软件工程课程设计实验报告三

组员：陶赟 李宜璟 周尊康　　任课教师：贾东宁

类的实现（分为基本属性、基本方法和高级方法）：

Virus 类基本属性：

id:整型，用于表征特定种类的病毒。

nextId:整型静态变量，用于表示下一个新的病毒种类的编号。

sexInfect:浮点型，用于表示该病毒通过繁衍途径传播的概率。

foodInfect:浮点型，用于表示该病毒通过捕食途径传播的概率。

selfSecure:浮点型，用于表示该病毒自愈的可能性。

deathRate:浮点型，病毒发作致死的概率。

sexInfectBasicPossibility:浮点常量，病毒通过繁衍途径传播的概率的参考值。

foodInfectBasicPossibility:浮点常量，病毒通过捕食途径传播的概率的参考值。

selfSecureBasicPossibility:浮点常量，自愈概率的参考数值。

deathRateBasicPossibility:浮点常量，病发死亡的概率的参考值。

virusOutbreakPossibility:浮点常量，新病毒在某区域爆发的基础概率。

Virus 类基本方法：

getId():返回整型，id 属性的 getter。

getSexInfect():返回浮点型，sexInfect 的 getter。

getFoodInfect():返回浮点型，foodInfect 的 getter。

getSelfSecure():返回浮点型，selfSecure 的 getter。

getDeathRate():返回浮点型，deathRath 的 getter。

Virus(int bioMass, int deathToll):构造函数，基于区域的生物总量和当前死亡总数来定义新病毒的属性。总的来说生物数量越多，区域内死亡的数目越大，出现的病毒就越强烈。

Virus(Virus v):构造函数，克隆出一个新的同类病毒。

equals(Virus):返回布尔值，通过对 id 属性的比较判断两个病毒是否为同一类病毒。

selfSecureTrial():返回布尔值，测试概率事件“自愈”是否发生。

deathTrial():返回布尔值，测试概率事件“病发死亡”是否发生。

sexInfectTrial():返回布尔值，测试概率事件“繁衍传播病毒”是否发生。

foodInfectTrial():返回布尔值，测试概率事件“食物传播病毒”是否发生。

virusOutBreakTest(int biomass, int deathToll):返回布尔值，测试概率事件“区域病毒爆发”是否发生，且发生的概率有一定的浮动，若生物密集，死亡过多，都会有效提高病毒爆发的概率。

Virus 类高级方法：

无。

AntiBody 类基本属性：

id:整型，表示抗体的种类，与病毒是一一对应的。

AntiBody 类基本方法：

getId():返回整型，id 的 getter。

AntiBody(Virus v):构造函数，用于构造克制相应病毒的抗体，令其 id 与病毒相同。

AntiBody(AntiBody a):构造函数，用于克隆抗体，用于抗体的传播到其它实体中。

equals(AntiBody a):返回布尔值，用于检测两个抗体是否是一类的，即 id 相同。

AntiBody 类的高级方法：

无。

Life 类的基本属性：

nextId:整型静态变量，用于表示下一个产生的新物种的标识。

id:整型，用于区别不同的生物种类。

level:浮点型，生命体的物种等级，用于模拟高级生物摄食低级生物的过

程。

foodDemand:浮点型，表示生命体对于自然资源的需求量。

satisfied:布尔型，表示生物是否摄取了足够的食物。

variationPossibility:浮点型常量，表示遗传突变概率。

newKindPossibility:浮点型常量，表示新物种出现的概率。

lowFoodDemand:浮点型常量,用于表示生命体对食物需求的较低水平标准。

lowLevel:浮点型常量，用于表示生命体物种等级的较低水平标准。

virusList:Virus 类的动态列表，用于表示该生命体所携带的病毒。

antiBodyList:AntiBody 类的动态列表，用于表示生命体所携带的抗体。

AntiBody 类的基本方法：

getLevel():返回浮点型，level 的 getter。

getFoodDemand():返回浮点型，foodDemand 的 getter。

getId():返回整型，id 的 getter。

getVirusList():返回 Virus 类动态列表，virusList 的 getter。

getAntiBodyList():返回 AntiBody 类动态列表，antiBodyList 类的 getter。

isSatisfied():返回布尔值，用于确定是否摄取足够的能量。

resetSatisfied():将 satisified 重置为 false。

Life(Life l):构造函数，用于克隆出完全相同的生命，在游戏初始化时及生成新物种时需要用到。

Life(double levelToSet,double foodDemandToSet):构造函数,用于构造出具有特定 level 和 foodDemand 的生命体，在游戏的初始化过程中需要用到。

Life():构造函数，用于随机生成一个新的弱小物种。

eat(double amount):返回布尔值，输入区域内剩余的食物量，返回该生物进食后区域的资源剩余量，并同时设置生物是否摄取了足量的食物。

hunt(Life l):返回布尔值，模拟向任一生物发起捕食的过程，返回是否捕食成功，且会同时设置是否已摄取足量食物。

selfImmune():比照 virusList 和 antiBodyList，并移出已经被免疫的病毒。

selfDuplicate():对 virusList 和 antiBodyList 进行去重处理。

deathTrial():返回布尔值，检测概率事件“生命体病发致死”是否发生。

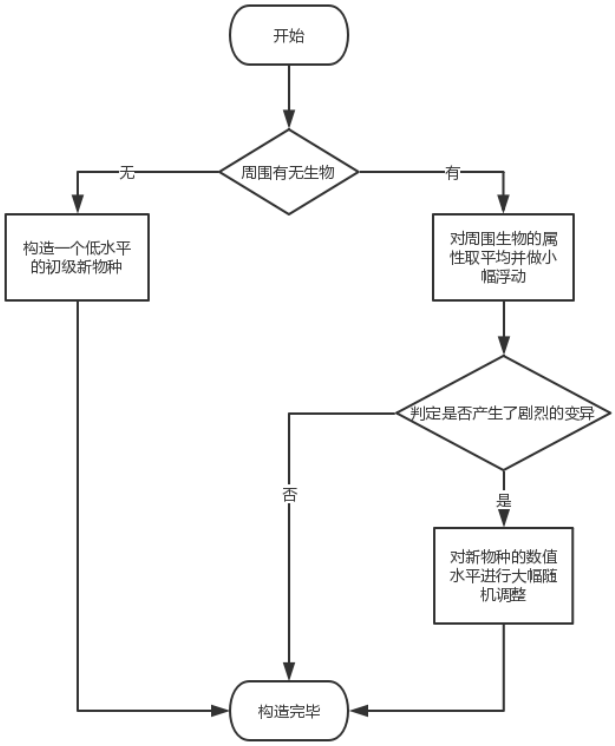
newKindTrial():静态函数，返回布尔值，检测概率事件“新物种诞生”是否发生。

equals():返回布尔值，依据 id 来判断两个生命体是否为同一物种。

Life 类的高级方法：

Life(ArrayList<Life>l,Section s):构造函数，依据特定位置周围若干个点的生物和所处区域的环境情况生成新的物种。

具体思路：



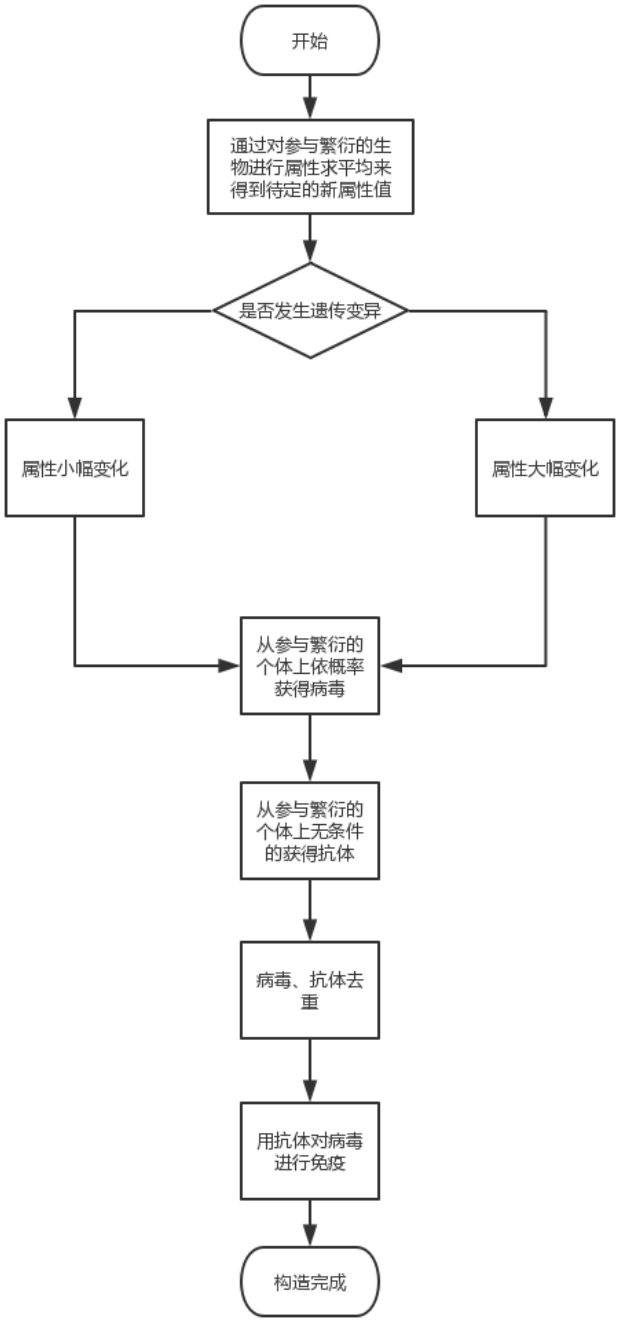
代码实现：

```
1.  public Life(ArrayList<Life> l, Section s){
2.      if(l.size() == 0){
3.          foodDemand = s.getProductivity()/(double)s.getFittestNumOfLife(
    );
4.          satisfied = false;
5.          level = Life.lowLevel * (Math.random() + 9.5)/10.0;
6.          virusList = new ArrayList<Virus>();
```

```
7.          antiBodyList = new ArrayList<AntiBody>();
8.      }
9.      else{
10.         foodDemand = s.getProductivity()/(double)s.getFittestNumOfLife(
    );
11.         int num = l.size();
12.         double sumFoodDemand = 0;
13.         double sumLevel = 0;
14.         virusList = new ArrayList<Virus>();
15.         antiBodyList = new ArrayList<AntiBody>();
16.         int i;
17.         for(i = 0; i < l.size(); i++){
18.             sumFoodDemand += l.get(i).getFoodDemand();
19.             sumLevel += l.get(i).getLevel();
20.         }
21.         double rate;
22.         if(variationTrial()){
23.         /* 借用下遗传变异的概念 */
24.             rate = (Math.random() - 0.5) * 2.0 + 1.0;
25.         }
26.         else{
27.             rate = (Math.random() + 9.5)/10.0;
28.         }
29.         foodDemand = (sumFoodDemand/(double)num) * rate;
30.         level = (sumLevel/(double)num) * rate;
31.         satisfied = false;
32.     }
33.     id = nextId;
34.     Life.nextId++;
35. }
```

