

《面向对象的程序设计》

GenshinChess游戏开发文档

小组成员：

汪子涵 U201916295

王懿瑶 U201916338

**目 录**

[一、项目概述 2](#_Toc89945042)

[1.1 项目范围 2](#_Toc89945043)

[1.2 项目活动记录 2](#_Toc89945044)

[1.2.1 项目工作内容及分工 3](#_Toc89945045)

[二、项目需求 3](#_Toc89945046)

[三、系统设计 3](#_Toc89945047)

[1.1 系统流程描述 3](#_Toc89945048)

[1.2 系统特色功能 4](#_Toc89945049)

[1.2.1 元素反应机制 4](#_Toc89945050)

[1.2.2 怪物特性 4](#_Toc89945051)

[1.3 系统线程设计 5](#_Toc89945052)

[1.4 系统类图 6](#_Toc89945053)

[1.5 接口和包的设计 7](#_Toc89945054)

[1.6 关键代码 8](#_Toc89945055)

[1.6.1 元素反应的实现 8](#_Toc89945056)

[1.6.2 多线程控制 10](#_Toc89945057)

[1.6.3 画面渲染 10](#_Toc89945058)

[1.6.4 地图设计和使用 11](#_Toc89945059)

[1.6.5 怪物移动 11](#_Toc89945060)

[1.6.6 防御塔守卫 11](#_Toc89945061)

[1.6.7 怪物受击 12](#_Toc89945062)

[游戏效果图 13](#_Toc89945063)

[总结 15](#_Toc89945064)

[代码附录 16](#_Toc89945065)

一、项目概述

该项目为塔防游戏。本游戏是基于【原神】的元素反应机制而设计开发的塔防游戏。游戏目标为在地图指定位置上安装特定类型的防御塔，以阻止路线上的怪物到达终点。防御塔有三种类型，冰元素塔、水元素塔和火元素塔；怪物有5种类型，且部分怪物对不同的元素也有不同的抵抗能力，需要玩家通过不同元素之间的相互配合，以更快地消灭路线上的敌人，取得最后的胜利。本项目借鉴了一般塔防游戏布阵设防的思路，使得操作界面亲切、玩家易上手；又加入了一般塔防游戏不具有的元素反应机制，增强了游戏的策略性。

* 1. 项目范围

1. 游戏显示界面，包括启动页、帮助页、游戏界面；其中游戏界面中包括顶部导航栏（里面有游戏启动/暂停、重新开始和返回按钮）、游戏动画面板、建造面板三大类；
2. 游戏内元素反应原理设计、实现。
3. 游戏内各种类型的防御塔、怪物实体；
4. 游戏所需图片、音频资源；
   1. 项目活动记录

1、在【11月20日 - 11月24日】内讨论选题，并最终决定开发塔防小游戏；完成了远程仓库的配置（<https://gitee.com/oliver-puppio/GenshinChess>）；

2、在【11月25日 – 11月27日】内设计完成游戏的UML类图，并根据类图完成了初始框架的搭建。

3、在【11月28日 – 12月1日】内补充了Monster及其子类、Monsters刷怪类的细节；

4、在【12月2日 – 12月3日】内补充了Tower、Bullet类的细节，补充了测试图片，开始调试线程控制；

5、在【12月4日】测试了GUI，怪物行走基本正常；

6、在【12月5日】更换了Monster的在刷怪类Monsters中的存储方式，使之容易实现线程控制；防御塔攻击显示正常；完成了元素反应的设计和初步实现；

7、在【12月6日】修复了游戏暂停、重开的bug，并完成了高等怪物的元素抵抗机制；

8、在【12月7日】新增了地图和美化了游戏内显示界面；

9、在【12月8日】修复了异常；继续美化页面，撰写文档。

* + 1. 项目工作内容及分工

表1. 项目组成员工作内容的分工

|  |  |
| --- | --- |
| 姓名 | 工作任务 |
| 汪子涵 | 制定游戏规则、设计元素反应原理；设计和实现所有实体类；撰写开发文档 |
| 王懿瑶 | 设计、实现游戏界面和实体类线程控制；搜集、创作游戏素材 ；项目调试、优化 |

二、项目需求

本游戏实现了传统塔防游戏的基本功能，还设计了元素反应的机制。此外，也实现了地图选关、游戏暂停、重开、退出、教程等功能。具体可分为如下几个部分：

1. 防御塔的购买、售卖、自动攻击；
2. 怪物的移动和受击；
3. 维护玩家金币值、生命值；
4. 游戏暂停、重开、退出；
5. 查看游戏教程。

三、系统设计

* 1. 系统流程描述

在开始界面，玩家可以选择点击直接开始游戏或者进入帮助页面查看游戏教程。点击开始游戏后系统会随机加载一张地图然后进入游戏界面。在游戏界面里，顶部是导航栏，有开始、重新开始和退出三个按钮，点击最左边的开始按钮后会正式开启游戏。

游戏界面中，左上角显示玩家当前的金币数、生命值和剩余怪物数量。玩家点击开始后，路线上会产生怪物。点击路线旁的标志点可以安放防御塔。当剩余怪物数为0且剩余生命值大于0时，游戏胜利；若玩家生命值小于0，则游戏失败。在游戏时，根据怪物特点选择合适的防御塔，会增加取胜的概率。

* 1. 系统特色功能

在传统塔防游戏的基础上，我们借鉴了【原神】游戏中的元素反应机制，为普通的塔防游戏带来了新的玩法。

* + 1. 元素反应机制

元素反应机制是本游戏的核心玩法之一。在本游戏中，一共有三种元素：冰、火、水。它们之间可以两两产生反应，从而对被攻击的怪物产生一定效果。以下为元素反应表：

表2. 元素反应类型表

|  |  |  |
| --- | --- | --- |
| 反应名称 | 需要元素 | 效果 |
| 蒸发 | 火、水 | 伤害增加1.2 |
| 融化 | 火、冰 | 伤害增加1.3 |
| 冻结 | 水、冰 | 伤害增加1.1，并会让怪物暂时不能行走 |

* + 1. 怪物特性

既然元素反应能对防御塔攻击带来正面收益，若怪物只能乖乖地接受增强的伤害的话，那元素反应就不会带来新的游戏体验。所以我们设置了怪物的特性。我们一共有5种怪物，可分为两类，一种是普通怪物，有丘丘人、藏镜侍女、丘丘暴徒；另一种是精英怪物，有火系深渊法师、水系深渊法师。他们的特性如下表：

表3. 怪物特性表

|  |  |  |
| --- | --- | --- |
| 怪物 | 特性 | 移动速度 |
| 普通丘丘人 | 无 | 慢 |
| 藏镜侍女 | 每当元素附着效果被移除时，能为周围怪物恢复自己当前生命值的30% |
| 丘丘暴徒 | 具有护盾，但火元素附着能造成护盾值持续损失 |
| 火深渊法师 | 具有护盾，只有蒸发、融化反应才能消除 | 快 |
| 水深渊法师 | 具有护盾，只有冻结、蒸发反应能消除 |

