**贪吃鱼笔记**

一：首先需求是创建窗体对象，面板对象和鱼对象。

1. 然后在窗体类文件里面利用构造方法创建一个用来游戏窗体
2. 通过设置来制定窗体的尺寸可见性等属性

二：与此同时，在JPanel类里面进行进行自定义面板，在这里可以对背景做出设置。通过重写JPanel类的paint方法，在里面画出背景。（此处有优化）

【优化】：通过建立一个FishUtil类文件，然后在里面定义map集合的IMAGES的常量，把所有存放的图片全部都缓存到这个常量集合，供之后使用图片时方便调取。

\* 定义一个常量：public static final 常量名一般大写 key--value :文件名---image对象

\*/

public static final Map<String, BufferedImage> IMAGES = new HashMap<String, BufferedImage>();

三：有了背景以后就开始着手进行玩家鱼的创建和导入

设置玩家鱼：

先在Fish类里面设置鱼图像的基本属性和get、set方法。

然后通过创建对象创造出一只玩家鱼，用myfish引用来引导这个对象。接着就是设置玩家鱼的图片。

将玩家鱼放入面板：fjp.setMyFish(myFish);

再将面板放入窗体： this.add(fjp);

【注意】在JPanel里面进行paint方法绘画背景和玩家鱼的时候，需要注意顺序，先画背景再绘图玩家鱼，否则鱼会被背景覆盖掉而不可见。

一个玩家鱼的文件里面含有十五张连起来可以完成鱼游动动画的效果，但是如果要玩成这样的效果，就必须把十五张小图分隔开，然后在同一个位置循环播放，达到动画的效果。

设置鱼的小图的宽和高

String infosStr = FishUtil.INFOS.get(myFish.getType());// "200 180 3 0"

String[] infosArr = infosStr.split(" ");

myFish.setWidth(Integer.parseInt(infosArr[0]));

myFish.setHeight(Integer.parseInt(infosArr[1]));

这里又涉及到了FishUtil类文件里面的一个常量优化INFOS，通过这个常量数组来储存所有鱼的小图应有的宽和高。

但是，鱼小图的宽和高值不同鱼是不同的，所以我们不能把它写死。

鱼的小图的数量=大图的宽/小图的宽；

在Fish类里面定义curIndex属性 表示当前的小图的下标；

还要给鱼设置它对应的小图的宽和高，还有类型。private int type;

myFish.setCount(myFish.getImage().getWidth()/myFish.getWidth());

将鱼放入了窗体以后，它还只是个不会移动的图片，根据我们的实际要求，鱼是需要随着鼠标的移动而移动的。

所以，要设置让鱼跟着鼠标光标一起移动；这里就必须实现鼠标动作监听器，实现MouseMotionAdapter适配器监听鼠标的移动。

先找到事件源----面板

然后创造监听器

最后绑定事件源

// 设置面板获取光标

fjp.addMouseMotionListener(new MouseMotionAdapter() {

@Override

public void mouseMoved(MouseEvent e) {

// super.mouseMoved(e);

// 1.获取鼠标的坐标

Point mouseP = e.getPoint();

//System.out.println("x===" + e.getPoint().x + "\ty==" + e.getPoint().y);

//2.设置鱼的方向:

//得到鱼的坐标

Point fishP = fjp.getMyFish().getPoint();

if(mouseP.x < fishP.x) {

fjp.getMyFish().setDir(1);

}else if(mouseP.x > fishP.x){

fjp.getMyFish().setDir(-1);

}

// 3.设置鱼的坐标为鼠标的坐标

fjp.getMyFish().setPoint(mouseP);

然后我们通过内部类的方式去定义线程类，制作出鱼儿游动的动画效果，根据之前的方法，我们可以很明确的知道一个种类的鱼素材里面有多少个小图，并且把这些当成元素都储存在了集合里面。接下来要做的事情就是通过循环提取集合中元素并展示出来就能达到动画的效果。