Life(ArrayList<Life>l):构造函数，依据特定位置周围若干个点的生物来决定该点繁衍出哪个生命及其属性。

具体思路：


开始
通过对参与繁衍的生物进行属性求平均来得到待定的新属性值
是否发生遗传变异
属性小幅变化
属性大幅变化
从参与繁衍的个体上依概率获得病毒
从参与繁衍的个体上无条件的获得抗体
病毒、抗体去重
用抗体对病毒进行免疫
构造完成

代码实现：

```java
public Life(ArrayList<Life> l){
    id = l.get(0).getId();
    int num = l.size();
    double sumFoodDemand = 0;
    double sumLevel = 0;
    virusList = new ArrayList<Virus>();
    antiBodyList = new ArrayList<AntiBody>();
    int i;
    for(i = 0; i < l.size(); i++){
        sumFoodDemand += l.get(i).getFoodDemand();
        sumLevel += l.get(i).getLevel();
    }
    double rate;
    if(variationTrial()){
        rate = (Math.random() - 0.5) * 2.0 + 1.0;
    }
    else{
        rate = (Math.random() + 9.5)/10.0;
    }
    foodDemand = (sumFoodDemand/(double)num) * rate;
    level = (sumLevel/(double)num) * rate;
    satisfied = false;
    for(i = 0;i <l.size(); i++){
        Life ll = l.get(i);
        ArrayList<Virus> v = ll.getVirusList();
        ArrayList<AntiBody> a = ll.getAntiBodyList();
        int j;
        for(j = 0; j < v.size(); j++){
            if(!virusList.contains(v.get(j))){
                if(v.get(j).sexInfectTrial()){
                    virusList.add(new Virus(v.get(j)));
                }
            }
        }
        for(j = 0; j < a.size(); j++){
            if(!antiBodyList.contains(a.get(j))){
                antiBodyList.add(new AntiBody(a.get(j)));
            }
        }
        selfImmune();
    }
}
```

Section 类的基本属性：

productivity:浮点型，表示一个区域内的生产力，每回合应根据生产力重置出等量的资源供生物摄取。

fittestNumOfLife:整型，表示区域内最适合生存的生物数量。也即环境承载力。

roundProductivity:浮点型，表示当前轮次内该区域还剩下的资源数量。

numOfLife:整型，表示当前区域内的生命总数。

numOfBirth:整型，表示当前区域内新诞生的生命总数。

numOfDeath:整型，表示当前区域内在该轮次消亡的生命总数。

productivityFloor:浮点常量，表示区域环境最恶化时其生产力的下限。

productivityUtmost:浮点常量，表示区域环境最优时其生产力的上限。

defaultFittestNumOfLife:整型常量，表示区域内默认的最适生命数。

defaultProductivity:浮点常量，表示一般性区域的生产力。

Section 类的基本方法：

getProductivity():返回浮点型，productivity 的 getter。

getFittestNumOfLife():返回整型，fittestNumOfLife 的 getter。

setNumOfLife(int s):将区域内的 numOfLife 设置为特定的整数 s。

getNumOfLife():返回整型，numOfLife 的 getter。

getRemainingProduct():返回浮点型，roundProductivity 的 getter。

consumeRemainingProduct(double d):将 roundProductivity 减去 d，d 的合法性由外部保证。

addNumOfBirth(int i):将区域内的 numOfBirth 加上 i。

addNumOfDeath(int i):将区域内的 numOfDeath 加上 i。

getNumOfDeath():返回整型，numOfDeath 的 getter。

getNumOfBirth():返回整型，numOfBirth 的 getter。

Section():构造函数，依据类中自带的常量来生成一个具有默认属性的区域。

Section(double productivityToSet, int fittestNumOfLifeToSet):构造函数，生成一个具有特定初始生产力和特定环境承载力的区域。

resetStatus():先模拟环境生产力的变化，再根据环境生产力重置一次环境内资源量，并将区域内的计数器重置。

virusOutBreakTrial():返回整型，将区域内的数据信息打包发送给病毒类测试区域内是否爆发新的病毒。

virusOutBreak(Life l):依据区域内的信息构造出新病毒，并将新病毒发送给上一级对象传来的生命体中。

Section 类的高级方法：

无。

Plat 类的基本属性：

lifeColor:Color 类数组，用于根据 id 来匹配相应的颜色。

lifeMap:Life 类二维数组，用于存储和表示生命在地图上的分布。

sectionMap:Section 类二维数组，用于存储地图上的分区。

newLife:Life 类动态列表，用于存储该回合内新诞生的物种的副本。

newVirus:Virus 类动态列表，用于存储该回合内新出现的病毒的副本。

newAntiBody:AntiBody 类动态列表，用于存储该回合内出现的新抗体的副本。

eatingAbilityComparator:Comparator 类对象，用于比较生命体的摄食能力。规则为生命体需要的资源越多，其摄取自然资源能力越低。

Plat 类的基本方法：

Plat():构造函数，为引用类型方法分配初始化空间。

getSectionMap():返回 Section 类的二维数组的引用。

sortEatingAbility(ArrayList<Life> l):对 Life 类动态列表进行排序，排序依赖于 eatingAbilityComparator。

getLifeAround(int x,int y):返回 Life 类动态列表，从 lifeMap 的(x,y)处搜寻周围的 8 格，若有生命体便将其加入待返回列表中。

selectLifeRandomly(int x,int y):返回 Life 类对象或者空指针。在区域(x,y)内随机选出一个生命体。

locateSection(int x, int y):返回 Section 类对象。返回坐标(x,y)所处的

Section 的引用。

selectSpaceRandomly(int x, int y):返回一个长度为二的数组，从(x,y)区域找到一个空的格子，并返回它的坐标，若没有找到，则返回[-1,-1]。

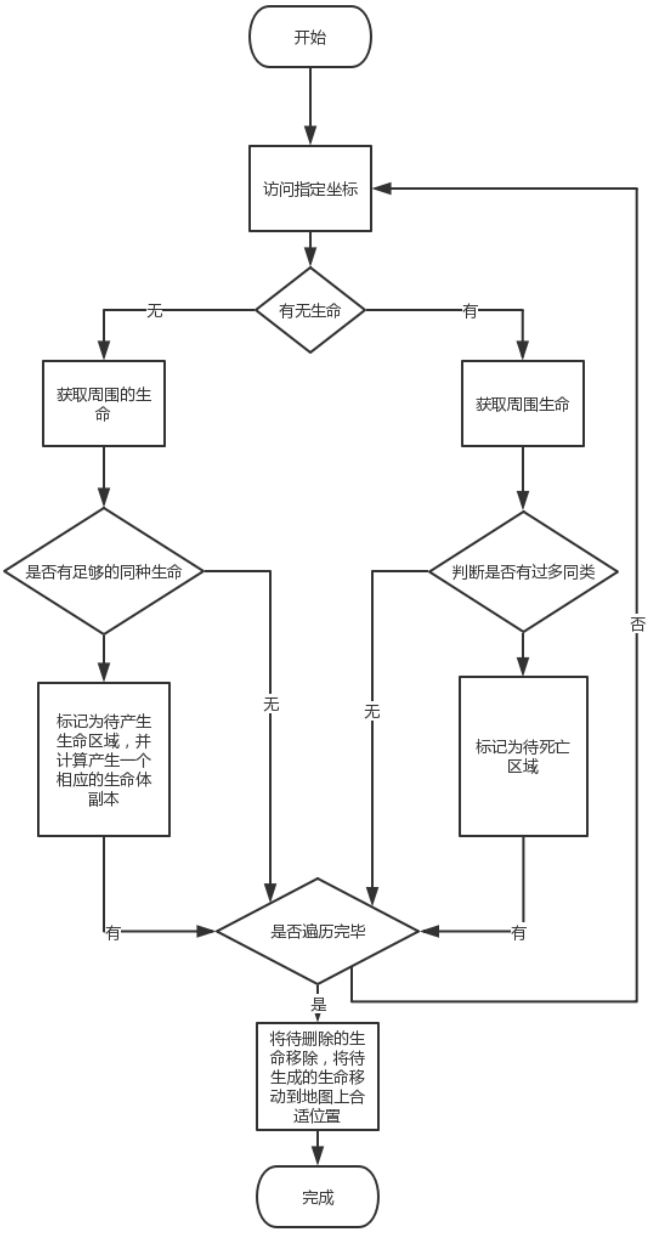
selectSectionRandomly():返回 Secion 类对象，从地图上随机选择一个区域的引用并返回。

resetStage():将 lifeMap 上的生物的饥饿度重置，并进行病毒去重，自我免疫。

Plat 类的高级方法：

reproductionAndWitherStage():繁衍/自然死亡阶段

具体思路：:



