Java 实现游戏模拟实验报告

软件学院软件工程专业 2019 级 2 班 韦诗睿 1913184

一、实验题目

（一）主要内容：使用 Java 运用类、对象、接口内容实现游戏对抗玩法。

（二）主要功能：

实现游戏总流程：主要内容包括设计可供选择的不同游戏模式、不同设定的游戏角色和游戏操作、系统自动选择角色并随机进攻或防守，最后输出用户和系统对抗的结果，完成模拟游戏对抗。

二、设计思路

（一）设定分析：对实验题目中的主要功能分析，本项目需要实现：

1. 游戏的入口和出口。入口：模式选择，相应的游戏类加载和游戏流程函数的调用；出口：一轮游戏结束后选择是否再来一局或退出游戏。

2. 游戏模式：本人设定不同的游戏模式包含不同的角色种类、角色操作和额外的功能。但将游戏模式抽象出来分析，不难得出彼此共性，即玩家数量的设置（人类和系统两位互相对抗），以及游戏的流程函数（完成实际游戏的总流程）。

3. 游戏角色：不同游戏角色都需要对其人设设置必备要素，如角色名字、ID、血量、状态、攻击指数等，区别主要体现在：（1）不同游戏中包含的角色不同，因此角色属于游戏模式；（2）对这些基本要素赋予的初始值不同和具体操作技能不同。

（二）流程分析：本人将实际的游戏流程运用一个综合函数 play 来加载，并在其中调用为必要的操作编写的不同函数，包括创建玩家、设定操作、电脑进行游戏对抗及最后的判定输赢操作。具体操作介绍详见游戏规则。

（三）类设计分析：由上分析可知，游戏模拟的类设置分为主程序、角色设定、游戏操作三个板块。

1. 主要板块：

（1）主程序设定 play 类：实现游戏的入口和出口

（2）角色设定：Actor 类：对必要的成员人设变量进行定义；实现 CanPlay 接口对其中角色的主要操作——攻击和防御进行定义；

（3）游戏操作 Game 类：抽象类，对游戏操作的共性（玩家数量、play 函数）给出模板。

2. 附属板块：

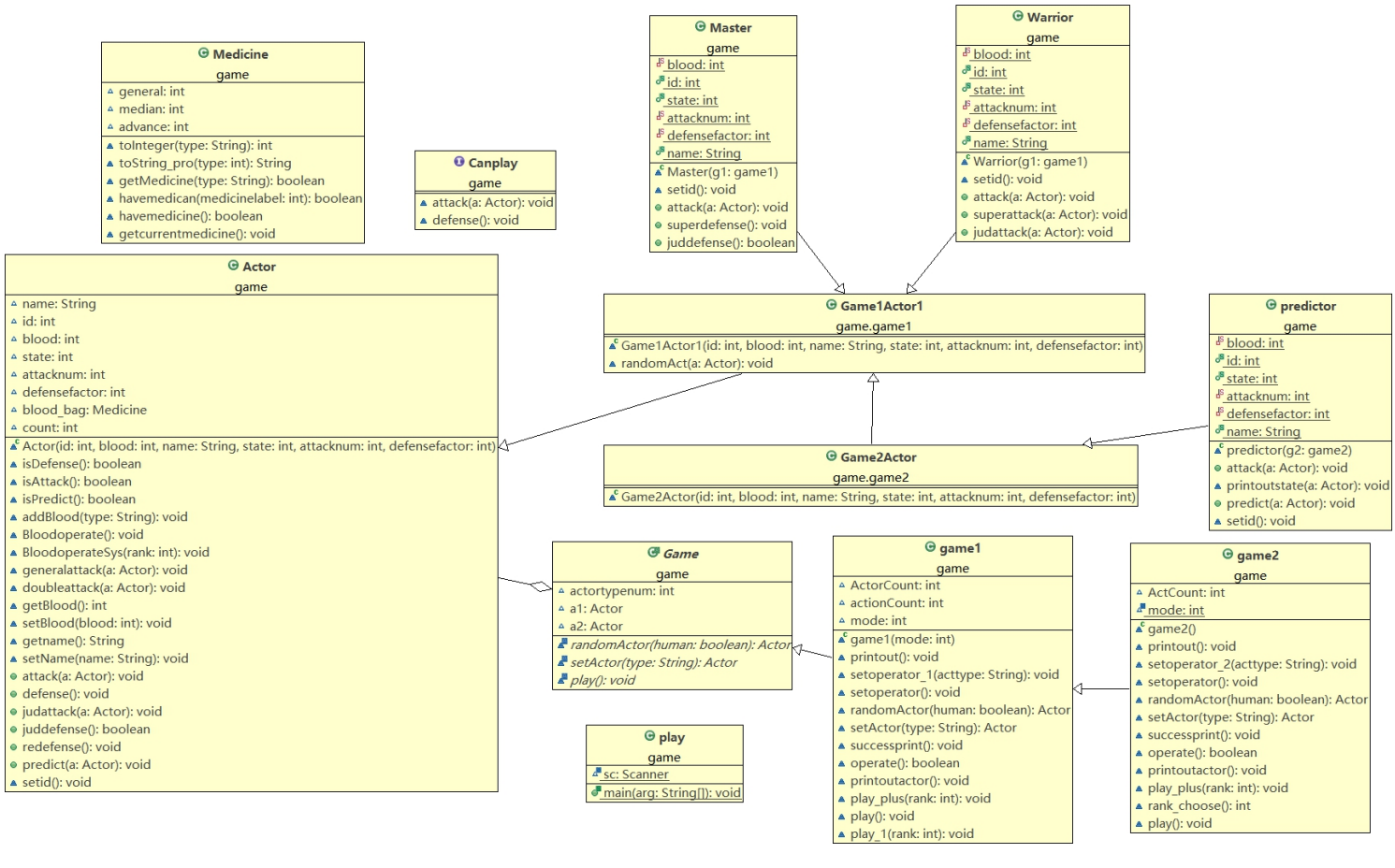
（1）两种游戏模式 game1 和 game2 继承自 Game，并对其中的模板进行实现。虽然在其中的游戏角色设定和 play 函数流程上有所不同，但为了程序的简便性和代码的可复用性，设置 game2 继承自 game1，并对以上不同点的对应函数进行重写。