**class FishThread extends Thread {**

**Random rd = new Random();**

**public void run() {**

**//1.控制的玩家鱼游动。循环curIndex自动1**

**while(true) {**

**// System.out.println("======"+isPause);**

**// try {**

**// sleep(10);**

**// } catch (InterruptedException e1) {**

**// // TODO Auto-generated catch block**

**// e1.printStackTrace();**

**// }**

**if(isPause) {**

**continue;**

**}**

**//玩家鱼的小图的下标+1**

**int index = myFish.getCurIndex()+1;**

**//如果绘制到最后1张小图，重新绘制第1张小图**

**if(index==myFish.getCount()) {**

**if(myFish.getStatus().equals(FishUtil.STATUS\_TURN)) {**

**myFish.setStatus(FishUtil.STATUS\_SWIM);**

**}**

**if(myFish.getStatus().equals(FishUtil.STATUS\_EAT)) {**

**myFish.setStatus(FishUtil.STATUS\_SWIM);**

**}**

**index = 0;**

**}**

**myFish.setCurIndex(index);**

这里如果不设置的话，鱼就只会播放一遍，必须做出判断然后规定每次播放到最后一张就要跳回第一张图然后继续顺序播放。

try {

Thread.sleep(100);（ Thread.sleep(毫秒整数); 让当前线程休眠多少毫秒 ）

} catch (InterruptedException e1) {

e1.printStackTrace();

}

这个时候试验出来，发现鱼的动作过于快速，形成了抽搐的感觉，这个时候可以通过在循环里面加入sleep休眠的方式来在每张鱼小图的间隔中插入一个100ms的休眠方式，这样就能稳定的把鱼的游动展现出来。

鱼的移动------main线程

鱼的游动-----自定义线程中

在玩家鱼完成了移动和游动以后，再对鱼的掉头进行处理，本来往左游的鱼突然要往右游动，这个时候的转身动作所读的小图，不再是之前游动时的鱼的小图，而是另一个图片文件中的，所以要设置玩家鱼的状态常量，以方便在后面所需要的“游动，转身，吃鱼”三种状态中进行转换。

/\*鱼的状态常量\*/

public static final String STATUS\_SWIM = "swim";

public static final String STATUS\_TURN = "turn";

public static final String STATUS\_EAT = "eat";

做转身操作以后，那么就需要通过镜像图片获取的方式来得到原本游动文件里面鱼往反方向游动的图像。获取镜像图片:输入一张图片，输出它的翻转图片

public static BufferedImage getOppoImage(BufferedImage img){

int width = img.getWidth();

int height = img.getHeight();

BufferedImage dstImage = new BufferedImage(width, height,2);

AffineTransform at = new AffineTransform( -1, 0, 0, 1, width, 0);

AffineTransformOp ato = new AffineTransformOp(at, 1);

return ato.filter(img, dstImage);

玩家鱼弄好了以后，就开始做野鱼了。现在先把野鱼创造出来，

我们要先确定每一关需要的野鱼数量要处于怎样的一个数字范围内，这里可以用random随机数来确定。

fishs.retainAll(fishs);

//1.野鱼的数量？？---随机 5-9

Random rd = new Random();

int count = rd.nextInt(5)+5;

在这之后，便是new出所有的鱼对象再进行设置；

.根据数量new出所有的鱼对象

for (int i = 0; i <count; i++) {

//这里是通过循环来做

Fish fish = new Fish();

int type = rd.nextInt(FishUtil.FISH\_TYPE.length-1)+1;//type=1~9

BufferedImage image = FishUtil.IMAGES.get(FishUtil.STATUS\_SWIM+"\_"+type+".png");

fish.setImage(image);

fish.setType(type);

String[] infos = FishUtil.INFOS.get(type).split(" ");

fish.setWidth(Integer.parseInt(infos[0]));

fish.setHeight(Integer.parseInt(infos[1]));

fish.setLevel(Integer.parseInt(infos[2]));

fish.setScore(Integer.parseInt(infos[3]));

fish.setStatus(FishUtil.STATUS\_SWIM);

fish.setCount(image.getWidth()/fish.getWidth());

fish.setCurIndex(0);

//这里setdead是描述的鱼的生存状态，如果是死的就会将它从游戏窗体内抹除。

fish.setDead(false);

【优化】 每条鱼小图的数量是不同

File定义count属性

初始化鱼的时候，设置鱼的小图的数量 =大图的宽度/小图的宽度

所有要使用小图数量的地方 getCount();