代码实现：

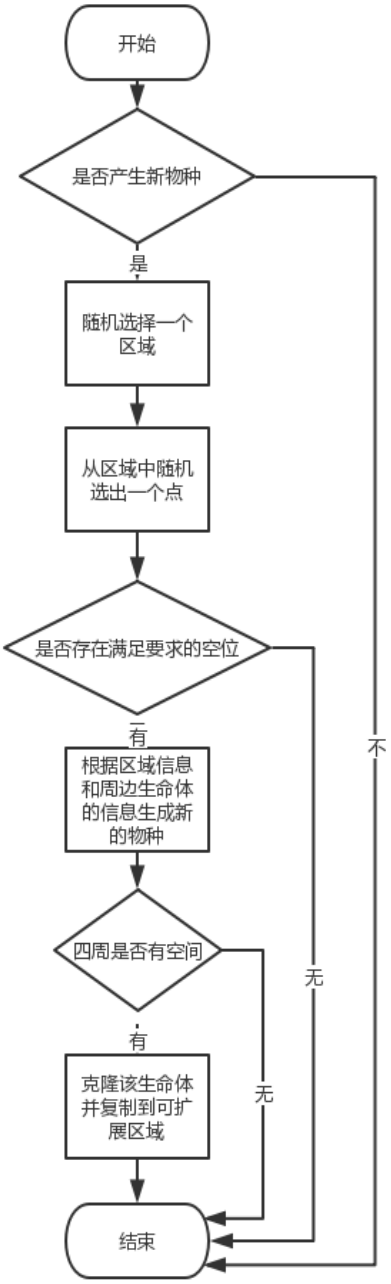
```
public void reproductionAndWitherStage(){
/* 繁衍/自然死亡阶段 */
    int i,j;
    int[][] rep = new int[10][10];
    int[][] die = new int[10][10];
    for(i = 0; i < 10; i++){
        for(j = 0; j < 10; j++){
            rep[i][j] = 0;
            die[i][j] = 0;
        }
    }
    ArrayList<Integer> repXs = new ArrayList<Integer>();
    ArrayList<Integer> repYs = new ArrayList<Integer>();
    ArrayList<Life> repLs = new ArrayList<Life>();
    for(i = 0; i < 100; i++){
        for(j = 0; j < 100; j++){
            if(lifeMap[i][j] == null){
                ArrayList<Life> l = getLifeAround(i,j);
                if(l.size() > 1){
                    int k;
                    boolean finded = false;
                    Life tester = null;
                    for(k = 0; k < l.size(); k++){
                        tester = l.get(k);
                        int m;
                        for(m = k + 1; m < l.size(); m++){
                            if(tester.equals(l.get(m))){
                                finded = true;
                                break;
                            }
                        }
                        if(finded){
                            break;
                        }
                    }
                    int m;
                    for(m = 0; m < l.size(); m++){
                        if(!tester.equals(l.get(m))){
                            l.remove(m);
                            m--;
                        }
                    }
```

```
                    if(!finded){
                        continue;
                    }
                    else{
                        rep[i/10][j/10]++;
                        repXs.add(i);
                        repYs.add(j);
                        repLs.add(new Life(l));
                        if(repXs.size() != repLs.size())System.out.print
ln(88);
                        l.clear();
                    }
                }
            }
        }
    }
    /* 消亡↓ */
    ArrayList<Integer> lifeXToRemove = new ArrayList<Integer>();
    ArrayList<Integer> lifeYToRemove = new ArrayList<Integer>();
    for(i = 0; i < 100; i++){
        for(j = 0; j < 100; j++){
            if(lifeMap[i][j] != null){
                ArrayList<Life> lifeAround = getLifeAround(i, j);
                int k;
                int cnt = 0;
                for(k = 0; k < lifeAround.size(); k++){
                    if(lifeAround.get(k).getId() == lifeMap[i][j].getId(
)){
                        cnt++;
                    }
                }
                if(cnt == 0 || cnt >= 4){
                    die[i/10][j/10]++;
                    lifeXToRemove.add(i);
                    lifeYToRemove.add(j);
                }
            }
        }
    }
    /* 消亡↑ */
    for(i = 0; i < repLs.size(); i++){
        lifeMap[repXs.get(i)][repYs.get(i)] = repLs.get(i);
    }
    for(i = 0; i < lifeXToRemove.size(); i++){
```

```
85.         lifeMap[lifeXToRemove.get(i)][lifeYToRemove.get(i)] = null;
86.     }
87.     for(i = 0; i < 10; i++){
88.         for(j = 0; j < 10; j++){
89.             sectionMap[i][j].addNumOfBirth(rep[i][j]);
90.             sectionMap[i][j].addNumOfDeath(die[i][j]);
91.         }
92.     }
93. }
```

newKindStage():新物种出现的阶段。

实现思路：



代码实现：

```
1.  public void newKindStage() {
2.      /* 新物种出现的阶段 注意那个函数应该接受一个区域作为参数 */
3.      if (Life.newKindTrial()) {
4.          Section s = selectSectionRandomly();
5.          int x, y;
6.          x = y = 0;
7.          int i, j;
8.          for (i = 0; i < 10; i++) {
9.              for (j = 0; j < 10; j++) {
10.                 if (sectionMap[i][j] == s) {
11.                     x = i;
12.                     y = j;
13.                 }
14.             }
15.         }
16.         int[] pair;
17.         pair = selectSpaceRandomly(x, y);
18.         if (pair[0] >= 0) {
19.             ArrayList<Life> l = getLifeAround(pair[0], pair[1]);
20.             Life newOne = new Life(l, s);
21.             newLife.add(newOne);
22.             lifeMap[pair[0]][pair[1]] = newOne;
23.             s.addNumOfBirth(1);
24.             x = pair[0];
25.             y = pair[1];
26.             for (i = x - 1; i <= x + 1; i++) {
27.                 for (j = y - 1; j <= y + 1; j++) {
28.                     if (i < 0 || j < 0 || i > 99 || j > 99 || (i == x &&
    j == y) || lifeMap[i][j] != null) {
29.                         continue;
30.                     }
31.                     else {
32.                         if (Math.random() < 0.8) {
33.                             lifeMap[i][j] = new Life(newOne);
34.                             s.addNumOfBirth(1);
35.                         }
36.                     }
37.                 }
38.             }
39.         }
40.     }
41. }
```

eatingStage():竞争自然资源

具体思路：先利用排序功能，将各个区域内的生命体按照摄食能力进行排序成为一个列表，再按顺序将资源优先分配给摄食能力靠前的生命体。

代码实现：

```
1.   public void eatingStage(){
2.   /* 竞争自然资源 延伸出的需求:区域资源重置 */
3.      int i,j;
4.      for(i = 0;i < 10; i++){
5.          for(j = 0; j < 10; j++){
6.              Section s = sectionMap[i][j];
7.              ArrayList<Life> l = new ArrayList<Life>();
8.              int p,q;
9.              for(p = i * 10; p < (i + 1) * 10; p++){
10.                 for(q = j * 10; q < (j + 1) * 10; q++){
11.                     if(lifeMap[p][q] != null){
12.                         l.add(lifeMap[p][q]);
13.                     }
14.                 }
15.             }
16.             sortEatingAbility(l);
17.             int k;
18.             for(k = 0; k < l.size(); k++){
19.                 double r = s.getRemainingProduct();
20.                 s.consumeRemainingProduct(l.get(k).eat(r));
21.                 if((int)s.getRemainingProduct() == 0){
22.                     break;
23.                 }
24.             }
25.         }
26.     }
27. }
```

huntingStage():捕食阶段模拟。

具体思路：先遍历 lifeMap 上的各个 Life 对象，若出于未饱和状态，则从该点周围寻找其他生命，找到其中等级与之相差至少 1.0 且非同种生命，将其移除，并将 Life 对象设置为饱和状态，若未能找到，则依旧保持未饱和状态。

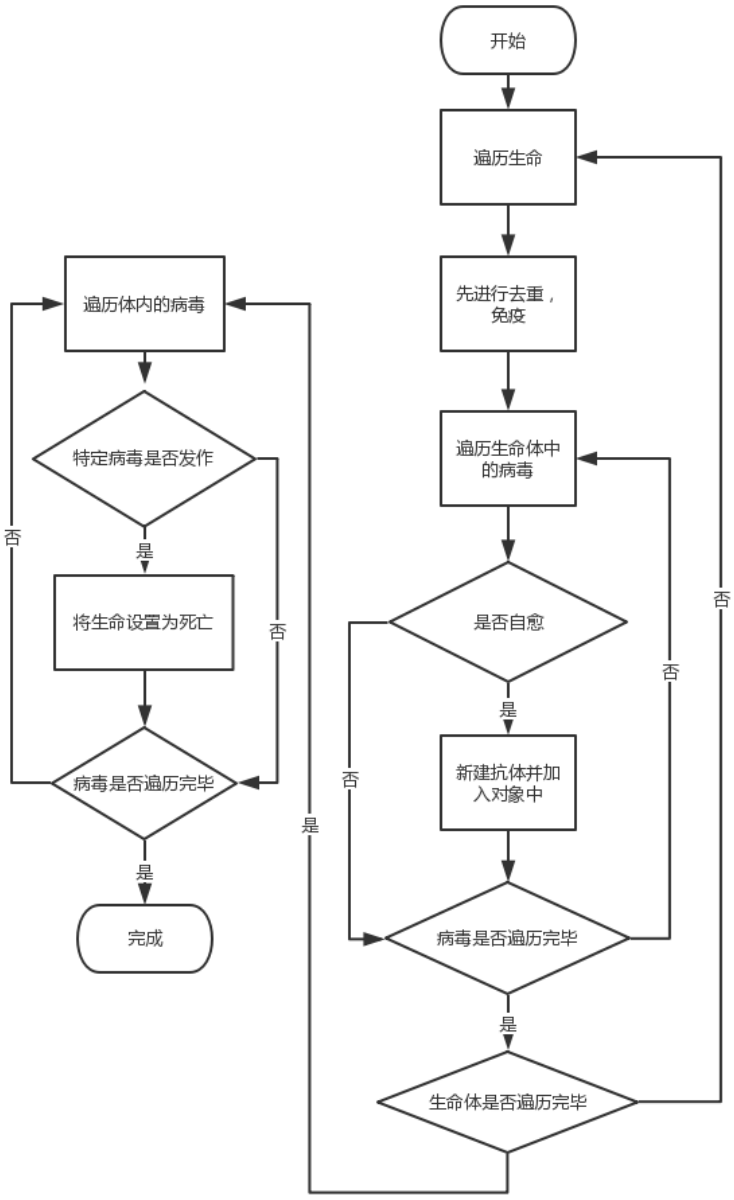
代码实现：

```
1.  public void huntingStage(){
2.     /* 掠食其他生物的阶段 */
3.      int i,j;
4.      for(i = 0; i < 100; i++){
5.          for(j = 0; j < 100; j++){
6.              if(lifeMap[i][j] != null && !lifeMap[i][j].isSatisfied()){
7.                  ArrayList<Life> l = getLifeAround(i, j);
8.                  int k;
9.                  for(k = 0; k < l.size(); k++){
10.                     if(l.get(k).getId() != lifeMap[i][j].getId()){
11.                         if(lifeMap[i][j].hunt(l.get(k))){
12.                             int p,q;
13.                             for(p = i - 1;p <= i + 1; p++){
14.                                 for(q = j - 1; q <= j + 1; q++){
15.                                     if(p < 0 || p > 99 || q < 0 || q > 9
    9 || p == i && q ==j){
16.                                         continue;
17.                                     }
18.                                     if(lifeMap[p][q] == l.get(k)){
19.                                         lifeMap[p][q] = null;
20.                                         locateSection(p,q).addNumOfDeath
    (1);
21.                                     }
22.                                 }
23.                             }
24.                             break;
25.                         }
26.                     }
27.                 }
28.             }
29.         }
30.     }
31. }
```

