开发工具及技术

1．C++高级语言程序。C++是在 C 语言的基础上开发的一种通用编程语言，其编程领域众广，常用于系统开发，引擎开发等应用领域，是至今为止最受广大受用的最强大编程语言之一。

2．Cocos2d-x 引擎。Cocos2d-x 是一个开源的移动游戏框架。Cocos2d-x 提供框架围绕 Cocos2d 跨平台为重点发展，其项目很容易在 iOS，Android 等手机平台上运行。

3．Microsoft Visual Studio 2013 。Vs2013 是微软公司在 2010 年第二季度推出的开发环境。其集成开发环境的界面被重新组织与设计，使用起来更加简单明了。这是本软件编写的主要工具。

4．Eclipse。Eclipse 是一个基于 Java 的、开源的、可扩展开发平台。是人气最旺的 Java IDE 之一。

5．Python 高级语言。Python 会在软件开发周期中有一些简单的运用。

系统基本功能

1．基本规则

各种各样的敌机从远处不断的袭来，他们攻击力、飞行速度、形态各不相同，但他们的目的是一样的，那就是杀死主角。当飞机碰到主角飞机是，主角飞机会受到伤害，同时敌机也会受到伤害，直至生命值为零。当然主角飞机会不断发射子弹以阻止无穷无尽的飞机冲向自己，当子弹碰到敌机时会给敌机造成伤害，直至生命值为零。

2．操控

无论是攻击敌机还是躲避敌机的攻击都需要玩家操控主角飞机进行移动。本游戏的移动操作是通过手指（鼠标）在屏幕上移动实现的。当手指（鼠标）触摸在飞机的碰撞体积上时，碰撞事件被触发。

3．敌机 AI

为了让游戏看起来更加生动，本游戏给所有的敌机添加了一定程度的 AI。有的敌机会按既定的曲线行进，有的会自爆似的冲向主角飞机，有的飞机并排出现。这些动作不一的敌机大大增加了游戏的乐趣和体验。

4．游戏模式设置

闯关模式。闯关模式设有三个关卡，每个关卡都会有不同的敌机和 BOSS。

无尽模式。无尽模式中没有 BOSS，只有无限多的普通敌机。

BOSS 模式。BOSS 模式中只有闯关模式中的 BOSS，是游戏模式中最难的，

5．游戏难度设置

游戏基础难度设置总共分为三种：easy、normal、hard。

若游戏为闯关模式。越到后面关卡敌机和 BOSS 的血量和攻击都会上升。

若游戏为无尽模式。无尽模式游戏难度不仅会随着游戏基础难度变化而变化，也会随着游戏积分的增加而增加。

若游戏为 BOSS 模式。BOSS 模式中的 BOSS 也会随着游戏基础难度的增加而增加。

6．游戏礼物

为了增加游戏的乐趣，本游戏为玩家添加了游戏道具。

本游戏的游戏礼物共有四种：1.增加子弹的列数。得到该道具后主角飞机外观会做出相应改变。2.增加子弹的伤害。得到该礼物后子弹的外观会做出相应改变。3.大招礼物。得到该礼物后，会有一次机会释放一排导弹，导弹从屏幕底部飞至屏幕上部。4.加血礼物。得到该礼物后，主角飞机血量会增加。

7．成就

为了增加玩家对游戏的不断挑战的乐趣。本游戏增加了成就系统。

8．本地储存

为了激励玩家挑战自己，游戏增加了最高分数设置。每次游戏结束，游戏都会读取本地历史最高分数，如果当前分数超过了历史最高分数，则将当前分数作为最高分数存储起来。

9．游戏帮助

为了使新手玩家能够更快适应游戏，本游戏设置有游戏帮助，使其能快速了解本游戏。

10．游戏音效

为了使游戏更加立体、生动，本游戏设置了游戏音效，当然如果主角不喜欢游戏中音乐，可以在游戏中关闭。

2.3 系统流程图

玩家启动游戏后，游戏会预先载入必要的游戏音效和图片等资源文件。进入欢迎界面后可以选择查看游戏帮助还是开始游戏。点击开始游戏进入选关界面，在此玩家可以选择自己想玩的关卡。进入游戏后游戏会检测玩家战机是否与敌机相撞、子弹是否击中敌人。当子弹打中敌人后，若敌人血量降为零，执行爆炸动画并增加积分。若主角与敌机发生碰撞，判断敌人是否血量降为零，执行爆炸动画并增加积分，并且判断主角血量是否降为零，若是，直接进入结束界面。若本场分数超过最高分数，则将分数写入本地。在结束界面中可以选择是否接受和是否重新开始。若玩家选择的是闯关模式，游戏会在一定时间后加入关主（BOSS）相关资源。若玩家选择的是无尽模式，游戏难度会随着时间难度的增加而增加。若玩家选择的是关主模式，游戏只会载入关主（BOSS）资源，不会载入普通敌机资源。系统流程图如图 2.1 所示。

