足当前6张1024\*1024图片的需求，如果还无法满足需求，再进一步对游戏中的各个显示模块所需的图片进行细分，不同的需求下加载不同的图片。对于选择不同的模式加载图片，可以通过考虑游戏的显示尺寸来区分，以480\*800为分界线，小于该分辨率的可以选择RGBA4444的模式加载图片（因为我们的游戏都是以480\*800为基准做的游戏，小于该分辨率的手机显示图片得缩小，显示本身就会有一定的模糊，以RGBA4444的模式载入图片，在显示效果上不会有太大的差别，同时能够增加一倍的载入容量）。

实现：可以针对低端手机做一个相应的pack文件，修改RGBA8888为RGBA4444，加载时判断手机的尺寸（或者判断手机的内存大小），加载相应文件就可以。另外也可以修改libgdx的源码，对于Texture或者 TextureAtlas可以修改相应的文理加载部分，添加相应的条件以及加载格式，对于异步加载的需要修改TextureLoader或者TextureAtlasLoader部分。如TextureAtlasLoader部分：

**public** Array<AssetDescriptor> getDependencies (String fileName, TextureAtlasParameter parameter) {

**...**

**if**(FindSomethingGame.screen\_size!=FindSomethingGame.screen\_small)

params.format = page.format;

**else**

params.format = Format.RGBA4444;

…

}

## 7.如何判断设备是否是平板电脑，android特殊设备尺寸的换算

目标：游戏可能需要考虑到平板和普通手机的不同处理，需要知道设备是否是平板电脑。

思想：因目前很多手机的分辨率都达到了1280\*800，以分辨率来区分设备是否是平板电脑不太准确。当前手机尺寸一般在6英寸以下，可以通过求出当前设备的物理尺寸来区分是否是平板电脑。求实际尺寸时，可以根据设备的dpi（屏幕像素密度，每英寸像素点）来计算，通过在水平和垂直方向上计算p/dpi就可以得到水平和垂直方向上的长度（英寸），再求出设备对角线长度即为设备的尺寸。屏幕的像素，屏幕像素密度都可以通过读取手机参数获得。

需要注意的是有些手机的屏幕采用了p排列技术（如摩托罗拉的pe860），p排列会用低分辨率去模拟高分辨率的效果，如一个采用p排列的手机读出的分辨率为960\*540，但实际的像素点却没有这么多，可能为800\*480（或者更少），而读出的屏幕像素密度却为实际的像素计算得到的，这样计算的结果会导致比实际计算的尺寸要大，导致手机会误判断为pad。这种情况可以考虑使用一个估算值desnityDpi，这个是根据最终像素确定的一个逻辑值，代表了屏幕的逻辑像素密度，如160，240等，是一个接近真实屏幕像素密度的一个范围值。在进行尺寸计算时，可以先判断实际的屏幕像素密度dpi和desnityDpi的差值，确定使用dpi还是desnityDpi进行计算。实现如下：

**public** **boolean** isPad(DisplayMetrics dm ) {

/\*\*

\* pe860手机采用的是p排列技术，因此实际的屏幕密度像素并不是真实的，还会模拟出多余的像素，要算屏幕实际尺寸，需要知道实际的像素或者是模拟后的单位屏幕像素值

\* denisty 是模拟后计算得出的（实际是一个范围值） dpi是实际真实的 densityDpi实际是通过denisty\*160计算的一个估算值

\*/

**double** x=0,y=0;

**if**(Math.*abs*(dm.densityDpi-dm.xdpi)>40 && dm.densityDpi>160 || Math.*abs*(dm.densityDpi-dm.xdpi)>20 && dm.densityDpi<=160 ){ //如果逻辑屏幕密度像素值与实际屏幕像素值相差大于40个像素或者20个像素（根据逻辑屏幕密度确定），认为像素是经过模拟的，计算实际尺寸使用屏幕像素密度进行计算，否则使用每英寸像素值进行计算

x = Math.*pow*(dm.widthPixels / dm.densityDpi, 2);

y = Math.*pow*(dm.heightPixels / dm.densityDpi, 2);

}**else**{

x = Math.*pow*(dm.widthPixels / dm.xdpi, 2);

y = Math.*pow*(dm.heightPixels / dm.ydpi, 2);