hungerStage():遍历 lifeMap 上的每一个有生命的点，若该点处生命未饱和，则记为死亡（不方便加回去影响排版了，所以写在这儿）。

saveAndDieStage():自愈及病发致死阶段。

具体思路：



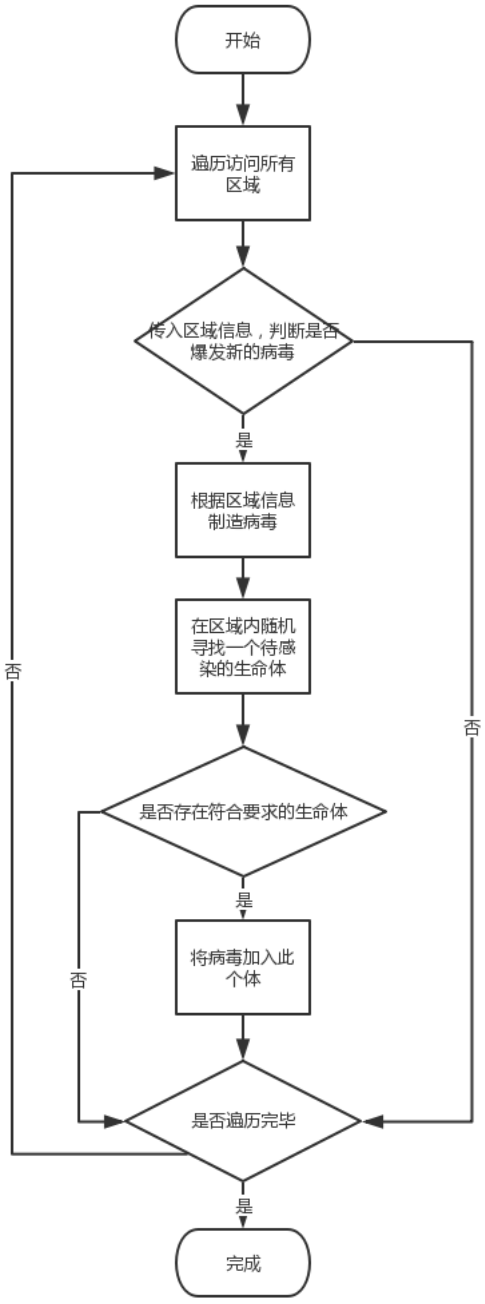
代码实现：

```
1.  public void saveAndDieStage(){
2.  /* 自愈及死亡阶段，注意应该在事前做好排除重复，自我免疫检测 */
3.      int i,j;
4.      for(i = 0; i < 100; i++){
5.          for(j = 0; j < 100; j++){
6.              if(lifeMap[i][j] != null){
7.                  lifeMap[i][j].selfDuplicate();
8.                  lifeMap[i][j].selfImmune();
9.                  ArrayList<Virus> v = lifeMap[i][j].getVirusList();
10.                 ArrayList<AntiBody> a = lifeMap[i][j].getAntiBodyList();

11.                 int k;
12.                 for(k = 0 ; k < v.size(); k++){
13.                     boolean help = false;
14.                     Virus vv = v.get(k);
15.                     ArrayList<Life> l = getLifeAround(i, j);
16.                     int p;
17.                     for(p = 0; p < l.size(); p++){
18.                         ArrayList<Virus> vl = l.get(p).getVirusList();
19.                         if(vl.contains(vv)){
20.                             help = true;
21.                         }
22.                     }
23.                     if(v.get(k).selfSecureTrial() || help){
24.                         AntiBody anti = new AntiBody(v.get(k));
25.                         a.add(anti);
26.                         newAntiBody.add(anti);
27.                         v.remove(k);
28.                         k--;
29.                     }
30.                 }
31.                 for(k = 0; k < v.size(); k++){
32.                     if(v.get(k).deathTrial()){
33.                         lifeMap[i][j] = null;
34.                         locateSection(i,j).addNumOfDeath(1);
35.                     }
36.                 }
37.             }
38.         }
39.     }
```

newVirusStage():新病毒尝试爆发。

实现思路：

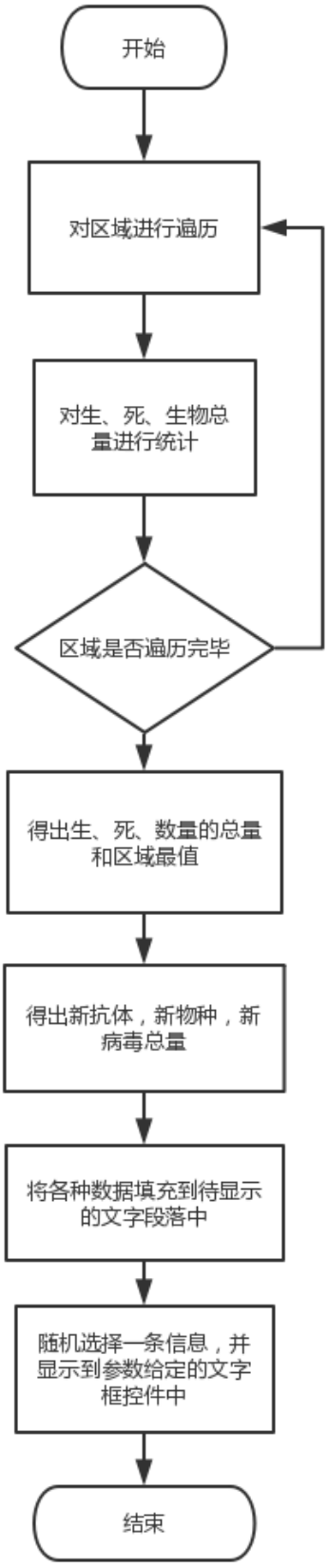


代码实现：

```
1.  public void newVirusStage(){
2.  /* 新病毒尝试爆发阶段 */
3.      int i,j;
4.
5.      for(i = 0; i < 10; i++){
6.
7.          for(j = 0; j < 10; j++){
8.
9.              if(sectionMap[i][j].virusOutBreakTrial()){
10.
11.                 Virus v = new Virus(sectionMap[i][j].getNumOfLife(),sect
    ionMap[i][j].getNumOfDeath());
12.
13.                 Life l = selectLifeRandomly(i,j);
14.
15.                 if(l != null){
16.
17.                     l.getVirusList().add(v);
18.
19.                     newVirus.add(v);
20.
21.                 }
22.
23.             }
24.
25.         }
26.
27.     }
28.
29. }
```

countStage(JTextField textfield):汇总一轮中的数据，并将其填充到控件中。

具体思路：


开始
对区域进行遍历
对生、死、生物总量进行统计
区域是否遍历完毕
得出生、死、数量的总量和区域最值
得出新抗体，新物种，新病毒总量
将各种数据填充到待显示的文字段落中
随机选择一条信息，并显示到参数给定的文字框控件中
结束

代码实现：（代码过长，可于课设的最后一份报告的附录中找到）

sectionRefreshStage():自然环境演替阶段，遍历 sectionMap 并对其分别调用 resetStatus()即可。

resetStage():重置全图所有区块信息，遍历 sectionMap 并调用相应的重置，去重函数即可。

Controller 类的基本属性：

landform:整型，用于选择预设地形。

p:Plat 单例对象，即游戏的数据载体。

frame:JFrame 对象，游戏的显示载体。

textfield:JTextField 对象，游戏过程中文字输出的区域。

runnable:Runnable 对象，存储了游戏执行一轮所要执行的代码段落。

thread:Thred 对象，用于开启独立线程反复执行游戏轮次，并允许进行暂停和继续。

panel:JPanel 对象，用于进行简单的图形绘制。

threadIsOpen:布尔值，用于表示游戏自动运行的过程是否开启，默认为 false。

Controller 类的基本方法：

moveARound():执行一轮游戏模拟过程。依次调用数据载体 p 的 reproductionAntWitherStage()、newKindStage()、eatingStage()、huntingStage()、hungerStage()、saveAndDieStage()、newVirusStage()。方法即可。

showMeaage():执行信息的输出和内部数据的调整。依次调用数据载体 p 的 countStage()、sectionRefreshStage()、resetStage()方法即可。

save():将环境中存在的对象存入指定的.txt 文件中即可，调用 ObjectOutputStream 类对象的 writeObject(Object o)方法即可将相关数据写入文件中。

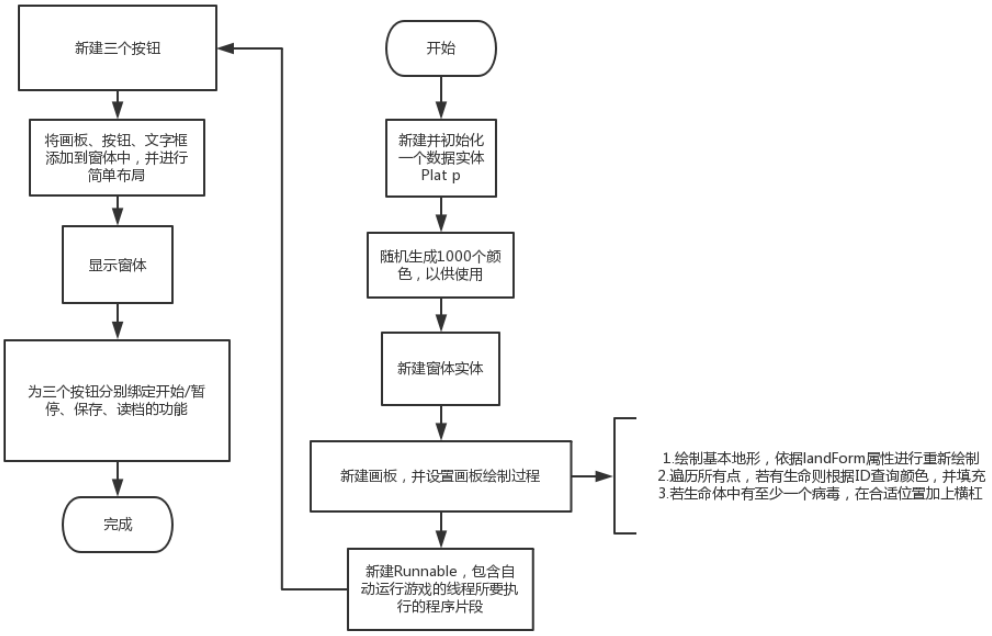
load():原理基本同上，调用 ObjectInputStream 类对象的 readObject()方法，可以将文件中的数据读取并还原为对象中的动态数据，而静态数据需要人工去设置，比如说当前下一物种的 id，需要对已有的生物进行取最大值处理获得，类似的还有病毒的 id 需要手动去设置。