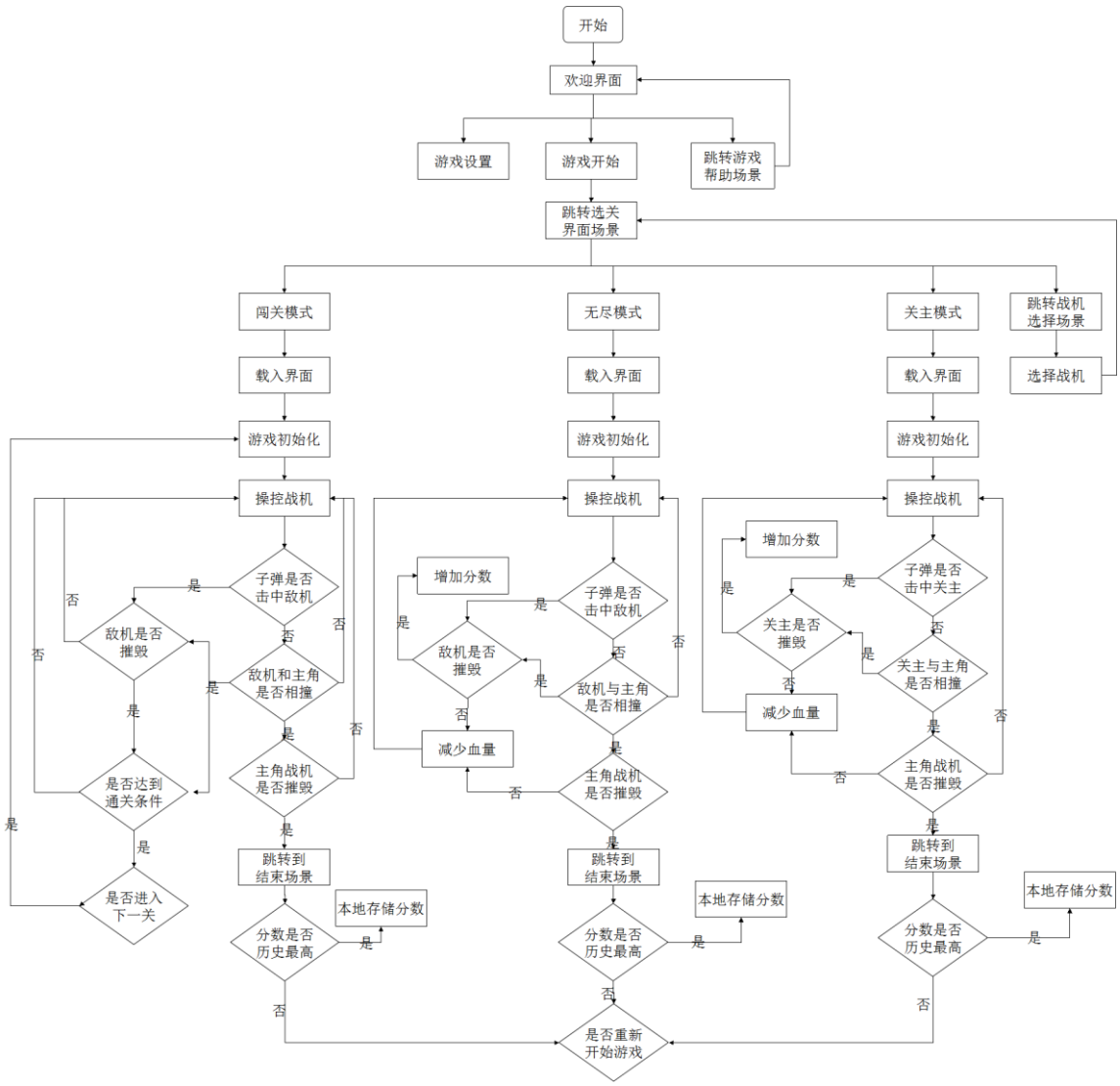

图 2.1 游戏流程图

系统基本思路

几乎所有软件的开发阶段都是从系统设计正式开始。这部分任务是把之前所做的分析模型转换为软件的设计模型。设计模型会描绘出软件的总体概貌。但由于此过程一般为迭代过程，在设计之后有可能再回到分析，又由于篇幅有限，所以在本文中会以最简洁的方式描述（若在系统设计中回到了分析，本文将不再过多描述，直接描述出主要成果）。最终，会把软件的每一个环节细化，把它加工成非常接近目标程序的模型表示。

3.3.1 游戏执行模块设计

游戏执行模块会在游戏每一帧中执行，算法分为以下几步：

1．判断主角战机是否与敌机相撞。若相撞则执行主角模块和敌机模块。

2．判断子弹是否击中敌机。若击中则执行敌机模块和积分模块。

3．判断主角是否与礼物相撞。若相撞执行相应礼物模块和主角模块。

4．判断敌机子弹是否击中主角。若击中则执行主角模块。

3.3.2 主角战机移动模块设计

游戏添加了三种战机，属性如表 3.1 所示：

表 3.1 主角战机属性表

主角战机	血量	攻击力	大招攻击力
主角战机一	500	150	800
主角战机二	1000	100	400
主角战机三	1500	50	200

主角战机移动算法分为以下几步：

1．检查每一帧，鼠标是否点击屏幕，若触摸则进入下一步。

2．判断鼠标点击位置是否在飞机碰撞矩形内，若在则进入下一步。

3．判断鼠标是否抬起，若未抬起进入下一步，若抬起返回第一步。

4．判断此时主角飞机是否超过屏幕，若超过则将飞机坐标设置为屏幕最大边缘处。若没有则执行下一步。

5．将飞机坐标设置为当前鼠标位置。

3.3.3 主角模块的设计

主角模块分为闯关模式主角模块、无尽模式主角模块、关主（BOSS）模式主角模块，分为以下几步:

1．游戏每一帧执行主角战机移动模块。

2．判断主角血量是否小于零，若小于则执行下一步。

3．执行主角爆炸动画，移除主角和主角发射的子弹。

3.3.4 子弹模块设计

子弹模块设计可以分为主角战机模块和 BOSS 子弹模块。子弹属性如表 3.2 所示：

表 3.2 子弹属性

子弹	初始伤害	加伤害礼物加伤害量
主角子弹	100	25

关主（BOSS）子弹	100	--

算法分为一下几步：

1．获取主角（BOSS）坐标位置。

2．计算主角（BOSS）坐标与屏幕顶部距离，再计算出子弹运行时间。

3．添加子弹精灵到场景中，并使子弹从飞机位置向屏幕顶部移动。

4．判断子弹是否击中敌机（主角），若击中跳转游戏执行模块并回收子弹，否则执行下一步。

5．子弹移动至屏幕外后回收子弹。

3.3.5 敌机模块设计

敌机一共有五种类型，属性如表 3.3 所示.

表 3.3 敌机模块设计

敌机	血量	速度（像素每秒）	飞行路径	积分	几率
敌机 1	300	200	贝塞尔曲线	100	30/100
敌机 2	400	200	三架并列飞行	200	15/100
敌机 3	500	180	单架敌机飞行	100	15/100
敌机 4	350	250	冲向主角	100	20/100
敌机 5	150	200	三架并排飞行	100	20/100

算法分为以下几步：

1．对五种飞机素材进行渲染，并加入到敌机数组中。

2．随机添加各种敌机精灵，并按各自的飞行速度以及飞行路径飞行。

3．若敌机被摧毁，执行爆炸帧动画，在敌机数组中删除敌机精灵并增加积分。

4．若敌机飞行至屏幕底部，在敌机数组中删除敌机精灵。

5．判断是否为闯关模式，若是则执行下一步。

6．判断是否加入 BOSS，若是添加 BOSS，并执行 BOSS 模块。

3.3.6 积分模块设计

积分模块算法分为以下几步：

1．每击毁一架敌机，增加相应分数并显示。

2．判断游戏结束时比较当前分数是否比历史最高分数高，若是则覆盖历史最高分数并显示。

3.3.7 礼物模块设计

礼物分为四种，分别是大招礼物、伤害礼物、子弹列数礼物和加血礼物。礼物会根据当前积分随机添加礼物。各礼物属性如表 3.4 所示。

表 3.4 礼物属性表

礼物	功用	出现概率
大招礼物	增加大招数目（最大为 1）	3/10
伤害礼物	增加每个子弹伤害	3/10
子弹列数礼物	增加子弹列数（最大为 3）	1/5
加血礼物	增加主角 100 血量	1/5

算法如下：

1．渲染礼物图片，添加礼物到礼物数组中并且随机添加一种礼物到场景中。

2．礼物从屏幕顶端移动至屏幕底部。

3．判断礼物是否与主角相撞，若是为主角增加相应功用，并从礼物数组中删除该礼物。否则执行下一步。

4．若礼物位置超过屏幕底部，则从礼物数组中删除该礼物。

3.3.8 关主模块设计

关主模块分为闯关模式关主模块和关主模式关主模块。其中闯关模式有三种 BOSS，关主模式有四种 BOSS。下面以关主模式关主模块为例。各 BOSS 属性如表 3.5 所示。