（2）不同游戏模式中由于角色设定的不同，因此在 game1 和 game2 中设置内部类 Game1Actor1 和 Game2Actor 对附属的游戏角色进行规定，继承自角色基本定义 Actor 类。遵循 game2 继承 game1 的规定，Game2Actor 同样继承自 Game1Actor1。

（3）不同游戏角色设置相应的类对具体的人设值和操作进行定义，有 Master、Warrior 以及 Predictor 三类角色，其中前两类属于 game1，继承自 Game1Actor 1；game2 游戏中添加了 Predictor 角色，继承自 Game2Actor。

（4）额外功能：加血类 Medicine，在游戏模式二中可以在多局游戏中选择加血补充血量。

3. 类图：（由 Eclipse 的 AmaterasUML 生成）



三、游戏规则&游戏测试输出

（一）游戏入口：选择游戏模式

1. 游戏模式设置：

游戏模式 1：game1：基本游戏——一局判定胜负，血量多的获胜。

游戏模式 2：game2：多局游戏——模拟现实游戏，即一方血量为 0 时才判定胜负；

2. 输入设置：限定输入 1、2，进行合法性判断，若输入不在范围内则提示重新输入。

```
Choose game mode:1-one game;2-multiple games:0
Wrong mode!
Choose game mode:1-one game;2-multiple games:1
choose actor or produced randomly:
```

3. 再来一局：当一轮游戏结束判定胜负后，提示是否再来一局，若想再次游戏可重新加载。

```
One more time?-Yes(y) or No(n)y
Choose game mode:1-one game;2-multiple games:1
choose actor or produced randomly:
1-choose yourself; 2-produced randomly
```

4. 游戏难度设定（模式 2）：由于模式 2 为多轮游戏且加入了加血(add blood)操作，因此设置不同 rank 的游戏。

rank = 0 时，easy 模式：系统不会加血而玩家可以选择加血。（后续设计将会保证系统获胜的概率并不会因为不能加血而减少，反而在一定情况下增加。）

rank = 1 时，middle 模式：系统将通过生成随机数选择自动加血或不加血，但只能加三次 General 血包（三种等级血包中最低等级血包）

rank = 2 时，advance 模式：系统拥有的血包和人类玩家相同，且将会从最高级的血包开始增加，保证系统的胜率。

```
choose level you wanna challenge:
0-easy(System player would not add blood automatically),
1-middle(System player would add blood randomly),
2-advance(System player would add blood to win):
0
```

（二）游戏进程：

1. 生成角色：（仅以模式一为展示）

人类玩家：可通过输入 1、2 选择用户手动输入或电脑自动生成，并对输入的角色名字进行合法性判断，若不在范围内则提示重新输入。

电脑玩家：自动生成。由于预言家角色（Predictor）技能的特殊性，因此限定系统玩家无法生成该角色，但玩家选择自动生成时可以生成该角色。

生成结束后会输出人类玩家和电脑玩家的角色名字和 ID。当人类玩家和电脑玩家的名字相同时，将会将电脑玩家的 ID+1。

手动生成：

```
choose actor or produced randomly:
1-choose yourself; 2-produced randomly1
choose Master or WarriorMaster
You are Master1
System is Warrior1
```

自动生成：

```
choose actor or produced randomly:
1-choose yourself; 2-produced randomly2
You are Warrior1
System is Master1
```

合法性测试：

```
choose actor or produced randomly:
1-choose yourself; 2-produced randomly0
Wrong range!
choose actor or produced randomly:
1-choose yourself; 2-produced randomly1
choose Master or Warriorrui
Wrong type! Please enter againWarrior
You are Warrior1
System is Master1
```

2. 角色数值设置：

角色中的数值设置经过大概计算。在 Master 和 Warrior 血量预先规定的基础上，基于规则计算双方对决时的攻击指数和防御指数，同时保证 Master 血量少但 double 攻击时对 Warrior 造成的伤害较高；Warrior 血量多但 general 攻击时对 Master 造成的伤害较小。Predictor 玩家由于技能的特殊性因此攻击指数较低。

（1）Master 玩家：血量：100；攻击指数：300；防御指数：15；

（2）Warrior 玩家：血量：300；攻击指数：300；防御指数：10；

（3）Predictor 玩家：血量：200；攻击指数：250；防御指数：15；

3. 选择操作：选择操作后电脑将会将双方的操作输出。

（1）模式 1：只可以选择攻击（Attack/a）或防御（Defense/d）

一般进攻：进攻造成的对方血量伤害 = 己方攻击指数/对方防守指数

double 进攻：进攻造成的对方血量伤害 = 己方进攻指数*2/对方防守指数

防御：防御状态下己方防守指数+10；该轮结束后-10 返回原值。

1）攻击模式：attack(Actor a)函数，state = 1;

Master 玩家：同类一般进攻，不同类 double 进攻；

Warrior 玩家：同类 double 进攻，不同类 double 进攻

2）防御模式：defense()函数，state = 0;

```
You are Warrior1
System is Warrior1
choose Attack(a) or Defense(d):a
Warrior1 Defense!
Warrior2 Double Attack!!
```

（2）模式 2：

1）在模式 1 的基础上，预言家角色（Predictor）新加了技能预言(predict/p)

预言：由于系统的操作为电脑自动生成，因此可以通过预言技能看到系统的操作（如下图，可看到系统操作为 Defense），并依据该条件重新选择己方攻击或防守，通过对规则的把控可增加人类玩家胜率。而由于该技能对人类玩家优势的增加，因此预言家角色在攻击时只有 general 模式而不会拥有 double Attack 等高级技能。

```
Your blood is:180
choose Attack(a) or Defense(d) or Predict(p):p
Warrior1Defense!
choose Attack(a) or Defense(d) or Predict(p):a
Predictor1 General Attack!
Warrior1 General Attack!
Predictor1Blood:160
Warrior1Blood:250
```

2）超级技能：特定条件下触发，保证了系统和玩家获胜概率大致相等，增加游戏的不确定性（因为攻击力的突然大增有可能会导致意外地先手死亡）和有趣性。该特殊技能不会出现名字而是自动代替 attack/defense 触发。同理，由于预言家角色技能的特殊性，该角色将不会拥有特殊技能。