main():静态方法，在这个主函数中只需无参数地实例化一个 Controller 即可。

Controller 类的高级方法：

Controller():构造函数。

具体思路：



实现代码：

```
1.  public class Controller implements Serializable{
2.  private int landForm;
3.  private Plat p;
4.  private JFrame frame;
5.  private JTextField textfield;
6.  private Runnable runnable;
7.  private Thread thread;
8.  private JPanel panel;
9.  private boolean threadIsOpen = false;
10. public Controller(){
11.     p = new Plat();
12.     landForm = 0;
13.     initialize();
14.     Plat.lifeColor = new Color[1000];
15.     int i;
16.     for(i = 0; i < 1000; i++){
17.         int r, g, b;
18.         r = (int)Math.floor(Math.random() * 256.0);
19.         g = (int)Math.floor(Math.random() * 256.0);
20.         b = (int)Math.floor(Math.random() * 256.0);
21.         Plat.lifeColor[i] = new Color(r, g, b);
22.     }
23.     frame = new JFrame();
24.     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25.     panel = new JPanel(){
26.         public void paintComponent(Graphics g){
27.             Graphics2D g2d = (Graphics2D)g;
28.             int i;
29.             int j;
30.             switch(landForm){
31.                 case 1:
32.                     g2d.setPaint(new Color(255, 140, 0));
33.                     g2d.fillRect(0, 0, 800, 800);
34.                     break;
35.                 case 2:
36.                     g2d.setPaint(new Color(152, 251, 152));
37.                     g2d.fillRect(0, 0, 800, 800);
38.                     break;
39.                 case 3:
40.                     g2d.setPaint(new Color(255, 228, 181));
41.                     g2d.fillRect(0, 0, 400, 800);
42.                     g2d.setPaint(new Color(173, 255, 47));
```

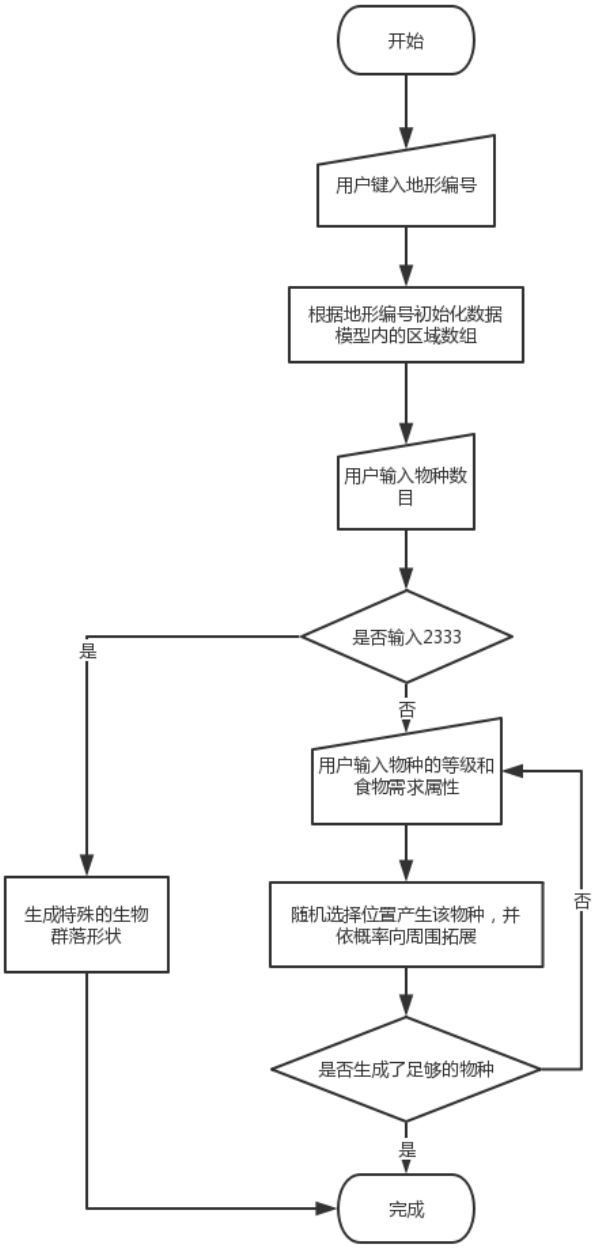
```
                    g2d.fillRect(400, 0, 400, 800);
                    break;
                case 4:
                    g2d.setPaint(new Color(0, 206, 209));
                    g2d.fillRect(0, 0, 800, 800);
                    g2d.setPaint(new Color(225,255,255));
                    g2d.fillRect(320, 320, 160, 160);
                    break;
            }
            for(i = 0; i < 100; i++){
                for(j = 0; j < 100; j++){
                    if(p.getLifeMap()[i][j] == null){
                        /*g2d.setPaint(Color.WHITE);
                        g2d.fillRect(i * 8, j * 8, 8, 8);*/
                    }
                    else{
                        g2d.setPaint(Plat.lifeColor[p.getLifeMap()[i][j]
.getId()]);
                        g2d.fillRect(i * 8, j * 8, 8, 8);
                        if(p.getLifeMap()[i][j].getVirusList().size()>0)
{
                            g2d.setPaint(Color.BLACK);
                            g2d.fillRect(i * 8, j * 8 + 3, 8, 2);
                        }
                    }
                }
            }
        }
    };
    runnable = new Runnable(){
        public void run(){
            while(true){
                moveARound();
                panel.repaint();
                showMessage();
                try{
                    Thread.sleep(100);
                }
                catch(InterruptedException e){
                    e.printStackTrace();
                }
            }
        }
    };
```

```
85.     JButton button = new JButton("Go A Round");
86.     textfield = new JTextField("");
87.     JButton buttonSave = new JButton("Save");
88.     JButton buttonLoad = new JButton("Load");
89.     frame.getContentPane().add(BorderLayout.SOUTH,button);
90.     frame.getContentPane().add(BorderLayout.CENTER,panel);
91.     frame.getContentPane().add(BorderLayout.NORTH,textfield);
92.     frame.getContentPane().add(BorderLayout.EAST,buttonSave);
93.     frame.getContentPane().add(BorderLayout.WEST,buttonLoad);
94.     frame.setSize(940,900);
95.     frame.setVisible(true);
96.     ActionListener goARoundListener = new ActionListener(){
97.         public void actionPerformed(ActionEvent event){
98.             if(!threadIsOpen){
99.                 thread = new Thread(runnable);
100.                thread.start();
101.                threadIsOpen = true;
102.            }
103.            else{
104.                thread.stop();
105.                threadIsOpen = false;
106.            }
107.        }
108.    };
109.    ActionListener saveListener = new ActionListener(){
110.        public void actionPerformed(ActionEvent event){
111.            try{
112.                save();
113.            }
114.            catch(Exception e){
115.                //do nothing
116.            }
117.        }
118.    };
119.    ActionListener loadListener = new ActionListener(){
120.        public void actionPerformed(ActionEvent event){
121.            try{
122.                load();
123.            }
124.            catch(Exception e){
125.                //do nothing
126.            }
127.        }
128.    };
```

```
129.     button.addActionListener(goARoundListener);
130.     buttonSave.addActionListener(saveListener);
131.     buttonLoad.addActionListener(loadListener);
132. }
```

initialize():由用户初始化部分数据。

具体思路：



代码实现：

```
public void initialize(){
    Scanner input = new Scanner(System.in);
    System.out.println("请选择地理条件(仅提供若干种预设):");
    System.out.println("1.平坦型(贫瘠) 2.平坦型(富饶) 3.沙漠绿洲型 4.盆地型
");
    landForm = input.nextInt();
    int i,j;
    switch(landForm){
        case 1:
            for(i = 0; i < 10; i++)
                for(j = 0; j < 10; j++)
                    p.getSectionMap()[i][j] = new Section(Section.defaultPro
ductivity, Section.defaultFittestNumOfLife);
            break;
        case 2:
            for(i = 0; i < 10; i++)
                for(j = 0; j < 10; j++)
                    p.getSectionMap()[i][j] = new Section(Section.defaultPro
ductivity * 4.0, Section.defaultFittestNumOfLife);
            break;
        case 3:
            for(i = 0 ; i < 5; i++)
                for(j = 0; j < 10; j++)
                    p.getSectionMap()[i][j] = new Section(Section.defaultPro
ductivity * 0.25, Section.defaultFittestNumOfLife / 4);
            for(; i < 10; i++)
                for(j = 0; j < 10; j++)
                    p.getSectionMap()[i][j] = new Section(Section.defaultPro
ductivity * 4.0, Section.defaultFittestNumOfLife);
            break;
        case 4:
            for(i = 0; i < 10; i++){
                for(j = 0; j < 10; j++){
                    if((i == 4 || i == 5) && (j == 4 || j == 5)){
                        p.getSectionMap()[i][j] = new Section(Section.defaul
tProductivity * 4.0, Section.defaultFittestNumOfLife);
                    }
                    else{
                        p.getSectionMap()[i][j] = new Section(Section.defaul
tProductivity * 0.5, Section.defaultFittestNumOfLife / 2);
                    }
                }
```

```
            }
            break;
        }
        System.out.println("请输入初始物种数目(输入 2333 可以设置为七夕专属物种布局):");
        int lifeNum = input.nextInt();
        if(lifeNum == 2333){
            Life a = new Life();
            Life b = new Life();
            Life[][] m = p.getLifeMap();
            m[15][15] = new Life(a);
            m[14][14] = new Life(a);
            m[13][13] = new Life(a);
            m[12][12] = new Life(a);
            m[11][12] = new Life(a);
            m[10][12] = new Life(a);
            m[9][13] = new Life(a);
            m[9][14] = new Life(a);
            m[10][15] = new Life(a);
            m[9][16] = new Life(a);
            m[9][17] = new Life(a);
            m[10][18] = new Life(a);
            m[11][18] = new Life(a);
            m[12][18] = new Life(a);
            m[13][17] = new Life(a);
            m[14][16] = new Life(a);
            m[8][80] = new Life(b);
            m[9][80] = new Life(b);
            m[10][80] = new Life(b);
            m[11][80] = new Life(b);
            m[12][80] = new Life(b);
            m[13][80] = new Life(b);
            m[14][80] = new Life(b);
            m[15][80] = new Life(b);
            m[16][80] = new Life(b);
            m[17][80] = new Life(b);
            m[8][82] = new Life(b);
            m[9][82] = new Life(b);
            m[10][82] = new Life(b);
            m[11][82] = new Life(b);
            m[12][82] = new Life(b);
            m[13][82] = new Life(b);
            m[14][82] = new Life(b);
            m[15][82] = new Life(b);
```

```
79.         m[16][82] = new Life(b);
80.         m[17][82] = new Life(b);
81.         m[11][79] = new Life(b);
82.         m[11][81] = new Life(b);
83.         m[11][83] = new Life(b);
84.         m[15][79] = new Life(b);
85.         m[15][81] = new Life(b);
86.         m[15][83] = new Life(b);
87.     }
88.     else{
89.         int k;
90.         ArrayList<Life> l = new ArrayList<Life>();
91.         for(k = 0; k < lifeNum; k++){
92.             System.out.println("请输入两个整数，代表第" + k + "种生命的等级和物
    资需求(参考值为1 1):");
93.             int levelToSet, foodDemandToSet;
94.             levelToSet = input.nextInt();
95.             foodDemandToSet = input.nextInt();
96.             l.add(new Life((double)levelToSet,(double)foodDemandToSet));
97.         }
98.         int lcounter = 0;
99.         int m,n;
100.         for(lcounter = 0; lcounter < l.size(); lcounter++){
101.             Section s = p.selectSectionRandomly();
102.             int x,y;
103.             x = y = 0;
104.             for(i = 0; i < 10; i++){
105.                 for(j = 0; j < 10; j++){
106.                     if(p.getSectionMap()[i][j] == s){
107.                         x = i;
108.                         y = j;
109.                     }
110.                 }
111.             }
112.             /* 获取了一个随机区域 */
113.             int[] pair;
114.             pair = p.selectSpaceRandomly(x, y);
115.             if(pair[0] >= 0){
116.                 Life newOne = l.get(lcounter);
117.                 p.getLifeMap()[pair[0]][pair[1]] = newOne;
118.                 x = pair[0];
119.                 y = pair[1];
120.                 for(i = x - 1; i <= x + 1; i++){
121.                     for(j = y - 1; j <= y + 1; j++){
```

```
122.                         if(i < 0 || j < 0 || i > 99 || j > 99 || (i == x &&
    j == y) || p.getLifeMap()[i][j] != null){
123.                             continue;
124.                         }
125.                         else{
126.                             if(Math.random() < 0.8){
127.                                 p.getLifeMap()[i][j] = new Life(newOne);
128.                             }
129.                         }
130.                     }
131.                 }
132.             }
133.         }
134.     }
135. }
```