表 3.5 关主属性表

关主	关主血量	出现几率
关主一	18000	3/10
关主二	23000	3/10
关主三	28000	2/5
关主四（仅关主模式）	28000	2/5

渲染 BOSS 图片，添加 BOSS 图片至 BOSS 数组中并且随机添加一个 BOSS 到场景中。BOSS 出现后开始发射子弹。

1．判断子弹与主角是否相撞，若是则执行主角模块。否则进行下一步。

2．判断 BOSS 与主角是否相撞，若是则执行主角模块。

3．判断 BOSS 是否摧毁，若是从 BOSS 数组中删除该 BOSS。并且跳转至结束场景。

3.3.9 游戏模式模块设计

游戏模式分为闯关模式、无尽模式和 BOSS 模式。闯关模式相当无尽模式会有三个关卡，每个关卡都有一个关主，通过每关后会进入下一关。BOSS 模式相对于闯关模式，区别是 BOSS 模式只会出现 BOSS 精灵，而且 BOSS 模式会比闯关模式多一种 BOSS。

3.3.10 碰撞检测模块设计

碰撞检测模块包括有主角子弹与敌机之间的碰撞检测、主角与敌机之间的碰撞检测、主角与 BOSS 子弹之间的碰撞检测、主角与 BOSS 之间的碰撞检测。

Cocos2d-x 中每个精灵自带一个碰撞矩形，如果两精灵矩形有交集的话说明两精灵发生了碰撞。如图 3.1 所示。

图 3.1 碰撞检测示例图

3.3.11 音效模块设计

游戏需添加静音按钮以方便有的玩家有不想开音乐的情况。并且游戏需为每个场景加载不同的背景音乐，需添加了子弹发射的音效以及敌机爆炸的音效等。

3.3.12 游戏帮助模块设计

添加帮助图片以帮助玩家更快的了解本游戏。

3.4.1 子弹渲染优化

在游戏的绘制渲染中，往往消耗很多资源和内存，当绘制精灵数量越多，游戏的卡顿会很明显，游戏中的子弹又是会很多，如果每个子弹都进行一次渲染，游戏会出现明显卡

顿。为了优化和提升渲染效率，Cocos2d-x 为我们提供了 SpriteBatchNode。

SpriteBatchNode 是批处理绘制精灵，主要是用来提高精灵的绘制效率的，需要绘制的精灵数量越多，效果越明显。因为 cocos2d-x 采用 opengl es 绘制图片的，opengl es 绘制每个精灵都会执行：open-draw-close 流程。而 SpriteBatchNode 是把多个精灵放到一个纹理上，绘制的时候直接统一绘制该 texture，不需要单独绘制子节点，这样 opengl es 绘制的时候变成了：open-draw()-draw()…-draw()-close()，节省了多次 open-close 的时间。SpriteBatchNode 内部封装了一个 TextureAtlas（纹理图集，它内部封装了一个 Texture2D）和一个 Array（用来存储 SpriteBatchNode 的子节点：单个精灵）。注意：因为绘制的时候只 open-close 一次，所以 SpriteBatchNode 对象的所有子节点都必须和它是用同一个 texture（同一张图片）。

3.4.3 数据存储优化

对于数据存储 Cocos2dx 提供了 SQLite、JSON、XML 解析等几种技术。由于游戏中需要处理的数据并不多，为了不给游戏带来不必要的游戏消耗，本游戏采用的是 XML 存储技术。XML 存储技术通过 UserDefault()函数来获取和存储本地数据。

3.5 系统中类的设计

Cocos2dx 引擎设计有几个基本概念：Direct（导演）、Scene（场景）、Layer（层）、

Sprite（精灵）。导演在每个游戏中只有唯一一个，导演负责对场景进行调度。场景可以有一个或多个层组成。游戏中的主角和战机等都是精灵。具体调度如图 3.2 所示。

由上所示，游戏中会有很多场景，场景中又有多个层，这就导致了游戏中类较多。由于篇幅有限，再次我们把所有的类分组，并只介绍几个比较主要。分别是：军火库组、音效组、子弹组、控制层组、敌机组、游戏帮助组、游戏运行组、主角战机组、结果显示组、选关组、礼物组、欢迎界面组。

1．军火库组包括：

军火库场景。继承 Scene 类，主要用于加载军火库层。

军火库层。继承 Layer 类，主要负责加载军火库背景、三种主角及其参数。

2．音效组包括：

音效层。继承 Layer 类，主要负责加载游戏中的背景音乐和音效加载、是否静音等。

3．子弹组包括：

主角子弹属性类。主要负责加载主角游戏的属性：子弹伤害、等级等。

主角子弹层。继承 Layer 类；重写 Layer 类的 init()函数，实现对子弹以及大招特效进行批处理渲染；控制主角子弹开始发射和停止发射；控制主角大招发射。控制主角战机子弹的伤害、列数、外观等属性。

关主子弹层。继承 Layer 类；重写 Layer 类的 init()函数；重写 Layer 类的 update 函数，函数每一帧都会执行，检测 BOSS 子弹是否会击中主角，更新主角血量显示；控制 BOSS 子弹发射；加载 BOSS 爆炸特效。

4．控制层组：

控制层。继承 Layer 类；重写 Layer 类的 init()函数，加载暂停/继续按钮，加载血量显示按钮，加载大招发射按钮，加载分数显示；更新主角血量函数；更新积分函数；

5．敌机组：

敌机属性类。初始化敌机血量。

敌机层。继承 Layer 类。重写 Layer 类 init()函数，批处理渲染敌机，初始化敌机血量；根据游戏模式添加相应的敌机，并设计敌机飞行路径，检测闯关模式下是否满足添加 BOSS 条件；无尽模式添加战机；关主模式添加 BOSS；设计 BOSS 移动路线；回收 BOSS 子弹以及战机；加载战机爆炸特效。

