

21级软件工程专业 初级软件实作课

实作报告

2022 ~ 2023 学年 第二学期

**教学单位：软件学院**

**课程名称：《初级软件设计实作》**

**实作名称：飞机大战（aircraft\_battle）**

**完成日期：2023年7月**

**组长：曾世纪**

**课题组成员：卢辉骏 陈梓杰 曹洋 蒋春鸿**

# 目录

[一、实作项目简介 1](#_Toc28719)

[二、前期准备 1](#_Toc17123)

[三、项目设计 2](#_Toc14722)

[3.1总体架构设计（系统结构图/分析类图） 2](#_Toc28110)

[3.2类设计（类图） 3](#_Toc10680)

[四、系统实现 5](#_Toc2428)

[4.1界面（com.tedu.show） 5](#_Toc5589)

[4.2控制（com.tedu.controller） 9](#_Toc31565)

[4.3资源管理（com.tedu.element） 15](#_Toc17165)

[五、游戏界面展示 20](#_Toc31810)

[六、总结 21](#_Toc29817)

[6.1设计亮点 21](#_Toc24772)

[6.2不足 21](#_Toc7105)

[6.3反思 22](#_Toc12057)

## 一、实作项目简介

《**飞机大战（aircraft\_battle）**》是一款射击类游戏。此游戏背景设定在未来，地球遭受了外星人的入侵。这些外星人来自未知的星球，身体强壮，拥有高超的科技水平和战斗能力。他们来到地球的目的不明，整个地球陷入了一片恐慌和混乱之中。

我们观察到，市面上有不少传统的射击游戏，但除了战地2042、Apex英雄等射击游戏外，部分设计游戏制作较为简朴，画面比例不够协调，玩家没有坚持使用的欲望；而战地2042、Apex英雄等往往更让玩家沉迷，从而失去设计射击游戏的初衷。这款游戏的设计理念是刺激和有趣的射击游戏，让玩家在游戏中体验到射击和掌控飞船的快感。游戏中的外星人将不断出现在不同的位置，玩家需要操纵飞船迅速反应并射击外星人所在的飞船，取得更高的分数。

图1 战地2042（左）& Apex英雄（右）

本游戏是一款曾风靡全国的小游戏《合金弹头》的仿作，游戏类型为射击类游戏，操作简单，趣味性强。玩家将扮演一名反抗军指挥官，驾驶自己的飞机与外星人展开激烈的战斗。

游戏共有五个关卡，在每一个关卡中，外星人星球的大boss都会派出众多的敌机对我方战机进行攻击，玩家需要快速驾驶自己的飞船，去击杀敌机，（每一关卡中只有在所有的敌机都被击杀完毕后才会出现大boss）在我方战机血量为负之前消灭所有敌机和大boss的残存血量，保护地球上所有的生命。

项目使用Java编程语言，采用**MVC设计模式**。

## 二、前期准备

* Eclipse
* JDK1.8
* 华为云托管平台
* 华为云CodeArt小组创建

在实作之前我们进行了一周的初级软件实作学习，充分了解了**MVC设计模式**，在课堂上，我们紧跟老师的脚步，积累了开发经验。组员们积极交流，确立分工，查找素材。

以下为项目组的成员分工和需求规划：

组长：

* 曾世纪，敌机随机生成，boss出现条件判定，道具掉落，游戏暂停

组员：

* 曹洋：元素的碰撞，boss血条
* 蒋春鸿：代码整合，文档
* 陈梓杰：关卡切换，敌机、boss和地图的显示
* 卢辉骏：答辩ppt制作，视频录制



图2 需求规划图

## 三、项目设计

### 3.1总体架构设计（系统结构图/分析类图）

本项目实现的核心功能就是击杀外星人飞船和大boss，因此需要拆分所需要的要素并一一实现，简单框架如2所示。

一则，在飞机的升级中，每一关卡的敌机模型都是不一样的，而且每一关卡都比上一关卡（如果有）的敌机击落难度更大。

二则，每一关卡都配置了不同的图片，视觉上给玩家形成了不同的震撼。下图中的从左至右分别为关卡1~关卡5的背景图片，所有图片均放置在image\background文件夹中。

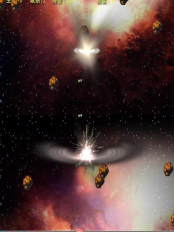


图3 游戏关卡背景图

三则，有三种道具的掉落。其中，黄色的“0”道具代表移动速度加快，黄色的“2”道具代表大boss血量减少到当前血量的50%，绿色的“1”道具代表增加我方的战机血量（可以理解成敌机被击毁并解体后敌机内部的能量被释放出来，供我方战机拾取利用）。

四则，敌机的分布和移动方向是随机的，能给游戏本身增加一定难度；大boss的血条在被我方战机发射的子弹击中后会降低，同时我方战机的血量同样也会同等程度地降低。

五则，我方战机与敌机碰撞后，敌机会自动消失（可以理解成战机的战斗力可以击败敌机），同时，玩家的战机也会相应的掉血，但战机不能直接撞毁大boss。

注：MVC设计模式中，M是指模型Model，V是视图View，C则是控制器Controller，使用MVC的目的是将M和V的实现代码分离，从而使同一个程序可以使用不同的表现形式。