该阶段内组员分工：

周尊康(15020031123)：参与编码实现，解决部分代码技术上出现的问题。并基于测试情况提出修改意见。

李宜璟(15020031034)：学习了完整版本游戏中的代码实现，调整并规范代码格式，绘制相关的流程图。完成实验报告。

陶赟(15020031070)：负责编码实现若干个复杂类，调试 BUG，提出了一些界面上的和功能上的新想法并加以实现。

软件工程课程设计实验报告四

组员：陶贇 李宜璟 周尊康　　任课教师：贾东宁

程序运行：

由于程序可以设置多种不同的初始条件，还可以较为直观的改变程序中的部分与游戏规则相关的常量。因此程序需要一定的尝试和探索，并且需要对图象输出进行适当的理解才能得到预期的观测结果。因此进行了若干次尝试，并且涉及到对以下参数的调整：

新物种诞生概率

病毒爆发概率

病毒传播概率

病毒自愈、死亡率

地图资源分布模式的选择

初始生物的种类数、层次、强度的选择

在进行了一定次数的尝试后，我们简单的模拟出了自然界中的一些规律：

1、　环境资源状况对于生物种群密度的限制

如下面图 1、图 2 所示，在资源相对匮乏的地区投放生物进行测试，得到的结果基本如图 1 结果所示，种群呈现出整体散落，局部密集的分布模式，而在环境承载力相对较大的地区投放生物进行测试，得到的结果如图 2 所示，生物种群在地图上分布得均匀且密集。因此可以感受到环境资源于生物种群密度呈正相关。

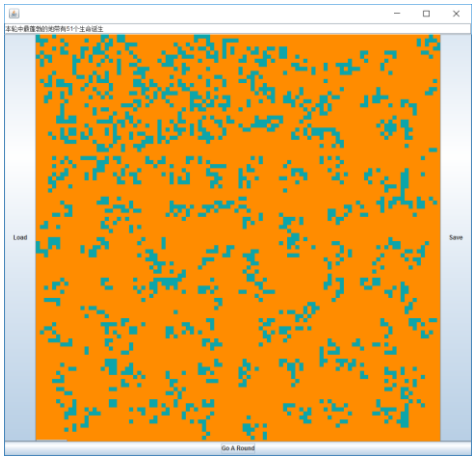


图 1

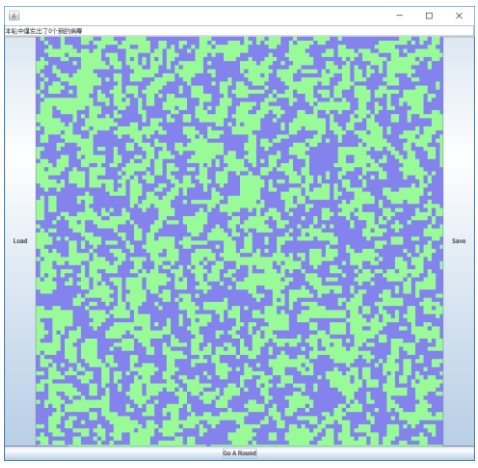


图 2

为了对上面的结论进行补充，进行了另一次尝试，将生物投放到两种资源分布情况兼顾的地图上，观察到图 3 所示的结果。

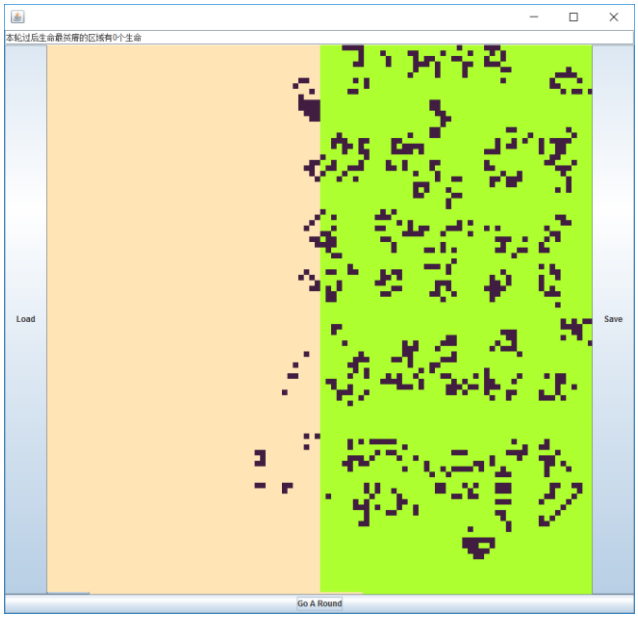


图 3

2、 物种共存的尝试

我们此处理解的共存是指 2 个或多个种群能在同一片区域长期共存，不发生大规模增加和消亡过程。鉴于两个属性设定完全相近的生物会均匀的混合共存，此处先进行的尝试的是具有简单摄食关系的两个生物。得到两次较为典型的结果如下：

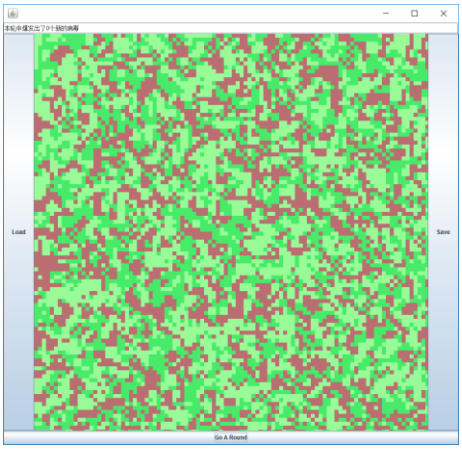


图 4

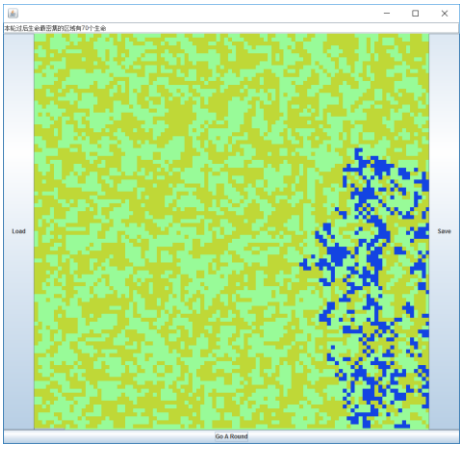


图 5

在图 4 中，两种生物基本较好的融为了一体，可以看到作为高级生物的紫色块相对稀疏一点，这也基本符合食物链底层生物多的规律。

在图 5 中，蓝色块表示的高级生物被限定在了一定区域内生活，摄食自然资源和黄绿色的底层生物。

在测试中我们发现只有两种不同等级生物种群融合接触得相对紧密了才会稳定存在，若分的较远，而在资源分配上产生了冲突，很可能发生低等生物抢走了远处高等生物的食物，而远处高等生物却无法捕获低等生物结果被大量饿死的尴尬情况，这与生物具一定的机动性这一客观事实相违背。因此会导致测试不理想。故对代码进行了调整，允许生物依据自己的等级决定摄食范围，更接近自然情况。在这种条件下，得到的测试图样如下。

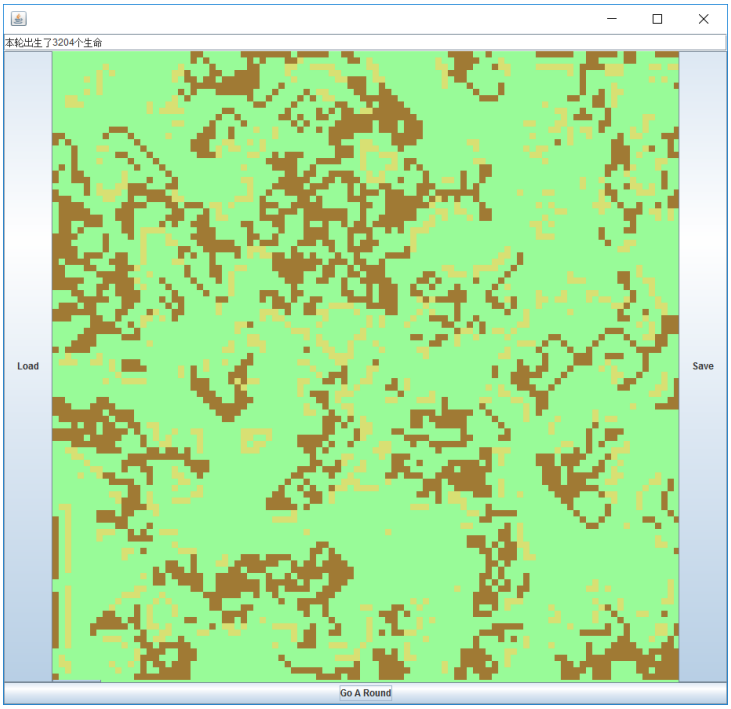


图 6

在图 6 中，我们可以观察到，作为高级生物的淡色生物虽然由于自然资源的原因不能过多的存在，但是它们可以较为方便的掠食低等生物保持生存，从图中可以看到，淡色的生物基本上围绕着深色的低等生物分布，并且在运行时，可以观察到明显的追赶的过程，较为符合实际情况。