重写 Layer 类 update()函数，函数每帧执行一次，功能如下：

① 检测游戏是否达到通关条件。

② 检测主角子弹是否与敌机发生碰撞以及碰撞后的动作。

③ 检测主角是否与敌机相撞以及相撞后动作。

④ 检测 BOSS 子弹与主角之间碰撞以及碰撞后动作。

⑤ 检测 BOSS 与主角之间碰撞以及碰撞后动作。

6．游戏帮助组：

游戏帮助场景。继承 Scene 类；负责加载游戏帮助背景等。

游戏帮助层。继承 Layer 类；重写 init()函数；加载背景；加载游戏帮助页面以及左翻右翻按钮。

7．游戏运行组：

游戏场景。继承 Scene 类；负责加载敌机层、主角战机层、控制层、礼物层等；

游戏背景层。继承 Layer 类；负责加载游戏场景；负责根据游戏模式加载不同游戏背景。

8．主角战机组：

主角属性类。初始化主角战机血量。

主角战机层。继承 Layer 类；重写 init()函数，初始化主角战机并加载；防止主角战机模型超出屏幕以外。

9．结果显示组：

结果显示场景。继承 Scene 类；负责加载结果显示层。

结果显示层。继承 Layer 类。显示本场游戏挑战结果（WIN/LOSE）；显示重新开始和返回主菜单按钮；显示本场分数；显示历史最高分数。

10．选关组：

选关场景。继承 Scene 类；负责加载选择选关层。

选关层。继承 Layer 类；负责加载背景；加载进入各种模式以及关卡的按钮；加载军火库按钮。

11．礼物组：

礼物属性类。设置礼物类型。

礼物层。继承 Layer 类；重写 Layer 层的 init()函数，实现礼物渲染批处理。重写 Layer 层 update()函数，函数每帧执行，检测礼物是否与主角相撞；加载礼物与主角相撞后动画。

12．欢迎界面组：

欢迎界面场景。继承 Scene 类；负责加载欢迎层；

欢迎层。继承 Layer 类；加载背景；加载 LOGO；加载开始游戏和游戏帮助按钮；加载游戏难度设置；加载静音按钮。

如图 3.3 游戏类图所示。

下面对游戏中主要类以及类中函数进行说明：

1．敌人层类 EnemyLayer。其属性和函数如表 3.6 和表 3.7 所示。

表 3.6 EnemyLayer 属性

属性	类型	说明
getBossSprite	sprite*	返回游戏中 BOSS
baseEnemyAppearProbability	float	用来计算 Boss 是否出现
deltaEnemyAppearProbability	float	用来计算 Boss 是否出现
nowEnemyAppearProbability	float	用来计算 Boss 是否出现
enemyTextureName	vector<string>	缓存各敌人纹理
enemyFlyTime	vector<int>	存储各敌人飞行时间

enemyInitHP	Vector<int>	存储各敌人血量

图 3.3 游戏类图

表 3.7 EnemyLayer 类函数

函数	返回类型	说明
init()	virtual bool	对敌机纹理进行渲染、飞行速 度及时间进行设置
enemyMoveFinished(Node* pSender)	void	回收敌机
startAddEnemy()	void	开始添加敌机
stopAddEnemy()	void	停止添加敌机
Update(float useless)	void	监测游戏各精灵是否发生碰撞，监测游戏是否达到结束标准，
addBossSprite()	void	闯关模式中添加 BOSS
changeSceneCallBack(float useless)	void	跳转结束场景
setBossWarningOn()	void	加入 BOSS 出现前警告声
bossStartMove()	void	设计 BOSS 移动路径
addEnemySprite(float useless)	void	添加闯关模式敌机精灵
addEnemySpriteEndless(float useless)	void	添加无尽模式敌机精灵
addEnemyBossSprite(float useless)	void	添加关主模式敌机精灵

2. 主角飞机类 PlaneLayer。PlaneLayer 属性和函数如表 3.8 和表 3.9 所示。

表 3.8 飞机类属性表

属性	类型	说明
myPlane	Sprite*	当前飞机精灵实例
initHP	int	主角飞机初始血量
winSize	Size	记录当前屏幕大小

表 3.9 飞机类函数表

函数	返回类型	说明
getMyPlane()	Sprite*	返回当前主角飞机实例
getInitHP()	int	返回当前主角飞机血量

init() override	virtual bool	游戏初始化、添加背景等
onTouchBegan(cocos2d::Touch*touch, cocos2d::Event *unused_event)	virtual bool	判断鼠标是否点击屏幕
onTouchMoved(cocos2d::Touch *touch, cocos2d::Event *unused_event)	virtual void	若手指触摸到手机屏幕且在飞机有效范围移动飞机

3. 子弹类 BulletLayer。其属性和函数如表 3.10 和表 3.11 所示。

表 3.10 子弹类属性

属性	类型	说明
allBullet	Vector<Sprite*>	管理游戏中所有子弹
eachBulletDamage	int	每个子弹伤害量
bulletTextureName	vector<string>	记录每种子弹在素材（.plist）中的名字
bulletBatchNodeVector	vector:SpriteBatchNode*>	对子弹进行批处理渲染
nowBulletLevel	int	记录当前子弹伤害量
BulletNum	int	记录当前子弹列数

表 3.11 子弹类函数

函数	类型	说明
bulletMoveFinished(Node* pSender)	void	回收子弹
stopShooting()	void	停止射击
setBulletLevelUP()	void	增加子弹伤害
setBulletNumUp()	void	增加子弹列数
launchBigBomb()	void	发射
init()	virtual bool	对子弹纹理进行渲染、飞行速 度及时间进行设置并显示在 屏幕上
startShooting(int BulletNum)	void	开始设计子弹（个数）
HeroBulletOne(float useless)	void	一排子弹

HeroBulletTwo(float useless)	void	两排子弹
HeroBulletThree(float useless)	void	三排子弹

4. 欢迎界面类 welcomeButtonLayer。欢迎界面类属性和函数如表 3.12 和表 3.13 所示。

表 3.12 欢迎界面类属性表

属性	类型	说明
buttonSoundOn	MenuItemSprite*	打开声音按钮
buttonSoundOff	MenuItemSprite*	关闭声音按钮
buttonGameEasy	MenuItemSprite*	设置游戏难度为简单按钮
buttonGameNormal	MenuItemSprite*	设置游戏难度为正常按钮
buttonGameHard	MenuItemSprite*	设置游戏难度为困难按钮

表 3.13 欢迎界面类函数表

函数	返回类型	说明
init()	virtual bool	设置游戏背景以及添加声音按钮和游戏难度按钮
startGameButtonCallback (Ref* pSender)	void	玩家点击开始按钮回调函数
helpGameButtonCallback(Ref* pSender)	void	玩家点击游戏帮助按钮时回调函数

5. 礼物类 UFOLayer。结果显示类属性和函数如表 3.14 和 3.15 所示

表 3.14 礼物类属性表

属性	类型	说明
giftTextureName	vector<string>	存储各种礼物在纹理包中的名称
giftFlyTime	vector<int>	每个礼物的飞行时间

winSize	Size	保存当前窗口大小
sequenceFront	Sequence*	主角飞机得到礼物后动画
sequenceBack	Sequence*	主角飞机得到礼物后动画
allGift	Vector<Sprite*>	存储当前礼物以便回收

表 3.15 礼物类函数表

函数	返回类型	说明
init()	virtual bool	初始化纹理名称容器和飞行时间容器
giftMoveFinished (Node* pSender)	void	回收礼物
update(float useless)	void	判断礼物是否与主角飞机相撞以及相撞后动作
showAnnotation(Sprite* gift)	void	礼物与主角飞机相撞否动画