以下为MVC设计模式的界面框架图。

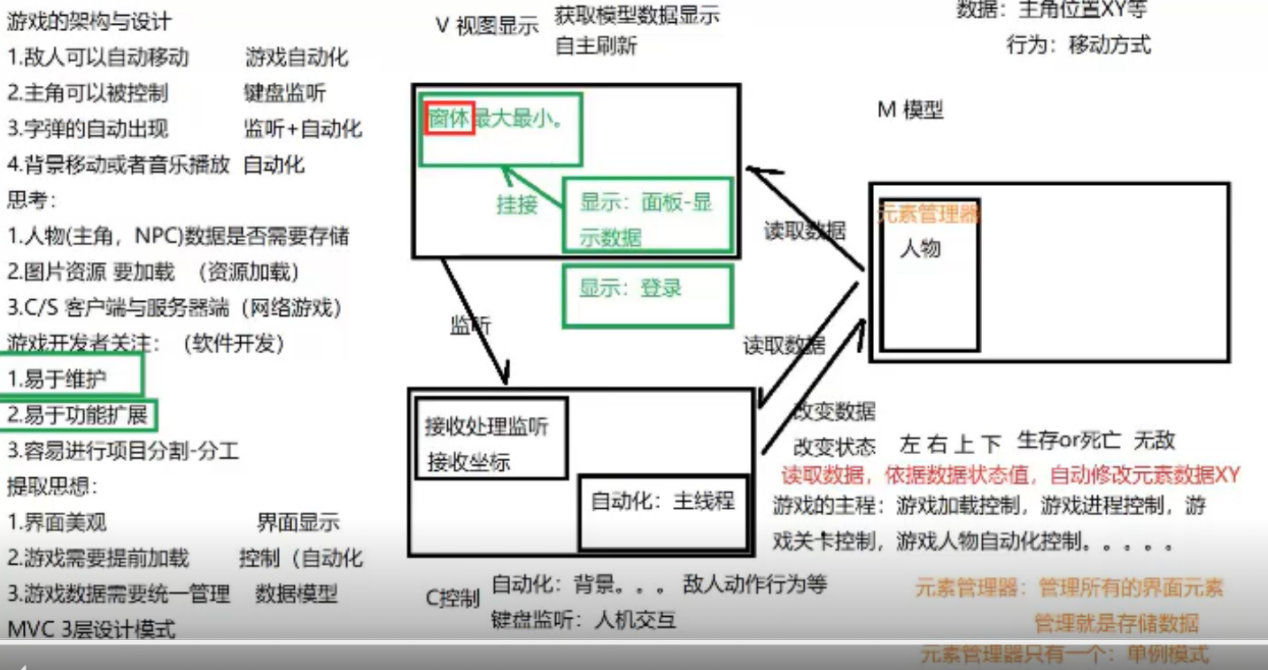


图4 MVC界面框架图

### 3.2类设计（类图）

设计时采用**分包开发**的思路，按功能模块进行开发包的划分。

* **com.tedu.controller包**

这里主要实现控制功能，包括主线程和监听。

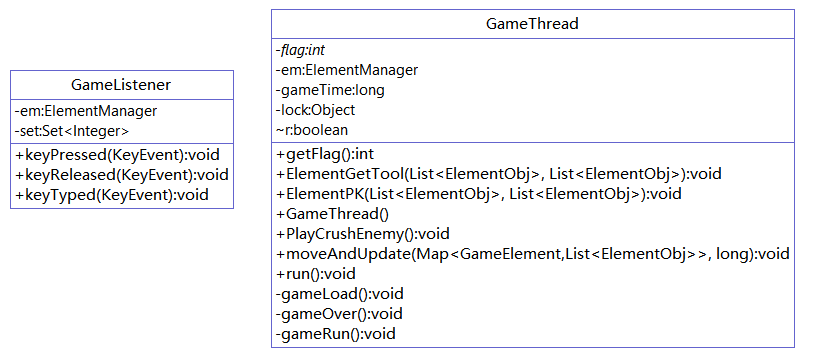


图5 controller包URL图

* **com.tedu.element包**

这是模型类，里面定义了所有的元素类。

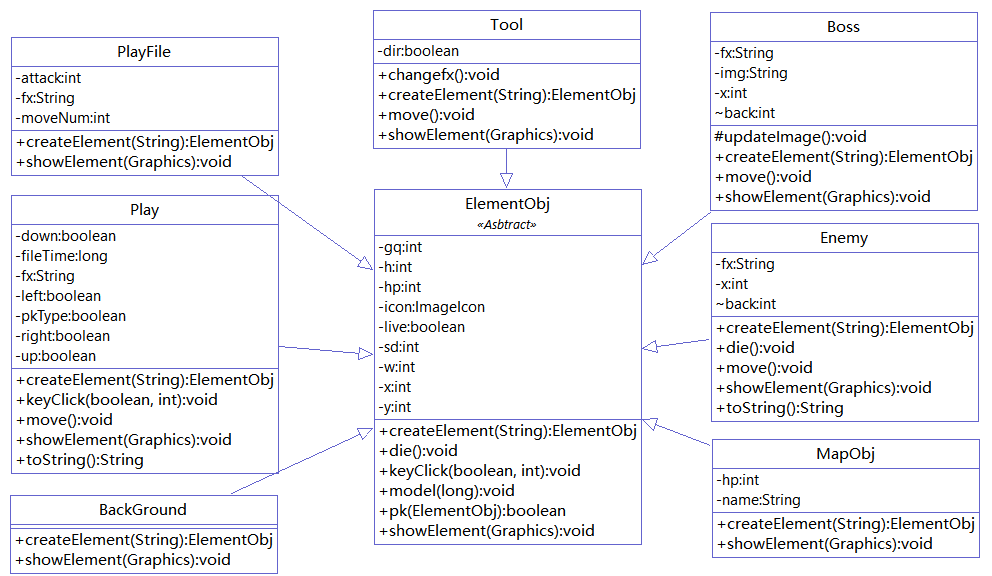


图6 element包URL图

* **com.tedu.game包**

这里主要实现启动游戏。

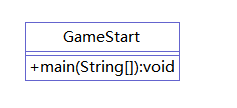


图7 game包URL图

* **com.tedu.manager包**

这是元素管理器和加载器，主要使用**工厂模式**。

注：**工厂模式**是Java中最常用的设计模式之一，提供了一种将对象的实例化过程封装在工厂类中的方式。通过使用工厂模式，可以将对象的创建与使用代码分离，提供一种统一的接口来创建不同类型的对象。在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。

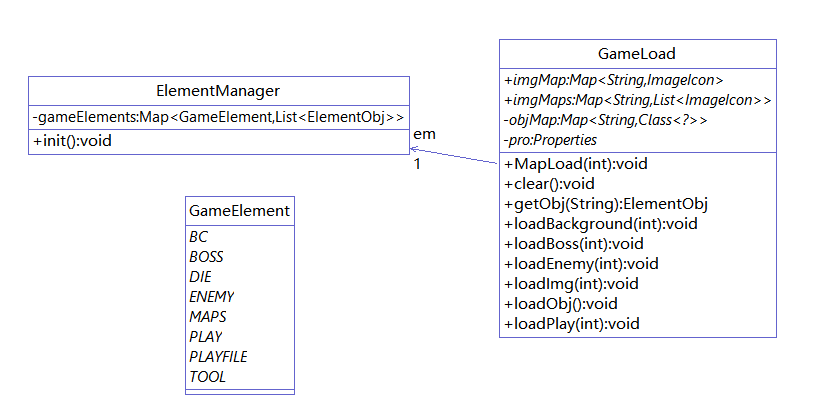


图8 manager包URL图

* **com.tedu.show包**

这里主要是界面的显示功能包。

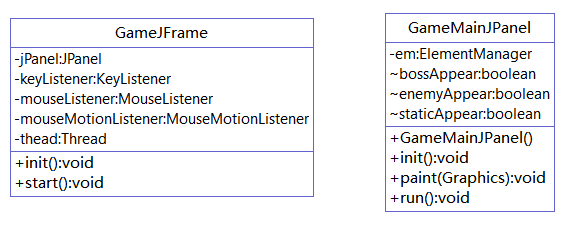


图9 show包URL图

* **com.tedu.text包**

这里存放配置文件，主要是游戏的配置。

* **image文件夹**

这里存放图片音乐。

## **四、系统实现**

以下选择**部分**重要的包中重要的代码进行分析与展示。

### **4.1界面（com.tedu.show）**

* **GameJFrame.java**

1. **package** com.tedu.show;
3. **import** java.awt.event.KeyListener;
4. **import** java.awt.event.MouseListener;
5. **import** java.awt.event.MouseMotionListener;
7. **import** javax.swing.JFrame;
8. **import** javax.swing.JPanel;
10. **public** **class** GameJFrame **extends** JFrame{
11. **public** **static** **int** GameX = 600;
12. **public** **static** **int** GameY = 600;
13. **private** JPanel jPanel =**null**;
14. **private** KeyListener keyListener=**null**;
15. **private** MouseMotionListener mouseMotionListener=**null**;