* 1. 系统线程设计

本游戏需要实时和玩家的操作互动，也使用了大量动画技术。我们采用多线程技术实现这些功能。使用线程的有游戏界面类、动画面板类、防御塔类、怪物类、怪物产生控制类。为了实现重新开始和暂停控制，我们也使用了玩家对象作为锁机制，控制所有线程的运行。

在以上线程中，游戏界面类主要负责检测游戏是否被暂停或者重设、玩家是否取得胜利；动画面板类线程负责绘制所有实体对象的动画效果；防御塔类线程负责检查是否有怪物靠近、并投掷炮弹；怪物类线程负责向控制怪物向终点前近；怪物产生控制类负责根据地图参数，定期在指定地方产生新的怪物。

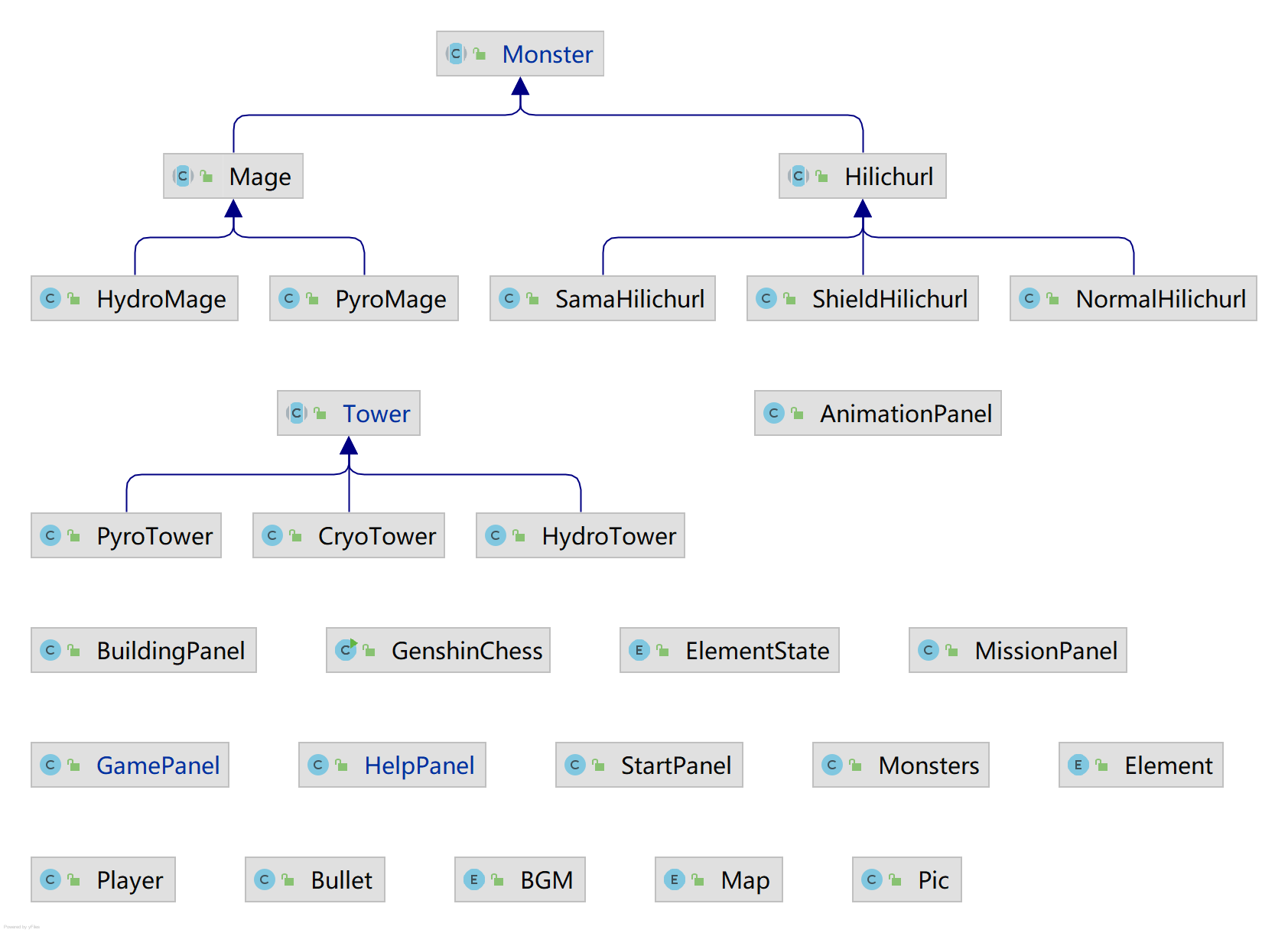
表4. 线程功能表

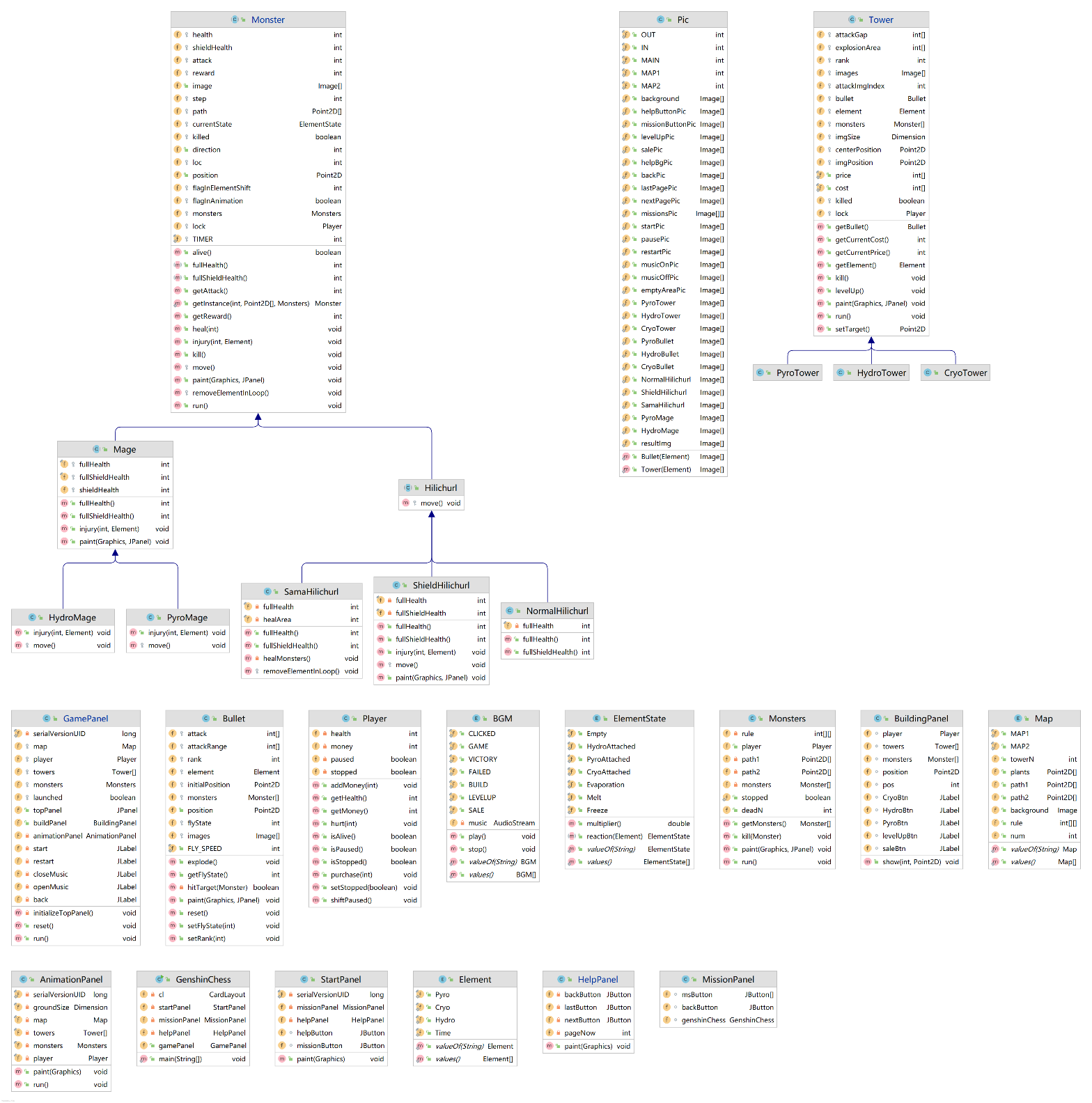
|  |  |
| --- | --- |
| 线程 | 功能 |
| GamePanel | 循环判定是否被暂停，检测是否游戏结束 |
| AnimationPanel | 绘制所有实体对象的位置和动画效果 |
| Tower | 检查是否有怪物靠近，控制投弹 |
| Monster | 控制怪物向终点前近 |
| Monsters | 不断产生新怪物 |

