**一、游戏总体框架及其模块：**

1. main.ts: 游戏的入口，对游戏主场景(mainStage)进行调用；

2. mainStage.ts: 游戏的主舞台，包括了背景，我机，敌机，子弹；同时还负责用数组进行这些元素的管理，以及这些元素的位置(注意：元素的移动不在主舞台，是元素调用this进行自我移动)。主舞台另外一个重要的作用是进行元素的碰撞检测，还有就是用对象池的概念创建回收数组，用于回收和再利用创建的实例对象。

3. background.ts: 游戏的背景，包括了背景图片，背景的滚动，以及背景音乐；

4. myplane.ts: 我机的类，主要负责构建我机的图像；

5. bullet.ts: 子弹的类，该类继承自egret.sprite，是对象容器类，不是图像本身，而是装着图像的容器，所以在操作自身时，要直接用this进行。该类主要负责子弹的创建，以及它自身的向下运动。此外，子弹本身还负责从父级容器中消除自己，然后还有两个标志位hit和recycle来表示子弹是否击中，以及子弹是否能被回收；

6. enemy1.ts: 敌机类，该类继承自egret.sprite，是对象容器类，不是图像本身，而是装着动画的容器，所以在操作自身时，要直接用this进行。该类主要负责敌机1的创建，以及它自身的向下运动。此外，敌机本身还包含着mc帧动画，来表现被击中后的爆炸效果。它同样存在着两个标志位hit和recycle来确定它自身是否能被回收；比较关键的是，该敌机的回收是在动画播放完才进行，所以它需要mc帧动画的回调函数来设置其recycle=true；

7. location.ts: 坐标的类，用于获取点击屏幕的坐标；

8. sould\_play.ts: 声音的类，用于获取声音资源并且播放出来，同时设置播放的条件；

9. recycle.ts: 回收的类，内置数组，用于进行元素的回收和查找，以及当元素被重复使用后对其进行弹出的出栈操作。(recycle最终并没有在游戏中使用，因为在mainStage有两个回收数组，它们通过入栈和出栈的操作，已经能很好的实现对象池的效果，能很好的实现对象的回收和再利用)。

二、游戏总体控制流程：

Notice: 不要着急做主场景，而先从单个的元素入手，再在主场景进行合成和逻辑处理：

背景制作-->子弹制作-->我机制作(注意跟子弹的位置同步)-->敌机制作-->碰撞检测(注意，碰撞检测最好在一个场景内进行，即子弹和敌机都在主场景，然后在主场景进行检测，并不太推荐在子弹内部进行检测)-->敌机爆炸效果的帧动画实现-->对象池进行实例的回收和再利用。

**三、关于游戏制作的若干难点的说明：**

1. 背景图片如何铺满屏幕：它需要获取当前屏幕的长度和宽度，所以需要利用舞台对象进行获取。注意舞台对象只有一个，不能用this的方式获取，要用：

public bgstage:egret.Stage = egret.MainContext.instance.stage;

此外，2. 对象的名字不要和类名字一样，要有区别；

2. 在制作子弹类的时候，由于类是一个对象容器，它里面会包含着图片，所以对于子弹本身的操作，如位移，要采用：

this.y -=16; //这里非常重要！错就错这了！注意了，这里的坐标应该是this的，而不是this.box的，因为this是整个容器，而box只是其中的一个对象，所以这里需要整体移动！否则在碰撞检测中会出错；

3. 子弹的移动是自己完成的，之所以这么设计，是为了符合flash中的"自己的动作代码嵌在自身"的特性，这样的好处是，只要写好内部逻辑，一旦被创建，子弹会自己进行移动，而不需要在主舞台进行控制和干预。(当然这么也带来一个问题，就是每一个子弹自己都会绑定enter\_frame事件，这会增加资源消耗)。