16. **private** MouseListener mouseListener=**null**;
17. **private** Thread thead=**null**;
19. **public** GameJFrame() {
20. init();
21. }
22. **public** **void** init() {
23. **this**.setSize(GameX, GameY);
24. **this**.setTitle("飞机大战（Aircraft\_Battle）");
25. **this**.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
26. **this**.setLocationRelativeTo(**null**);
27. }
28. **public** **void** addButton() {
30. }
31. **public** **void** start() {
32. **if**(jPanel!=**null**) {
33. **this**.add(jPanel);
34. }
35. **if**(keyListener !=**null**) {
36. **this**.addKeyListener(keyListener);
37. }
38. **if**(thead !=**null**) {
39. thead.start();
40. }
41. **this**.setVisible(**true**);
42. **if**(**this**.jPanel **instanceof** Runnable) {
43. Runnable run=(Runnable)**this**.jPanel;
44. Thread th=**new** Thread(run);
45. th.start();
46. System.out.println("是否启动");
47. }
48. }
50. **public** **void** setjPanel(JPanel jPanel) {
51. **this**.jPanel = jPanel;
52. }
53. **public** **void** setKeyListener(KeyListener keyListener) {
54. **this**.keyListener = keyListener;
55. }
56. **public** **void** setMouseMotionListener(MouseMotionListener mouseMotionListener) {
57. **this**.mouseMotionListener = mouseMotionListener;
58. }
59. **public** **void** setMouseListener(MouseListener mouseListener) {
60. **this**.mouseListener = mouseListener;
61. }
62. **public** **void** setThead(Thread thead) {
63. **this**.thead = thead;
64. }
65. }