* 1. 系统类图

本系统类结构可分为三大类：GUI类、实体类和资源类。

如下图所示：



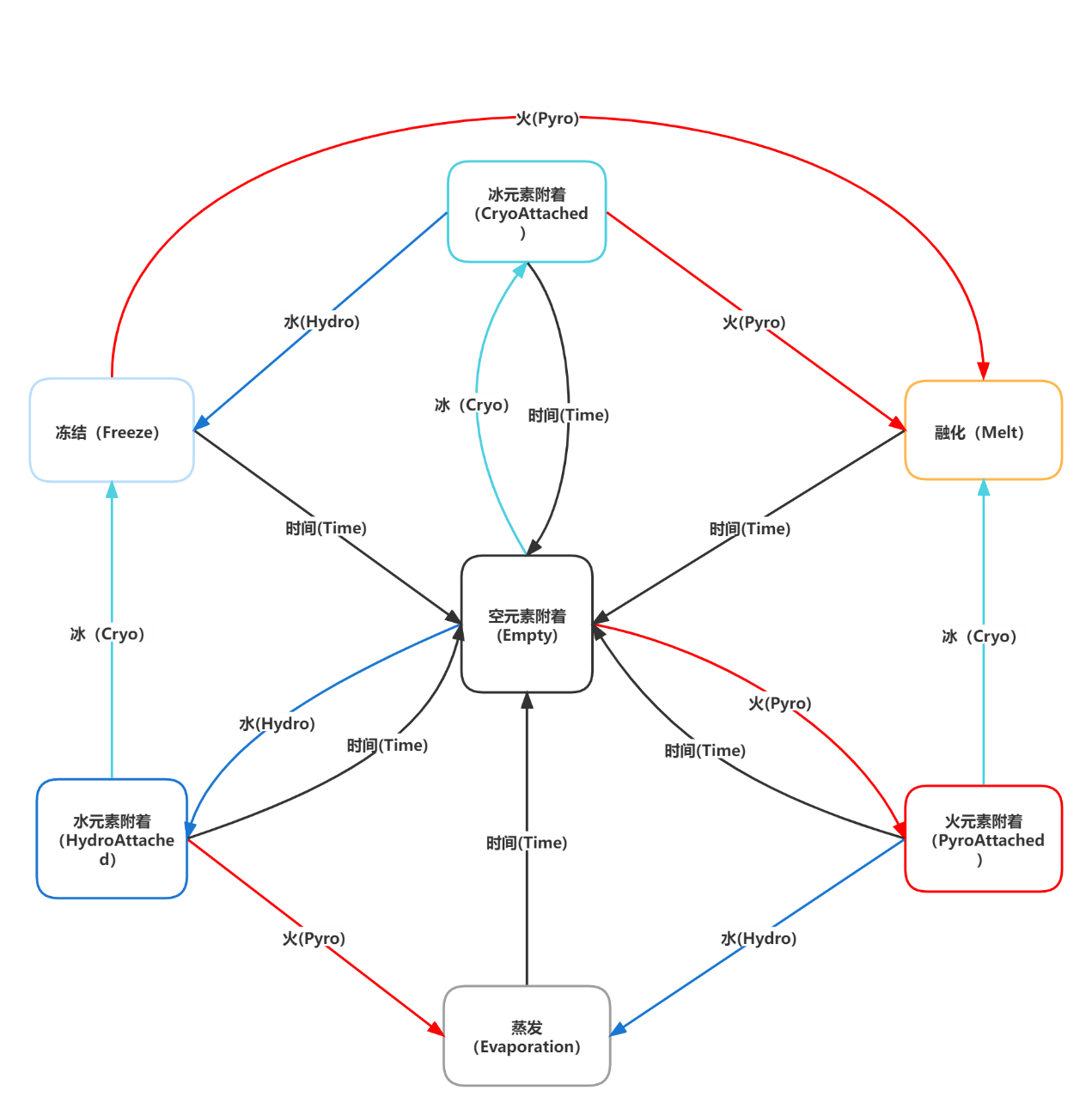


* 1. 接口和包的设计

|  |  |
| --- | --- |
| 包名 | 功能 |
| bullet | 炮弹类 |
| element | 元素及元素反应机制 |
| game | 游戏入口、玩家类 |
| gui | 系统界面类 |
| tower | 各种防御塔类 |
| monster | 各种怪物类和刷怪生成器类 |
| res | 系统所需的资源，包括图片、音乐、地图 |

* 1. 关键代码
     1. 元素反应的实现

实现元素反应时，我们考虑到了元素和反应结果，确定了元素反应状态图。为了更好地表达攻击时发生了哪种反应，以及这种反应给怪物带来了哪些效果。我们引入了空状态(Empty)和时间元素(Time)。我们规定所有怪物一开始都处于Empty状态，当接收到元素攻击，怪物状态发生改变，故我们设计了ElementState.reaction(element)方法；元素是会先附着在怪物身上，被触发反应才造成效果的，因此我们用Empty状态可以很好地把这一过程转化为元素反应，即Empty和特定元素反应，状态会变成“对应元素附着”；元素附着和反应结果都有一段持续时间，我们可以把这个过程想象成时该元素状态在Time元素下发生转换，得到了如下的状态图。