6. 游戏控制界面类 CotrolLayer。其属性和函数如表 3.16 和表 3.17 所示。

表 3.16 游戏控制界面类属性表

属性	类型	说明
score	int	记录当前分数
scoreLabel	Label*	显示“积分”标签
pauseButtonItem	MenuItemSprite*	暂停\继续按钮精灵
launchButtonItem	MenuItemSprite*	大招发射按钮精灵
HPIndicator	ProgressTimer*	血条显示
pauseButton	Menu*	暂停\继续按钮
launchButton	Menu*	大招按钮

表 3.17 游戏控制界面函数表

函数	返回类型	说明
updateScore()	void	更新分数
menuPauseCallback(cocos2d::Ref* pSender)	void	暂停/继续按钮回调函数
init()	virtual bool	各种精灵的初始化
menuLaunchCallback(cocos2d::Ref* pSender)	void	发射大招按钮回调函数

onKeyReleased(EventKeyboard::KeyCode keycode, Event* event)	virtual void	监听键盘

3.6 系统主要部分顺序图

在本游戏关键的几个层（Layer）中都会重写 Layer 类中的 update()函数，update()函数会在游戏的每一祯中执行。游戏中有三种模式：闯关模式、无尽模式、关主模式。每种模式都有一套顺序图。这里以闯关模式中的顺序图为例。

在游戏的每一帧中，先在游戏中添加主角飞机、敌机；然后主角飞机开始发射子弹；判断子弹是否与敌机碰撞，若碰撞，计算敌机血量，若敌机摧毁，回收敌机并增加分数；判断敌机与主角飞机是否相撞，若敌机摧毁，回收敌机并增加分数，若主角摧毁，跳转结束场景；添加礼物，判断礼物与敌机是否相撞，若相撞则增加相应的效果；达到条件后增加 BOSS，BOSS 发射子弹，判断 BOSS 子弹是否与主角飞机相撞，同时判断 BOSS 是否与主角飞机相撞；监测玩家是否按下暂停按钮，点击之后执行相关操作。最后，游戏重新绘制游戏帧。顺序图如下图 3.4 所示。

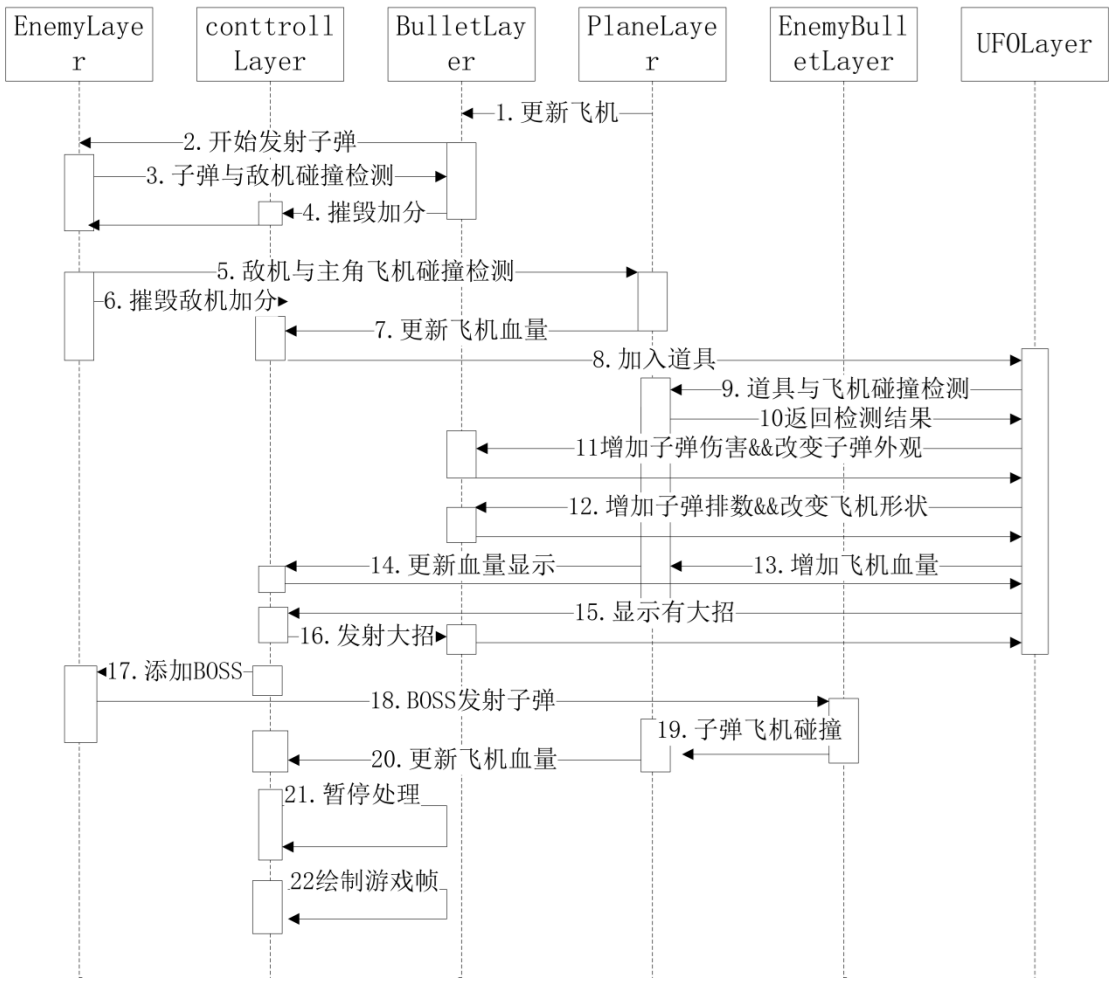

图 3.4 游戏顺序图

3.7 系统功能结构

本游戏的系统功能主要有主角管理、模式选择、暂停界面、子弹管理、结果显示、游戏帮助、分数存储、音效控制、碰撞检测、分数显示、礼物管理、敌机管理、血量显示、

大招显示等。其中模式选择分为无尽模式、闯关模式和关主（BOSS）模式；分数存储分为分数读取与分数更新；游戏帮助有玩法说明；礼物管理分为子弹排数礼物、子弹伤害礼物、加血礼物和大招礼物；敌机管理分为增加敌机、敌机回收、增加关主（BOSS）和关主（BOSS）回收；子弹管理分为主角飞机的子弹管理和关主（BOSS）的子弹管理。

4 系统实现

4.1 安装开发环境

1．安装 Python2.7 并配置环境变量，配置完成后打开 cmd 控制台，输入 python，如果出现如下提示，则说明 python 安装成功。如图 4.1 所示。