A. 超级攻击-回光返照（superattack）：只有 Warrior 角色拥有，当 Warrior 角色血量小于 70 时将会触发（因为 Master 的攻击力将会在下一次攻击时造成己方血量-60）。此时攻击指数将为己方攻击指数*4/防御指数。

B. 超级防御（superdefense）：只有 Master 角色拥有，当 Master 角色血量小于 30 时将会触发（因为 Warrior 的攻击力将会在下一次攻击时造成己方血量-25）。此时防御指数将增大至 30，且不会回复至原先指数。

3）加血操作：每一轮操作结束后将会在仍有血包存量的情况下询问是否需要加血。每一个角色在模式二的情况下会自动生成同样数量的血包，分别为：

general 血包：3 个，加血效果+40；

middle 血包：2 个，加血效果+80；

advance 血包：1 个，加血效果+120；

每次选择加血时将会检查是否还有该类型的血包，若没有则加血失败；若还有则加血成功。

添加 general 血包：

```
Predictor1Blood:200
Master1Blood:100
If you want to add your Blood?-Yes(y)/No(n)
y
Your medicine storage:
current medican:
general:3
median:2
advance:1
Choose blood bags type:g/m/a
g
Your blood is:240
```

加血失败：

```
If you want to add your Blood?-Yes(y)/No(n)
y
Your medicine storage:
current medican:
general:2
median:2
advance:0
Choose blood bags type:g/m/a
a
Failure add!Your blood is:360
```

4. 游戏对抗：选择防守的先出招。

（1）模式一：为了提高观赏性，当双方都选择防守时，电脑将会变为攻击模式。

（2）模式二：由于模式二为多轮对战，将取消模式一的设置；并在每次操作时判断是否要触发超级技能。

5. 操作判定

（1）模式 1：一局定胜负，血量多的人获胜；

```
You are Master1
System is Warrior1
choose Attack(a) or Defense(d):a
Master1 Double Attack!!
Warrior1 General Attack!
You lose!
Master1Blood:80
Warrior1Blood:240
```

（2）模式 2：多局游戏，直至一方减至 0 时方能判定输赢。为了防止对多局游戏的疲劳，采取每 5 轮输出提示是否放弃（give up），若选择放弃则会判定人类玩家为输方。

不愿放弃：

```
Your blood is:100
Would you like to give up? -Y(y)/N(n)
n
choose Attack(a) or Defense(d):
```

直到血量为 0 时方能结束：

```
choose Attack(a) or Defense(d):d
Master1 Defense!
Warrior1 General Attack!
You lose!
Master1Blood:-3
Warrior1Blood:210
```

（三）游戏便利性设计：

所有的输入板块都支持全称输入和简写输出，以及大小写不敏感输入，尽最大可能地使每一项输出使用一个按键即可完成，方便用户进行交互。

```
Choose game mode:1-one game;2-multiple games:1
choose actor or produced randomly:
1-choose yourself; 2-produced randomly1
choose Master(m) or Warrior(w)m
You are Master1
System is Warrior1
choose Attack(a) or Defense(d):a
Warrior1 Defense!
Master1 Double Attack!!
You lose!
Master1Blood:100
Warrior1Blood:270
One more time?-Yes(y) or No(n)y
```

```
choose actor or produced randomly:
1-choose yourself; 2-produced randomly2
You are Predictor1
System is Warrior1
choose Attack(a) or Defense(d) or Predict(p):p
Warrior1Defense!
choose Attack(a) or Defense(d) or Predict(p):a
Predictor1 General Attack!
Warrior1 General Attack!
Predictor1Blood:180
Warrior1Blood:275
If you want to add your Blood?-Yes(y)/No(n)
n
Your blood is:180
```

四、心得体会：

在刚开始接到项目任务时觉得像是大一上做的五子棋一样的任务量，刚开始看着老师的类图设计也很茫然，不知道接口、或者 GameActor1 这种中间类有什么作用。但慢慢捋清楚思路后就开始理解这样设计的好处，后又逐渐为了代码可复用抽出一些东西进行重写或添加，最后到想为自己设计的游戏添加一些有意思的东西，如加血操作、超级技能操作等等。平常生活中玩游戏较少，因此对于游戏的一些更有趣的功能实在缺乏想象力，但通过目前已设计的东西在和电脑交互时确实也感受到了许多乐趣。

五、源代码：

GitHub 地址：*https://github.com/RuiNov1st/2021NKU_java_course/tree/main/game*

（一）主程序入口：

```
1.  package game;
2.  import java.util.*;
3.
4.
5.  public class play {
6.   static Scanner sc = new Scanner(System.in);
7.   public static void main(String arg[]) {
8.    //选择是否再玩一局
9.    String choice = "";
10.   //启动游戏
11.   do {
12.
13.   int mode = 0;
14.
15.   //非法字符和游戏模式合法性判断：
16.   boolean judge = true;
17.   mylabels:
```

```
18.     while(true) {
19.      //模式选择
20.      System.out.print("Choose game mode:1-one game;2-multiple games:");
21.      judge = sc.hasNextInt();
22.      if(!judge) {
23.       System.out.print("Wrong type!"+"\n");
24.       sc.nextLine();
25.       continue mylabels;
26.      }
27.      mode = sc.nextInt();
28.      if(mode==1||mode==2) {
29.       break mylabels;
30.      }
31.      System.out.print("Wrong mode!"+"\n");
32.      sc.nextLine();
33.     }
34.
35.     //生成游戏
36.     Game G = new game1(mode);
37.     if(mode==1) {
38.      G = new game1(mode);
39.     }else {
40.       G = new game2();
41.     }
42.
43.     G.play();
44.     //选择是否再玩一局
45.     System.out.print("One more time?-Yes(y) or No(n)");
46.     Scanner sc = new Scanner(System.in);
47.     choice = sc.next();
48.     }while(choice.equals("Yes")||choice.equals("yes")||choice.equals("y")||choice.equals("Y")
    );//大小写不敏感
49.
50.     sc.close();
51.    }
52.   }
```