[ElementState](#元素状态属性)

* + 1. 多线程控制

一局游戏中存在用于产生新怪物的Monsters线程、游戏中存活的每个怪Monster以及建造的每个防御塔Tower线程，这些线程都统一在一局游戏中，玩家暂停、继续以及重新开始本局游戏的操作都应当能够统一地控制本轮游戏中的所有线程。为此我们将玩家Player类作为控制类，每轮游戏中的各线程都引用该游戏对应的Player对象，并获取Player对象的方法判断线程是否应阻塞等待（暂停）、是否应该结束线程。

将Player设置为同步锁对象，当玩家选择暂停，会触发监听事件将Player标记为暂停，在此条件下，本局游戏所有相关的线程皆会因此阻塞，进入该Player对象的等待集。当玩家选择继续游戏，则鼠标监听事件将会取消该Player的暂停标记，并同时调用该对象的notifyAll()方法，唤醒等待集中的所有线程，使游戏在暂停后能够继续接着运行。

关于游戏的重置，我们设置Player的停止属性stopped为游戏中所有线程的判断条件，一旦该线程引用的Player对象为停止状态，线程将跳出循环，从而终止线程。

[Player](#Player对象)

[Tower.run](#Tower_run)

[Monster.run](#Monster_run)

[Monsters.run](#Monsters_run)

[AnimationPanel.run](#AnimationPanel_run)

* + 1. 画面渲染

前端界面部分我们全部使用JavaGUI的swing框架完成。相关类包括GenshinChess（继承JFrame）、StartPanel、HelpPanel以及GamePanel。GenshinChess采用卡片布局管理器，使能在各个面板方便切换显示。

考虑到GUI中JButton类型视觉效果欠佳，我们采用JLabel标签替代按钮，在JLabel中添加设置所需图片，并添加相应的监听事件，达到和JButton相同的按钮效果。

鼠标监听事件中，我们为每个JLabel标签设置了鼠标移入/移出监听，通过显示不同图片，提升用户视觉体验与交互效果。

除了静态页面的显示，游戏中我们还需不断刷新画面，以实现画面中怪的移动、塔攻击与子弹飞行。这里我们重写了相关类的paint()方法，根据游戏的当前状态绘制不同的静态图片，再通过AnimationPanel线程中run()方法循环调用该类的repanit()方法不停重绘，渲染出动态游戏画面。

[Bullet,Tower,Monster,AnimationPanel的paint](#Paint)

* + 1. 地图设计和使用

丰富的地图是游戏有趣的标准之一。为此我们设置了枚举类Map以方便地创建地图信息。在Map类中，可以自定义地图背景图片、防御塔的数量、怪物数量、刷怪规则、2条怪物路线。

在GamePanel被创建时，就会随机载入地图，并完成一些控件、实体的实例化。

[Map](#Map)

* + 1. 怪物移动

怪物移动move方法在线程run方法中被循环调用。该方法会首先判断怪物是否处于冻结状态，若是则不执行，否则会改变怪物的位置和方向。若怪物是丘丘暴徒，还会增加是否处于火元素附着状态且护盾存在的判断，用来持续扣除该状态下的丘丘暴徒的护盾值。

Move方法调用完后，run方法还会调用removeElementInLoop方法进行元素状态的更新。该方法使用了一个初值为0的flagInElementShift，每当该方法被调用时，就会使得flagInElementShift的值加1，当其达到25时，就会让当前元素和Time元素反应，即更新状态到无元素附着状态。并让flagInElementShift的值重新归0。这样就实现了元素状态的刷新。

[Monster.run](#Monster_run)

[Monster.move](#Monster_move)

[Monster.removeElementInloop](#Monster_removeElementInLoop)

* + 1. 防御塔守卫

防御塔线程不断调用setTarget方法，在setTarget方法中遍历怪物列表，若发现有存活的怪物在其攻击范围之内，则会发起攻击，返回Point2D类值对象。

当返回值不为空时，调用发射动画、并不停更改炮弹bullet的位置，炮弹运动到目标位置后调用其类的explode方法，对范围内怪物造成元素伤害，随后重置bullet的状态；当返回值为空，即没有找到目标时，则休息间隔时间，随后开始继续搜索目标。

[Tower.run](#Tower_run)

[Tower.setTarget](#Tower_setTarget)

[Bullet.explode和Bullet.reset](#Bullet_explode)

* + 1. 怪物受击

当怪物受到攻击时，调用injury方法，在injury方法内完成生命值的扣除和怪物附着的元素的更新。由于蒸发和融化是瞬间反应完成的，所有injury传入的元素参数若导致了这两种反应的发生，则会再发生一次和Time元素的反应，从而将状态更新为Empty；否则则会保留下来，并通过removeElementInLoop()和move()的发生，更新怪物身上的元素状态。

针对特殊的丘丘暴徒、深渊法师，我们重载了injury方法，完善了其在护盾存在时应进行的操作。

[Monster.injury](#Monster_injury)

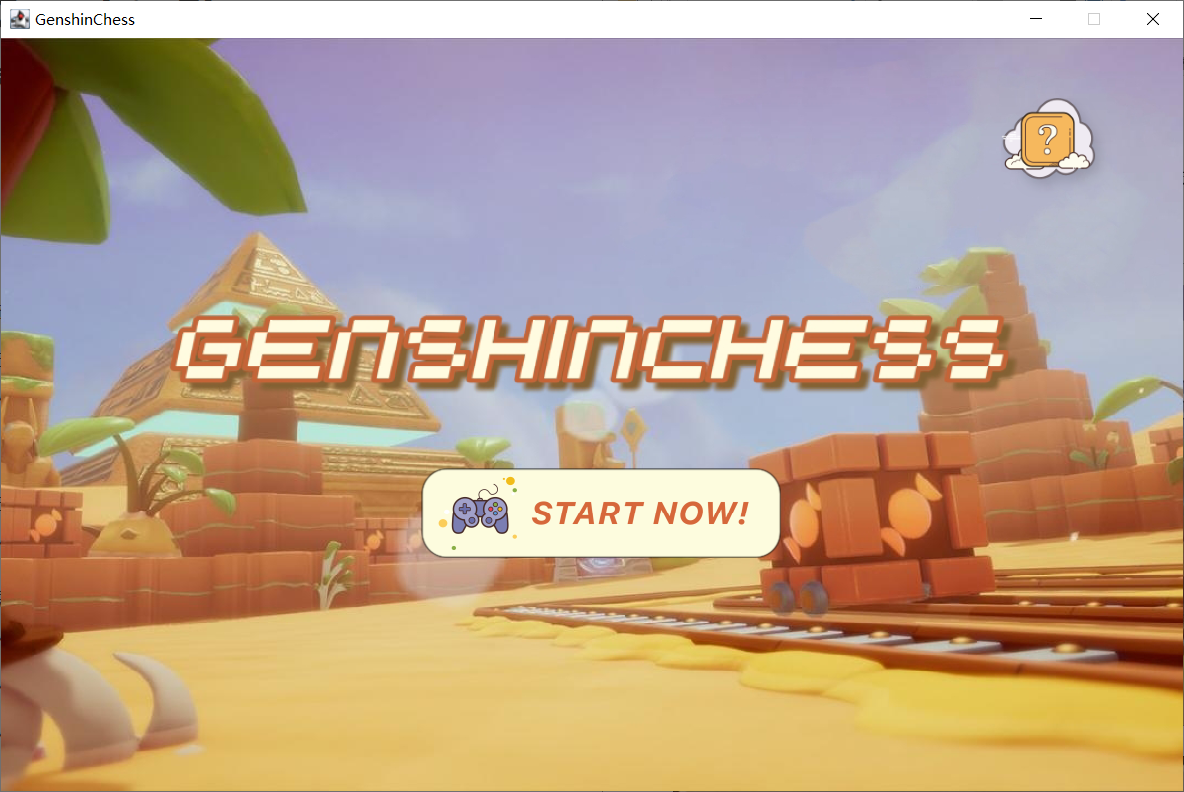
[HydroMage.injury](#HydroMage_injury)