变量GameX和GameY设置了游戏窗体的大小，jPanel定义了正在显示的面板，同时设置了变量代表了键盘监听、鼠标监听和游戏的主线程的判定。

* **GameMainJPanel.java**

1. **package** com.tedu.show;
3. **import** java.awt.Color;
4. **import** java.awt.Font;
5. **import** java.awt.Graphics;
6. **import** java.util.List;
7. **import** java.util.Map;
8. **import** java.util.Set;
10. **import** javax.swing.ImageIcon;
11. **import** javax.swing.JPanel;
13. **import** com.tedu.controller.GameThread;
14. **import** com.tedu.element.ElementObj;
15. **import** com.tedu.element.Play;
16. **import** com.tedu.manager.ElementManager;
17. **import** com.tedu.manager.GameElement;
19. **public** **class** GameMainJPanel **extends** JPanel **implements** Runnable {
20. **private** ElementManager em;
22. **boolean** enemyAppear;
23. **boolean** bossAppear;
24. **boolean** staticAppear;
26. **public** GameMainJPanel() {
27. init();
28. }
30. **public** **void** init() {
31. em = ElementManager.getManager();
32. enemyAppear = **false**;
33. bossAppear = **false**;
34. }
36. @Override
37. **public** **void** paint(Graphics g) {
38. **super**.paint(g);
39. Map<GameElement, List<ElementObj>> all = em.getGameElements();
40. **for** (GameElement ge : GameElement.values()) {
41. List<ElementObj> list = all.get(ge);
42. **for** (**int** i = 0; i < list.size(); i++) {
43. ElementObj obj = list.get(i);
44. obj.showElement(g);
45. }
46. }
47. **if** (!em.getElementsByKey(GameElement.ENEMY).isEmpty())
48. enemyAppear = **true**;
49. **if** (!em.getElementsByKey(GameElement.BOSS).isEmpty())
50. bossAppear = **true**;
51. **if** (enemyAppear && bossAppear || !enemyAppear && !bossAppear)
52. staticAppear = **true**;
53. **else**
54. staticAppear = **false**;
55. **if** (staticAppear && em.getElementsByKey(GameElement.BOSS).isEmpty()) {
56. em.init();
57. g.setColor(Color.RED);
58. g.setFont(**new** Font("微软雅黑", Font.BOLD, 50));
59. g.drawString("恭喜你通过本关！！", 50, 300);
60. init();
61. }
62. }
64. @Override
65. **public** **void** run() {
66. **while** (**true**) {
67. **this**.repaint();
68. **try** {
69. Thread.sleep(10);
70. } **catch** (InterruptedException e) {
71. e.printStackTrace();
72. }
73. }
74. }
75. }

paint方法是进行绘画元素。绘画时是有固定的顺序，先绘画的图片会在底层，后绘画的图片会覆盖先绘画的。最后的run()函数时接口的实现。一般情况下，多线程都会使用一个休眠，用以控制速度。

### **4.2控制（com.tedu.controller）**

* **GameListener.java**

1. **import** java.awt.event.KeyEvent;
2. **import** java.awt.event.KeyListener;
3. **import** java.util.HashSet;
4. **import** java.util.List;
5. **import** java.util.Set;
7. **import** com.tedu.element.ElementObj;
8. **import** com.tedu.manager.ElementManager;
9. **import** com.tedu.manager.GameElement;
11. **public** **class** GameListener **implements** KeyListener{
12. **private** ElementManager em=ElementManager.getManager();
13. **private** Set<Integer> set=**new** HashSet<Integer>();
15. @Override
16. **public** **void** keyTyped(KeyEvent e) {
18. }
20. @Override
21. **public** **void** keyPressed(KeyEvent e) {
22. System.out.println("按下"+e.getKeyCode());
23. **int** key=e.getKeyCode();
24. **if**(set.contains(key)) {
25. **return**;
26. }
27. set.add(key);
28. List<ElementObj> play = em.getElementsByKey(GameElement.PLAY);
29. **for**(ElementObj obj:play) {
30. obj.keyClick(**true**, e.getKeyCode());
31. }
32. }
33. @Override
34. **public** **void** keyReleased(KeyEvent e) {
35. **if**(!set.contains(e.getKeyCode())) {
36. **return**;
37. }
38. set.remove(e.getKeyCode());
39. List<ElementObj> play = em.getElementsByKey(GameElement.PLAY);
40. **for**(ElementObj obj:play) {
41. obj.keyClick(**false**, e.getKeyCode());
42. }
43. }
44. }