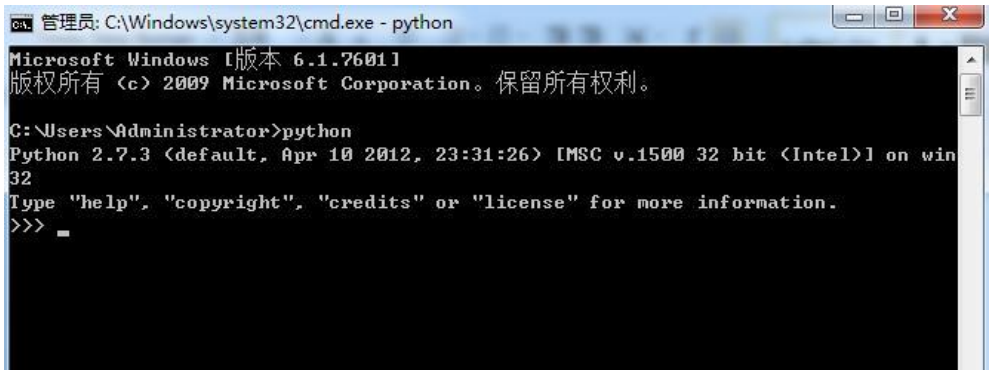

图 4.1 Python 安装成功图

2．安装 cocos2d-x-3.3

cocos2d-x-3.3 项目无需安装，下载 cocos2d-x-3.3 文件并解压。

3．安装 MicrosoftVisualStudio2013 。一步步按提示安装即可。

4．创建项目

进入到目录 cocos2d-x-3.3/tools/cocos2d-console/bin/cocos.py

打开终端运行 cocos.py 脚本创建文件：

python cocos.py new Plane3 -p com.coco2dx.org -l cpp -d E:\cocos_2d

等待完成即可。完成界面如图 4.2 所示：

图 4.2 生成项目完成图

5．然后在目录 E:\cocos_2d 中打开项目可以看见各个快平台的文件夹。

如图 4.3 所示。

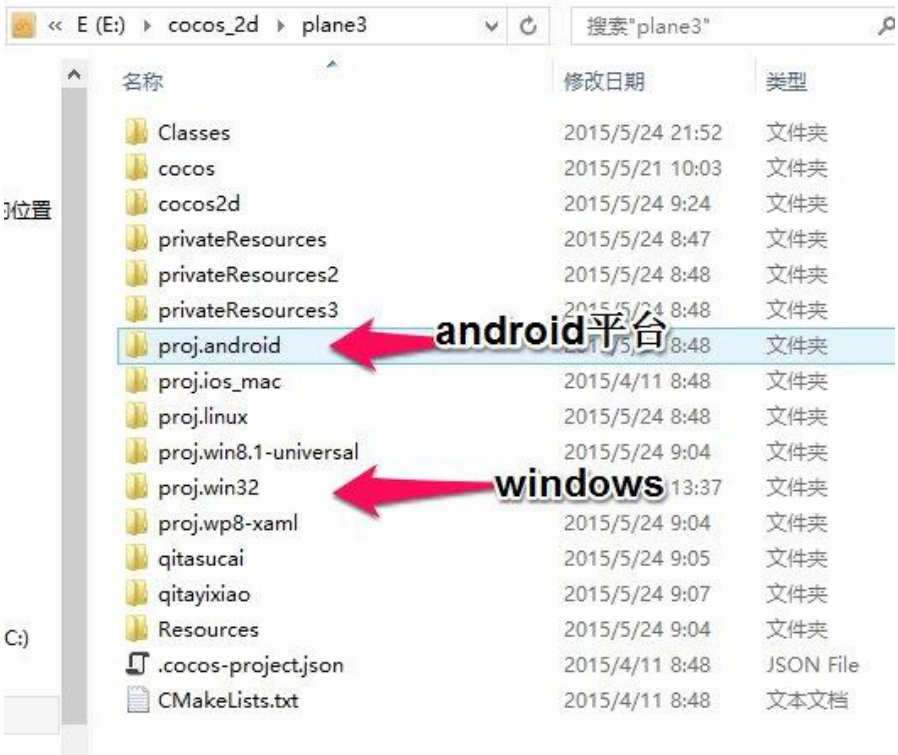

图 4.3 项目内部图

4.2 游戏关键部分实现

4.2.1 地图无限滚动

游戏让背景不断的向下滚动来实现在视觉上主角飞机看起来是不断向上飞的状态。

为了使主角飞机有不断向前飞行的效果，本游戏采用使地图背景不断逆向滚动的形式来达到主角飞机不断前行的效果。

其实就是两张相同的背景图片，然后同时向下滚动，当一张图片完全出视野后，就把它调到最上面，于此同时另一张背景图片出现，形成两个图片交替出现。的效果。不过，一般为游戏中我们都感觉像是一张图片，那是因为两张图片的头尾连接处是连起来的。

如图 4.4 所示。

具体代码如下：

```
background1->setPositionY(background1->getPositionY()-backgroundMoveSpeed)
background2->setPositionY(background1->getPositionY()+
            background1->getContentSize().height*2 - backgroundMoveSpeed);
if(background2->getPositionY() < 0){
       background1->setPositionY(0);
   }
```