4. 敌机的设计和子弹类似，它也是属于"自己动自己"，"自己的代码内嵌"，自己的移动交由自己完成。

5. 注意，这里不用管其坐标，由于是整体移动(this)，所以它的坐标都是跟主舞台一致的。

6. 敌机是一个对象容器，它包含了一个帧动画，第一帧是正常的敌机，后面的几帧则是爆炸的动画。它需要先用flash做成背景透明的gif动画格式(背景透明可在flash发布设置中定义)，然后用Texture Merger进行导出，然后会生产两个文件xxx.png和xxx.json。把这两个文件放入assert文件，则Egret会自动添加。加载帧动画后，让动画对象停在第一帧。

7. 敌机类和子弹类都设置了两个标志位：hit 和 recycle，hit用于标识当前对象是否有碰撞；recycle用于标识当前对象是否能被回收。

7. 在主场景中，通过enter\_frame进行每帧的碰撞检测。碰撞检测会检测两个事儿，第一是否碰撞，第二是否回收，

注意：

7.1 碰撞时的子弹对象和敌机对象都需要存在，不能是undefined，此外它们的hit标识也应该为false，如果为hit=true，那么说明这个对象已经被碰撞过，不再响应后续可能的碰撞(也就是说，对象只能碰一次)

7.2 如果碰撞检测为真，那么把子弹和敌机两个对象的hit改为true即可。

7.3 注意碰撞检测的判断和是否删除对象的判断是分开的，它们都在for循环内部，但是有先后顺序，首先是做碰撞检测，然后再做回收检测，它们不是同时完成，而是分开完成。碰撞检测是对hit位进行修改:hit=true；回收检测是对对象进行释放和回收处理，对象数组弹出对象(splice)，而回收数组则存入该对象。(注意：修改回收标志位recycle不是在这里进行，而是每一个对象元素内部自己判断完成)

8. 子弹(bullet)的消除，回收，以及再利用：

8.1. 子弹的消除相对简单，因为它没有动画，可以即时进行消除和回收。

8.2. 子弹一开始被创造的时候，它有两个标示位，hit和recycle，默认值均为false。当hit=false的时候，它会不断向上位移(enter\_frame)。hit的值会在碰撞产生后产生改变(hit=true)，这个设置是在主场景完成。一旦hit==true，则意味着它需要停下来(移除自身的enter\_frame)，并且立即在主场景消除自己，并且进行回收。

首先，设置自身的recycle=true，

然后，通过父级容器消除自己(这里非常重要，自己消除自己的关键在于使用this.parent.removeChild(this); )，注意在调用parent之前，一定要判断其父级容器是否存在: if(this.parent)。

然后，移除它的帧监听(this.removeEventListener(egret.Event.ENTER\_FRAME,this.box\_move,this);)

8.3. 还有一种情况会导致子弹的消除，即它的位置不在舞台内了(位置判断是子弹自己完成的，不在主场景)。这时候也要做上面的三件事儿。

8.4. 子弹的回收:

在主场景(mainStage)中，进行每一帧的判断(enter\_frame中调用的hit\_target()函数)，看子弹数组里是否有recycle==true的对象，如果有，则把该对象放入之前准备好的回收数组(recycle\_bullet.push)，然后在本身的数组中删除它(bullet\_array.splice)。

8.5 子弹怎么再利用呢？

在主场景中，每创建一个子弹的时候(bullet\_new())，首先判断回收子弹数组(recycle\_bullet)是否有元素，如果有，则调用它的第一个元素，让它成为新的子弹，然后把它装入子弹数组(bullet\_array.push)，同时删除回收子弹数组的第一个元素(recycle\_bullet.shit())。(为啥要用第一个元素？因为它是最早进入数组的，先进先出嘛，跟队列的机制一样)。这样就能实现子弹的循环利用了。

此外，在取出子弹后，需要调用它的initial()方法来对它进行初始化。(为啥要进行初始化？因为这颗子弹是之前使用过的，它的位置，还有标识位hit和recycle，以及它自身的帧监听enter\_frame，都需要进行重置)。