KeyListener是监听类，用以监听用户的操作。keyPressed()函数拿到了玩家的集合，判定集合中是否已经包含某个对象，并返回该对象。如果不存在，就调用keyReleased()函数，停止，并移除数据。

* **GameThread.java**

1. **package** com.tedu.controller;
3. **import** java.util.List;
4. **import** java.util.Map;
6. **import** javax.swing.ImageIcon;
8. **import** com.tedu.element.ElementObj;
9. **import** com.tedu.element.Enemy;
10. **import** com.tedu.element.Play;
11. **import** com.tedu.manager.ElementManager;
12. **import** com.tedu.manager.GameElement;
13. **import** com.tedu.manager.GameLoad;
15. **public** **class** GameThread **extends** Thread{
16. **private** ElementManager em;
18. **private** **static** **int** flag = 1;
20. **public** **static** **int** getFlag() {
21. **return** flag;
22. }
24. **public** GameThread() {
25. em=ElementManager.getManager();
26. }
28. @Override
29. **public** **void** run() {
30. **while**(flag < 6) {
31. gameLoad();
32. gameRun();
33. gameOver();
34. **try** {
35. sleep(50);
36. } **catch** (InterruptedException e) {
37. e.printStackTrace();
38. }
39. }
40. }
41. **private** **void** gameLoad() {
42. GameLoad.loadImg(flag);
43. GameLoad.loadBackground(flag);
45. GameLoad.loadPlay(flag);
47. GameLoad.loadEnemy(flag);
49. }
51. **private** **long** gameTime = System.currentTimeMillis();
52. **boolean** r = **false**;
53. **private** **final** Object lock = **new** Object();
54. **private** **void** gameRun() {
55. **while**(**true**) {
56. **while**(Play.getPauseGame()) {
57. **try** {
58. Thread.sleep(1);
59. } **catch** (InterruptedException e) {
60. e.printStackTrace();
61. }
62. }
64. Map<GameElement, List<ElementObj>> all = em.getGameElements();
65. List<ElementObj> enemys = em.getElementsByKey(GameElement.ENEMY);
66. List<ElementObj> files = em.getElementsByKey(GameElement.PLAYFILE);
67. List<ElementObj> maps = em.getElementsByKey(GameElement.MAPS);
68. List<ElementObj> boss = em.getElementsByKey(GameElement.BOSS);
69. List<ElementObj> tool = em.getElementsByKey(GameElement.TOOL);
70. List<ElementObj> play = em.getElementsByKey(GameElement.PLAY);
71. moveAndUpdate(all,gameTime);
72. ElementPK(boss, files);
73. ElementPK(enemys,files);
74. ElementPK(files,maps);
75. PlayCrushEnemy();
77. ElementGetTool(play,tool);
78. gameTime++;
79. **try** {
80. sleep(10);
81. } **catch** (InterruptedException e) {
82. e.printStackTrace();
83. }
84. **if**(em.getElementsByKey(GameElement.ENEMY).size()==0 && !r) {
85. GameLoad.loadBoss(flag);
86. r = **true**;
87. }
88. **if**(em.getElementsByKey(GameElement.BOSS).size() == 0 && r) {
89. r=**false**;
90. **return**;
91. }
92. }
93. }
95. **public** **void** ElementPK(List<ElementObj> listA,List<ElementObj>listB) {
96. **for**(**int** i=0;i<listA.size();i++) {
97. ElementObj enemy=listA.get(i);
98. **for**(**int** j=0;j<listB.size();j++) {
99. ElementObj file=listB.get(j);
100. **if**(enemy.pk(file)) {
101. enemy.setHp(enemy.getHp()-10);
102. **if**(enemy.getHp()<0) {
103. enemy.setLive(**false**);
104. }
106. file.setLive(**false**);
107. **break**;
108. }
109. }
110. }
111. }
113. **public** **void** PlayCrushEnemy() {
114. **for**(**int** i = 0; i < em.getElementsByKey(GameElement.PLAY).size(); i++) {
115. ElementObj play = em.getElementsByKey(GameElement.PLAY).get(i);
116. **for**(**int** j = 0; j < em.getElementsByKey(GameElement.ENEMY).size(); j++) {
117. ElementObj enemy = em.getElementsByKey(GameElement.ENEMY).get(j);
118. **if**(play.pk(enemy)) {
119. play.setHp(play.getHp() - 5);
120. **if**(play.getHp() < 0)
121. play.setLive(**false**);
122. enemy.setLive(**false**);
123. **break**;
124. }
125. }
126. }
127. }
129. **public** **void** ElementGetTool(List<ElementObj> listA,List<ElementObj>listB) {
130. **for**(**int** i=0;i<listA.size();i++) {
131. ElementObj play=listA.get(i);
132. **for**(**int** j=0;j<listB.size();j++) {
133. ElementObj tool=listB.get(j);
134. **if**(play.pk(tool)) {
135. System.out.println(listB);
136. tool.setLive(**false**);
137. **if**(tool.getHp()==0){
138. List<ElementObj> playlist = ElementManager.getManager().getElementsByKey(GameElement.PLAY);
139. **for**(**int** k = 0 ;k<playlist.size();k++) {
140. ElementObj player = playlist.get(k);
141. player.setSd(player.getSd()\*2);
142. }
143. }
144. **if**(tool.getHp()==2){
145. List<ElementObj> bosslist = ElementManager.getManager().getElementsByKey(GameElement.BOSS);
146. **for**(**int** k = 0 ;k<bosslist.size();k++) {
147. ElementObj boss = bosslist.get(k);
148. boss.setHp(boss.getHp()/2);
149. }
150. }
151. **if**(tool.getHp()==1){
152. List<ElementObj> playlist = ElementManager.getManager().getElementsByKey(GameElement.PLAY);
153. **for**(**int** k = 0 ;k<playlist.size();k++) {
154. ElementObj player = playlist.get(k);
155. player.setHp(player.getHp()+10);
156. }
157. }
158. **break**;
159. }
160. }
161. }
162. }