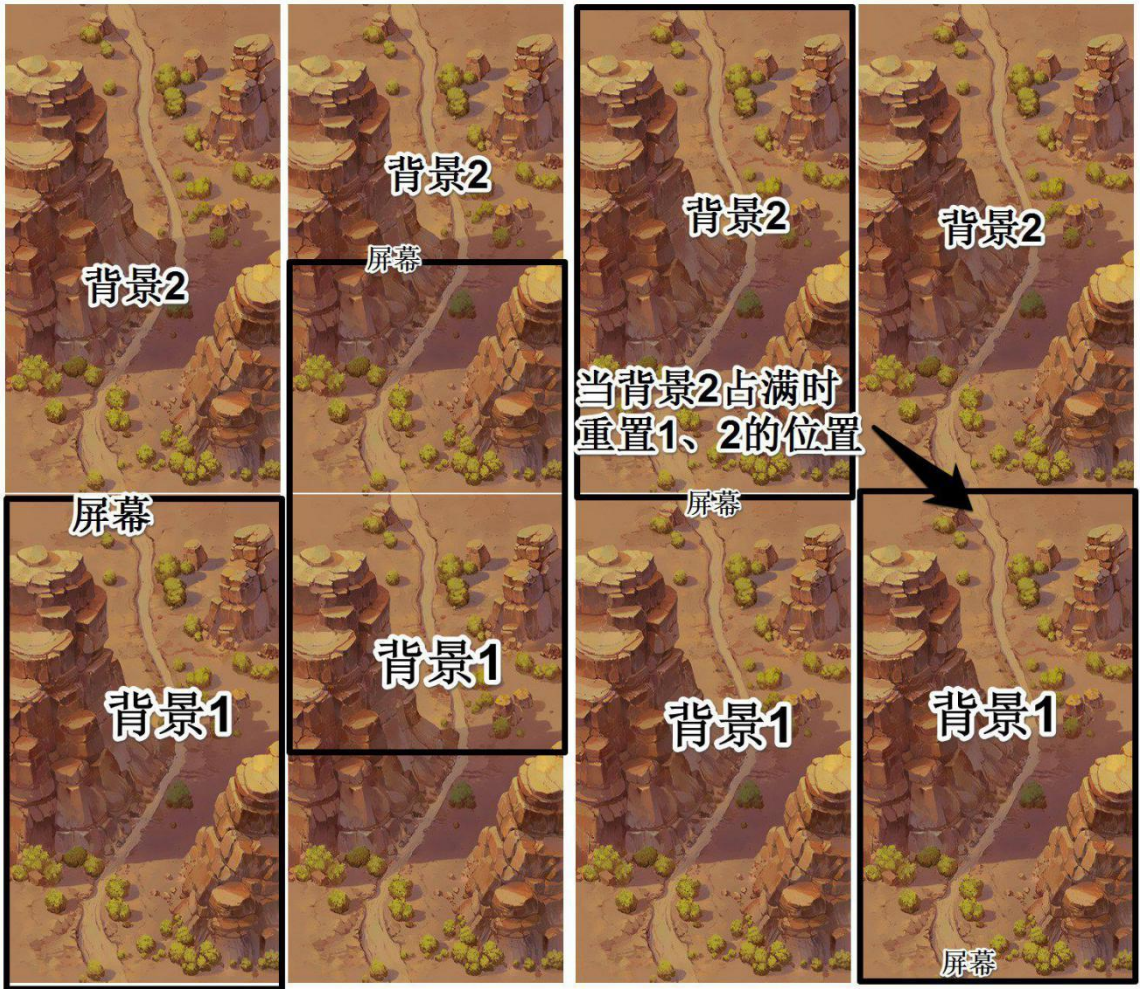

图 4.4　地图无限滚动原理图

4.2.2 防止主角飞机超出屏幕外

在进行游戏时，如果主角飞机超出屏幕外，会严重影响玩家的视觉体验。游戏实现时必须避免这种情况。实现代码如下：

```
//飞机超出左边边界，设置飞机 X 轴位置为 0
if((myPlane->getPositionX()-myPlane->getContentSize().width/4< 0)){
    myPlane->setPositionX(0 + myPlane->getContentSize().width
                                                        /  4);
}
//飞机超出右边边界，设置飞机 X 轴位置为右边界
else if ((myPlane->getPositionX() +
        myPlane->getContentSize().width / 4) >winSize.width)
{
    myPlane->setPositionX(winSize.width-myPlane->
                                getContentSize  ().width / 4);
}
//飞机超出下边边界，设置飞机 X 轴位置为下边界。
```

```
if (myPlane->getPositionY() < 0) {
      myPlane->setPositionY(0);
}
//飞机超出上边边界，设置飞机 X 轴位置为上边界
else if (myPlane->getPositionY()+
         myPlane->getContentSize().height/2>winSize.height)
{
    myPlane->setPositionY(
            winSize.height-myPlane->getContentSize().height/2);

}
```

4.2.3 控制主角飞机移动

要在游戏层使用单点触摸首先需要在初始化中设置可触摸，语句为 setTouchEnabled(true)。设置可触摸后，我们还要对 Cocos2d-x 中的两个回调函数进行重写。

```
virtual bool ccTouchBegan(CCTouch *pTouch, CCEvent *pEvent) override;//当触摸到屏幕一瞬间会调用
virtual void ccTouchMoved(CCTouch *pTouch, CCEvent *pEvent)override;//手指在屏幕上移动时调用
```

对两函数重写后代码如下：

```
bool PlaneLayer::onTouchBegan(ouch *touch, Event *unused_event) {
    return true;//表示当前层接收触摸事件处理
}

void PlaneLayer::onTouchMoved(Touch *touch, Event *unused_event) {
    //将飞机坐标位置设为手指抬起位置
    myPlane->setPosition(myPlane->getPosition() + touch->getDelta());
}
```

4.2.4 碰撞检测

碰撞检测是游戏中核心内容，包括：

1. 主角飞机与敌机之间的碰撞检测；
2. 主角飞机子弹与敌机之间的碰撞检测；
3. 主角飞机与 BOSS 之间碰撞检测；
4. 主角飞机与 BOSS 子弹之间碰撞检测；
5. 主角飞机与礼物之间碰撞检测；

碰撞检测代码实现如下：

```
//通过调用精灵自带函数的 intersectsRect()函数来实现
```

```
if (enemy->getBoundingBox().intersectsRect(getMyPlane()->getBoundingBox())
{
    .........
}
```

4.2.5 敌机 AI 的设定

为了使游戏更加生动好玩，游戏为战机设定了简单的飞行路线。

1. 敌机一的飞行路径：从左上侧出现，从右下侧消失。

敌机一的实现，用到了贝赛尔曲线，贝塞尔曲线是应用于二维图形应用程序的数学曲线。曲线的定义有四个点：起始点、终止点（也称锚点）以及两个相互分离的中间点。定好起止点和终止点后，只要改变两个中间点就可以改变曲线。

```
ccBezierConfig tr0;
enemySprite->setPosition(Vec2(0, 450));//起始点
tr0.controlPoint_1 = Vec2(200, 300);//中间点 1
tr0.controlPoint_2 = Vec2(140, 200);//中间点 2
tr0.endPosition = Vec2(0, 20);//终止点
ActionInterval* bezierFoward = BezierTo::create(3.f, tr0);//创建运行的贝塞尔曲线
ActionInterval* forwardBy = RotateBy::create(3.f, 90); // 第二个参数：如果是正数则是顺时针，否则逆时针
Spawn* spawn = Spawn::create(bezierFoward, forwardBy, NULL);//创建合成动作
……
```

2. 敌机二的飞行路径：从屏幕上方冲向主角飞机。其代码如下。

```
//添加敌机精灵
Sprite* enemySprite =
                    createWithSpriteFrameName(enemyTextureName[3].c_str());
//设置敌机从屏幕上方随机飞出
int randomX = CCRANDOM_0_1()*(winSize.width
                                    - enemySprite->getContentSize().height)
                                + enemySprite->getContentSize().height / 2;
 enemySprite->setPosition(randomX
                                    - enemySprite->getContentSize().width/5,
                 winSize.height + enemySprite->getContentSize().height/3);

float x = getMyPlane()->getPosition().x;
float y = getMyPlane()->getPosition().y;
float a = randomX - x;
```

```
float b = winSize.height - y;

//弧度转角度，让敌机“冲向”主角
float radians = atanf(a / b);
float mDegree = CC_RADIANS_TO_DEGREES(radians);
enemySprite->setRotation(mDegree);
//添加敌机
this->addChild(enemySprite);
//设置结束地点
float endX = randomX - (a / b)*winSize.height;//winSize.width
float endY = 0;

float flyVelocity = 250;//运行速度
float flyLen = sqrt((winSize.width - endX)*(winSize.width - endX)
                  +  (winSize.height -   endY)*(winSize.height - endY));
float realFlyDuration = flyLen / flyVelocity;//实际飞行的时间
//添加动作
FiniteTimeAction* enemyMove = MoveTo::create(realFlyDuration,
                                                    Point(endX, endY));
FiniteTimeAction* enemyRemove =

CallFuncN::create(CC_CALLBACK_1(EnemyLayer::enemyMoveFinished
                                                          , this));
Action* enemyAction = Sequence::create(enemyMove, enemyRemove,
                                                            NULL);
enemySprite->runAction(enemyAction);
```