对于更多种生物，测试若干次后也得到了几次相对理想的共存的结果：

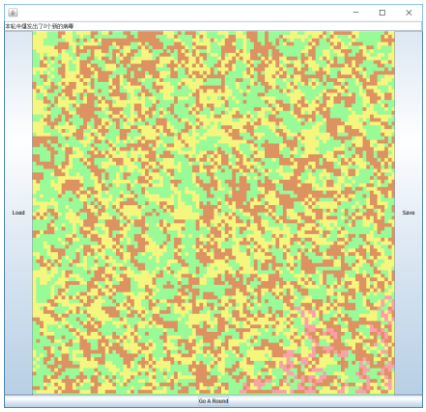


图 7

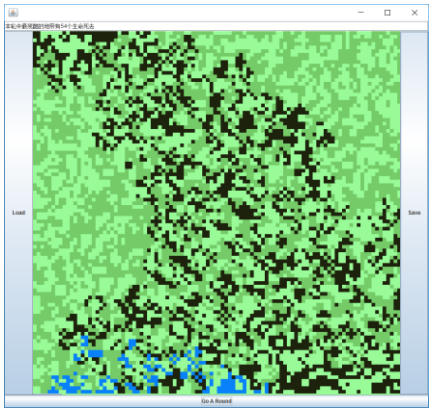


图 8

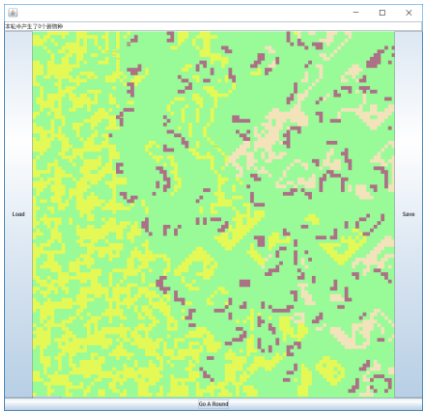


图 9

在尝试的过程中我们发现，往往生存下来的都是一个较为低级的生命和一个较为高级的生命，而作为牺牲品的中间的物种有一定的摄食能力和掠食能力，却因为既要面临摄取自然资源的竞争，又要面临被摄食的风险，而较为艰难的存活着，这不禁让人想到地球上的很多动物，也正处于这样的境地。

3、 疾病的模拟

在游戏的初始设定中，疾病是以低概率出现的。并且可以进行一定规模的快速传播，随后可以被治愈，然而在测试的时候却出现了极端的情况。

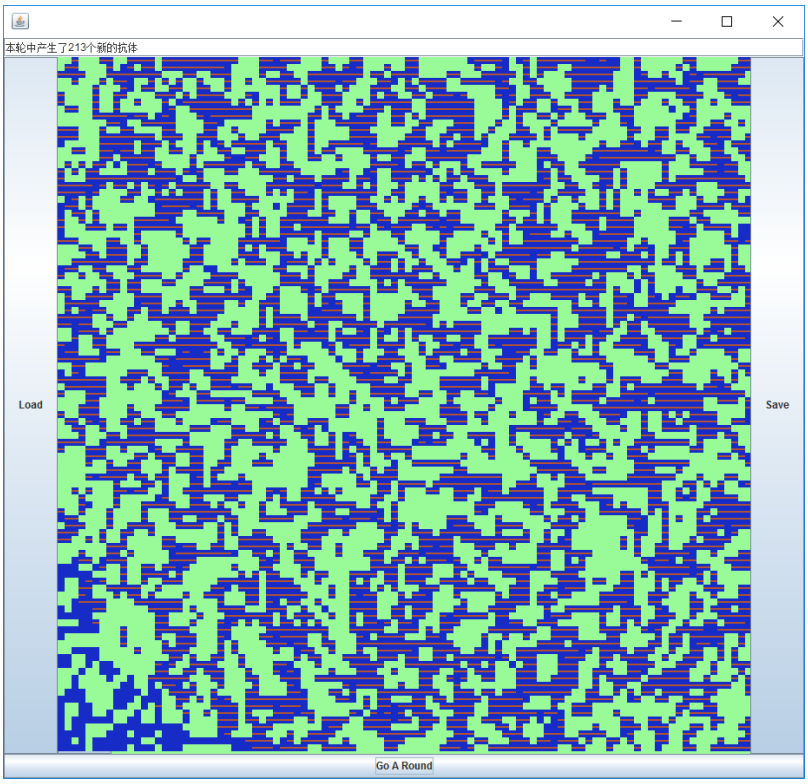


图 10

比如图 10 所示的情况，病毒在出现后立刻占领了整个地图，且无法被清除。本以为是病毒的自愈概率和抗体的传播概率过低，进行了调整，却又出现了如图 11 所示的情况，在图 11 中，病毒仅能片刻的存在，很快就被直接大量的自愈所消灭。

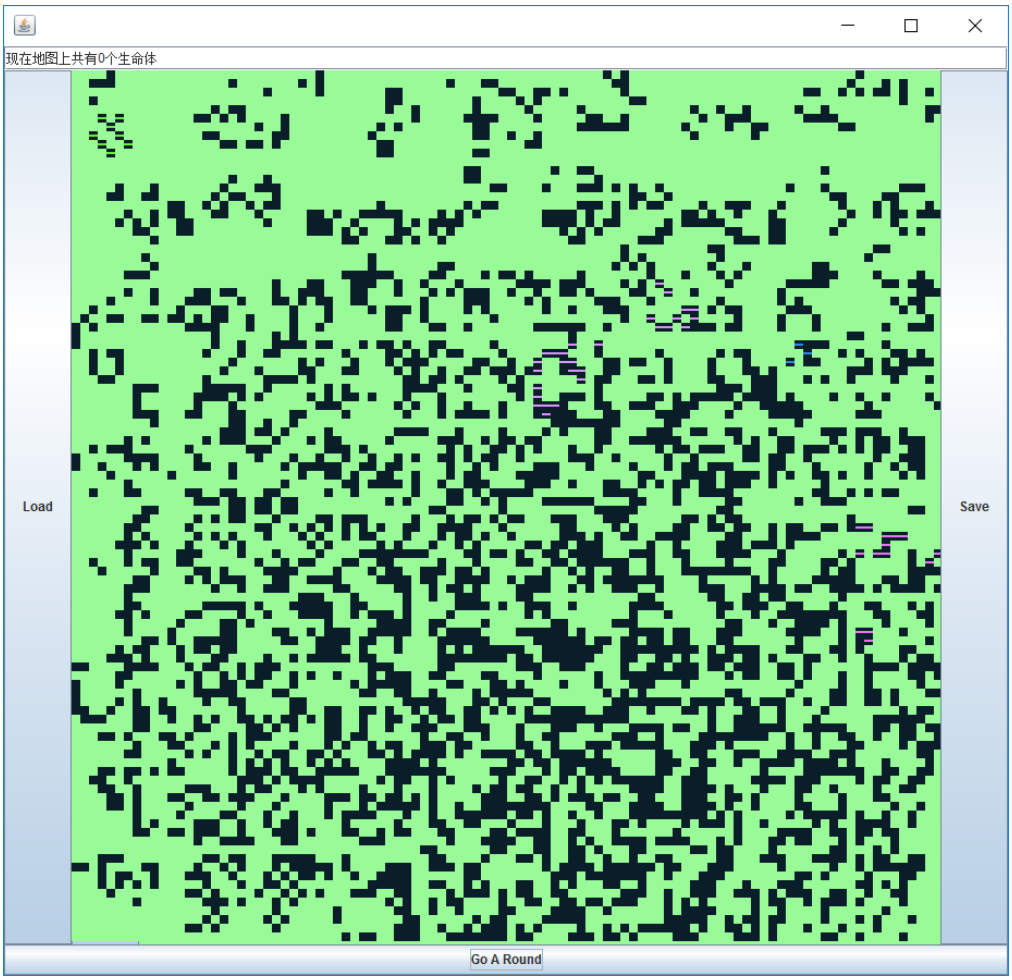


图 11

在仔细回顾代码后，发现病毒原来的概率设置并没有什么问题，真正应该关注的是为什么第一次测试时大量病毒却没有被抗体所抑制，再检查发现抗体传播的途径存在实现时的逻辑漏洞，导致了抗体只能在很邻近的范围内传播且有可能无法有效的消除病毒。修复并将数据倒回原来的状态后，可以观察到健康的疾病发展情况，例如图 12、图 13、图 14 所示。