165. **public** **void** moveAndUpdate(Map<GameElement, List<ElementObj>> all, **long** g) {
166. **for**(GameElement ge: GameElement.values()) {
167. List<ElementObj> list = all.get(ge);
168. **for**(**int** i = 0; i < list.size(); i++) {
169. ElementObj obj = list.get(i);
170. **if**(!obj.isLive()) {
171. obj.die();
172. list.remove(i--);
173. **continue**;
174. }
176. obj.model(g);
177. }
178. }
179. gameTime=System.currentTimeMillis();
180. }

183. **private** **void** gameOver() {
184. **try** {
185. sleep(3000);
186. } **catch** (InterruptedException e) {
187. e.printStackTrace();
188. }
189. **if**(flag > 4) {
190. System.exit(0);
191. }
192. flag++;
193. System.out.println(flag);
194. GameLoad.clear();
195. }
196. }

游戏的主线程，用于控制游戏加载，游戏关卡，游戏运行时自动化。run()方法是游戏的主线程，while(flag<6)表示一共设置了五个关卡，其中，游戏开始前的工作是读取进度条，加载游戏资源和场景资源，通过gameLoad()函数实现；游戏进行时执行gameRun()函数；游戏场景结束的工作是游戏资源和场景资源的回收，通过gameOver()函数实现。

### **4.3资源管理（com.tedu.element）**

* **ElementObj.java**

1. **package** com.tedu.element;
3. **import** java.awt.Graphics;
4. **import** java.awt.Rectangle;
6. **import** javax.swing.ImageIcon;
8. **public** **abstract** **class** ElementObj {
9. **private** **int** x;
10. **private** **int** y;
11. **private** **int** w;
12. **private** **int** h;
13. **private** ImageIcon icon;
14. **private** **int** hp = 10;
15. **private** **int** sd = 3;
16. **private** **int** gq = 1;
17. **public** **int** getGq() {
18. **return** gq;
19. }
20. **public** **void** setGq(**int** gq) {
21. **this**.gq = gq;
22. }
23. **public** **void** setSd(**int** sd) {
24. **this**.sd=sd;
25. }
26. **public** **int** getSd() {
27. **return** **this**.sd;
28. }
29. **public** **void** setHp(**int** hp) {
30. **this**.hp = hp;
31. }
33. **public** **int** getHp() {
34. **return** **this**.hp;
35. }
37. **private** **boolean** live=**true**;
38. **public** ElementObj() {
40. }
42. **public** ElementObj(**int** x, **int** y, **int** w, **int** h, ImageIcon icon) {
43. **super**();
44. **this**.x = x;
45. **this**.y = y;
46. **this**.w = w;
47. **this**.h = h;
48. **this**.icon = icon;
49. }
51. **public** **abstract** **void** showElement(Graphics g);
53. **public** **void** keyClick(**boolean** bl,**int** key) {
54. System.out.println("测试使用");
55. }
57. **protected** **void** move() {
59. }
60. **public** **final** **void** model(**long** gameTime) {
61. updateImage();
62. move();
63. add(gameTime);
64. }
65. **protected** **void** updateImage() {}
66. **protected** **void** add(**long** gameTime){}
68. **public** **void** die() {
70. }
72. **public**  ElementObj createElement(String str) {
73. **return** **null**;
74. }
75. **public** Rectangle getRectangle() {
76. **return** **new** Rectangle(x,y,w,h);
77. }
78. **public** **boolean** pk(ElementObj obj) {
79. **return** **this**.getRectangle().intersects(obj.getRectangle());
80. }
82. **public** **int** getX() {
83. **return** x;
84. }
85. **public** **void** setX(**int** x) {
86. **this**.x = x;
87. }
88. **public** **int** getY() {
89. **return** y;
90. }
91. **public** **void** setY(**int** y) {
92. **this**.y = y;
93. }
94. **public** **int** getW() {
95. **return** w;
96. }
97. **public** **void** setW(**int** w) {
98. **this**.w = w;
99. }
100. **public** **int** getH() {
101. **return** h;
102. }
103. **public** **void** setH(**int** h) {
104. **this**.h = h;
105. }
106. **public** ImageIcon getIcon() {
107. **return** icon;
108. }
109. **public** **void** setIcon(ImageIcon icon) {
110. **this**.icon = icon;
111. }
112. **public** **boolean** isLive() {
113. **return** live;
114. }
115. **public** **void** setLive(**boolean** live) {
116. **this**.live = live;
117. }
118. }