}

// 屏幕尺寸

**double** screenInches = Math.*sqrt*(x + y);

// 大于6尺寸则为Pad

**if** (screenInches >= 6.0) {

**return** **true**;

}

**return** **false**;

}

## 8.理解android版本号作用以及误区

目标：理解minSdkVersion，maxSdkVersion，targetSdkVersion，前向兼容，后向兼容

理解：

1. 前向兼容：低版本编译的应用在高版本上的设备上兼容
2. 后向兼容：高版本编译的应用在低版本设备上兼容
3. minSdkVersion指定应用程序安装所需要的最小版本号，不指定默认为1，为匹配后向兼容问题，如高版本增加了低版本7没有的api，可以设定最小版本号为7
4. maxSdkVersion指定应用程序安装的最大版本号，如果设备系统升级后maxSdkVersion小于系统当前的版本号，应用会被删除，官方文档上不建议设置该值，目前的作用可能是googleplay上能够被过滤，或者设定应用程序的版本生命周期
5. targetSdkVersion指定应用程序的目标版本号，不指定情况下等于minSdkVersion。官方文档上这么说的：

This attribute informs the system that you have tested against the target version and the system should not enable any compatibility behaviors to maintain your app's forward-compatibility with the target version.

这个属性会告知系统对于从最低版本号到指定的目标版本号不开启前向兼容测试。比如说，如果设定目标版本号为11，那么应用在运行到系统版本11或者更低的时候，不启动兼容性测试。不启动兼容测试会显示高版本的一些新的特性效果，如舍弃显示虚拟的menu键盘等。当然，对于运行的系统的api的等级高于目标版本号时，为保证应用的正常运行，系统会开启兼容性测试以保证应用以预期的方式运行。

#### 关于设定targetSdkVersion不同出现的问题：

Findobjects在设定targetSdkVersion=11时，开启对战模式进行横竖平切换时，游戏正常，但是设定targetSdkVersion=14时，游戏运行在设备系统版本号4.0时游戏会直接退出。原因在于，设定android:configChanges时没有添加sreenSize属性，这个属性是在版本号13时引入的，这个属性在13号版本及以上时会被检测，如果设定了targetSdkVersion=14，表示在系统4.0及以下版本都进行了兼容性测试，当应用运行在4.0版本时，系统不会做兼容性检查，会以configChanges相关的属性进行检测，会检测到sreenSize发生了变化，而configChanges中却没有对screenSize的变化进行配置，因而activity会重新启动。运行在版本号13以下时，系统不处理sreenSize的改变，因而不现错误。

因此为保证应用能正确运行，测试至少应该覆盖系统版本从minSdkVersion到targetSdkVersion的设备。另外，在设定targetSdkVersion时，应该要关注一下目标版本号是否引入了新的特性，这些特性可能在系统不启动兼容性运行应用时出现问题。

参考：

<http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#filtering>

<http://developer.android.com/guide/topics/manifest/activity-element.html>

<http://hi.baidu.com/jenus_hi/item/3ee82cecb695f308560f1dc9>

## 9.理解android启动模式及支付问题

目标：android任务栈，启动模式，支付流程

理解：

#### android任务栈

官网上的定义：

A task is a collection of activities that users interact with when performing a certain job. The activities are arranged in a stack (the "back stack"), in the order in which each activity is opened.

一个task就是一系列的activity的集合，在同一个task中的activity是以栈的形式排列的，不同的task可以切换到前台显示。task一般是通过应用程序启动界面的icon启动时创建，默认情况下，每启动一个不同的应用都会启动一个task。另外，当从一个activity启动另一个 activity时，会根据目标activity设置的启动模式来判断是否直接将生成的activity加入到当前的task还是新建一个task，或者是将已经存在的task移动到前台显示。

#### android启动模式: standard，singleTop，singleTask，singleInstance

standard：总是会产生一个新的activity实例，添加到当前task的栈顶

singleTop：当当前的activity不在task的栈顶的时候，会产生一个新的activity实例并添加到当前task的栈顶，否则会复用当前task栈顶的activity

singleTask：当当前的activity没有实例时，会新创建一个task，并把当前的activity作为新建的task的根activity。如果当前的activity已经存在一个实例，那么会将当前activity所在的task移动到前台，并把task栈中在当前activity上面的activity全部移除，显示当前的activity（task中只有当前activity）

singleInstance：启动时候与singleTask类似，只是task栈中永远只会有一个activity实例，如果在一个包含singleInstance的activity的task中要启动另外的一个activity，如果要启动的activity启动模式为standard或者singleTop，会重新的启动一个新的task，并把新启动的activity作为栈顶；如果为singleTask或者singleInstance，找到包含该目标activity的task，移动到前台，否则新建task和activity。