[PyroMage.injury](#PyroMage_injury)

[ShieldHilichurl.injury](#ShieldHilichurl_injury)

游戏效果图

游戏能够正常的进行防御塔建立，怪兽死亡，胜利，失败，道具使用。项目目标中的功能基本全部实现。

图一：开始界面

图二：帮助界面



图三：游戏界面一



图四：游戏界面二



总结

该项目前后历时两周时间。通过这次项目实战，小组成员熟练掌握了Java语言的基础语法，对多线程开发、GUI编程有了新的认识。虽然小组成员只有两人，但在开发过程中并未因为人少不足而浪费太多时间，相反，合理的项目目标、细致的工作任务分解、公开资源的收集、利用和开发工具的熟练使用，使我们取得了事半功倍的效果。

在本次项目中，小组成员表现出了如下的特点：

1. 熟练运用各种工具。项目选题一开始确立，就着手准备系统类图的绘制。根据系统类图，很快就完成了项目代码的框架布局。在开发过程中，远程仓库的使用让协同开发更高效，快捷键的熟练使用也加快了设计思路从思想到实现的转变；
2. 养成了良好的注释习惯。组内统一了注释的格式，使得代码可读性强，便于debug和开发新功能；
3. 具有较强的学习能力。当项目缺少图片素材时，小组成员能迅速上手画图软件，画出像素小人图片；当遇到难以解决的bug时，能通过查阅API文档、查找参考书等方式，找到规范的使用方法；
4. 积极参与、沟通及时。小组成员每天投入大量精力，遇到问题时都会及时反馈、共同解决。

代码附录

1. **package** element;
3. /\*\*
4. \* 元素状态属性
5. \*/
6. **public** **enum** ElementState {
7. Empty {
8. **public** ElementState reaction(Element element) {
9. **switch** (element) {
10. **case** Hydro:
11. **return** HydroAttached;
12. **case** Pyro:
13. **return** PyroAttached;
14. **case** Cryo:
15. **return** CryoAttached;
16. **default**:
17. **return** Empty;
18. }
19. }
20. },
21. HydroAttached {
22. **public** ElementState reaction(Element element) {
23. **switch** (element) {
24. **case** Pyro:
25. **return** Evaporation;
26. **case** Cryo:
27. **return** Freeze;
28. **case** Time:
29. **return** Empty;
30. **default**:
31. **return** HydroAttached;
32. }
33. }
34. },
35. PyroAttached {
36. **public** ElementState reaction(Element element) {
37. **switch** (element) {
38. **case** Hydro:
39. **return** Evaporation;
40. **case** Cryo:
41. **return** Melt;
42. **case** Time:
43. **return** Empty;
44. **default**:
45. **return** PyroAttached;
46. }
47. }
48. },
49. CryoAttached {
50. **public** ElementState reaction(Element element) {
51. **switch** (element) {
52. **case** Hydro:
53. **return** Freeze;
54. **case** Pyro:
55. **return** Melt;
56. **case** Time:
57. **return** Empty;
58. **default**:
59. **return** CryoAttached;
60. }
61. }
62. },
63. Evaporation {
64. **public** ElementState reaction(Element element) {
65. **switch** (element) {
66. **case** Time:
67. **return** Empty;
68. **default**:
69. **return** CryoAttached;
70. }
71. }
73. @Override
74. **public** **double** multiplier() {
75. **return** 1.2;
76. }
77. },
78. Melt {
79. **public** ElementState reaction(Element element) {
80. **switch** (element) {
81. **case** Time:
82. **return** Empty;
83. **default**:
84. **return** Melt;
85. }
86. }
88. @Override
89. **public** **double** multiplier() {
90. **return** 1.3;
91. }
92. },
93. Freeze {
94. **public** ElementState reaction(Element element) {
95. **switch** (element) {
96. **case** Pyro:
97. **return** Melt;
98. **case** Time:
99. **return** Empty;
100. **default**:
101. **return** Freeze;
102. }
103. }
105. @Override
106. **public** **double** multiplier() {
107. **return** 1.1;
108. }
109. };
111. /\*\*
112. \* 伤害乘数
113. \* @return 返回伤害乘数，默认为1
114. \*/
115. **public** **double** multiplier() {
116. **return** 1;
117. }
119. /\*\*
120. \* 元素反应方法，返回反应后的元素状态
121. \* @param element 参与反应的元素
122. \* @return 新的元素状态
123. \*/
124. **abstract** **public** ElementState reaction(Element element);
125. }

128. /\*\*
129. \* 玩家对象
130. \*/
131. **public** **class** Player {
132. **private** **int** health = 10;
133. **private** **int** money = 300;
135. **private** **boolean** paused;//暂停所有线程的标志
137. **public** **boolean** isStopped() {
138. **return** stopped;
139. }
141. **private** **boolean** stopped;//切断所有线程的标志
143. **public** Player() {
144. paused = **false**;
145. stopped = **false**;
146. }
148. **public** **boolean** isPaused() {
149. **return** paused;
150. }
152. **public** **void** shiftPaused() {
153. **this**.paused = !paused;
154. }
156. **public** **void** setStopped(**boolean** stopped) {
157. **this**.stopped = stopped;
158. }
160. **public** **int** getHealth() {
161. **return** health;
162. }
164. **public** **int** getMoney() {
165. **return** money;
166. }
168. /\*\*
169. \* 让玩家对象受到伤害
170. \*
171. \* @param damage 伤害值
172. \*/
173. **public** **void** hurt(**int** damage) {
174. health -= damage;
175. }
177. /\*\*
178. \* 增加玩家金币总数
179. \*
180. \* @param money 增加金币数
181. \*/
182. **public** **void** addMoney(**int** money) {
183. **this**.money += money;
184. }
186. /\*\*
187. \* 购买防御塔
188. \*
189. \* @param price 防御塔价格
190. \*/
191. **public** **void** purchase(**int** price) {
192. **if** (money >= price) {
193. money -= price;
194. }
195. }
197. /\*\*
198. \* 用户对象是否还活着
199. \*
200. \* @return 返回是否还活着
201. \*/
202. **public** **boolean** isAlive() {
203. **return** health > 0;
204. }
206. }