（二）Actor 角色设置：

```
1.    package game;
2.
3.    import java.util.Scanner;
4.    import java.util.*;
5.
6.    //Canplay:
7.    //设置基本玩法的接口
```

```
interface Canplay{
void attack(Actor a);
void defense();
}

//Medican：
//设置药包种类的不同加不同的血量
//不同的药包有着不同的数量
class Medicine{
 //药包的种类设置
 int general = 3;
 int median = 2;
 int advance = 1;

 //成员函数设置

 //将字符串转化为数字
 int toInteger(String type) {
  switch(type) {
  case "general":return 1;
  case "median":return 2;
  case "advance":return 3;
  }
  return 0;
 }
 //将数字转化为字符串
 String toString_pro(int type) {
  switch(type) {
  case 1: return "general";
  case 2:return "median";
  default:return "advance";
  }
 }

 //获取药包：
 boolean getMedicine(String type) {
 int medicinelabel = 0;
 medicinelabel = this.toInteger(type);
  //看是否还有
  if(havemedican(medicinelabel)) {
   switch(medicinelabel) {
   case 1:general--;break;
   case 2:median--;break;
   case 3:advance--;break;
```

```
52.       }
53.       return true;
54.      }else {
55.       return false;
56.      }
57.     }
58.
59.     //判断某种药是否还有药包：
60.     boolean havemedican(int medicinelabel) {
61.      boolean flag = true;
62.      switch(medicinelabel) {
63.      case 1: if(general==0)flag = false;break;
64.      case 2: if(median==0)flag = false;break;
65.      case 3: if(advance==0)flag = false;break;
66.      }
67.      return flag;
68.     }
69.     //判断所有的药是否还有药包
70.     boolean havemedicine() {
71.      boolean flag = true;
72.      if(general==0&&median==0&&advance==0) {
73.       flag = false;
74.      }
75.      return flag;
76.     }
77.     //输出当前药包的存量
78.     void getcurrentmedicine() {
79.      System.out.print("current medican:\n");
80.      System.out.print("general:"+general+"\n");
81.      System.out.print("median:"+median+"\n");
82.      System.out.print("advance:"+advance+"\n");
83.     }
84.
85.    }
86.
87.
88.
89.
90.   //角色基类:
91.   //1. 实现 CanPlay 接口
92.   //2. 定义基本属性
93.   //3. 定义基础方法
94.   //4. 要被 Game1Actor 继承
95.   class Actor implements Canplay {
```

```
96.  //成员变量
97.  String name;//角色名字
98.  int id;//角色 id
99.  int blood;//角色血量
100. int state;//角色状态：进攻或防守，定义进攻为 1，防守为 0
101. int attacknum;//进攻指数
102. int defensefactor;//防守指数
103. Medicine blood_bag;//血包：附属于每一个角色
104. int count = 3;
105.
106. //构造函数
107. Actor(int id,int blood,String name,int state,int attacknum,int defensefactor){
108.  this.id = id;
109.  this.blood = blood;
110.  this.name = name;
111.  this.state = state;
112.  this.defensefactor = defensefactor;
113.  this.attacknum = attacknum;
114.  blood_bag = new Medicine();//每个角色在赋初值时都附属自己的血包
115. }
116. //成员函数
117.
118. //判断当前是否为防守状态
119. boolean isDefense() {
120.  if(this.state==0) {
121.   return true;
122.  }else {
123.   return false;
124.  }
125. }
126. //判断当前是否为进攻状态
127. boolean isAttack() {
128.  if(this.state==1) {
129.   return true;
130.  }else {
131.   return false;
132.  }
133. }
134. //是否发动预言状态
135. boolean isPredict() {
136.  if(this.state==2) {
137.   return true;
138.  }else {
139.   return false;
```

```
    }
  }
  //加血
  //rui:根据药包种类的不同加不同的血量
  void addBlood(String type) {
   switch(type) {
   case "general":this.blood = this.blood+40;break;
   case "median":this.blood= this.blood+80;break;
   case "advance":this.blood = this.blood+120;break;
   }

  }
  //加血的玩家操作流程
  void Bloodoperate() {
   String jud = "";
   //判断是否还有血包
   //所有的血包都没了
   if(!blood_bag.havemedicine()) {
    System.out.print("There is no any blood bags left.\n");
   }else {
   //还有血包
   System.out.print("If you want to add your Blood?-Yes(y)/No(n)"+"\n");
   Scanner sc4 = new Scanner(System.in);
   //判断输入的合法性，若不合法则直接退出加血，（谁叫你连加血都输错）
   if(sc4.hasNext()) {
    jud = sc4.next();
   }else {
    System.out.print("Wrong Input!"+"\n");
   }
  }
  //大小写不敏感
  if(jud.equals("Yes")||jud.equals("yes")||jud.equals("y")||jud.equals("Y")) {
  //输出当前血包存量：
  System.out.print("Your medicine storage:\n");
  blood_bag.getcurrentmedicine();
  System.out.print("Choose blood bags type:g/m/a\n");
  Scanner sc5 = new Scanner(System.in);
  String type = sc5.next();
  //判断是否还有该类型的血包
  //若有，则添加;没有，则添加失败
  switch(type) {
  case "g":type = "general";break;
  case "m":type = "median";break;
  case "a":type = "advance";break;
```

```
184.    default:break;
185.    }
186.     if(blood_bag.getMedicine(type)) {
187.      this.addBlood(type);
188.     }else {
189.      System.out.print("Failure add!");
190.      }
191.    }
192.  }
193.
194.  //系统玩家加血
195.  //低级玩法：不加血。
196.  //中级玩法：产生随机数看是否加血，加血也只是加最低级的血，加血只能加三次。
197.  //高级玩法：为了赢，则从最大的开始添加
198.  void BloodoperateSys(int rank) {
199.   if(rank==1) {
200.      //中级玩法：
201.      Random r = new Random();
202.      int type = r.nextInt(2);
203.
204.      //当 type = 0 时不加血
205.      if(type!=0) {
206.       if(count>0) {
207.       this.addBlood(blood_bag.toString_pro(1));
208.       count = count-1;
209.      }
210.   }
211.   }else {
212.    if(rank==2) {
213.  //高级玩法：从最高开始加。
214.  //还有血包
215.  if(blood_bag.havemedicine()) {
216.   if(blood_bag.havemedican(3)) {
217.    this.addBlood("advance");
218.   }else {
219.    if(blood_bag.havemedican(2)) {
220.     this.addBlood("median");
221.    }else {
222.     if(blood_bag.havemedican(1))
223.     this.addBlood("general");
224.     }
225.   }
226.  }
227.   }
```

```
228.
229. }
230. }
231.
232. //一般进攻：进攻等于攻击指数/防守指数
233. void generalattack(Actor a) {
234.  int attackindex = this.attacknum/a.defensefactor;
235.  a.blood = a.blood-attackindex;
236. }
237. //double 进攻：进攻等于攻击指数*2/防守指数
238. void doubleattack(Actor a) {
239.  int attackindex = this.attacknum*2/a.defensefactor;
240.  a.blood = a.blood-attackindex;
241. }
242.
243. //获取当前血量
244. int getBlood() {
245.  return this.blood;
246. }
247. //根据角色的不同设置不同的血量
248. void setBlood(int blood) {
249.  this.blood = blood;
250. }
251. //获取角色类型
252. String getname() {
253.  return this.name;
254. }
255. //设置角色名字
256. void setName(String name) {
257.  this.name = name;
258. }
259.
260.
261. //接口的实现，被具体角色重写
262. //attack 函数：更改进攻状态，
263. public void attack(Actor a) {
264. }
265.
266.
267. //defense 函数，更改防守状态
268. public void defense() {
269.  //防守状态下防守指数增加。
270.  this.defensefactor +=10;
271.  this.state = 0;
```

```
272.   //输出当时状态]
273.    System.out.print(this.name+String.valueOf(this.id)+' '+"Defense!"+"\n");
274.
275.   }
276.   public void judattack(Actor a) {
277.    this.attack(a);
278.   }
279.   public boolean juddefense() {
280.    this.defense();
281.    return false;
282.   }
283.   //防守指数状态的恢复
284.   public void redefense() {
285.    this.defensefactor-=10;
286.   }
287.   //在预言家类中重写
288.   public void predict(Actor a) {
289.
290.   }
291.   void setid() {
292.    this.id+=1;
293.    }
294.   }
```