启动模式问题：

游戏的主activity设置为singleTop模式，游戏通过googleplay打开后，会新建一个游戏主activity的实例，因为未考虑到多个实例的变量问题，导致在新开的游戏中出现一些异常bug。 （目前公司的关于设置singleTop模式的游戏，在通过googleplay打开后，基本上都会有各种莫名其妙的问题）。

Task，启动模式参考地址，官网上说的比较详细和权威，建议看第一个：

<http://developer.android.com/guide/components/tasks-and-back-stack.html>

<http://blog.csdn.net/go_to_learn/article/details/9855881>

<http://marshal.easymorse.com/archives/2950>

#### 支付流程

大概流程如下，详细过程可以参考官网：

1. 初始化参数，请求服务，建立支付流程
2. 如果建立成功，查询是否有商品可以消费，如果可以消费，消费商品
3. 向服务器请求商品信息，服务器返回商品信息，根据信息生成一个PendingIntent
4. 用户购买商品，发送购买流，根据服务器的返回结果和操作结果确定是否购买成功，如果成功就消费。消费是在PendingIntent退出的时候，游戏的主activity会处理onActivityResult，在onActivityResult中处理购买的返回值并调用实际的消费函数。

支付问题：

当购买游戏道具成功后，还未等PendingIntent退出的时候，按了home键返回到手机的menu界面，然后再进入游戏后，会出现购买的道具数量未能及时加上的问题，重启游戏后道具数量正常加上。原因在于：游戏的主activity设置启动模式为singleTask模式，按home键后再次返回游戏时候，主activity上的PendingIntent将会被移除掉，在onActivityResult中未能捕获到正常的信息，导致不能执行商品消费。重启游戏商品信息加上是因为在activity启动时会建立支付过程，支付建立完成后会查询是否有未消费的商品（商品是否消费信息保存在服务器上），如果有就消费该商品。把将游戏的主activity设置为singleTop模式，home键返回时PendingIntent不会被移除，后续流程正常，可以消除上述操作商品未能及时消费的问题。

#### 关于singleTask与singleTop的使用注意：

不建议游戏的主activity设置为singleTop模式，因为游戏实体只应该在手机上存在一份，一方面考虑到游戏占用的内存，另一方面考虑到数据的一致性（除非都是全局变量，否则必然导致不一致）。实际上解决上述支付问题还有另外一种方式，就是在游戏每次onresume的时候，对服务器做一次请求，询问是否有未被消费的商品，就能够解决问题（这种方式我没有测试，有兴趣的同学可以试一下，如果坚持使用singleTask模式的话）。其实这种问题影响也不大，出现的概率也小，用户不会在支付的时候按掉home建，即使如此，实际的商品信息也没有丢失。

SingleTask模式除了会出现上述支付问题外，还有一点体验不好的地方就是打开moregame后，按home键再返回moregame会被移除，当然只要在当前任务栈中并在主activity以上的activity都会被移除，目前没有碰到其它的问题。

如果使用singleTop需要注意的地方：

始终要考虑到可能会有多个游戏实体在运行，对于变量以及静态变量的初始化工作要认真考虑：

建议游戏中的图片，声音，文件，用户的商品信息等关于资源部分的引用全部使用静态变量

一般的类中的成员变量，如果是静态的最好是为final类型的，不建议可变的变量使用静态的，如果不得已使用了可变的静态变量，游戏启动时最好初始化。

支付流程参考地址：

<http://developer.android.com/google/play/billing/api.html>

## 10.建议

1. 理解libgdx的组件，多使用其整体框架，可以多继承和重写组件，但不建议自己维护框架
2. 整体动画一定使用矩阵实现，不要单个去计算

关于矩阵的使用部分，可以参考libgdx 的Group组件中的使用

1. 游戏性能及游戏中静态变量的使用及注意地方

游戏的性能主要是关于渲染部分的，运算部分的靠自己去优化。

虽然libgdx 的spritebatch对2d的渲染做了很好的封装，但是我们不注意时可能会在渲染方面耗费较大的代价。一种可能的情况是：当用texturepacker合成两张大图，两张大图中有很多的小图，在程序中渲染的部分因为渲染次序的关系导致在两张大图频繁切换，耗费较大的代价。因此，如果有可能有类似的情况发生时，应该要注意能否减少这种切换，比如可以在添加actor时考虑到相关纹理的先后顺序来添加actor，同时对于比较大的图比如loading，背景图等建议分开来单独合成。

关于静态变量的使用见支付的后部分说明。