8.6 如果回收子弹数组的长度为0，即里面没有可再利用的子弹，则new一个新的子弹对象，把它装入子弹数组(bullet\_array.push)。

9. 敌机(enemy1)的动画播放，消除，回收，以及再利用：

9.1. 敌机也是一个sprit类，它本质上是一个对象容器，包含了一个帧动画(mc)，然后让其停止在第一帧(mc.gotoAndStop(1))。

9.2 敌机被创建的时候，和子弹类似，它也有两个标示位，hit和recycle，默认值均为false。当hit=false的时候，它会不断向下位移(enter\_frame)。hit的值会在碰撞产生后产生改变(hit=true)，这个设置是在主场景完成。一旦hit==true，则意味着它需要停下来(移除自身的enter\_frame)，并且立即在主场景消除自己，并且进行回收。

9.3 和子弹不同，敌机在碰撞后不会直接消除，而是要播放帧动画(mc.gotoAndPlay(2,3))(这里两个参数的意思是：从第2帧开始播放，播放3次)，这就意味着它在播放完动画之前仍然存在，不会消除。所以这里需要等动画结束后再用回调函数实现消除自己：

if(this.hit==false){ //如果尚未产生碰撞，则继续往下面走

this.y += 10;

}

else{ //如果击中，则hit==true，然后移除当前的enter\_frame函数

this.removeEventListener(egret.Event.ENTER\_FRAME,this.box\_move,this); //移除自身的enter\_frame监听，停止自身的移动

this.mc.gotoAndPlay(2,2); //播放自身帧动画

var sound = new sound\_play("explosion\_small");

var that = this; //用that来表示this，因为在子函数中,this指针会指向子函数本身，而不是对象本身，所以需要增加标示符that来在子函数中指向对象

this.mc.addEventListener(egret.Event.COMPLETE, function (e:egret.Event):void { //监听动画播放完毕事件，然后激活回调函数function

if(that.parent){ //如果当前对象的父对象存在

that.parent.removeChild(that); //调用父对象删除自己

that.recycle = true; //修改自身的标识符recycle，告知主场景，动画已经播放完毕，可以进行回收

} }, this);

} // else

从上面可以看出，碰撞产生时，敌机是不会被回收的，它要等动画播放结束后，才会把recycle=true，同意被回收。

8.4. 敌机的回收:

在主场景(mainStage)中，每一帧都进行判断(enter\_frame调用hit\_target()函数)，看子弹数组里是否有recycle==true的对象，如果有，则把该对象放入之前准备好的回收数组(recycle\_enemy1.push)，然后在本身的数组中删除它(enemy1\_array.splice)。

9.5. 敌机是怎么再利用呢？

在主场景中，每创建一个敌机的时候(enemy1\_new())，首先判断它的回收数组(recycle\_enemy1)是否有元素，如果有，则调用它的第一个元素，让它成为新的敌机，然后把它装入敌机数组(enemy1\_array.push)，同时删除回收数组的第一个元素(recycle\_enemy1.shit())。(为啥要用第一个元素？因为它是最早进入数组的，先进先出嘛，跟队列的机制一样)。这样就能实现敌机的循环利用了。

此外，在取出敌机后，需要调用它的initial()方法来对它进行初始化。(为啥要进行初始化？因为这架敌机是之前使用过的，它的位置，还有标识位hit和recycle，以及它自身的动画和帧监听enter\_frame，都需要进行重置)。

9.6 如果回收敌机数组的长度为0，即里面没有可再利用的敌机，则new一个新的敌机对象(new enemy1)，把它装入敌机数组(enemy1\_array.push)。

10. 对象池的原理小结：

1. 对象数组和对象回收数组是两个数组；

2. 每一帧都进行判断，看是否存在可回收的元素，如果有(recycle==true)，则把该元素装入回收数组(recycle.push)，同时把该元素从对象元素中删除(array.splice)；

3. 在创建新的元素时，首先判断回收数组是否存有可用的元素，如果有，则取出第一个元素，把该元素装入对象数组(array.push)，然后把该元素从回收数组中删除(recycle.shift)；

4. 在使用回收数组中的元素时，需要首先对该元素进行初始化操作。