（三）游戏操作：

1. 抽象类 Game

```
1.    package game;
2.
3.    import java.util.*;
4.    //游戏基类
5.    // 1.定义角色数量
6.    // 2.实现角色（被 Game1 继承）
7.    // 3. 定义成员函数（被 Game1 继承）
8.    abstract class Game {
9.     //角色种类数量
10.    int actortypenum = 2;
11.    //角色设置
12.    Actor a1;
13.    Actor a2;//系统玩家
14.
15.    //成员函数
16.    //设置角色类型，创建随机角色（系统角色）
17.    abstract Actor randomActor(boolean human);
18.
```

```
19.    //根据用户输入的角色创建特定角色
20.    abstract Actor setActor(String type);
21.
22.    abstract void play();
23.
24.   }
```

2. 模式 1：Game1

```
1.    package game;
2.
3.    import java.util.*;
4.
5.    //定义游戏 1 的类，继承自 game 类
6.    // 1. 实现 game 类中继承的东西，如角色数量，角色的具体创建（调用 Warrior 或
      Master）
7.    //2.定义 play 方法：a1 和 a2 对抗，选择防守的先出招，输出 blood 比较高的角色
8.    class game1 extends Game {
9.     int ActorCount = 2;
10.    int actionCount = 2;
11.    int mode = 0;
12.    //构造函数无参
13.     game1(int mode){
14.      this.mode = mode;
15.     }
16.
17.    //定义内部类
18.    public class Game1Actor1 extends Actor{
19.     //构造函数
20.     //通过子类构造对象来构造 Actor 类的引用
21.     Game1Actor1(int id,int blood,String name,int state,int attacknum,int defensefactor){
22.     //调用 Actor 构造函数
23.      super(id,blood,name,state,attacknum,defensefactor);
24.     }
25.     //成员函数
26.
27.     //随机操作
28.     void randomAct(Actor a){
29.      //获取随机数
30.      Random r = new Random();
31.      //生成 0 或 1 的随机数并修改 a 的状态
32.      //系统状态不允许成为预言家
33.      a.state = r.nextInt(2);
34.      }
35.     }
36.    //设置输出文字
```

```
37.    void printout() {
38.     //输入操作：
39.     System.out.print("choose Attack(a) or Defense(d):");
40.    }
41.    //将操作设置独立出来
42.    //分为模式 1 和模式 2
43.    void setoperator_1(String acttype) {
44.     //大小写不敏感
45.     //检查输入是否合法
46.    while(!acttype.equals("Attack")&&!acttype.equals("attack")&&!acttype.equals("Defense
    ")&&!acttype.equals("defense")&&!acttype.equals("a")&&!acttype.equals("d")) {
47.      System.out.print("Wrong type! Please choose again!"+"\n");
48.      System.out.print("choose Attack(a) or Defense:");
49.      Scanner sc3 = new Scanner(System.in);
50.      acttype = sc3.next();
51.     }
52.      if(acttype.equals("Attack")||acttype.equals("attack")||acttype.equals("a")) {
53.       a1.state = 1;
54.      }else {
55.       if(acttype.equals("Defense")||acttype.equals("defense")||acttype.equals("d")) {
56.       a1.state = 0;
57.       }
58.      }
59.    }
60.    void setoperator() {
61.     printout();
62.     Scanner sc2 = new Scanner(System.in);
63.     String acttype = sc2.next();
64.
65.     //改变玩家操作状态:
66.     //首先判断 a1 的状态是否没被更改
67.     if(a1.state!=-1) {
68.      a1.state=-1;
69.     }
70.
71.     setoperator_1(acttype);
72.
73.     //系统生成随机操作
74.     Game1Actor1 ga1 = new game1.Game1Actor1(a2.id, a2.blood, a2.name, a2.state, a2.att
    acknum, a2.defensefactor);
75.     ga1.randomAct(a2);
76.
77.    }
78.
```

```
79.
80.    //设置角色类型，创建随机角色（系统角色）
81.    Actor randomActor(boolean human) {
82.     //获取随机数
83.     Random r = new Random();
84.     //生成 0 或 1 的随机数
85.     int i= r.nextInt(2);
86.
87.     if(i==0) {
88.      return new Master(this);
89.     }else {
90.      return new Warrior(this);
91.     }
92.    }
93.    //根据人类输出创建角色
94.    Actor setActor(String type) {
95.     //保证获取正确的角色名字
96.     while(!type.equals("Master")&&!type.equals("Warrior")&&!type.equals("m")&&!type.
    equals("M")&&!type.equals("w")&&!type.equals("W")) {
97.      System.out.print("Wrong type! Please enter again");
98.      Scanner sc = new Scanner(System.in);
99.      type = sc.next();
100.     //sc.close();
101.    }
102.    //根据角色名字创建特定角色
103.    switch(type) {
104.    case "Master":return new Master(this);
105.    case "m":return new Master(this);
106.    case "M":return new Master(this);
107.    default:return new Warrior(this);//默认生成法师
108.    }
109.
110.   }
111.
112.   //判定输赢
113.   void successprint() {
114.    //获取
115.    int a1blood = a1.getBlood();
116.    int a2blood = a2.getBlood();
117.    //模式一：只有一局，谁的血最多谁赢
118.    //判定胜利
119.    if(a1blood==a2blood) {
120.     System.out.print("Drew!\n");
121.    }else {
```

```
122.     if(a1blood>a2blood) {
123.      System.out.print("You are the winner!\n");
124.     }else {
125.      System.out.print("You lose!\n");
126.     }
127.    }
128.
129.    //输出血量
130.    System.out.print(a1.name+String.valueOf(a1.id)+"Blood:"+String.valueOf(a1blood)+"\
        n");
131.    System.out.print(a2.name+String.valueOf(a2.id)+"Blood:"+String.valueOf(a2blood)+"\
        n");
132.   }
133.
134.   //将操作独立出来
135.   boolean operate() {
136.    //当两者的操作相同时
137.    //若两者皆为防守，则人类玩家防守，系统玩家进攻（模式一，为了提高观赏性）
138.    if(a1.isDefense()) {
139.     a1.defense();
140.     a2.attack(a1);
141.     a1.redefense();
142.    }
143.    else {
144.     //人类玩家进攻，系统玩家可进攻可防守
145.     //系统玩家防守
146.     if(a2.isDefense()) {
147.      a2.defense();
148.      a1.attack(a2);
149.      a2.redefense();
150.     }else {
151.     //系统玩家进攻，则人类玩家先手
152.      a1.attack(a2);
153.      a2.attack(a1);
154.     }
155.    }
156.    return true;
157.   }
158.
159.   //输出创建角色名
160.   void printoutactor() {
161.    System.out.print("choose Master(m) or Warrior(w)");
162.   }
163.
```

```
164.  void play_plus(int rank) {
165.   //操作
166.   setoperator();
167.   operate();
168.   //判定输赢
169.   successprint();
170.  }
171.
172.  //设置游戏规则
173.  void play() {
174.   play_1(0);
175.  }
176.  void play_1(int rank) {
177.   //创建玩家
178.   boolean human = false;
179.   mylabels:
180.   while(true) {
181.    System.out.print("choose actor or produced randomly:\n");
182.    System.out.print("1-choose yourself; 2-produced randomly");
183.    Scanner sc = new Scanner(System.in);
184.    int i = 0;
185.    i = sc.nextInt();
186.
187.    //设置变量记录是否人类自动创建角色
188.
189.    //手动创建
190.    if(i==1) {
191.     printoutactor();
192.     Scanner sc1 = new Scanner(System.in);
193.     String actortype = sc1.next();
194.     //人类玩家
195.     a1 = this.setActor(actortype);
196.     break mylabels;
197.    }else {
198.     if(i==2) {
199.     //机器随即创建
200.     human = true;
201.     a1 = this.randomActor(human);
202.     human = false;
203.     break mylabels;
204.     }else {
205.      System.out.print("Wrong range!\n");
206.      continue mylabels;
207.     }
```

```
208.     }
209.   }
210.
211.   //系统玩家
212.    a2 = this.randomActor(human);
213.
214.   //若系统玩家角色和人类玩家相同，需要更改 id：
215.    if(a1.getname().equals(a2.getname())) {
216.     a2.setid();
217.    }
218.
219.   //输出玩家和系统角色
220.    System.out.print("You are"+" "+a1.getname()+String.valueOf(a1.id)+"\n");
221.    System.out.print("System is"+" "+a2.getname()+String.valueOf(a2.id)+"\n");
222.
223.   //出手规则：
224.
225.   //若为模式一，则每次只有一局
226.    play_plus(rank);
227.      }
228. }
```