图 12 展现的是病毒爆发、图 13 表展现的是病毒在蔓延的同时开始被抑制，图 14 展现的是病毒最终被消灭。

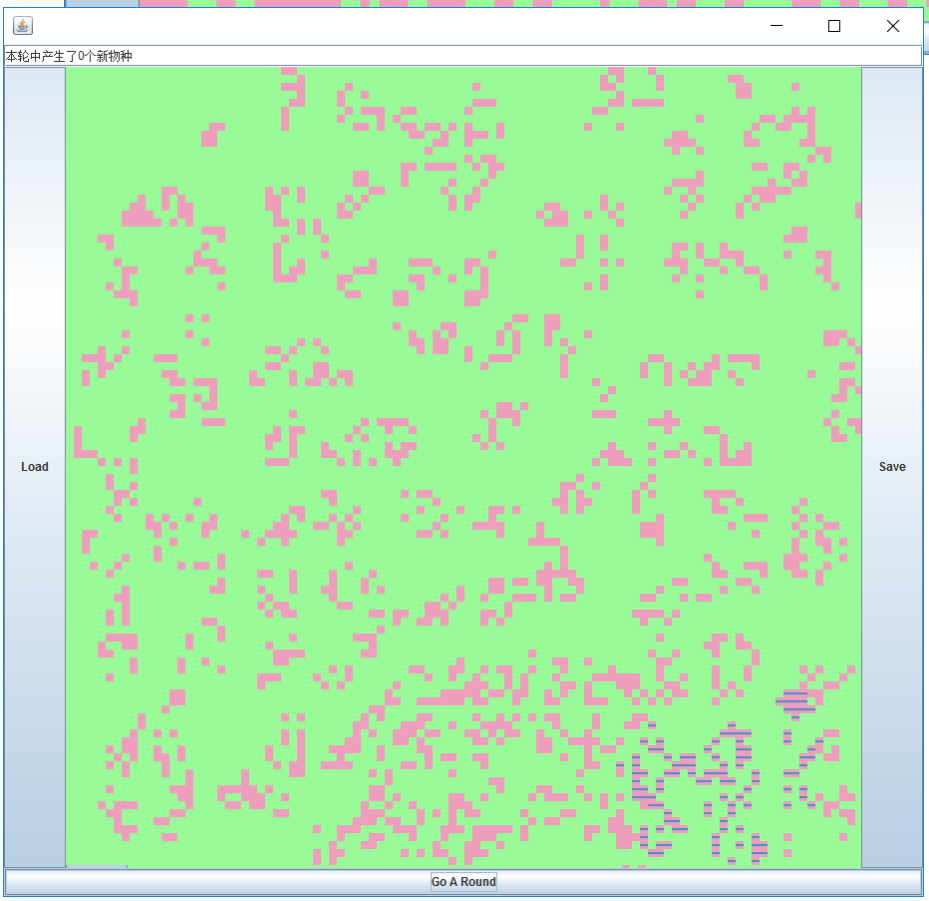

本轮中产生了0个新物种
Load
Save
Go A Round

图 12

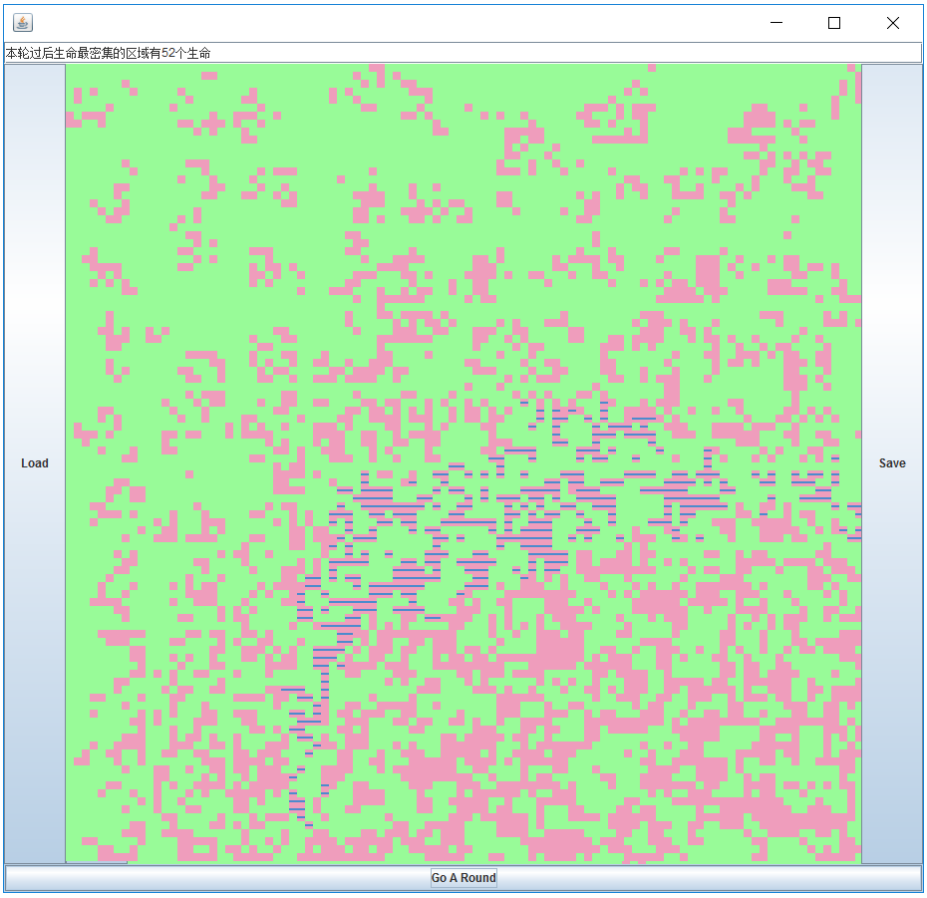

本轮过后生命最密集的区域有52个生命
Load
Save
Go A Round

图 13

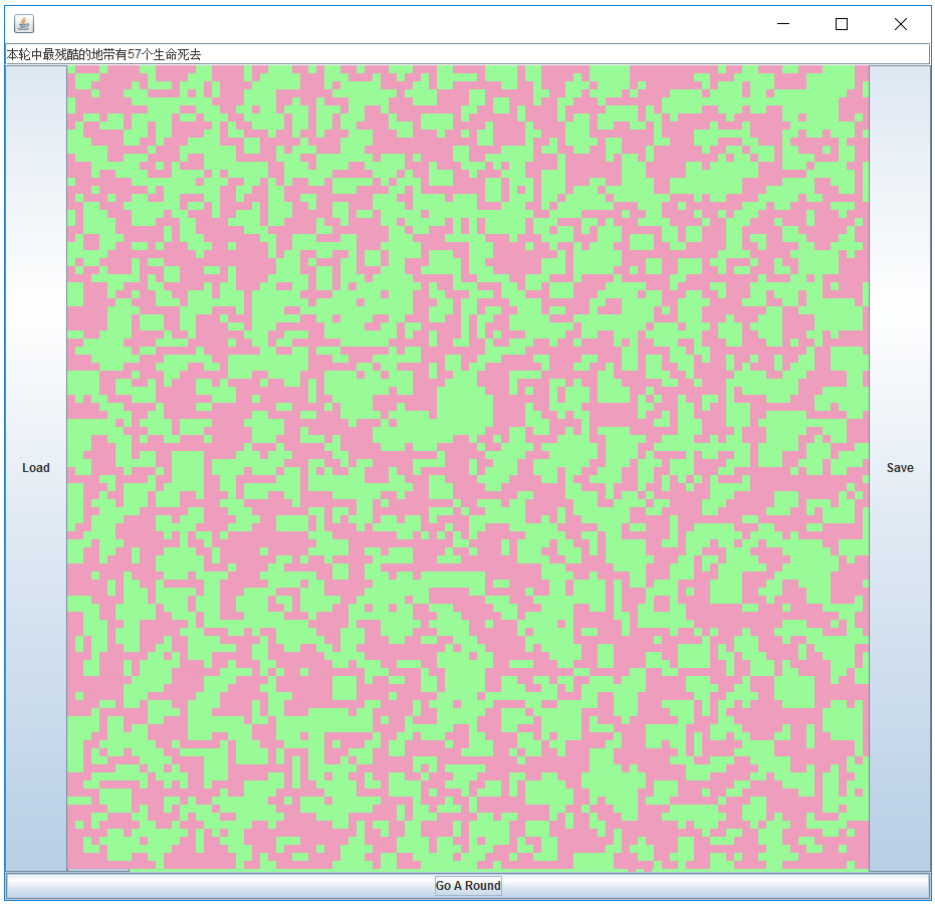


图 14

小结

实验的收获：

通过本次课程设计，我们完成了从设计到实现的完整过程，明确了在一个产品的各个阶段应该着重完成哪些工作，在这次课程设计中，主要分为了分析、设计、实现、使用这四个过程。通过一次完整流程的开发也使得我们意识到各个阶段的重要性，用以明确目标，明确架构等。

同时为了提升动手能力，选择尝试了一门新的面向对象语言来实现这个项目，了解了一些语言特性如对象引用，一些封装好的数据类型如 List 等以更高效的完成代码工作。

通过 JAVA 的图形库用纯代码相对直观快速的实现了图形界面交互，虽然界面相比之下显得比较简单，但是能让人相对透彻的理解窗体程序的建立、

各种控件的排布，事件的监听，和二维图形绘制等过程。同时也了解到其他语言的图形界面的实现方式也是大同小异的。

不足：

代码实现过程是相对比较仓促的，基本是在不到一周的时间完成了 90% 的工作，在这个过程中存在很多违背了原设计的行为，比如为了图省事，忘了接口的存在而直接修改权限访问数据，或者是遇到不清楚该由哪个对象来承担相应功能的时候，随意决定基于哪个对象如何实现。导致的直接问题是对象间的关系更加复杂，想要进行功能上的修改往往会找不准从哪儿下手。打击了后期的代码修改的积极性。

规则和常量的分离做的还是不够，按照先前的思路，用于描述规则的数据都应该被参数化或者作为常量分离出来，方便进行调整，但是考虑到这样做的代码量和逻辑连续性会受到影响，故有所偷工减料，导致在调试运行时无法非常灵活的调整规则。

交互缺少灵活性，本程序的在数据逻辑上进行了较多的设计和考量，整个游戏的规则也是相对比较丰富的，模拟了近 10 个群落演替中的阶段，然而在展现形式上只有动态的画板，画板上呈现生物，病毒而已，另外也只有一个文字框输出了少量信息，在后期修改中，再从命令行输出了一些调试用的数据。并且在初始化数据后是无法中途修改全局参数的。因此在这方面的设计还是欠些考虑的。

心得：

各个阶段的事情应该分开踏实的完成，比如规则设计阶段，就应该避免模棱两可，把想要完成的东西说明确，比方说其中的掠食模型，在设计时只是交代了有猎捕周围生物的过程，而在实现的时候需要具体的描述，显然在不合适的时间思考不合适的事情会导致一些错误，比如在实现时图方便就直接取邻近的格子模拟掠食过程，而导致模拟效果很差且找了很久才发现原因。又比如说交互设计方面，只迫切的想到要呈现出视图效果，而忽视了一般的输入输出的需要。所以应该认真对待每一个有意义的过程而不是急功近利，急于看到成果。

该项目有些同学是基于现有的模板和框架拓展完成的，而作为大部分纯

代码实现的项目，不仅是为了帮助牢固对于语言和设计逻辑的理解，也是为了完成更自由的设计实现，然而由于结构组织上缺乏考量，实际做出的效果、创意和性能可能还不如某些基于框架模板实现的，我觉得通过这个问题，我们在设计这样一个小规模应用程序的时候，还应该考虑一些类似于设计模式的问题。即如何组织对象的层次关系，如何组织通信，如何组织结构流程等。而不仅仅局限于如何尽快的实现一个创意和功能。

时间有限，而且同时有三门课设要去完成，相对来说就有些赶了。

该阶段内组员分工：

周尊康(15020031123)：修改部分在模拟时发现的程序逻辑错误。

李宜璟(15020031034)：设计并进行若干次模拟测试，记录图像并总结规律。

陶赟(15020031070)：组织小组讨论，对项目进行小结并撰写实验报告。