Tower的run()方法：

1. /\*\*
2. \* 负责检测有没有靠近的怪物，若有则启动炮弹，发动进攻
3. \*/
4. **public** **void** run() {
5. **while** (!killed) {
6. Point2D target = setTarget();
7. **if** (target != **null**) {
8. //发射动作动画
9. **synchronized** (lock) {
10. **while** (lock.isPaused()) {
11. **try** {
12. lock.wait();
13. } **catch** (InterruptedException e) {
14. e.printStackTrace();
15. }
16. }
17. **if** (lock.isStopped()) **break**;
18. }
20. **for** (**int** i = 0; i < images.length - 3; i++) { //每次循环是一帧
21. attackImgIndex = i;
22. **try** {
23. Thread.sleep(100);//进攻动作图片的切换间隔0.1s
24. } **catch** (InterruptedException e) {
25. e.printStackTrace();
26. }
27. }
28. attackImgIndex = 0;//回到第一帧
30. bullet.setFlyState(0);
31. **double** cos = (**double**) (target.x - centerPosition.x) / centerPosition.distance(target);
32. **double** sin = (**double**) (target.y - centerPosition.y) / centerPosition.distance(target);
33. **while** (bullet.position.distance(target) > FLY\_SPEED) {
34. **synchronized** (lock) {
35. **while** (lock.isPaused()) {
36. **try** {
37. lock.wait();
38. } **catch** (InterruptedException e) {
39. e.printStackTrace();
40. }
41. }
42. **if** (lock.isStopped()) **break**;
43. }
44. bullet.position.x += FLY\_SPEED \* cos;
45. bullet.position.y += FLY\_SPEED \* sin;
46. **try** {
47. Thread.sleep(80);
48. } **catch** (InterruptedException e) {
49. e.printStackTrace();
50. }
51. }
52. bullet.setFlyState(1);
53. bullet.explode();
54. bullet.reset();
56. **try** {
57. //每完整攻击一次后，间隔一段时间再次攻击
58. Thread.sleep((**long**) attackGap[rank] \* 80);
59. } **catch** (InterruptedException e) {
60. e.printStackTrace();
61. }
62. }
63. //休息间隔时间检测一次
64. **try** {
65. Thread.sleep(attackGap[rank]);
66. } **catch** (InterruptedException e) {
67. e.printStackTrace();
68. }
69. }
70. System.out.println("Tower " + **this** + " run out==");
71. }

Monster的run方法：

1. /\*\*
2. \* 怪物线程，当怪物活着时进行位置移动,并且更新元素状态
3. \*/
4. **public** **void** run() {
5. **while** (!killed) {
6. **synchronized** (lock) {
7. **while** (lock.isPaused()) {
8. **try** {
9. lock.wait();
10. } **catch** (InterruptedException e) {
11. e.printStackTrace();
12. }
13. }
14. **if** (lock.isStopped()) **break**;
15. }
16. move();
17. removeElementInLoop();
18. **try** {
19. sleep(TIMER);
20. } **catch** (Exception e) {
21. }
22. }
23. System.out.println(**this** + " is dead");
24. }

Monsters的run()方法：

1. /\*\*
2. \* 该线程负责根据地图类型定时刷怪
3. \*/
4. @Override
5. **public** **void** run() {
6. **int** loc = 0;
7. **int** rows = rule.length;//刷怪的波数
8. **int** cols = 5;//怪物种类数
9. **if** (!stopped) {
10. System.out.println(**this** + "Monsters are being created:");
11. **for** (**int** i = 0; i < rows; i++) {
12. **for** (**int** j = 0; j < cols; j++) {
13. **for** (**int** k = 0; k < rule[i][j]; k++) {//遍历第i波第j个怪的总数
14. **synchronized** (player) {
15. **while** (player.isPaused()) {
16. **try** {
17. player.wait();
18. } **catch** (InterruptedException e) {
19. e.printStackTrace();
20. }
21. }
22. **if** (player.isStopped()) **break**;
23. }
24. **try** {
25. sleep(2000);
26. } **catch** (InterruptedException e) {
27. e.printStackTrace();
28. }
29. monsters[loc] = Monster.getInstance(j, **new** Random().nextBoolean() ? path1 : path2, **this**);
30. **new** Thread(monsters[loc++]).start();
31. }
32. **if** (player.isStopped()) **break**;
33. }
34. **try** {
35. sleep(20000);
36. } **catch** (InterruptedException e) {
37. e.printStackTrace();
38. }
39. **if** (player.isStopped()) **break**;
40. System.out.println("Current I="+i);
41. }
42. **try** {
43. sleep(10000);
44. } **catch** (InterruptedException e) {
45. e.printStackTrace();
46. }
47. }
48. System.out.println( **this** + "刷怪结束");
49. }

AnimationPanel的run()方法：

1. @Override
2. **public** **void** run() {
3. **while** (**true**) {
4. **synchronized** (player) {
5. **if** (player.isPaused()) {
6. **try** {
7. player.wait();
8. } **catch** (InterruptedException e) {
9. e.printStackTrace();
10. }
11. }
12. **if** (player.isStopped()) **break**;
13. }
15. repaint();
16. **try** {
17. Thread.sleep(20);
18. } **catch** (InterruptedException e) {
19. e.printStackTrace();
20. }
21. }
22. System.out.println("animation " + **this**.hashCode() + " run out==");
24. }

Monster.injury:

1. /\*\*
2. \* 怪物受到元素伤害，同时怪物的元素状态会变成三种元素附着、冻结、空其中一种。
3. \* 若元素攻击造成怪物的元素状态发生变化，则会刷新元素状态的附着时间。
4. \* 若怪物死亡，则会被移出生存者列表
5. \*
6. \* @param damage  受到的伤害值
7. \* @param element 攻击的元素类型
8. \*/
9. **public** **void** injury(**int** damage, Element element) {
10. ElementState nextState = currentState.reaction(element);//记录下一状态的元素状态
11. **if** (nextState != currentState) {
12. flagInElementShift = 0;//重置附着时间
13. }
14. currentState = nextState;
15. health -= damage \* currentState.multiplier();//造成元素伤害
16. /\*当元素状态为蒸发和融化时，状态会瞬间发生转移\*/
17. **if** (currentState == ElementState.Evaporation || currentState == ElementState.Melt) {
18. currentState = currentState.reaction(Element.Time);
19. }
20. /\*若怪物生命值小于等于0，则会被判定死亡\*/
21. **if** (health <= 0) {
22. killed = **true**;
23. monsters.kill(**this**);
24. }
25. }