所有元素的基类。hp代表血量，sd代表移速。在抽象类ElementObj中定义了x和y即左上角x、y坐标，w即宽度，h即高度，icon即图标，live即生存状态（其中true为存活，false为死亡）。另外，updateImage()函数表示换装，move()函数表示移动，add(gameTime)函数表示发射子弹，其中gameTime参数为游戏间隔时间。

* **Enemy.java**

1. **package** com.tedu.element;
3. **import** java.awt.Graphics;
4. **import** java.util.List;
5. **import** java.util.Random;
7. **import** javax.swing.ImageIcon;
9. **import** com.tedu.manager.ElementManager;
10. **import** com.tedu.manager.GameElement;
11. **import** com.tedu.manager.GameLoad;
13. **public** **class** Enemy **extends** ElementObj{
14. **private** String fx = "right";
15. **private** **int** x;
16. **int** back = (**int**) (Math.random() \* 200);
18. @Override
19. **public** **void** showElement(Graphics g) {
20. g.drawImage(**this**.getIcon().getImage(),
21. **this**.getX(), **this**.getY(),
22. **this**.getW(), **this**.getH(), **null**);
23. }
25. @Override
26. **public** ElementObj createElement(String str) {
27. Random ran=**new** Random();
28. **int** x=ran.nextInt(600);
29. **int** y=ran.nextInt(300);
30. **this**.setX(x);
31. **this**.setY(y);
32. **this**.setW(50);
33. **this**.setH(50);
34. **this**.setHp(1);
35. **this**.setIcon(**new** ImageIcon("image/enemy/"+str+".png"));
36. **return** **this**;
37. }
39. @Override
40. **public** **void** move() {
41. **if**(fx.equals("right")) {
42. **if**((**this**.getX() < 770 - **this**.getW()) && (**this**.getX() - x < back)) {
43. **this**.setX(**this**.getX() + 1);
44. }
45. **else** {
46. **this**.fx = "left";
47. back = (**int**) (Math.random() \* 400);
48. }
49. }
50. **if**(fx.equals("left")) {
51. **if**(**this**.getX() > 0 && (x - **this**.getX() < back)) {
52. **this**.setX(**this**.getX() - 1);
53. }
54. **else** {
55. **this**.fx = "right";
56. back = (**int**) (Math.random() \* 400);
57. }
58. }
59. }
60. @Override
61. **public** **void** die() {
62. ElementObj obj = GameLoad.getObj("tool");
63. ElementObj tool = obj.createElement(**this**.toString());
64. ElementManager.getManager().addElement(tool, GameElement.TOOL);
66. }
68. @Override
69. **public** String toString() {
70. **int** a = **new** Random().nextInt(3);
71. **int** x = **this**.getX();
72. **int** y = **this**.getY();
73. **return** "x:"+x+",y:"+y+",hp:"+a;
74. }
75. }

敌人类，包括showElement显示敌人、createElement创建敌人、重写move敌人移动、重写die敌人死亡和toString五个方法。其中的类成员fx代表敌机移动方向，back代表敌机往一个方向移动随机距离后掉头。通过敌人类的move方法可以实现敌机的自动左右移动；die方法加载了tool工具类，实现了敌机死亡后的道具掉落，道具能给玩家主角提供增益效果。

## **五、游戏界面展示**

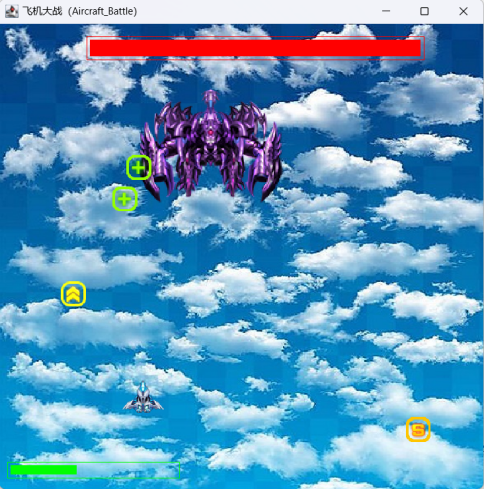


图10 第一关卡至第五关卡界面



图11 通关提示界面

## **六、总结**

### **6.1设计亮点**

* **玩家可以选择暂停游戏**

正常的游戏应该有暂停和继续的考虑，在本项目中玩家可以在游戏结束前的任意时刻选择按下键盘上的“P”键（注意在英文状态下）选择暂停游戏；当玩家想要继续参与体验游戏时，同样地按下键盘上的“P”键（注意在英文状态下）即可恢复游戏暂停前的状态。