3. 模式 2：Game2

```
1.    package game;
2.
3.    import java.util.*;
4.
5.    import game.game1.Game1Actor1;
6.
7.    //将 mode=2 分离
8.    class game2 extends game1{
9.     int ActCount = 3;
10.    static int mode = 2;
11.
12.    game2(){
13.     super(mode);
14.    }
15.
16.   //多设置两个角色
17.    public class Game2Actor extends game1.Game1Actor1{
18.     //构造函数
19.     Game2Actor(int id,int blood,String name,int state,int attacknum,int defensefactor){
20.      super(id,blood,name,state,attacknum,defensefactor);
21.     }
22.
```

```
23.     }
24.     //设置输出文字
25.     void printout() {
26.      //输出操作
27.      if(a1.name.equals("Predictor")) {
28.       System.out.print("choose Attack(a) or Defense(d) or Predict(p):");
29.      }else {
30.       System.out.print("choose Attack(a) or Defense(d):");
31.      }
32.     }
33.     //将操作设置独立出来
34.     //分为模式 1 和模式 2
35.     void setoperator_2(String acttype) {
36.      //大小写不敏感
37.      //检查输入是否合法
38.      while(!acttype.equals("Attack")&&!acttype.equals("attack")&&!acttype.equals("Defens
    e")&&!acttype.equals("Predict")&&!acttype.equals("p")&&
39.        !acttype.equals("defense")&&!acttype.equals("a")&&!acttype.equals("d")) {
40.        System.out.print("Wrong type! Please choose again!"+"\n");
41.        System.out.print("choose Attack(a) or Defense(d) or Predict(p):");
42.        Scanner sc3 = new Scanner(System.in);
43.        acttype = sc3.next();
44.       }
45.       if(acttype.equals("Attack")||acttype.equals("attack")||acttype.equals("a")) {
46.         a1.state = 1;
47.       }else {
48.        if(acttype.equals("Defense")||acttype.equals("defense")||acttype.equals("d")) {
49.        a1.state = 0;
50.        }else {
51.         //预言状态
52.         if(acttype.equals("Predict")||acttype.equals("predict")||acttype.equals("p")) {
53.          a1.state = 2;
54.         }
55.        }
56.       }
57.     }
58.     //随机生成操作
59.     void setoperator() {
60.      printout();
61.      Scanner sc2 = new Scanner(System.in);
62.      String acttype = sc2.next();
63.
64.     //改变玩家操作状态:
65.     //首先判断 a1 的状态是否没被更改
```

```
66.     if(a1.state!=-1) {
67.      a1.state=-1;
68.     }
69.
70.     //预言家模式可以有预言操作
71.     if(a1.name.equals("Predictor")) {
72.      setoperator_2(acttype);
73.     }else {
74.     setoperator_1(acttype);
75.     }
76.
77.     //系统生成随机操作
78.     Game1Actor1 ga1 = new game1.Game1Actor1(a2.id, a2.blood, a2.name, a2.state, a2.att
    acknum, a2.defensefactor);
79.     ga1.randomAct(a2);
80.    }
81.    //设置角色类型，创建随机角色（系统角色）
82.    Actor randomActor(boolean human) {
83.    //获取随机数
84.    Random r = new Random();
85.    int i = 0;
86.    if(human==true) {
87.    //生成 0 或 1 的随机数
88.    i= r.nextInt(3);
89.    }else {
90.    //机器不能创建预言家
91.    i = r.nextInt(2);
92.    }
93.
94.    if(i==0) {
95.     return new Master(this);
96.    }else {
97.     if(i==1) {
98.     return new Warrior(this);
99.     }else {
100.     return new predictor(this);
101.    }
102.   }
103.   }
104.   //根据人类输出创建角色
105.   Actor setActor(String type) {
106.   //保证获取正确的角色名字
```

```
107.  while(!type.equals("Master")&&!type.equals("Warrior")&&!type.equals("Predictor")&&!type.equals("m")&&!type.equals("M")&&!type.equals("w")&&!type.equals("W")&&!type.equals("p")&&!type.equals("P")) {
108.  System.out.print("Wrong type! Please enter again");
109.  Scanner sc = new Scanner(System.in);
110.  type = sc.next();
111.  }
112.  //根据角色名字创建特定角色
113.  switch(type) {
114.  case "Master":return new Master(this);
115.  case "m":return new Master(this);