HydroMage.injury

1. /\*\*
2. \* 重载Monster类的run方法,添加了护盾存在时的护盾状态
3. \* @param damage 受击伤害值
4. \* @param element 攻击元素类型
5. \*/
6. @Override
7. **public** **void** injury(**int** damage, Element element) {
8. **if** (shieldHealth > 0) {
9. **double** multiplier;
10. **switch** (element) {
11. **case** Cryo:
12. multiplier = 3;
13. currentState = ElementState.Freeze;
14. **break**;
15. **case** Pyro:
16. multiplier = 1.5;
17. **break**;
18. **case** Hydro:
19. multiplier = 0;
20. **break**;
21. **default**:
22. multiplier = 0.4;
23. }
24. shieldHealth -= multiplier \* damage;
25. } **else** {
26. **super**.injury(damage, element);
27. }
28. }

PyroMage.injury

1. /\*\*
2. \* 重载Monster类的run方法,添加了护盾存在时的护盾状态
3. \* @param damage 受击伤害值
4. \* @param element 攻击元素类型
5. \*/
6. @Override
7. **public** **void** injury(**int** damage, Element element) {
8. **if** (shieldHealth > 0) {
9. **double** multiplier;
10. **switch** (element) {
11. **case** Hydro:
12. multiplier = 3;
13. **break**;
14. **case** Cryo:
15. multiplier = 1.5;
16. **break**;
17. **case** Pyro:
18. multiplier = 0;
19. **break**;
20. **default**:
21. multiplier = 0.4;
22. }
23. shieldHealth -= multiplier \* damage;
25. } **else** {
26. **super**.injury(damage, element);
27. }
28. }

ShieldHilichurl.injury

1. /\*\*
2. \* 重载了injury方法，当护盾存在时，攻击对护盾造成微弱伤害，并更新元素状态
3. \* @param damage  受到的伤害值
4. \* @param element 攻击的元素类型
5. \*/
6. @Override
7. **public** **void** injury(**int** damage, Element element) {
8. **if** (shieldHealth > 0) {
9. currentState = currentState.reaction(element);
10. shieldHealth -= 0.3 \* damage;
11. } **else**
12. **super**.injury(damage, element);
13. }
14. @Override
15. **protected** **void** move() {
16. **super**.move();
17. **if** (currentState == ElementState.PyroAttached) {
18. shieldHealth -= 2;
19. }
20. }

Tower.setTarget

1. /\*\*
2. \* 寻找攻击目标，若找到则返回目标的位置，否则返回空
3. \*
4. \* @return 目标位置，null表示目标不存在
5. \*/
6. **public** Point2D setTarget() {
7. **if** (bullet.getFlyState() != -1) {
8. **return** **null**;
9. }
10. //遍历aliveMonsters中每一个对象
11. **for** (Monster mon : monsters)
12. **if** (mon != **null** && mon.alive() && centerPosition.distance(mon.position) <= explosionArea[rank])
13. **return** **new** Point2D(mon.position.x, mon.position.y);
14. **return** **null**;
15. }

Bullet.explode和Bullet.reset

1. /\*\*
2. \* 炮弹发动攻击,并将状态设置为初始状态
3. \*/
4. **public** **void** explode() {
5. **if** (flyState != 1) {//如果子弹处于攻击状态，
6. **return**;
7. }
8. flyState = 1;
9. //到达目标，对范围内怪物发起元素攻击
10. **for** (Monster i : monsters) {
11. **if** (i != **null** && i.alive() && hitTarget(i)) {
12. i.injury(attack[rank], element);
13. }
14. }
15. **try** {
16. sleep(100);
17. } **catch** (InterruptedException e) {
18. e.printStackTrace();
19. }
21. }

24. **public** **void** reset(){
25. flyState = -1;
26. position = **new** Point2D(initialPosition.x, initialPosition.y);
27. }

Monster.move

1. /\*\*
2. \* 怪物移动规则,方向0、1、2、3分别表示下、左、上、右
3. \*/
4. **protected** **void** move() {
5. /\*若为冻结状态，则直接返回\*/
6. **if** (currentState == ElementState.Freeze || killed) {
7. **return**;
8. }
9. /\*若当前位置距离目标位置小于步长，则将目标位置切换至路径的下一段\*/
10. **if** (position.distance(path[loc]) < step) {
11. **if** (loc == path.length - 1) {//若到达终点，则计入玩家伤害
12. System.out.println(**this**+" is killed here!");
13. monsters.kill(**this**);
14. **return**;
15. }
16. loc++;
17. }
19. /\*更新怪物位置\*/
20. **float** dx = path[loc].x - position.x;
21. **float** dy = path[loc].y - position.y;
22. **if** (dx != 0) {
23. position.x += (dx > 0) ? step : -step;
24. }
25. **if** (dy != 0) {
26. position.y += (dy > 0) ? step : -step;
27. }
29. /\*更新方向\*/
30. **if** (dx > 0.5 \* step) {
31. direction = 3;
32. } **else** **if** (dx < -0.5 \* step) {
33. direction = 1;
34. } **else** {
35. direction = dy < 0 ? 2 : 0;
36. }
37. direction += flagInAnimation ? 4 : 0;//若flagInAnimation为真，则切换同一方向的另一张图片
38. flagInAnimation = !flagInAnimation;
39. }

Monster.removeElementInLoop

1. /\*\*
2. \* 用于刷新怪物附着元素（水、火、冰、冻结）状态。每次被调用时，会使flag值变化一次。每调用25次会解除一次怪物身上附着的元素
3. \*/
4. **protected** **void** removeElementInLoop() {
5. **if** ((flagInElementShift++) == 25) {
6. currentState = currentState.reaction(Element.Time);
7. flagInElementShift = 0;
8. }
9. }