3. 敌机三的飞行路径：从屏幕上方飞至屏幕下方。代码较简单，此处省略。
4. 敌机四的飞行路径：三架并列从屏幕上方飞至屏幕下方。代码较简单，此处省略。
5. 敌机五的飞行路径：三架并排从屏幕左方飞至屏幕右方。代码较简单，此处省略。

4.2.6 主角飞机子弹

主角发射一个子弹，其实现代码如下：

```
//添加子弹精灵
Sprite* bullet =
  createWithSpriteFrameName(bulletTextureName[nowBulletLevel]);
Point planePosition =static_cast<GameScene*>(
                                  this->getParent())->getPlaneLayer()->
                                  getChildByName("PLANE")->getPosition();
```

```
//设置子弹出现位置
Point bulletPosition = Point(planePosition.x, planePosition.y +
static_cast<GameScene*>(
            this->getParent())->getPlaneLayer()->getChildByName("PLANE")
                                              ->getContentSize().height);

bullet->setPosition(bulletPosition);
bullet->setUserData(new                    BulletUserData(eachBulletDamage,
nowBulletLevel));
bullet->setScale(0.5f);
allBullet.pushBack(bullet);
this->bulletBatchNodeVector[nowBulletLevel]->addChild(bullet);
//设置子弹飞行时间
float bulletFlyLenth = Director::getInstance()->getWinSize().height -
                bulletPosition.y + (bullet->getContentSize().height / 2);
float bulletFlySpeed = 900 / 1;
float bulletFltTime = bulletFlyLenth / bulletFlySpeed;
//设计子弹飞行动作
FiniteTimeAction* bulletMove = MoveTo::create(bulletFltTime,
   Point(bulletPosition.x,Director::getInstance()->getWinSize().height +
                                  bullet->getContentSize().height / 2));
FiniteTimeAction* bulletRemove = CallFuncN::create(

CC_CALLBACK_1(BulletLayer::bulletMoveFinished, this));
//加入子弹动作
auto bulleAction = Sequence::create(bulletMove, bulletRemove, NULL);
bullet->runAction(bulleAction)
```

主角发射两排子弹。就是在一排子弹基础上加一排子弹然后改一下位置。代码较简单，此处省略。

主角发射三排子弹。除了多出一排子弹外，还要设计子弹的偏转度。说明如图 4.3 所示。

可以得到主角位置为(X,Y)，根据图示可以得到右排子弹的结束坐标(h+h*tan(angle),height)。同理可得左排子弹结束坐标(h-h*tan(angle),height)

实现偏转都代码如下：

```
//angle 是偏转角度
float bulletFlyLenth3 = planeHeight / cos(angle);
float bulletFlySpeed = 850 / 1;
float bulletFltTime1 = bulletFlyLenth1 / bulletFlySpeed;
//设计子弹飞行动作
```

```
FiniteTimeAction* bulletMove1 = MoveTo::create(bulletFltTime1,
                          Point(bulletPosition1.x-planeHeight * tan(angle),
                            Director::getInstance()->getWinSize().height +
                                  bullet1->getContentSize().height / 2));
FiniteTimeAction* bulletRemove1 =CallFuncN::create(CC_CALLBACK_1
                                  ( BulletLayer::bulletMoveFinished, this));
```

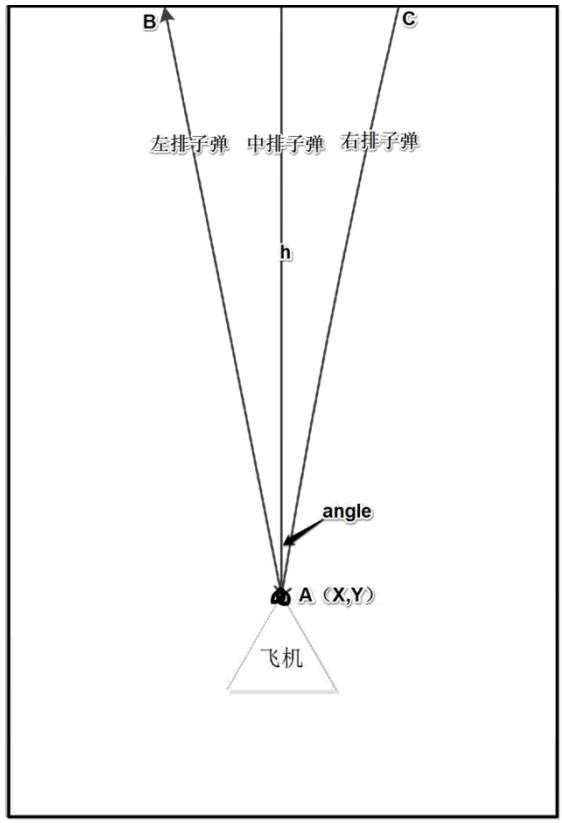

图 4.3 三排子弹示例图

4.3 软件移植到 Android

1. 安装 ADT(Software Development Kit)，可以理解为开发工具包集合，是整体开发中所用到的工具包。

2. 安装 JDK。

3. 安装 ANT。

4. 安装 Eclipse 软件。

5. 用 Eclipse 软件导入现有工程。

6. 进入 proj.android 文件夹下的 jni 文件夹中，打开 Android.mk 文件。在其中添加游戏源代码路径。

7. 把 cocos2dx\platform\android\java\src 下的 org 文件夹复制到工程文件 android 文件夹下的 src 中。

8. 把资源文件复制到 proj.android 文件夹下的 assets 中。

9. 运行项目。

5.2 游戏帮助

5.2.2 游戏玩法

游戏模式分为关主模式、无尽模式和闯关模式。

闯关模式分为三个关卡，每个关卡开始不断有敌机飞来挑战主角飞机，每个关卡会添加一种新的敌机。摧毁敌机增加积分，积分每满 1000 分，会随机飞来一种礼物，主角飞机得到礼物后会增加相应效果。当达到某种条件后，会出先 BOSS。每一关都有一个属于自己的关主，关主出现后会发射密集子弹。击败关主决定是否进入下一关或者重新开始。

无尽模式只有一个关卡。关卡开始后不断由敌机飞来。同闯关模式一样，积分每满 1000 分，会随机飞来一种礼物，主角飞机得到礼物后会增加相应效果。随着积分的增加，游戏难度会增加，直至主角飞机被摧毁。

关主模式中没有普通敌机的加入，只有闯关模式中的关主外加另一种关主。关主随机出现。关主摧毁增加积分，游戏难度随着积分增加而增加.