116.  case "M":return new Master(this);
117.  case "Predictor":return new predictor(this);
118.  case "p":return new predictor(this);
119.  case"P":return new predictor(this);
120.  default:return new Warrior(this);//默认生成法师
121.  }
122.
123. }
124.   //判定输赢
125. void successprint() {
126.  //获取
127.  int a1blood = a1.getBlood();
128.  int a2blood = a2.getBlood();
129.
130.  //模式二：打到一方的血小于 0 为止才能判定输赢。
131.  if(a1blood<=0) {
132.  System.out.print("You lose!\n");
133.  }else {
134.  if(a2blood<=0) {
135.  System.out.print("You are the winner!\n");
136.  }
137.  }
138.  //输出血量
139.  System.out.print(a1.name+String.valueOf(a1.id)+"Blood:"+String.valueOf(a1blood)+"\n");
140.  System.out.print(a2.name+String.valueOf(a2.id)+"Blood:"+String.valueOf(a2blood)+"\n");
141.  }
142.
143.  //将操作独立出来
144.  boolean operate() {
145.  //模式二：
146.  boolean flag1 = false;//判断 a1 是否会触发超级防护
```

```
147.  boolean flag2 = false;//判断 a2 是否会触发超级防护
148.  //人类玩家预言状态下
149.  if(a1.isPredict()) {
150.   a1.predict(a2);
151.   //再次进行操作选择
152.   setoperator();
153.  }
154.   if(a1.isDefense()) {
155.    if(a1.juddefense()) {
156.     flag1 = true;
157.    }
158.    if(a2.isAttack()) {
159.     a2.judattack(a1);
160.    }else {
161.     if(a2.juddefense()) {
162.      flag2 = true;
163.     }
164.    }
165.    //若不是触发超级防护，则防守指数需要回复。
166.    if(!flag1) {
167.    a1.redefense();
168.    }
169.    if(!flag2) {
170.    a2.redefense();
171.    }
172.   }else {
173.    //人类玩家进攻，系统玩家可进攻可防守
174.    //系统玩家防守
175.    if(a2.isDefense()) {
176.     if(a2.juddefense()) {
177.      flag2 = true;
178.     }
179.     a1.judattack(a2);
180.     if(!flag2) {
181.      a2.redefense();
182.     }
183.    }else {
184.    //系统玩家进攻，则人类玩家先手
185.     a1.judattack(a2);
186.     if(a2.getBlood()<=0) {
187.      successprint();
188.      return true;
189.     }
190.     a2.judattack(a1);
```

```
191.    }
192.   }
193.   return false;
194.  }
195.
196.  //输出创建角色名
197.  void printoutactor() {
198.   System.out.print("choose Master(m) or Warrior(w) or Predictor(p)");
199.  }
200.  //模式 2，只要没达到双方没血都可以继续打下去
201.  void play_plus(int rank) {
202.   int countmatch = 1;
203.   mylabels:
204.   while(a1.getBlood()>0&&a2.getBlood()>0) {
205.    if(countmatch%5==0) {
206.     System.out.print("Would you like to give up? -Y(y)/N(n)\n");
207.     Scanner sc8 = new Scanner(System.in);
208.     String choice = sc8.next();
209.     if(choice.equals("Y")||choice.equals("y")) {
210.      a1.blood = 0;
211.      successprint();
212.      continue mylabels;
213.     }
214.    }
215.   setoperator();
216.   if(!operate()) {
217.   successprint();
218.   }
219.   if(a1.getBlood()>0&&a2.getBlood()>0) {
220.   //低级模式情况下系统随机加血
221.   a1.Bloodoperate();
222.   a2.BloodoperateSys(rank);
223.   //查看当前血量
224.   System.out.print("Your blood is:"+String.valueOf(a1.getBlood())+"\n");
225.  }
226.   //可以选择放弃比赛。
227.   countmatch++;
228.
229.   }
230.
231.  }
232.  //难度等级选择
233.  int rank_choose() {
234.   //选择难度等级
```

```
235.    System.out.print("choose level you wanna challenge:\n0-easy(System player would not add blood automatically),\n1-middle(System player would add blood randomly),\n2-advance(System player would add blood to win):\n");
236.    Scanner sc5 = new Scanner(System.in);
237.    while(!sc5.hasNextInt()) {
238.     System.out.print("Wrong range!\n");
239.     sc5.nextLine();
240.    }
241.    int rank = sc5.nextInt();
242.    return rank;
243.   }
244.   //独立操作
245.   void play() {
246.    int rank = rank_choose();
247.    play_1(rank);
248.   }
249.
250.  }
```