Map

1. **package** res;
3. **import** com.sun.javafx.geom.Point2D;
5. **import** java.awt.\*;
7. **public** **enum** Map {
8. MAP1(
9. 6,
10. **new** Point2D[]{ //这里是塔的位置，第一个参数是塔的数量
11. **new** Point2D(400, 150),
12. **new** Point2D(400, 450),
13. **new** Point2D(700, 150),
14. **new** Point2D(700, 450),
15. **new** Point2D(100, 150),
16. **new** Point2D(100, 450),
18. },
19. **new** Point2D[]{ //path1
20. **new** Point2D(760, -30),
21. **new** Point2D(760, 120-50),
22. **new** Point2D(260, 120-50),
23. **new** Point2D(260, 350-50),
24. **new** Point2D(1160, 350-50)
25. },
26. **new** Point2D[]{ //path2
27. **new** Point2D(-20, 350-40),
28. **new** Point2D(900, 350-40),
29. **new** Point2D(900, 560-40),
30. **new** Point2D(270, 560-40),
31. **new** Point2D(270, 660-40),
32. },
33. Pic.background[Pic.MAP1],
34. **new** **int**[][]{
35. {2, 0, 1, 0, 0},
36. {2, 2, 1, 1, 0},
37. {0, 0, 1, 2, 2},
38. {0, 0, 0, 3, 3},
39. {1, 2, 1, 3, 3}
40. },
41. 30
42. ), MAP2(
43. 8,
44. **new** Point2D[]{
45. **new** Point2D(100, 300 - 50),
46. **new** Point2D(300, 200 - 50),
47. **new** Point2D(300, 400 - 50),
48. **new** Point2D(500, 100 - 50),
49. **new** Point2D(500, 300 - 50),
50. **new** Point2D(600, 200 - 50),
51. **new** Point2D(600, 400 - 50),
52. **new** Point2D(790, 300 - 50),
53. },
54. **new** Point2D[]{
55. **new** Point2D(420, -30),
56. **new** Point2D(420, 300-10),
57. **new** Point2D(230, 300-10),
58. **new** Point2D(230, 500-10),
59. **new** Point2D(720, 500-10),
60. **new** Point2D(720, 200-10),
61. **new** Point2D(920, 200-10),
62. **new** Point2D(920, 400-10),
63. **new** Point2D(1120, 400-10),
64. },
65. **new** Point2D[]{
66. **new** Point2D(420, -30),
67. **new** Point2D(420, 300-10),
68. **new** Point2D(230, 300-10),
69. **new** Point2D(230, 500-10),
70. **new** Point2D(720, 500-10),
71. **new** Point2D(720, 200-10),
72. **new** Point2D(920, 200-10),
73. **new** Point2D(920, 400-10),
74. **new** Point2D(1120, 400-10),
75. },
76. Pic.background[Pic.MAP2],
77. **new** **int**[][]{
78. {2, 0, 1, 0, 0},
79. {2, 2, 1, 1, 0},
80. {0, 0, 1, 2, 2},
81. {0, 0, 0, 3, 3},
82. {1, 2, 1, 3, 3}
83. }, 30
84. );
86. **public** **int** towerN;              //塔的数量
87. **public** Point2D[] plants;          //塔的位置
88. **public** Point2D[] path1;           //第一条路
89. **public** Point2D[] path2;           //第二条路
90. **public** Image background;        //背景图片
91. **public** **int**[][] rule;   //该地图对应的怪物种类和生成次序
92. **public** **int** num;//怪物数量
94. /\*\*
95. \* @param tn     塔的数量
96. \* @param plants 塔的位置坐标数组
97. \* @param path1  第一条路
98. \* @param path2  第二条路
99. \* @param bg     背景图片
100. \* @param mn     二维数组，按顺序表示5种怪的数量和生成次序
101. \* @param num    怪物总数
102. \*/
103. Map(**int** tn, Point2D[] plants, Point2D[] path1, Point2D[] path2, Image bg, **int**[][] mn, **int** num) {
104. **this**.towerN = tn;
105. **this**.plants = plants;
106. **this**.path1 = path1;
107. **this**.path2 = path2;
108. **this**.background = bg;
109. **this**.rule = mn;
110. **this**.num = num;
111. }

114. }

Bullet,Tower,Monster,AnimationPanel的paint

1. /\*\*
2. \* 绘制当前子弹轨迹
3. \*
4. \* @param graphics 沿用调用方法的参数
5. \* @param panel    监听位置
6. \*/
7. **public** **void** paint(Graphics graphics, JPanel panel) {
8. **if** (flyState == -1) {
9. **return**;
10. }
11. graphics.drawImage(Pic.Bullet(element)[flyState], (**int**) position.x, (**int**) position.y, 30, 30, panel);//运动中的图片
12. }

15. /\*\*
16. \* 绘制塔：根据等级rank绘制塔的样式、根据attackImgIndex绘制对应的攻击动作图片
17. \*
18. \* @param g     沿用调用方法的g
19. \* @param panel 观察者
20. \*/
21. **public** **void** paint(Graphics g, JPanel panel) {
22. **if** (setTarget() != **null**) {
23. g.drawImage(Pic.Tower(element)[attackImgIndex + rank], (**int**) imgPosition.x, (**int**) **this**.imgPosition.y, imgSize.width, imgSize.height, panel);
24. }
25. g.drawImage(Pic.Tower(element)[rank], (**int**) **this**.imgPosition.x, (**int**) **this**.imgPosition.y, imgSize.width, imgSize.height, panel);
26. }

29. /\*\*
30. \* 画怪物的图片、血条,若有护盾则画护盾
31. \*
32. \* @param g      沿用调用语句中的g
33. \* @param jPanel 绘制位置
34. \*/
35. **public** **void** paint(Graphics g, JPanel jPanel) {
36. g.drawImage(image[direction], (**int**) position.x - 10, (**int**) position.y - 10, jPanel);//播放第1张图片
37. g.setColor(Color.cyan);
38. g.drawRect((**int**) position.x - 11, (**int**) position.y - 15, 35, 7);//显示血条边框
39. g.setColor(Color.red);
40. g.fillRect((**int**) position.x - 10, (**int**) position.y - 14, (35 \* health / fullHealth()), 6);//显示血条
41. }

44. /\*\*
45. \* 显示游戏主要内容：防御塔、怪物、地图
46. \*
47. \* @param g 画图对象
48. \*/
49. **public** **void** paint(Graphics g) {
50. **super**.paint(g);
51. g.drawImage(map.background, 0, 0, 1182, 703, **this**);
53. /\*绘制玩家信息\*/
54. g.setColor(Color.black);
55. g.fillRect(50, 5, 120, 90);
56. g.setFont(**new** Font("宋", Font.BOLD, 15));
57. g.setColor(**new** Color(251, 251, 21, 250));
58. g.drawString("金币：" + player.getMoney(), 60, 23);
59. g.setColor(Color.RED);
60. g.drawString("生命：" + player.getHealth(), 60, 53);
61. g.setColor(Color.LIGHT\_GRAY);
62. g.drawString("剩余怪物：" + (map.num - monsters.deadN), 60, 83);
64. /\*绘制防御塔和怪物\*/
65. **for** (**int** i = 0; i < towers.length; i++) {
66. **if** (towers[i] != **null**) {
67. towers[i].paint(g, **this**);
68. towers[i].getBullet().paint(g, **this**);
69. } **else** {  //这里表示towers[i]==null，即没有被建造。和isBuilt[i]==false相同（？
70. g.drawImage(Pic.emptyAreaPic[Pic.OUT], (**int**) map.plants[i].x, (**int**) map.plants[i].y, 100, 100, **this**);
71. }
72. }
73. monsters.paint(g, **this**);
74. }