野鱼全部创造出来以后还需要给它放入到窗体中去，不过很显然不能把这么多鱼都放在一个坐标上进行堆叠，所以将它们放入窗体的时候，我们还需要随机去产生他们的初始位置。

int x = rd.nextInt(Constant.WIDTH-fish.getWidth()\*fish.getLevel()/FishUtil.FISH\_BASE\_LINE);

//0~(799-鱼的宽度)

int y = rd.nextInt(Constant.HEIGHT-fish.getHeight()\*fish.getLevel()/FishUtil.FISH\_BASE\_LINE);;

Point point = new Point(x,y);

当鱼全部放入以后，现在要让所有的野鱼都能动起来，这个时候就需要给他们一个目标，只有知道了目标所在的位置，才能让它们往目标坐标进行移动。从开始位置移动目标位置（速度）

Point start = fish.getPoint();

Point end = fish.getTargetPoint();

int dx = (end.x-start.x);

int dy = (end.y-start.y);

int distance= (int)Math.sqrt(Math.pow(dx, 2)+Math.pow(dy, 2));

当鱼快接近目标地点的时候必须快速的重新随机出新的目标坐标，否则当鱼到达目标坐标的时候就会停下来不再移动，而且设置这个感应距离不宜过近，然后也需要设置鱼移动的速度，这样才能保证他不会因为移动过快而直接到了目标坐标点上。

//重写产生新的目标位置

/\*设置鱼的目标位置\*/

int tx = rd.nextInt(Constant.WIDTH-fish.getWidth()\*fish.getLevel()/FishUtil.FISH\_BASE\_LINE);//0~(799-鱼的宽度)

int ty = rd.nextInt(Constant.HEIGHT-fish.getHeight()\*fish.getLevel()/FishUtil.FISH\_BASE\_LINE);;

Point targetPoint = new Point(tx, ty);

fish.setTargetPoint(targetPoint );

fish.setPoint(point );

fish.setDir(calcDirection(point,targetPoint));

/\*设置鱼的目标位置\*/

int tx = rd.nextInt(Constant.WIDTH-fish.getWidth()\*fish.getLevel()/FishUtil.FISH\_BASE\_LINE);//0~(799-鱼的宽度)

int ty = rd.nextInt(Constant.HEIGHT-fish.getHeight()\*fish.getLevel()/FishUtil.FISH\_BASE\_LINE);;

Point newTargetPoint = new Point(tx, ty);

fish.setTargetPoint(newTargetPoint);

fish.setDir(FishUtil.calcDirection(start, newTargetPoint));

}

fish.setPoint(newPoint);

结束了所有鱼类的设置以后，接下来就是游戏的主要内容了，必须要完成鱼吃鱼的功能，我们需要让两个图片碰到一起的时候执行吃这个方法，我们在野鱼的全身和玩家鱼的嘴巴附近设置上矩形。从而判断是否能够吃到野鱼 -------矩形碰撞

Rectangle myFishRect = fjp.getMyFish().getRect();

Fish myFish = fjp.getMyFish();

for(Fish fish :fjp.getFishs()) {

if(!fish.isDead()) {

Rectangle fishRect = fish.getRect();

if(fishRect.intersects(myFishRect)) {

System.out.println("被吃......");

//1.野鱼消失（1.从集合中删掉这条鱼 2.设置鱼的状态isDead）

fish.setDead(true);

//2.玩家鱼的分数增加

int newScore = myFish.getScore()+fish.getScore();

myFish.setScore(newScore);

//3.玩家鱼状态变为吃 eat

myFish.setStatus(FishUtil.STATUS\_EAT);

break;//停止循环

当鱼被吃了以后，它显然不能继续存在在游戏页面，这个时候就用//重新调用FishJPanel的paint();

repaint(); 重绘图的方式把它消失掉。

鱼全部吃掉以后，我们就必须产生新的鱼来开始下一局游戏。

判断是否鱼都死了

boolean flag = true;//假设全死了

for(Fish fish :fishs) {

if(!fish.isDead()) {//找到没有死的

flag = false;

}

}

if(flag) {

count++;

}

if(count==20) {

//产生新的鱼

FishUtil.createFishs(fishs);

count = 0;

}

//3.休眠 100毫秒

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

//4.重新绘制

repaint();