（四）具体角色设定：

1. Master：

```
1.    package game;
2.
3.    //继承 gam1Actor 类
4.    class Master extends game1.Game1Actor1 {
5.     //参数设置
6.     private static int blood = 100;
7.     public static int id = 1;
8.     public static int state = -1;//默认状态由玩家选择
9.     private static int attacknum = 300;//法师的攻击指数
10.    private static int defensefactor = 15;//法师的防守指数
11.    public static String name = "Master";
12.    //构造函数
13.    Master(game1 g1){
14.     //生成法师角色
15.     g1.super(id, blood, name, state, attacknum, defensefactor);
16.    }
17.    //攻击函数
18.    public void attack(Actor a) {
19.     //设置标签看进攻方式
20.     int label = 0;//默认为general attack
21.
22.     //同类，一般进攻
23.     if(this.getname().equals(a.getname())) {
```

```
24.      this.generalattack(a);
25.
26.     }else {
27.     //不同类，double 进攻
28.      this.doubleattack(a);
29.      label = 1;
30.     }
31.     //输出当前状态
32.     if(label==0) {
33.      System.out.print("Master"+String.valueOf(this.id)+' '+"General Attack!"+"\n");
34.     }else {
35.      System.out.print("Master"+String.valueOf(this.id)+' '+"Double Attack!!"+"\n");
36.     }
37.
38.    }
39.    //超级防护：特定条件触发（血量下降到一定程度时）
40.    //发动超级防御：增大防御指数，且不会回复
41.    public void superdefense() {
42.     defensefactor=30;
43.    }
44.
45.    public boolean juddefense() {
46.    //判断是否会触发超级防护
47.     boolean flag = false;
48.     if(this.getBlood()<30) {
49.      this.superdefense();
50.      this.defense();
51.      flag = true;
52.     }else {
53.      this.defense();
54.      flag = false;
55.     }
56.     return flag;
57.    }
58.
59.   }
```

2. Warrior:

```
1.      package game;
2.
3.      //继承 gam1Actor 类
4.      class Warrior extends game1.Game1Actor1 {
5.       private static int blood = 300;
6.       public static int id = 1;
7.       public static int state = -1;//默认状态由玩家选择
```

```
8.      private static int attacknum = 300;
9.      private static int defensefactor = 10;
10.     public static String name = "Warrior";
11.
12.     //构造函数
13.     Warrior(game1 g1){
14.      //生成勇士角色
15.      g1.super(id, blood, name, state, attacknum, defensefactor);
16.     }
17.     //攻击函数
18.     public void attack(Actor a) {
19.      //设置标签看进攻方式
20.      int label = 0;//默认为general attack
21.      //同类，double 进攻
22.      if(a.getname().equals(this.getname())) {
23.       this.doubleattack(a);
24.       label = 1;
25.      }else {
26.       //不同类，普通进攻
27.       this.generalattack(a);
28.      }
29.      //输出当前状态
30.      if(label==0) {
31.      System.out.print("Warrior"+String.valueOf(this.id)+' '+"General Attack!"+"\n");
32.      }else {
33.      System.out.print("Warrior"+String.valueOf(this.id)+' '+"Double Attack!!"+"\n");
34.      }
35.
36.     }
37.     //超级大招：回光返照（特定条件下触发）
38.     //当血量下降到一定程度时可触发
39.     public void superattack(Actor a) {
40.      int attackindex = 0;
41.      if(a.getname().equals(this.getname())) {
42.      attackindex = attacknum*4/a.defensefactor;
43.      }else {
44.       attackindex = attacknum*2/a.defensefactor;
45.      }
46.      a.blood = a.blood-attackindex;
47.     }
48.     public void judattack(Actor a) {
49.      if(this.getBlood()<70) {
50.       this.superattack(a);
51.      }else {
```

```
52.          this.attack(a);
53.         }
54.        }
55.       }
```

3. Predictor:

```
1.    package game;
2.
3.    class predictor extends game2.Game2Actor {
4.     //参数设置
5.     private static int blood = 200;
6.     public static int id = 1;
7.     public static int state = -1;//默认状态由玩家选择
8.     private static int attacknum = 250;//预言家的攻击指数
9.     private static int defensefactor = 15;//法师的防守指数
10.    public static String name = "Predictor";
11.
12.    //构造函数
13.    predictor(game2 g2){
14.     //生成勇士角色
15.     g2.super(id, blood, name, state, attacknum, defensefactor);
16.    }
17.    //攻击函数
18.    public void attack(Actor a) {
19.     //只有普通进攻
20.     this.generalattack(a);
21.     System.out.print("Predictor"+String.valueOf(this.id)+' '+"General Attack!"+"\n");
22.    }
23.    void printoutstate(Actor a) {
24.     if(a.state==0) {
25.     System.out.print(a.getname()+String.valueOf(a.id)+"Defense!\n");
26.     }else {
27.      System.out.print(a.getname()+String.valueOf(a.id)+"Attack!\n");
28.     }
29.    }
30.    //预言家：查看他人的操作状态
31.    public void predict(Actor a) {
32.     this.state = 2;//预言状态
33.     //查看对手的操作状态。
34.     printoutstate(a);
35.    }
36.    void setid() {
37.     id = id+1;
38.    }
39.   }//1001 行
```