* **成长系统与角色发展**

游戏设计了一个成长系统，玩家驾驶的战机通过击败敌机、通过关卡或收集道具等来获取经验值和升级点数，提升飞机的属性和性能。

游戏里对玩家的身份进行了合理的设定以展现游戏世界的深度和情节发展，不仅有丰富的关卡设计，多个层次的背景（如云层、山脉、城市等），还有多样化的敌人和武器，多种补充血量或击杀敌机的拾取道具种类，同时实现了包括但不限于飞机升级、关卡选择、敌机随机生成、碰撞、计分进度面板、boss血条等一系列直观简洁的功能。

* **游戏交互性和体验感强**

精美的游戏画面和特效也是提升游戏体验的重要因素。流畅的飞行动画和精细的飞机模型，使玩家感觉自己真正在操控飞机。华丽的爆炸效果和粒子效果增加了战斗的震撼力和视觉上的奖励感。

* **耦合性低，重用性高**

这本身也是MCV框架本身的优点，与游戏具体实现无关。

一方面，视图层和[业务层](https://baike.baidu.com/item/%E4%B8%9A%E5%8A%A1%E5%B1%82/22287753?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)分离，这样就允许更改视图层代码而不用重新编译模型和控制器代码，同样，一个应用的[业务流程](https://baike.baidu.com/item/%E4%B8%9A%E5%8A%A1%E6%B5%81%E7%A8%8B/10590352?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)或者[业务规则](https://baike.baidu.com/item/%E4%B8%9A%E5%8A%A1%E8%A7%84%E5%88%99/10826802?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)的改变只需要改动MVC的模型层即可。因为模型与控制器和视图[相分离](https://baike.baidu.com/item/%E7%9B%B8%E5%88%86%E7%A6%BB/9552390?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)，所以很容易改变[应用程序](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F/5985445?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)的[数据层](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%B1%82/20352986?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/MVC%E6%A1%86%E6%9E%B6/_blank)和业务规则。

另一方面，MVC模式允许使用各种不同样式的视图来访问同一个服务器端的代码，因为多个视图能共享一个模型，它包括任何WEB（HTTP）浏览器或者无线浏览器（wap），比如，用户可以通过电脑也可通过手机来订购某样产品，虽然订购的方式不一样，但处理订购产品的方式是一样的。由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。

### **6.2不足**

* **我方战机的机头不能转换方向**

在对敌机和boss进行攻击时，我方飞机的方向永远朝向以电脑屏幕为参考系的正上方，而且当键盘按下向右键时，飞机能向右方飞行，这不符合日常生活中的客观规律。**正确的处理方法应该是根据角度生成不同方向的子弹再使其移动**，但未实现。

* **敌机不能攻击我方战机**

正常的游戏应该有暂停和继续的考虑，但如果玩家在某个关卡停止操作，道具会正常掉落，会导致一直停留在某一个关卡而无法进入下一关卡从而延长了游戏的时间，降低游戏交互的体验感。

* **我方飞机被boss覆盖**

我方飞机向上移动时，会出现被boss覆盖的情况，仿佛我方飞机和boss出现在两个图层中，且我方飞机只能与敌方飞机发生碰撞而不能对boss造成碰撞伤害。

* **视图与控制器间的过于紧密的连接并且降低了视图对模型数据的访问**

视图与控制器是相互分离，但却是联系紧密的部件，视图没有控制器的存在，其应用是很有限的，反之亦然，这样就妨碍了他们的独立重用。依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。这也是MCV框架本身容易出现的不足之处，与游戏具体实现无关。

### **6.3反思**

这次的项目开发，有许多值得肯定的地方。

首先，我们项目组的成员一致认为，经过对本游戏项目的开发，我们已经对MVC框架有一个较为深刻且全面的理解，在进行开发时，也很好地运用了这个框架的中心思想。

其次，我们的开发过程遇到了众多的问题，如在实现游戏暂停的过程中，我们尝试了多种方法将按键和线程等待相关联，但是只能做到线程暂停，由于暂停后线程无法再进行监听，即暂停了无法继续游戏。但我们后来找到了一种曲线的方法，在监听到暂停按键后，我们不使得主线程等待，而是选择让主线程一直循环休眠，保证了主线程继续运行但展示的游戏画面在停止，成功地解决了问题。遇事不决主动求索是一种好习惯，同样也是日后工作岗位上必备的能力。

同样，也存在着许多不足的地方。

代码设计中，我们虽然对MVC框架有了全面的了解，也通过重写父类方法的方式实现子类功能。但是由于文件众多，文件之间互相传参，导致有些代码冗余，没有封装成一个通用的方法。同时，因为一个项目分成多位成员共同协作，每位成员的代码编写风格各异，导致不能及时看懂其他人的代码，也出现了一个模块有多位成员在共同编写的情况，使得代码可读性较差。

固然，我们承认对课程的理解不够深刻，今后也将不断努力，在具体实践中夯实基础，积累开发经验。